

# Performance of MOS

Narendran Sachindran  
Eliot Moss

July 2002



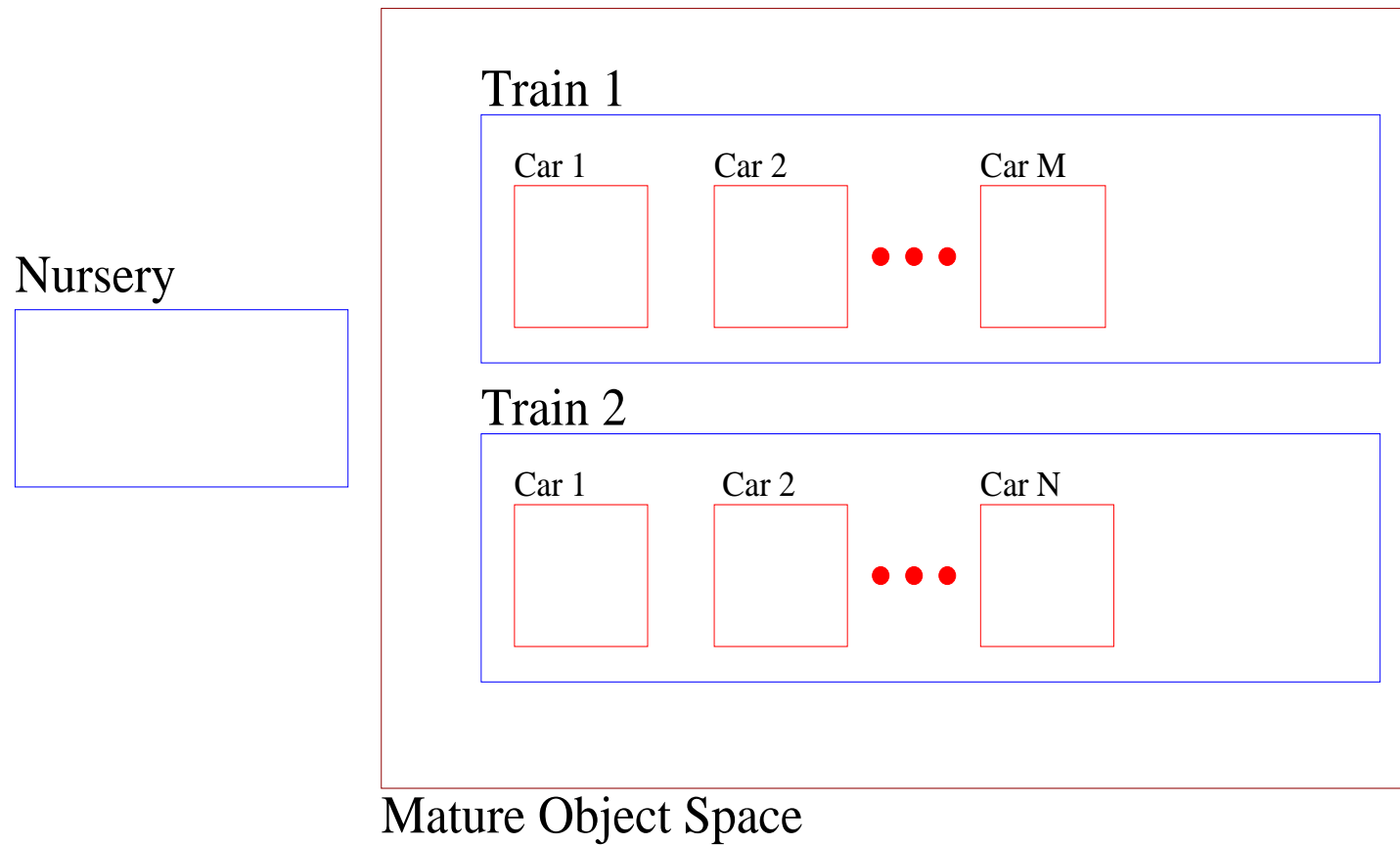
# Generational Collection

- **Heap is divided into areas called generations**
- **Generations segregate objects by age**
- **Objects are allocated into the youngest generation**
- **Objects are promoted into higher generations if they survive a collection**

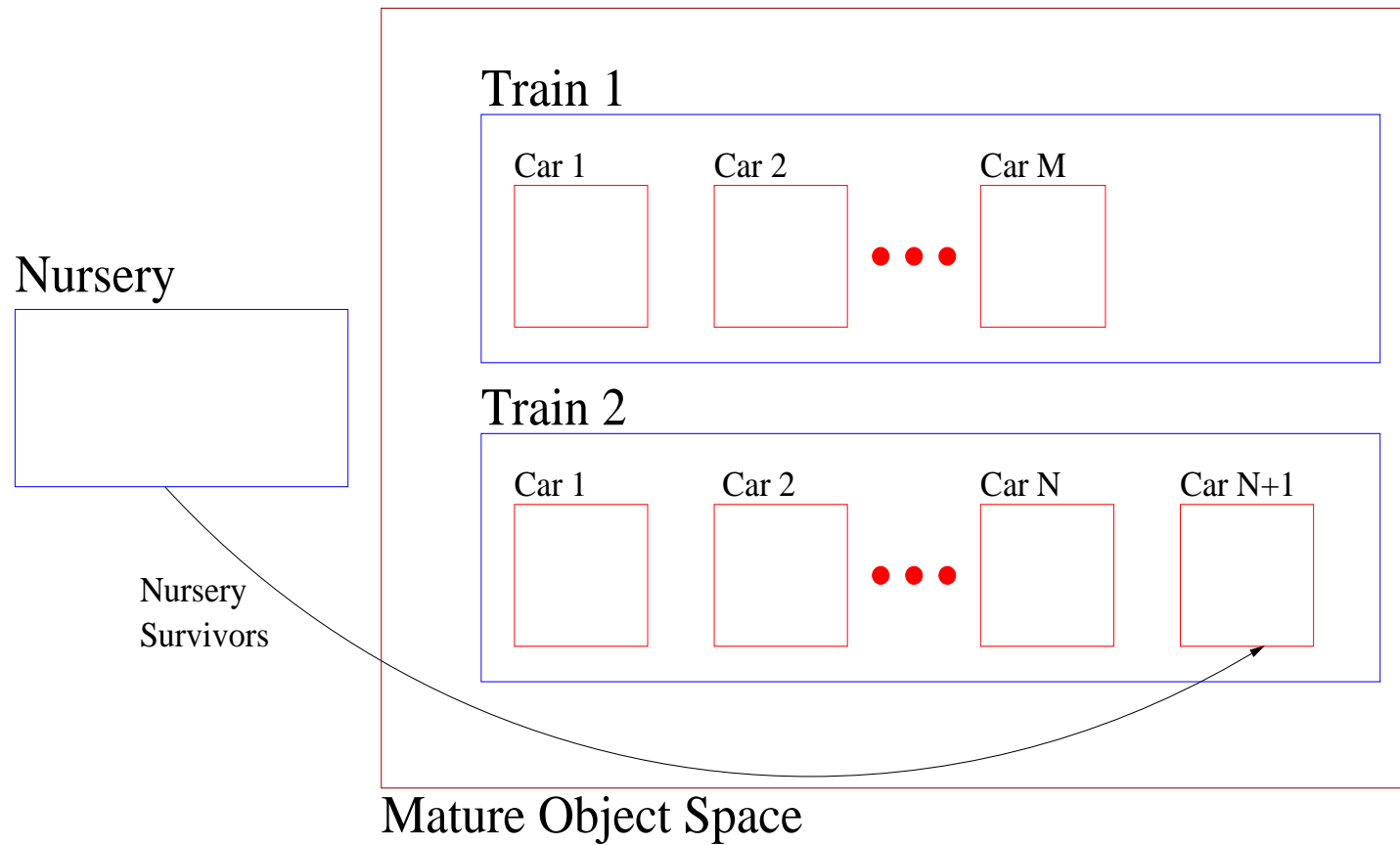
# Mature Object Space Collection

- **Extends generational scavenging to collect older generations non disruptively**
- **Promotes old objects out of the generational scheme into a mature object space**
- **Mature Object Space is divided into fixed size areas called cars, each of which has a remembered set**
- **One car is scavenged at each collection, thus bounding pause time**
- **Cars are linked together to form a train**

