Document WG 14/N1456

What's New in Objective-C*

WG14 April 21, 2010

*Derived from an Apple *World Wide Developer Conference* (WWDC) talk given in June 2008

Blaine Garst
Wizard of Runtimes

Objective-C Evolution

Objective-C

Objective-C 2.0

1988-1991

@interface
@implementation
non-fragile methods

Class
Categories
forwarding

Protocols

NeXT
Mac OS X 10.0
Mac OS X 10.1
Mac OS X 10.2

2003

@try@catch@finally
@synchronized

Mac OS X 10.3
Mac OS X 10.4

2007

for..in
@optional in protocols
Properties
- Declarations
- Synthesized impl
- Synthesized ivars

Non Fragile ivars

Opt-In GC

Mac OS X10.5
iPhone

2008

Associative References
Scalable GC

Blocks for C, ObjC, C++

Mac OS X10.6
"SnowLeopard"

Modern Runtime

- Available on Mac OS X 64-bit and iPhone
- API instead of structure definitions for the runtime
- Non-fragile instance variables
- Unified exception model with C++
- Linkage restrictions on @private access
- New @package directive enforced

GC write-barrier

```
void foo() {
    static id AGlobalID;
    id localID;
    Foo *fooObject = ...;

    localid = [...];     // no assign helper
    AGlobalId = localid; // objc_assign_global(...);
    fooObject->ivar = [...];  // objc_assign_ivar(...);
}
```

Opt-in GC write-barriers were dynamically optimized on PPC to be a two instruction overhead under non-GC use (bla absolute; return)

