

Software Persistent Memory

Raju Rangaswami, Ming Zhao, Jason Liu
raju@cs.fiu.edu

School of Computing and Information Sciences
College of Engineering and Computing
Florida International University



HEC FSIO 2009 Conference

Motivation

Recoverability

- ▶ HPC applications execute for days, weeks, or even months
- ▶ If there is a failure, the application has to be restarted

Motivation

Recoverability

- ▶ HPC applications execute for days, weeks, or even months
- ▶ If there is a failure, the application has to be restarted

Record-Replay

- ▶ HPC applications generate enormous amounts of data
- ▶ Scientists often wish to not only analyze the result but also intermediate steps (e.g data visualization)

Motivation

Recoverability

- ▶ HPC applications execute for days, weeks, or even months
- ▶ If there is a failure, the application has to be restarted

Record-Replay

- ▶ HPC applications generate enormous amounts of data
- ▶ Scientists often wish to not only analyze the result but also intermediate steps (e.g data visualization)

Execution Branching

Scientists may further analyze a simulation execution by

- ▶ changing a parameter value at an execution point
- ▶ exploring alternate models

Software Persistent Memory (*SoftPM*)

Applications allocate, and interact with, persistent memory in much the same way as they work with volatile memory.

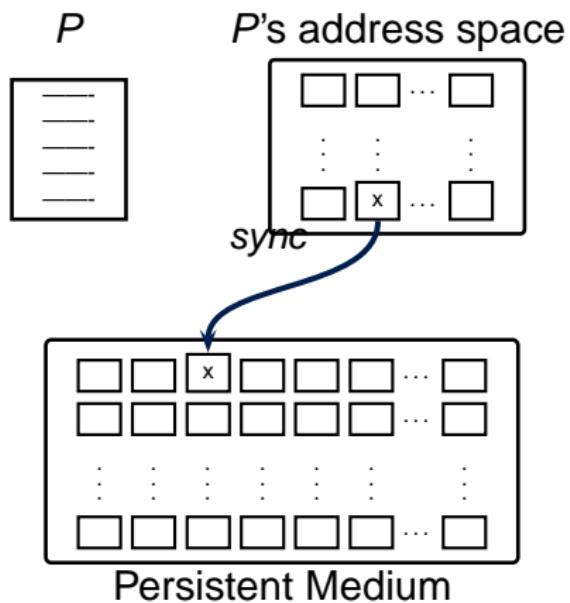
Software Persistent Memory (*SoftPM*)

Applications allocate, and interact with, persistent memory in much the same way as they work with volatile memory.

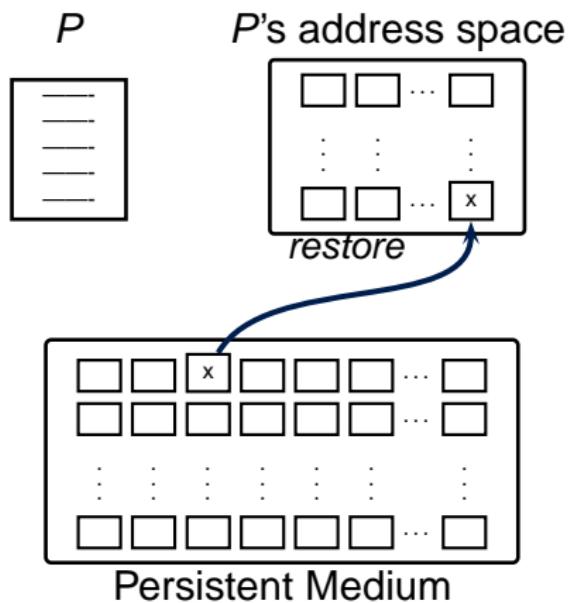
Two Persistence Notions in *SoftPM*

- ▶ *Durable* persistence
- ▶ *Immutable* persistence

High-level Idea behind SoftPM



High-level Idea behind SoftPM



The Container Abstraction

Simple container

- ▶ Self-describing unit of persistent memory
- ▶ One or more containers per application of arbitrary size
- ▶ Each persistent allocation is housed within a container
- ▶ Container wide persistence policy
- ▶ A container is restoreable with a single operation

The Container Abstraction

Simple container

- ▶ Self-describing unit of persistent memory
- ▶ One or more containers per application of arbitrary size
- ▶ Each persistent allocation is housed within a container
- ▶ Container wide persistence policy
- ▶ A container is restoreable with a single operation

Versioned container

- ▶ Each persistence point creates new container version
- ▶ Past versions are restoreable with a single operation
- ▶ Past versions are browsable
- ▶ Container version is branch-able

SoftPM API

API v0.1

- ▶ Container administration
 - `pCAlloc / pCFree / pCDelete`
- ▶ Container memory management
 - `pMalloc / pMFree`
- ▶ Container persistence management
 - `pCSetAttr / pCGetAttr / pPoint / pPointSync`
- ▶ Restoring and navigating through branches
 - `pCRestore / pCSwitch / pCClone`

Using SoftPM for Recoverability: An Example

```
typedef struct l {
    int v;
    struct l *n;
} list;
► a() {
    list *l = malloc(...);
    l->v = X;
    list *e1 = malloc(...);
    e1->v = Y;
    l->n = e1;
    list *e2 = malloc(...);
    e2->v = Z;
    e1->n = e2;
    :
}
```

Using SoftPM for Recoverability: An Example

```
typedef struct l {
    int v;
    struct l *n;
} list;
a() {
    ► list *l = malloc(...);
    l->v = X;
    list *e1 = malloc(...);
    e1->v = Y;
    l->n = e1;
    list *e2 = malloc(...);
    e2->v = Z;
    e1->n = e2;
    :
}
```



