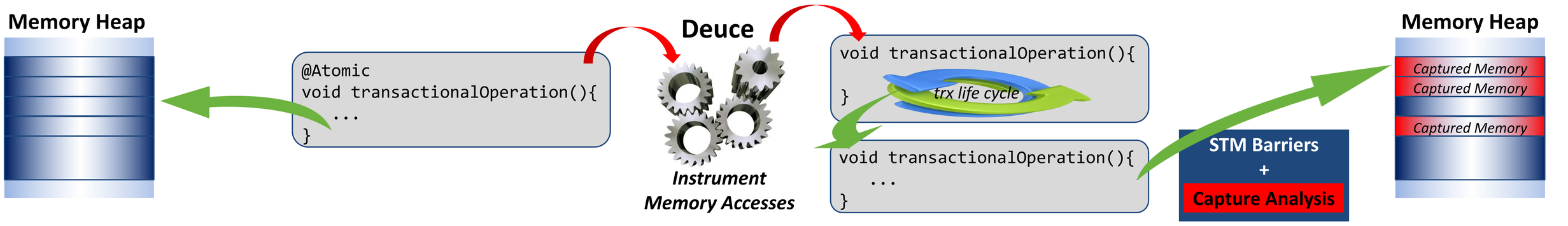
Runtime Elision of Transactional Barriers for Captured Memory

Fernando Miguel Carvalho mcarvalho@cc.isel.ipl.pt **João Cachopo** joao.cachopo@ist.utl.pt

INESC-ID Lisboa / Instituto Superior Técnico / Technical University of Lisbon

Overview



Motivation and problem statement

STM frameworks may provide a Transparent API (e.g. Deuce):

```
@Atomic
public void transactionalOperation(){
   ...
}
```

Which data is shared from within an atomic operation?

Overzealous STM compilers may protect every memory access with an STM barrier, even for unshared data!!!

Loss of performance

Remembering past proposals

Heterogeneous API examples:

```
void m(){
   ...
   opAccessUnsharedData();
}
```

```
@tm_waiver
void opAccessUnsharedData(){
   ...
}
```

```
void m(){
   ...
   ImmutableClass.op();
}
```

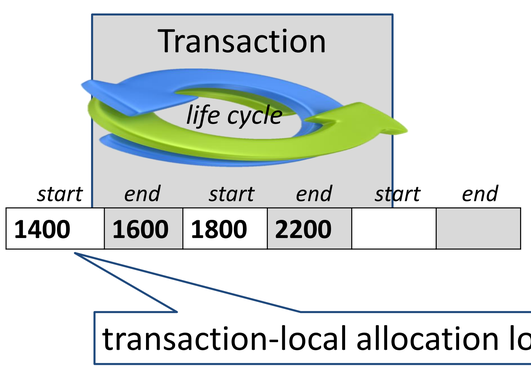
```
@Exclude
Class ImmutableClass{
  ...
}
```

