零声学院

www.0voice.com

一切只为渴望更优秀的你！

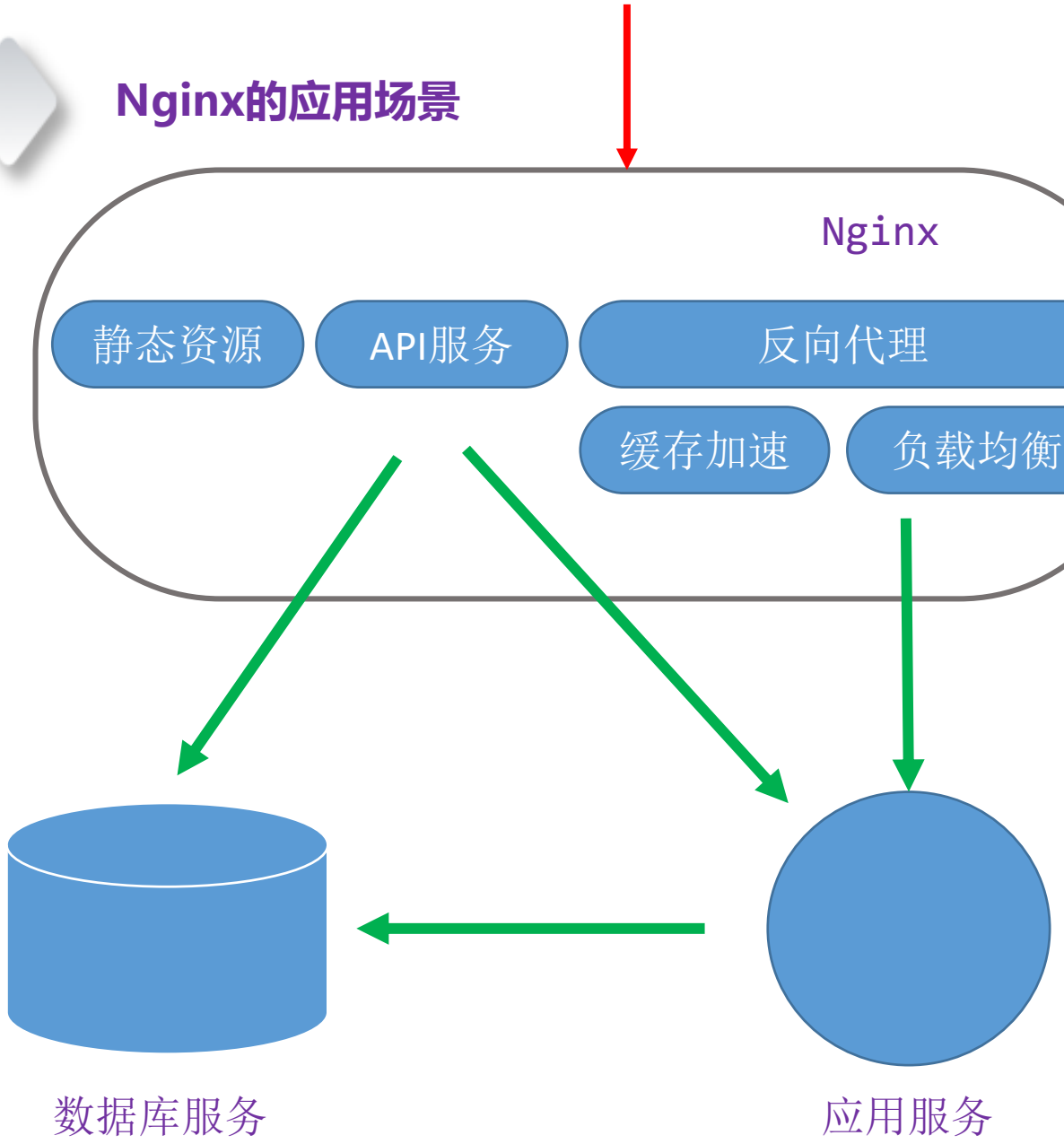
线程池原理与实现

技术交流 King老师 3147964070
往期视频 秋香老师 2207032995
课程咨询 依依老师 2693590767

课程提纲

- Nginx为什么需要线程池
- 线程池的原理
- 线程池的实现
- 线程池的升级与定制
- 架构师的知识树

Nginx的应用场景



静态资源服务

- 本地文件系统提供服务