Using SoftPM for Recoverability: An Example

```
typedef struct l {
    int v;
    struct l *n;
} list;
a() {
    list *l = malloc(...);
    ► l->v = X;
    list *e1 = malloc(...);
    e1->v = Y;
    l->n = e1;
    list *e2 = malloc(...);
    e2->v = Z;
    e1->n = e2;
    :
}
```



Using SoftPM for Recoverability: An Example

```
typedef struct l {
    int v;
    struct l *n;
} list;
a() {
    list *l = malloc(...);
    l->v = X;
    ► list *e1 = malloc(...);
    e1->v = Y;
    l->n = e1;
    list *e2 = malloc(...);
    e2->v = Z;
    e1->n = e2;
    :
}
```



Using SoftPM for Recoverability: An Example

```
typedef struct l {
    int v;
    struct l *n;
} list;
a() {
    list *l = malloc(...);
    l->v = X;
    list *e1 = malloc(...);
    e1->v = Y;
    l->n = e1;
    list *e2 = malloc(...);
    e2->v = Z;
    e1->n = e2;
    :
}
```

X

Y

Using SoftPM for Recoverability: An Example

```
typedef struct l {
    int v;
    struct l *n;
} list;
a() {
    list *l = malloc(...);
    l->v = X;
    list *e1 = malloc(...);
    e1->v = Y;
    l->n = e1;
    list *e2 = malloc(...);
    e2->v = Z;
    e1->n = e2;
    :
}
```



Using SoftPM for Recoverability: An Example

```
typedef struct l {
    int v;
    struct l *n;
} list;
a() {
    list *l = malloc(...);
    l->v = X;
    list *e1 = malloc(...);
    e1->v = Y;
    l->n = e1;
    list *e2 = malloc(...);
    e2->v = Z;
    e1->n = e2;
    :
}
```



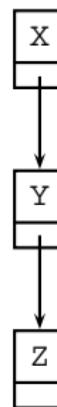
Using SoftPM for Recoverability: An Example

```
typedef struct l {
    int v;
    struct l *n;
} list;
a() {
    list *l = malloc(...);
    l->v = X;
    list *e1 = malloc(...);
    e1->v = Y;
    l->n = e1;
    list *e2 = malloc(...);
    e2->v = Z;
    e1->n = e2;
    :
}
```



Using SoftPM for Recoverability: An Example

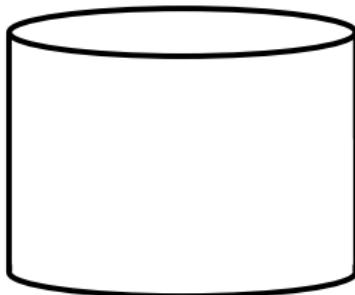
```
typedef struct l {
    int v;
    struct l *n;
} list;
a() {
    list *l = malloc(...);
    l->v = X;
    list *e1 = malloc(...);
    e1->v = Y;
    l->n = e1;
    list *e2 = malloc(...);
    e2->v = Z;
    e1->n = e2;
    :
}
```



Creating a Persistence Point

```
► struct pcont {  
    list *l;  
} C;  
  
► a() {  
    if (!(C = pCRestore(...)))  
        C = pCAlloc(...);  
    C->l = pmalloc(C, ...);  
    C->l->v = X;  
    list *e1 = pmalloc(C, ...);  
    e1->v = Y;  
    C->l->n = e1;  
    list *e2 = pmalloc(C, ...);  
    e2->v = Z;  
    e1->n = e2;  
    pPoint(C);  
}  
:  
}
```

Process Memory

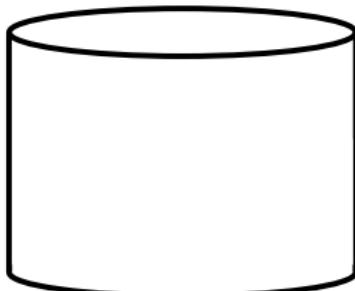
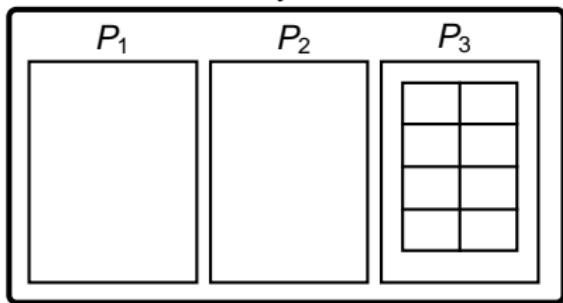


Persistent Medium

Creating a Persistence Point

```
► struct pcont {  
    list *l;  
} C;  
a() {  
    if (!(C = pCRestore(...)))  
        C = pCAlloc(...);  
    C->l = pmalloc(C, ...);  
    C->l->v = X;  
    list *e1 = pmalloc(C, ...);  
    e1->v = Y;  
    C->l->n = e1;  
    list *e2 = pmalloc(C, ...);  
    e2->v = Z;  
    e1->n = e2;  
    pPoint(C);  
}  
:  
}
```

Process Memory

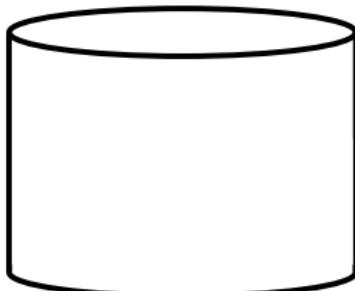
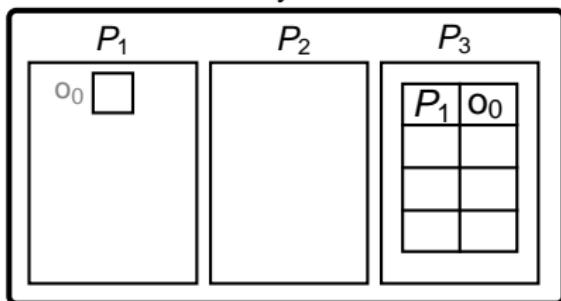


