Improving Python's Memory Allocator

Evan Jones

ejones@uwaterloo.ca

http://evanjones.ca/

Outline

- The Problem
- Inner workings of the memory allocator
- A Solution
- The Future?

Finding the Problem

- Application with “bursty” memory usage:
  - Large computation (10 - 30 minutes)
  - Many, many short simulations (2 - 3 hours)
- Result: 2 GB of memory occupied by Python for hours
  - Simulation performance suffered
  - Shared system: other users not able to use it

The Cause

Python never releases memory

- Good if Python is the only process
  - Very low overhead
- Bad if need to co-operate

Workarounds

Do not allocate memory in a long running Python process

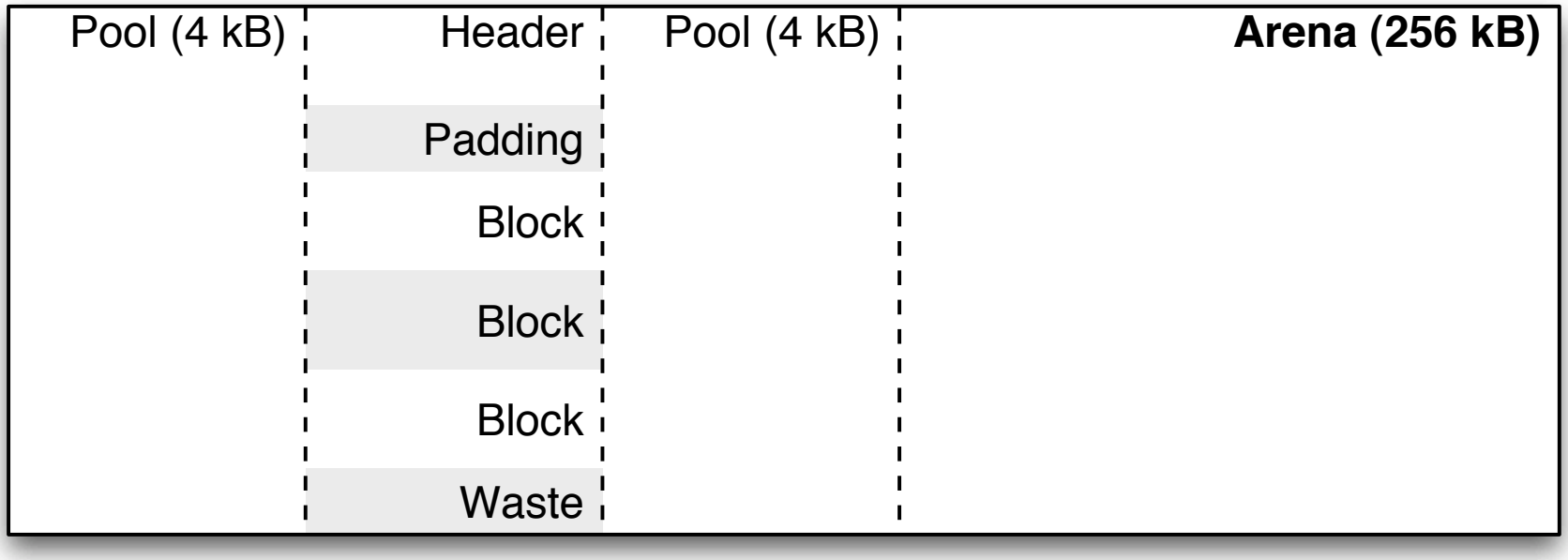
- Perform one-time computations via `fork()`
- Store temporary results in the file system

