How does SoftPM work

We rely on a combination of static code C analysis and runtime instrumentation.

At compile time we add hints to:
- Record the location and size of memory allocations
- Register the location of all pointers at initialization
  - Explicit initialization: \( l\)-value of assignment operations
  - Implicit initialization: memory copying/move operations

<table>
<thead>
<tr>
<th>Type Definition</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>struct list {}</td>
<td></td>
</tr>
<tr>
<td>int val;</td>
<td></td>
</tr>
<tr>
<td>struct list *next;</td>
<td></td>
</tr>
<tr>
<td>};</td>
<td></td>
</tr>
</tbody>
</table>

Evaluation

Performance

The workload adds, queries, and deletes 512 byte elements with 16 byte keys. The dashed line represents a memory only solution.

- Only minor code changes required
- With SoftPM we achieve up to 10X better performance

Scalability

- Guaranteed sequential writes even with multiple threads
- I/O time scales linearly

Code Complexity

- Using SoftPM reduces the LOC required for persistence
- No need for application level I/O barriers

Conclusions and Future Directions

SoftPM provides:
- A memory abstraction to work with persistent memory
- High-Performance persistence I/O with existing technologies
- Conventional volatile interfaces for easy use and adoption

Future work:
- Support for versioning, record-replay, and execution branching
- Support for sharing containers across multiple processes
- Support for other programming languages (C++, Fortran)