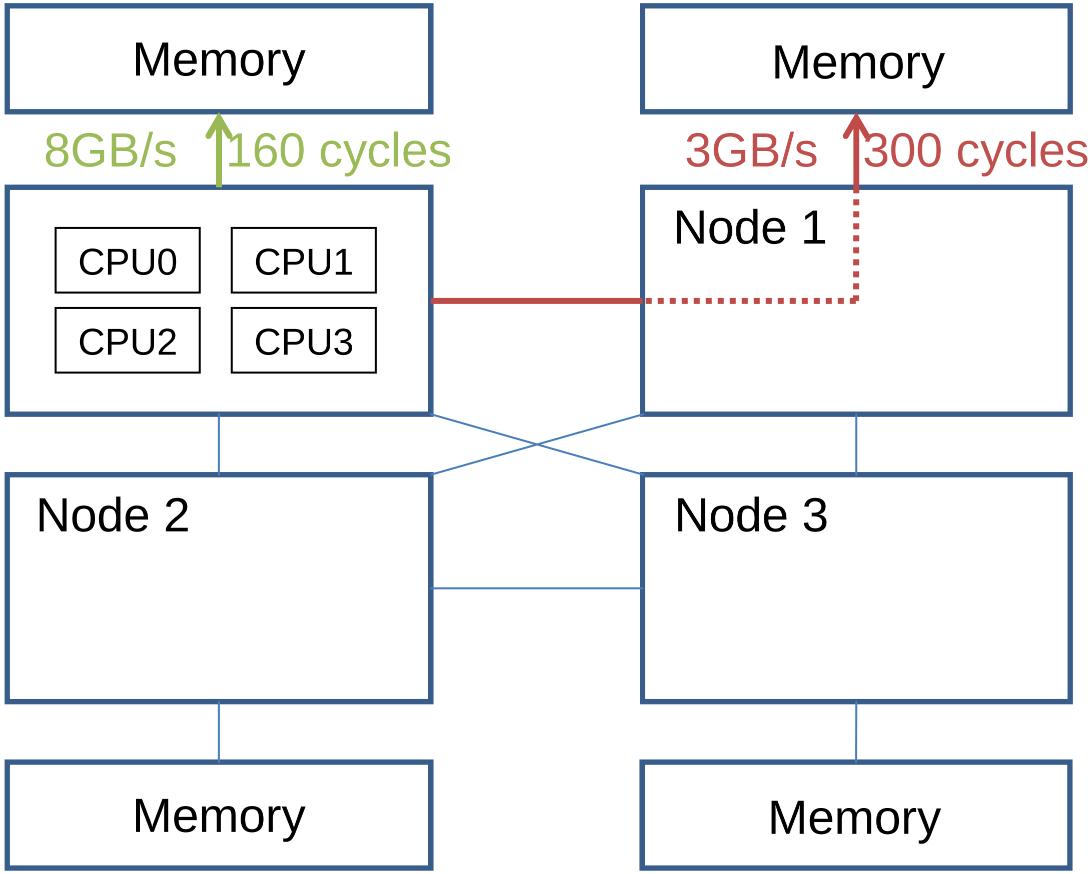
MemProf: a Memory Profiler for NUMA Multicore Systems

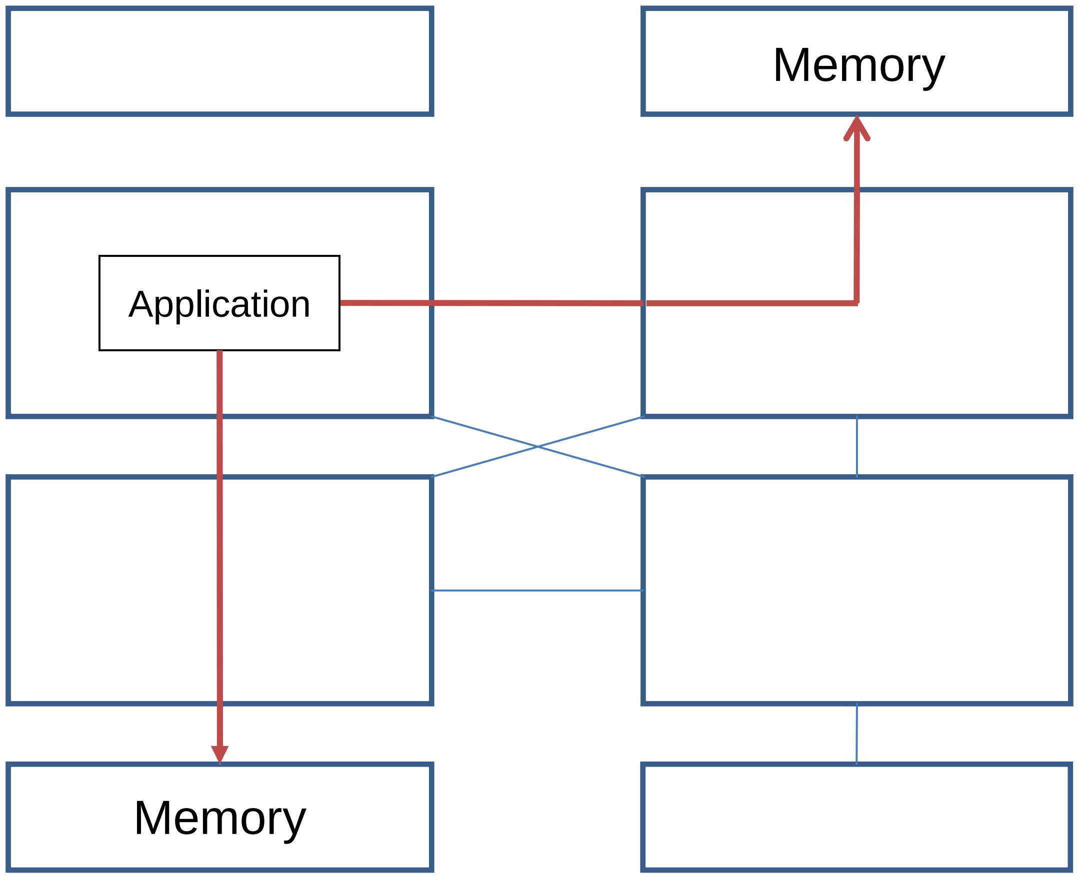
Renaud Lachaize, Baptiste Lepers, Vivien Quéma

