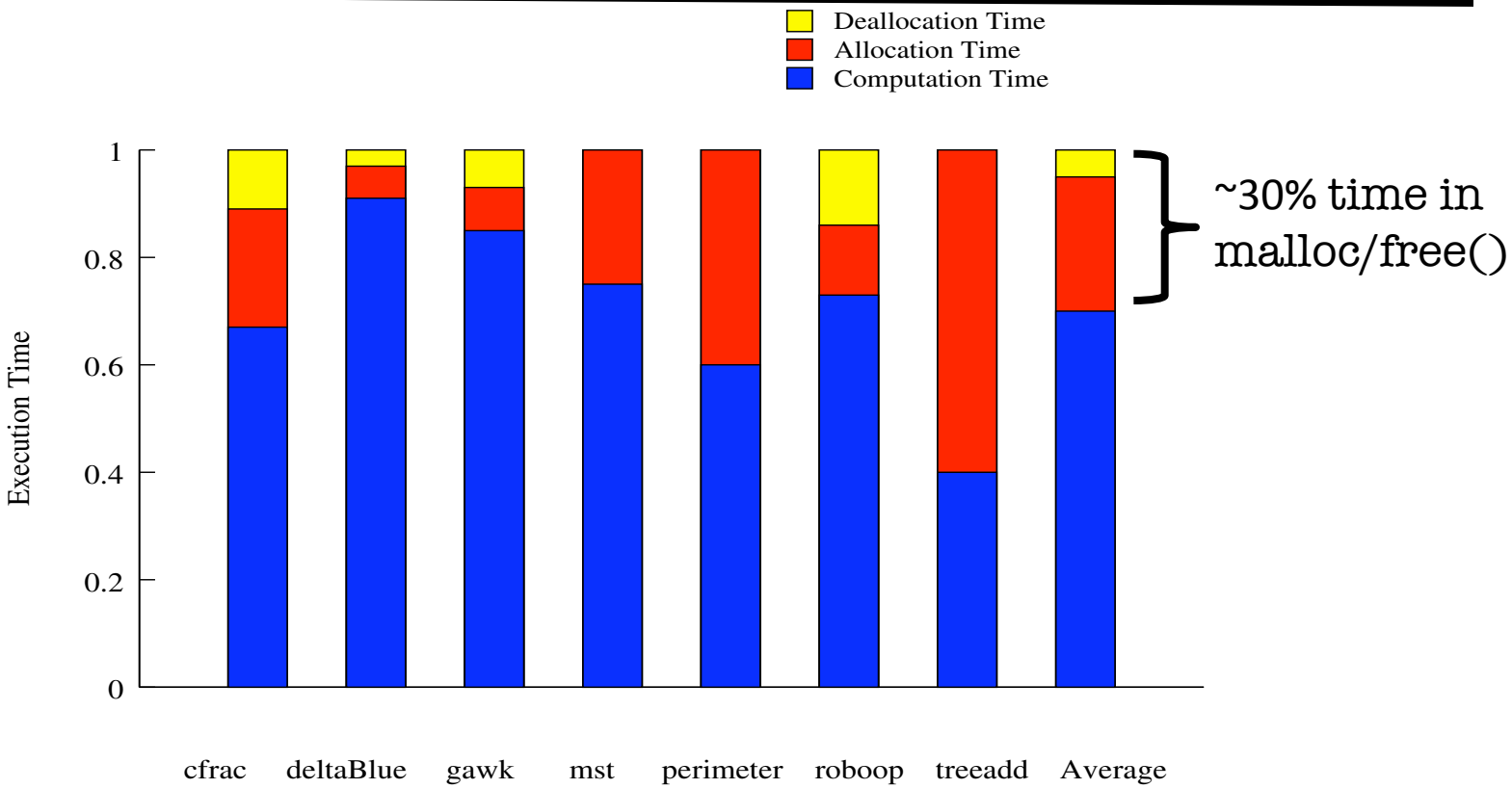
MMT: Exploiting Fine-Grained Parallelism in Dynamic Memory Management

Devesh Tiwari, Sanghoon Lee, James Tuck and Yan Solihin
ARPERS Research Group
Electrical and Computer Engineering
NC State University

NC STATE UNIVERSITY

April 19-23 2010
IPDPS 2010 Atlanta Georgia USA

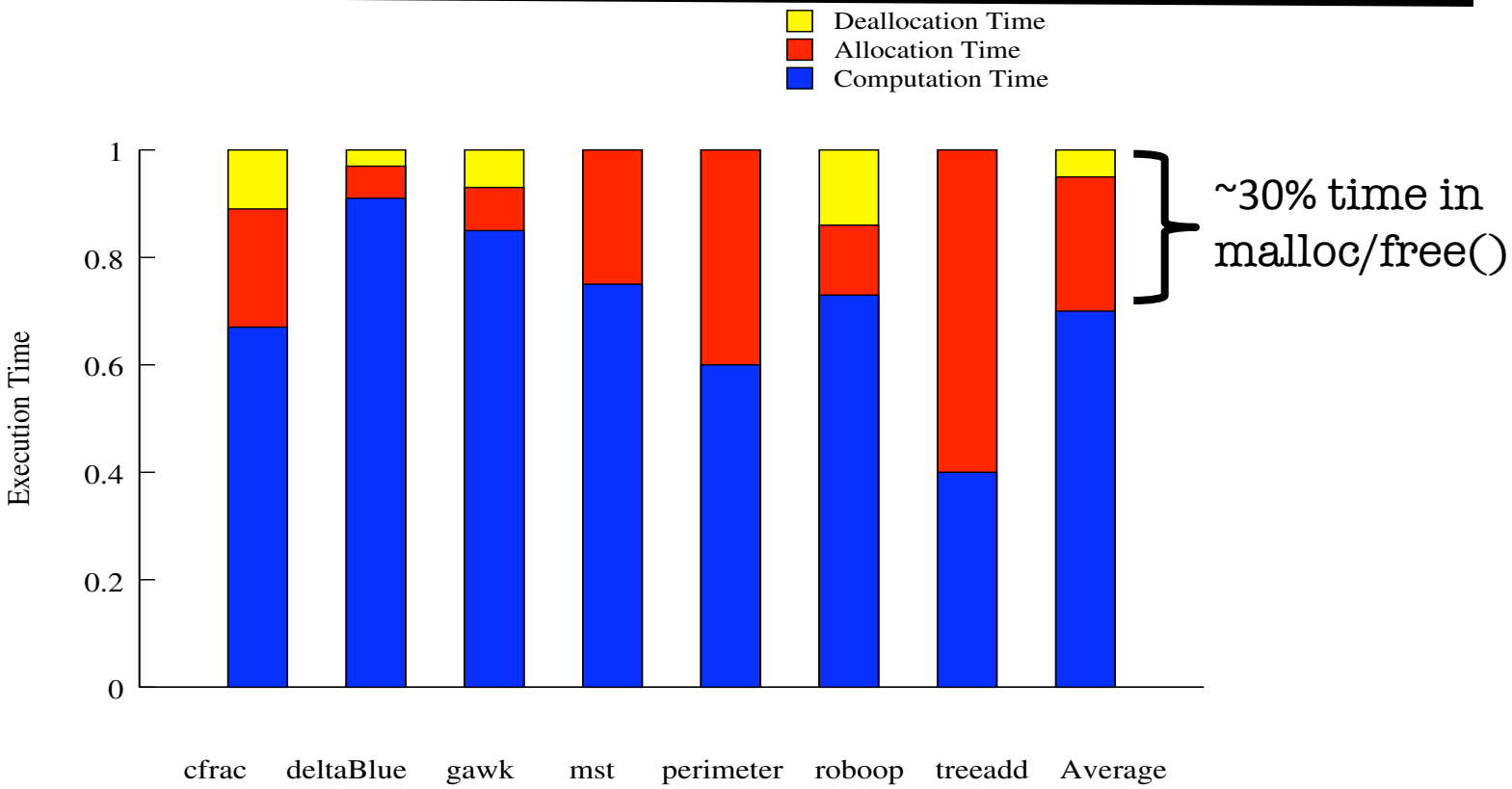
Motivation



Trends in Heap Management

- Ubiquitous but **expensive**

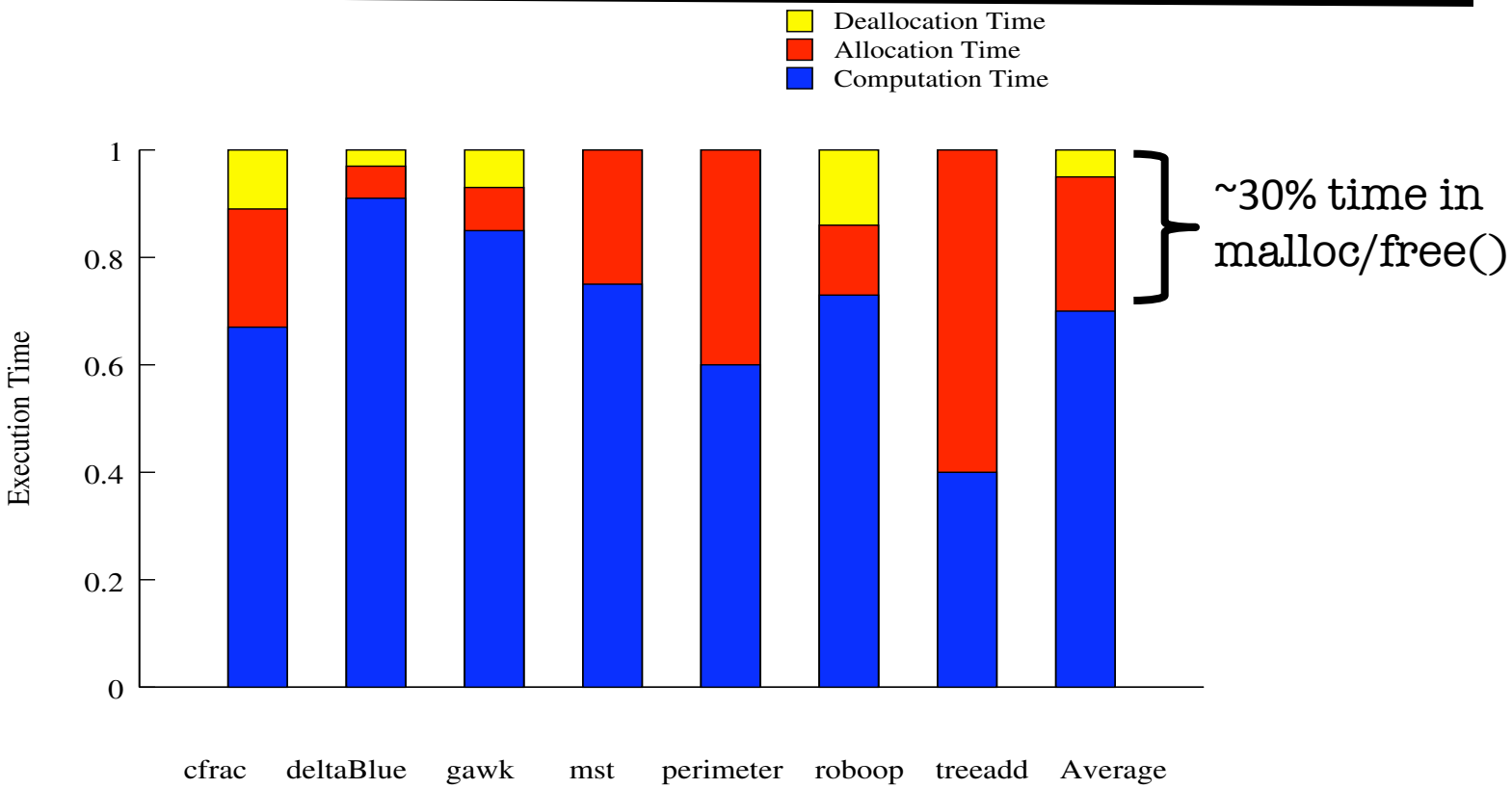
Motivation



Trends in Heap Management

- Ubiquitous but **expensive**
- If security checks enabled: overheads increase (21% on average)

Motivation



Trends in Heap Management

- Ubiquitous but **expensive**
- If security checks enabled: overheads increase (21% on average)
- Increasingly used due to object oriented languages

Motivation

Trends in Hardware

- Chip makers turned to multicore processors
- Multicores provide opportunity to run multiple threads in parallel

Motivation

Trends in Hardware

- Chip makers turned to multicore processors
- Multicores provide opportunity to run multiple threads in parallel

How can multi core parallelism be exploited to speed up heap management for sequential applications?

Outline

- Motivation
- Memory Management Thread : MMT Approach
- MMT Design and Implementation
- Evaluation
- Conclusion

Outline

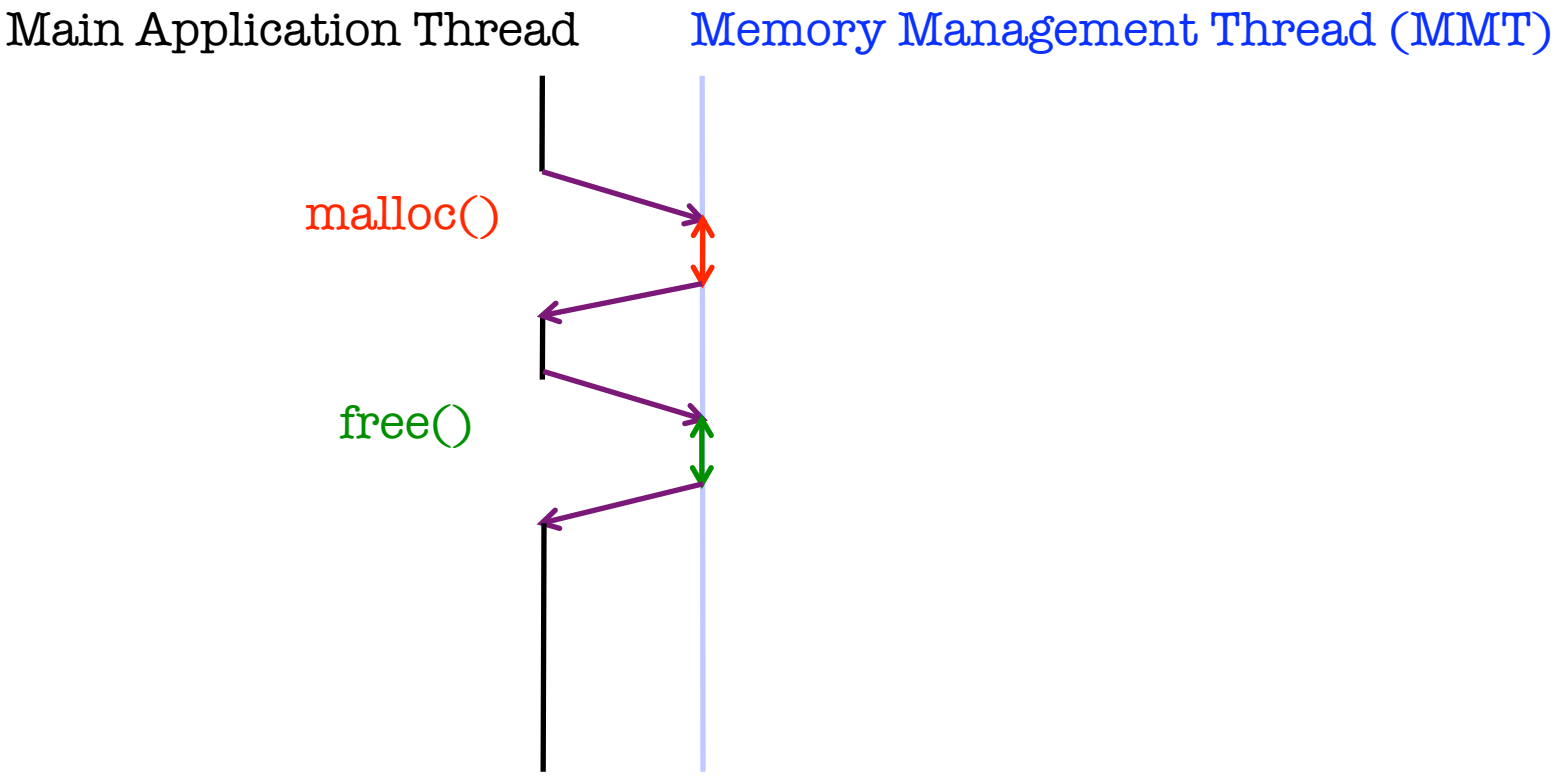
- Motivation
- Memory Management Thread : MMT Approach
  - The Idea
  - Challenges and Contributions
- MMT Design and Implementation
- Evaluation
- Conclusion

MMT: The Idea

A dedicated thread that executes all memory management functions on behalf of the main application