This shows that Python's memory management is not solving the whole problem

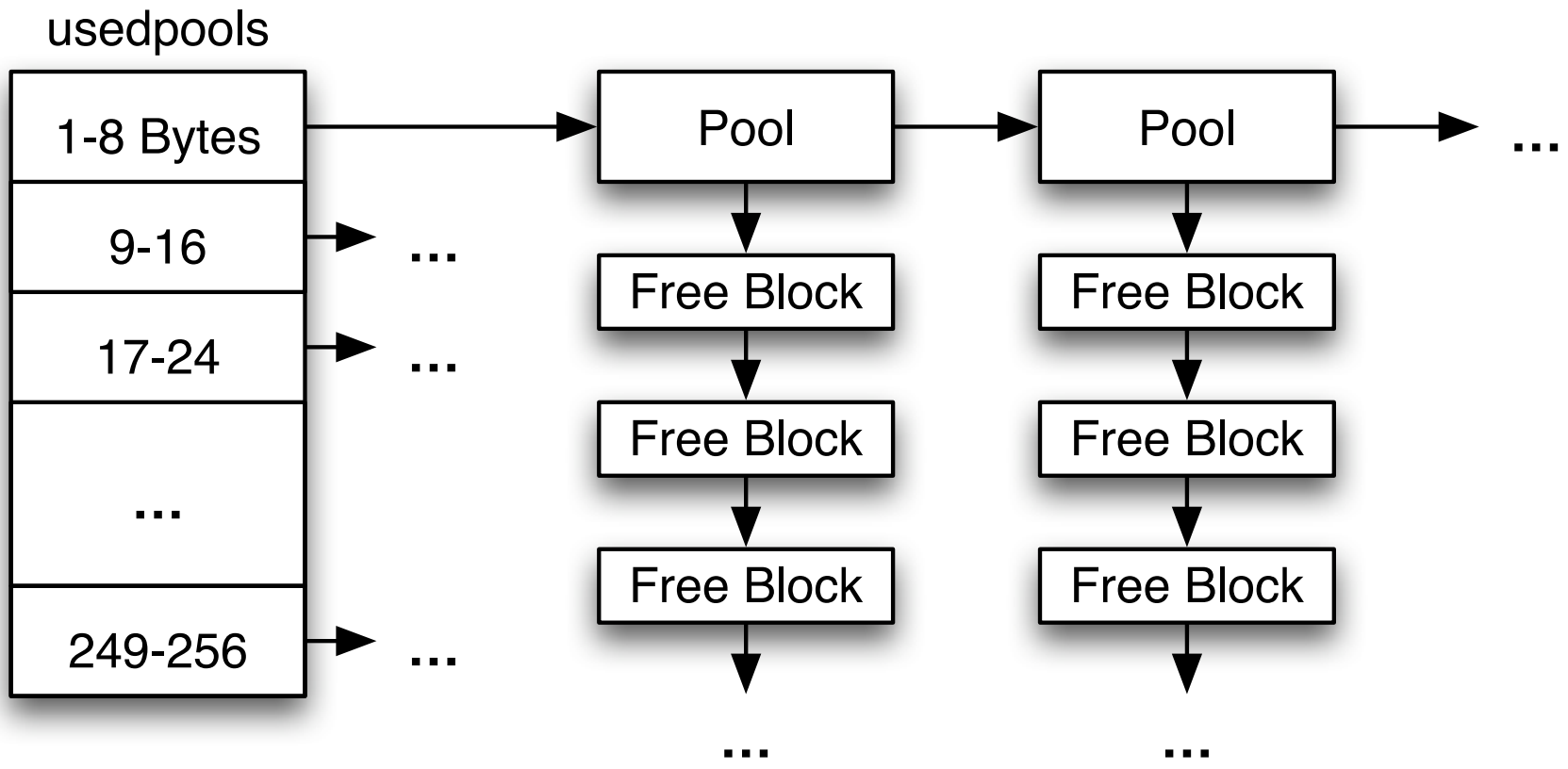
Memory Allocator Gory Details

- Pymalloc: default in 2.3
- Allocates memory in 256kB chunks (arenas)
- Used for objects ≤ 256 bytes in size

