GCD,
hop on now

Patrick Stein aka Jolly
www.jinx.de/teclog or @jollyjinx






Overview

- Blocks
- Grand Central Dispatch
- Demo

Blocks

other languages call them
closures
anonymous functions

but they are not

- Blocks are like inline functions right were you need them
- Like functions just use ^ instead of * : ^ { }
- Blocks are a C-extension available in Clang and GCC
- Obj-C and C++

Blocks

- How does it look like ?

```
^ { printf("hello world\n"); }

^ { return 23; }

^ { return 'c'; };
```

Blocks

- Declaration:
- Typedefs:

```
void (^block)(void);

typedef void (^workblock_type)(void);
```

```
void repeat(int n, workblock_type workblock)
{
    for (int i=0; i<n; i++) workblock();
}
```

```c
#include <libc.h>

typedef void(^helloblock_type)(void);

int main()
{
   int              i;
   helloblock_type  helloblock = ^{ printf("hello world.\n");  };

   for(i=0; i< 10; i++)
   {
         helloblock();
   }

   return EXIT_SUCCESS;
}
```

hello world.
hello world.
hello world.

```
#include <libc.h>

typedef void(^helloblock_type)(void);

int main()
{
   int              i=0;
   helloblock_type  helloblock = ^{ printf("hello world.%d\n",i);  };

   for(i=0; i< 10; i++)
   {
         helloblock();
   }

   return EXIT_SUCCESS;
}
```

```
hello world.0
hello world.0
hello world.0
```

```
#include <libc.h>

typedef void(^helloblock_type)(void);

int main()
{
   int               i=0;
   helloblock_type  helloblock = ^{ printf("hello world.%d %p\n",i,&i);  };

   printf("%p\n",&i);
   for(i=0; i< 10; i++)
   {
         helloblock();
   }
   return EXIT_SUCCESS;
}
```

```
0x7fff5fbff6fc
hello world.0 0x7fff5fbff6ac
hello world.0 0x7fff5fbff6ac
```

```
#include <libc.h>

typedef void(^helloblock_type)(void);

int main()
{
   __block int       i=0;
   helloblock_type  helloblock = ^{ printf("hello world.%d\n",i);  };

   for(i=0; i< 10; i++)
   {
         helloblock();
   }

   return EXIT_SUCCESS;
}
```

hello world.0
hello world.1
hello world.2

```c
#include <libc.h>

typedef void(^helloblock_type)(int);

int main()
{
    __block int       i=0;
    helloblock_type  helloblock = ^(int j){ printf("hello:%d %d\n",i,i+j);};

    for(i=0; i< 10; i++)
    {
          helloblock(i);
    }

    return EXIT_SUCCESS;
}
```

```
hello:0 0
hello:1 2
hello:2 4
```

old style

```
...
{
   FILE *fp = fopen(filename, "r");
   if( !fp )
   {
      return;
   }

   char line[1024];
   while( fgets(line, sizeof(line), fp) )
   {
      // do the work here
   }
   fclose(fp);
}
...
```

blocks version

```
typedef void (^workblock_t)(char *);

void foreachlineinfile(char *filename, workblock_t block)
{
   FILE *fp = fopen(filename, "r");
   if( !fp )
   {
      return;
   }

   char line[1024];
   while( fgets(line, sizeof(line), fp) )
   {
      block(line);
   }
   fclose(fp);
}
```

```
...
foreachlineinfile(filename,^(char *line){
      // do the work here
}
```

qsort

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

with context

```
void qsort_r(void *base, size_t nel, size_t width, void *thunk, int
(*compar)(void *, const void *, const void *));
```

with blocks

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

function sorting

```
int mycomparefunction(void *a, void *b, void *context)
{
   book *booka = a;
   book *bookb = b;

   booksortcontext *sorting = context;

   for( int rank=0; rank < sorting->compareoptioncount; rank++)
   {
      case(sorting->compareoption[rank])
      {
         AUTHOR_NAME:   int result=compare_books_name(a,b);
               if( 0 != result ) return result;
               break;
         PUBLISHER_NAME: ..
         …
      }
   }
   return 0;
}
```

function sorting

```
...
   qsort( bookarray, bookcount, sizeof(book), &context, &mycomparefunction) ;
...
```