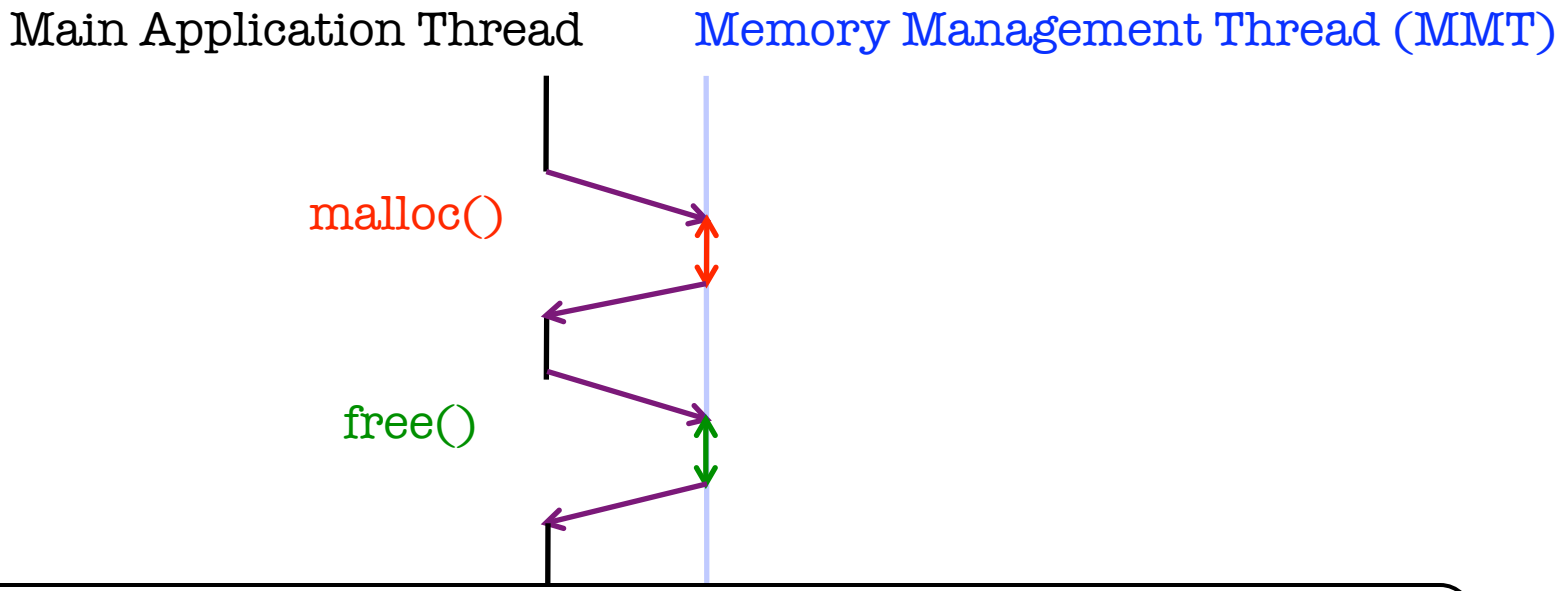
MMT: The Idea

A dedicated thread that executes all memory management functions on behalf of the main application



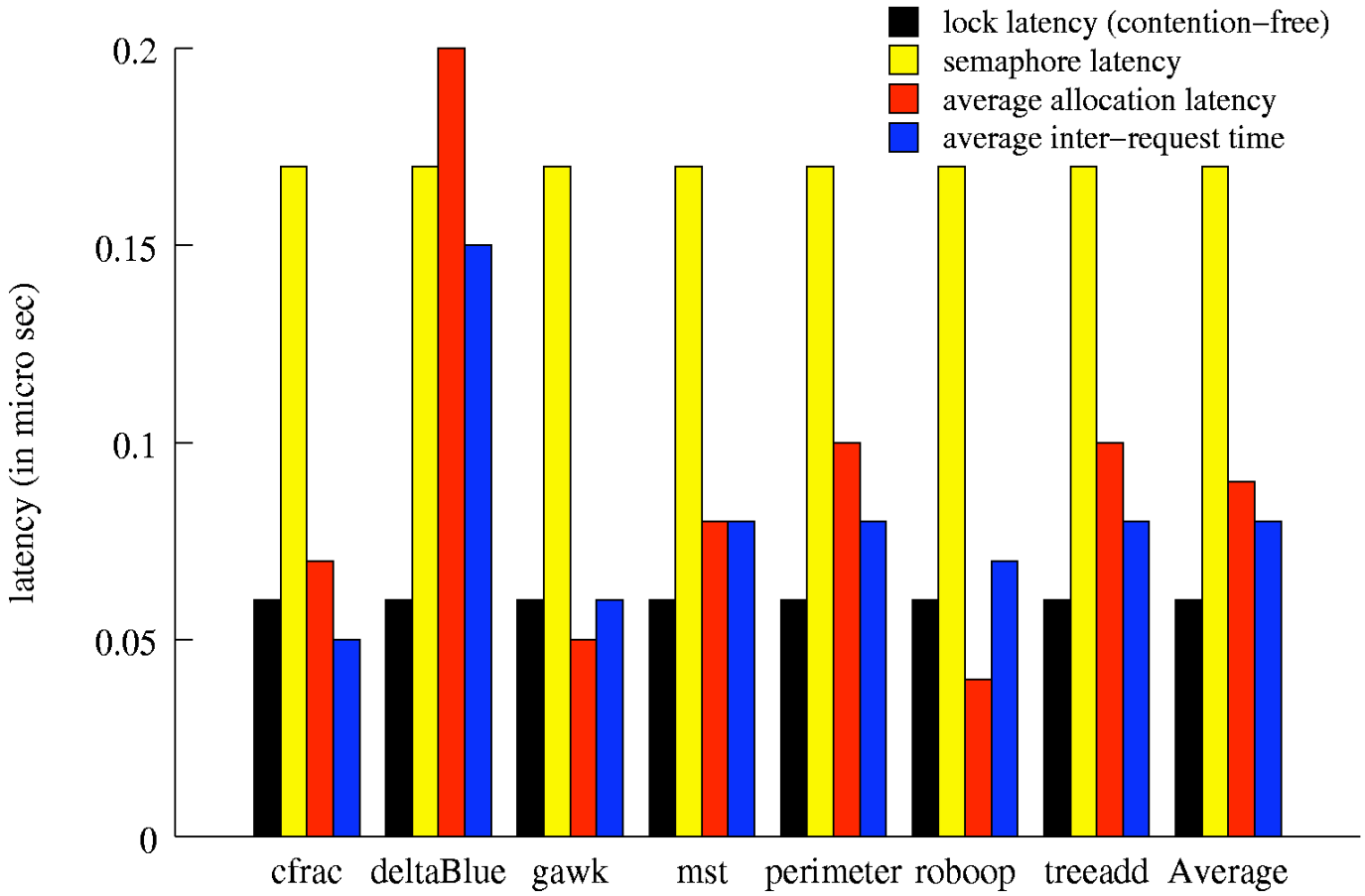
MMT: The Idea

A dedicated thread that executes all memory management functions on behalf of the main application



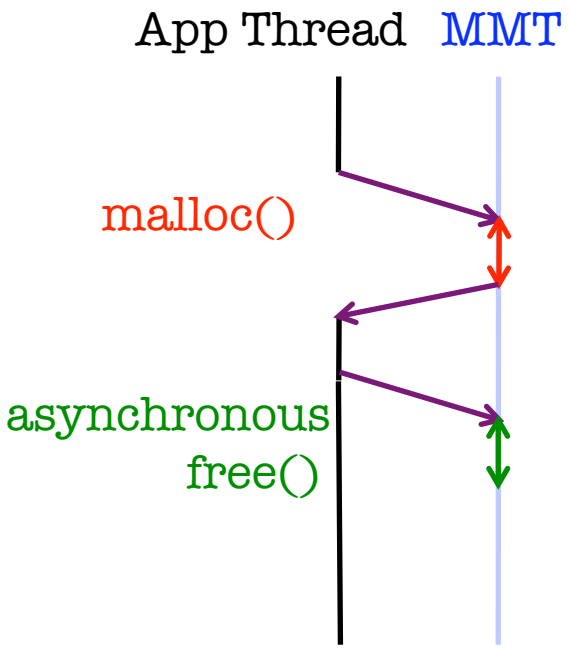
Need to synchronize and communicate with MMT

A Key Challenge

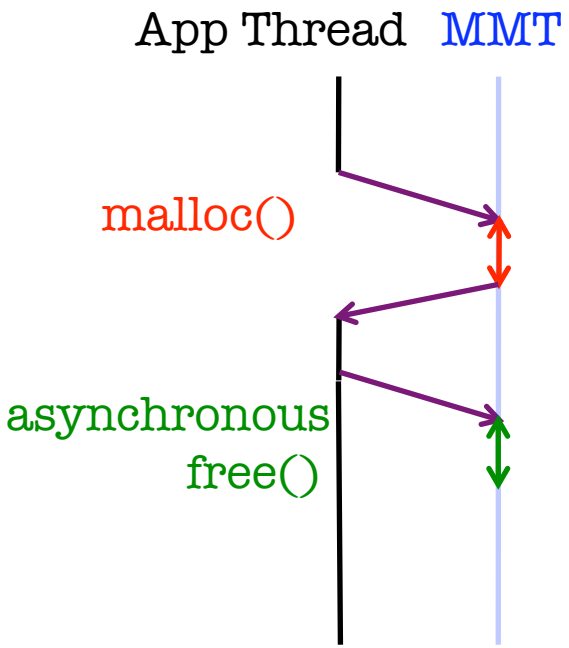


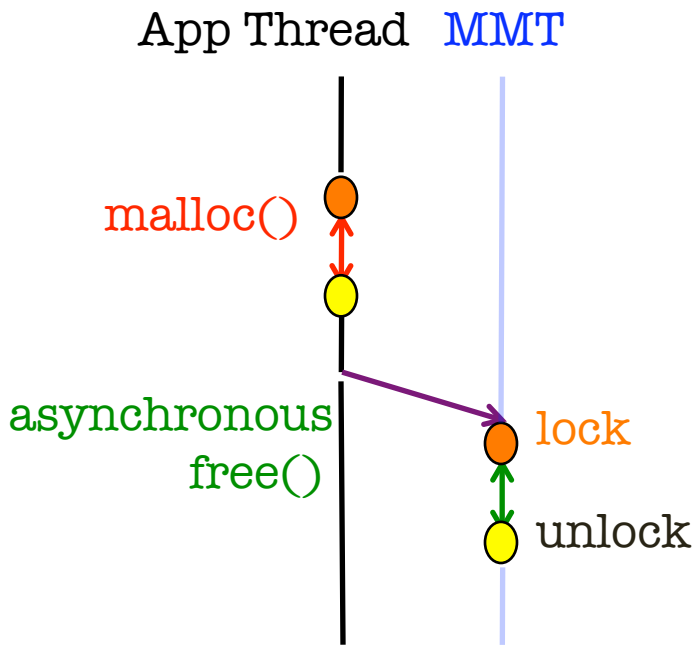
- Cost of semaphore > allocation latency & inter-request time
- Cost of locking ≈ allocation latency & inter-request time

Simple Solutions are Inadequate

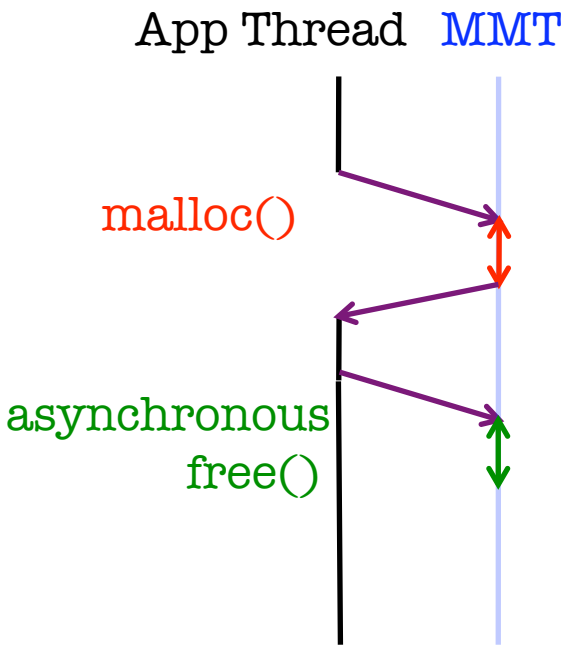


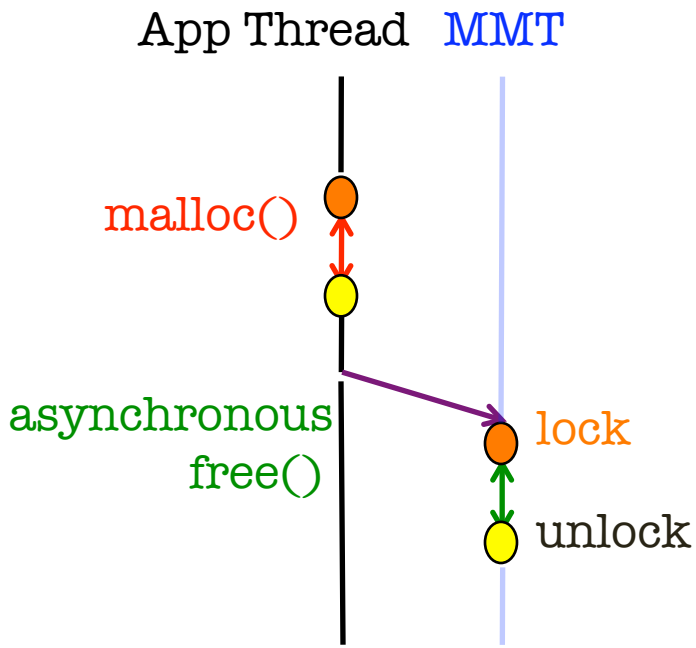
Simple Solutions are Inadequate





Simple Solutions are Inadequate





Thus, must exploit **enough parallelism** with **little communication cost**

Contributions

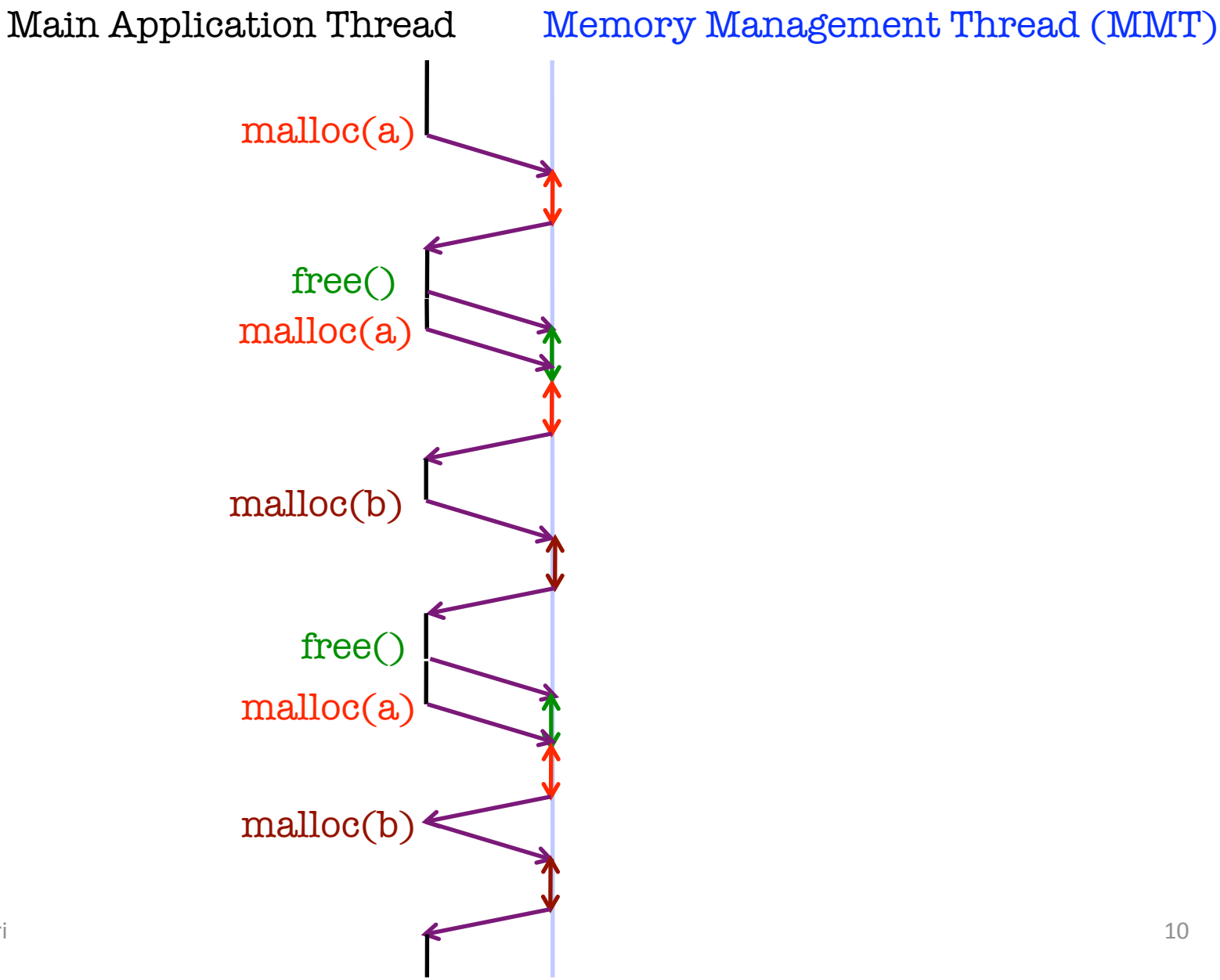
MMT for Speeding up Heap Allocation Intensive Sequential Applications

- ✓ Exploiting fine-grained function parallelism
- ✓ Reducing & hiding the high communication cost
- ✓ Agnostic to the underlying memory management library
- ✓ No hardware/compiler support
- ✓ No source code modification
- ✓ Speed up of 1.19x over heap intensive sequential applications