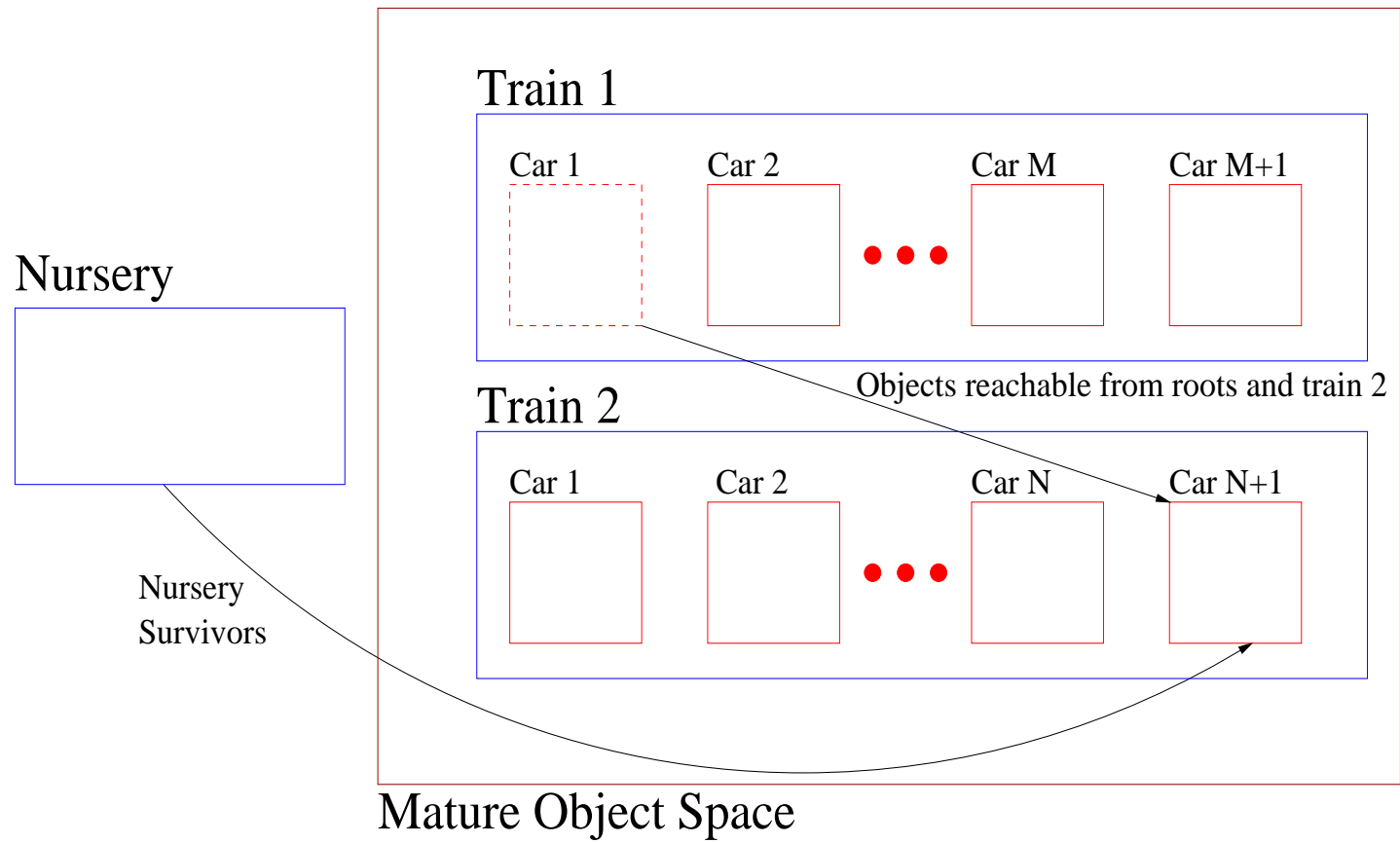
# Heap Layout



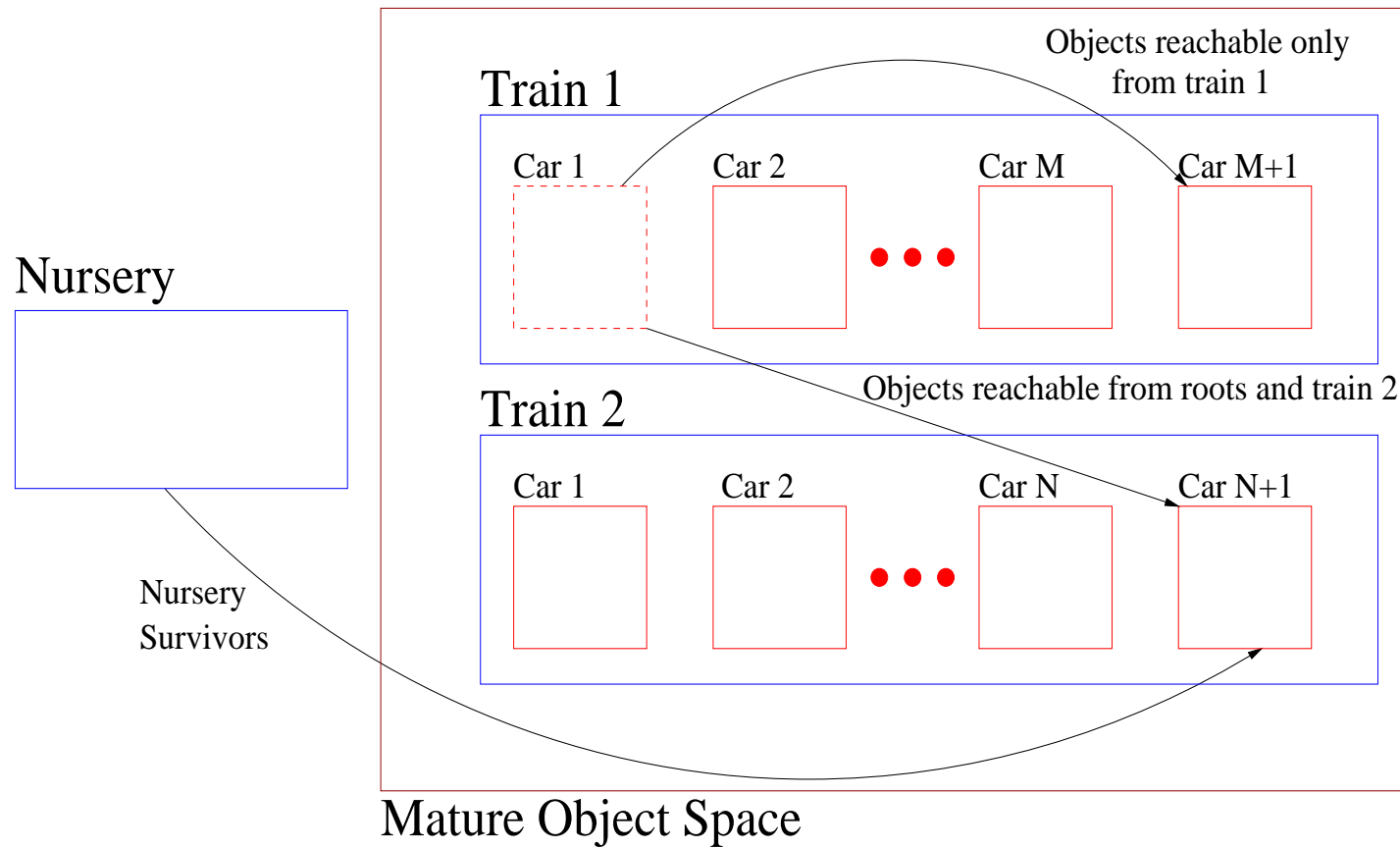
# MOS Collection



# MOS Collection



# MOS Collection



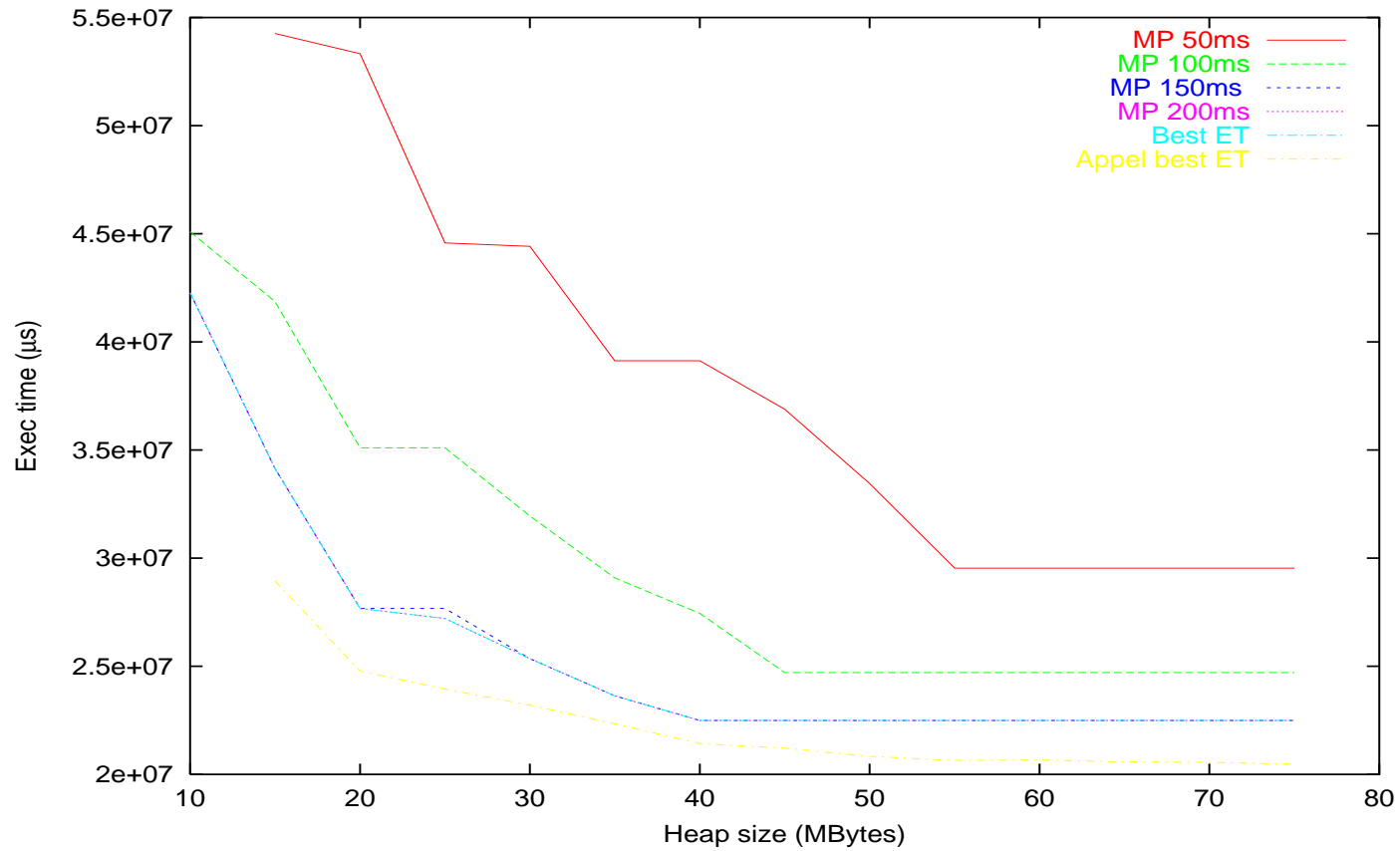
# Experimental Results

- **MOS collector was implemented in Jikes RVM/UMass GCTk**
- **All runs used the Fast build with all system classes in the boot image**
- **Experiments were run on a Linux PowerPC machine with a G4 processor and 384M memory**