(presented at USENIX ATC 2012)

Machines are NUMA



Applications ignore NUMA



That is problematic

Application	% remote memory accesses in default version
FaceRec (ALPBench)	63%
Streamcluster (Parsec)	75%
Psearchy (Mosbench)	13%
Apache	75%

That is problematic

Application	% remote memory accesses in default version	% performance improvement provided by an adequate NUMA optimization
FaceRec (ALPBench)	63%	42%
Streamcluster (Parsec)	75%	161%
Psearchy (Mosbench)	13%	8.2%
Apache	75%	20%

Application-Agnostic Heuristics exist

- Thread scheduling and page migration *(USENIX ATC'11)*
- Thread Clustering *(EuroSys'07)*
- Page replication *(ASPLOS'96)*
- Etc.

… But they do not always improve performance

Example: Apache

	% remote memory accesses	% performance impact over default version
On default Linux	75%	-
With thread scheduling and migration (USENIX'11)	75%	-5%

We want to understand the causes of remote memory accesses

… In order to select an adequate optimization

- Custom allocation policy
- Memory replication
- Memory migration
- Memory interleaving
- Custom thread scheduling policy

Can we understand the causes of
remote memory accesses
using existing profilers?

Let’s take an example

FaceRec

- Facial recognition engine
- 63% of DRAM accesses are remote
- 42% gain when modified based on MemProf output

Existing profilers point out

- The functions that perform remote accesses
- The memory pages that are remotely accessed
- The global static objects that are remotely accessed

Existing profilers point out (FaceRec)

- The functions that perform remote accesses
  - *transposeMultiplyMatrixL = 98%*
- The memory pages that are remotely accessed
  - 1/3 of the allocated pages
- The global static objects that are remotely accessed
  - No such object

What can we conclude?

- Should we change the allocation policy?
  - No idea

- Should we migrate memory pages?
  - No idea

- Should we replicate memory pages?
  - No idea

- Etc.

So… We need a new profiler!

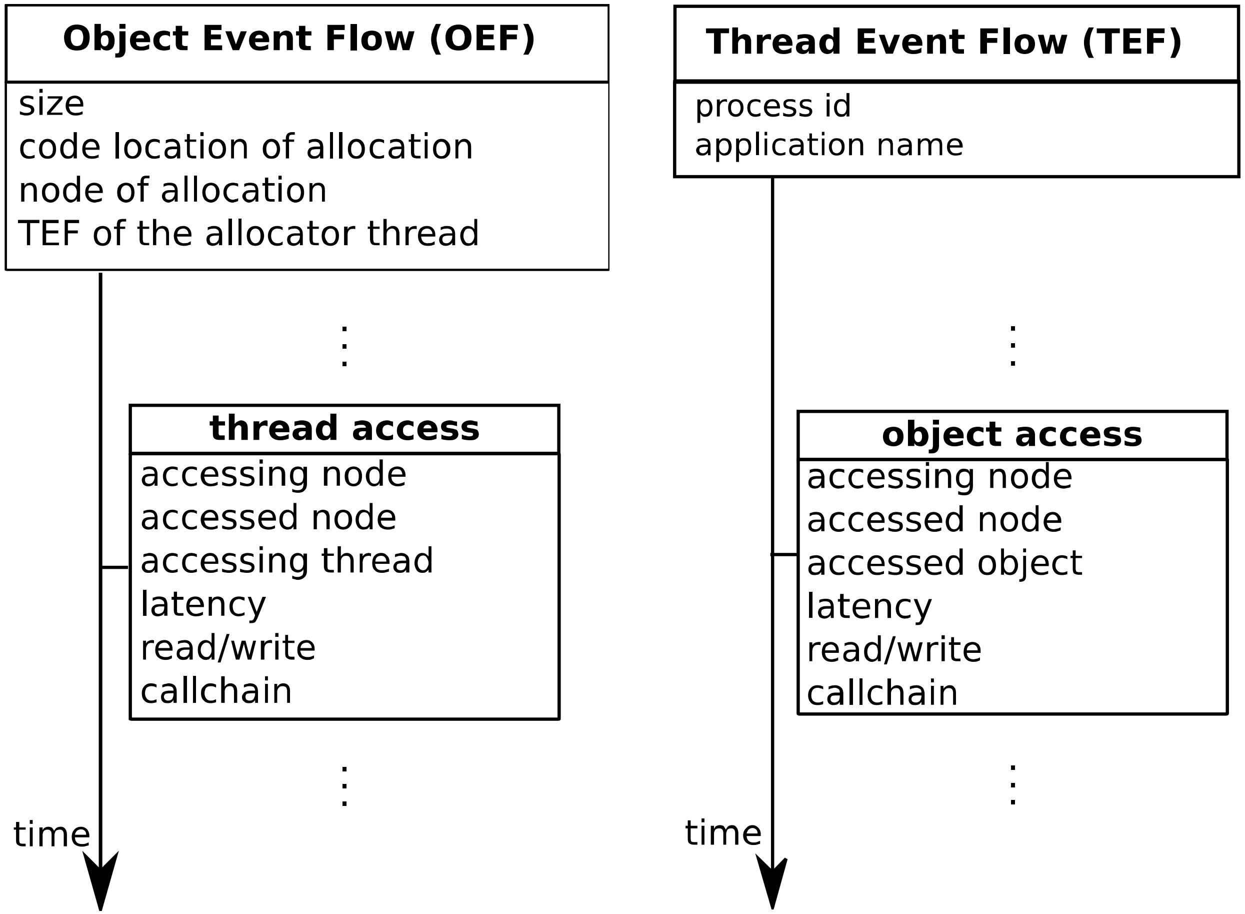
We designed MemProf, a profiler that points out

