libdispatch

- Grand Central Dispatch
- Asynchronous & concurrent programming model
- From apple
- http://libdispatch.macosforge.org/

chenj@lemote.com

Based on Queues

- split tasks to blocks and send them to different queues.
- A block is scheduled in its target queue.
- Notifacation when a group of blocks finish executing.
- Queue types: Global Concurrent Queues, Main Queue, Private Serial Queues

Global Concurrent Queues

- q = dispatch_get_global_queue(

  DISPATCH_QUEUE_PRIORITY_DEFAULT,

  NULL /* reserved for future use */);
- Execute function complex_calculation 100 times:
  - dispatch_apply_f(100, q,

    user_data, complex_calculation);
  - complex_calculation(user_data, i); /* i ∈ [0, 100) */
  - more than one complex_calculation run parallely

Main Queue

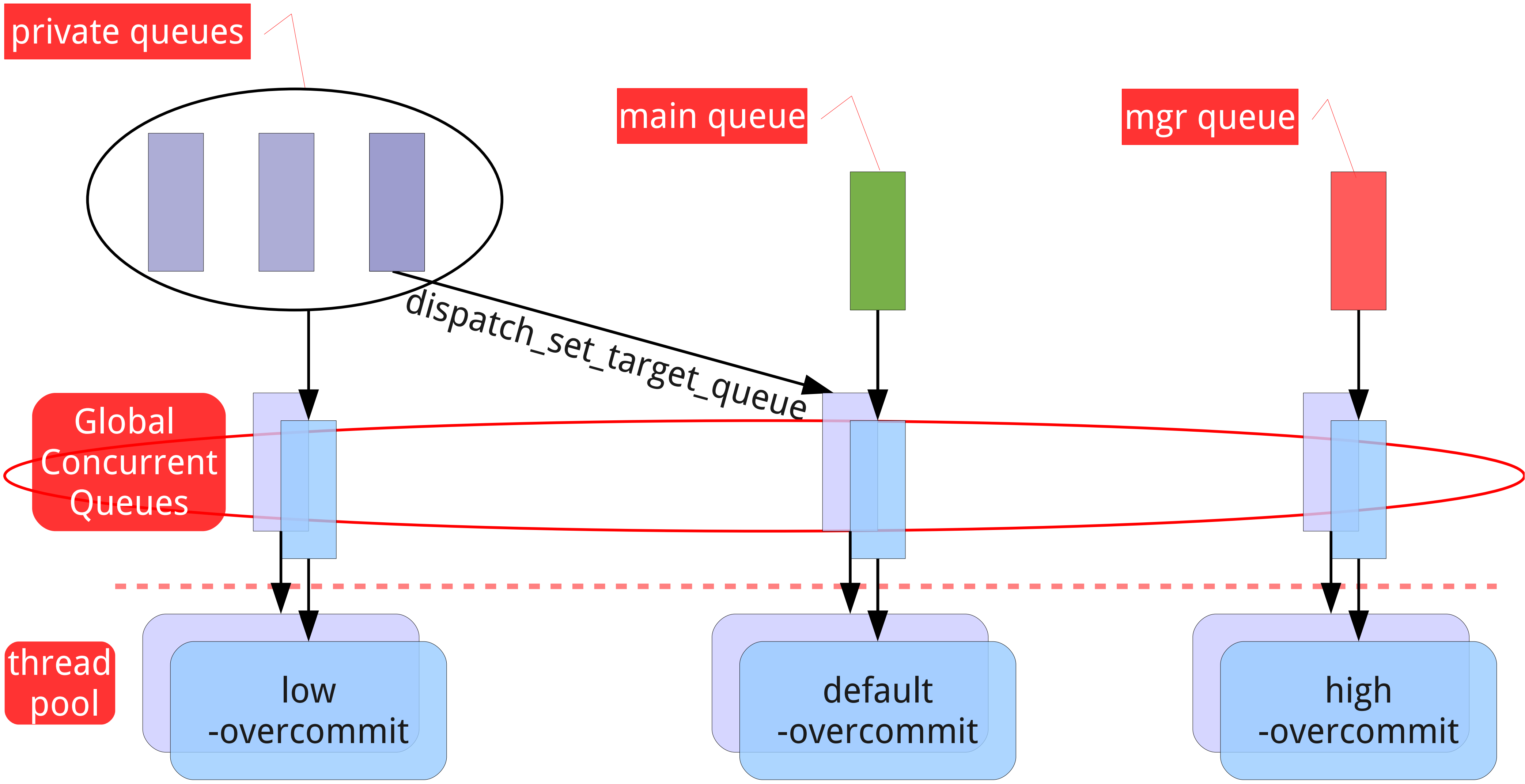
- Is a serial queue (back up by one thead)
- q_main = dispatch_get_main_queue();
- Is a global queue
- To integrate with Apple's Cocoa framework

