slimfast® Reducing Metadata Redundancy in Race Detection

Yuanfeng Peng, Christian DeLozier, Ariel Eizenberg, William Mansky, Joseph Devietti

Penn
University of Pennsylvania

What is a data race?

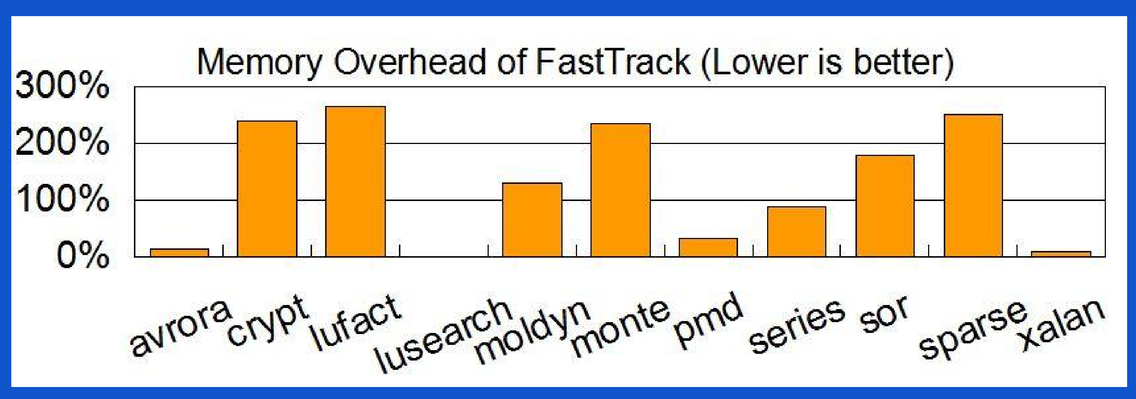
2 threads access a shared location without synchronization, and at least one access is a write.

Precondition: x==0 && y==0	
Thread 1	Thread 2
x = 1;	y=x;
Postcondition: x==1 && y==?	

Consequences:
- complicated program semantics
  (e.g. non-sequentially-consistent execution)
- undefined semantics for C/C++ programs

Race detectors need to go on a diet!

Spatial overhead for precise race detectors is HIGH
Each shared location needs >= 24 bytes of metadata



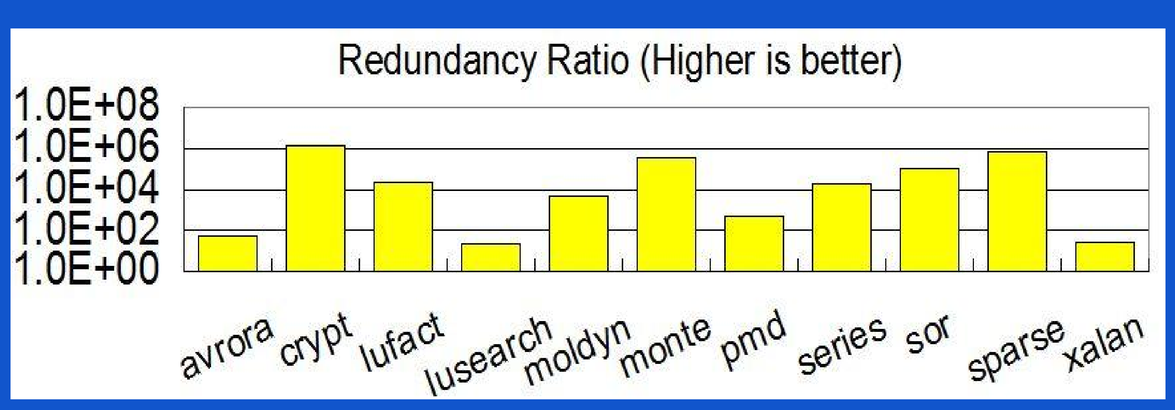
Race detection metadata is highly redundant

Redundancy arises when locations are accessed without intervening synchronization

Metadata objects shown in the orange box have identical values!

thread operation	[read, write] metadata
acquire L	
write X	[- , t1@1]
write Y	[- , t1@1]
write Z	[- , t1@1]
release L	

How much redundancy is there in real programs?