sorting with blocks

```
....
qsort_b( bookarray, bookcount, sizeof(book), int ^(void *a, void *b)
{
   book *booka = a;
   book *bookb = b;

   for( int rank=0; rank < sorting->compareoptioncount; rank++)
   {
      case(sorting->compareoption[rank])
      {
         AUTHOR_NAME:   int result=compare_books_age(a,b);
               if( 0 != result ) return result;
               break;
         PUBLISHER_NAME :..
         ...
```

sorting with blocks

```
....
qsort_b( bookarray, bookcount, sizeof(book), int ^(void *a, void *b)
{
   book *booka = a;
   book *bookb = b;

   int compareresult;

   if( 0 == (compareresult=compare_books_age(a,b)) )
       return compareresult;
   if( 0 == (compareresult=compare_books_publisher(a,b)) )
       return compareresult;

   ...
```

Blocks and Objective-C

- All Blocks are Objective-C Objects
  `[^{ do something; } copy]`
- respond to -copy, -retain, -release, -autorelease
- `- (void) foo:(int)x withBlock:(void (^)(int)) aBlock;`
- blocks are used in many method arguments now

Blocks and Objective-C

- **NSArray**:
  - `-enumerateObjectsUsingBlock:`
  - `-indexesOfObjectsPassingTest:`
- **NSDictionary**
  - `-enumerateKeysAndObjectsUsingBlock:`
  - `-keysOfEntriesPassingTest:`
- **NSBlockOperation : NSOperation : NSObject**

and many more...

NSDictionary

```
_block id myResult = nil;

[myDictionary enumerateKeysAndObjectsUsingBlock:
   ^ (id aKey, id aValue, BOOL *stop) {
        if( [aKey isEqual:magicObject)] )
        {
             myResult = aKey;
             *stop    = YES;
        }
   }];
```

NSDictionary

```
NSSet myResultSet = nil;

myResultSet = [myDictionary keysOfEntriesPassingTest:
   (BOOL (^)(id aKey, id aValue, BOOL *stop) {
        if( [aValue containsString:@"magic"]] )
        {
             return YES;
        }
        return NO;
   }];
```

NSArray

```
NSArray myResultArray = nil;

myResultArray = [myArray sortedArrayUsingComparator:
    ^ NSComparisonResult (id objectA, id objectB) {

      if( [objectA age] > [objectB age] )
       ....
           return NSOrderedAscending;
       ....
           return NSOrderedDescending;
       ....
           return NSOrderedSame;
    }];
```

Using blocks in methods

```
- (void) setWorkBlock:(workBlk_t)aBlock
{
    [myBlock release];
    myBlock = [aBlock copy];
}
```

copy not retain ! cause of stacks

```
typedef void(^workBlk_t)(void);

@interface Fasel:NSObject
{
      workBlk_t myBlock;
}
@property(setter=setWorkBlock:,copy) workBlk_t myBlock;
@end

@implementation Fasel
@synthesize myBlock;
@end
```

Memory management

```
- (void) myMethod
{
            id localObject    = ...;
    __block id i              = ...;
...
    = Block_copy( ^{
     instanceVar = [localObject addInteger:i];
    });
...
}
```

Block_copy()
Block_release()

Apple prefers [-copy]

- *Block_copy* - copies block to the heap
- Instance variables just work (no __block needed)
- *instanceVariable* is used - self is retained
- *localObject* will get a -retain

Grand Central Dispatch (GCD)

- All Macs now have multiple CPU's
- Harnessing the power can be complex
  - pthreads, NSThreads, NSOperation, Intel® TBB,...
  - synchronization / messaging between threads
  - current system state (#CPUs, Powerstate, other programs)

threads tend to become...

```
....
   [screentrackerLock lock];
      if( SCREEN_WILL_CHANGE == [screentrackerLock condition] )
      {
         [screentrackerLock unlockWithCondition:WAITING_FOR_CHANGE];
         [screentrackerLock lockWhenCondition:SCREEN_DID_CHANGE];
      }
   [screentrackerLock unlockWithCondition:NOT_WORKING];

   pthread_mutex_lock(&tiles.tilestosendcounterlock);
       tiles.tilestosendcounter = tiles.width*tiles.height;
   pthread_mutex_unlock(&tiles.tilestosendcounterlock);
   pthread_cond_signal(&tiles.tilestosendcondition);
....
```

GCD makes things easier

- Queues and Blocks are easy to understand
- You have to break up the App into blocks
- `man dispatch`
- `#include <dispatch/dispatch.h>`
- GCD takes care of
  - creation/destruction and (fast) scheduling of threads
  - event and synchronization handling