Private Serial Queues

- q_sum = dispatch_queue_create("com.example.sum", NULL);
- Serialize access to shared data structures:

```
#define COUNT 128
double sum = 0;
void calc_func(void *data, size_t i) {
    double x = complex_calculation(i);
    double *sum = (double *)data;
    dispatch_async(q_sum, ^{ *sum += x});
}
dispatch_apply_f(COUNT, q_default, &sum, calc_func);
```

Relations between queues



Main classes and inheritance

dispatch_object_s

```
const void *do_vtable;
struct x *volatile do_next;
```

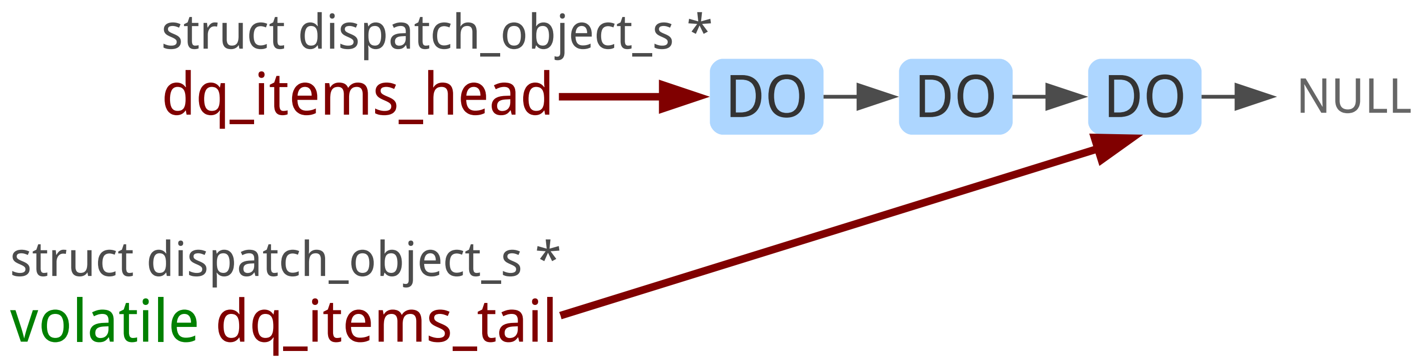
dispatch_continuation_s

```
unsigned int do_ref_cnt;
unsigned int do_xref_cnt;
unsigned int do_suspend_cnt;
struct dispatch_queue_s *do_targetq;
void *do_ctxt;
dispatch_function_t do_finalizer;
```

```
→dispatch_queue_s
  ↳ dispatch_source_s
→dispatch_queue_attr_s
→dispatch_source_attr_s
→dispatch_semaphore_s  =  dispatch_group_s
```