Persistent Medium

Creating a Persistence Point

```
► struct pcont {  
    list *l;  
} C;  
a() {  
    if (!(C = pCRestore(...)))  
        C = pCAlloc(...);  
    C->l = pmalloc(C, ...);  
    C->l->v = X;  
    list *e1 = pmalloc(C, ...);  
    e1->v = Y;  
    C->l->n = e1;  
    list *e2 = pmalloc(C, ...);  
    e2->v = Z;  
    e1->n = e2;  
    pPoint(C);  
}  
:  
}
```

Process Memory

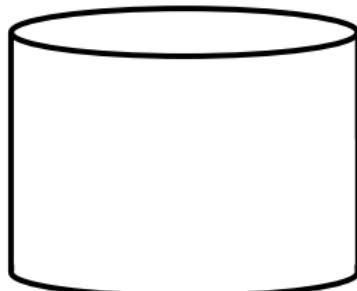
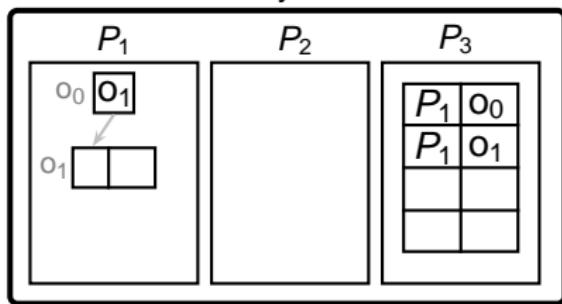


Persistent Medium

Creating a Persistence Point

```
► struct pcont {  
    list *l;  
} C;  
a() {  
    if (!(C = pCRestore(...)))  
        C = pCAlloc(...);  
    C->l = pmalloc(C, ...);  
    C->l->v = X;  
    list *e1 = pmalloc(C, ...);  
    e1->v = Y;  
    C->l->n = e1;  
    list *e2 = pmalloc(C, ...);  
    e2->v = Z;  
    e1->n = e2;  
    pPoint(C);  
}  
:  
}
```

Process Memory

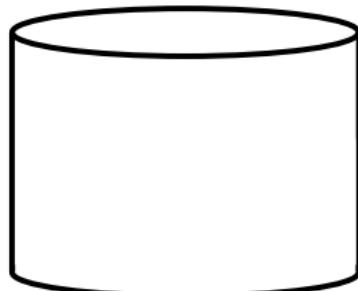
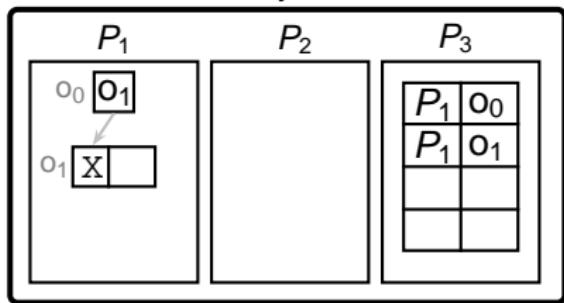


Persistent Medium

Creating a Persistence Point

```
► struct pcont {  
    list *l;  
} C;  
a() {  
    if (!(C = pCRestore(...)))  
        C = pCAlloc(...);  
    C->l = pmalloc(C, ...);  
    C->l->v = X;  
    list *e1 = pmalloc(C, ...);  
    e1->v = Y;  
    C->l->n = e1;  
    list *e2 = pmalloc(C, ...);  
    e2->v = Z;  
    e1->n = e2;  
    pPoint(C);  
}  
:  
}
```

Process Memory

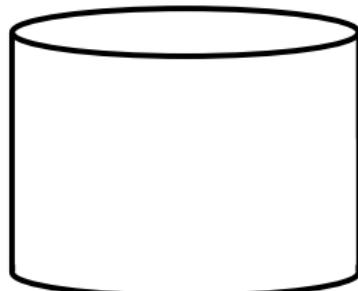
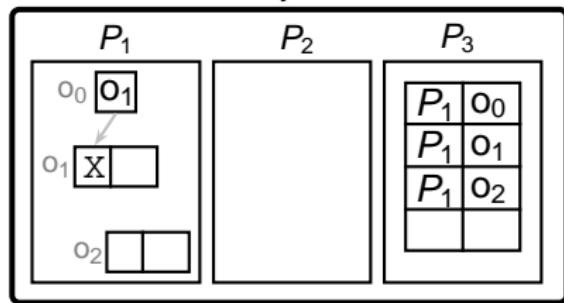


Persistent Medium

Creating a Persistence Point

```
► struct pcont {  
    list *l;  
} C;  
a() {  
    if (!(C = pCRestore(...)))  
        C = pCAlloc(...);  
    C->l = pmalloc(C, ...);  
    C->l->v = X;  
    list *e1 = pmalloc(C, ...);  
    e1->v = Y;  
    C->l->n = e1;  
    list *e2 = pmalloc(C, ...);  
    e2->v = Z;  
    e1->n = e2;  
    pPoint(C);  
}  
:  
}
```

Process Memory

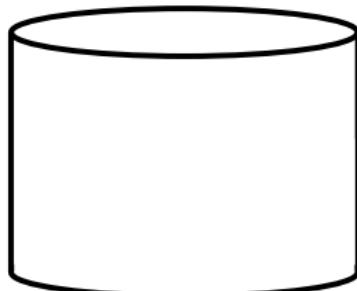
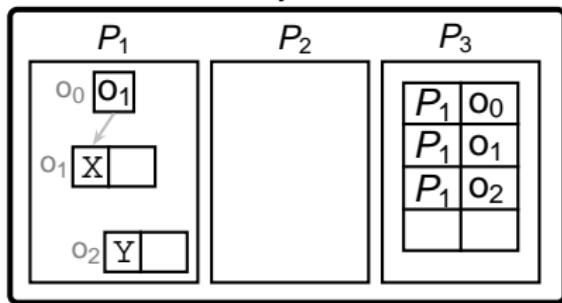