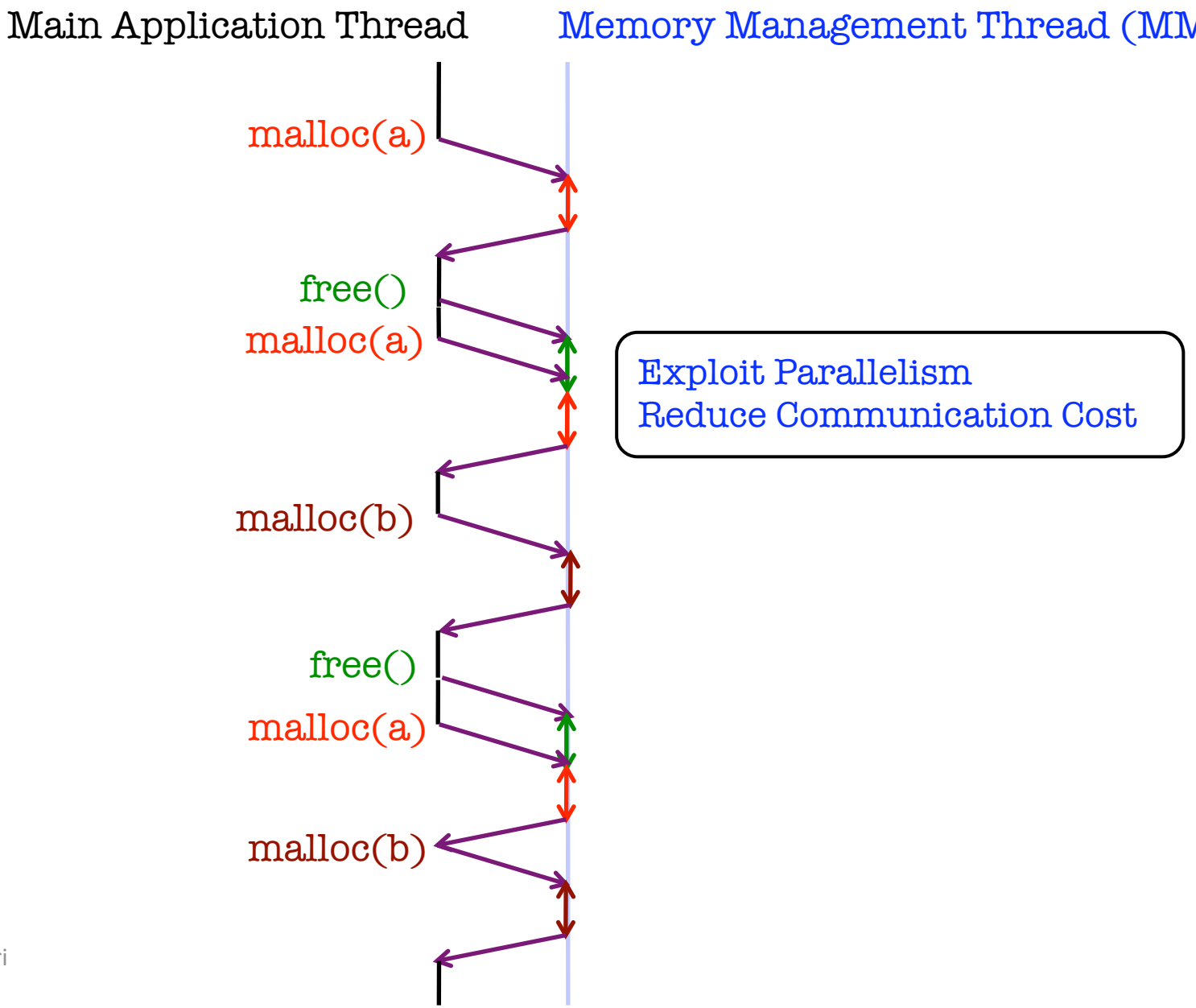
Outline

- Motivation
- Memory Management Thread : MMT Approach
- MMT Design and Implementation
  - Speculative memory Management
  - Bulk Memory Management
  - Understanding the Interaction
- Evaluation
- Conclusion

