STMGC-C7

Armin Rigo

www.pypy.org

Remi Meier

Department of Computer Science
ETH Zürich

Fast Software Transactional Memory for Dynamic Languages

Current Situation

- Dynamic languages popular (Python, Ruby, PHP, JavaScript)
- Parallelization is a problem
- **GIL**
  - Atomicity & isolation for bytecode instructions
  - No real parallelism
- **Multi-process**
  - Exchanging data explicitly
  - Only suitable for some kinds of applications

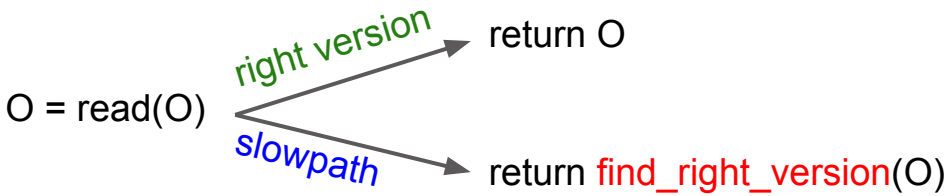
Transactional Memory: Our Goals

- **Goal 1:** A transaction executes N bytecodes
  - Parallelization for existing multithreaded programs
  - The **whole** program runs in transactions
    → good performance is essential

- **Goal 2:** Improved multithreading model
  - Better model for the programmer
  - Transaction boundaries controlled by the program
  - Much **longer** transactions
    → HTM is far too limited for now