Persistent Medium

Creating a Persistence Point

```
struct pcont {
    list *l;
} C;
a() {
    if (!(C = pCRestore(...)))
        C = pCAlloc(...);
    C->l = pmalloc(C, ...);
    C->l->v = X;
    list *e1 = pmalloc(C, ...);
    e1->v = Y;
    C->l->n = e1;
    list *e2 = pmalloc(C, ...);
    e2->v = Z;
    e1->n = e2;
    pPoint(C);
}
:
```

Process Memory

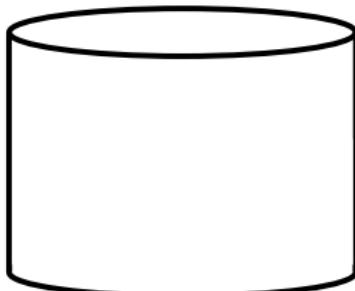
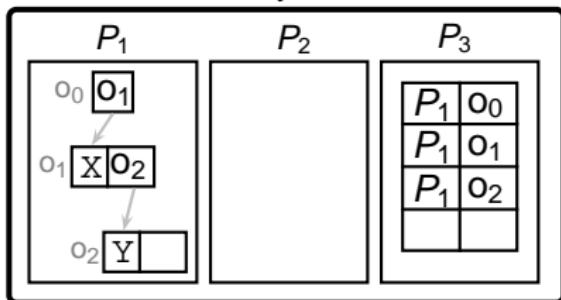


Persistent Medium

Creating a Persistence Point

```
struct pcont {
    list *l;
} C;
a() {
    if (!(C = pCRestore(...)))
        C = pCAlloc(...);
    C->l = pmalloc(C, ...);
    C->l->v = X;
    list *e1 = pmalloc(C, ...);
    e1->v = Y;
    C->l->n = e1;
    list *e2 = pmalloc(C, ...);
    e2->v = Z;
    e1->n = e2;
    pPoint(C);
}
:
```

Process Memory

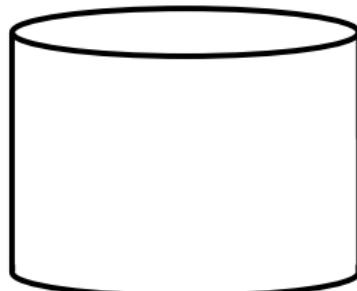
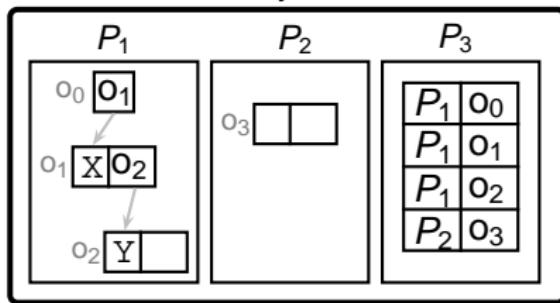


Persistent Medium

Creating a Persistence Point

```
struct pcont {
    list *l;
} C;
a() {
    if (!(C = pCRestore(...)))
        C = pCAlloc(...);
    C->l = pmalloc(C, ...);
    C->l->v = X;
    list *e1 = pmalloc(C, ...);
    e1->v = Y;
    C->l->n = e1;
    list *e2 = pmalloc(C, ...);
    e2->v = Z;
    e1->n = e2;
    pPoint(C);
}
:
```

Process Memory

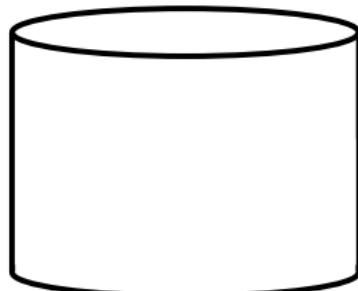
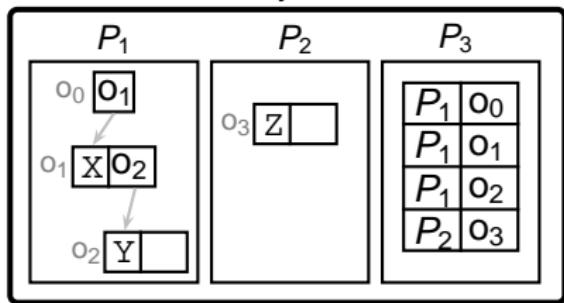


Persistent Medium

Creating a Persistence Point

```
struct pcont {
    list *l;
} C;
a() {
    if (!(C = pCRestore(...)))
        C = pCAlocate(...);
    C->l = pmalloc(C, ...);
    C->l->v = X;
    list *e1 = pmalloc(C, ...);
    e1->v = Y;
    C->l->n = e1;
    list *e2 = pmalloc(C, ...);
    e2->v = Z;
    e1->n = e2;
    pPoint(C);
}
:
```

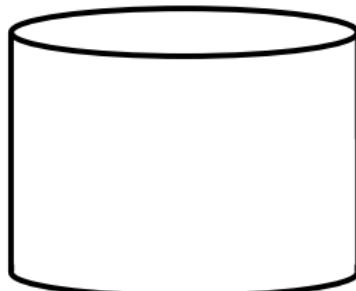
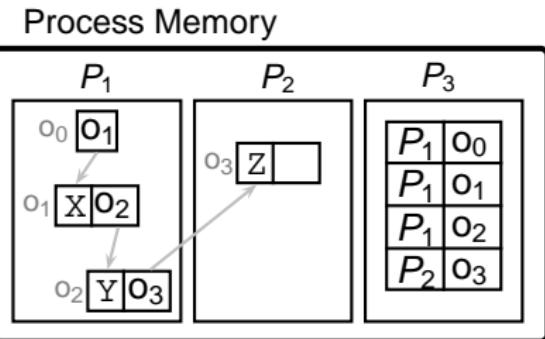
Process Memory