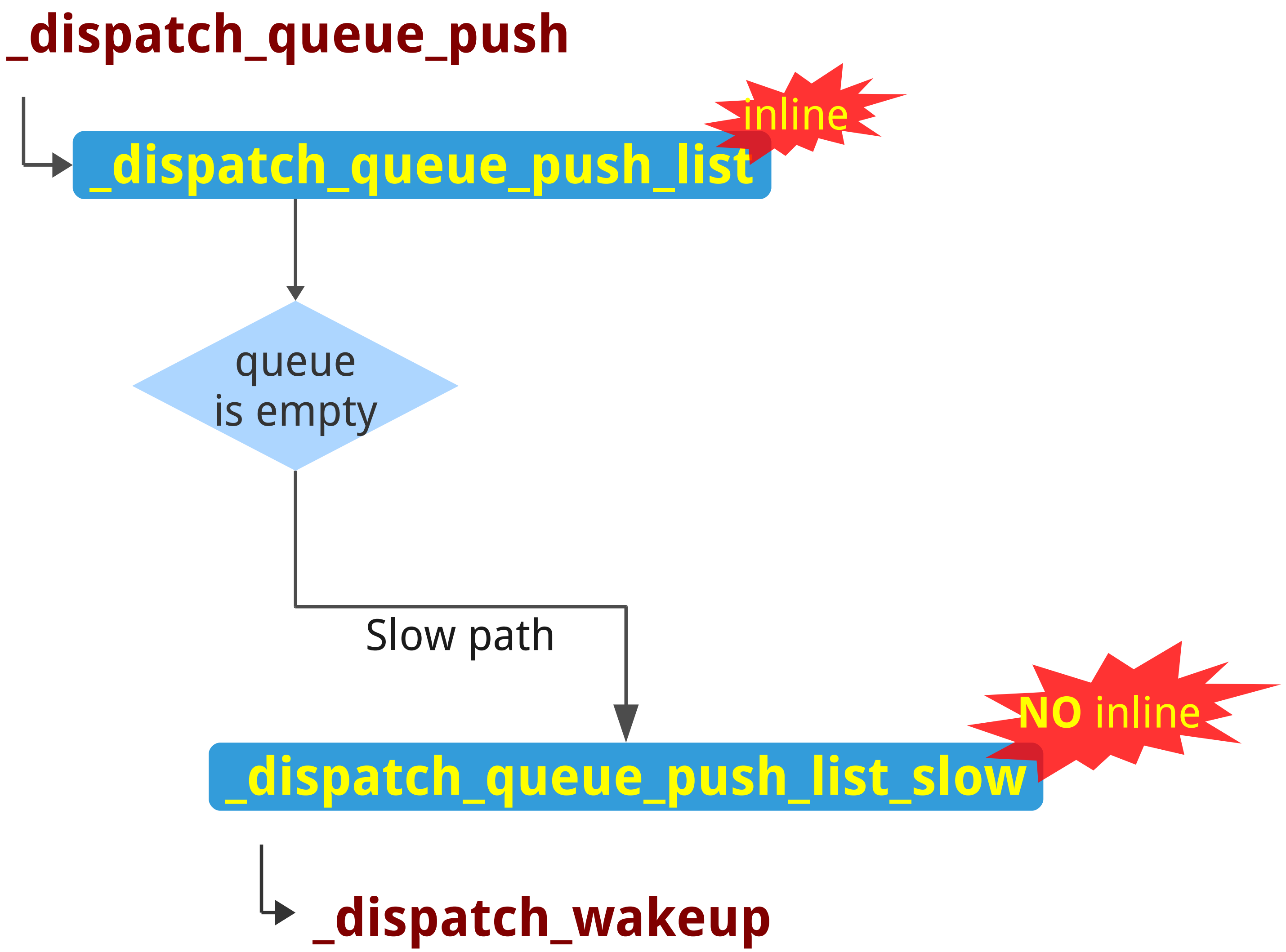
dispatch_queue_s

- Contain a list of DO(dispatch_object_s)



- Num of Running DO: uint32_t dq_running;
- Width of concurrency: uint32_t dq_width;

Enqueue


_dispatch_queue_push
_dispatch_queue_push_list
inline
queue is empty
Slow path
_dispatch_queue_push_list_slow
NO inline
_dispatch_wakeup

Dequeue

- _dispatch_queue_concurrent_drain_one
    - Get and return a DO concurrently
- _dispatch_queue_drain
    - Get and process all DOs in the queue
    - Lock the queue before calling: _dispatch_queue_trylock(dq)

How a block be executed?