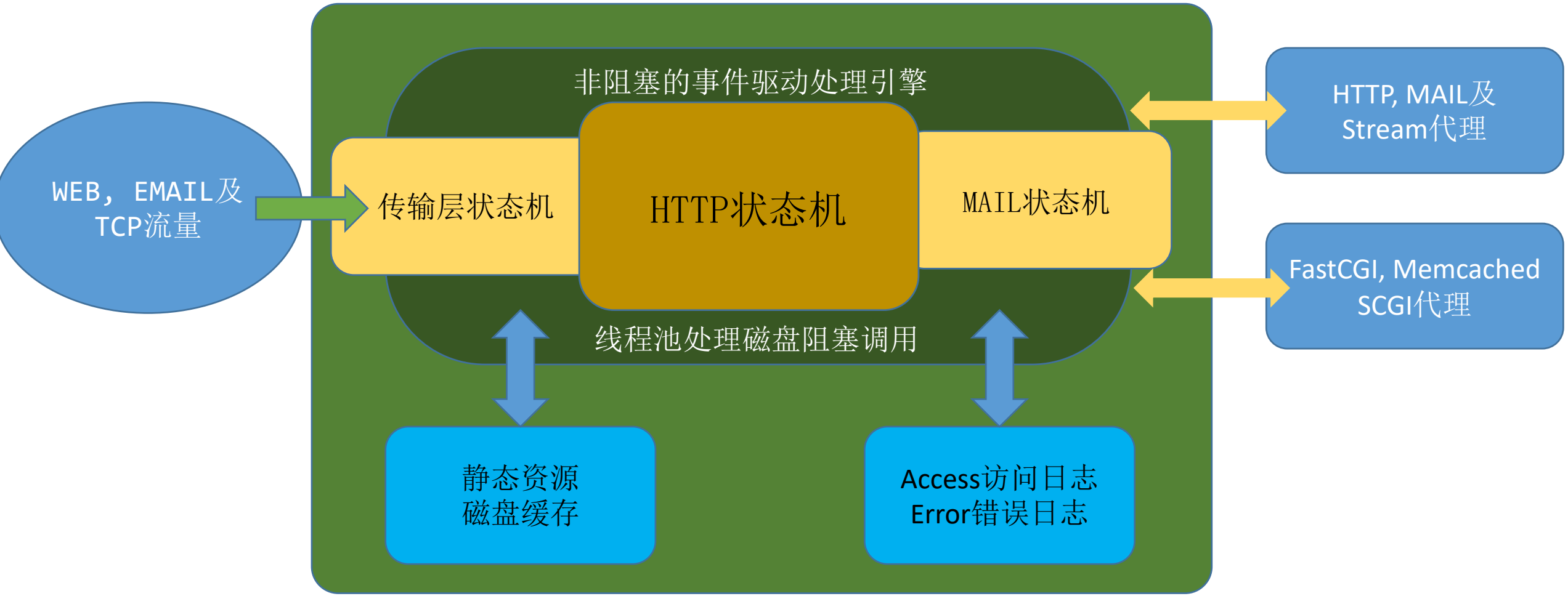
反向代理服务

- Nginx强大的性能
- 缓存
- 负载均衡

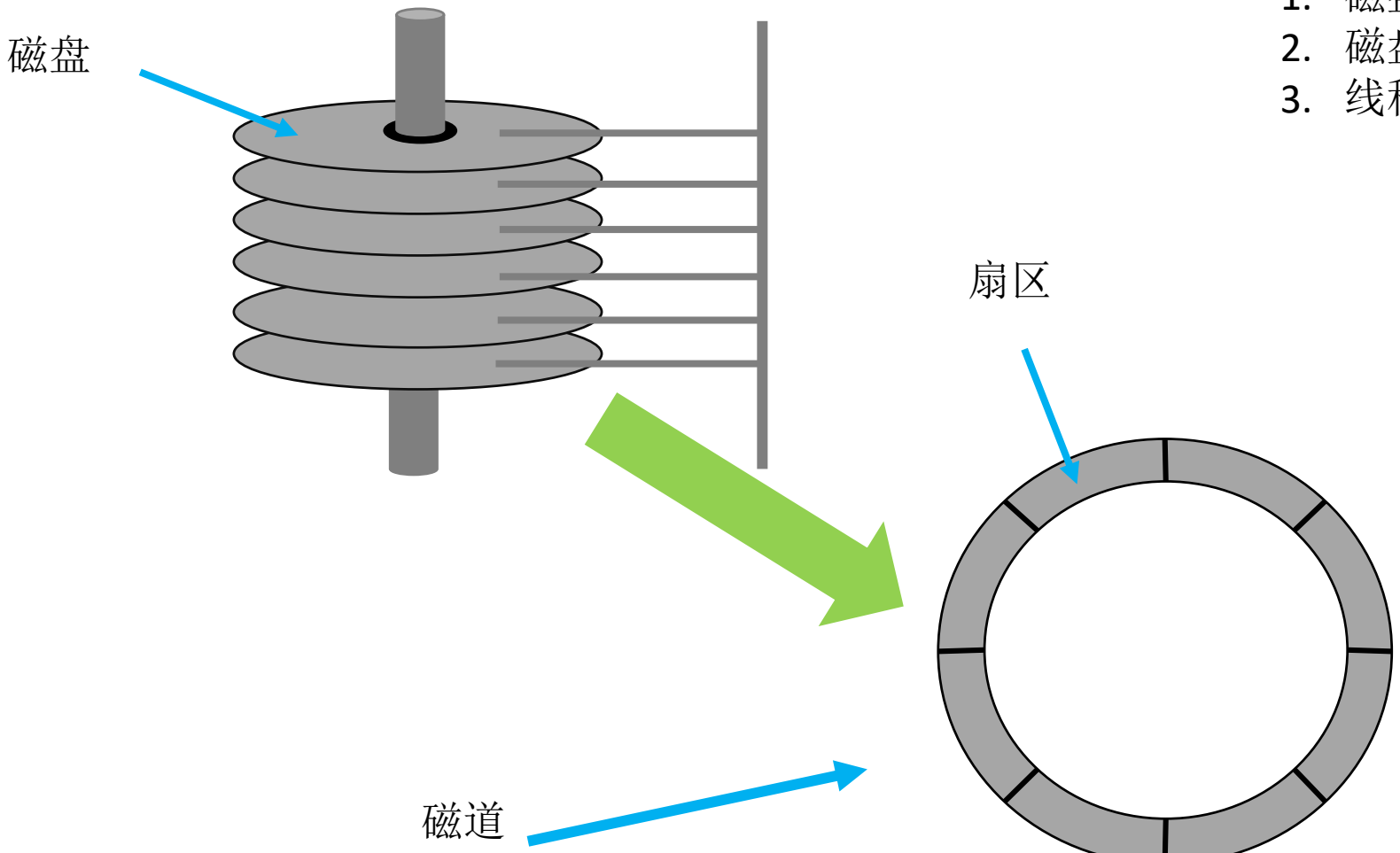
API服务

- OpenResty

Nginx请求处理流程

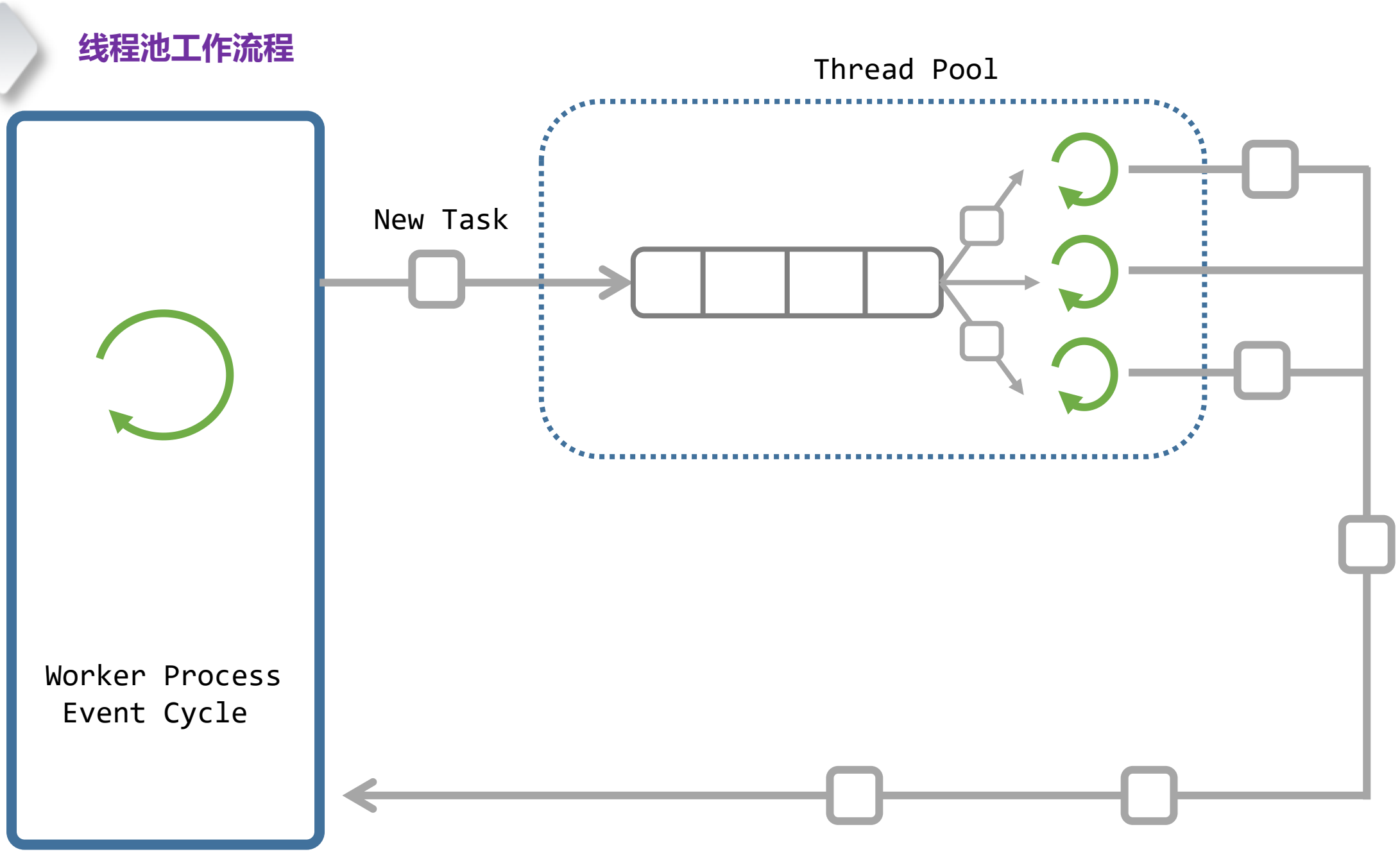


磁盘结构



1. 磁盘的工作原理？
2. 磁盘IO为什么阻塞？
3. 线程池的作用？

线程池工作流程



线程池的功能组件

King老师编程实现中
Coding ……

线程池的功能组件

```c
typedef struct NWORKER {
    pthread_t thread;
    int terminate;
    struct NWORKQUEUE *workqueue;
    struct NWORKER *prev;
    struct NWORKER *next;
} nWorker;
```

1. 执行（线程ID，终止标识，池管理组件对象）

```c
typedef struct NJOB {
    void (*job_function)(struct NJOB *job);
    void *user_data;
    struct NJOB *prev;
    struct NJOB *next;
} nJob;
```

2. 任务（任务回调函数，任务执行的参数）

```c
typedef struct NWORKQUEUE {
    struct NWORKER *workers;
    struct NJOB *waiting_jobs;
    pthread_mutex_t jobs_mtx;
    pthread_cond_t jobs_cond;
} nWorkQueue;
```