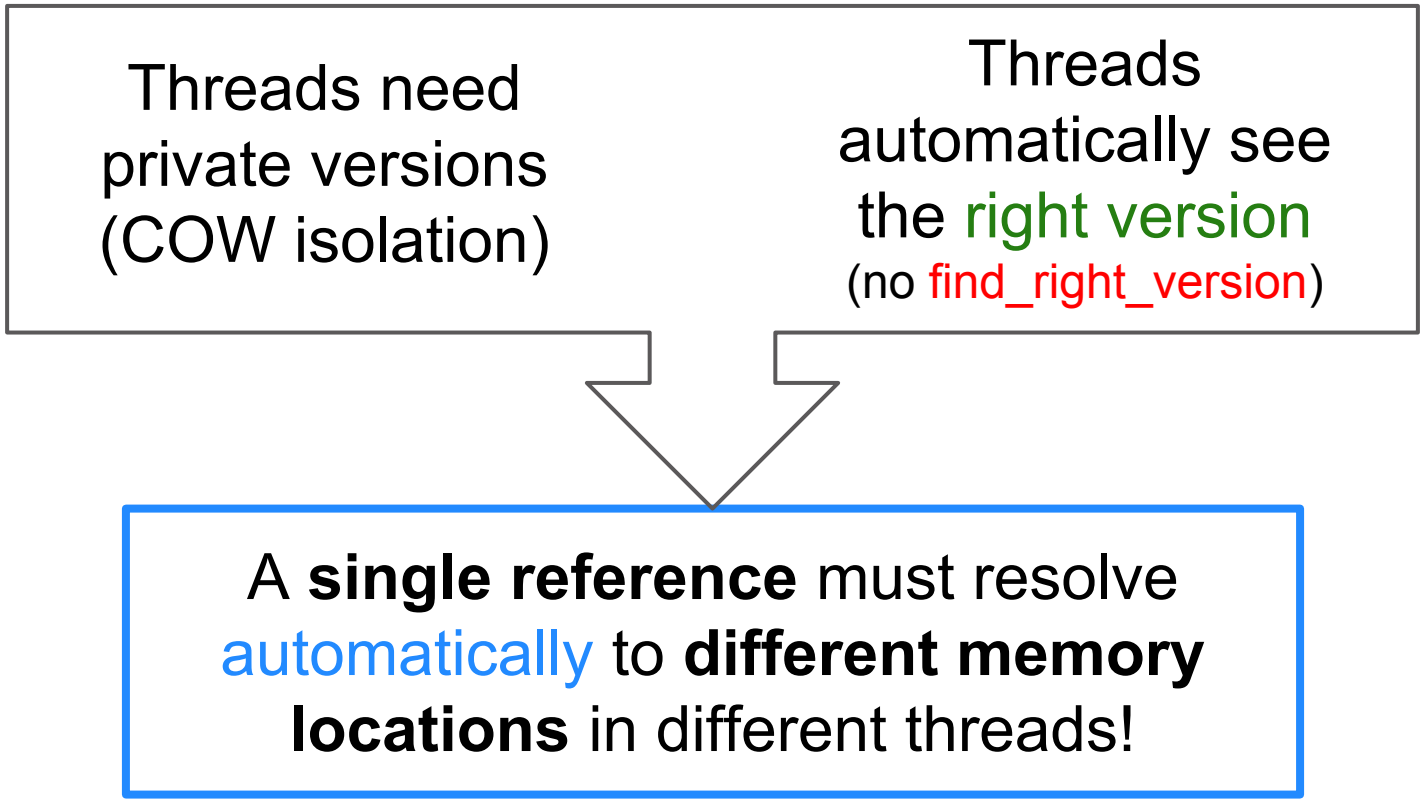
Background: STM Overhead

- Often 100% - 1’000%
- Major source of **S**TM overhead is **barriers**
  - All over the place
  - Isolation (Copy-On-Write, Locking, …)
  - Validation (conflict detection)
  - **Reference resolution** (for COW):



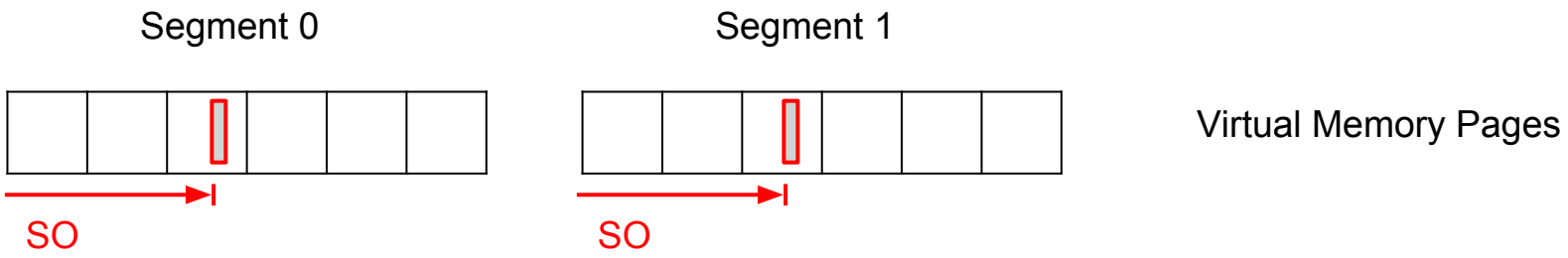
C7: A nice trick

How to avoid reference resolution in barriers if using Copy-On-Write?



C7: Segmentation

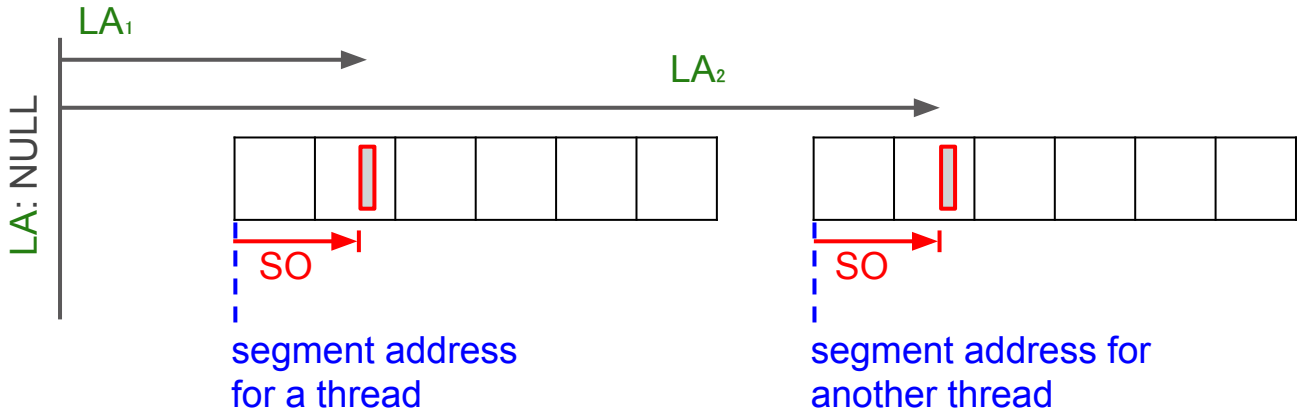
- Partition virtual memory into segments
- 1 segment per thread
- Each segment is a **copy**
  → same contents in all segments
- **All copies** of an object are at the **same** *segment offset* (SO) in each segment