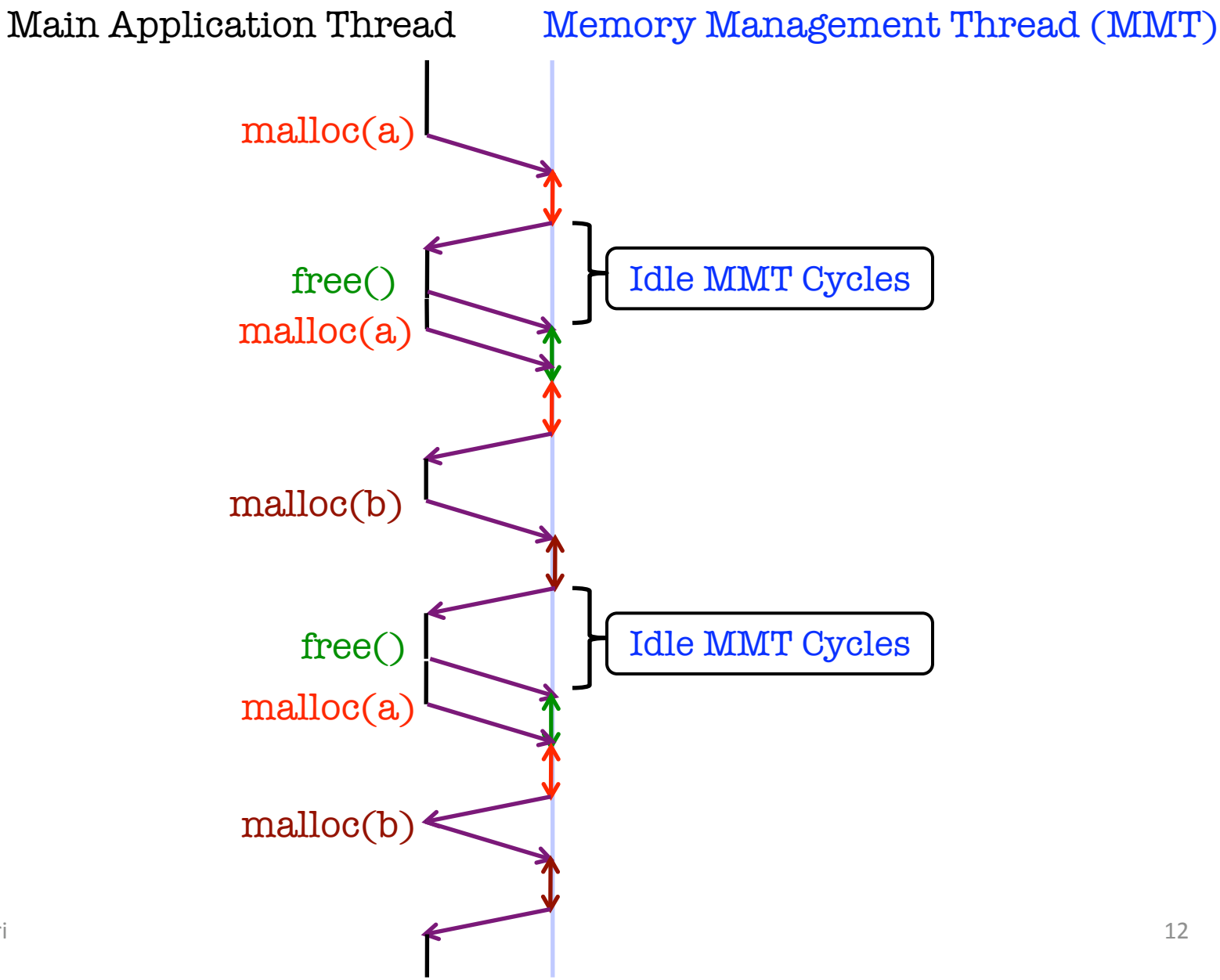
Speculative Memory Management



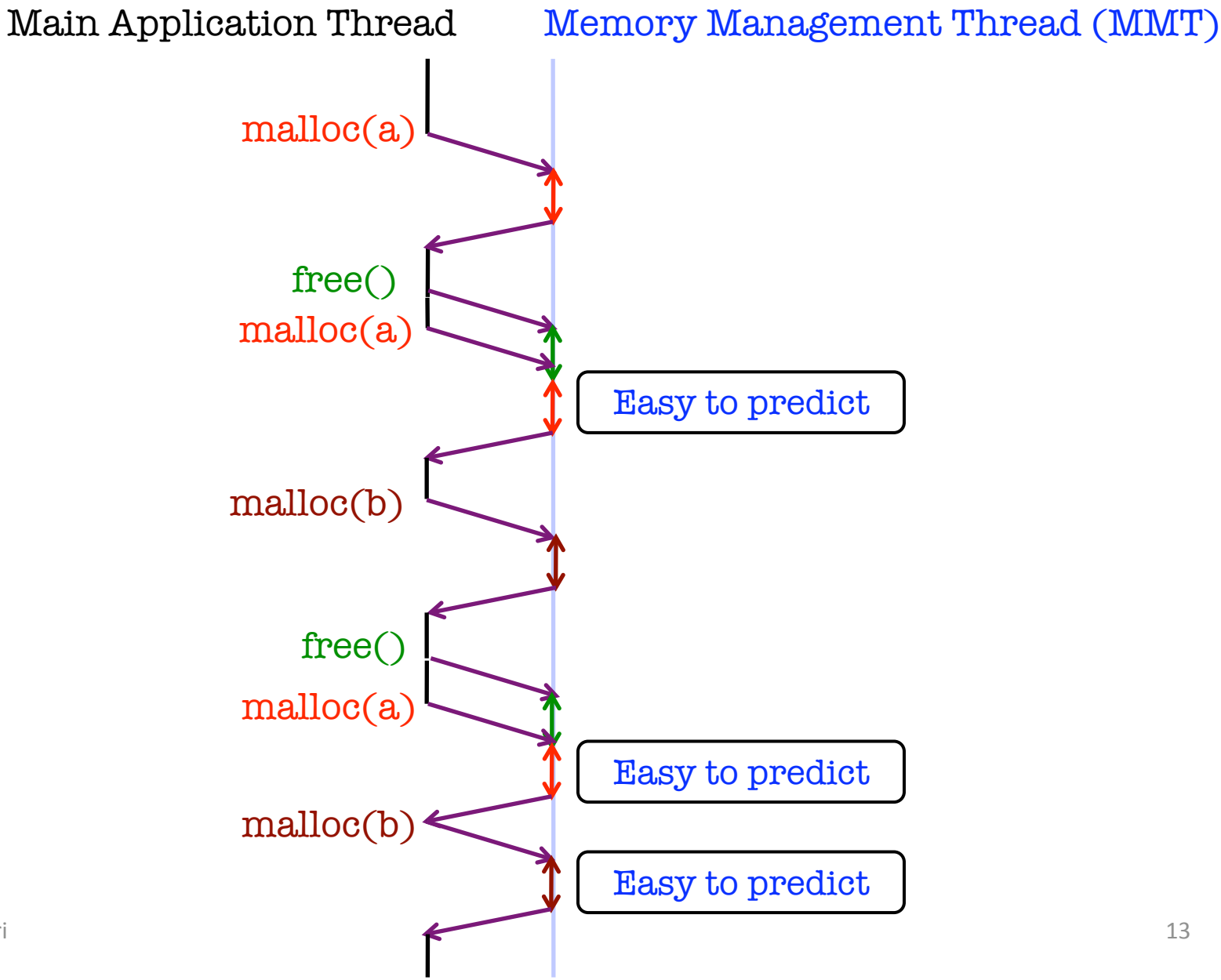
Speculative Memory Management



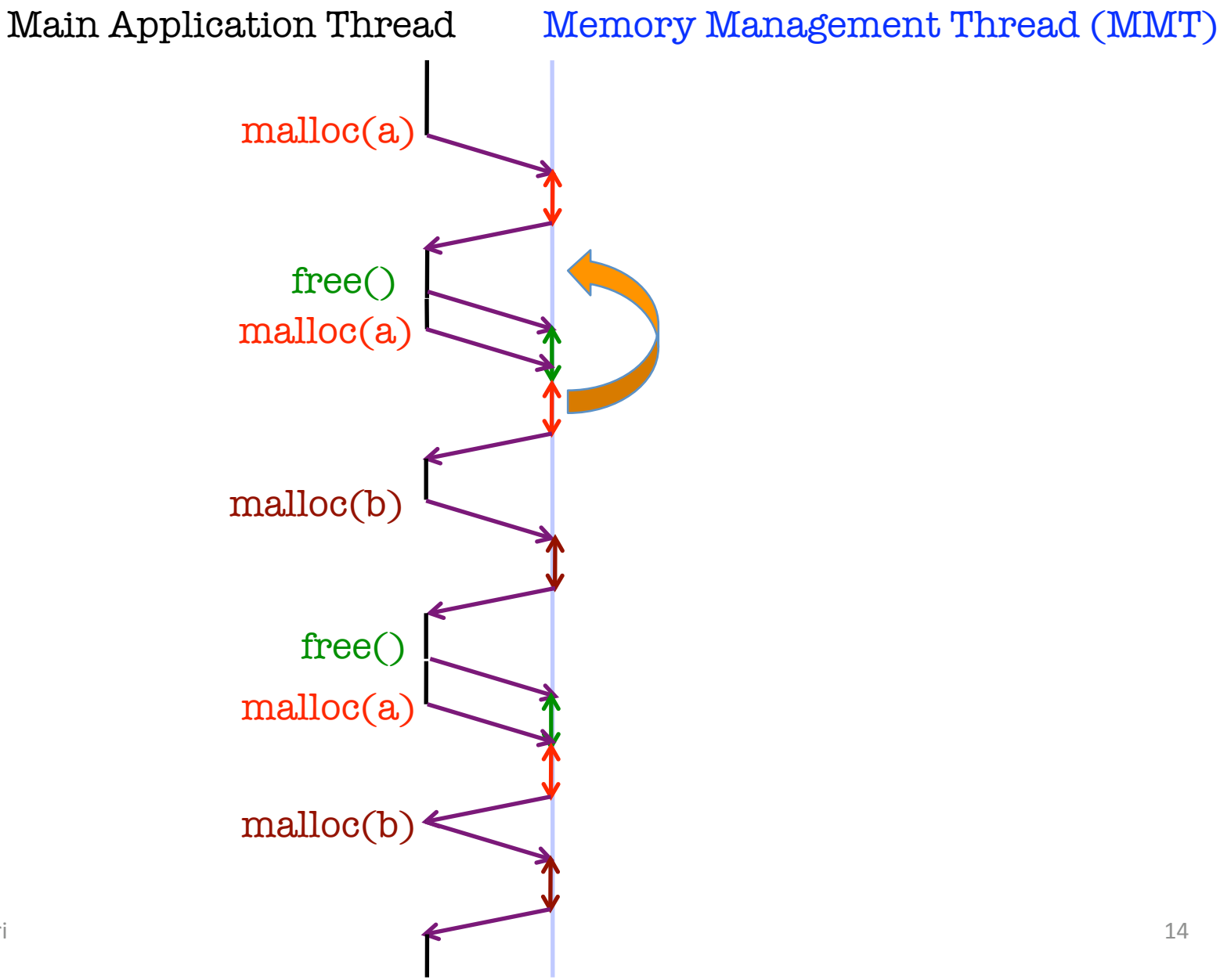
Speculative Memory Management



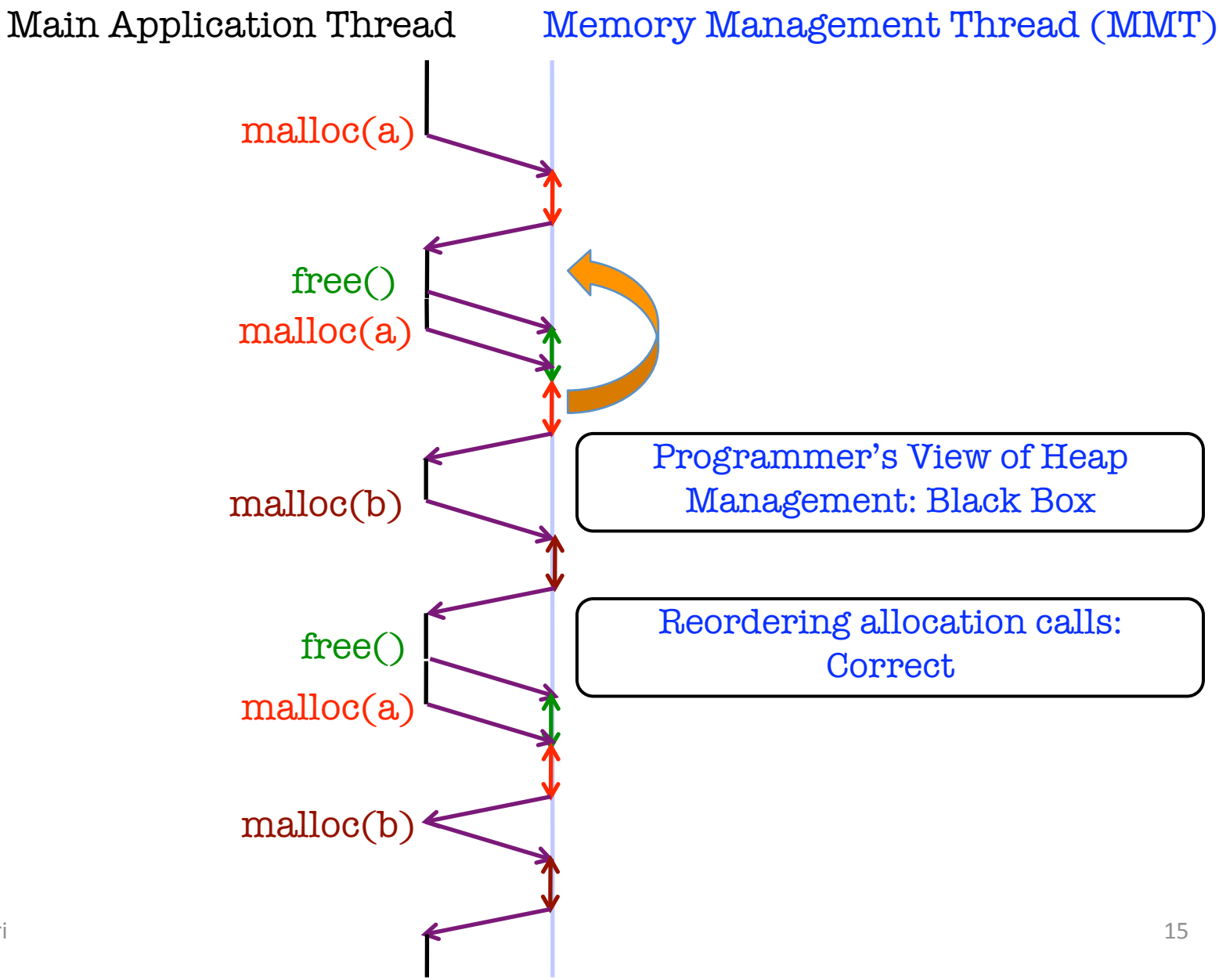
Speculative Memory Management



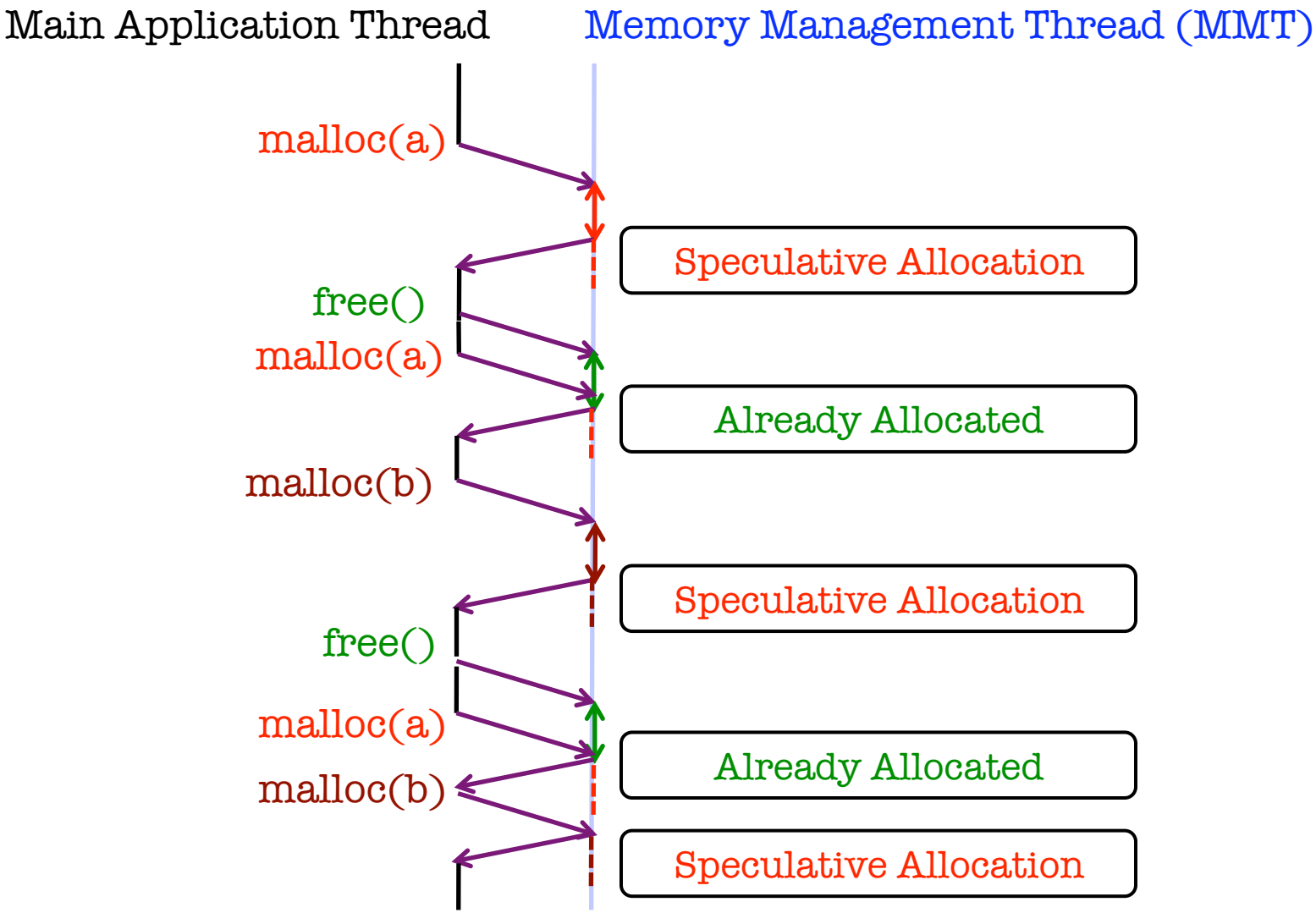
Speculative Memory Management