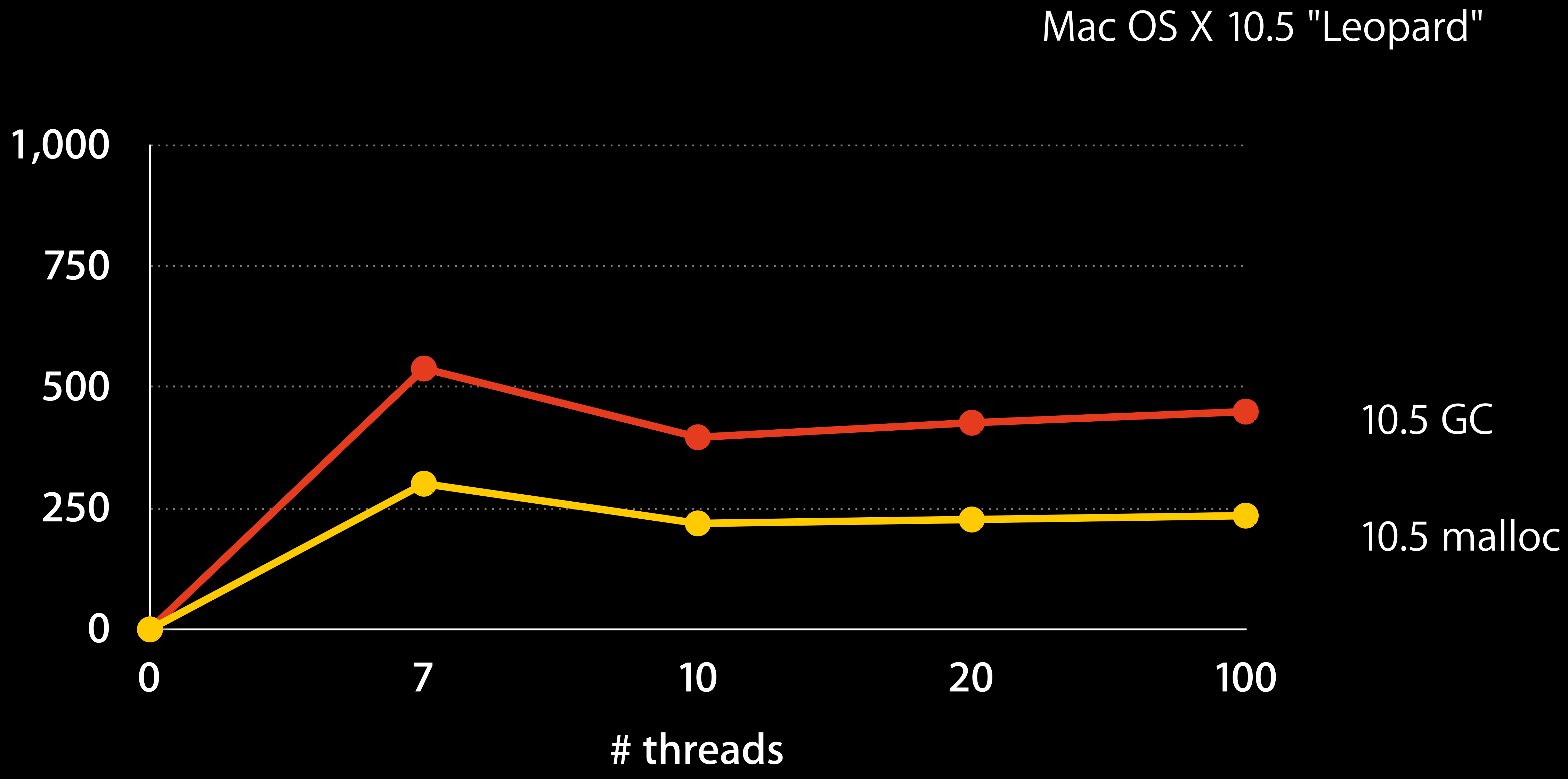
Thread Local Collection

- The Generational Hypothesis
  - Traditional: “most objects die young”
  - Mac OS X 10.6: “most objects die local”
- How fast can one create garbage?

```
void deathByAlloc(void *arg) {
    NSThread *mythread = [NSThread currentThread];
    while (1) {
        NSAutoreleasePool *pool = [[NSAutoreleasePool alloc] init];
        for (int counter = 0; counter < 100; ++counter) {
            [[[NSObject alloc] init] autorelease];
        }
        [pool drain];
        OSAtomicIncrement64(&Nallocations);
    }
}
```

