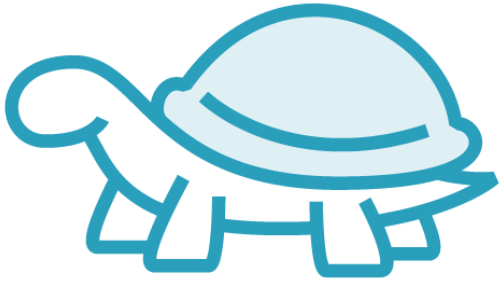
Understanding Object Allocation Rate Problems

Richard Warburton

@richardwarburto www.monotonic.co.uk

Two Types of Problems



Latency/Responsiveness

Too much allocation causes long GC pauses

Throughput/Efficiency

Spending too much time allocating uses up valuable CPU time


CARPOOL
HS 3
OR MORE
PERSONS
PER
VEHICLE

Business Defined Targets

Latency	Throughput
A request takes at most Xms.	Number of requests/second
In practice latency rises under load	How many requests/second does my application have to support at peak times?
Y% of requests take at most Xms at Zrequests/second	
How fast to keep users happy?	

Module Outline

Improving Efficiency and Throughput

Improving Latency

Conclusions

Improving Efficiency and Throughput

"Allocating objects is basically instant and free of costs"

– **Common Developer Myth**

Two Main Costs

CPU Cache Locality

Allocating Objects reduces the effectiveness of your CPU's Cache

Time Spent Allocating

Actually allocating objects takes time in and of itself

Modern CPUs have multilayered Caches

Allocating objects writes to memory, washing the Cache

Effect ameliorated by LRU Caches

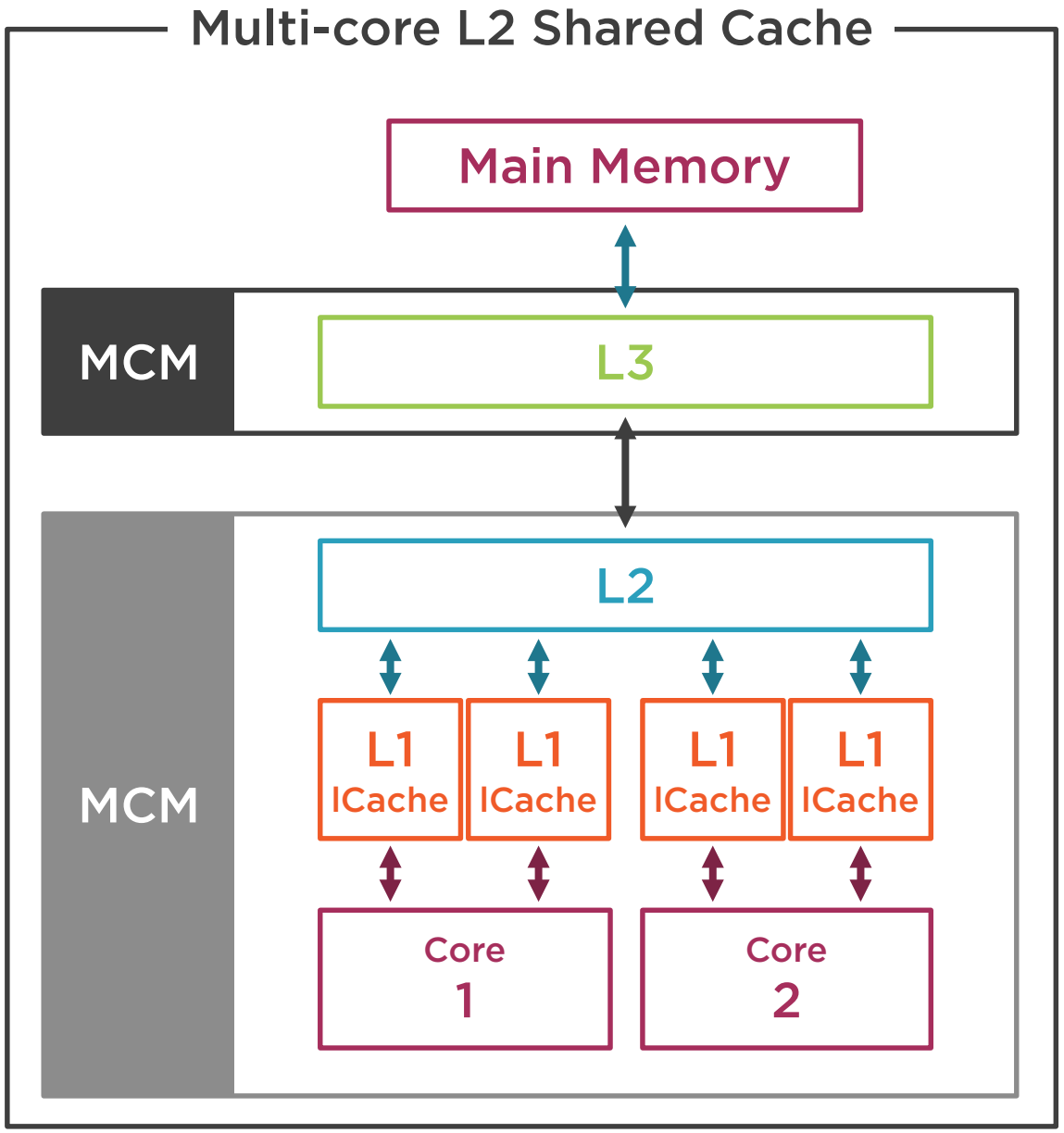


Image Source: Ferruccio Zulian – Milan, Italy

Allocation Costs

Allocation in GC'd systems is very fast

But it's not free!

Different Collectors have different costs

- Eg: Parallel Collector faster than G1

If you allocate a lot these costs add up

Execution Profilers should tell you about this ...

But They Don't!

Sampling happens at "safepoints"

Application never sampled during allocation

Profilers can slow down Application

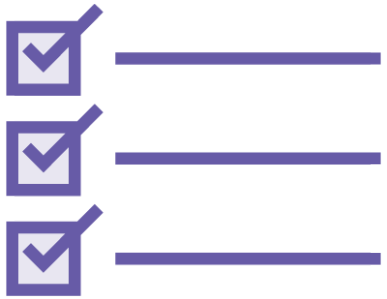
Actually makes allocation costs look less

Execution Profilers tell you about code

Don't even have a way of referencing allocation rates