Persistent Medium

Creating a Persistence Point

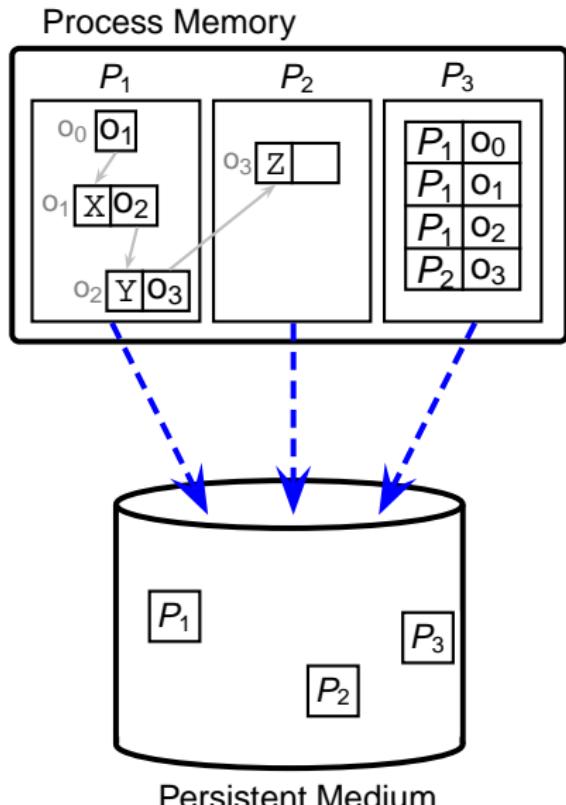
```
struct pcont {
    list *l;
} C;
a() {
    if (!(C = pCRestore(...)))
        C = pCAlloc(...);
    C->l = pmalloc(C, ...);
    C->l->v = X;
    list *e1 = pmalloc(C, ...);
    e1->v = Y;
    C->l->n = e1;
    list *e2 = pmalloc(C, ...);
    e2->v = Z;
    e1->n = e2;
    pPoint(C);
}
:
```



Persistent Medium

Creating a Persistence Point

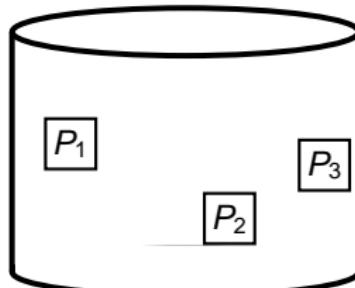
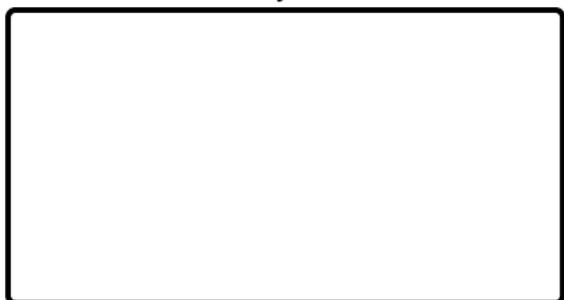
```
struct pcont {
    list *l;
} C;
a() {
    if (!(C = pCRestore(...)))
        C = pCAlloc(...);
    C->l = pmalloc(C, ...);
    C->l->v = X;
    list *e1 = pmalloc(C, ...);
    e1->v = Y;
    C->l->n = e1;
    list *e2 = pmalloc(C, ...);
    e2->v = Z;
    e1->n = e2;
    pPoint(C);
}
:
```



Restoring a Persistence Point

```
► struct pcont {  
    list *l;  
} C;  
a() {  
    if (!(C = pCRestore(...)))  
        :  
    }  
    :  
}
```

Process Memory



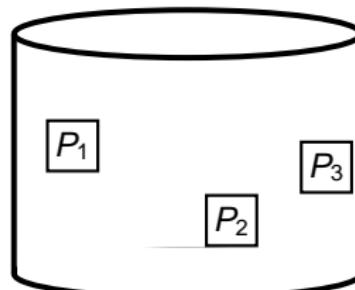
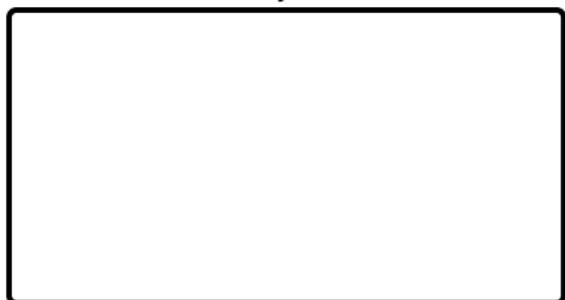
Persistent Medium

Restoring a Persistence Point

```
► struct pcont {  
    list *l;  
} C;  
a() {  
    if (!(C = pCRestore(...)))  
        :  
    }  
    :  
}
```

Prev	Curr

Process Memory

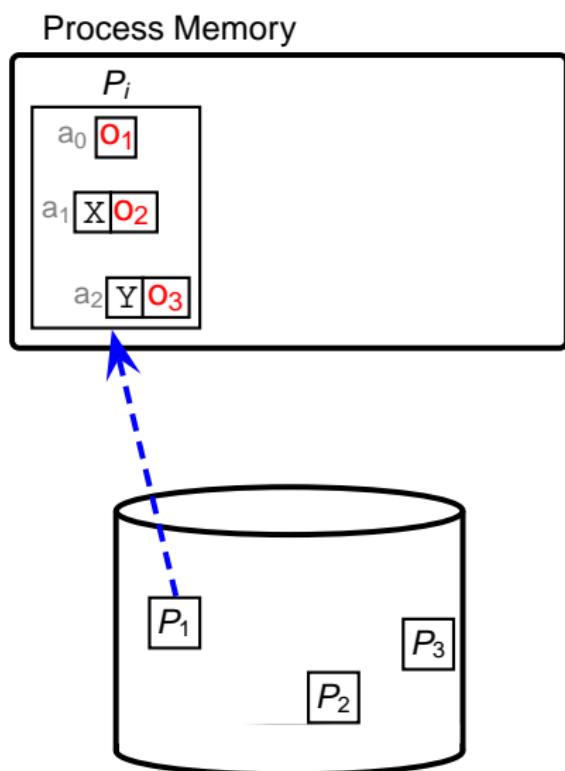


Persistent Medium

Restoring a Persistence Point

```
► struct pcont {  
    list *l;  
} C;  
a() {  
    if (!(C = pCRestore(...)))  
        ...  
    }  
    ...  
}
```