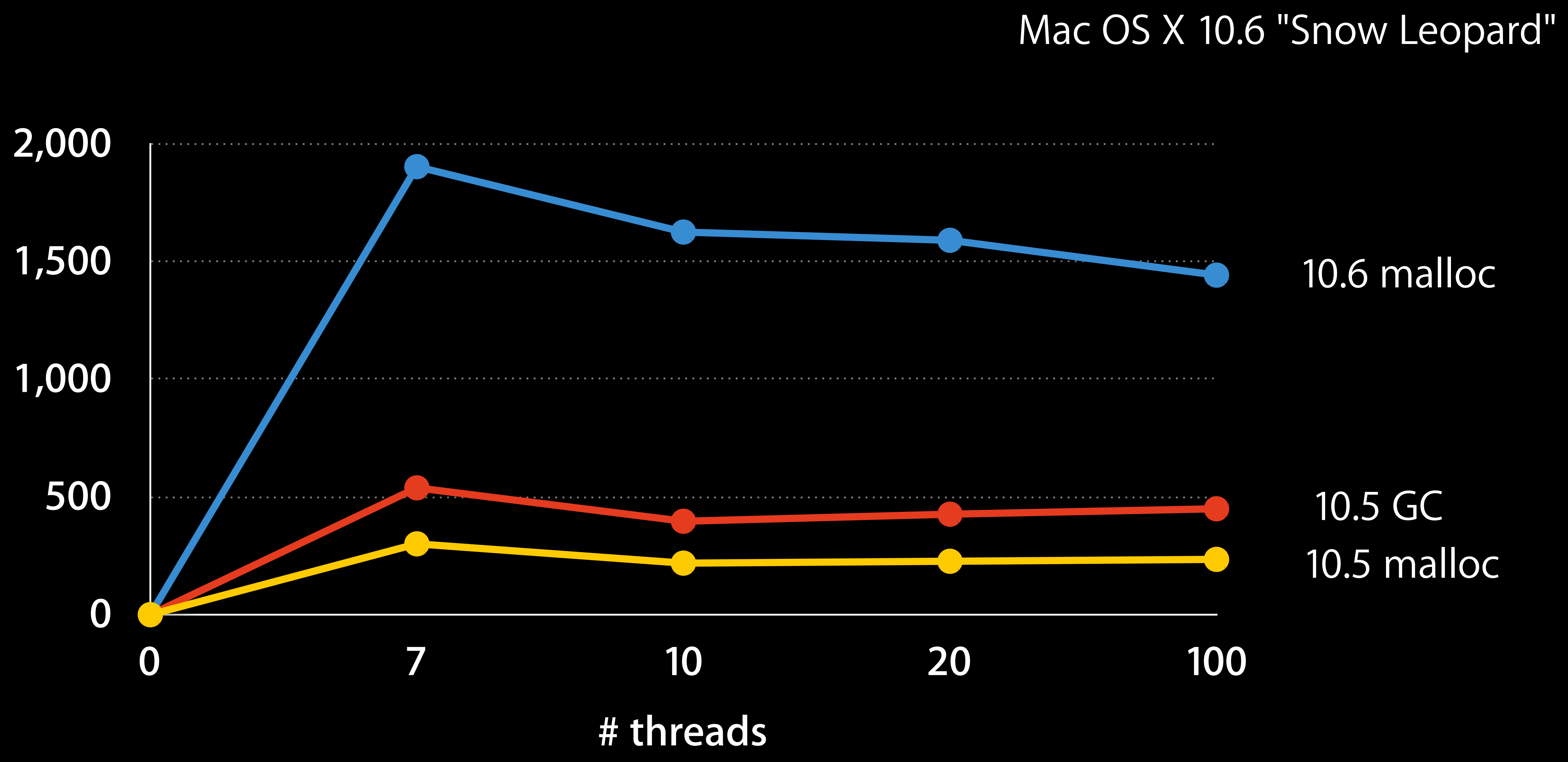
deathByAlloc Microbenchmark

1000s of allocations+recovery/sec



Mac OS X 10.6: Improved malloc!

