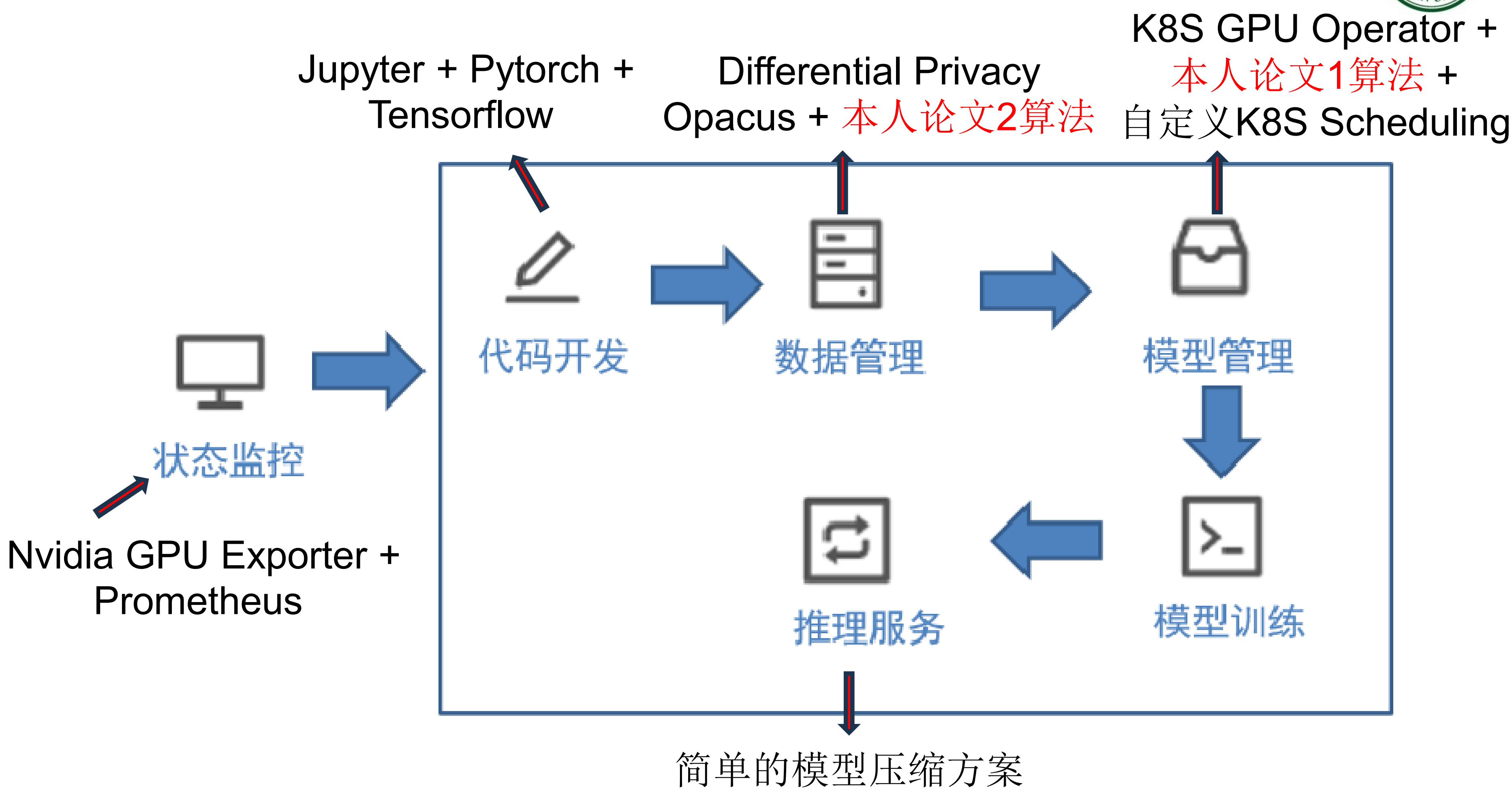
简历简要补充图片

肖霖畅

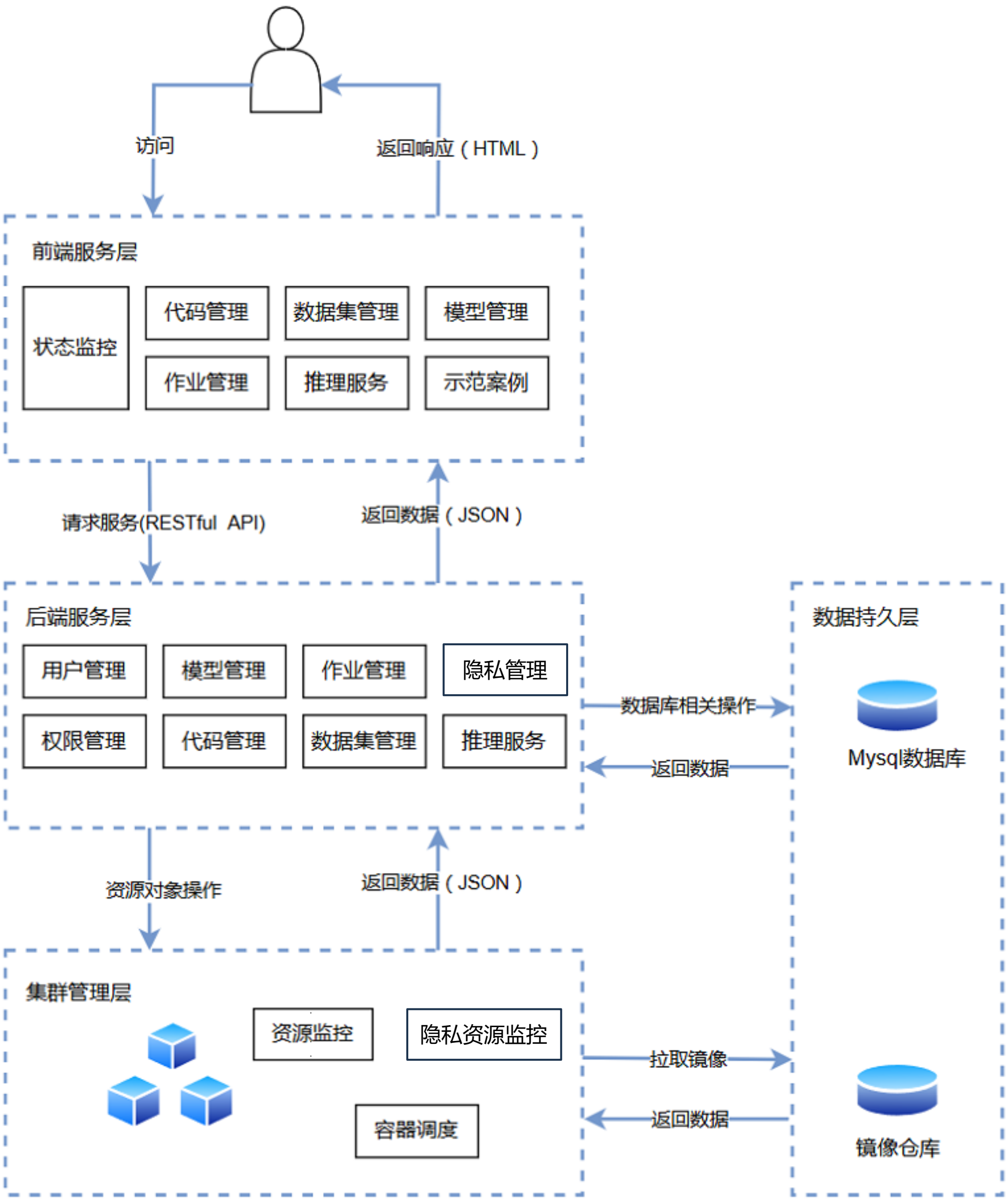
大纲

1. **DeepAI架构**
2. 基于异构超算的多元任务运行时系统
3. 实习经历相关

DeepAI的核心任务流



DeepAI的集群系统架构

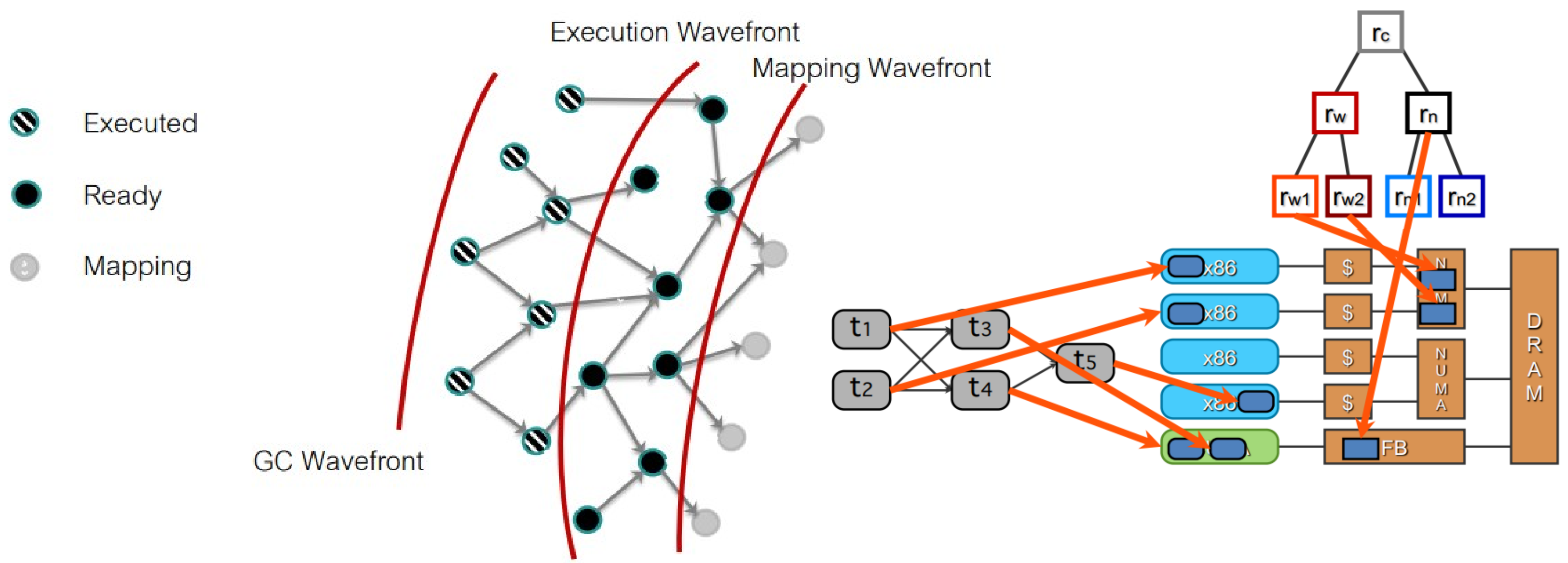


大纲

1. DeepAI
2. **基于异构超算的多元任务运行时系统**
3. 实习经历相关

运行时系统调度概述

- 调度策略
  - 需要决定：Task在哪里执行；Data在哪里放置
  - 用户可以自己决定调度具体策略



运行时系统的用户界面例: 并行斐波那契

```
enum TaskIDs {
  TOP_LEVEL_TASK_ID,
  FIBONACCI_TASK_ID,
  SUM_TASK_ID,
};
```

第一步，注册任务的唯一ID

```
int main(int argc, char **argv)
{
  Runtime::set_top_level_task_id(top_id: TOP_LEVEL_TASK_ID);

  {
    TaskVariantRegistrar registrar(task_id: TOP_LEVEL_TASK_ID, variant_name: "top_level");
    registrar.add_constraint(constraint: ProcessorConstraint(kind: Processor::LOC_PROC));
    Runtime::preregister_task_variant<top_level_task>(registrar, task_name: "top_level");
  }

  {
    TaskVariantRegistrar registrar(task_id: FIBONACCI_TASK_ID, variant_name: "fibonacci");
    registrar.add_constraint(constraint: ProcessorConstraint(kind: Processor::LOC_PROC));
    Runtime::preregister_task_variant<int, fibonacci_task>(registrar, task_name: "fibonacci");
  }

  {
    TaskVariantRegistrar registrar(task_id: SUM_TASK_ID, variant_name: "sum");
    registrar.add_constraint(constraint: ProcessorConstraint(kind: Processor::LOC_PROC));
    registrar.set_leaf(is_leaf: true);
    Runtime::preregister_task_variant<int, sum_task>(registrar, task_name: "sum");
  }

  // Callback for registering the inline mapper
  Runtime::add_registration_callback(callback: mapper_registration);

  return Runtime::start(argc, argv);
}
```

