A Sharing Problem we all Share:
Data-Centric OSes, Persistent Memory, and OS Evolution

Daniel Bittman            Peter Alvaro            Ethan Miller
UC Santa Cruz
A Sharing Problem we all Share:
Data-Centric OSes, Persistent Memory, and OS Evolution

Daniel Bittman       Peter Alvaro       Ethan Miller
UC Santa Cruz
Hardware Trends

(artistic rendering; actual implementation may vary)

~300 ns
Growing, becoming persistent

~1 us
Outdated interface

~1-10 ms
Cannot compute on directly

Persistent data should be operated on directly and like memory
Hardware Trends

Multiplicity of Computing Devices and Heterogeneous Memory
## Hardware’s Needs vs. Software’s Needs

<table>
<thead>
<tr>
<th>Consideration</th>
<th>Hardware</th>
<th>Software</th>
</tr>
</thead>
<tbody>
<tr>
<td>Latency</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>In-memory Data Structures</td>
<td>X</td>
<td>✓</td>
</tr>
<tr>
<td>Data Lifetime and Persistent Data References</td>
<td>X</td>
<td>✓</td>
</tr>
<tr>
<td>Memory Heterogeneity and Data Movement</td>
<td>✓</td>
<td>X</td>
</tr>
</tbody>
</table>
Heterogeneity and Autonomy

Access data A

NIC

Access data A

FPGA

DRAM

BNVM
Data Movement

Access data
A

Move data
A

NIC

FPGA

DRAM

BNVM
In short...

**Software** cares about long-lived data relationships, even across program runs.

**Hardware** must provide consistent data access, even if it moves in memory.

Virtual memory is the **wrong** abstraction.

Virtual memory is fine.

Software is easier to change than hardware
Twizzler: A new OS

The kernel is “out of the way”

Presents a unified interface for data sharing, security, and persistent pointers
The Death of the Process
A global object space

Persistent data should be operated on *directly* and *like memory*

An object is a unit of semantically similar information

E.g. a b-tree, or part of one.
A global object space

Persistent data should be operated on *directly* and *like memory*

An object is a unit of semantically similar information

E.g. a b-tree, or part of one.

Pointers may be *cross-object*: referring to data within a different object
Persistently pointers in Twizzler

Virtual addresses are the *wrong* abstraction

Process 1

```
A  B
```

Process 2

```
C  B  A
```

```
object-id  offset
```

A

B
Twizzler’s pointers

### Foreign Object Table

<table>
<thead>
<tr>
<th>1</th>
<th>object ID or Name</th>
<th>Name Resolver</th>
<th>flags</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>object ID or Name</td>
<td>Name Resolver</td>
<td>flags</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Object Layout

<table>
<thead>
<tr>
<th>FOT</th>
<th>Data</th>
</tr>
</thead>
</table>
Example pointer resolution

FOT entry of >0 means “cross-object”—points to a different object.
Pointer implementation

\[ \text{int } *\text{tmp0} = \text{lea}(O, \text{ptr0}); \]
\[ \text{int } *\text{tmp1} = \text{lea}(O, \text{ptr1}); \]
\[ x = *\text{tmp1}; y = *\text{tmp0}; \]
Two-level Mapping

Virtual Space

Object A: r-x
Object B: rw-

Object Space

Object A: rwx
Object C: r--

Physical Memory

DRAM
NVRAM

Security Contexts!

n+m page tables! (instead of n*m)
Hey look it’s a Venn Diagram

- Persistent Pointers
- Data sharing
- Twizzler
- Security model

PMDK
Case Study: KVS

250 lines of simple C code is all you need

(store is the reverse of lea: convert a virtual address into a persistent pointer)
Add access control to the existing design

bucket = get_bucket(key)
item.ptr = lea(Index, bucket.ptr)
item.len = bucket.len

Index points to **different data objects** with **different access control**.

Can hand out pointers to these objects, which can **only be dereferenced with proper permissions**.
Benchmark: SQLite, throughput

![Chart showing SQLite performance across different workloads](chart.png)
Benchmark: SQLite, latency

![Graph showing SQLite performance metrics]
We need to consider persistent memory programming in the context of sharing and security.
A flexible persistent pointer design enables sharing, upgrades, and late-binding.
Takeaways - 3

We are building Twizzler to explore new programming models for NVM

We must evolve our storage models for new technology
Thank You!
questions / discussion

Daniel Bittman
dbiltman@ucsc.edu
@danielbittman

Peter Alvaro
palvaro@ucsc.edu

Ethan L. Miller
elm@ucsc.edu
Late-binding of access control

Super user
User manager
User Database rw-

Normal user
User manager
User Database r--