- Remotely accessed objects
- Thread-Object interaction patterns

Objects

- Global statically allocated objects
- Dynamically allocated objects
- Memory-mapped files
- Code sections mapped by the OS
- Thread stacks

Thread-Object interaction patterns



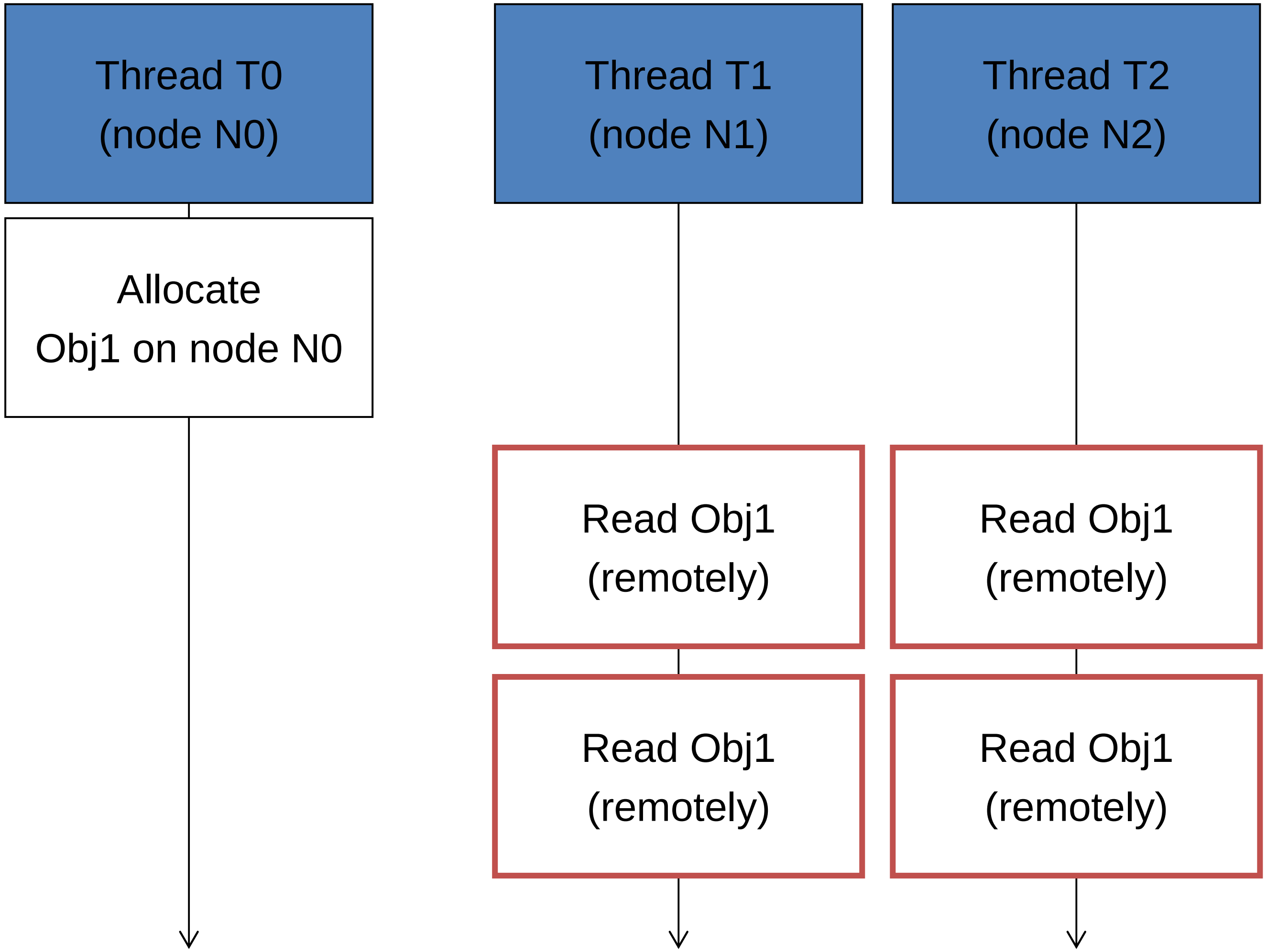
Scripting

- A simple script computing the average time between two memory accesses by distinct threads to an object

