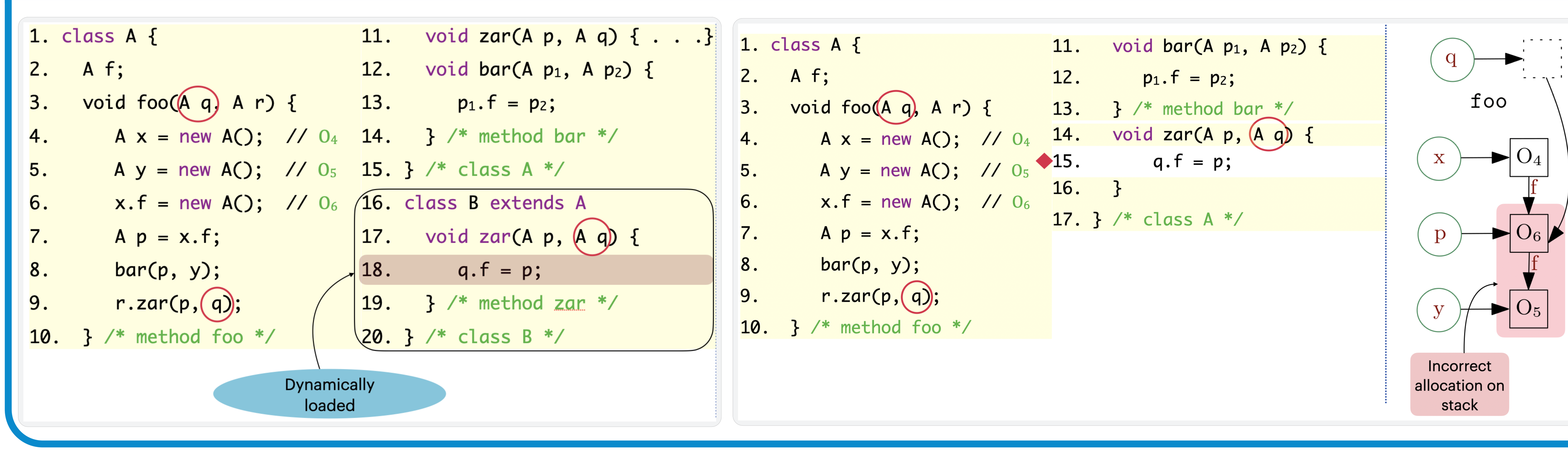
Program Analysis for Managed Runtimes in Presence of Dynamic Features*