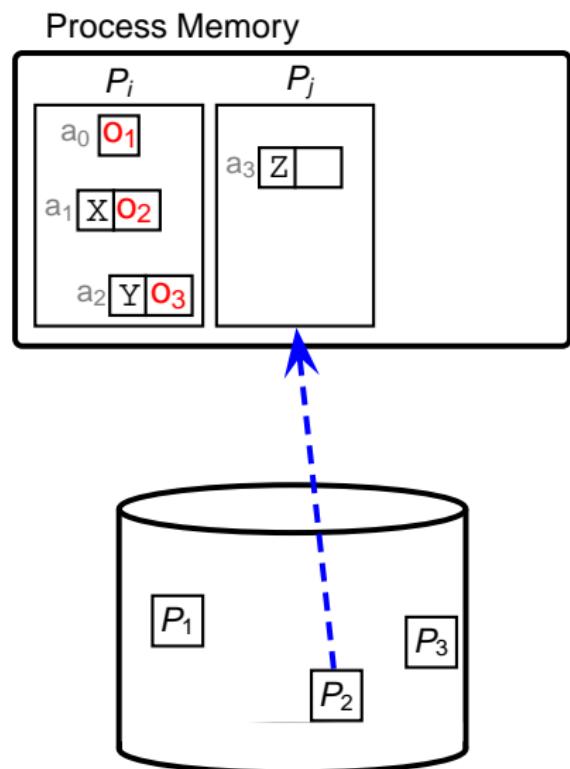
Prev	Curr
P_1	P_i



Restoring a Persistence Point

```
► struct pcont {  
    list *l;  
} C;  
a() {  
    if (!(C = pCRestore(...)))  
        ...  
    }  
    ...  
}
```

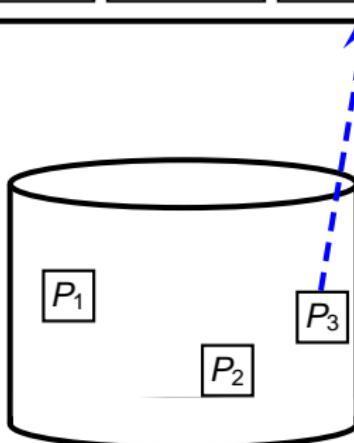
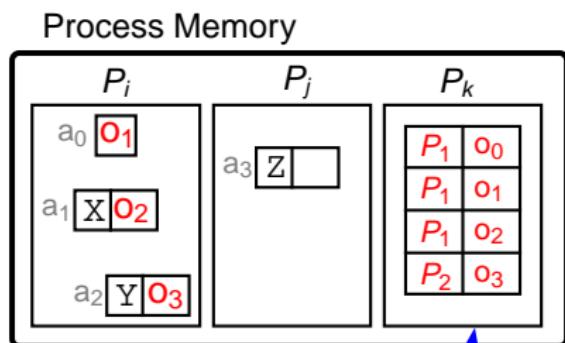
Prev	Curr
P_1	P_i
P_2	P_j



Restoring a Persistence Point

```
► struct pcont {  
    list *l;  
} C;  
a() {  
    if (!(C = pCRestore(...)))  
        ...  
    }  
    ...  
}
```

Prev	Curr
P_1	P_i
P_2	P_j
P_3	P_k



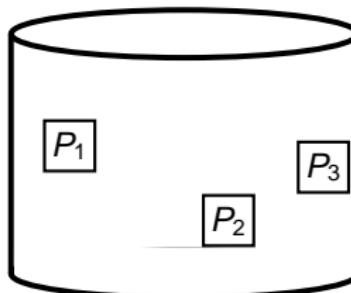
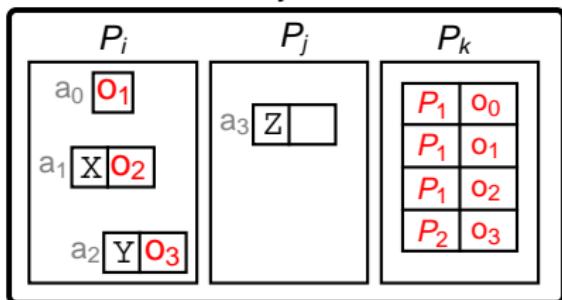
Restoring a Persistence Point

```
► struct pcont {  
    list *l;  
} C;  
a() {  
    if (!(C = pCRestore(...)))  
        ...  
    }  
    ...  
}
```

►

Prev	Curr
P_1	P_i
P_2	P_j
P_3	P_k

Process Memory



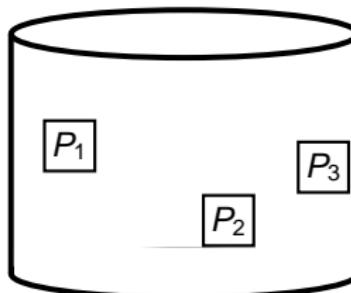
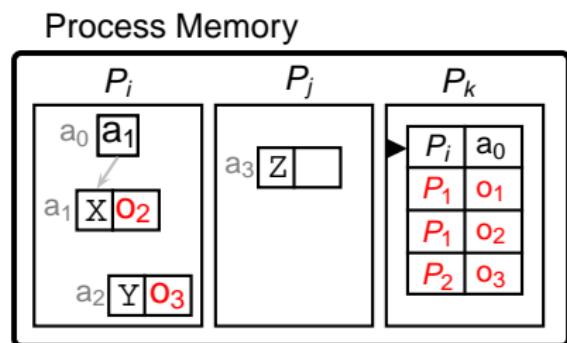
Persistent Medium