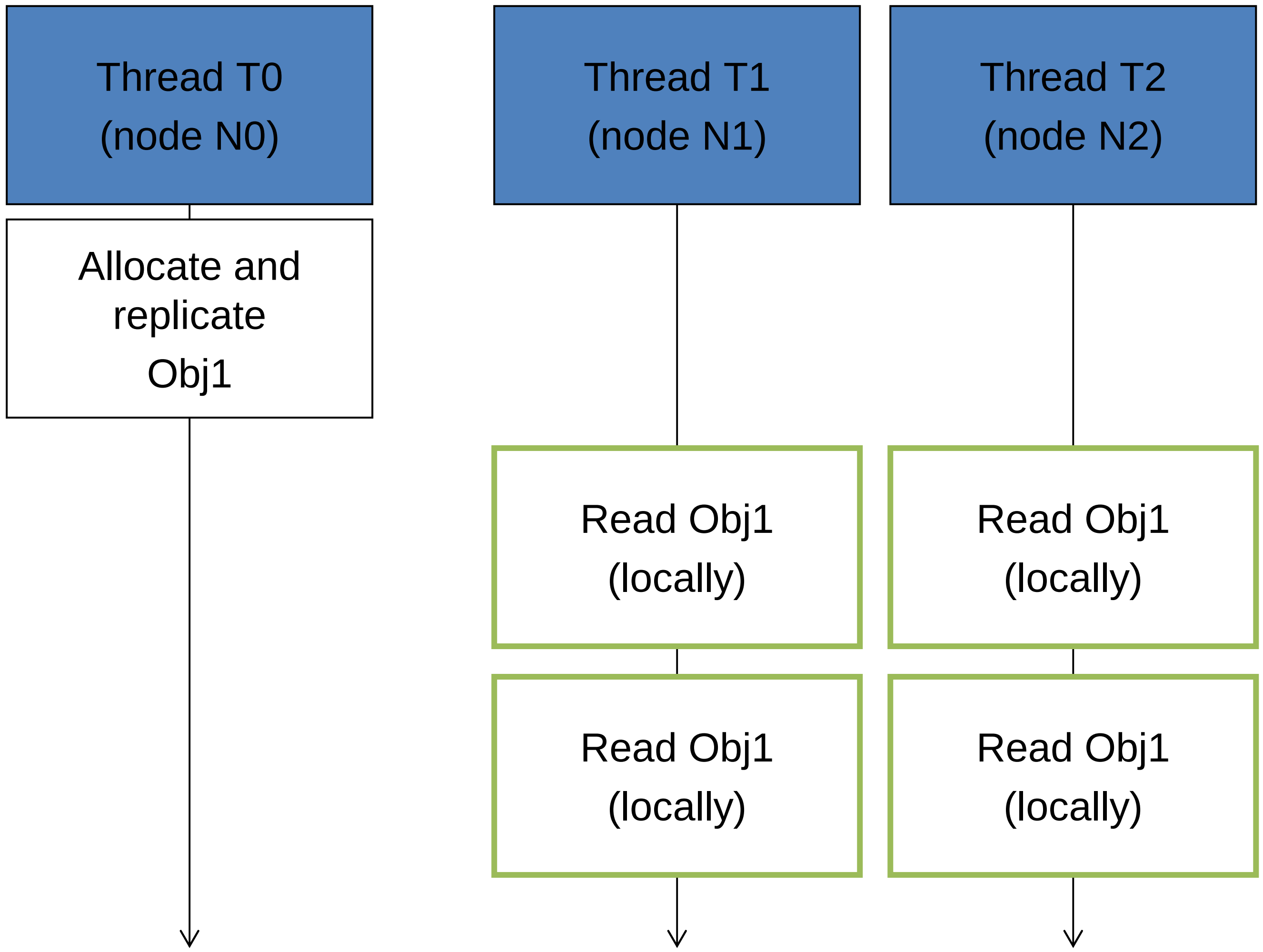
```
oef o = ...;
thread_acces a;
u64 last_tid = 0, last_rdt = 0;
u64 nb_tid_switch = 0;
u64 time_per_tid = 0;
foreach_taccess(o, a) {
  if(a.tid == last_tid)
    continue;
  nb_tid_switch++;
  time_per_tid += a.rdt - last_rdt;
  last_tid = a.tid;
  last_rdt = a.rdt;
}
printf("Avg time: %lu cycles (%lu switches)\n",
    time_per_tid/nb_tid_switch, nb_tid_switch);
```

What can we do with MemProf?