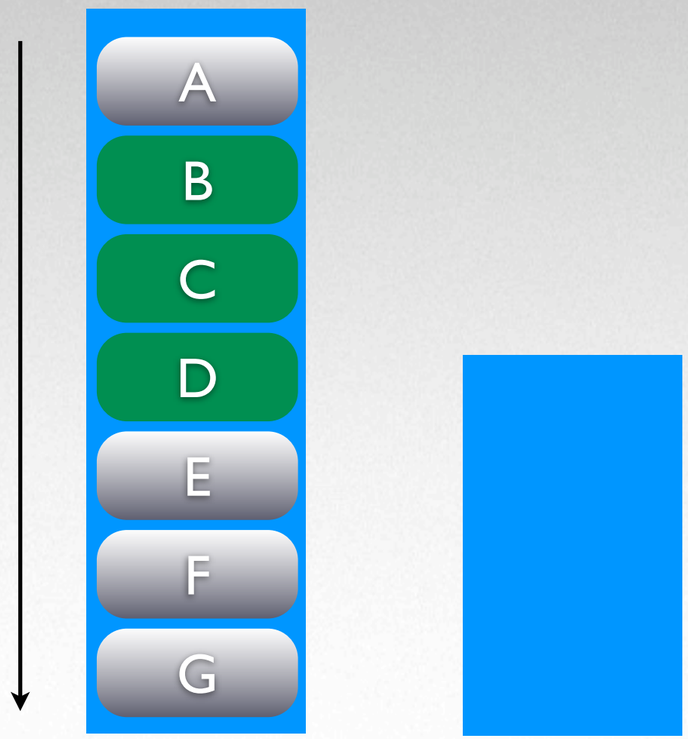
All there is to GCD

- Queues
- Groups
- Semaphores
- Event Sources
- Time, Once

Queues

- A queue is a list of execution blocks
- Asynchronous execution
- Enqueue/Dequeue is FIFO
- Enqueue blocks in queues:
  `dispatch_async(queue, ^{ printf("Hello GCD\n"); });`
- Enqueue functions in queues:
  `dispatch_async_f(queue, context, function);`