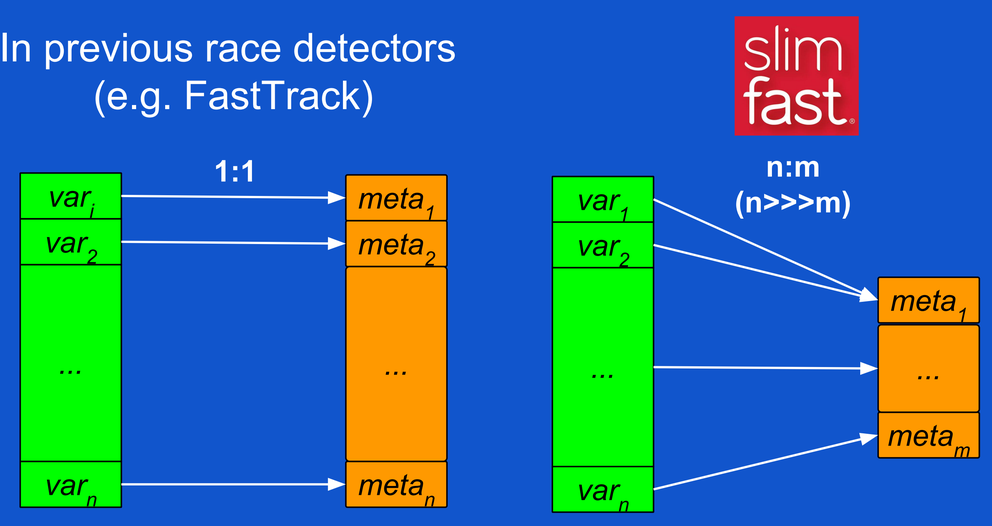


To remove such redundancy, we developed slimfast®

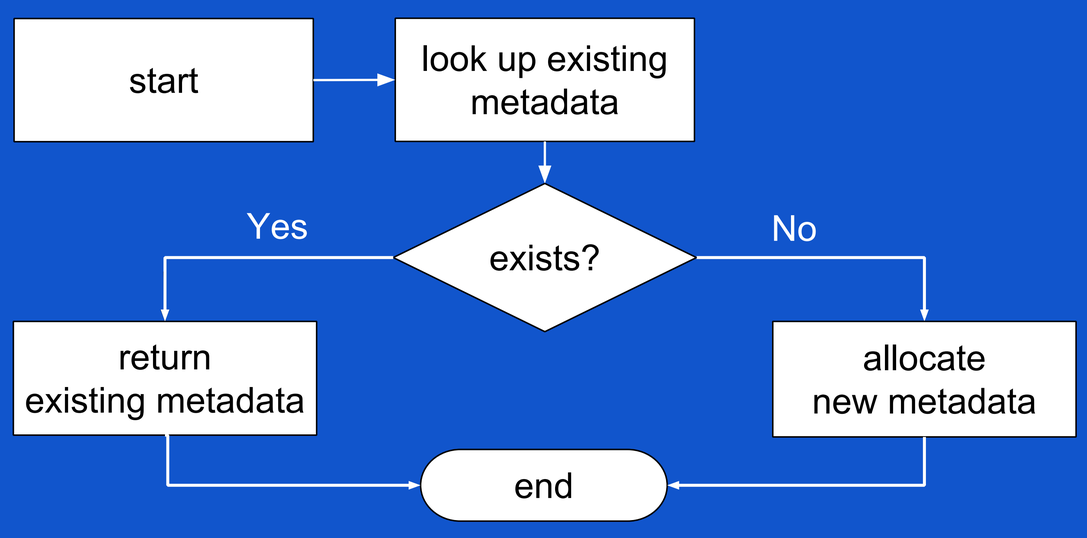
How does SlimFast work?

How are shared locations mapped to their metadata?

In previous race detectors (e.g. FastTrack)

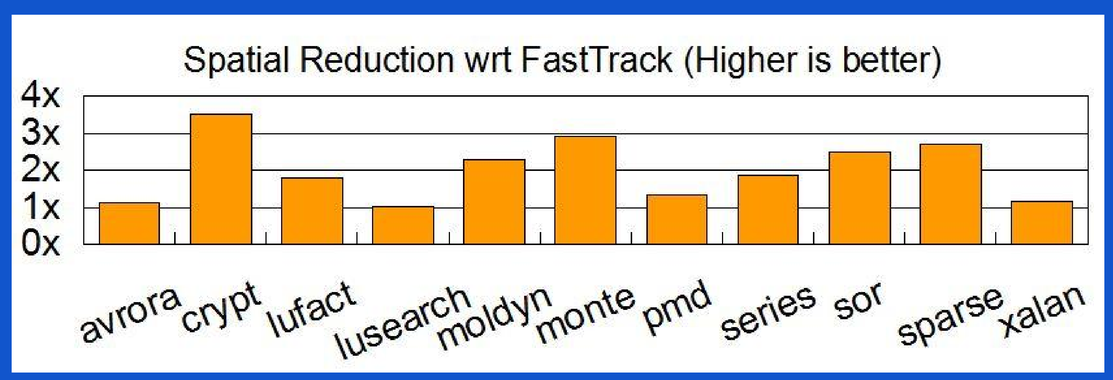


In SlimFast, metadata instances are immutable
Metadata is updated as follows:



Results

SlimFast consumes less memory than FastTrack on all benchmarks.
Average reduction: 2.02x Max reduction : 3.51x (crypt)



Overall, SlimFast runs faster than FastTrack.
Average speedup: 1.2x Max speedup: 2.3x (lufact)