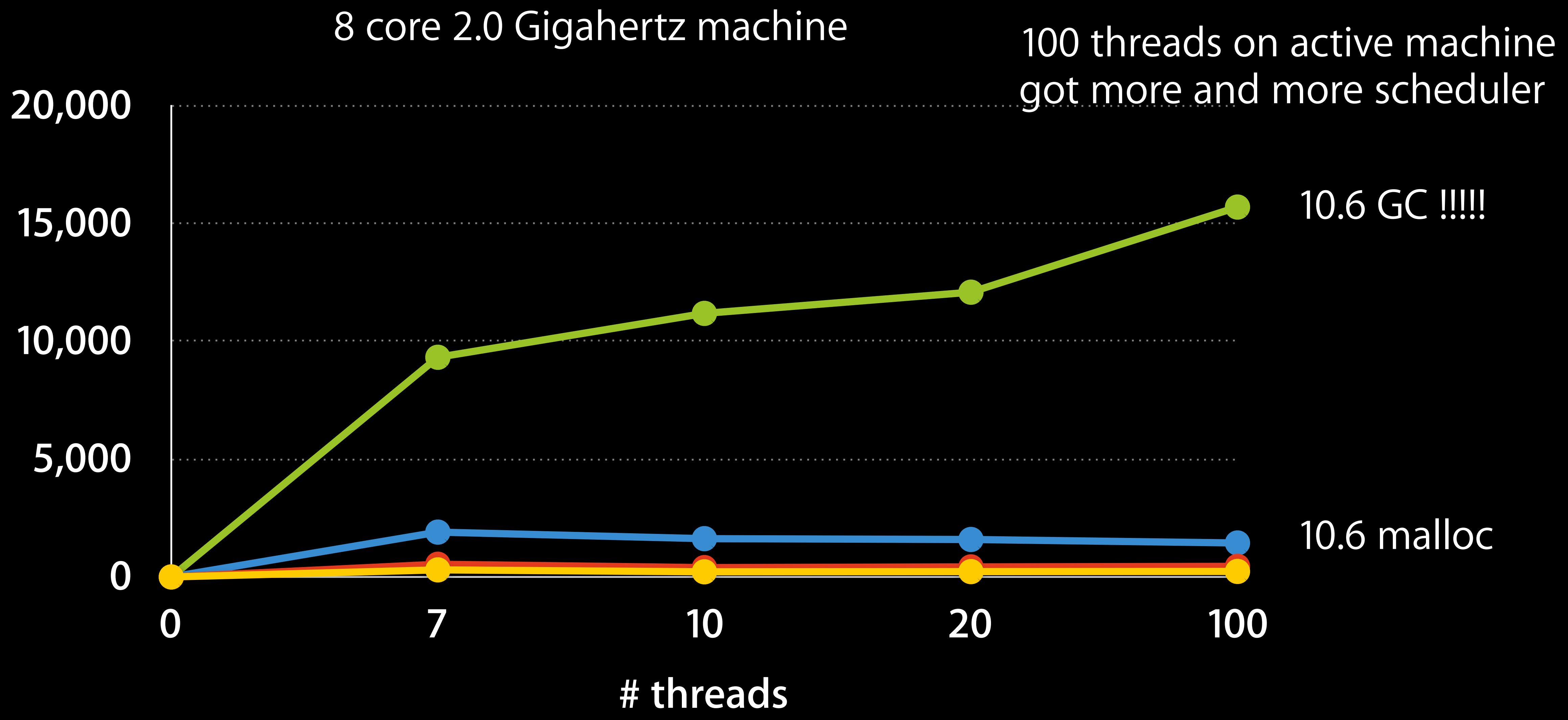
1000s of allocations+recovery/sec



TLC === "Thread Local Cache/Collector"

Mac OS X 10.6: TLC Performance!!

1000s of allocations+recovery/sec



~ 1.3 million allocation+recover/sec/core

~ half of all GUI allocations benefit

Associative References: The Problem

```
@interface NSObject (MyStuff)
- (void)makeSomethingNifty;
- niftyValue;
@end
```

```
@implementation NSObject (MyStuff)
- (void)makeSomethingNifty {
    Nifty *nifty = ...
    ???
}
- niftyValue { return ???; }
@end
```

Global MapTable?
—only if object never dies

GC Weak/Strong MapTable?
—unrecoverable cycle!!!

Associative References: The Solution!

```
@implementation NSObject (MyStuff)
- (void)makeSomethingNifty {
   Nifty *nifty = ...
   objc_setAssociatedObject(self,
       &uniqueLocation,
       nifty,
       OBJC_ASSOCIATION_RETAIN);
}
- niftyValue {
  return objc_getAssociatedObject(self,
             &uniqueLocation);
}
@end
```

Associative References

- Add data to arbitrary objects without their collusion
- Came from unexpected GC “leak”
    - Strong values in global table holds cycles
- Works in GC/non-GC
- *New design pattern!!*
- *Not trivially cheap*

System framework APIs

```
void    *bsearch_b(const void *key,
           const void *base, size_t nel, size_t width,
           int (^compar)(const void *, const void *));

int      heapsort_b(void *base, size_t nel, size_t width,
           int (^compar)(const void *, const void *));

int      mergesort_b(void *base, size_t nel, size_t width,
           int (^compar)(const void *, const void *));

void     qsort_b(void *base, size_t nel, size_t width,
           int (^compar)(const void *, const void *));
```

System framework APIs

```
int scandir_b(const char *dirname, struct dirent ***namelist,
   int (^select)(struct dirent *),
   int (^compar)(const void *, const void *));


void  err_set_exit_b(void (^exitb)(int));

int   atexit_b(void (^blk)(void));

int   glob_b(const char * __restrict pattern, int flags,
           int (^errblk)(const char *, int),
           glob_t * __restrict pglob);
```

Blocks: What are they good for?

- Concisely express units of work
- Provide fast and concise variations on iterators
- Greatly simplify function-with-void-* situations
- Provide multi-language support for callbacks

Something for Fun

```
@interface NSObject (AttachFinalizeBlock)
- (void)attachFinalizeBlock:(void (^)(void))block;
@end
```

```
@interface DeathWatcher : NSObject
@property(copy) void (^deathBlock)(void);
@end
@implementation DeathWatcher
@synthesize deathBlock;
- (void)finalize {  deathBlock(); [super finalize]; }
@end
@implementation NSObject (AttachFinalizeBlock)
- (void)attachFinalizeBlock:(void (^)(void)block) {
    DeathWatcher *death = [[DeathWatcher alloc] init];
    death.deathBlock = block;
    objc_setAssociatedObject(self,
                    [DeathWatcher self],
                    death,
                    OBJC_ASSOCIATION_RETAIN);
}
@end
```