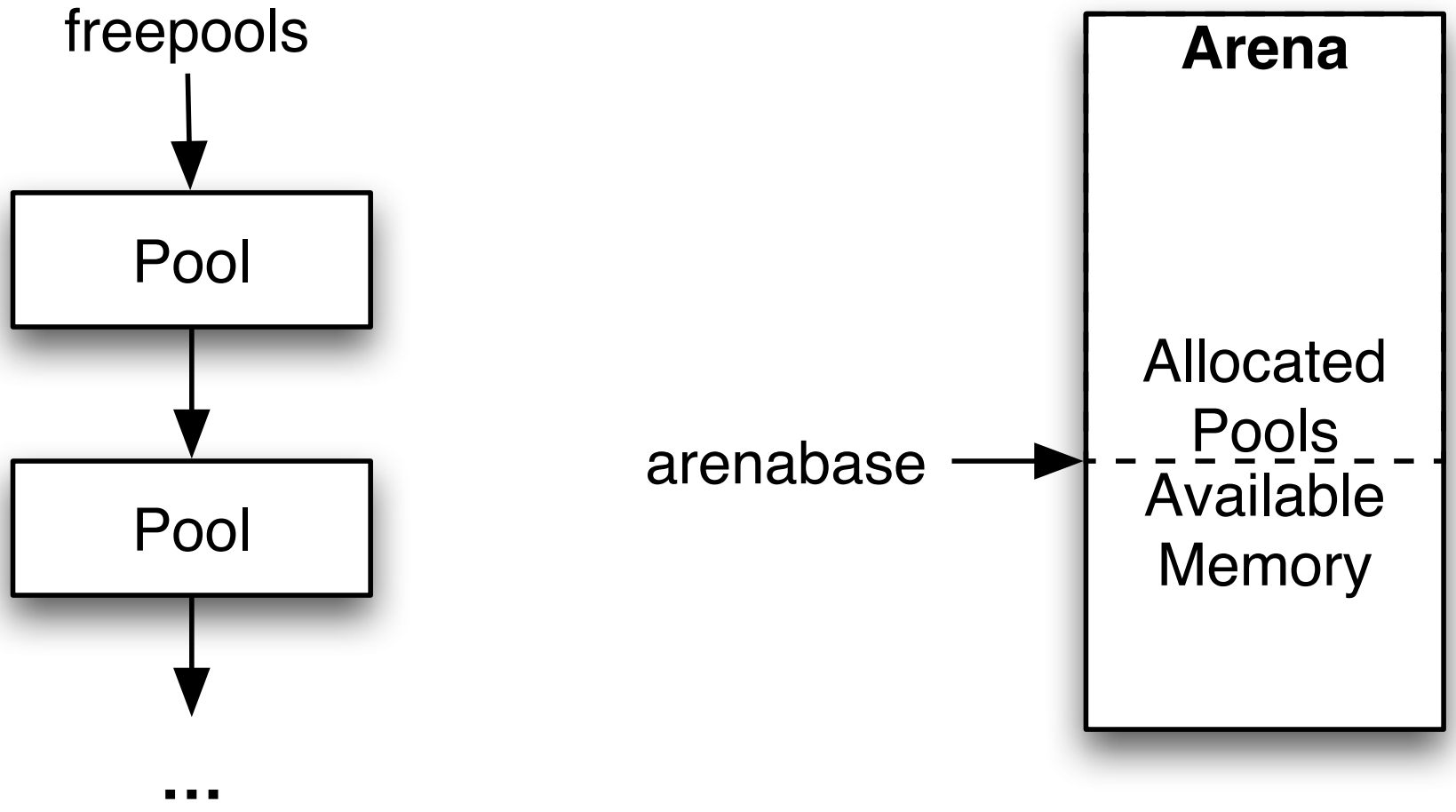
Memory Layout



Allocating Memory



Allocating More Pools



Freeing Memory

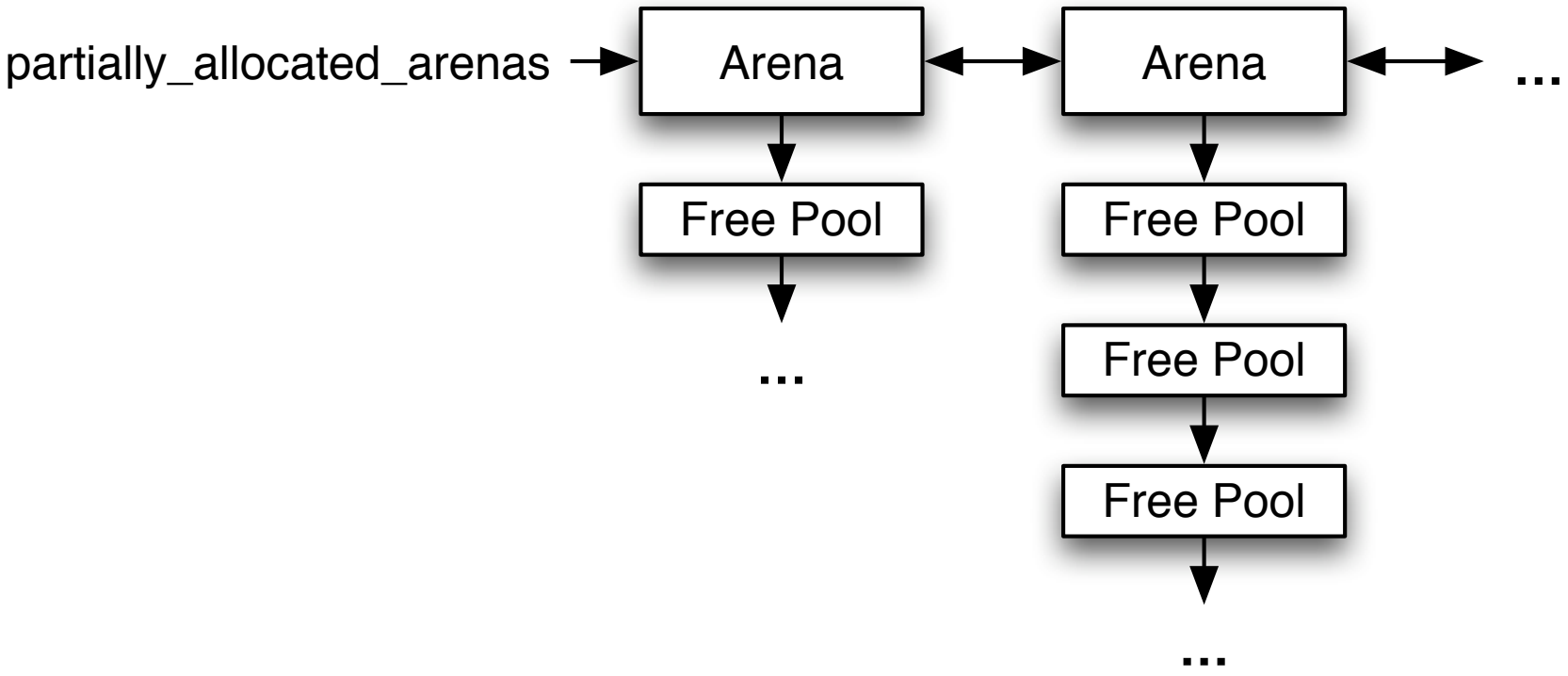
- Add block to pool's free list
- If there were no other free blocks:
  - Add pool to usedpools
- If the pool is completely available:
  - Remove from usedpools, add to freepools

Solving the Problem

- Need to collect pools from each arena
- When arena is unused, free it

- Need data structure to track pools
  - Maintain more information about each arena