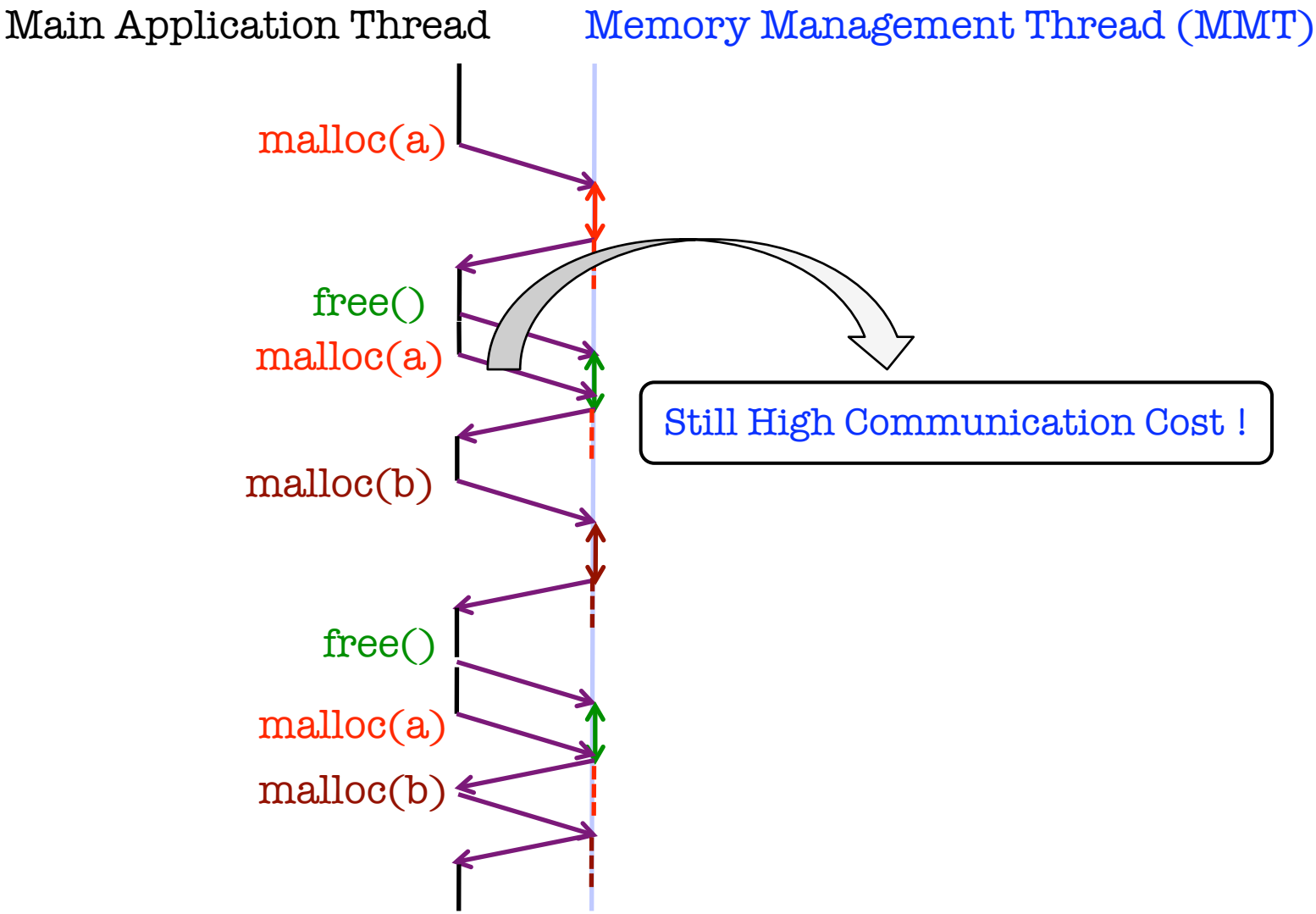
Speculative Memory Management



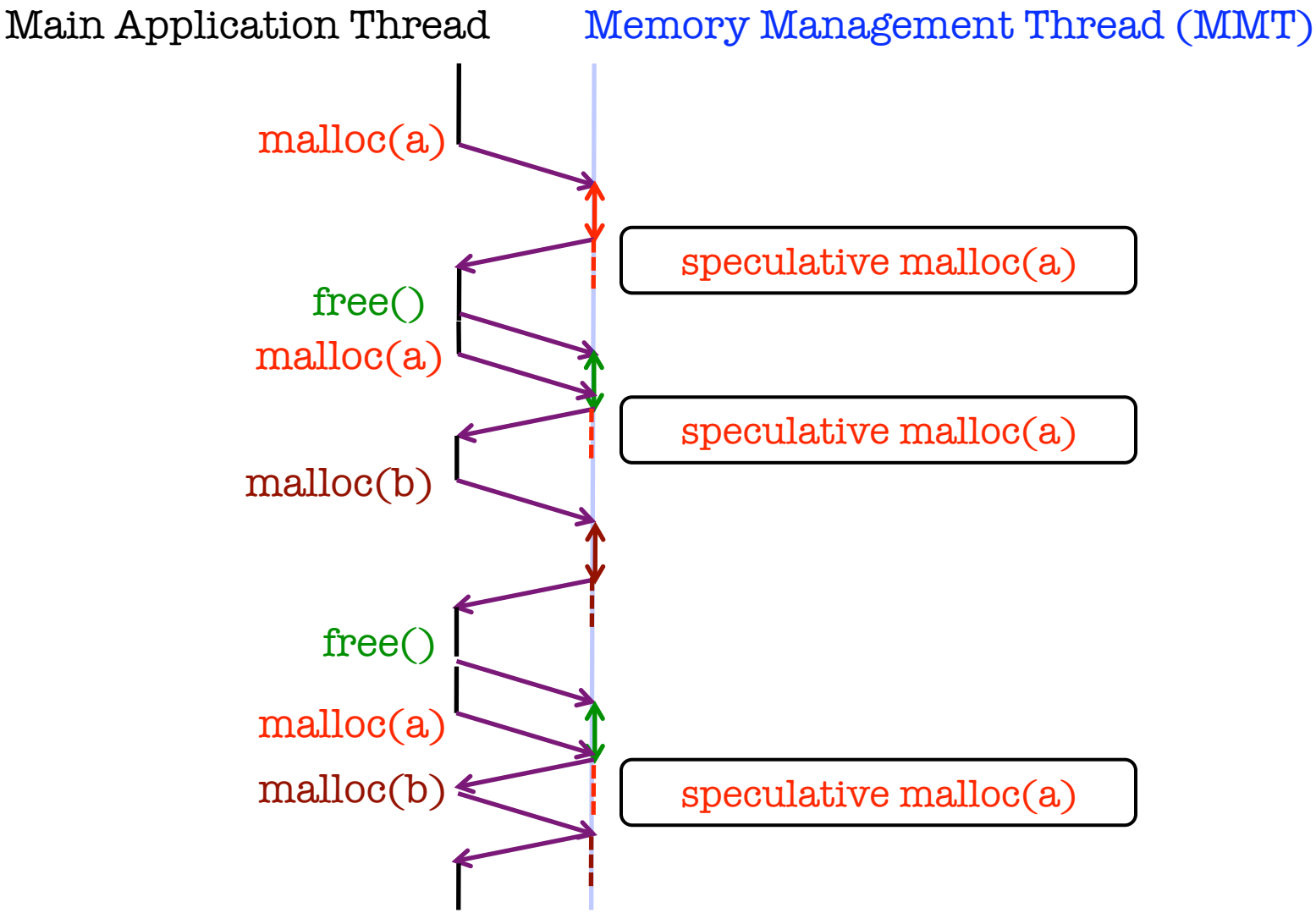
Speculative Memory Management



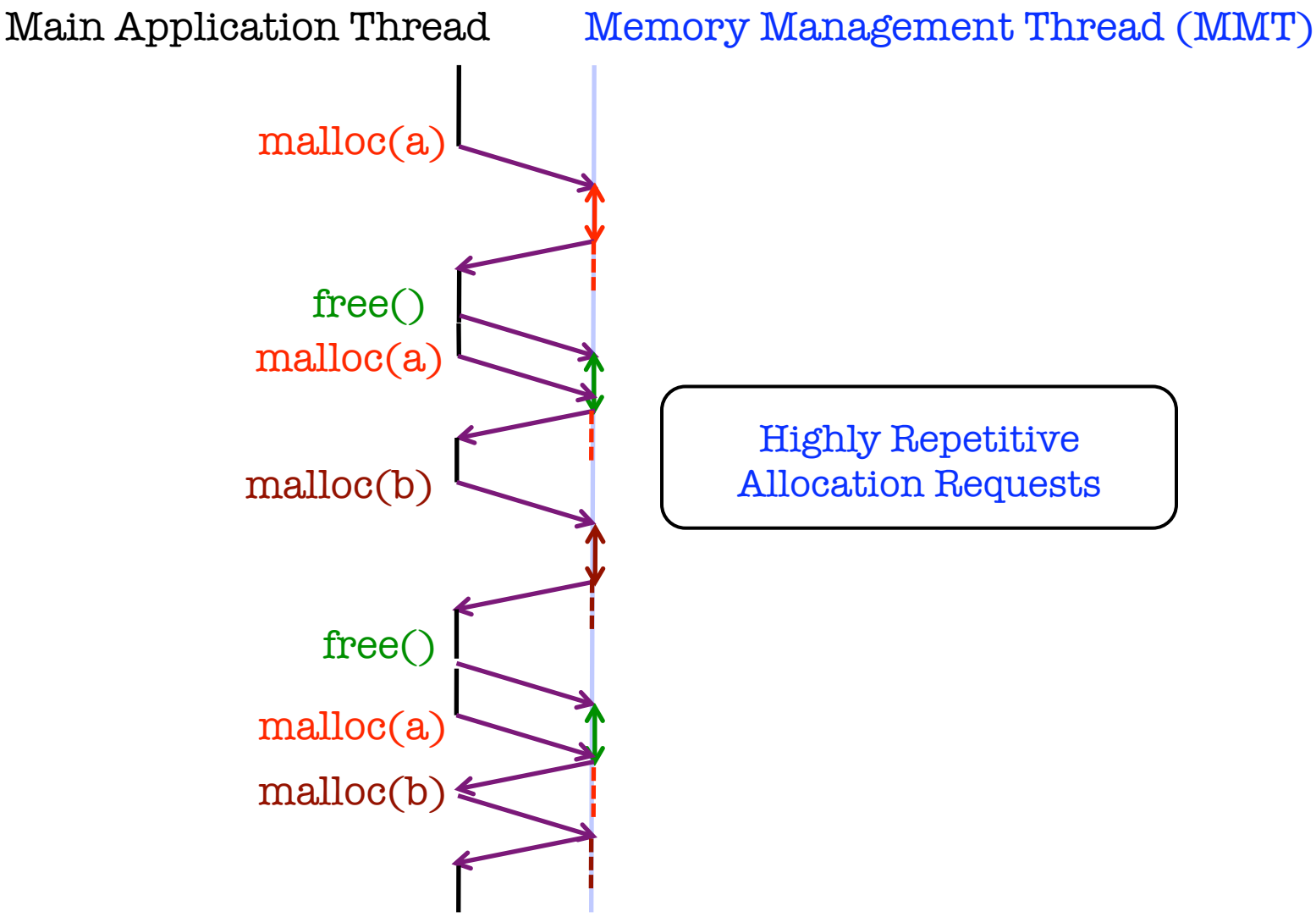
Speculative Memory Management



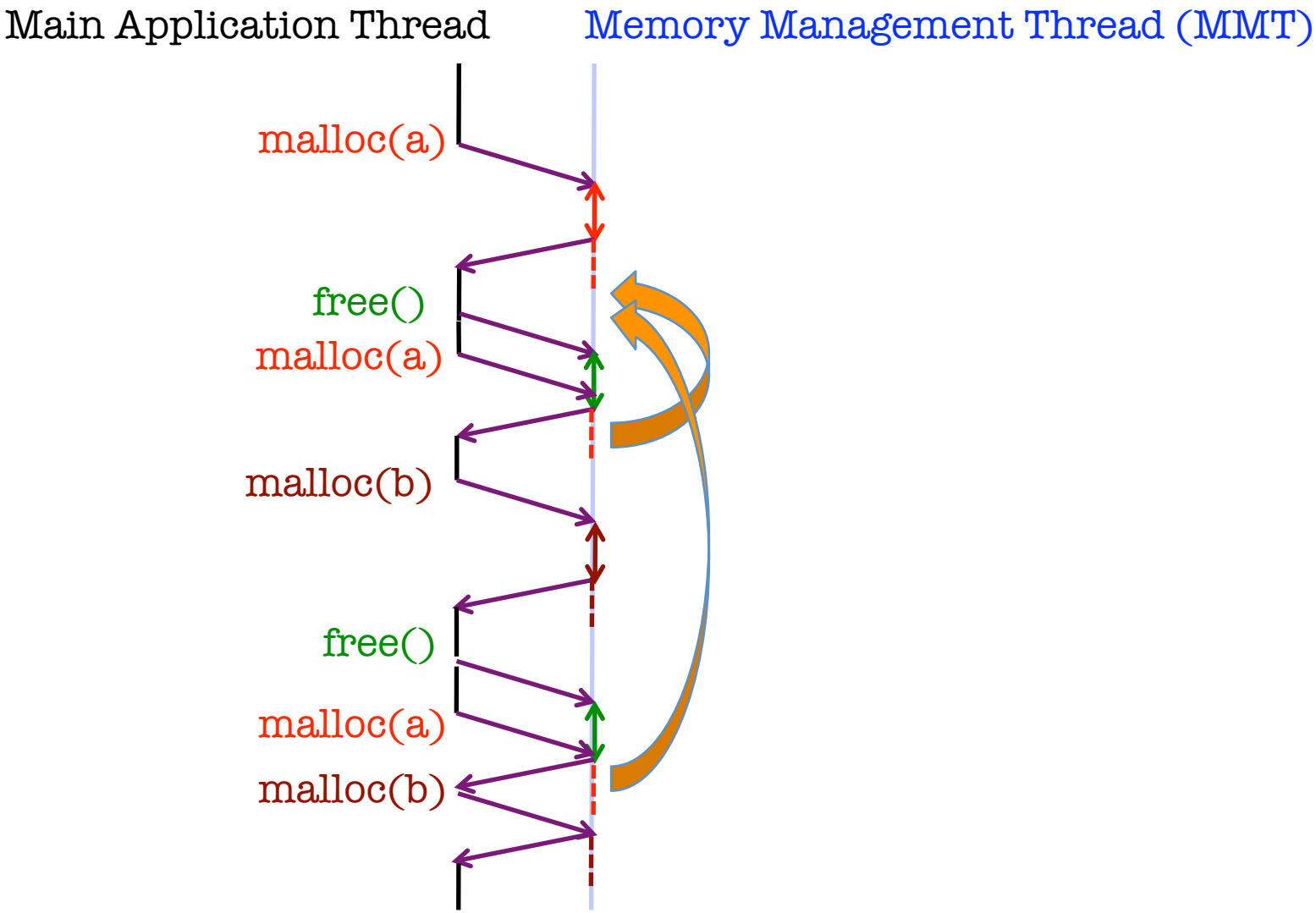
Speculative Memory Management



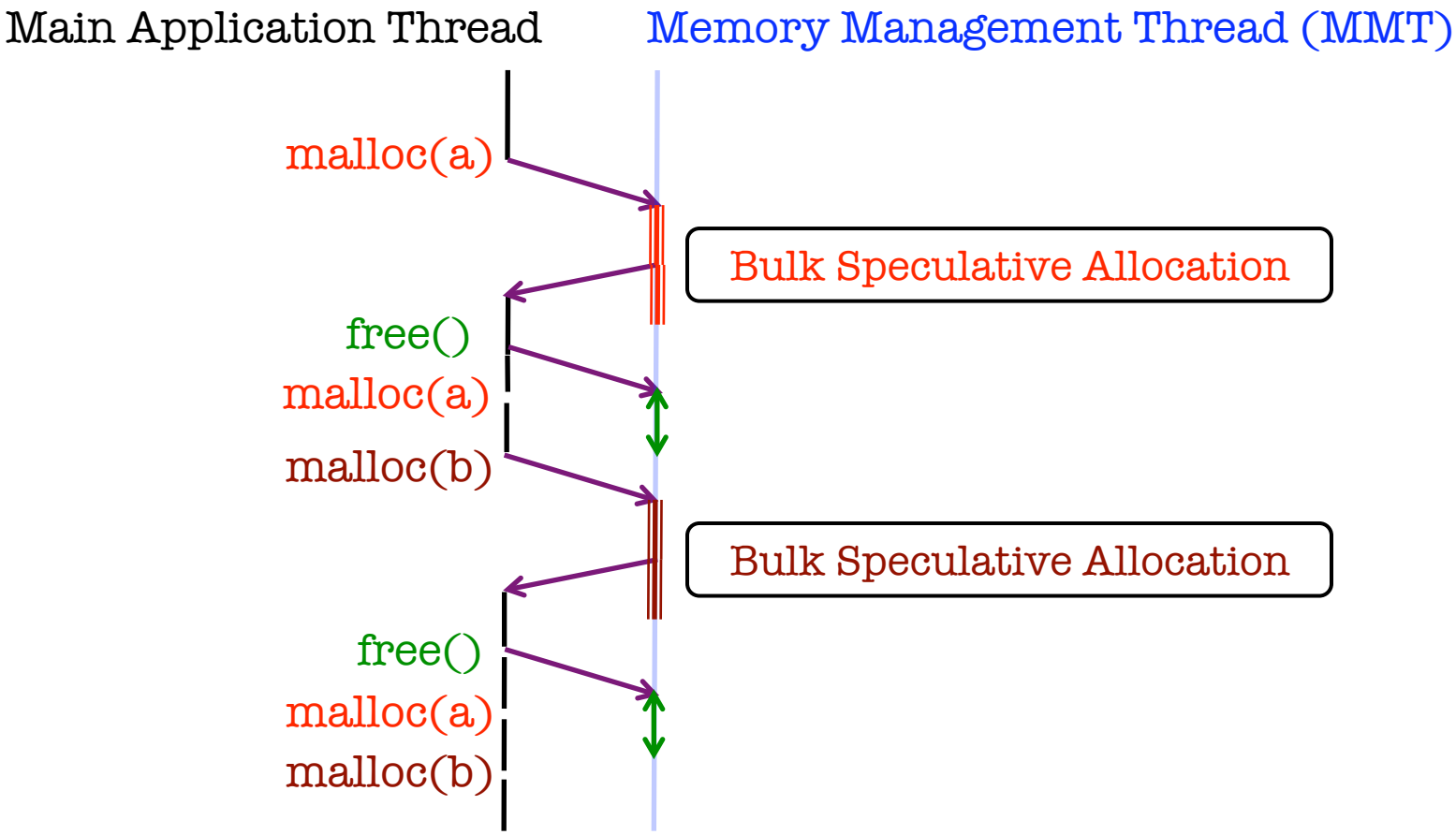
Speculative Memory Management