Restoring a Persistence Point

```
► struct pcont {  
    list *l;  
} C;  
a() {  
    if (!(C = pCRestore(...)))  
        ...  
    }  
    ...  
}
```

►

Prev	Curr
P_1	P_i
P_2	P_j
P_3	P_k

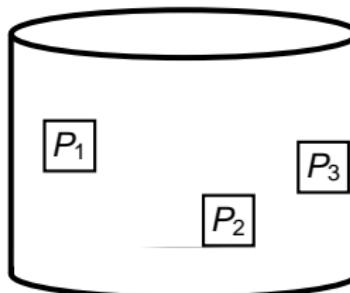
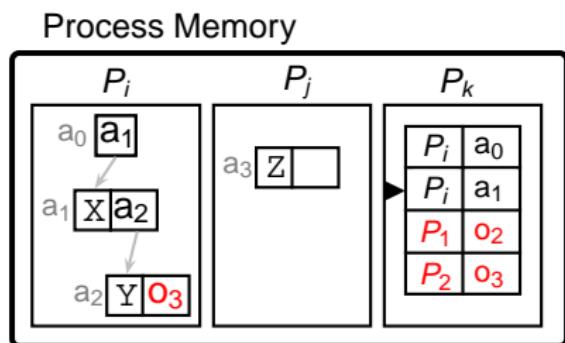


Restoring a Persistence Point

```
► struct pcont {  
    list *l;  
} C;  
a() {  
    if (!(C = pCRestore(...)))  
        ...  
    }  
    ...  
}
```

►

Prev	Curr
P_1	P_i
P_2	P_j
P_3	P_k



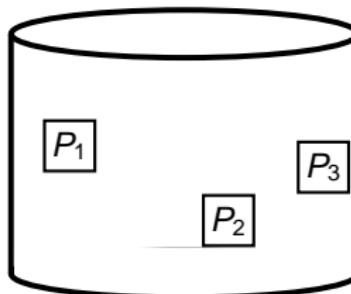
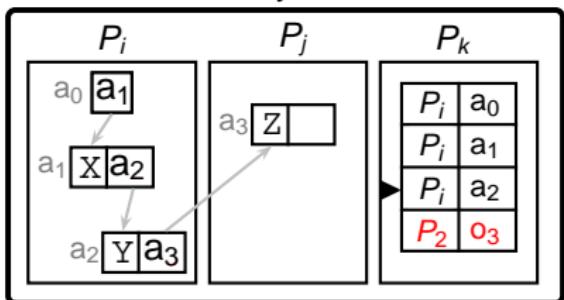
Restoring a Persistence Point

```
► struct pcont {  
    list *l;  
} C;  
a() {  
    if (!(C = pCRestore(...)))  
        ...  
    }  
    ...  
}
```

►

Prev	Curr
P_1	P_i
P_2	P_j
P_3	P_k

Process Memory

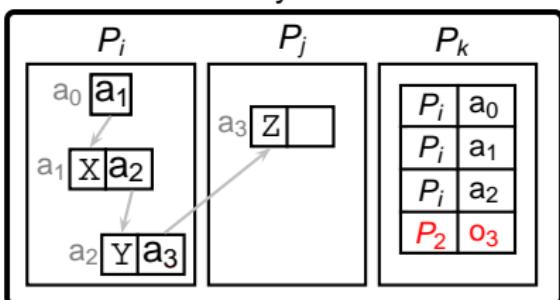


Persistent Medium

Restoring a Persistence Point

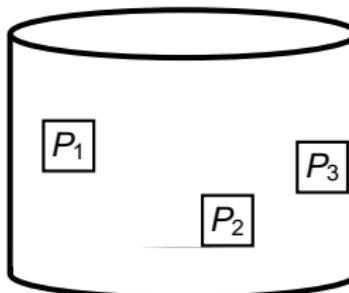
```
► struct pcont {  
    list *l;  
} C;  
a() {  
    if (!(C = pCRestore(...)))  
        ...  
    }  
    ...  
}
```

Process Memory



A table mapping previous persistence points to current ones:

Prev	Curr
P_1	P_i
P_2	P_j
P_3	P_k

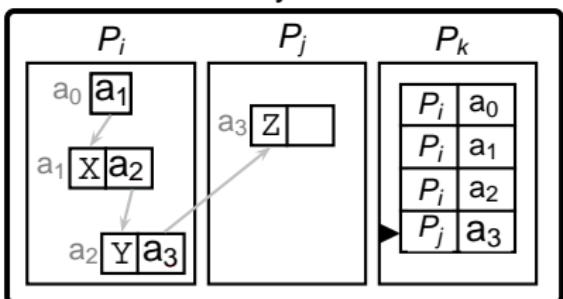


Persistent Medium

Restoring a Persistence Point

```
► struct pcont {  
    list *l;  
} C;  
a() {  
    if (!(C = pCRestore(...)))  
        ...  
    }  
    ...  
}
```

Process Memory



A cylinder labeled "Persistent Medium" contains objects P_1 , P_2 , and P_3 . Arrows point from the cylinder to a table below.