C7: Segmentation

- Use SO as object reference
- Need to translate to *linear address* (LA):

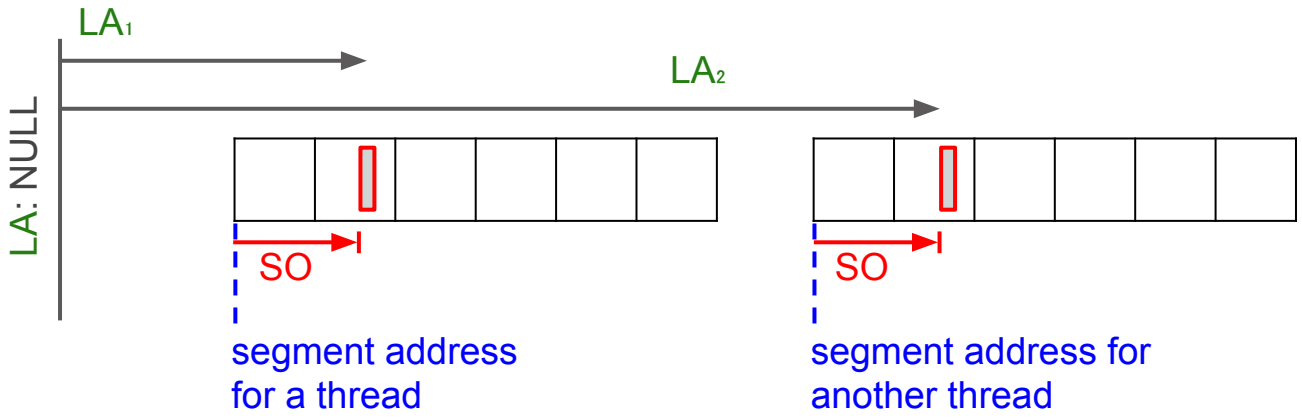
  LA = segment address + SO
- Hardware supported ⇒ on every SO access
- SO translated to different LAs in different threads



C7: Segment Offset

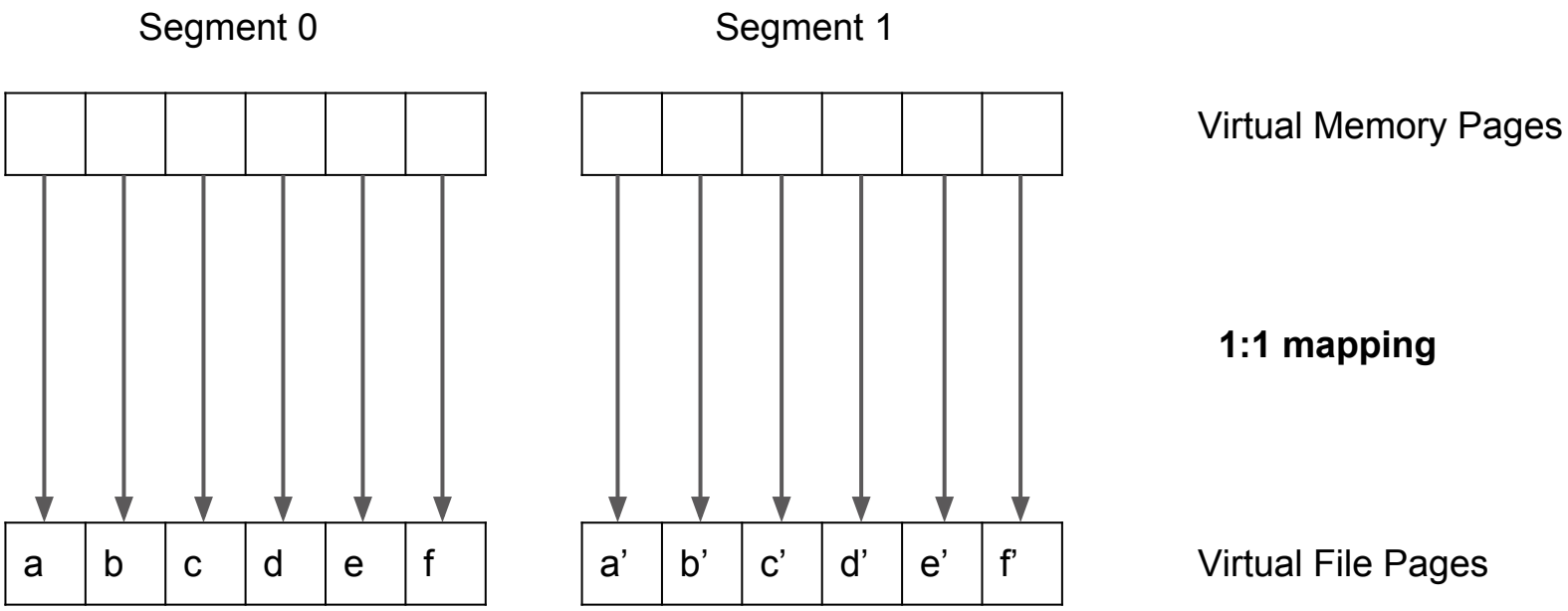
- One SO → multiple LAs ✓
- **Extremely inefficient:**
  a. N-times the memory
  b. 1 allocation ⇒ N allocations
  c. 1 write ⇒ N writes