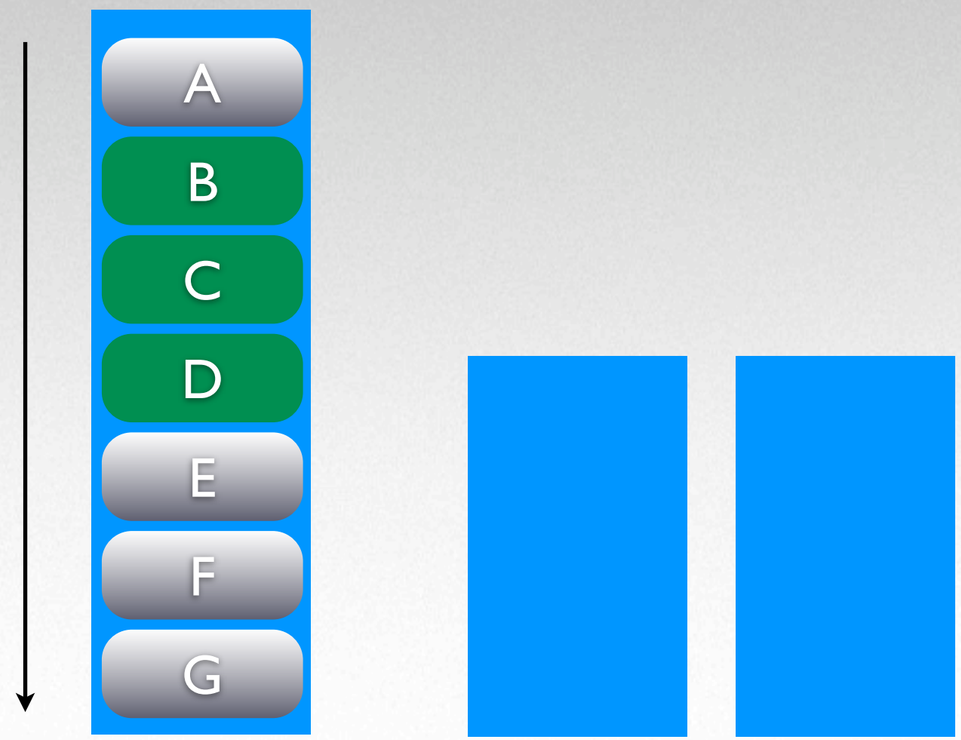
Serial Queues



Serial Queues

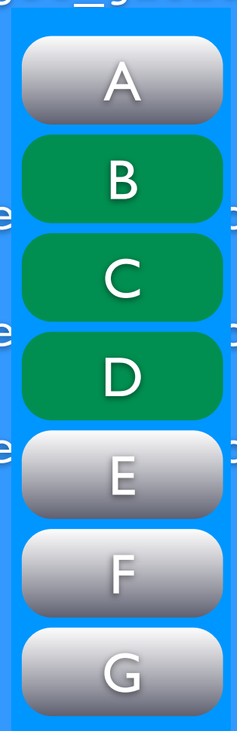
```
queue = dispatch_queue_create("de.macoun",NULL);

calculate A;

dispatch_async(queue,                    late B });

dispatch_async(queue,                    late C });

dispatch_async(queue,                    late D });

calculate E;

calculate F;
```

Global Queue (concurrent)



Global Queue (concurrent)

```
queue = dispatch_get_global_queue(0,0);

calculate A;

dispatch_async(que          calculate B });

dispatch_async(que          calculate C });

dispatch_async(que          calculate D });

calculate E;

calculate F;
```



Global Queue (concurrent)

- Global Dispatch Queue: concurrent execution of blocks

```
dispatch_queue_t global_queue = dispatch_get_global_queue(0,0);

dispatch_async(global_queue,^{printf("hello\n");});
dispatch_async(global_queue,^{printf("macoun\n");});
dispatch_async(global_queue,^{printf("2009\n");});
```

- Main Queue: serial execution on main thread
  - dispatch_main() or NSApplicationMain()
- Your Queues are always serial but not on the main thread

Queues

- dispatch_queue_t dispatch_get_main_queue(void);
- dispatch_queue_t dispatch_get_global_queue(long priority, unsigned long flags);
- dispatch_queue_t dispatch_queue_create(const char *label, dispatch_queue_attr_t attr);
- void dispatch_async(dispatch_queue_t queue, void (^block)(void));
- void dispatch_sync(dispatch_queue_t queue, void (^block)(void));
- void dispatch_main(void);

Groups

- Grouping multiple blocks
- Do something when a group finishes:

```
dispatch_group_t my_group = dispatch_group_create();

dispatch_group_async(my_group,queueOne, ^{ /* do something */ });
dispatch_group_async(my_group,queueTwo, ^{ /* do something */ });

dispatch_group_notify(my_group, dispatch_get_main_queue(),^{
  // group has been finished
});

dispatch_release(my_group);
```

Groups

- Wait for a group to finish

```
dispatch_group_t my_group = dispatch_group_create();

dispatch_group_async(my_group,queueOne, ^{ /* do something */ });
dispatch_group_async(my_group,queueTwo, ^{ /* do something */ });

dispatch_group_wait(my_group, DISPATCH_TIME_FOREVER);
dispatch_release(my_group);
```

Groups

- dispatch_group_t dispatch_group_create(void);
- long dispatch_group_wait(dispatch_group_t group, dispatch_time_t timeout);
- void dispatch_group_notify(dispatch_group_t group, dispatch_queue_t queue, void (^block)(void));
- void dispatch_group_async(dispatch_group_t group, dispatch_queue_t queue, void (^block)(void));

Event sources

- Source can be:
  data, file descriptors, processes, timers, mach ports, signals, vnodes
- Usage:
  - Create a source,
  - Setup eventhandler
  - Resume the source

Event sources

leeway & battery

```
dispatch_source_t source;
source = dispatch_source_create(DISPATCH_SOURCE_TYPE_TIMER, 0, 0,
dispatch_get_main_queue());

dispatch_source_set_event_handler(source,^{printf("hello.\n");});

int64_t         interval  = 1ull * NSEC_PER_SEC;
int64_t         leeway    = 5ull * NSEC_PER_SEC;

dispatch_time_t start     = dispatch_time(DISPATCH_TIME_NOW, interval);

dispatch_source_set_timer(source, start, interval, leeway);
dispatch_resume(source);
```

Event sources

- dispatch_source_t dispatch_source_create(dispatch_source_type_t type, uintptr_t handle, unsigned long mask, dispatch_queue_t queue);
- void dispatch_source_set_event_handler(dispatch_source_t source, void (^block)(void));