Prev	Curr
P_1	P_i
P_2	P_j
P_3	P_k

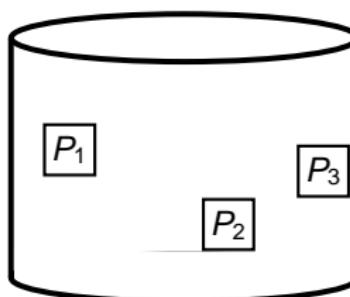
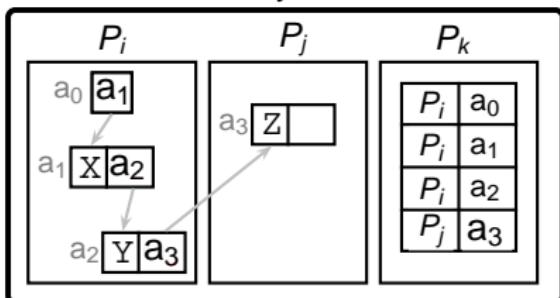
Restoring a Persistence Point

```
► struct pcont {  
    list *l;  
} C;  
a() {  
    if (!(C = pCRestore(...)))  
        ...  
    }  
    ...  
}
```

►

Prev	Curr
P_1	P_i
P_2	P_j
P_3	P_k

Process Memory



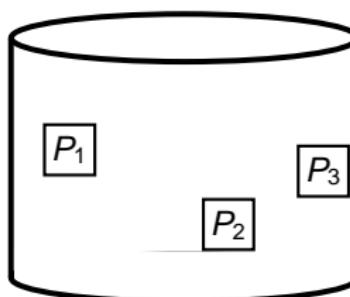
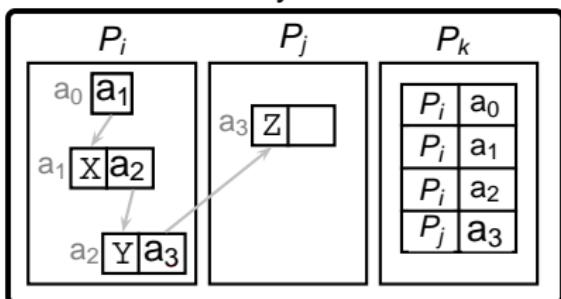
Persistent Medium

Restoring a Persistence Point

```
struct pcont {  
    list *l;  
} C;  
a() {  
    if (!(C = pCRestore(...)))  
        ...  
    }  
    ...  
}
```

Prev	Curr
P_1	P_i
P_2	P_j
P_3	P_k

Process Memory



Persistent Medium

Persistent Data Management Challenges

Challenges

- ▶ Self-describing containers
- ▶ Automatic change detection and updating
- ▶ Container versioning and branching support

Persistent Data Management Challenges

Challenges

- ▶ Self-describing containers
- ▶ Automatic change detection and updating
- ▶ Container versioning and branching support

Key Ideas

- ▶ Container address-space of aligned container pages
- ▶ Multi-level indexing (memory page, disk offset, dirty-bit)
- ▶ Container root struct provides access to entire container
- ▶ Compiler / pre-processor assisted pointer tracking
- ▶ Auxiliary container metadata in container address-space
- ▶ Read-only pages until dirty
- ▶ Read-only container restoration and writeable clones

Storage Management Challenges

Challenges

- ▶ High-performance I/O
- ▶ Atomic persistence points
- ▶ Scalability

Storage Management Challenges

Challenges

- ▶ High-performance I/O
- ▶ Atomic persistence points
- ▶ Scalability

Key Ideas

- ▶ Chunk-remapping writes
- ▶ Asynchronous I/O with barrier write containing commit ID
- ▶ Data sieving and prefetching for browsing
- ▶ PFS delegation
- ▶ Collective data access

Preliminary Evaluation

Microbenchmark

- ▶ Single system, single disk
- ▶ Random memory updates, Vary Tx size, and # containers

Preliminary Evaluation

Microbenchmark

- ▶ Single system, single disk
- ▶ Random memory updates, Vary Tx size, and # containers

Storage Layers

- ▶ **LOG**: write transactions sequentially in an infinite log.
- ▶ **Logging-Checkpointing Writer (LCW)**: Traditional write-ahead log
- ▶ **CREW**: *SoftPM*'s hard disk I/O driver in the SID layer.

Preliminary Evaluation

Microbenchmark

- ▶ Single system, single disk
- ▶ Random memory updates, Vary Tx size, and # containers

Storage Layers

- ▶ **LOG**: write transactions sequentially in an infinite log.
- ▶ **Logging-Checkpointing Writer (LCW)**: Traditional write-ahead log
- ▶ **CREW**: *SoftPM*'s hard disk I/O driver in the SID layer.

Results Summary – Details available offline

- ▶ Sweet-spot in CREW chunk size
- ▶ CREW 2X worse than LOG, but 3X better than LCW
- ▶ Results scale with Tx size and # containers

Summary

SoftPM simplifies software for HEC applications via

- ▶ A transformative and enabling abstraction
- ▶ Fully automated persistence
- ▶ High-performance persistence

Summary

SoftPM simplifies software for HEC applications via

- ▶ A transformative and enabling abstraction
- ▶ Fully automated persistence
- ▶ High-performance persistence

SoftPM applications beyond HEC

- ▶ Self-managing storage
- ▶ Workflows of chained applications
- ▶ File systems
- ▶ Databases

Next Steps

Short term

- ▶ Automated pointer detection
- ▶ Design for solid-state drives
- ▶ HEC application case-studies
- ▶ Experience with large-scale

Long term

- ▶ Programming language portability issues
- ▶ Lightweight OS restrictions
- ▶ New persistence capabilities

Acknowledgements

Collaborators

Ming Zhao

Jason Liu

Project Co-PI @ FIU

Project Co-PI @ FIU

Jorge Guerra

Mike Torchio

PhD student @ FIU

Undergraduate student @ U Vermont

Problem
oo

Approach
oooo

An Example
ooo

Challenges
oo

Evaluation
o

Summary
ooo

Thank you!