How to share memory?



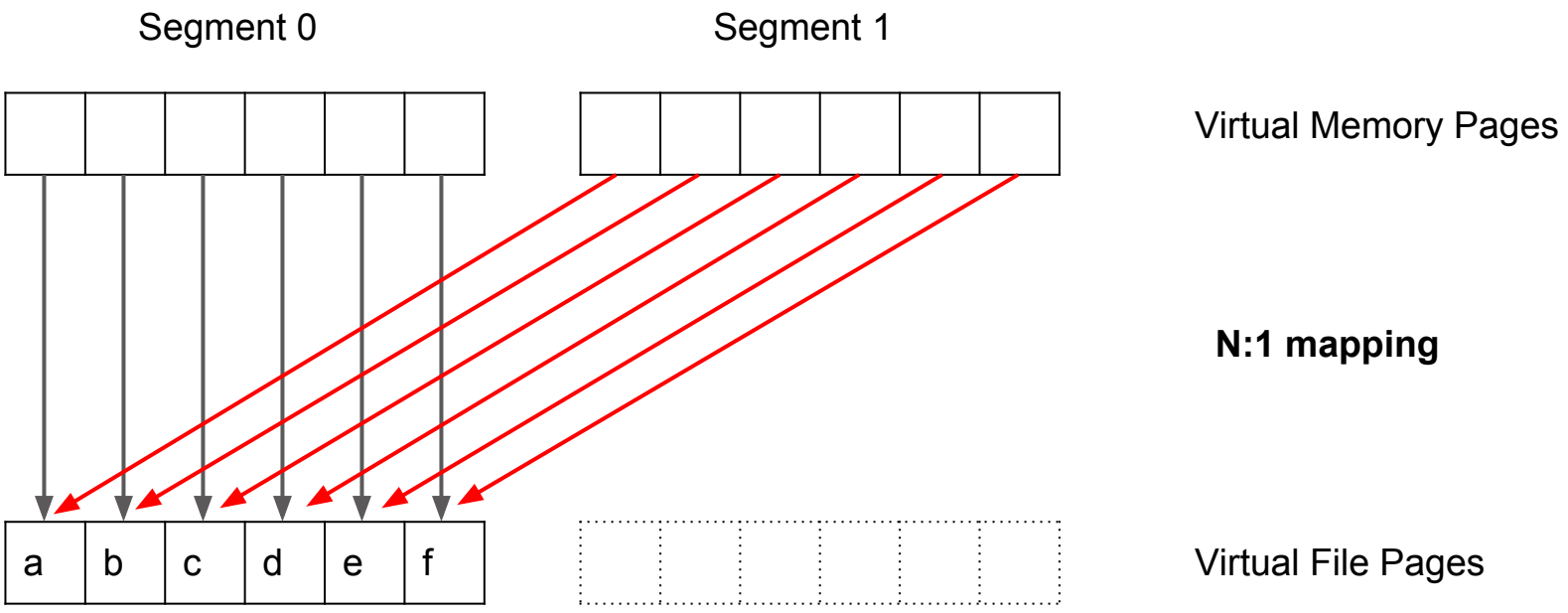
C7: Page Sharing

Partition virtual memory into segments: each segment is backed by different memory



