"objects that last"
overview

• Persistent dynamic memory management library
  – memory manager independent
  – support for garbage collector

• Implemented over ARIES storage management (l1add)
  – but is generally storage independent

• Atomicity through transaction semantics
  – logical consistency in the presence of failure

• Automatic reconstruction and crash recovery
  – physical / binary compatibility of memory blocks (intra-object)
  – topological reconstruction of references (inter-object)

• Emphasis on flexibility and usability
  – support for legacy code / library processing
  – mixed persistent / transient object management
  – logical consistency factored out
Don’t mess with…

- files and I/O
- representation conversions
- reconstruction
- atomicity and crash recovery
- storage management and garbage collection
- dumping recursive data structures

```c
Node *list = NULL;
void add_line (char *line) {
    int len = strlen (line);
    pobj_start ();
    Node *node = (Node *)
        pobj_malloc (sizeof (Node));
    char *str = (char *)
        pobj_malloc (sizeof (char) * (len + 1));
    pobj_ref_typify (node, node_ref_fields);
    strcpy (str, line);
    pobj_update (str);
    POBJ_SET_REF (node, str, str);
    POBJ_SET_REF (node, next, list);
    pobj_static_set_ref (&list, node);
    pobj_end ();
}
int main (int argc, char **argv) {
    pobj_init ();
    while (get_line (line, sizeof (line)))
        add_line (line);
    print_list ();
}
```
persistent memory objects

- Memory backing store
- Static variables
- Static references repository
- Persistent objects repository

- Persistency header
  - Link to repository
  - Object size

- Object data
  - Native layout

- Type info (optional)
  - Reference fields
  - Transient fields

- List
  - Str
  - Next
The obvious case (below) but also more interesting ones (right)…
Persistent objects implemented in *Titanium*, an SPMD Java dialect

- Persistency statically known through separate type hierarchy
- Persistent operations compiled to use `pobj` functions

```java
Persistent p = new PersistentString();
```

- Safe operation aggregation through *transaction* blocks
- Support for persistent arrays through qualifiers

```java
int[] persistent x = new int[4] persistent;
```

`pobj` integrated with Titanium’s Boehm-Weiser garbage collector
Use...

• persistent objects that extend Persistent
• static variables to point to roots of data structures
• transaction blocks to maintain consistency

```java
static class Node extends Persistent {
    ...}
static Node head, tail;
static void addLine (String s) {
    transaction {
        Node node = new Node(s);
        if (head == null)
            head = tail = node;
        else {
            tail.next = node;
            // Crash in the middle of a transaction.
            if (node.num == 4) throw new Error();
            tail = node;
        }
    }
}

// List only built on first run.
public static void main (String[] args) {
    String line;
    if (head == null) // Check if first run.
        while ((line = getLine ()) != null)
            addLine (line);
    printList ();
}
```

### Program execution results

<table>
<thead>
<tr>
<th>input</th>
<th>run #1 output</th>
<th>run #2+ output</th>
</tr>
</thead>
<tbody>
<tr>
<td>line1</td>
<td>(crash)</td>
<td>line1</td>
</tr>
<tr>
<td>line2</td>
<td></td>
<td>line2</td>
</tr>
<tr>
<td>line3</td>
<td></td>
<td>line3</td>
</tr>
<tr>
<td>line4</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
evaluation

- Different approach than RDS/RVM
  - topological storage and reconstruction
    - nice abstraction for objects
    - recursive operations: update, mark-and-sweep
  - independent of memory and store managers
  - persistent / transient allocated on a single region
    - possibly improves spatial cache locality
  - but probably not as efficient (benchmarks are future work!)
- Some numbers…
future work

Short term
• dynamic persistency
• type descriptors
• generalized statics
• transient fields
• fast checksum comparison

Long term
• flexible pointers
• on-the-fly rollback
• delayed recovery (static initializers)

Sci-Fi
• object synchronization
• automatic lock management (deadlock resolution)
• persistency semantics for Titanium/Java
  – persistify predefined types through qualifiers
  – runtime detection of persistent operations
  – transaction block optimization (reduce number of updates)