Performance

Burdens programmers

vs

- **Static Analysis**
- **Runtime Capture Analysis:**

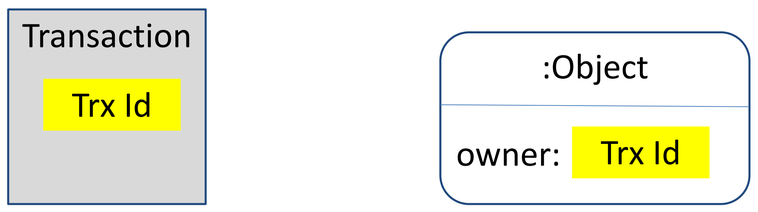


Automatic

Performance

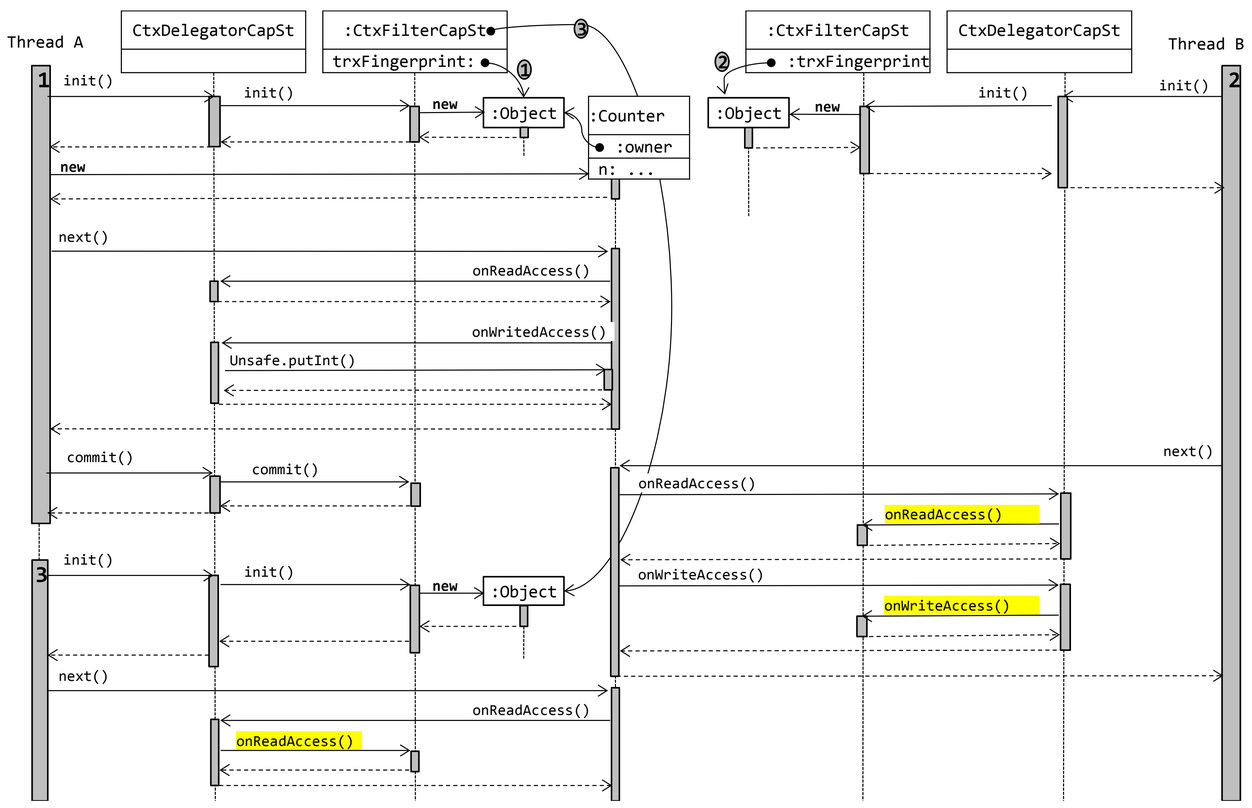
Idea

+ Labeling objects with the allocating transaction ID.

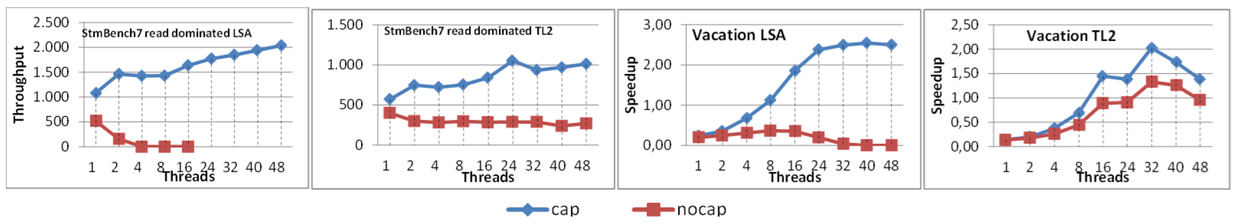


+ Rely on the GC to recycle IDs, avoiding additional synchronization.

Lightweight runtime capture analysis



Experimental results



Automatic STM barriers elision on:

- Iterators => STMBench7 operations traverse a large graph of objects, leading to an intensive use of collection iterators.
- Auxiliary arrays => several parameterized arrays provide the required arguments for the execution of the Vacation operations.

Acknowledgements

This work was supported by national funds through FCT - Fundação para a Ciência e a Tecnologia, both under project PEst-OE/EEI/LA0021/2011 and under project PTDC/EIA-EIA/108240/2008 (the RuLAM project).

PPoPP'13, February 23–27, 2013, Shenzhen, China.
ACM 978-1-4503-1922/13/02.