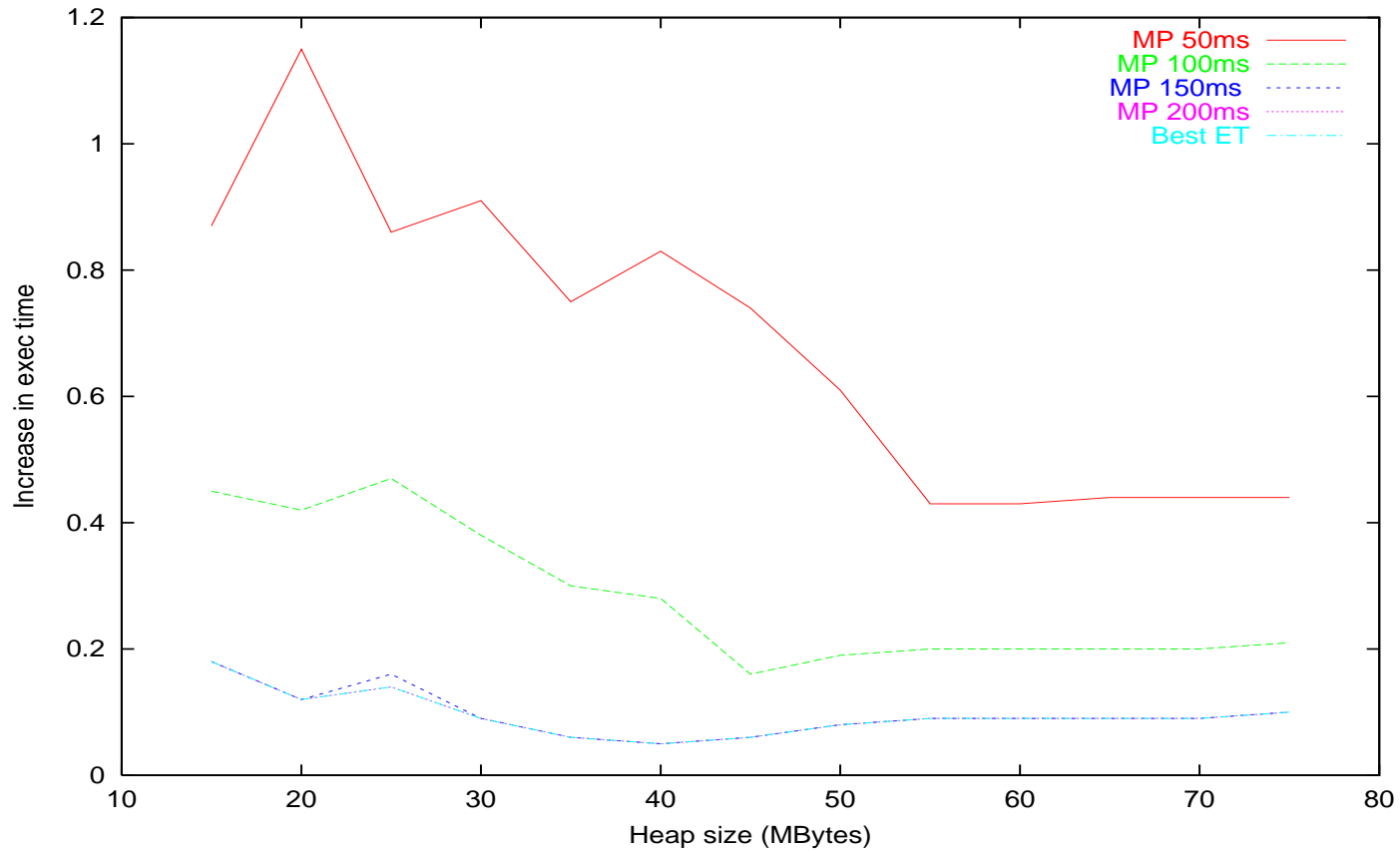
# Experimental Results

- **Parameters to the collector were - nursery size, number of cars to collect and train collection threshold**
- **Nursery sizes ranged from 256K-4M**
- **Train Collection Threshold is size to which train space needs to grow before train collection is enabled - ranged from 5M-25M**
- **Number of cars scavenged in each train collection ranged from 1-8 (128K-1M)**