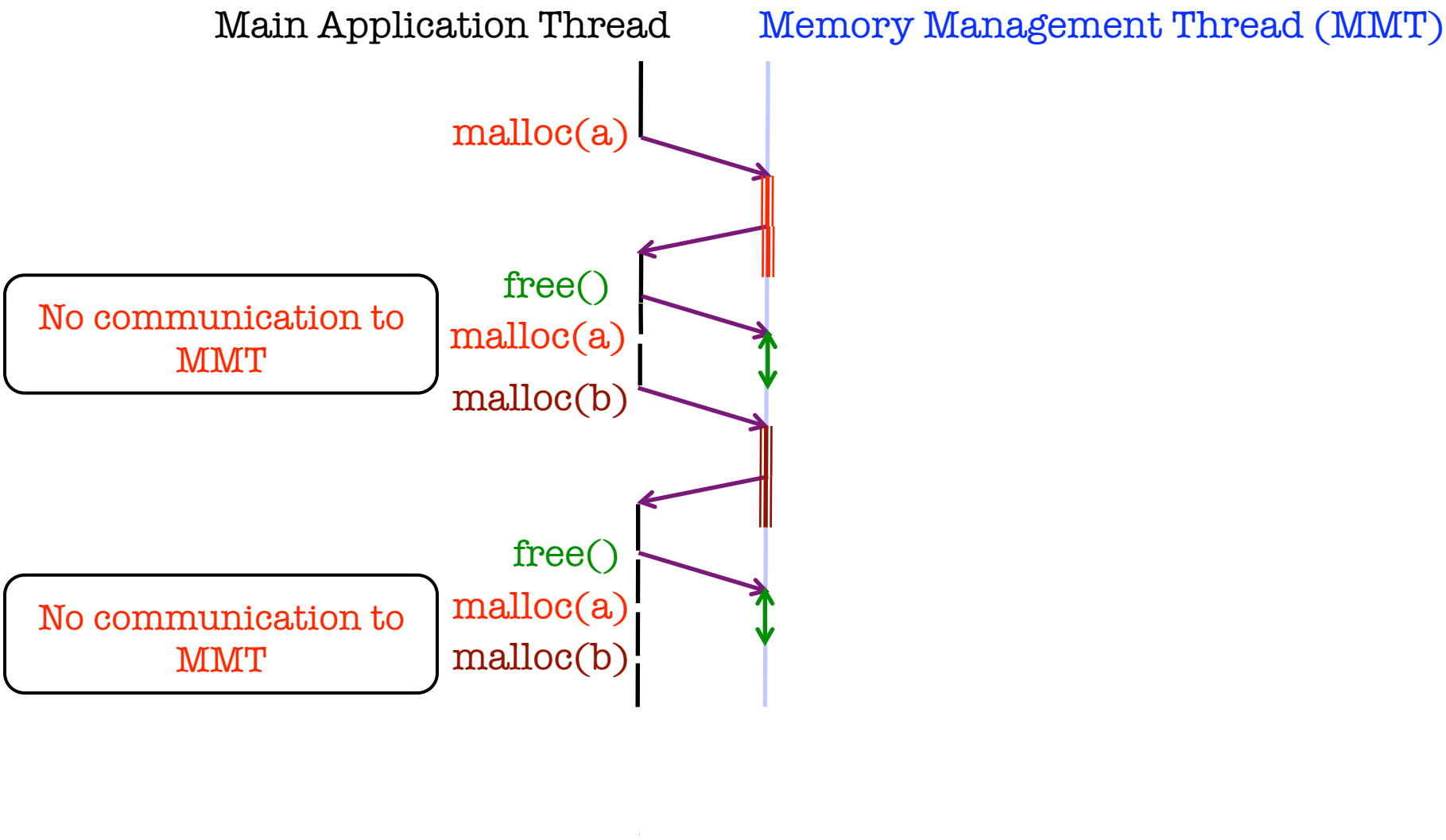
Speculative Memory Management



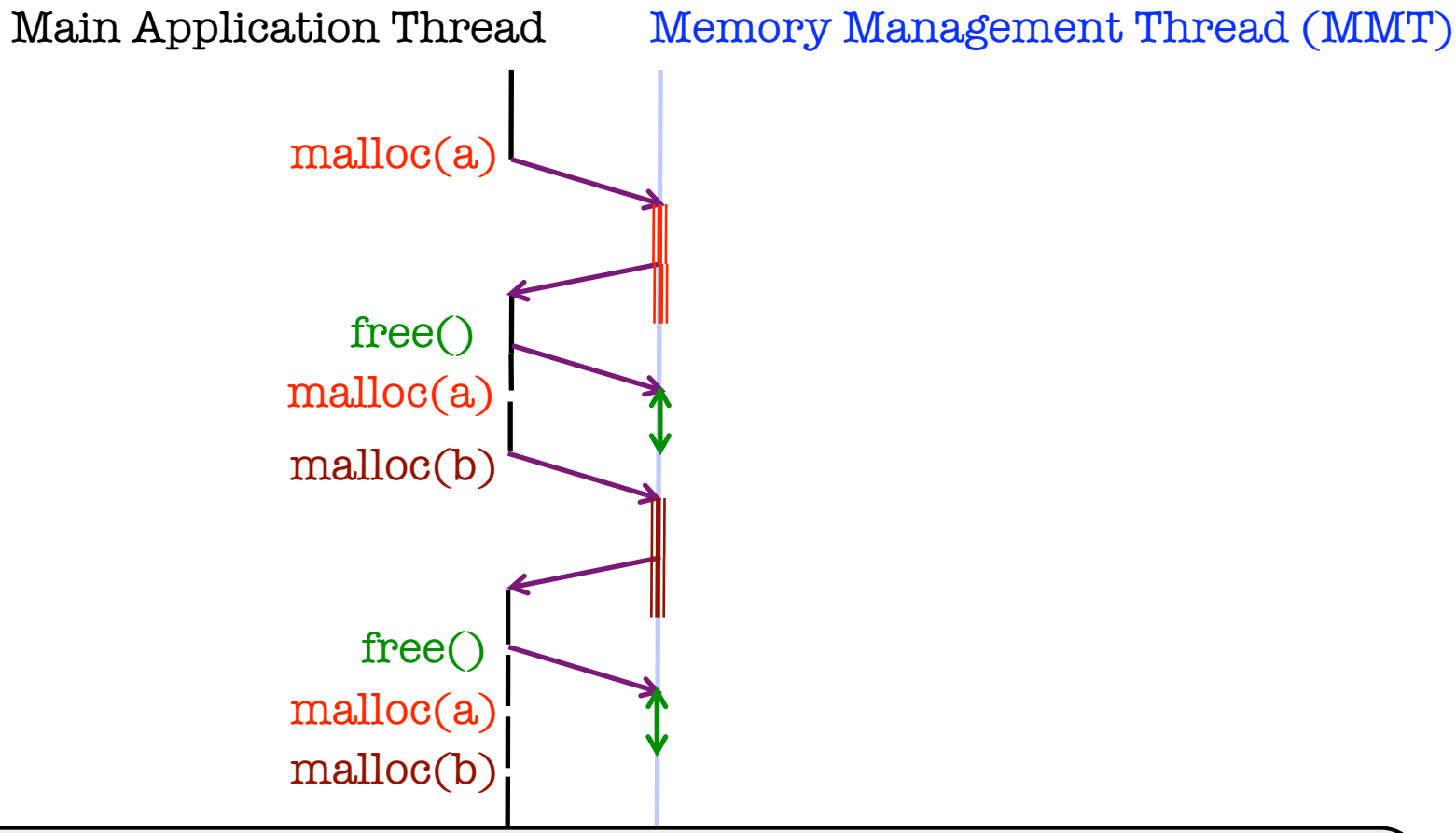
Bulk Memory Management



Bulk Memory Management



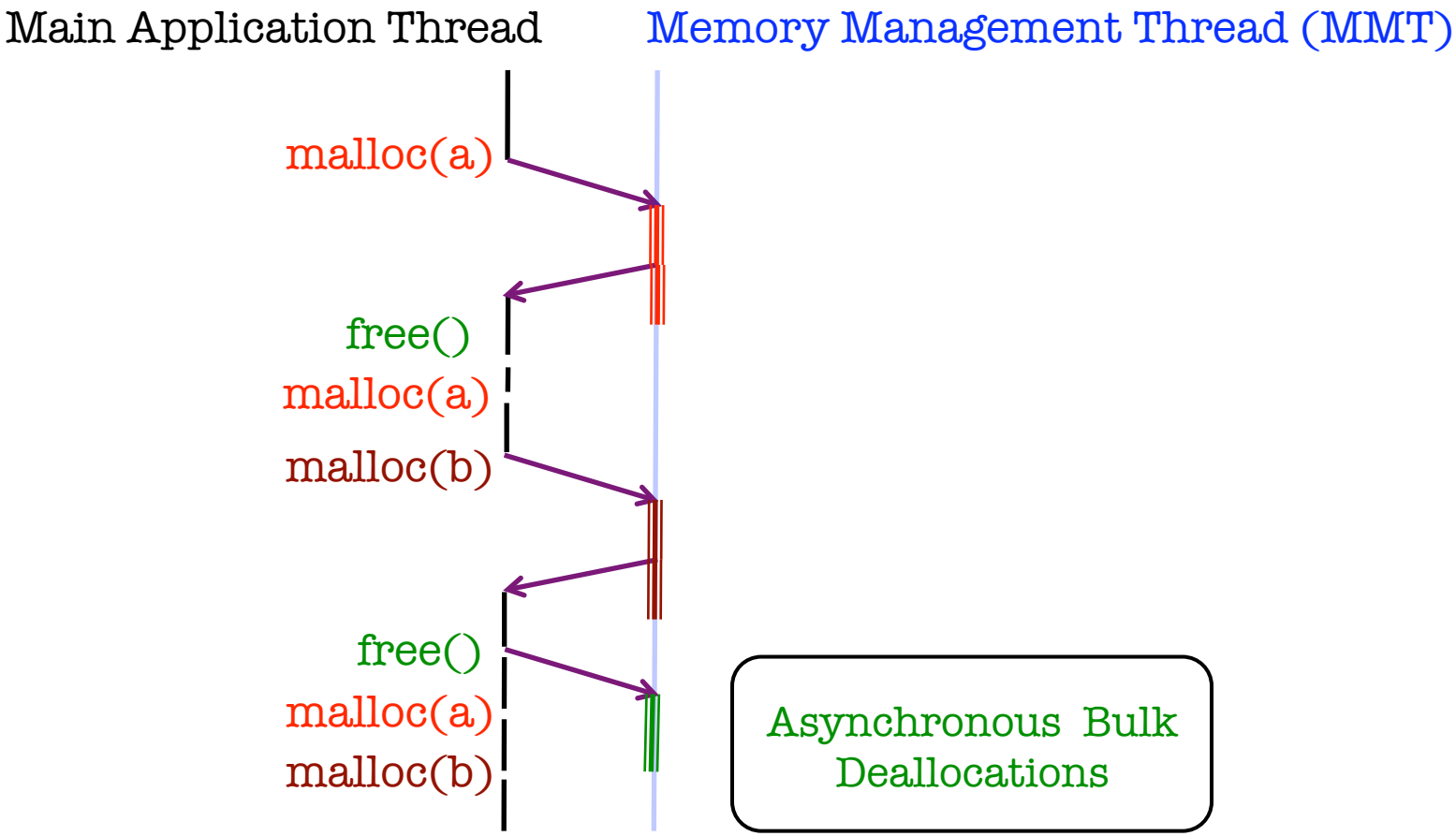
Bulk Memory Management



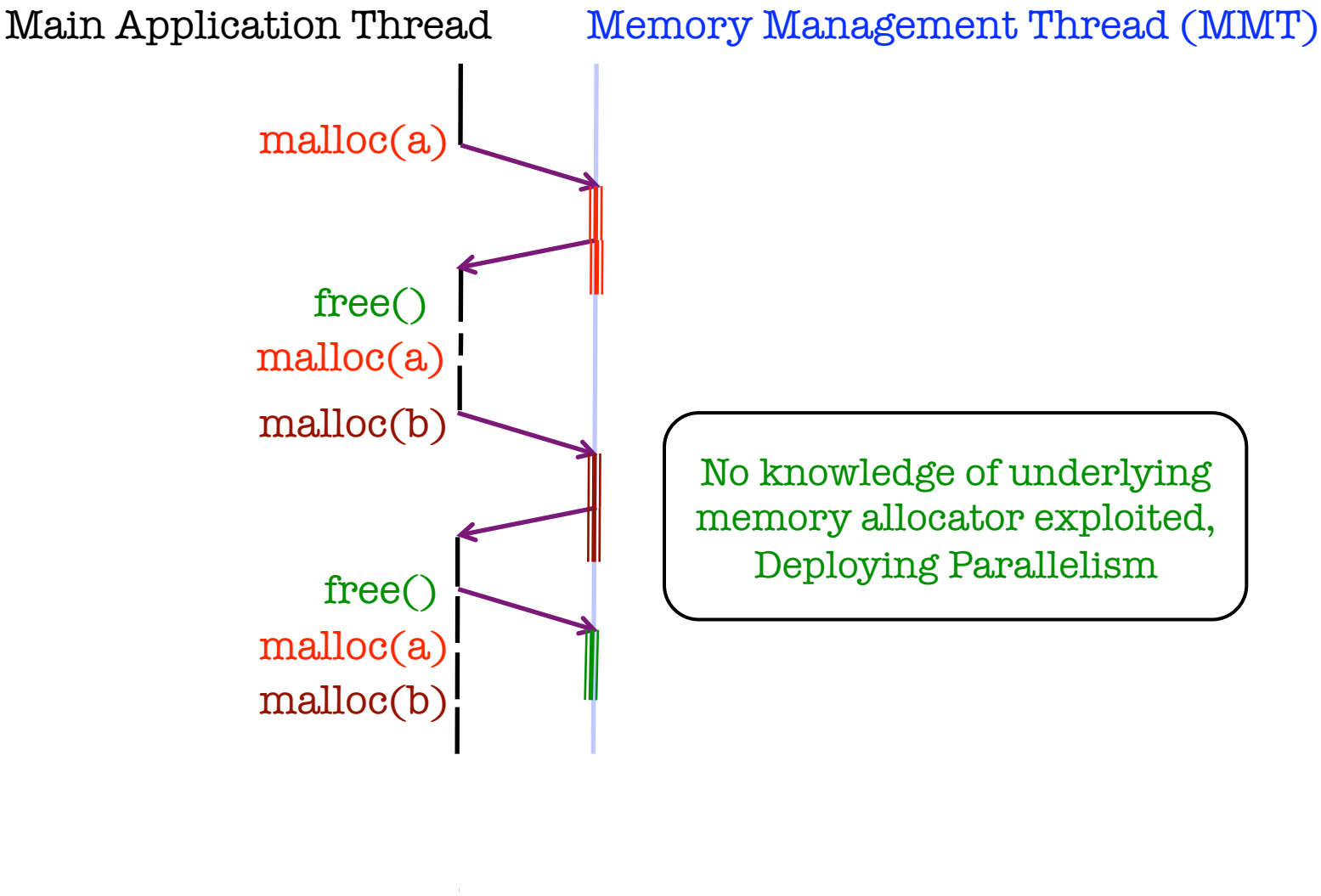
Speculative Bulk Allocation

- Cost of Miss-speculated Bulk Allocations
- Starting Time and Quanta for Bulk Preallocations