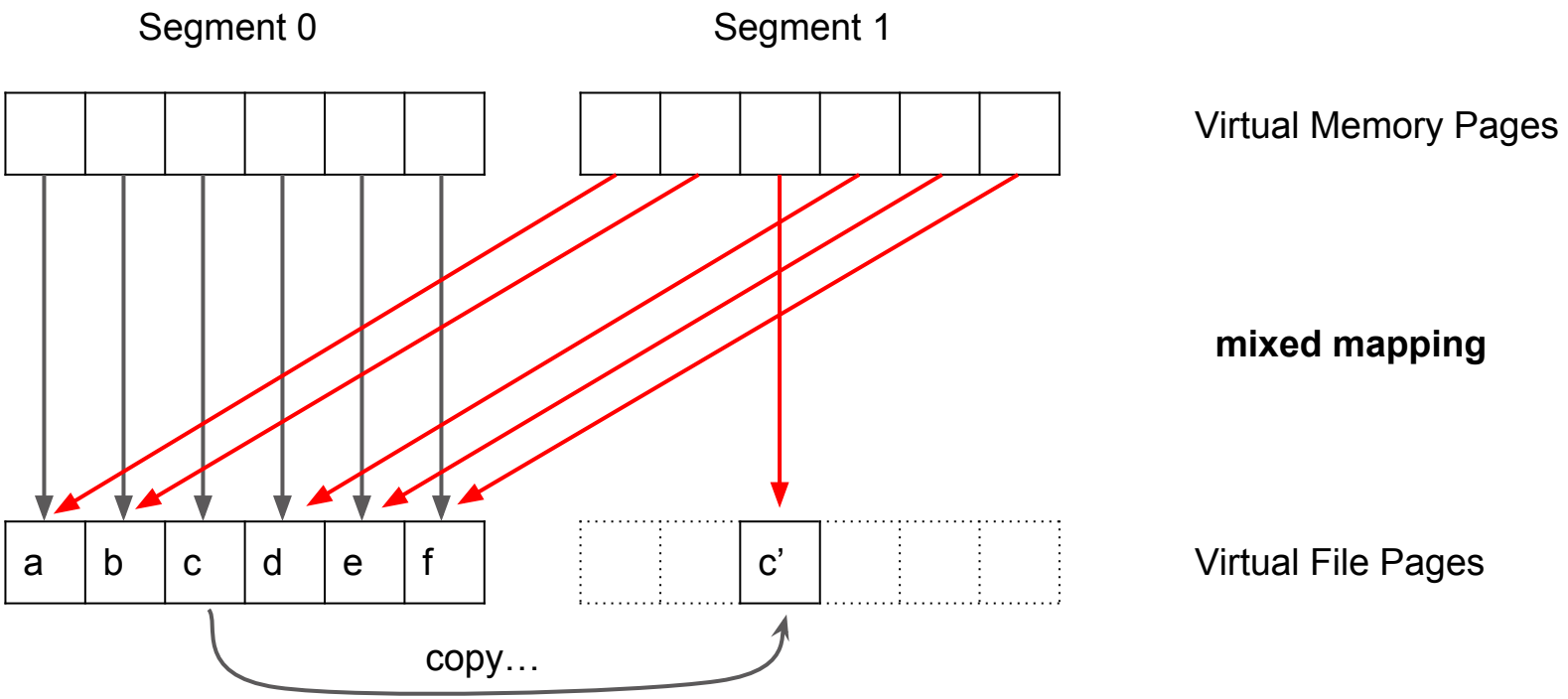
C7: Page Sharing

Remap segment 1: Both segments share the same memory



C7: Page Sharing

We can unshare / privatize pages



C7: Read Barrier

- Address translation on each object access:
  a. segment address + SO → LA
  b. LA → memory location (private / shared)
- SO **never changes**
- SO always translates to the right version

  → no "right version" check

  → no find_right_version()
- Read barrier only has to **mark** it as read

C7: Write Barrier

- Objects created in the same transaction are ignored
- Copy-On-Write
  - Privatize all pages of the object
  - Only on first access to a page
  - Re-share pages at major collections