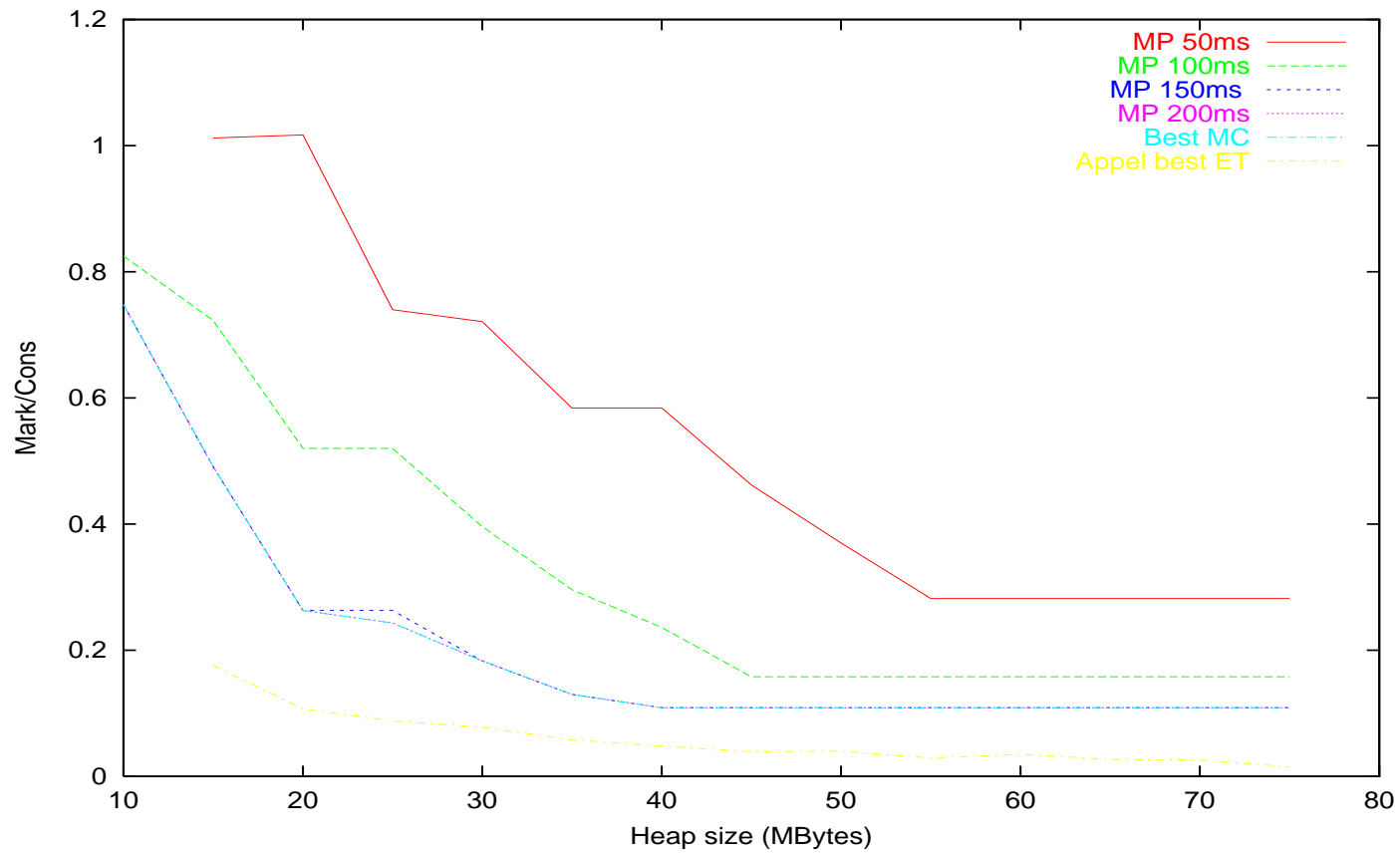
# jess Execution Times



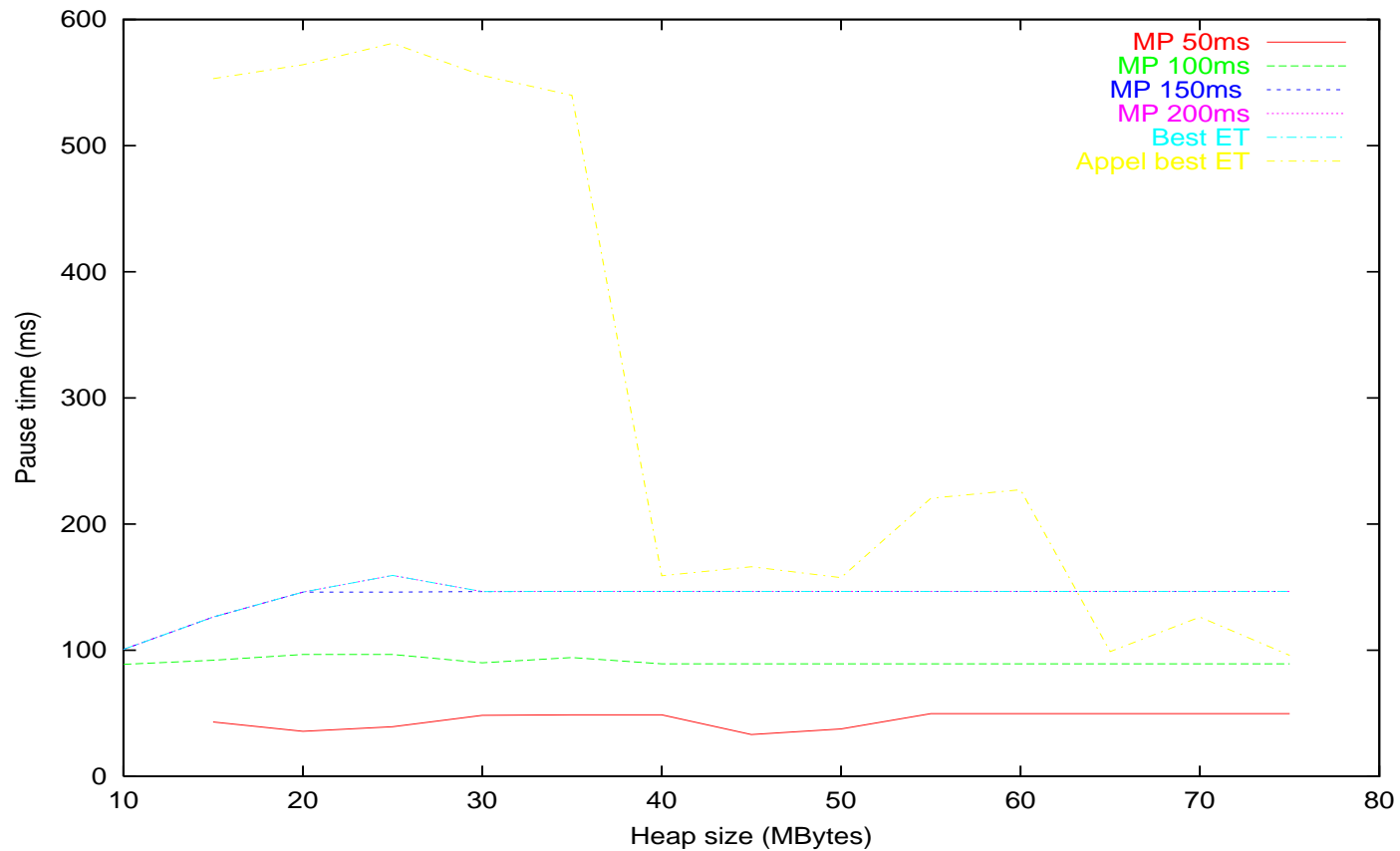
# jess Exec Time relative to Appel



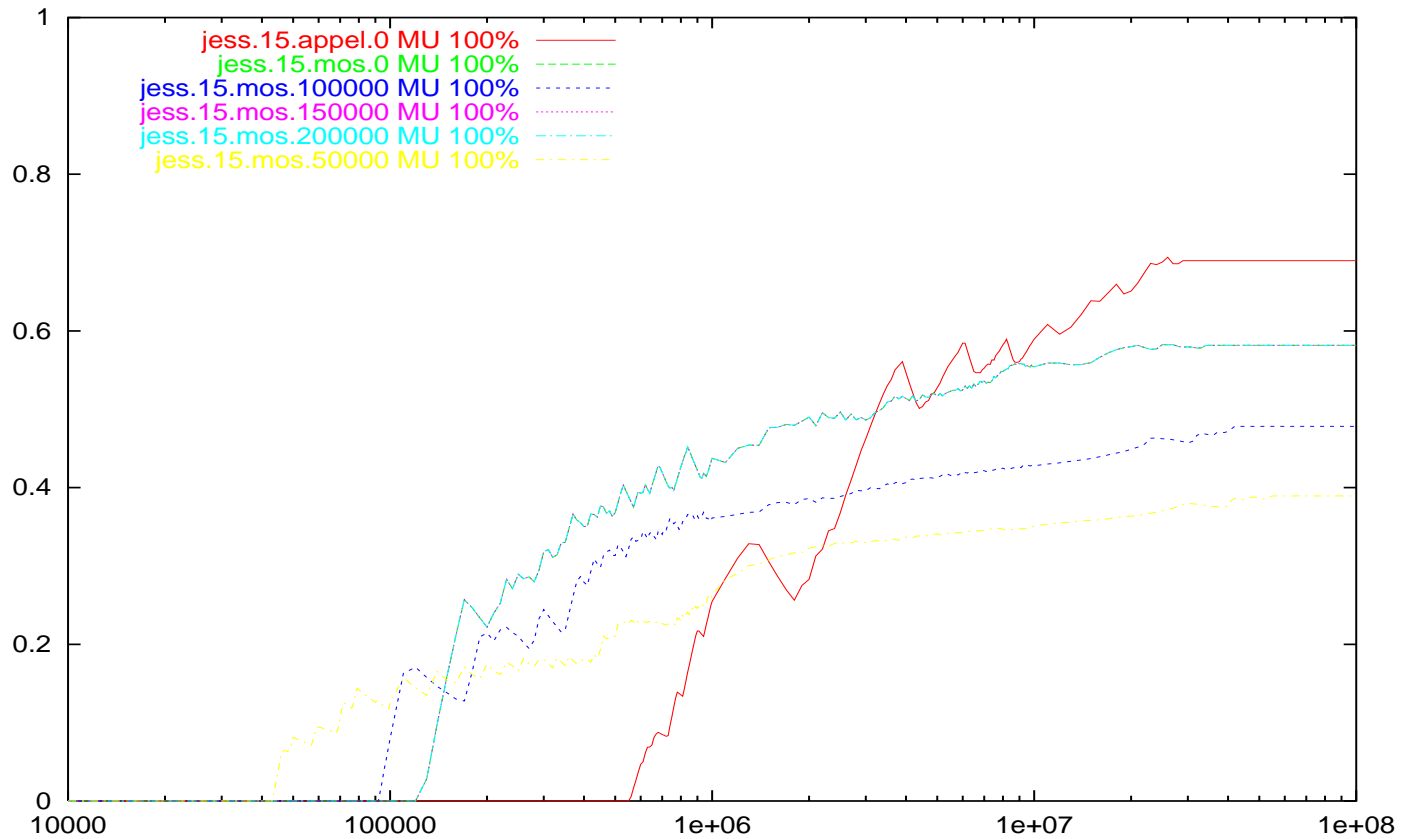
# jess Mark/Cons



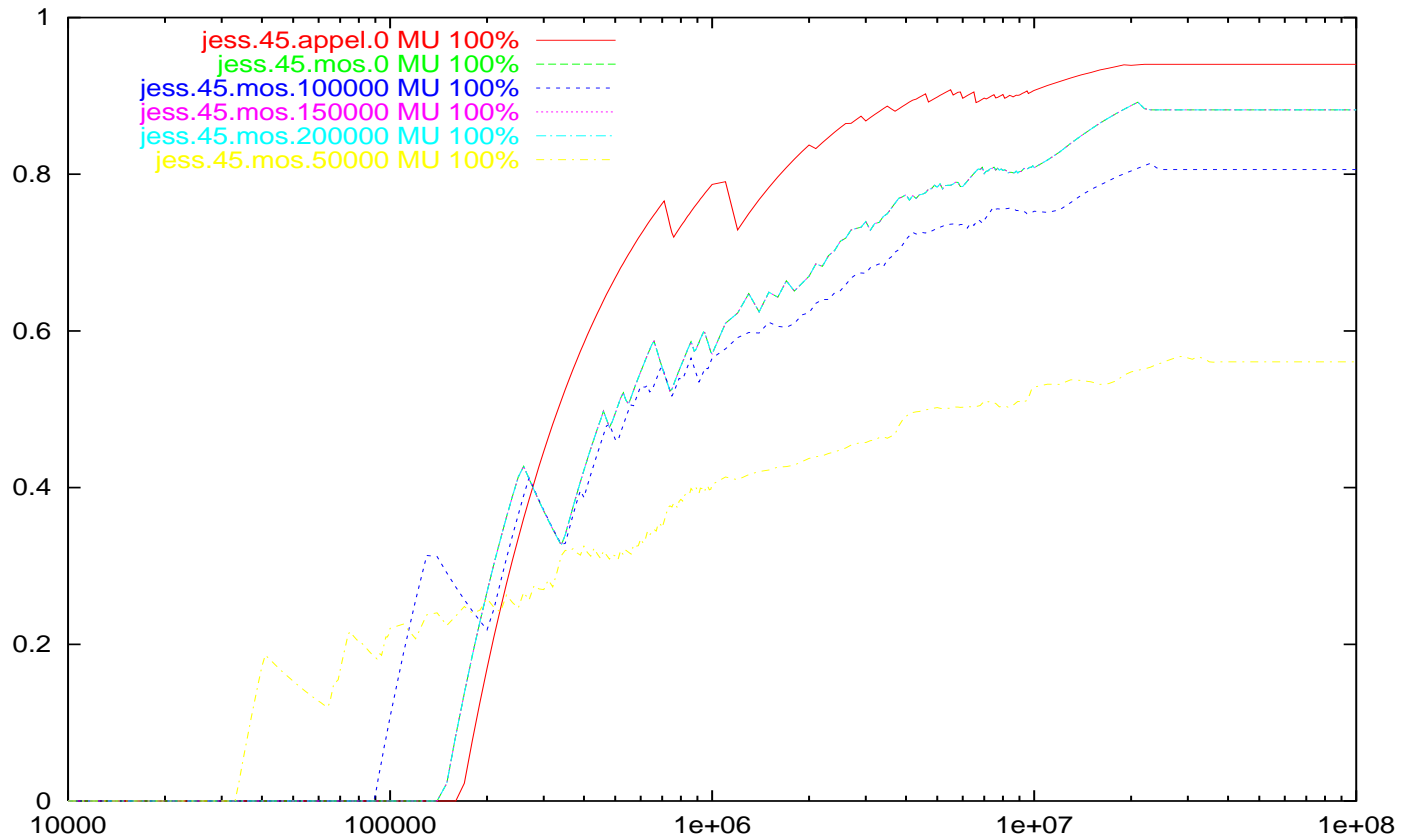
# jess GC Pause Times



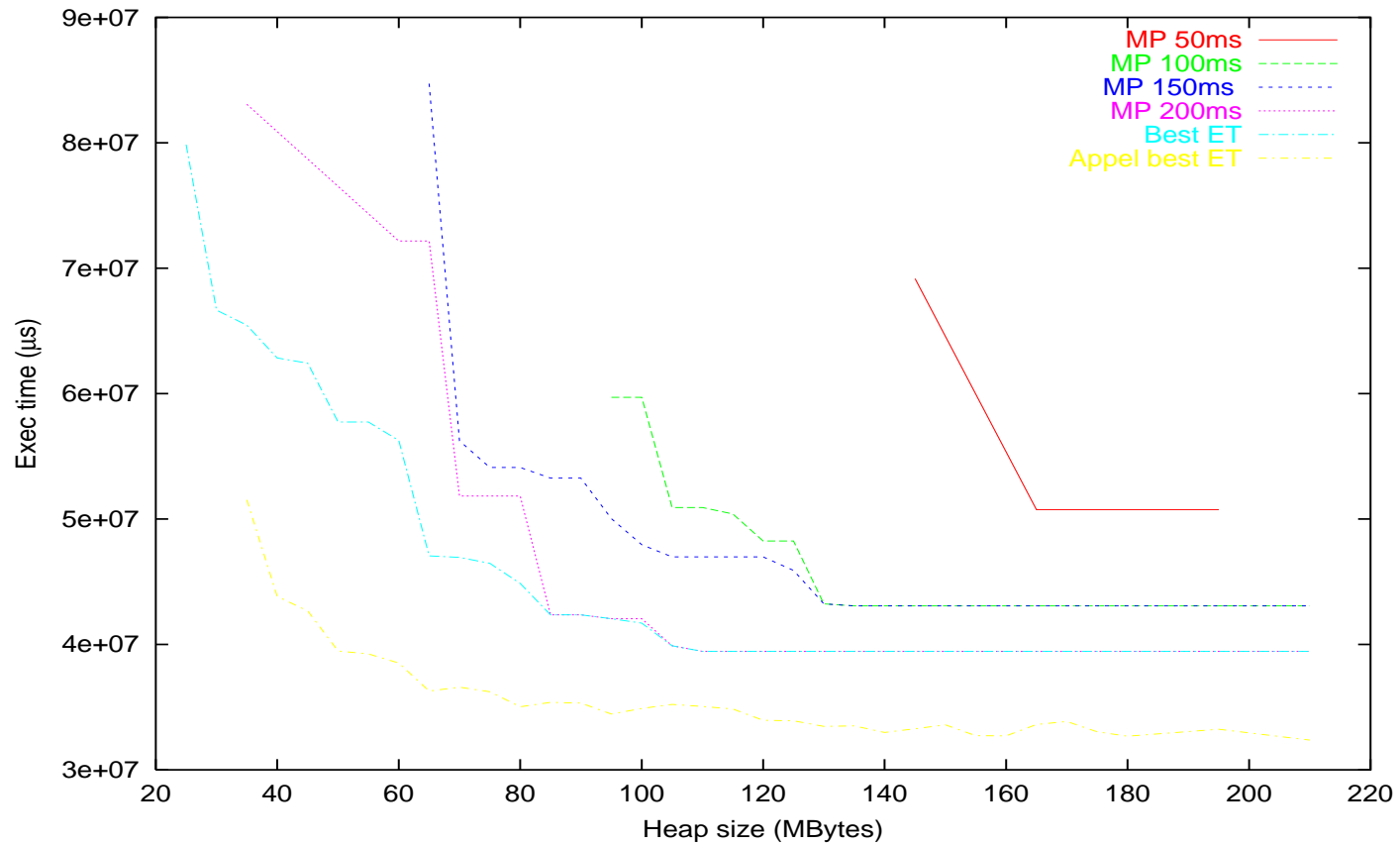