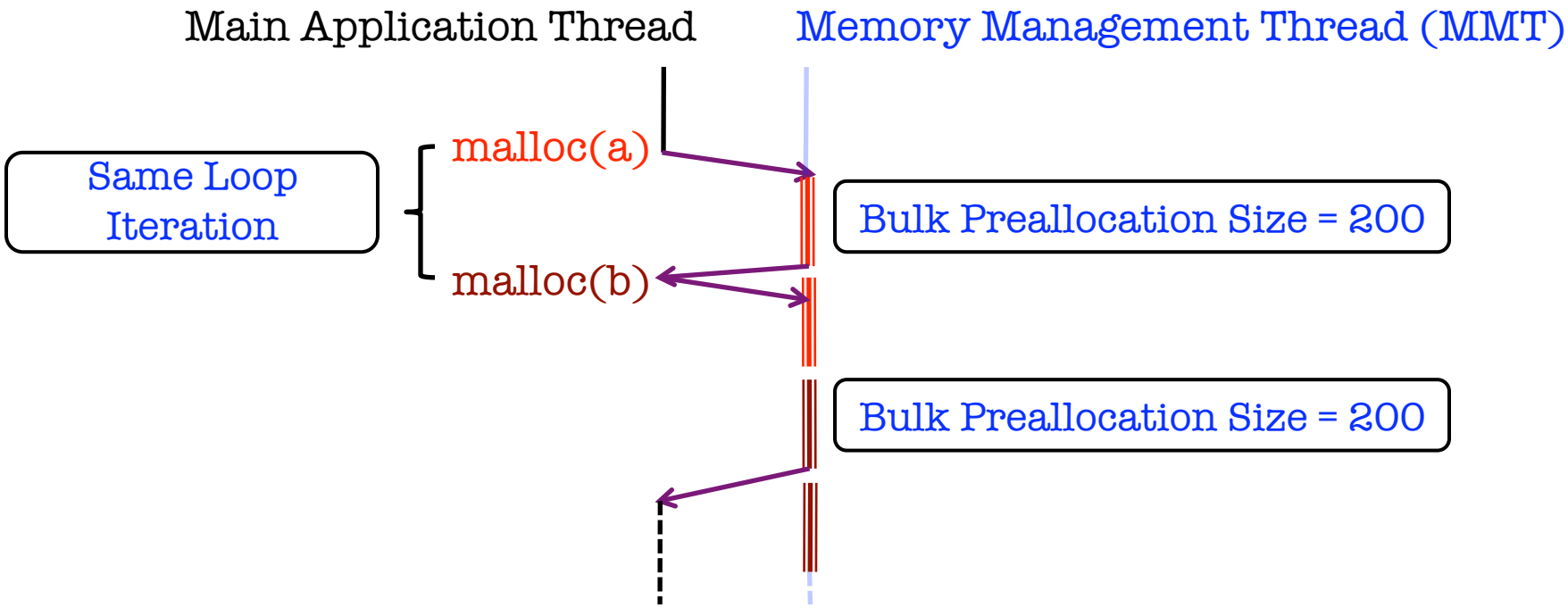
Bulk Memory Management



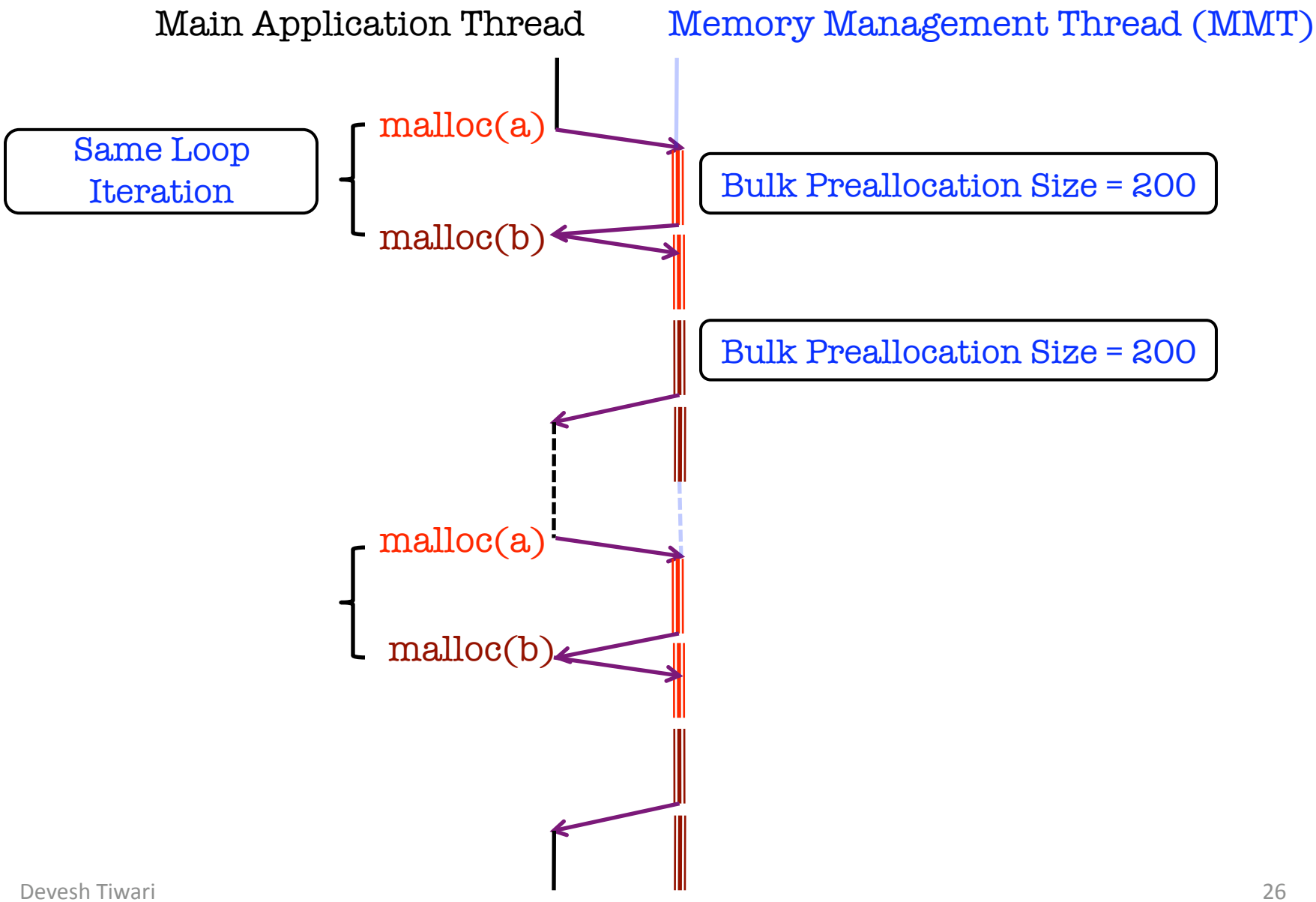
Bulk Memory Management



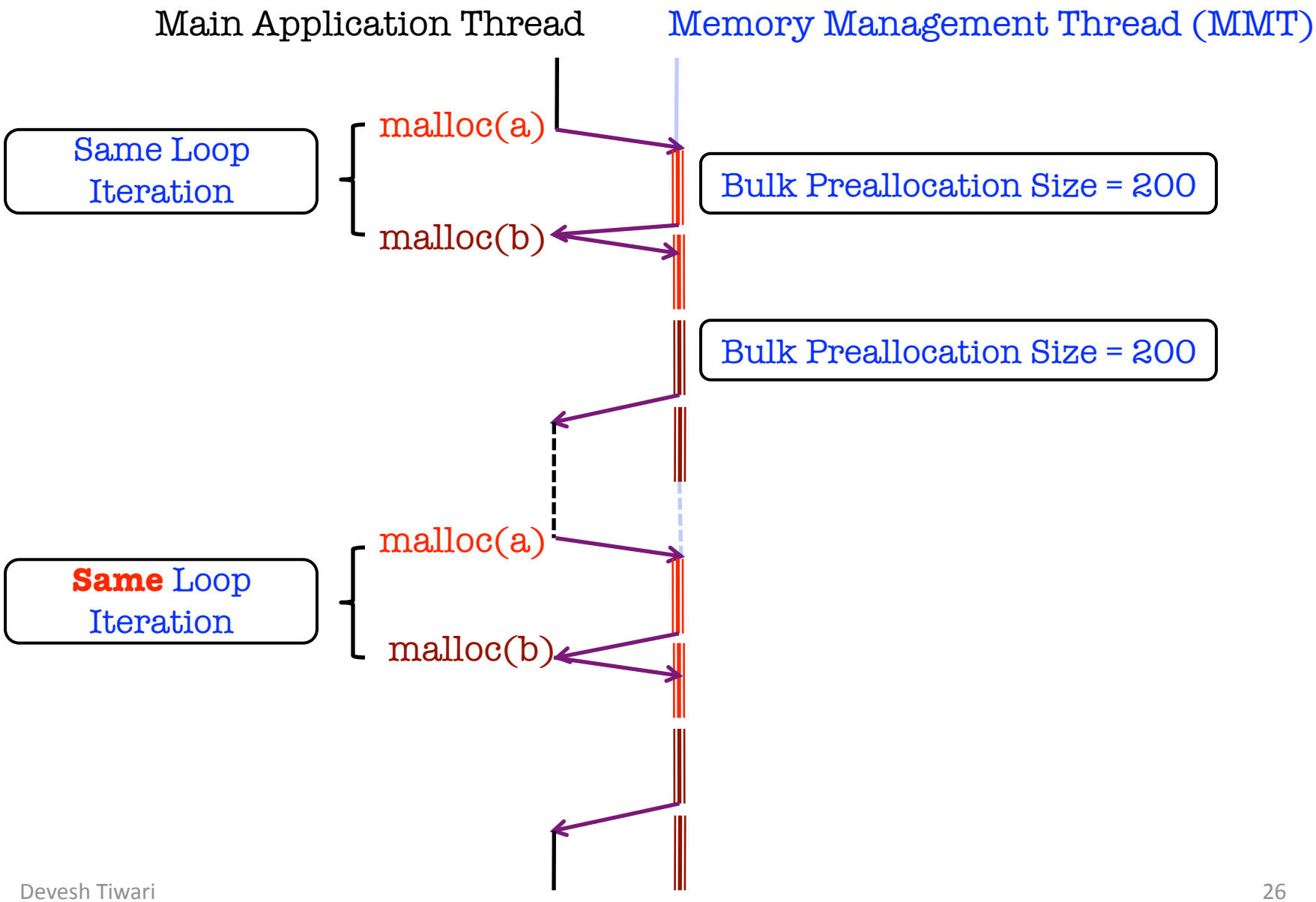
Bulk Memory Management



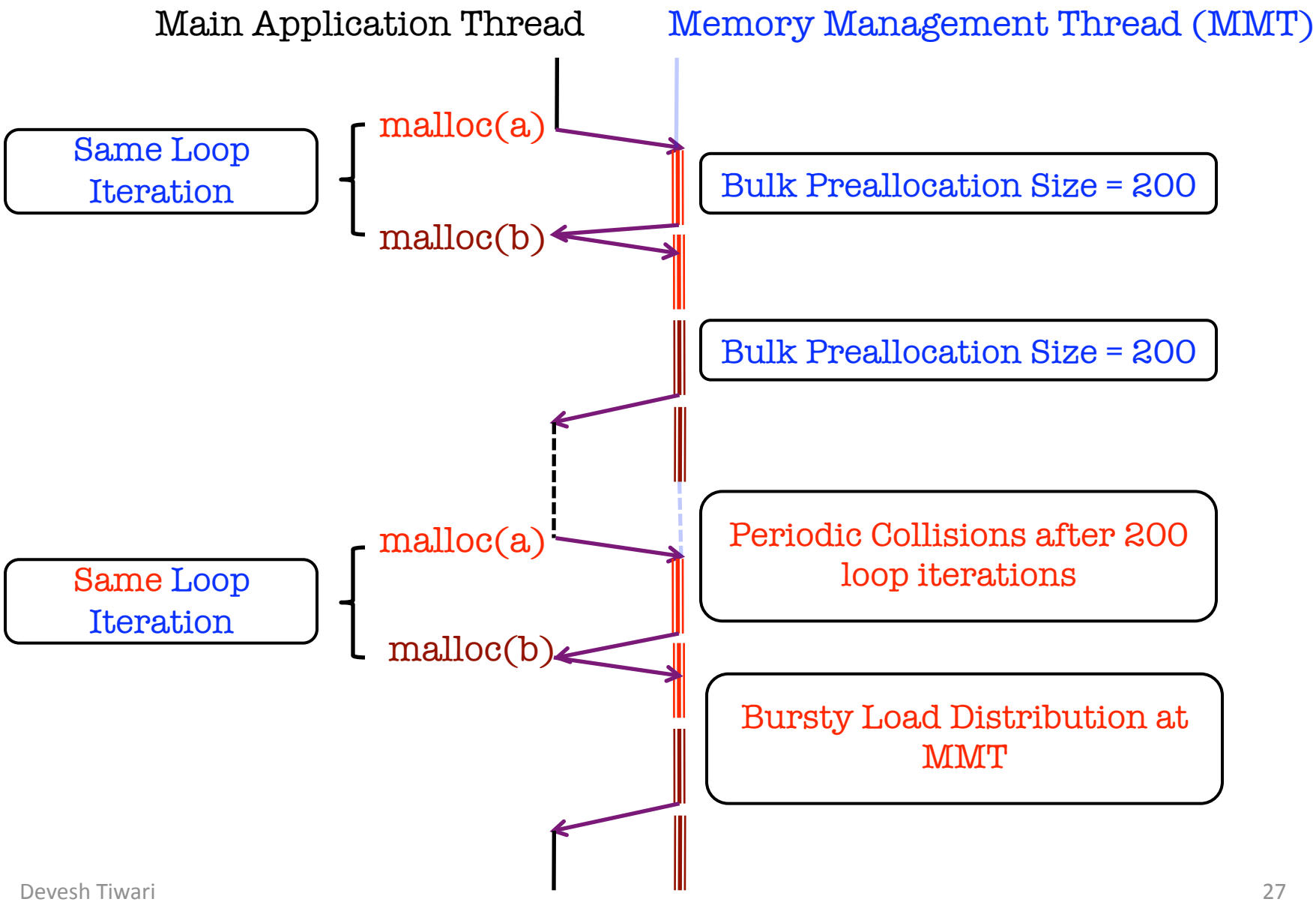
Bulk Memory Management



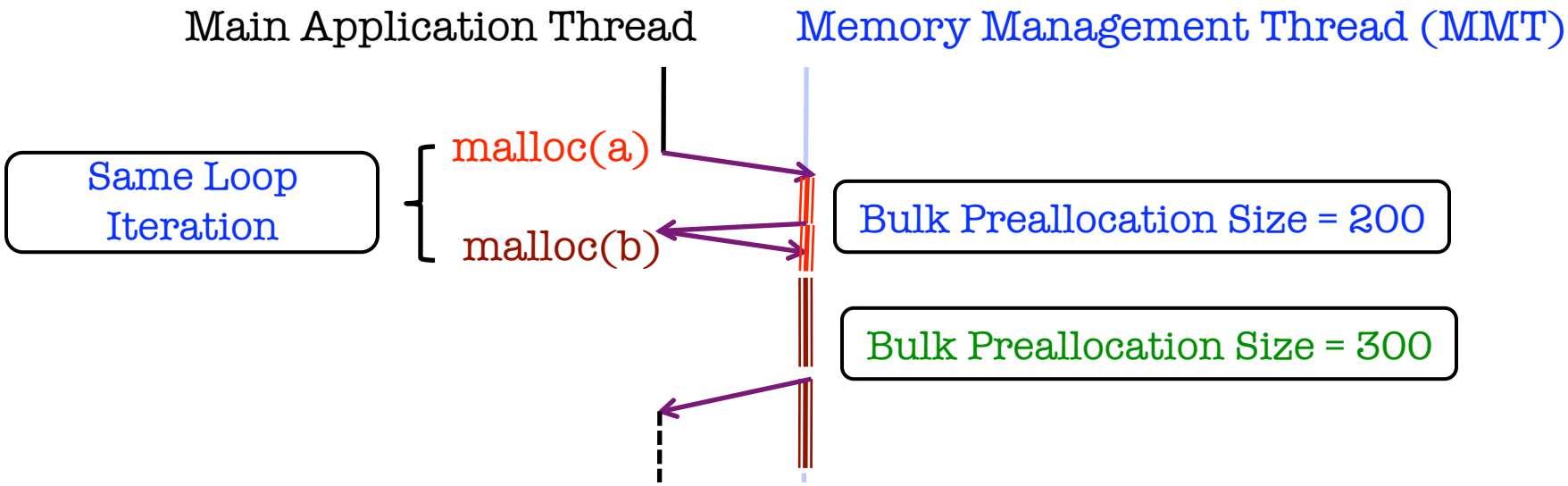
Bulk Memory Management



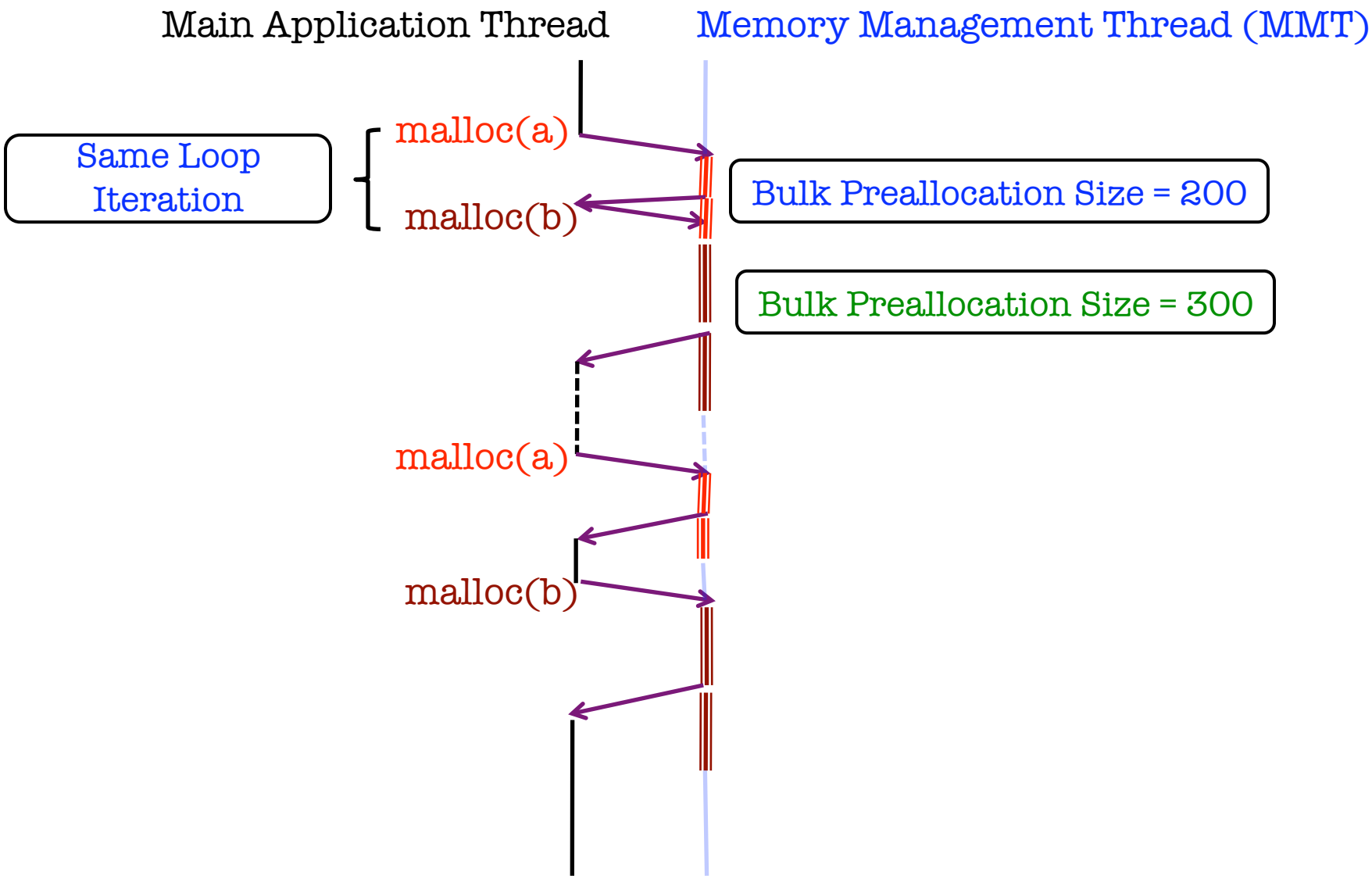
Bulk Memory Management



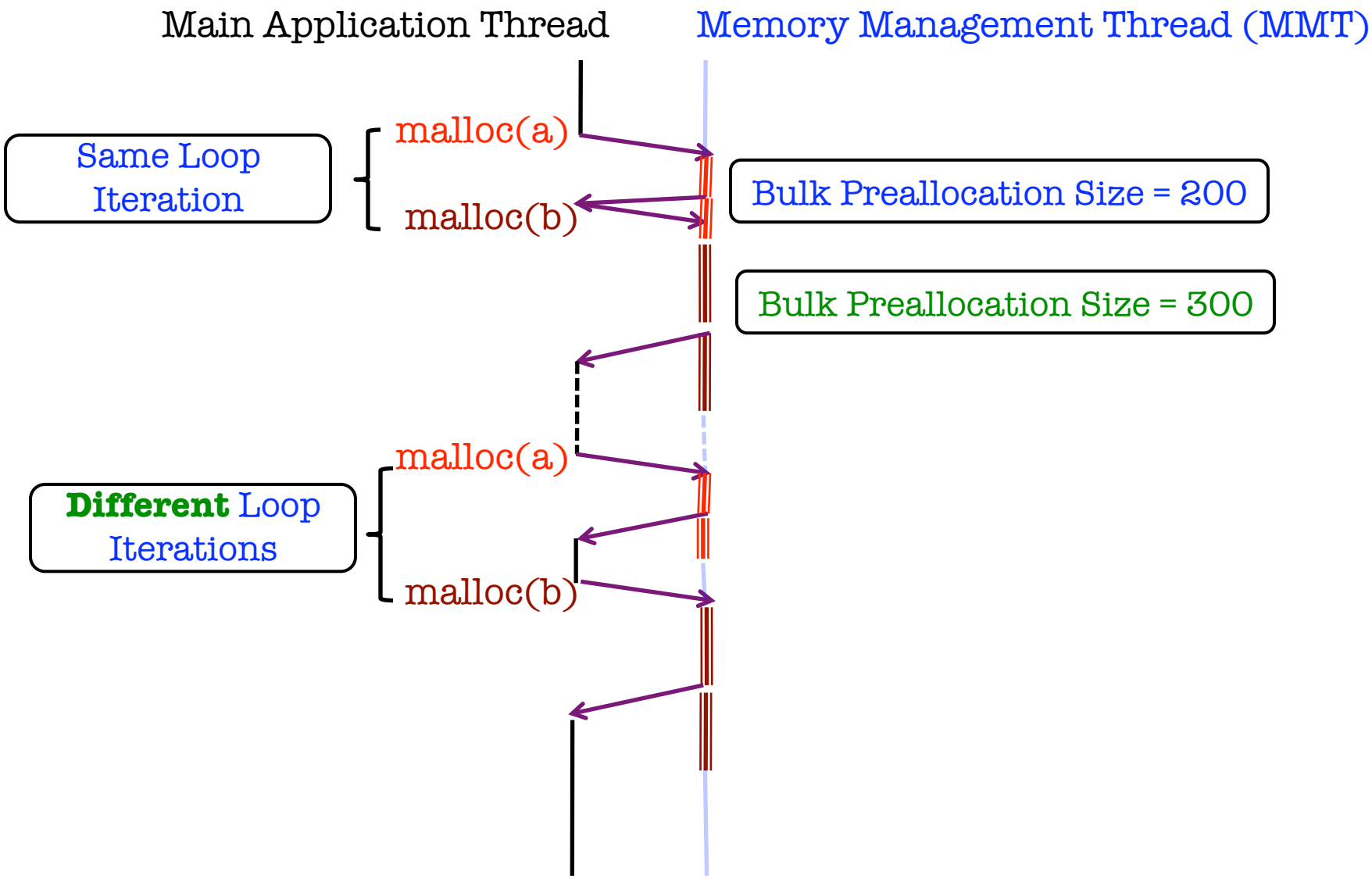
Bulk Memory Management



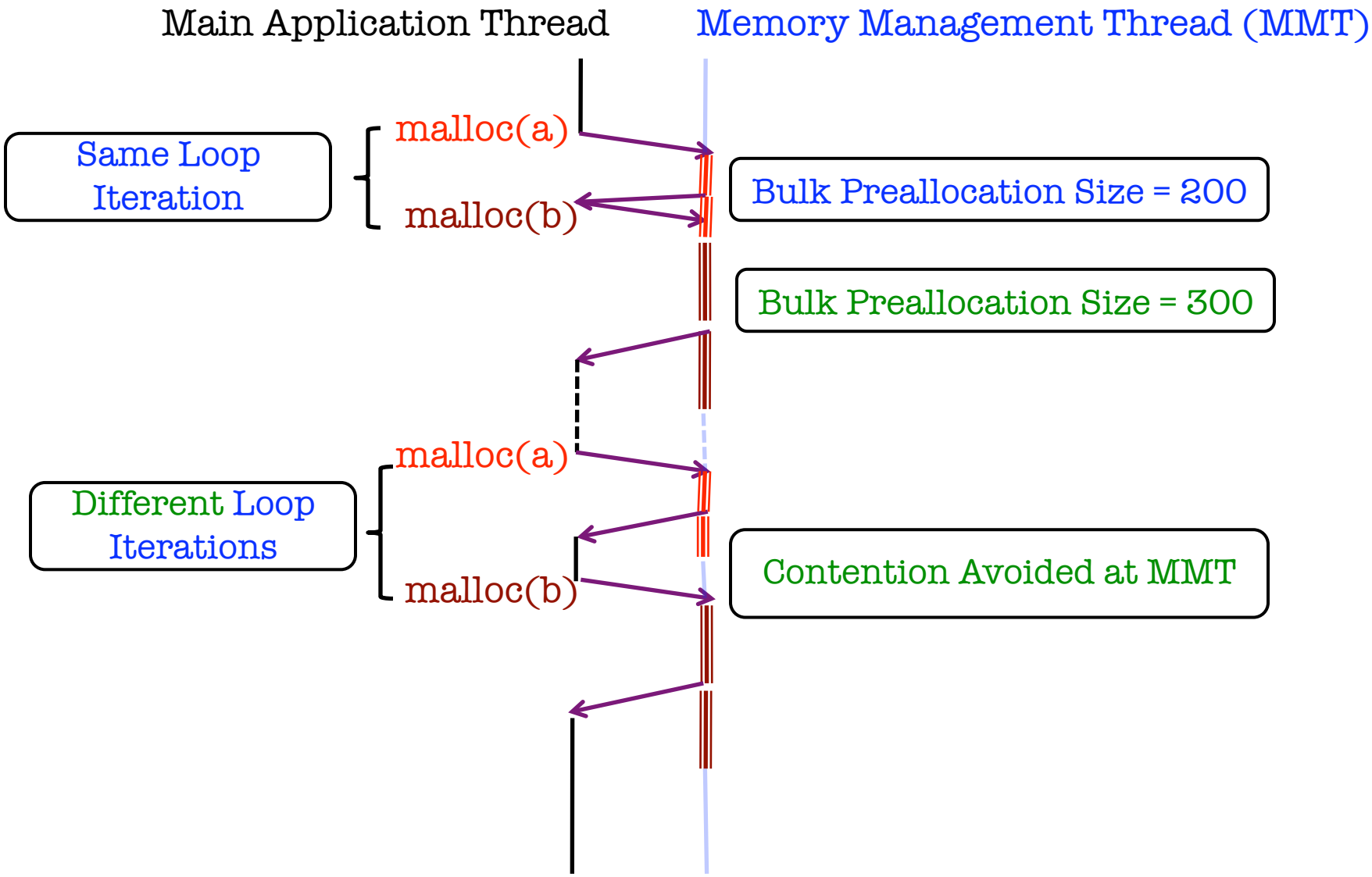
Bulk Memory Management



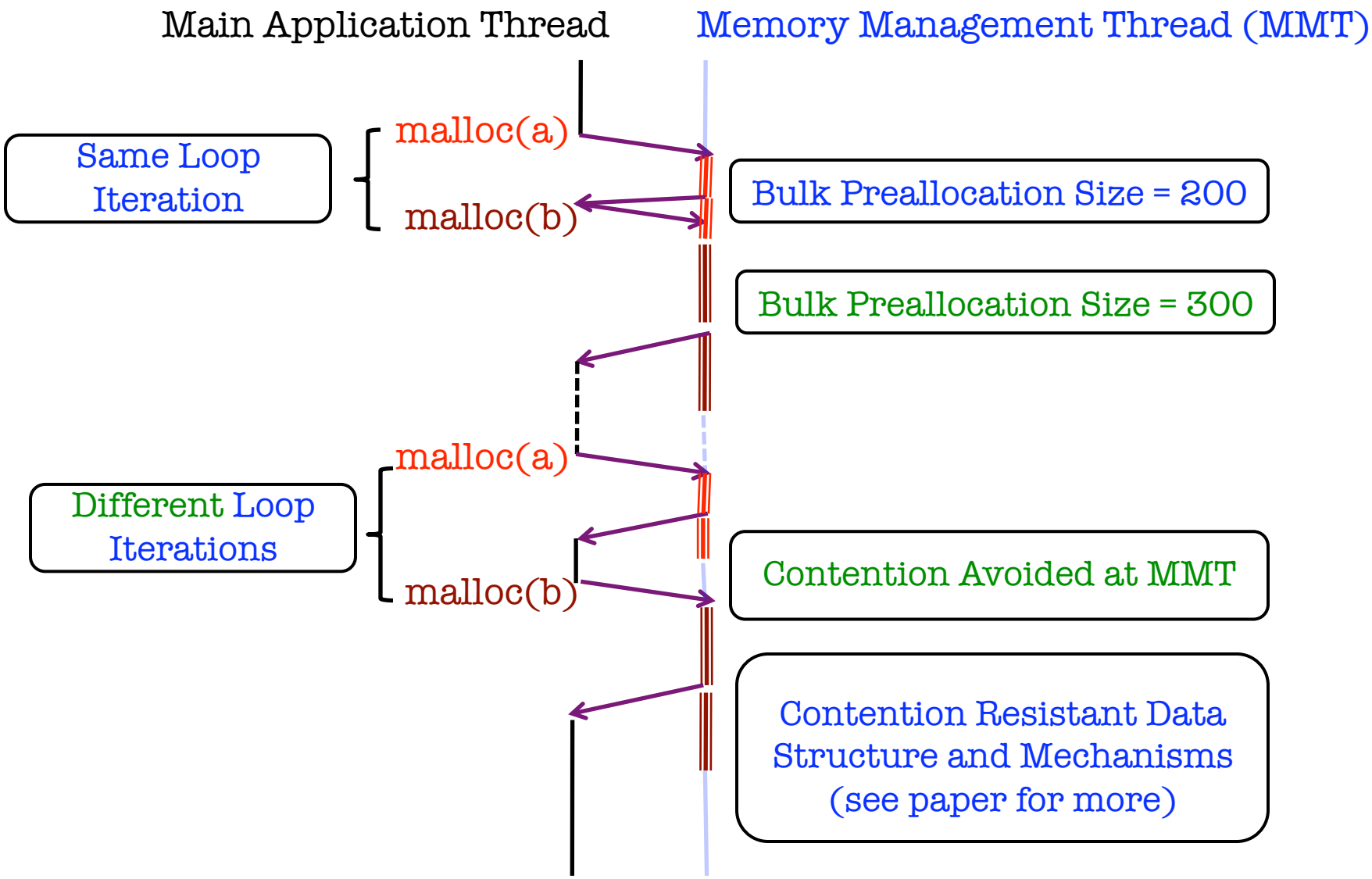
Bulk Memory Management



Bulk Memory Management



Bulk Memory Management



Outline

- Motivation
- Memory Management Thread : MMT Approach
- MMT Design and Implementation
- Evaluation
- Conclusion