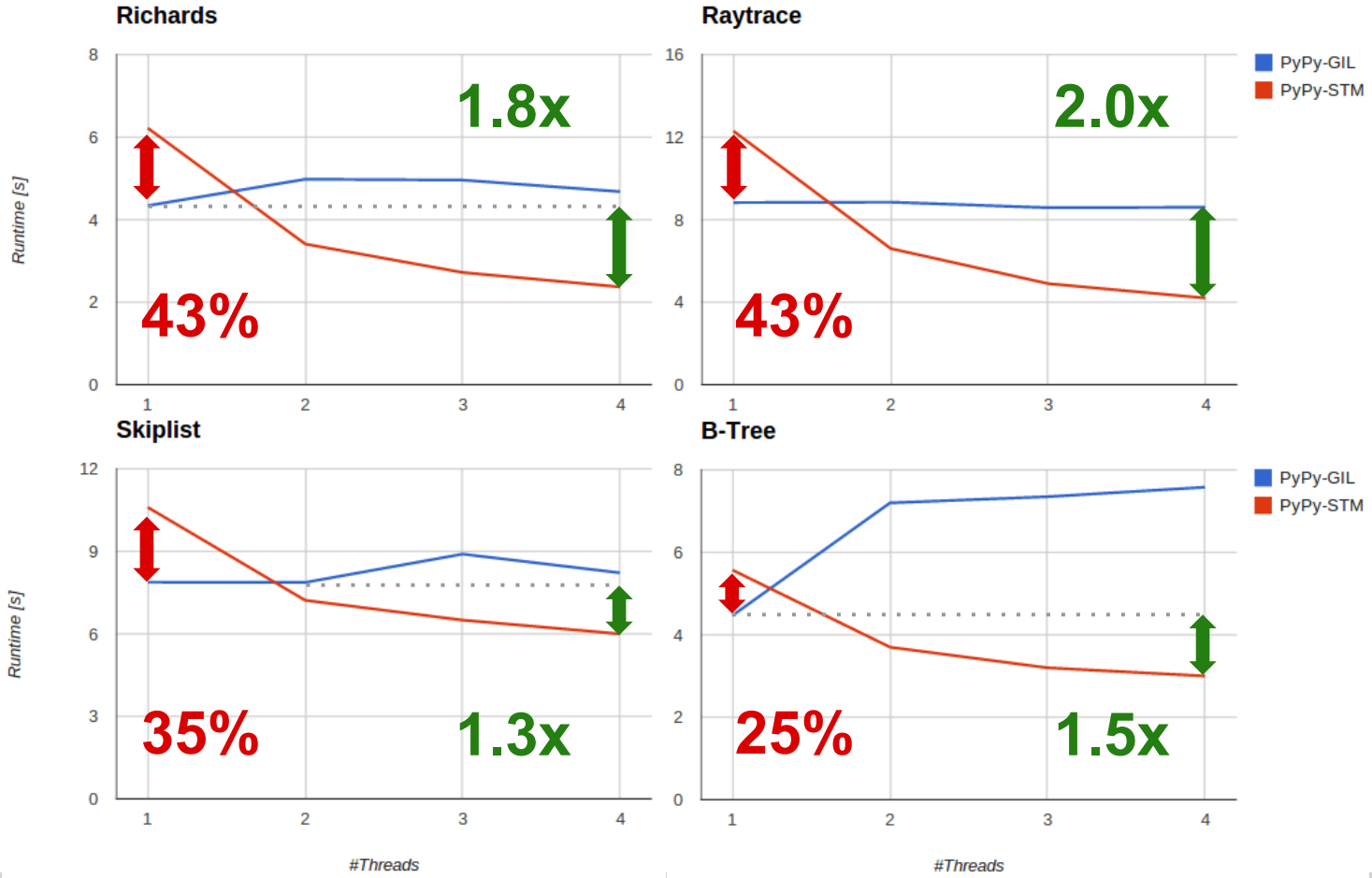
- Low-cost, page-level COW
- Object-level conflict detection

C7: Total Costs

- Extremely cheap barriers
- Integrated with garbage collection
  - Most new objects die quickly and don’t need barriers
  - One write barrier for both STM and GC
- Commit-time costs
  - Detecting read-write conflicts
  - Copy modifications in private pages to other segments

Results

- Python interpreter (GIL version vs. STM version)



Summary

- Total STM overhead **< 50%**
- Large address space needed (64bit)
- Optimized for low #CPUs
- Optimized for dynamic language VMs
- **S**TM, not **H**TM

→ flexibility and long transactions