Aditya Anand† and Manas Thakur†

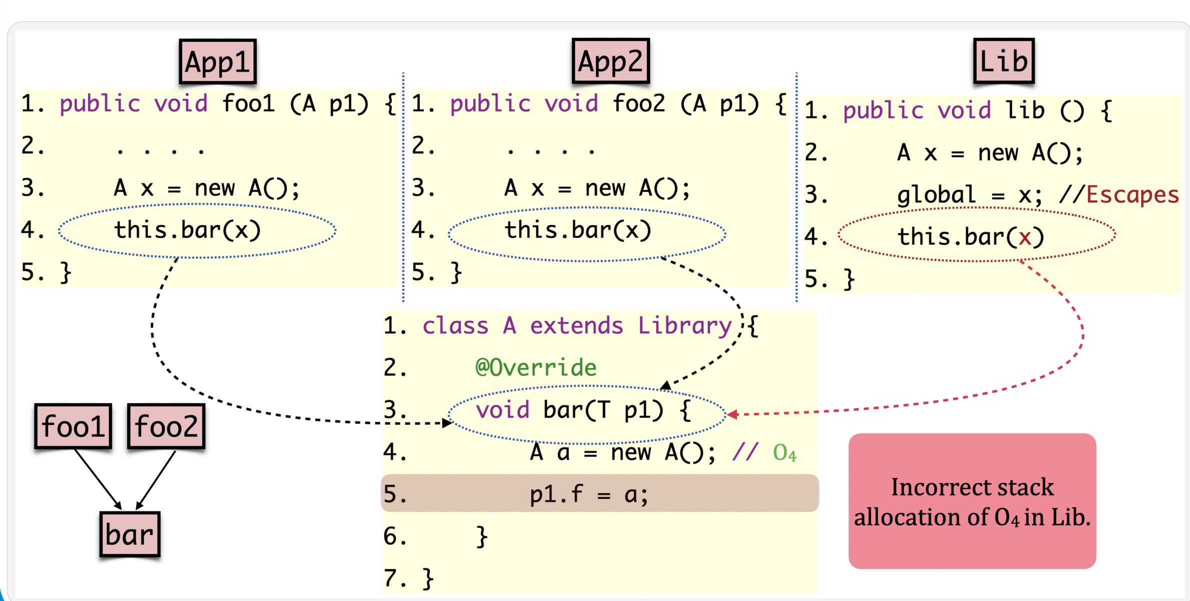
Objects Allocation

- Objects in Java are allocated on the heap.
- Access time from heap is high. Garbage collection is an overhead.
- Optimization – Method-local stack allocation of objects.
- JIT Time Analysis – Highly imprecise.
- Static Analysis – Affected by dynamic features like DCL, HCR, Callbacks etc.

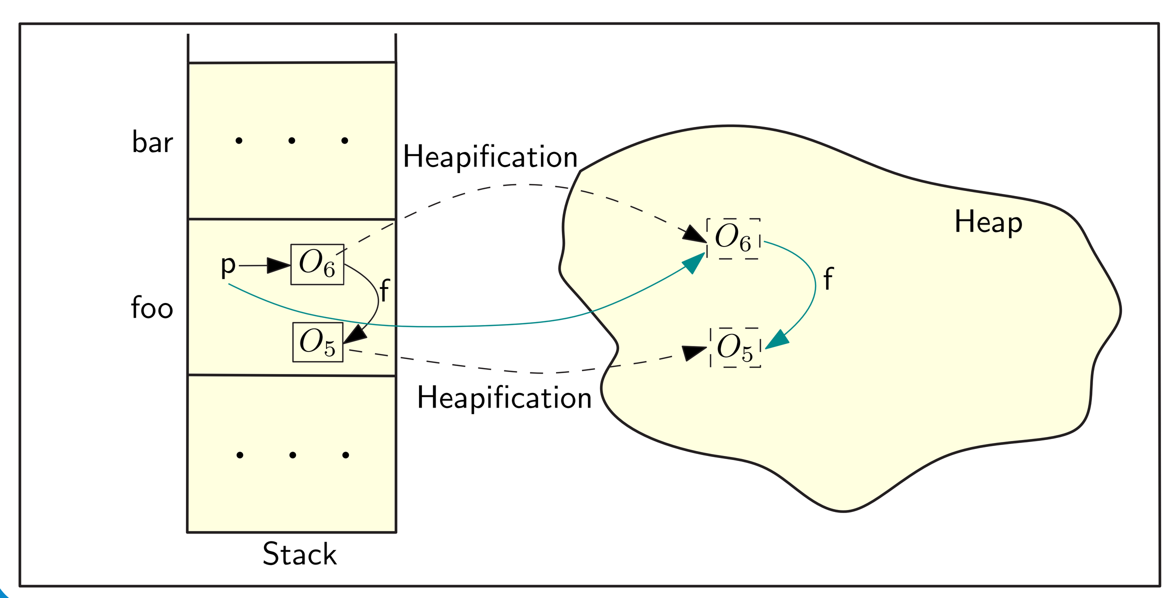
Dynamic Class Loading & HotCode Replacement