Memory Profilers Can Help

Measure
Use Memory Profiler to identify allocation hotspots



Correlate
See if they happen in execution Hotspots

Optimize
Reduce memory allocation within hotspots

Summary

There's a cost to memory allocation

Execution Profilers won't highlight it

Memory Profilers can do

Improving Latency

Generational Hypothesis

Most Objects Die Young

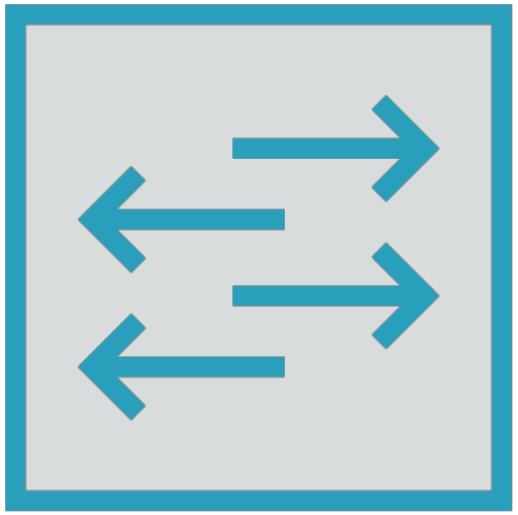
Based upon empirical research of object lifetimes

Split Memory into generations

Collect younger generations more frequently

GC Pauses don’t directly correlate to allocation rates

Throughput Collector



Young Generation

- Copies live objects to next generation.
- costs scale with size of live dataset

Old Generation

- Compacts objects down to contiguous memory
- Costs scale with size of heap

GC Frequency based upon generational region being full

G1 Collector

Splits heap up into many Regions

Young Gen

- All collected every pause

Old Gen

- Split collection up over different Young Gen Pauses

GC can’t necessarily keep up with too much allocation

Object Allocation Implications

Fill Up Young Gen
More frequent GC Pauses

Premature Promotion
Objects get promoted to old Gen

More Frequent Full GCs
Eventually Old Gen is also full, longer GC pauses

Conclusions

Module Summary

Memory Allocation can cause Performance Problems

Cost of allocation can reduce throughput

Cleanup of dead objects can cause GC Pauses

Three Big Problem Areas

Memory Leaks

Can run out of memory

Memory Overconsumption

Can also run out of memory

High Allocation Rates

Performance Problems

Three Common Tools

Object Histograms

Brief snapshot of what's using memory

Heap Dumps

In depth snapshot of what's using memory

Allocation Profiling

Continuous measurement of what's allocating memory