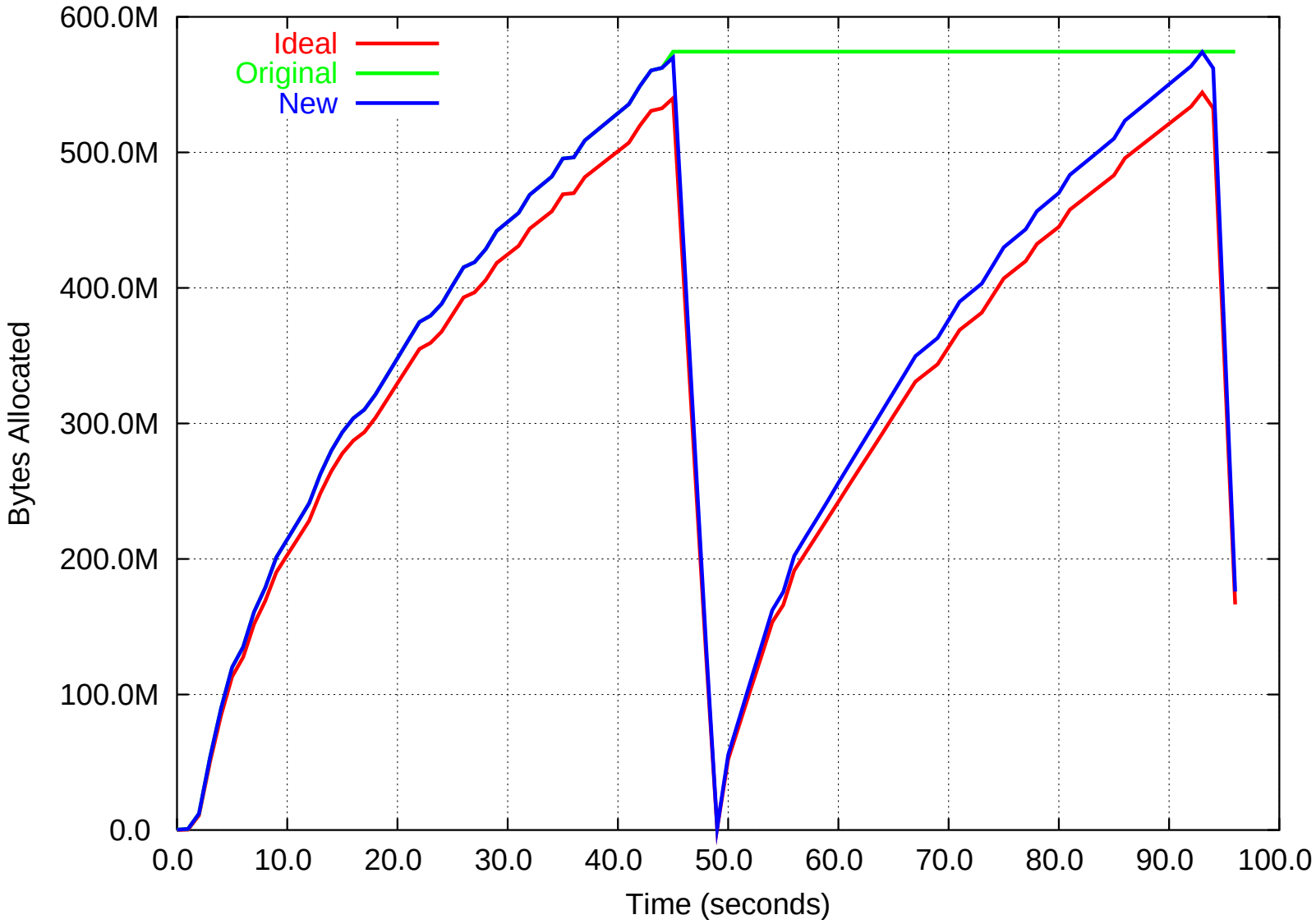
Used Arenas Data Structure



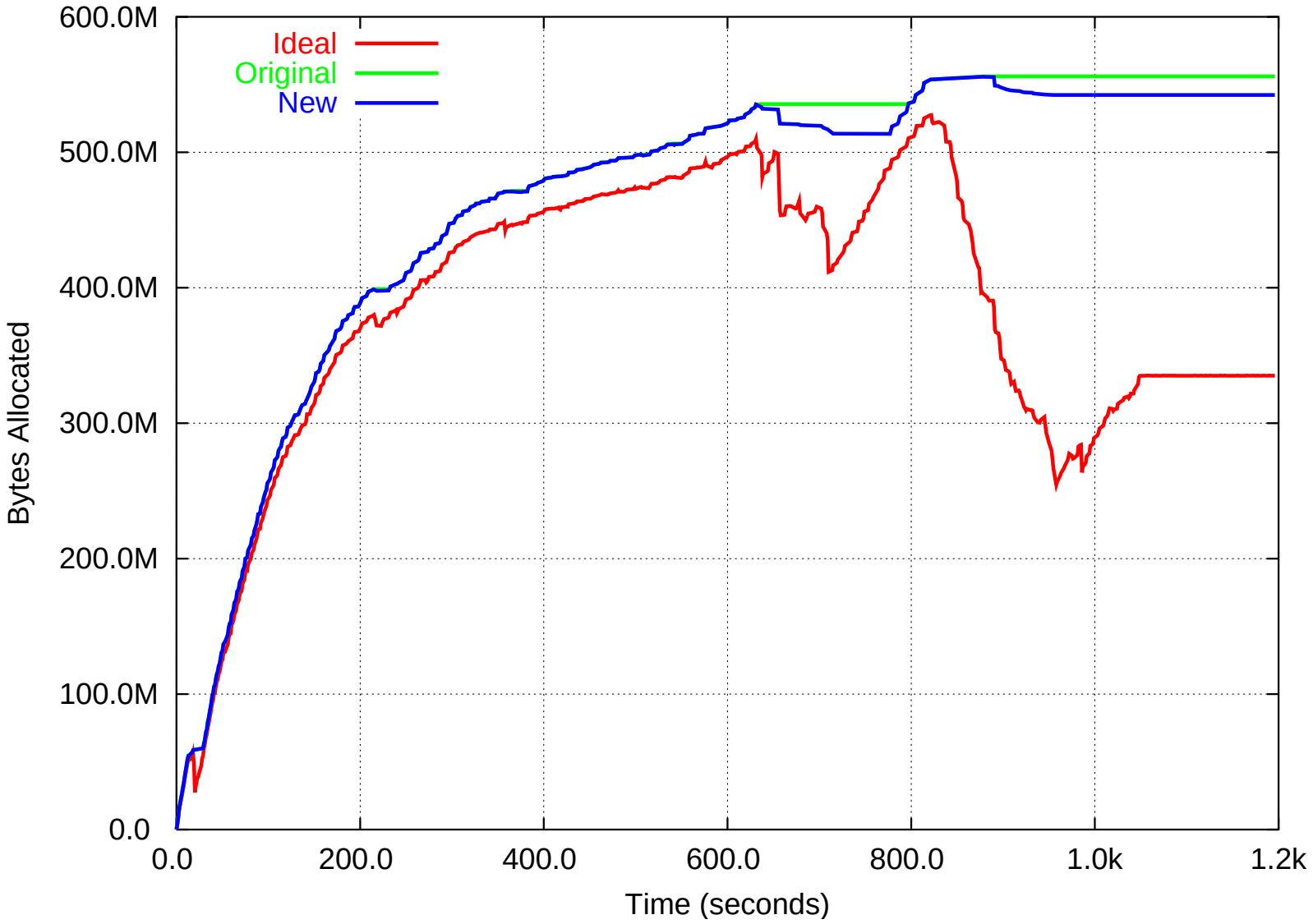
Results

- Python now releases memory
- Small overhead when pools freed/allocated
- Extra overhead if cyclically allocating/deallocating many objects