We can detect that an object is simultaneously read by several remote threads…



And thus decide to replicate this object on several nodes

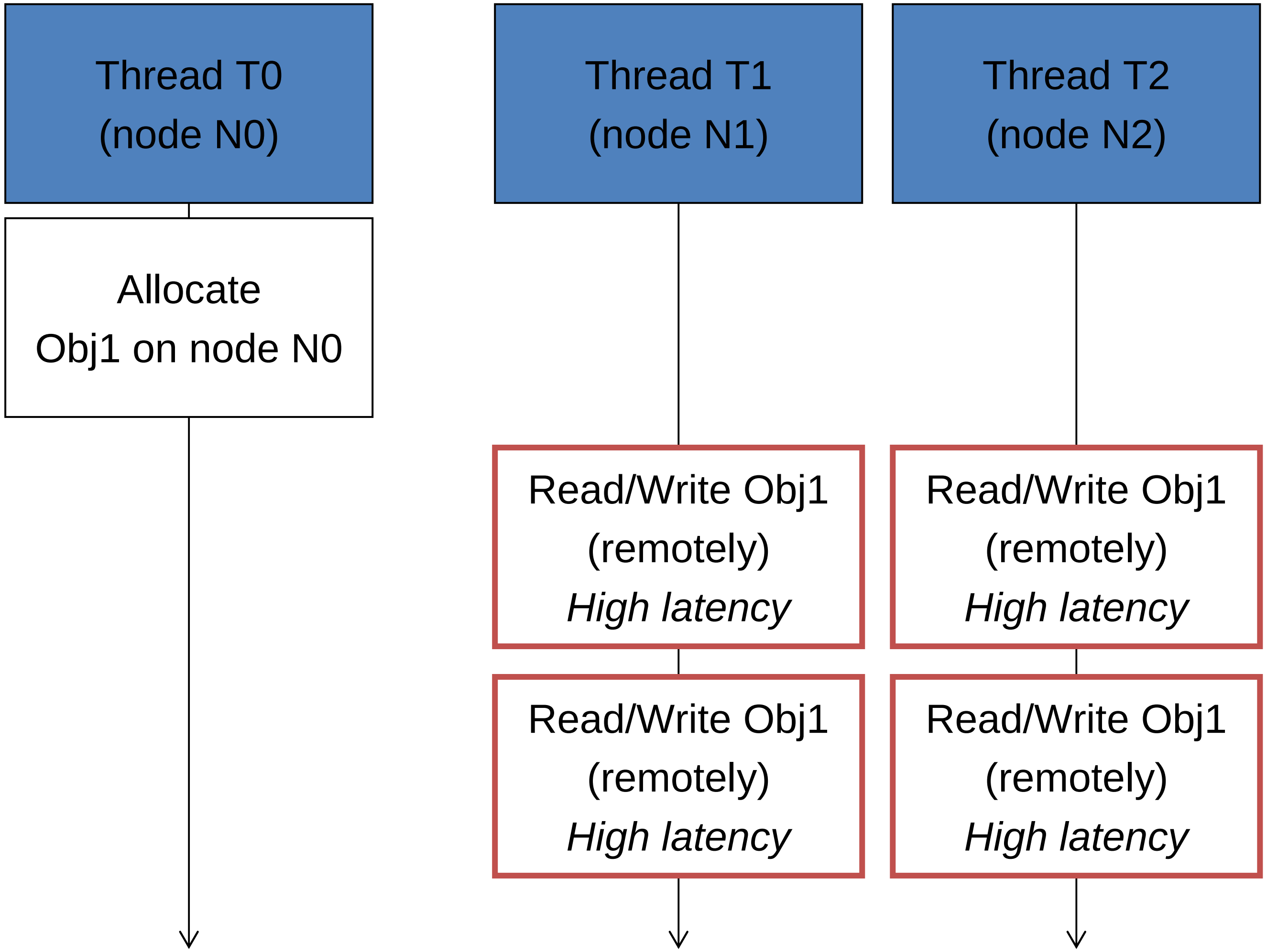


This is the pattern observed in FaceRec

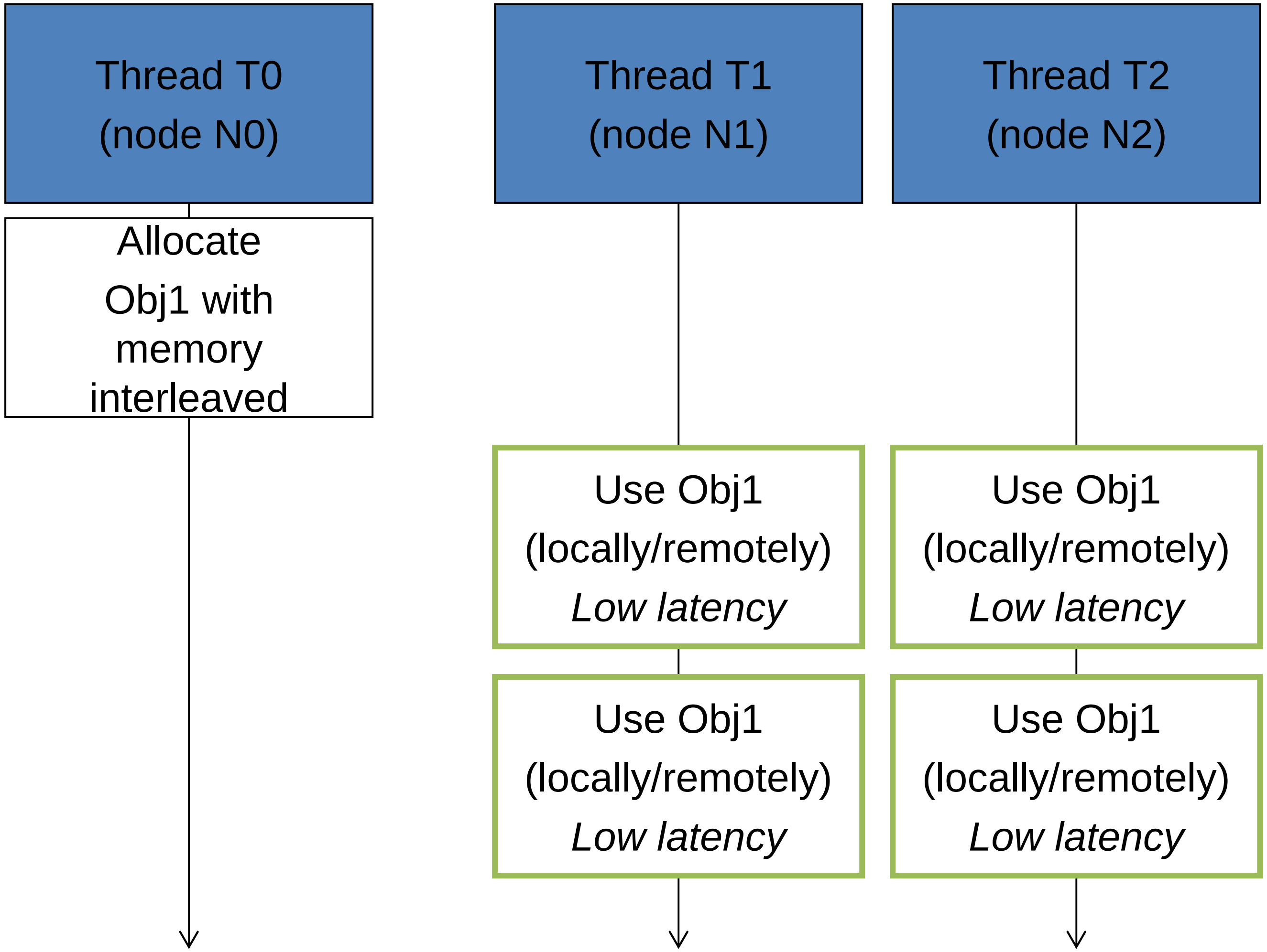
- 193 matrices
- 1 matrix induces 98% of the remote accesses
- This matrix is first written and then read by all threads