4. 池管理组件（互斥锁，条件变量，执行队列，任务队列）

线程池的创建

```c
int ntyThreadPoolCreate(nThreadPool *workqueue, int numWorkers) {

    if (numWorkers < 1) numWorkers = 1;
    memset(workqueue, 0, sizeof(nThreadPool));

    pthread_cond_t blank_cond = PTHREAD_COND_INITIALIZER;
    memcpy(&workqueue->jobs_cond, &blank_cond, sizeof(workqueue->jobs_cond));

    pthread_mutex_t blank_mutex = PTHREAD_MUTEX_INITIALIZER;
    memcpy(&workqueue->jobs_mtx, &blank_mutex, sizeof(workqueue->jobs_mtx));

    int i = 0;
    for (i = 0;i < numWorkers;i ++) {
        nWorker *worker = (nWorker*)malloc(sizeof(nWorker));
        if (worker == NULL) {
            perror("malloc");
            return 1;
        }

        memset(worker, 0, sizeof(nWorker));
        worker->workqueue = workqueue;

        int ret = pthread_create(&worker->thread, NULL, ntyWorkerThread, (void *)worker);
        if (ret) {

            perror("pthread_create");
            free(worker);

            return 1;
```

线程池的销毁

```c
void ntyThreadPoolShutdown(nThreadPool *workqueue) {
    nWorker *worker = NULL;

    for (worker = workqueue->workers;worker != NULL;worker = worker->next) {
        worker->terminate = 1;
    }

    pthread_mutex_lock(&workqueue->jobs_mtx);

    workqueue->workers = NULL;
    workqueue->waiting_jobs = NULL;

    pthread_cond_broadcast(&workqueue->jobs_cond);

    pthread_mutex_unlock(&workqueue->jobs_mtx);

}
```

线程池的push任务

```
void ntyThreadPoolQueue(nThreadPool *workqueue, nJob *job) {

    pthread_mutex_lock(&workqueue->jobs_mtx);

    LL_ADD(job, workqueue->waiting_jobs);

    pthread_cond_signal(&workqueue->jobs_cond);
    pthread_mutex_unlock(&workqueue->jobs_mtx);

}
```

线程池的线程任务

```c
static void *ntyWorkerThread(void *ptr) {
    nWorker *worker = (nWorker*)ptr;

    while (1) {
        pthread_mutex_lock(&worker->workqueue->jobs_mtx);

        while (worker->workqueue->waiting_jobs == NULL) {
            if (worker->terminate) break;
            pthread_cond_wait(&worker->workqueue->jobs_cond, &worker->workqueue->jobs_mtx);
        }

        if (worker->terminate) {
            pthread_mutex_unlock(&worker->workqueue->jobs_mtx);
            break;
        }

        nJob *job = worker->workqueue->waiting_jobs;
        if (job != NULL) {
            LL_REMOVE(job, worker->workqueue->waiting_jobs);
        }

        pthread_mutex_unlock(&worker->workqueue->jobs_mtx);

        if (job == NULL) continue;

        job->job_function(job);
    }
```

线程池的测试程序

```c
void king_counter(nJob *job) {

    int index = *(int*)job->user_data;

    printf("index : %d, selfid : %lu\n", index, pthread_self());

    free(job->user_data);
    free(job);
}

int main(int argc, char *argv[]) {

    nThreadPool pool;

    ntyThreadPoolCreate(&pool, KING_MAX_THREAD);

    int i = 0;
    for (i = 0;i < KING_COUNTER_SIZE;i ++) {
        nJob *job = (nJob*)malloc(sizeof(nJob));
        if (job == NULL) {
            perror("malloc");
            exit(1);
        }

        job->job_function = king_counter;
        job->user_data = malloc(sizeof(int));
        *(int*)job->user_data = i;

        ntyThreadPoolQueue(&pool, job);
```

实践思考题

1. 为线程池实现增加线程？
2. 为线程池减少线程？

模块封装思路

```c
typedef struct _ThreadPool {
    const void *_;
    nThreadPool *wq;
} ThreadPool;

typedef struct _ThreadPoolOpera {

    size_t size;
    void* (*ctor)(void *_self, va_list *params);
    void* (*dtor)(void *_self);
    void (*addJob)(void *_self, void *task);

} ThreadPoolOpera;
```

```c
void *New(const void *_class, ...) {
    const AbstractClass *class = _class;
    void *p = calloc(1, class->size);
    memset(p, 0, class->size);

    assert(p);
    *(const AbstractClass**)p = class;

    if (class->ctor) {
        va_list params;
        va_start(params, _class);
        p = class->ctor(p, &params);
        va_end(params);
    }
    return p;
}


void Delete(void *_class) {
    const AbstractClass **class = _class;

    if (_class && (*class) && (*class)->dtor) {
        _class = (*class)->dtor(_class);
    }

    free(_class);
}
```