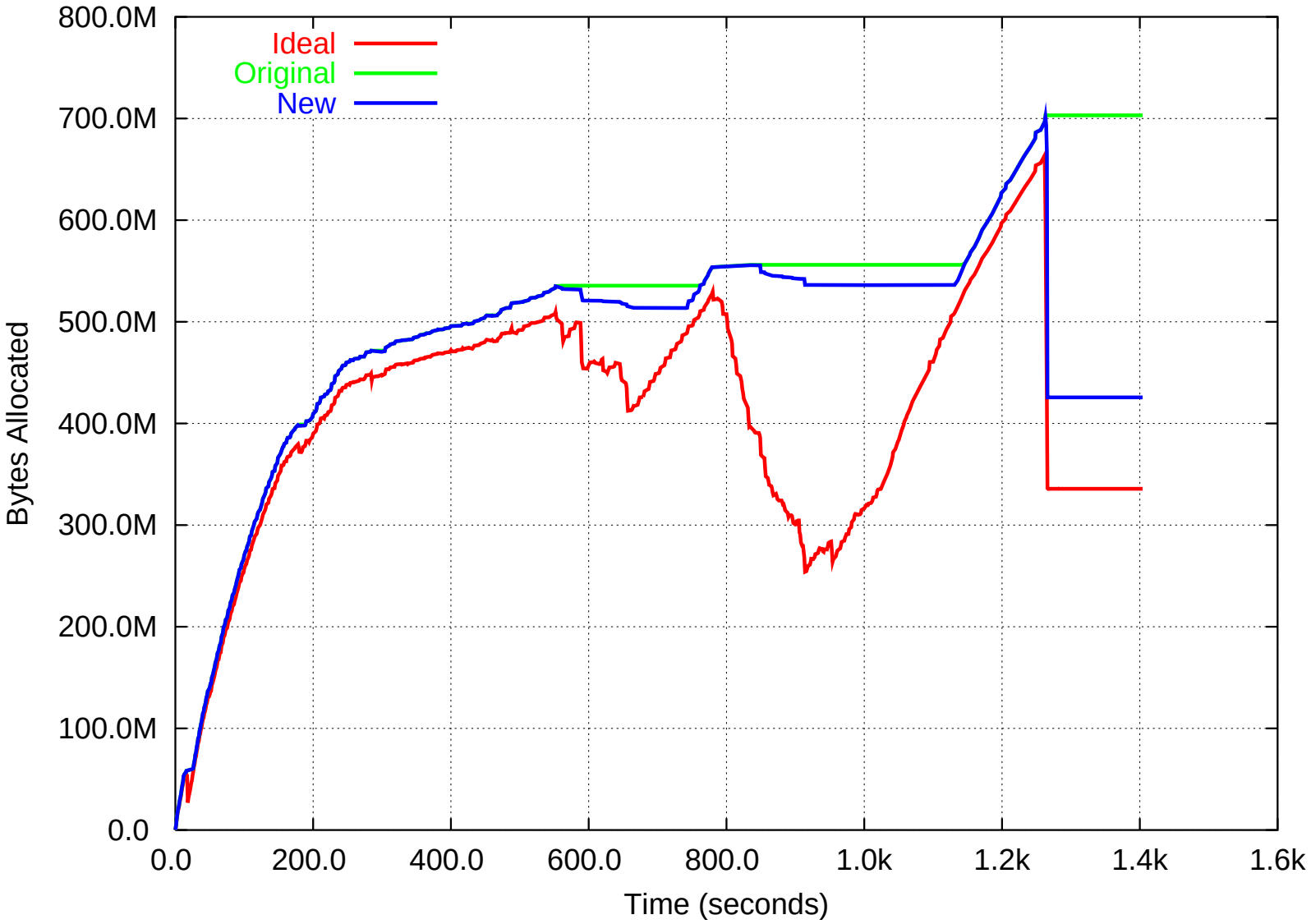
Example: Cyclic Allocation



Example: Real Application



Example: With Reallocation



Current Status

- Patch in the patch tracker
- Works on 2.3, 2.4 and 2.5

Future work:

- Free lists for integers, floats, lists, dicts

Questions?

Evan Jones

ejones@uwaterloo.ca

http://evanjones.ca/