Callbacks



Heapification



Heapification Algo

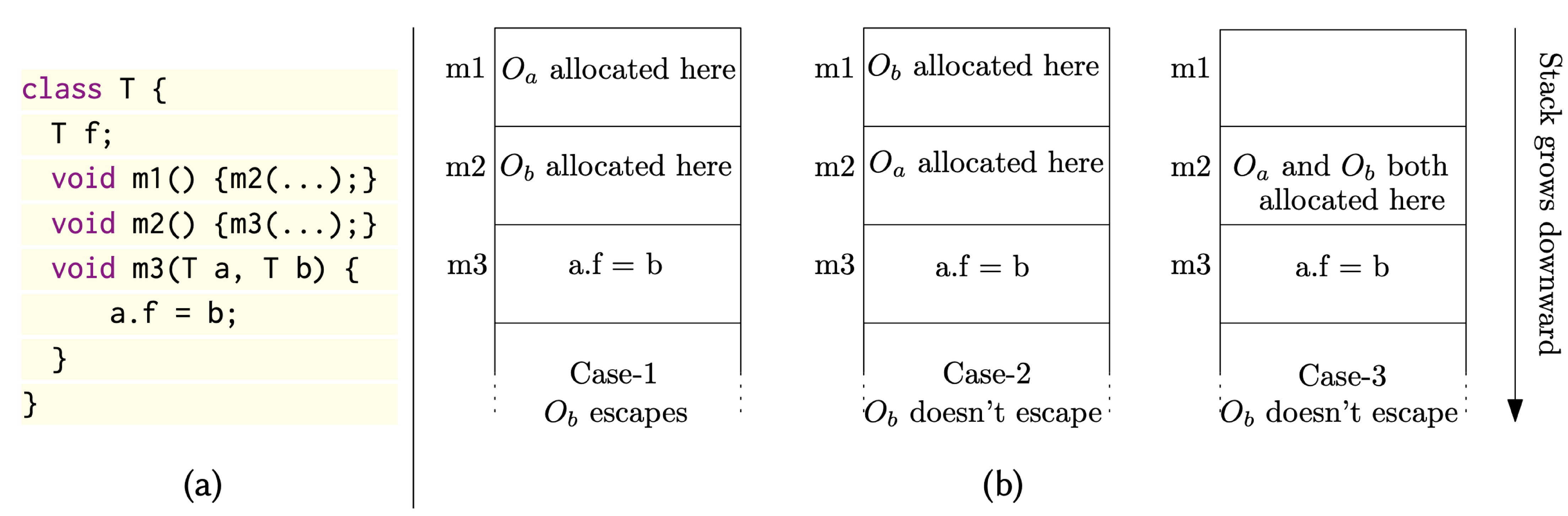
```
1 Procedure HeapificationCheckAtStore(lhs, rhs)
2   if rhs object is outside stack bounds then
3     No heapification required.
4   else
      /* The rhs object is present on the stack */
5     if lhs object is outside stack bounds then
6       Heapify starting from the rhs object.
7     else
        /* Both lhs and rhs objects are on the stack */
8       if lhs object has longer life time than rhs object then
9         Heapify the rhs object.
```

a.f = b; (lhs_obj, rhs_obj)

Stack Ordering

- Traversing stack-frames for parameters while checking for heapification is costly.
- Establish object ordering to enable address comparison for heapification checks, minimizing the need for frequent stack walks.
- Statically create a partial order of stack allocatable objects and use the stack-ordering in the VM to re-order the list of stack allocated objects.
- Reduces cost of heapification checks for the cases where objects doesn't escape.