Semaphores

- Semaphore used for synchronized access

```
sema = dispatch_semaphore_create(0);

dispatch_async(queue, ^{
  foo();
  dispatch_semaphore_signal(sema);
});

bar();

dispatch_semaphore_wait(sema, DISPATCH_TIME_FOREVER);
```

Semaphores

- restricting access to limited resources

```
sema = dispatch_semaphore_create(getdtablesize() / 4);

....
int openfile(char *filename)
{
  dispatch_semaphore_wait(sema, DISPATCH_TIME_FOREVER);
  return open(filename, O_RDONLY);
}
....
void closefile(fd)
{
  close(fd);
  dispatch_semaphore_signal(sema);
}
```

Semaphores

create 0 synchronizing
create n counting

signal increment
wait decrement

- dispatch_semaphore_t dispatch_semaphore_create(long count);
- long dispatch_semaphore_signal(dispatch_semaphore_t semaphore);
- long dispatch_semaphore_wait(dispatch_semaphore_t semaphore, dispatch_time_t timeout);

Dispatch Objects

- Queues, Groups, Semaphores, Event sources are dispatch objects
- void dispatch_retain(dispatch_object_t object);
- void dispatch_release(dispatch_object_t object);
- void dispatch_suspend(dispatch_object_t object);
- void dispatch_resume(dispatch_object_t object);
- void * dispatch_get_context(dispatch_object_t object);
- void dispatch_set_context(dispatch_object_t object, void *context);

Time, Once

- dispatch_time_t
  dispatch_walltime(struct timespec *base, int64_t offset);
- void dispatch_after(dispatch_time_t when, dispatch_queue_t queue, void (^block)(void));
- void dispatch_once(dispatch_once_t *predicate, void (^block)(void));

Objective-C and GCD

- NSAutoreleasePool is created for you
- GCD works in GC environment
- dispatch objects are (currently) not collectable

Objective-C and GCD

```
- (IBAction)startLongCalculation:(id)sender
{
      dispatch_async(dispatch_get_global_queue(0, 0),^{

          // do long calculation

          dispatch_async(dispatch_get_main_queue(), ^{
             [self updateViews];
          });
      });
}
```

Things to think about

- Producer/consumer should have about the same speed (memory issues, thread pool creation/destruction costs)
- Blocks are lightweight but not completely free of cost (for loop creates block or block creates for loop)
- Array striding

Zur Demo: Mit Taste „**H**“ oder „cmd + Tab“ auf den Finder wechseln.

Danach mit „cmd + Tab“ oder Klick auf Keynote-Icon zurück in die Präsentation.

Demo

Links

http://arstechnica.com/apple/reviews/2009/08/mac-os-x-10-6.ars/10
http://cocoasamurai.blogspot.com/2009/09/guide-to-blocks-grand-central-dispatch.html
http://cocoasamurai.blogspot.com/2009/09/making-nsoperation-look-like-gcd.html
http://developer.apple.com/mac/articles/cocoa/introblocksgcd.html
http://developer.apple.com/mac/library/releasenotes/MacOSX/WhatsNewInOSX/Articles/MacOSX10_6.html
http://devwhy.blogspot.com/2009/08/grand-central-dispatch.html
http://eschatologist.net/blog/?p=232
http://libdispatch.macosforge.org/
http://parmanoir.com/8_ways_to_use_Blocks_in_Snow_Leopard
http://th30z.netsons.org/2009/09/grand-central-dispatch-first-look/
http://thirdcog.eu/pwcblocks/
http://www.mcubedsw.com/blog/index.php/site/comments/xcode_3.2_teh_awesome_edition/
http://www.mikeash.com/?page=pyblog/friday-qa-2009-08-28-intro-to-grand-central-dispatch-part-i-basics-and-dispatch-queues.html
http://www.mikeash.com/?page=pyblog/friday-qa-2009-09-11-intro-to-grand-central-dispatch-part-iii-dispatch-sources.html
http://www.mikeash.com/?page=pyblog/gcd-is-not-blocks-blocks-are-not-gcd.html
http://www.subfurther.com/blog/?p=699

Fragen ?

Patrick Stein aka Jolly
www.jinx.de/teclog or @jollyjinx






Vielen Dank

Patrick Stein aka Jolly
www.jinx.de/teclog or @jollyjinx






Hier ist das Ende

Macoun'09