第二步，设置根任务

第三步，注册每个任务和对应的函数名，给任务增加计算资源（即处理器的约束条件）

第四步，开始执行前的回调注册（自定义调度方案、注册运行时变量等）

第五步，启动并行程序执行

运行时系统的用户界面例: 并行斐波那契

```
void top_level_task(const Task *task,
                    const std::vector<PhysicalRegion> &regions,
                    Context ctx, Runtime *runtime) {
    int num_fibonacci = 15;
    const InputArgs &command_args = Runtime::get_input_args();
    for (int i = 1; i < command_args.argc; i++) {
        // Skip any legion runtime configuration parameters
        if (command_args.argv[i][0] == '-') {
            i++;
            continue;
        }

        num_fibonacci = atoi(nptr: command_args.argv[i]);
        assert(num_fibonacci >= 0);
        break;
    }
    printf(format: "Computing the first %d Fibonacci numbers...\n", num_fibonacci);

    std::vector<Future> fib_results;

    Future fib_start_time = runtime->get_current_time(ctx);
    std::vector<Future> fib_finish_times;

    for (int i = 0; i < num_fibonacci; i++) {
        TaskLauncher launcher(tid: FIBONACCI_TASK_ID, arg: TaskArgument(arg: &i, argsize: sizeof(i)));
        fib_results.push_back(x: runtime->execute_task(ctx, launcher));
        fib_finish_times.push_back(x: runtime->get_current_time(ctx, precondition: fib_results.back()));
    }

    for (int i = 0; i < num_fibonacci; i++) {
        int result = fib_results[i].get_result<int>();
        double elapsed = (fib_finish_times[i].get_result<double>() -
                          fib_start_time.get_result<double>());
        printf(format: "Fibonacci(%d) = %d (elapsed = %.9f s)\n", i, result, elapsed);
    }

    fib_results.clear();
}
```

返回Future

支持创建子任务

等待Future结果的返回并获取值

运行时系统的用户界面例: 并行斐波那契

斐波那契

```
int fibonacci_task(const Task *task,
                   const std::vector<PhysicalRegion> &regions,
                   Context ctx, Runtime *runtime) {
    assert(task->arglen == sizeof(int));
    int fib_num = *(const int *) task->args;
    if (fib_num == 0)
        return 0;
    if (fib_num == 1)
        return 1;

    // Launch fib-1
    const int fib1 = fib_num - 1;
    TaskLauncher t1(tid: FIBONACCI_TASK_ID, arg: TaskArgument(arg: &fib1, argsize: sizeof(fib1)));
    Future f1 = runtime->execute_task(ctx, launcher: t1);

    // Launch fib-2
    const int fib2 = fib_num - 2;
    TaskLauncher t2(tid: FIBONACCI_TASK_ID, arg: TaskArgument(arg: &fib2, argsize: sizeof(fib2)));
    Future f2 = runtime->execute_task(ctx, launcher: t2);

    TaskLauncher sum(tid: SUM_TASK_ID, arg: TaskArgument(arg: NULL, argsize: 0));
    sum.add_future(f: f1);
    sum.add_future(f: f2);
    Future result = runtime->execute_task(ctx, launcher: sum);

    return result.get_result<int>();
}
```

求和

```
int sum_task(const Task *task,
             const std::vector<PhysicalRegion> &regions,
             Context ctx, Runtime *runtime) {
    assert(task->futures.size() == 2);
    Future f1 = task->futures[0];
    int r1 = f1.get_result<int>();
    Future f2 = task->futures[1];
    int r2 = f2.get_result<int>();

    return (r1 + r2);
}
```