# jess 15M Mutator Utilization



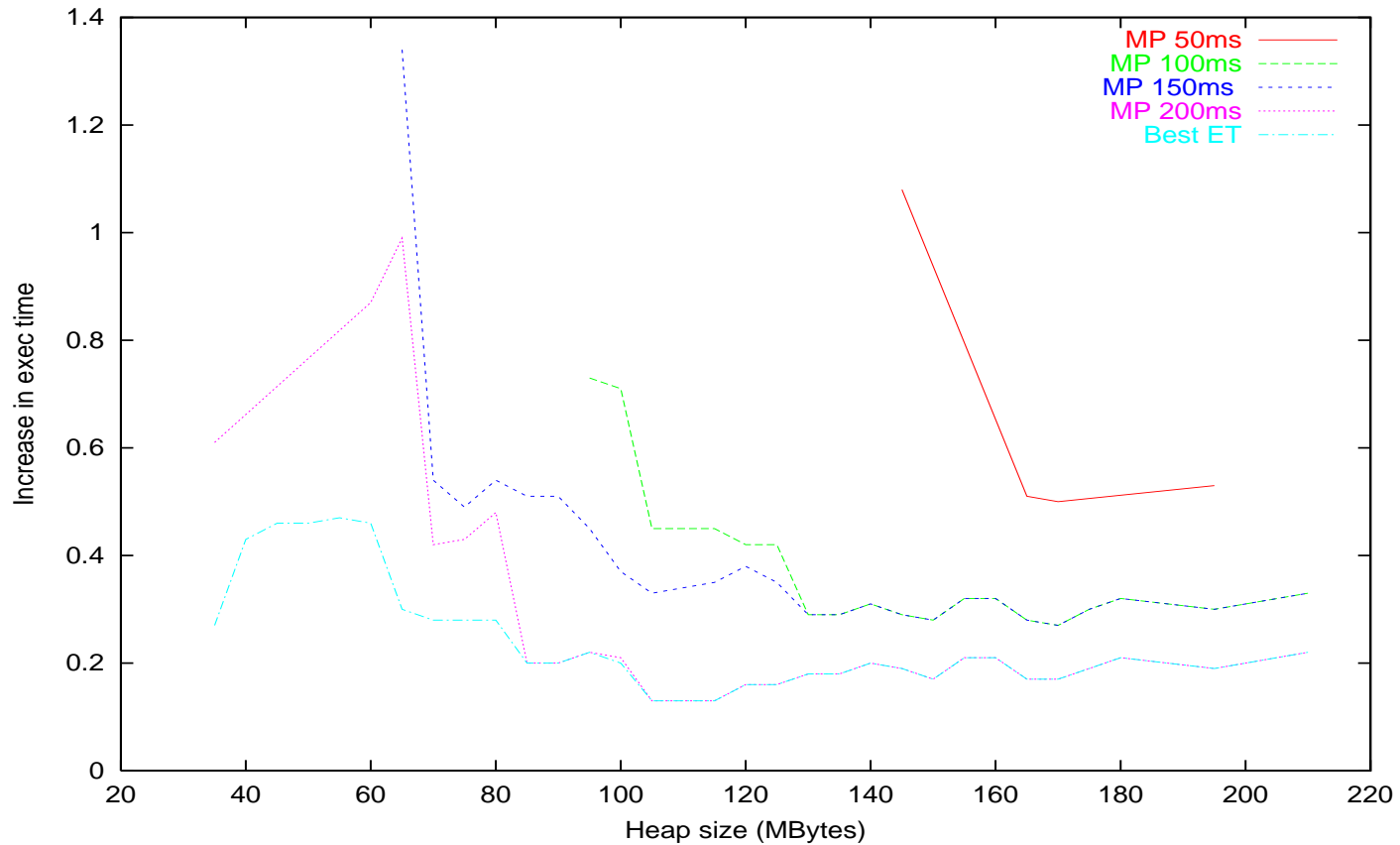
# jess 45M Mutator Utilization



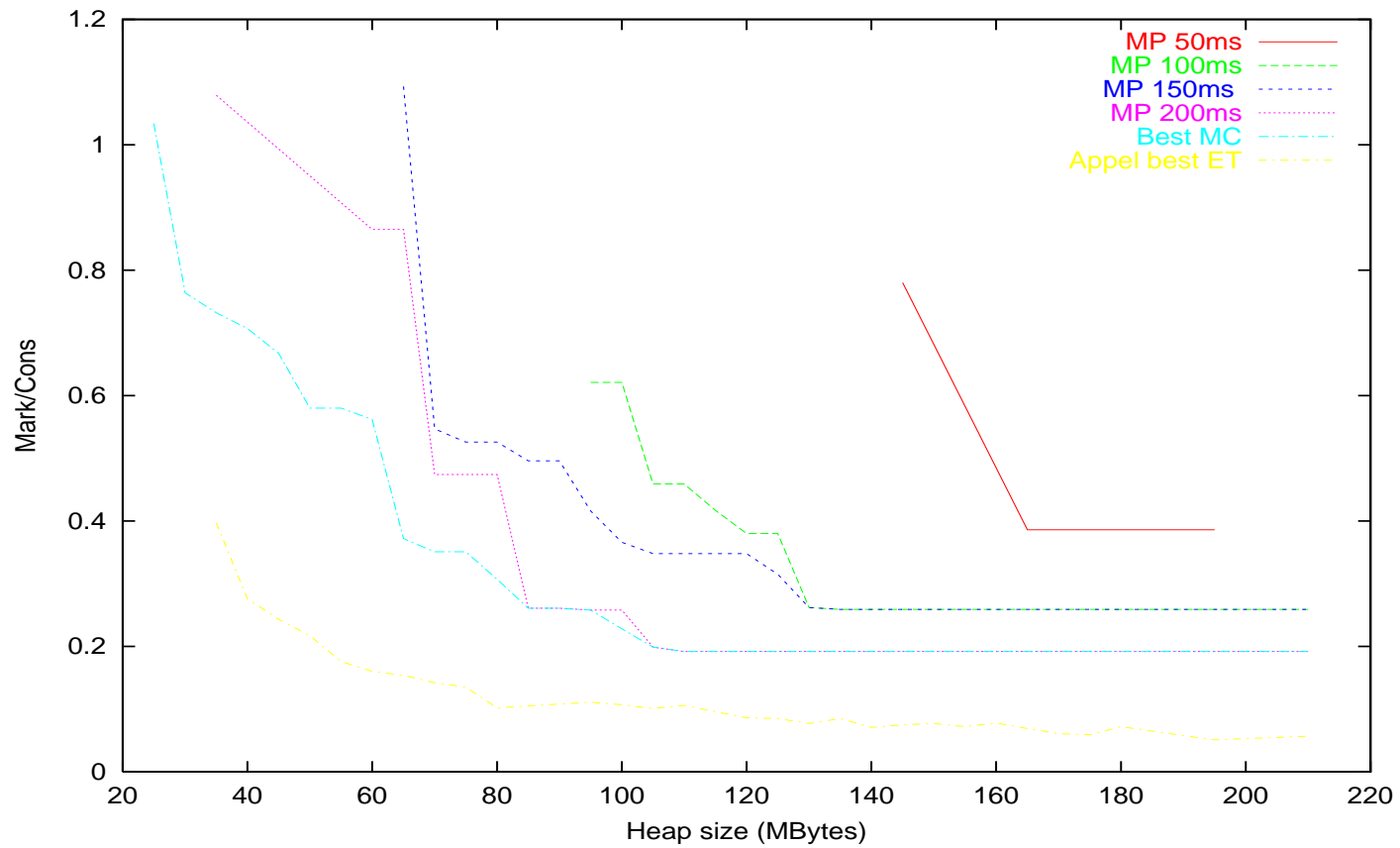
# javac Execution Times



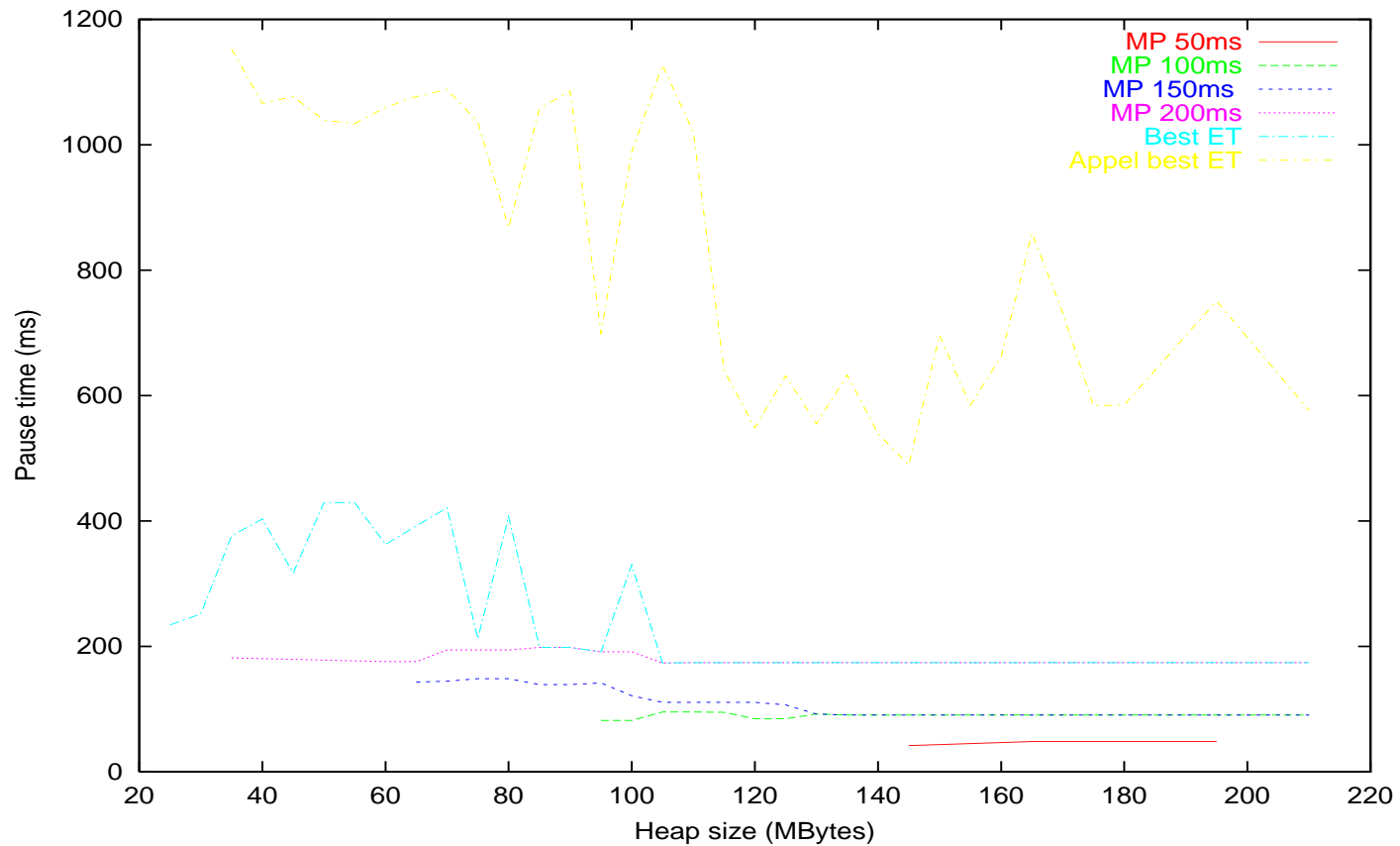