- We replicate the matrix (10 lines of code)
- Performance improvement: **42%**

We can detect that an object is simultaneously read and written by several threads with a high latency



And thus decide to interleave this object

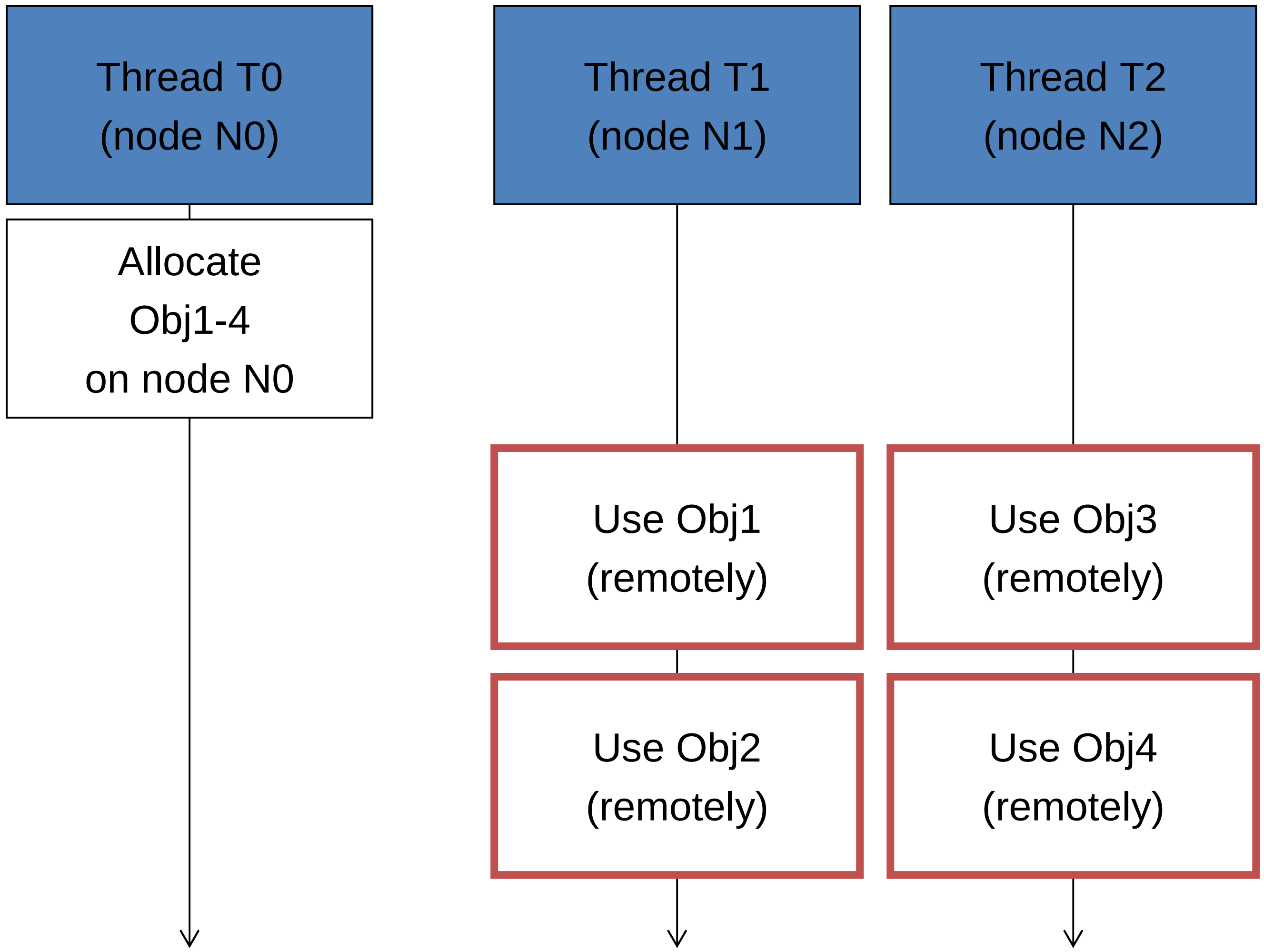


This is the pattern observed in Streamcluster

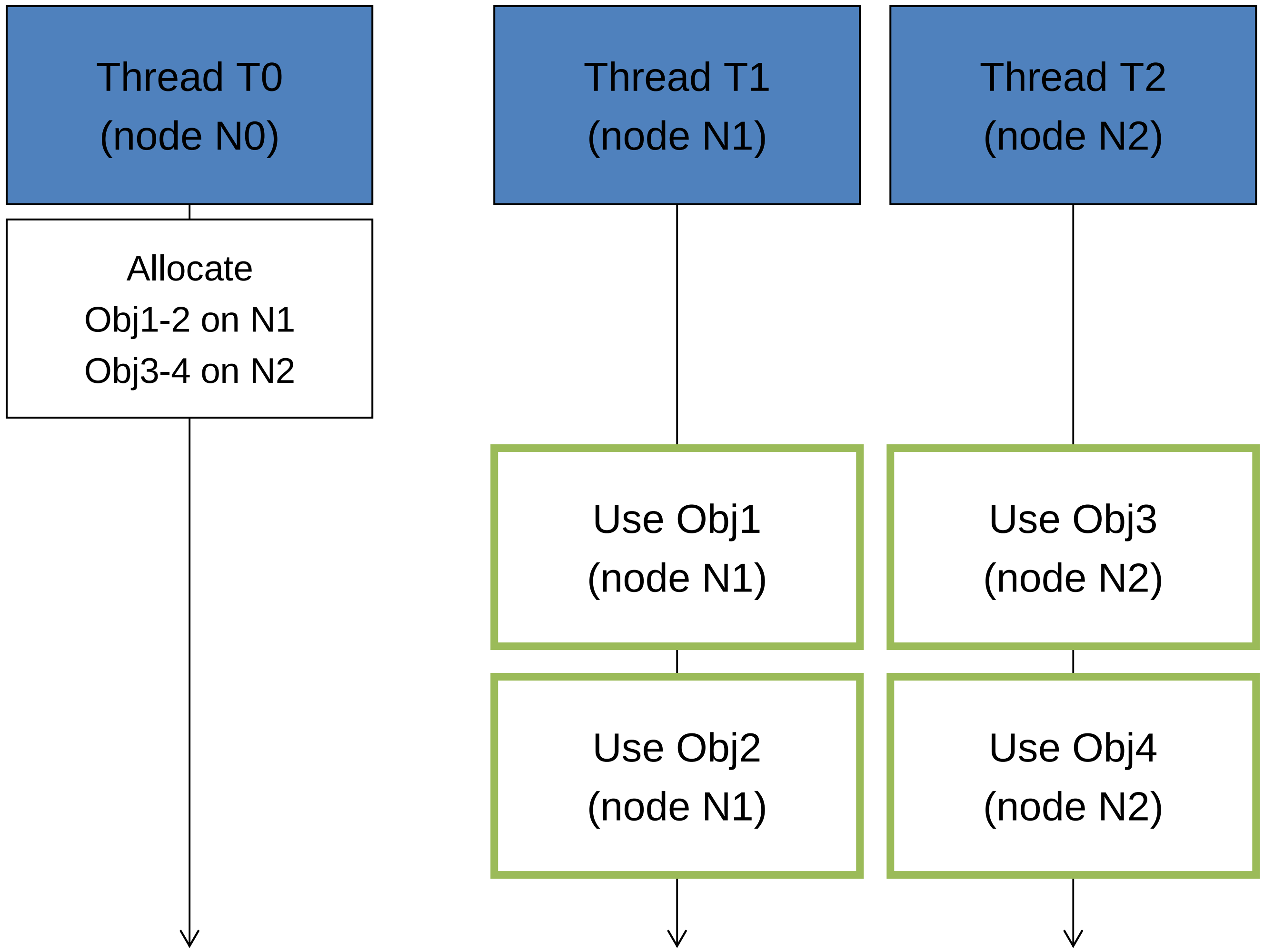
- 1000 objects allocated
- 1 represents 80% of remote memory accesses
- It is accessed read/write by all threads

- We interleave this object (1 line of code)
- Performance improvement: **161%**

We can detect that threads do not share objects



And thus decide to change the allocation policy



This is the pattern observed in Psearchy

- Remote accesses are done on private variables

- We forced local allocations (2 lines of code)
- Performance improvement: **8.2%**

Last use case: Apache

- Apache is a popular Web server
- 75% of memory accesses are remote

Optimizing Apache with existing profilers

- Output of existing profilers:
  - Functions that perform remote memory accesses:

% of total remote memory accesses	Function
5,8	memcpy
2,8	_zend_alloc_int

  - No function stands out; the top functions are related to memory operations and are called from many different places, on many different variables
  - Some pages are accessed at different time intervals by different threads
  - Some pages are simultaneously accessed by multiple threads (Apache threads are not supposed to share memory → memory allocation problem?)