大纲

1 DeepAI

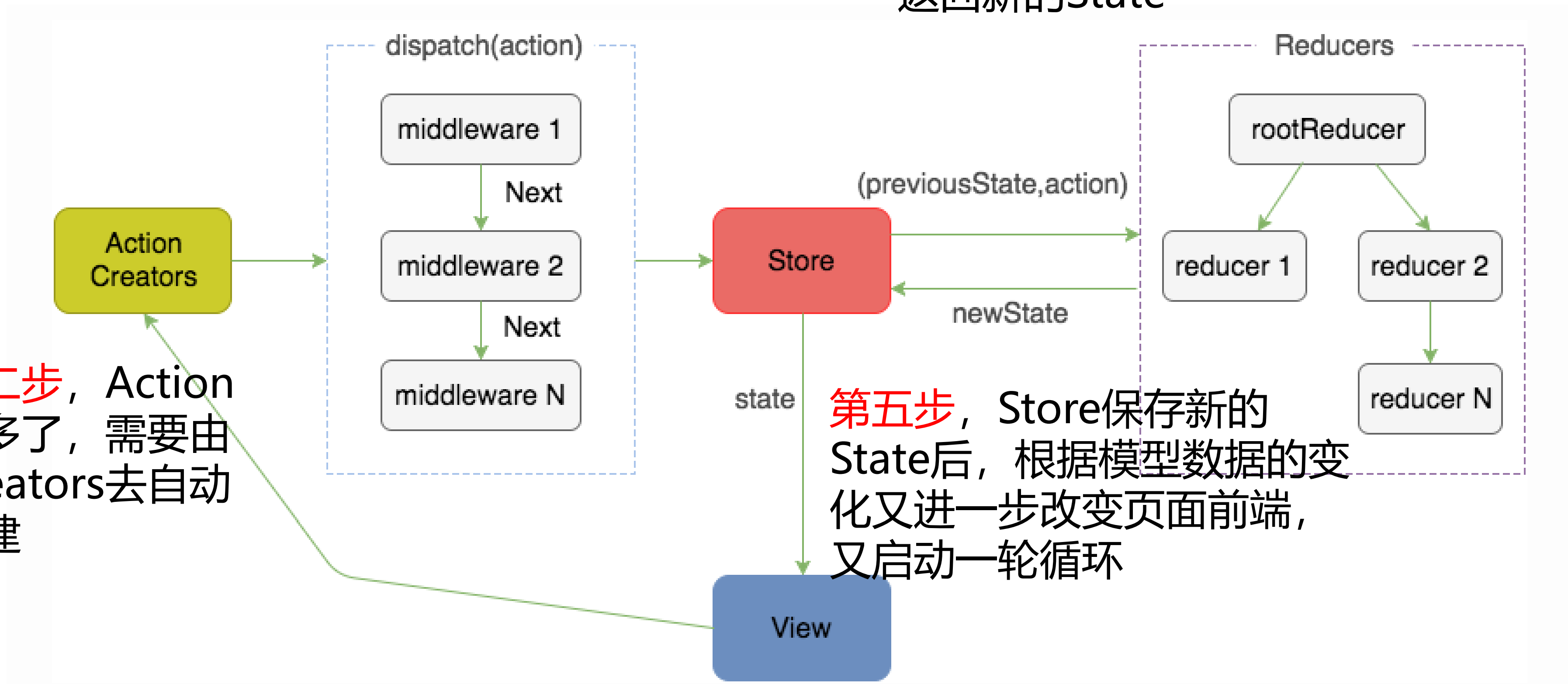
2 基于异构超算的多元任务运行时系统

3 **实习经历相关**

实习 – Redux设计模式(统一管理异步交互)

第三步（可选），修改Store前，由各部分中间件[日志、错误处理、API请求、路由等]进行预处理

第四步，Store收到Action后，发送给Reducers，经过判断逻辑后返回新的State



第二步，Action太多了，需要由Creators去自动创建

第五步，Store保存新的State后，根据模型数据的变化又进一步改变页面前端，又启动一轮循环

第一步，用户操作页面，需要更新页面的模型数据，用户不应该直接操作后端状态，因此用Action发送