# javac Exec Time relative to Appel



# javac Mark/Cons



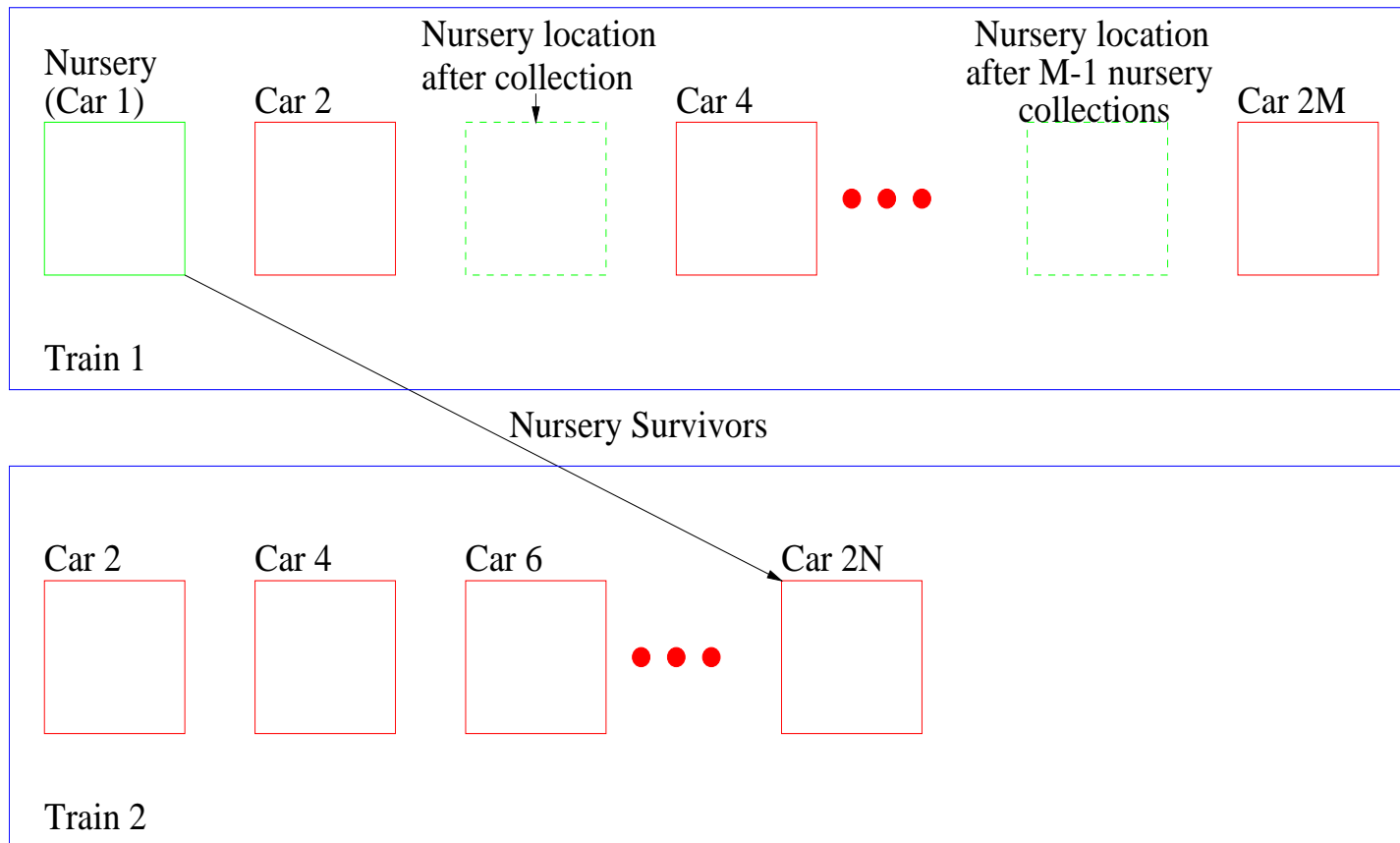
# javac GC Pause Times



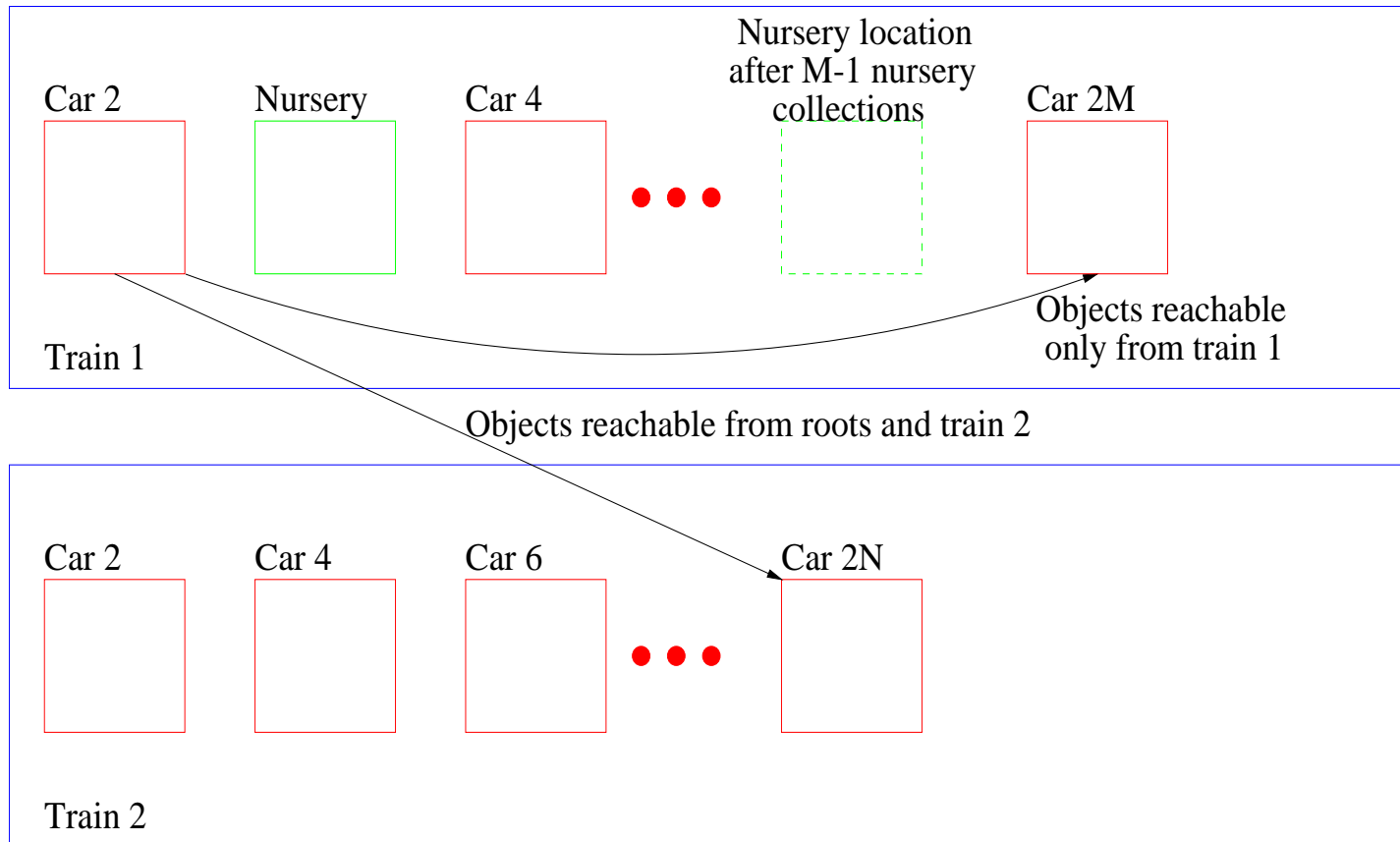
## **Two phase MOS Collection**

- **Nursery and train space are collected alternately in separate collections**
- **Cars are allocated with gaps in the logical address space**
- **After a nursery collection, the nursery is assigned an address greater than the last car to be collected (but smaller than the subsequent cars)**
- **This arrangement is required so that pointer stores from nursery into train space are recorded by the write barrier**