- Possible optimizations:
  - Page migration (5% performance decrease)
  - Local memory allocation (same performance)
  - Thread pinning (2% improvement)

Optimizing Apache with MemProf

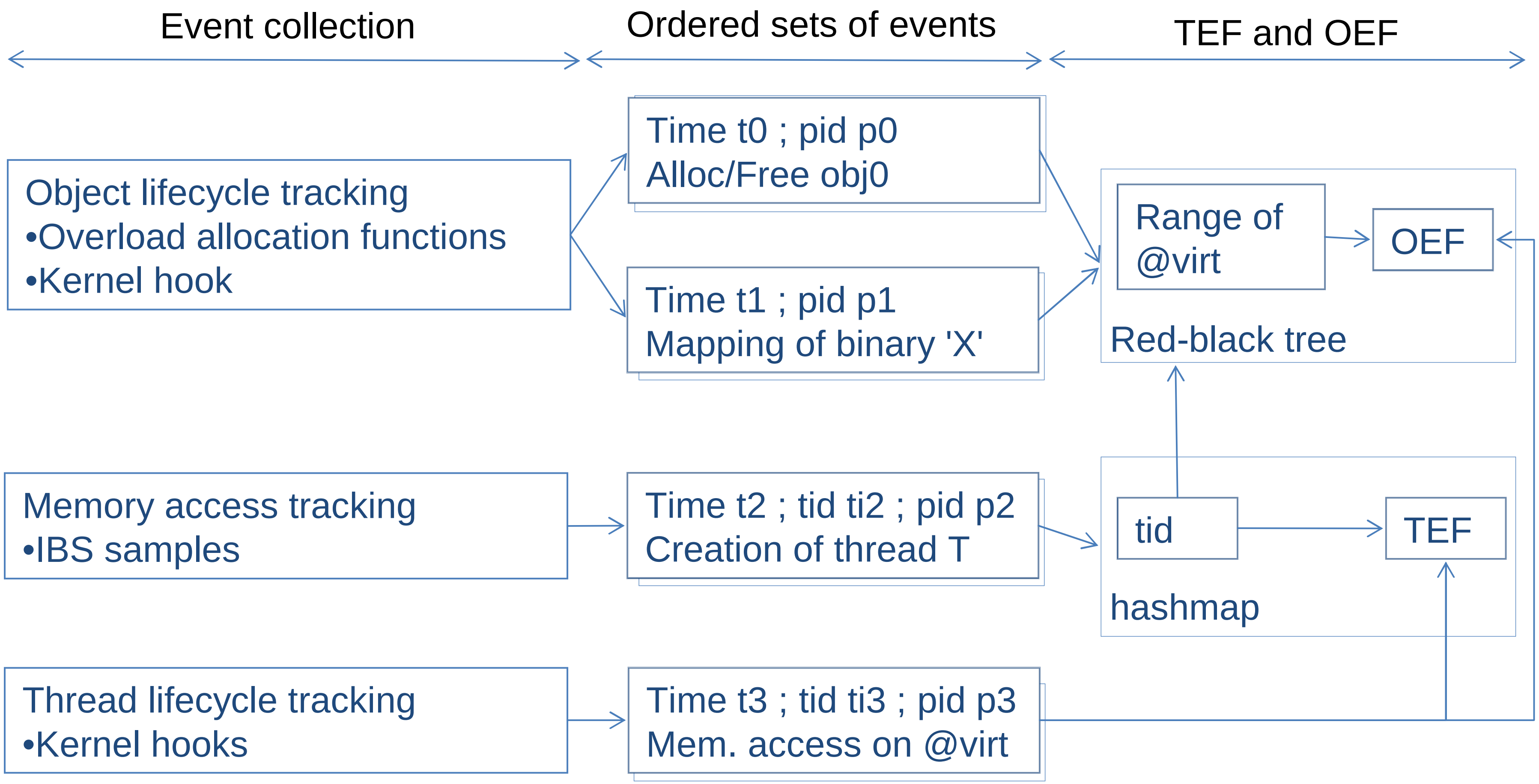
- Output of MemProf:
  - Most remote memory accesses are performed on 2 types of objects:
    - *apr_pools* variables
    - Pointer relocation table
  - Each of these objects is shared between a set of threads belonging to the same process

- Possible optimization:
  - Pin all threads belonging to the same process on the same node (**20%** improvement)
    - 10 lines of code
    - Remote memory accesses: 10%

As a summary

- MemProf allows finding memory access patterns
- Knowing memory access patterns allows designing simple and efficient optimizations

A word on the implementation



Event collection
Ordered sets of events
TEF and OEF
Object lifecycle tracking
•Overload allocation functions
•Kernel hook
Time t0 ; pid p0
Alloc/Free obj0
Time t1 ; pid p1
Mapping of binary 'X'
Range of @virt
OEF
Red-black tree
Memory access tracking
•IBS samples
Time t2 ; tid ti2 ; pid p2
Creation of thread T
tid
TEF
hashmap
Thread lifecycle tracking
•Kernel hooks
Time t3 ; tid ti3 ; pid p3
Mem. access on @virt

MemProf – Online Profiling

- Memory access tracking
  - IBS samples
- Object lifecycle tracking
  - Overloading of allocation functions
  - Kernel hooks
- Threads lifecycle tracking
  - Kernel hooks

MemProf – Offline Analysis

- Sort samples by time
- Match memory addresses with objects
  - Leverages object lifecycle tracking
  - Leverages thread lifecycle tracking
- Create object-thread interaction flows
  - Leverages thread lifecycle tracking

Overhead

- 5% slowdown
- 2 sources of overhead:
  - IBS sampling collection: one interrupt every 20K cycles
  - Object lifecycle tracking

Conclusion

- Remote memory accesses are a major source of inefficiency
- Existing profilers do not pinpoint the causes of remote memory accesses
- We propose MemProf, a memory profiler that allows:
  – Finding which objects are accessed remotely
  – Understanding the memory access patterns to these objects
- Using MemProf, we profiled and optimized 4 applications on 3 machines
  – Optimizations are simple: less than 10 lines of code
  – Optimizations are efficient: up to 161% improvement

QUESTIONS?