1. wrap a block to *dispatch_continuation_s*
2. _dispatch_queue_push to its target queue → _dispatch_wakeup the target queue if empty
3. _dispatch_wakeup do the following:
   - If SUSPENDED, return NULL
   - Run vtable->do_probe, if return false and the queue is empty, return NULL
   - _dispatch_trylock (object lock), if lock fail, return NULL
   - _dispatch_queue_push(dou.do->**do_targetq**, dou._do);
4. Finally _dispatch_queue_push to a root queue (i.e. Global Concurrent Queue, do_targetq == NULL)

Send to thread pool

$$\begin{tikzcd} \textbf{\_dispatch\_wakeup}(\textit{root queue}) \arrow[d] & & \\ \text{vtable->do\_probe} \arrow[rr, dashed] & & \boxed{\textbf{\_dispatch\_queue\_wakeup\_global}} \end{tikzcd}$$

Send to thread pool

_dispatch_wakeup(*root queue*)

└→ vtable->do_probe - - - - - - - -▶ **_dispatch_queue_wakeup_global**

```
int
pthread_workqueue_additem_np (
     pthread_workqueue_t workq,
     void *( *workitem_func)(void *), void * workitem_arg,
     pthread_workitem_handle_t * itemhandlep, unsigned int *gencountp)
```

Send to thread pool

_dispatch_wakeup(*root queue*)

↳ vtable->do_probe - - - - - - - -► **_dispatch_queue_wakeup_global**

```
int
pthread_workqueue_additem_np (
    pthread_workqueue_t workq,
    void *( *workitem_func)(void *), void * workitem_arg,
    pthread_workitem_handle_t * itemhandlep, unsigned int *gencountp)
```

_dispatch_worker_thread2

```
↳ while ((item = fastpath(_dispatch_queue_concurrent_drain_one(dq))))
          _dispatch_continuation_pop(item);
```

Executing

- _dispatch_continuation_pop
  - Is a "dispatch_continuation_s" ?
    - ➔ Process flag: DISPATCH_OBJ_ASYNC_BIT
    - ➔ Process flag: DISPATCH_OBJ_GROUP_BIT
    - ➔ dc->dc_func(dc->dc_ctxt)
  - Or is a "dispatch_queue_s"?
    - ➔ Run _dispatch_queue_invoke
      1. Check SUSPEND state and try to acquire *queue lock*
      2. _dispatch_queue_drain
      3. Release *queue lock*
      4. Release *object lock* (locked in _dispatch_wakeup)

When wake up queues?

- push to an empty queue
- dq_running is 0
- _dispatch_queue_wakeup_global in _dispatch_queue_concurrent_drain_one (fork more working threads)

Implementation of thread pool

- Use Darwin's extension to POSIX threads
  - ➔ Create thread pool: pthread_workqueue_create_np
  - ➔ Adjust pool size by the overall load on the system
  - ➔ Add a job: pthread_workqueue_additem_np
- Built-in lightweight implementation
  - Pool size: dgq_thread_pool_size
  - Worker function: _dispatch_worker_thread
  - When all jobs complete, working thread will sleep on a signal several seconds, unless be waken up or quit on timeout

Other implementation technique

- Two reference counts
  - Internal reference count (do_ref_cnt)
  - External reference count (do_xref_cnt) – Better error detection for client code
- An simple but efficient memory allocation cache
  - Only cache dispatch_continuation_t
  - Per-thread, single link
  - Only flush cache on some points, usually when a working thread finishes all jobs
- fastpath, slowpath

Port to Linux

- By Mark Heily
- http://packages.debian.org/squeeze/libdispatch0
- Related libraries:
  1. libkqueue (implement kevent on top of epoll, inotify, signalfd and timerfd)
  2. libpthread_workqueue (implement pthread_workqueue in userspace)

END