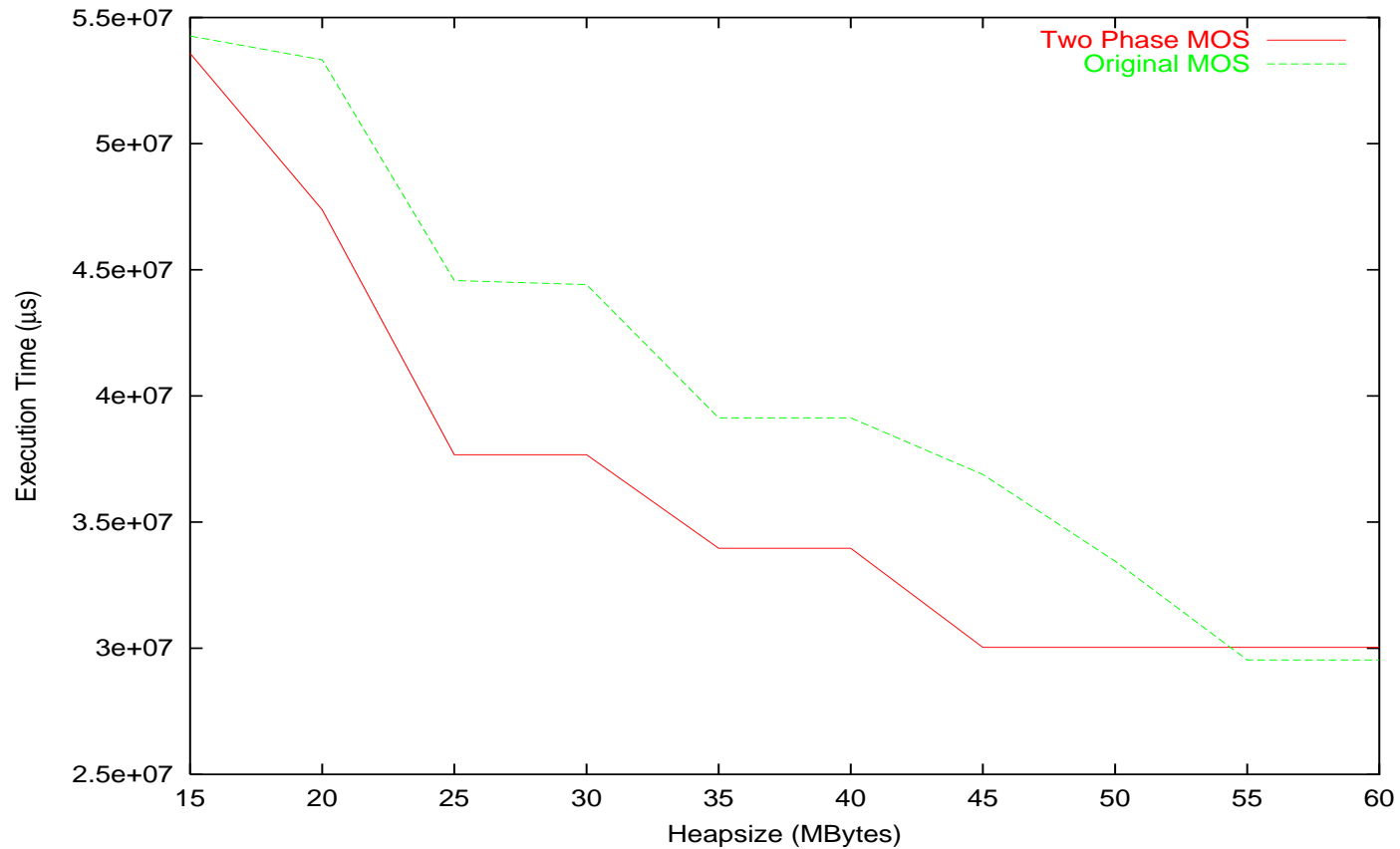
# Two Phase - Nursery Collection



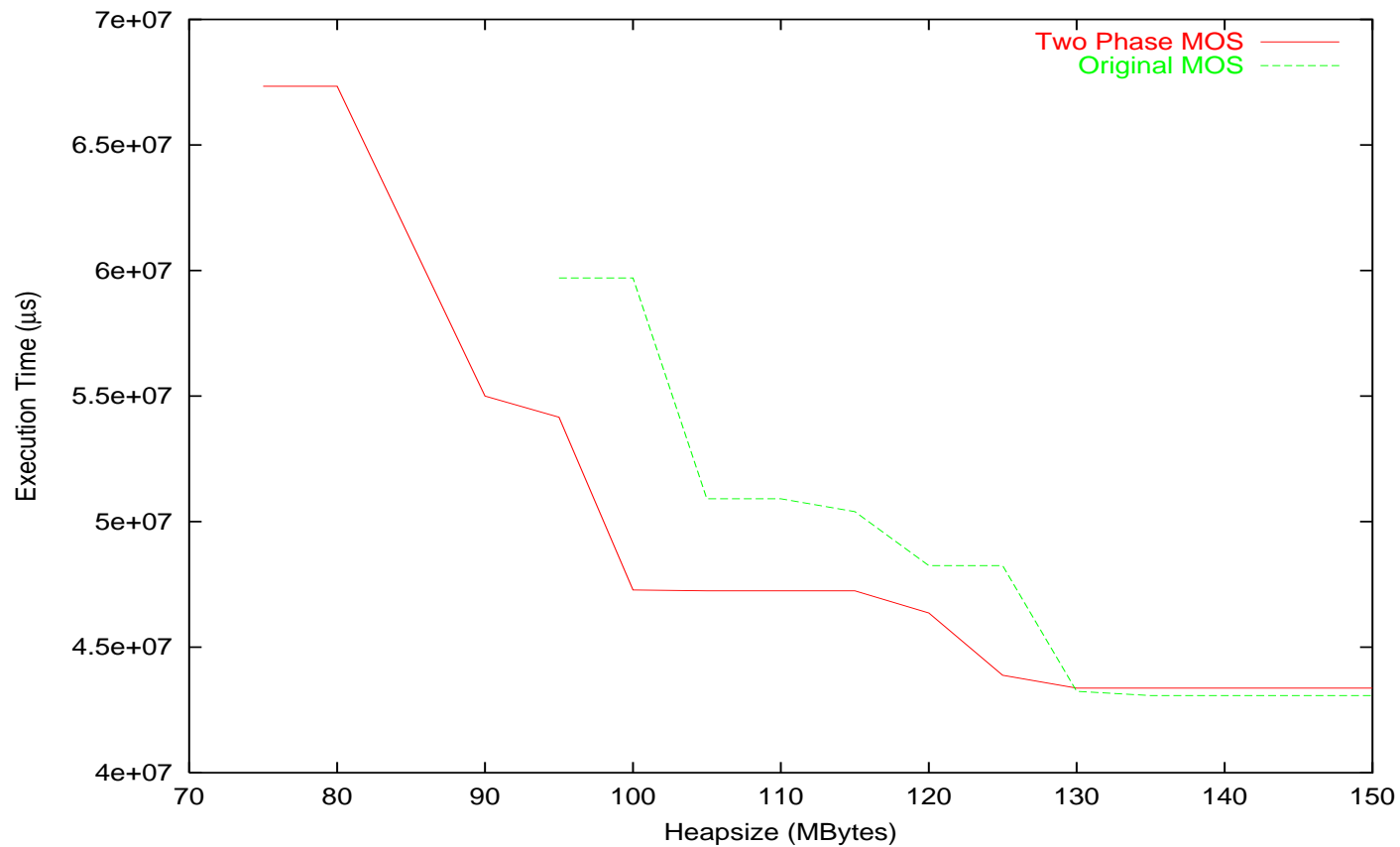
# Two Phase - Car Collection



# jess(50 ms. MPT) performance



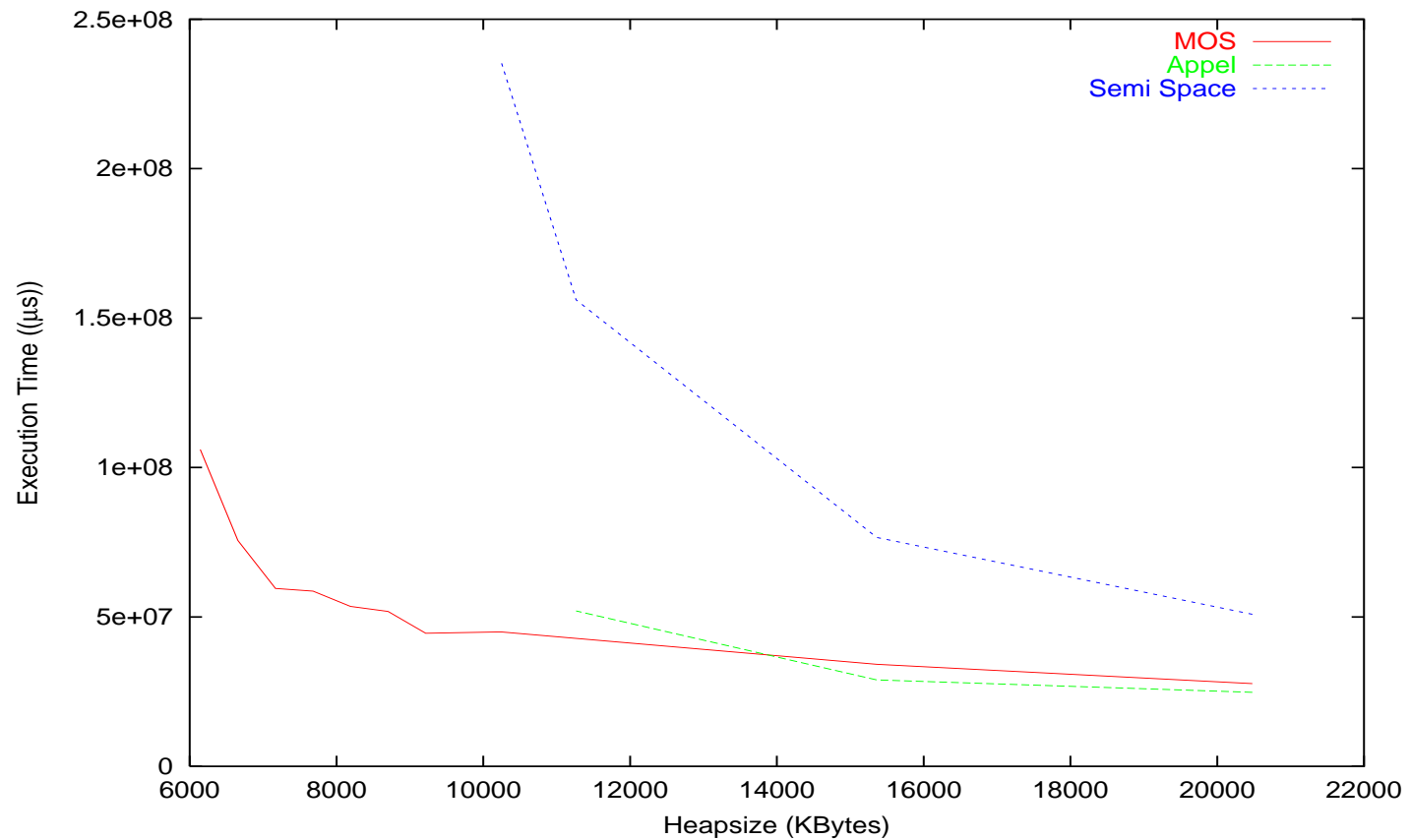
# javac(100 ms. MPT) performance



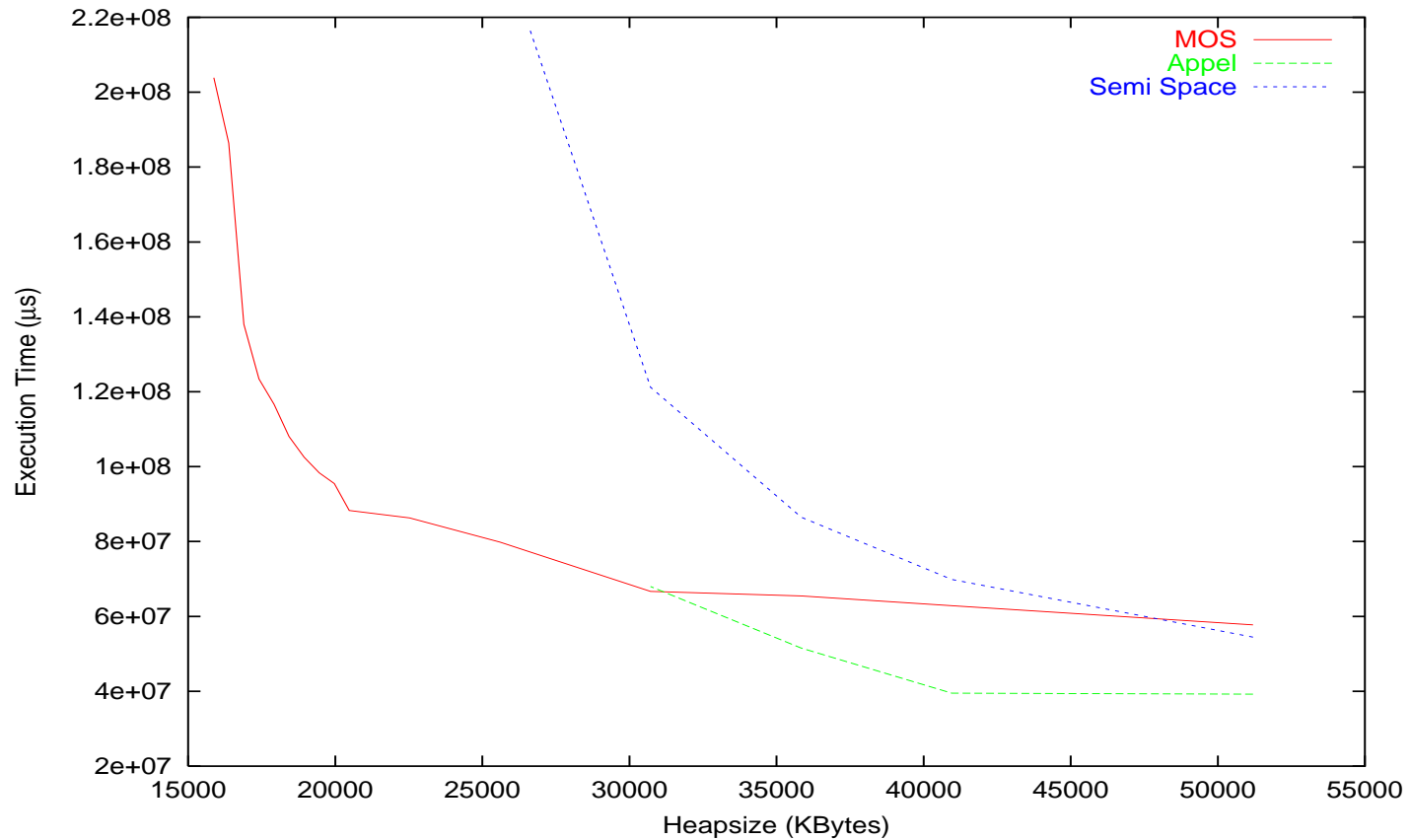
# Small Heaps

- **Copying collectors (Appel, Semi Space) require heapsize to be at least twice the maximum live size**
- **MOS does not have this requirement, additional space required is nursery size+car size**
- **This property allows MOS to run in much smaller heaps than other copying collectors**
- **However, repeated train collections are usually required at each collection point, in order to reclaim sufficient space**

# jess(MLS 5M) performance



# javac(MLS 13M) performance



# Remsets

- Large remsets exist, caused by 'popular' objects
- Number of large remsets (> 25000 entries) is low
- Percentage of large remsets for jess, < 0.3%
- Percentage of large remsets for javac, < 4%
- Large remsets have an effect on pause time
- Overall execution time not significantly affected

# Conclusions

- **Bounded pause times can be achieved using MOS**
- **High amount of copying causes poor execution times**
- **Requires larger heap sizes, about 4 times the min heap size for acceptable performance**
- **MOS however can also run in very small heaps**
- **Better mutator utilization for smaller time windows, overall utilization much lower than Appel.**
- **Remsets are an issue for pause time but do not affect execution time significantly**