Improving efficiency by Stack Ordering



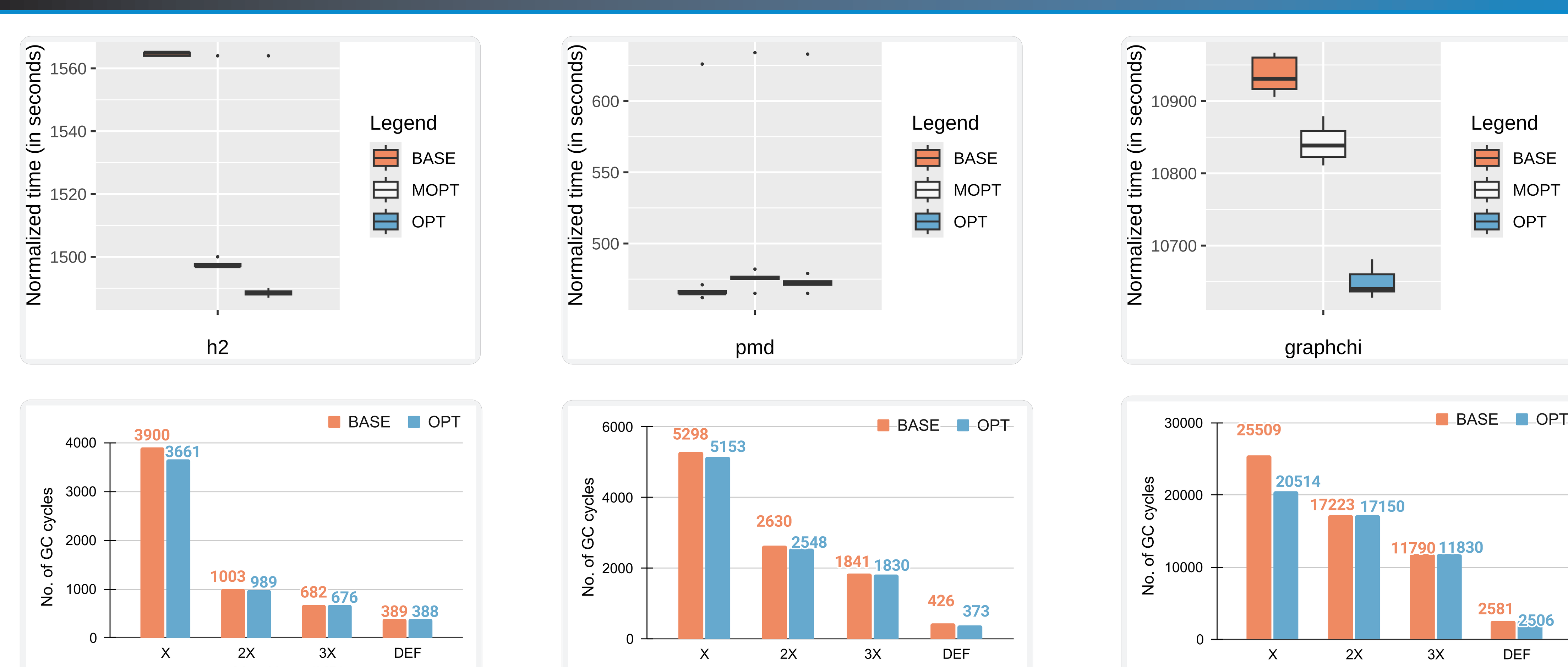
Stack Allocation

h2			
BASE		OPT	
Stack-Objects	Stack-Bytes	Stack-Objects	Stack-Bytes
29M	0.5 GB	452M	10.8 GB

pmd			
BASE		OPT	
Stack-Objects	Stack-Bytes	Stack-Objects	Stack-Bytes
52M	1.3 GB	105M	2.4 GB

graphchi			
BASE		OPT	
Stack-Objects	Stack-Bytes	Stack-Objects	Stack-Bytes
0.0M	0 GB	506M	9.1 GB

Performance Improvement



Conclusion

- Proposed an idea to have dynamic checks for potential incorrect stack allocations, along with repairing memory layout by heapifying escaping objects and correcting their references.
- An efficient approach for performing heapification checks by ordering objects on the stack.

Future Work

- **Future Work**: Perform more aggressive stack-allocation & enable further optimizations in the JIT compilers.

Research and Innovation Symposium in Computing (RISC 2025), IIT Bombay

† Author addresses: {adityaanand, manas}@cse.iitb.ac.in. Department of CSE, IIT Bombay, Mumbai.

* Anand et al. "Optimistic Stack Allocation and Dynamic Heapification for Managed Runtimes." In Proceedings of the ACM on Programming Languages **(PLDI)**, Copenhagen, Denmark, June 24-28, 2024. URL: https://doi.org/10.1145/3656389