Evaluation

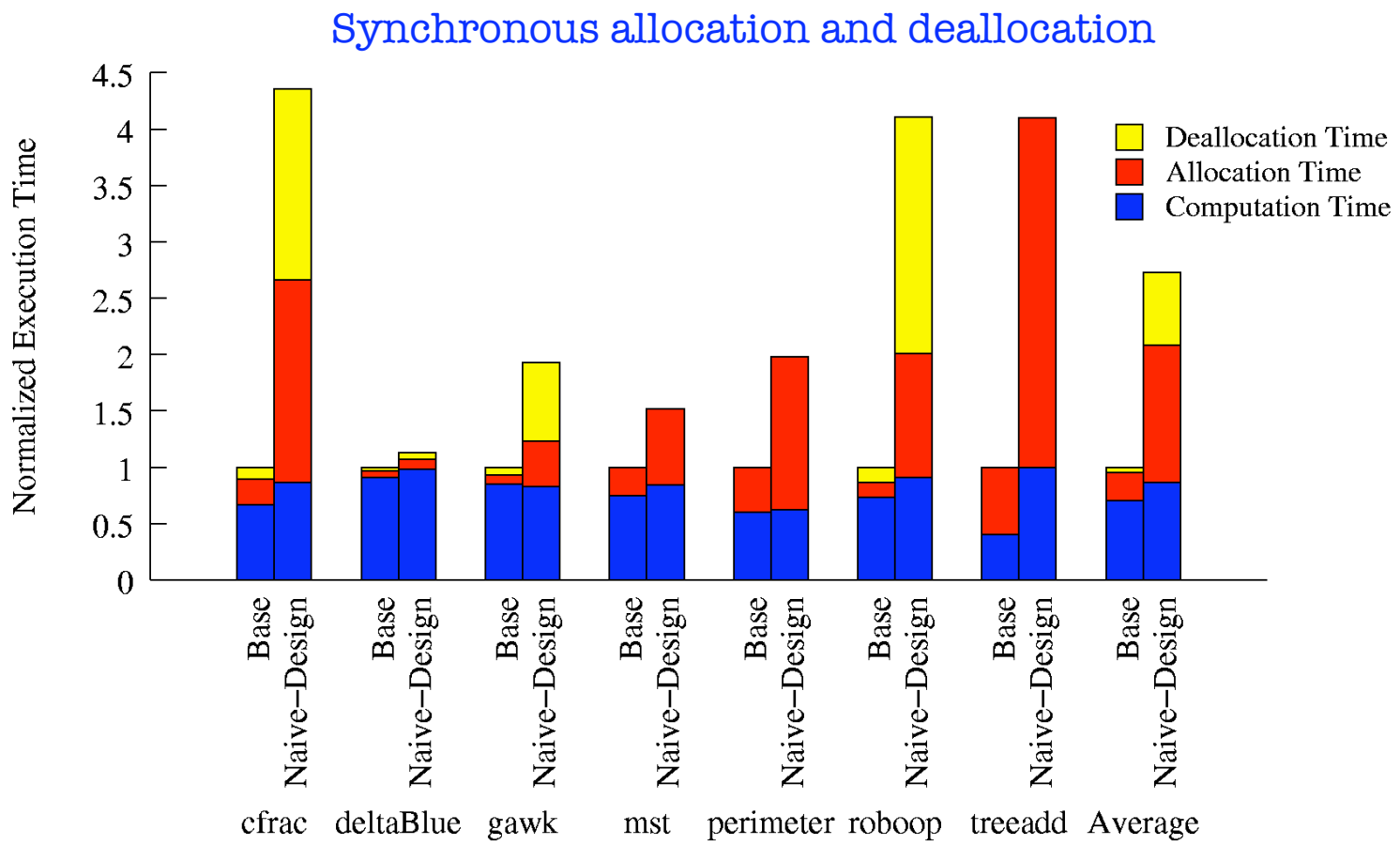
Benchmark	Language	Input
cfrac	C	A 35 digit number
deltaBlue	C++	1000000
gawk	C	large.awk
mst	C	8192 nodes
Perimeter	C	13 levels
Roboop	C++	bench
Treeadd	C	27 levels

Machine: 2.4GHz Intel Core 2 Quad machine,
Linux Version 2.6.18
Compiler : gcc –O3 optimization level

Profiler: Oprofile for hardware performance counters

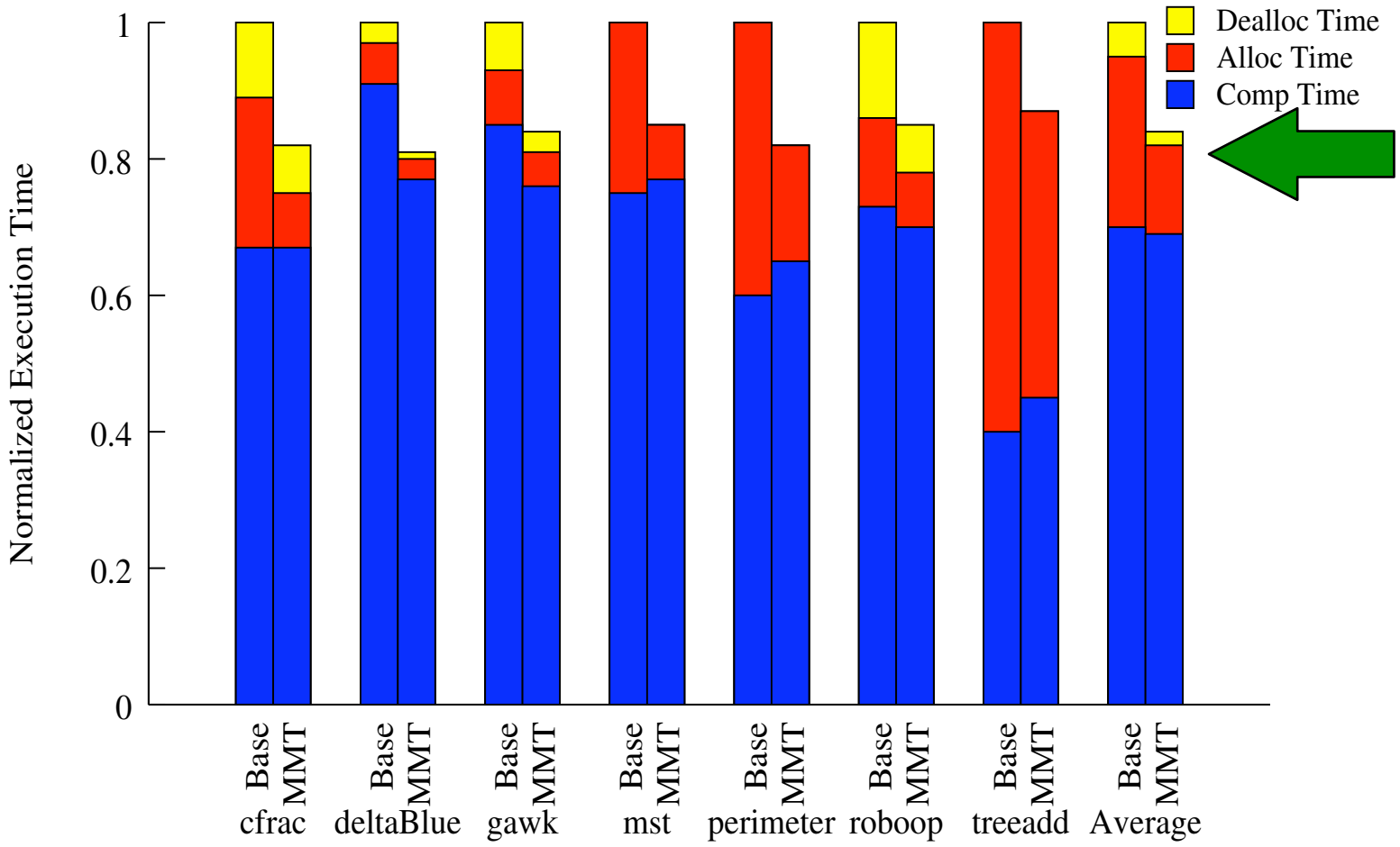
Offloaded allocator: Doug Lea's, and phkMalloc

Basic MMT Performance



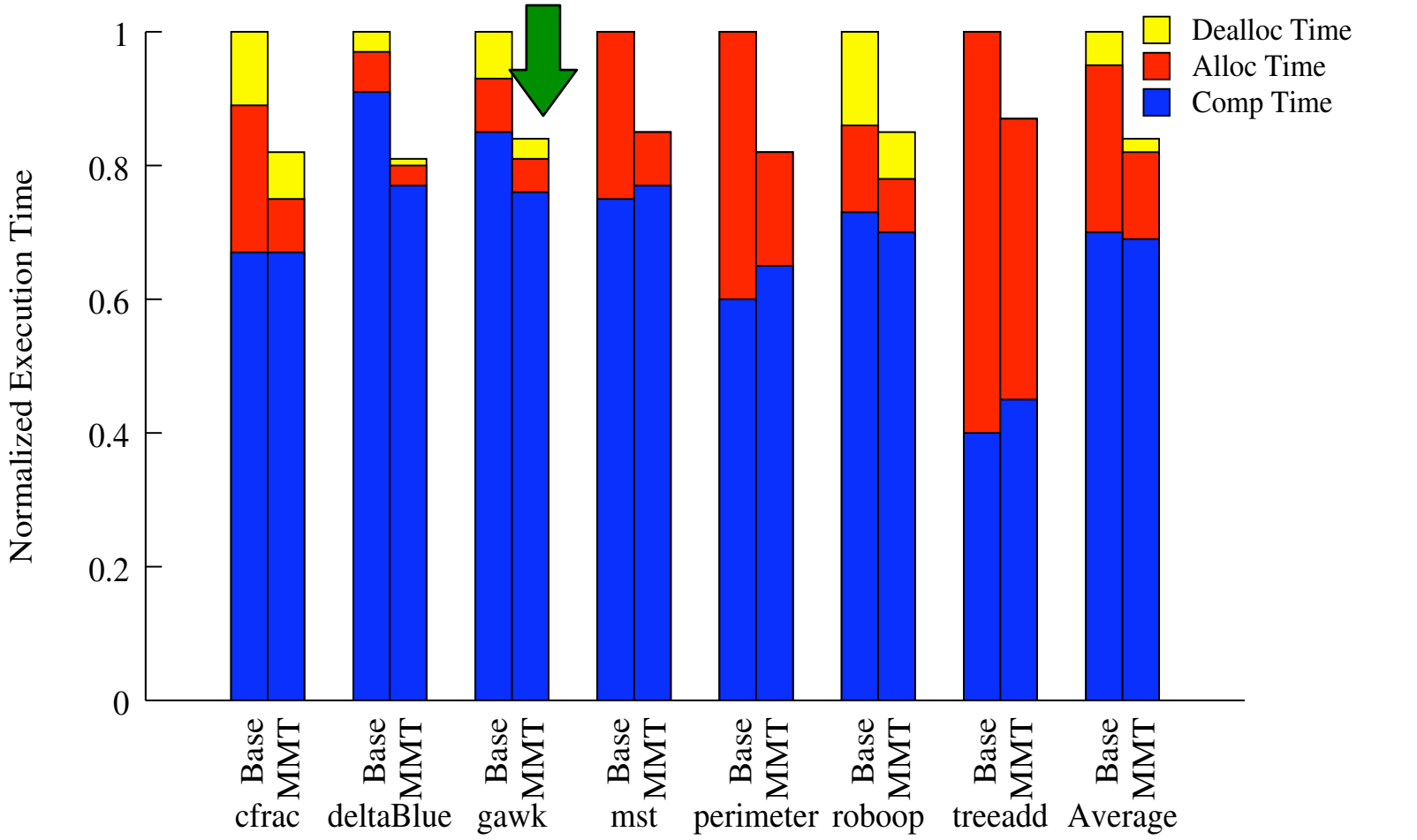
Huge slowdown due to the cost of communication and synchronization

MMT Performance



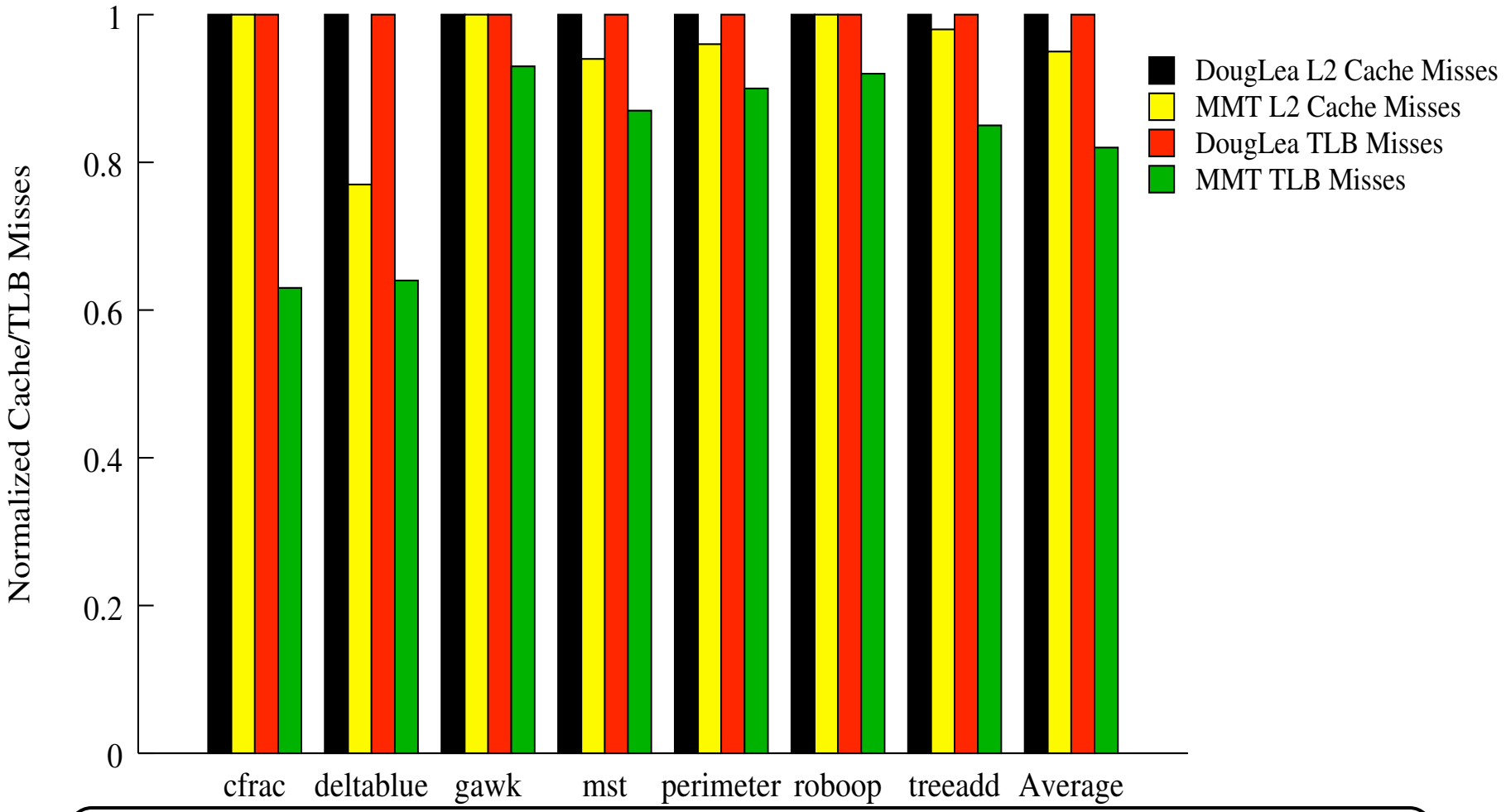
Allocation and Deallocation Time reduced by 50-70%
Average Speed Up : 1.19x

MMT Performance



Allocation and Deallocation Time reduced by 50-70%
Average Speed Up : 1.19x

MMT Decoupling Effect



Different code and cache behavior in allocation library and main program
Decoupling: Improved locality and Reduced interference

Outline

- Motivation
- Memory Management Thread : MMT Approach
- MMT Design and Implementation
- Evaluation
- Conclusion

Conclusion

Novel Memory Management Thread (MMT) Approach for Speeding up Heap Intensive Sequential Applications on Multi-cores

Conclusion

Novel Memory Management Thread (MMT) Approach for Speeding up Heap Intensive Sequential Applications on Multi-cores

✓Speculative Memory Management

✓Bulk Memory Management

✓Positive Decoupling Effects and Synergetic Interactions

✓Agnostic to underlying Memory Management Library Details

✓Useful for high overhead tasks e.g. Security Checking

✓Potential for offloading other high overhead tasks and providing Hardware/compiler support for more performance

More in Paper

- Minimalistic Synchronization
- Contention Resistant Data Structures and Mechanism
- Lessons learned with Task Partitioning
- phkMalloc MMT Performance
- Security Overhead Reduction with MMT Approach

Thanks !

Conclusion

Novel Memory Management Thread (MMT) Approach for Speeding up Heap Intensive Sequential Applications on Multi-cores

- ✓Speculative Memory Management
- ✓Bulk Memory Management
- ✓Positive Decoupling Effects and Synergetic Interactions
- ✓Agnostic to underlying Memory Management Library Details
- ✓Useful for high overhead tasks e.g. Security Checking
- ✓Potential for offloading other high overhead tasks and providing Hardware/compiler support for more performance