JUNO离线软件数据管理系统的并行化

黄性涛，Irakli Chakaberia，李腾

山东大学

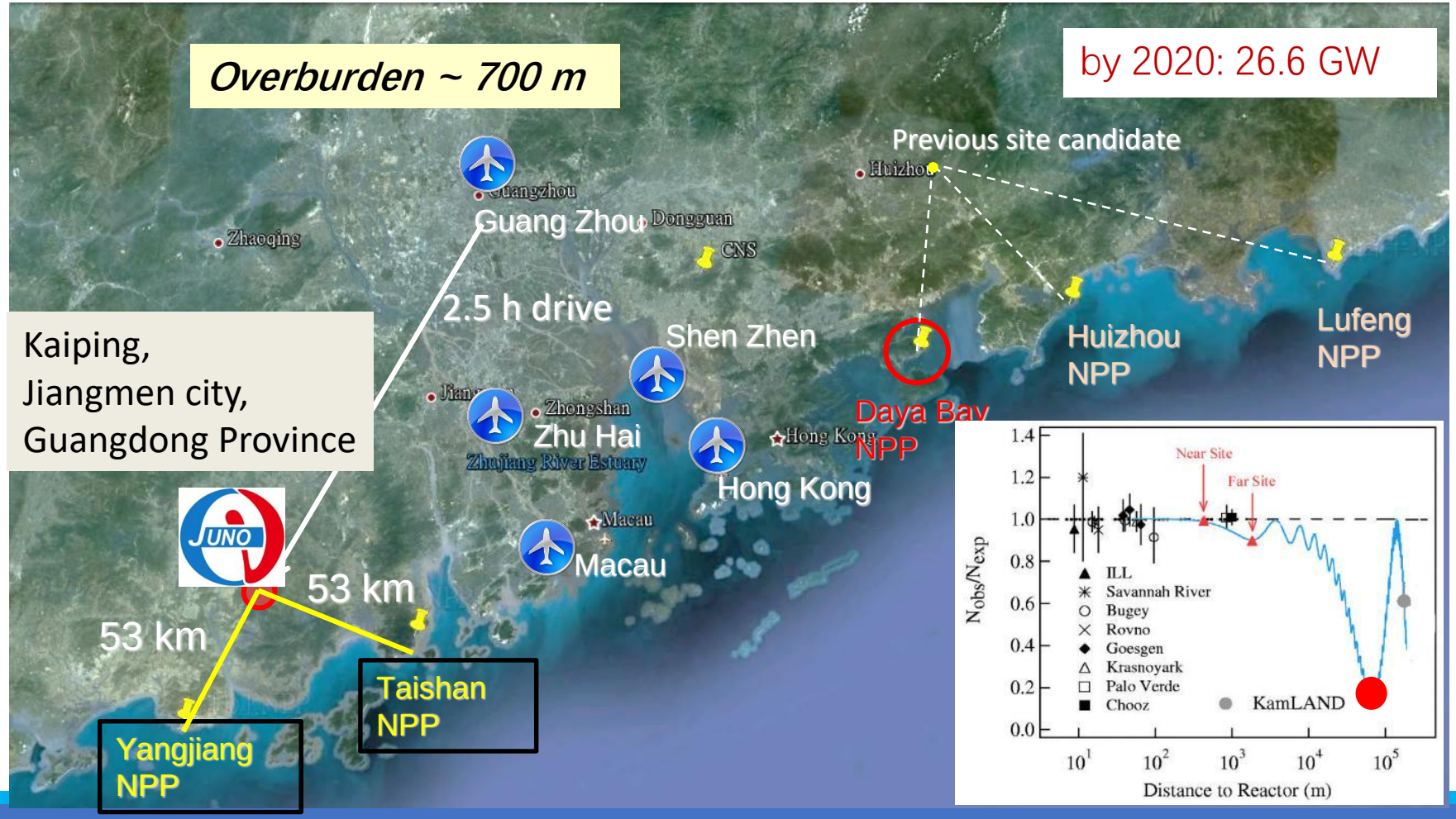
2017.6.6

大纲

- JUNO离线软件与数据管理系统
- 并行计算原型
- 设计方案讨论
- 总结与展望

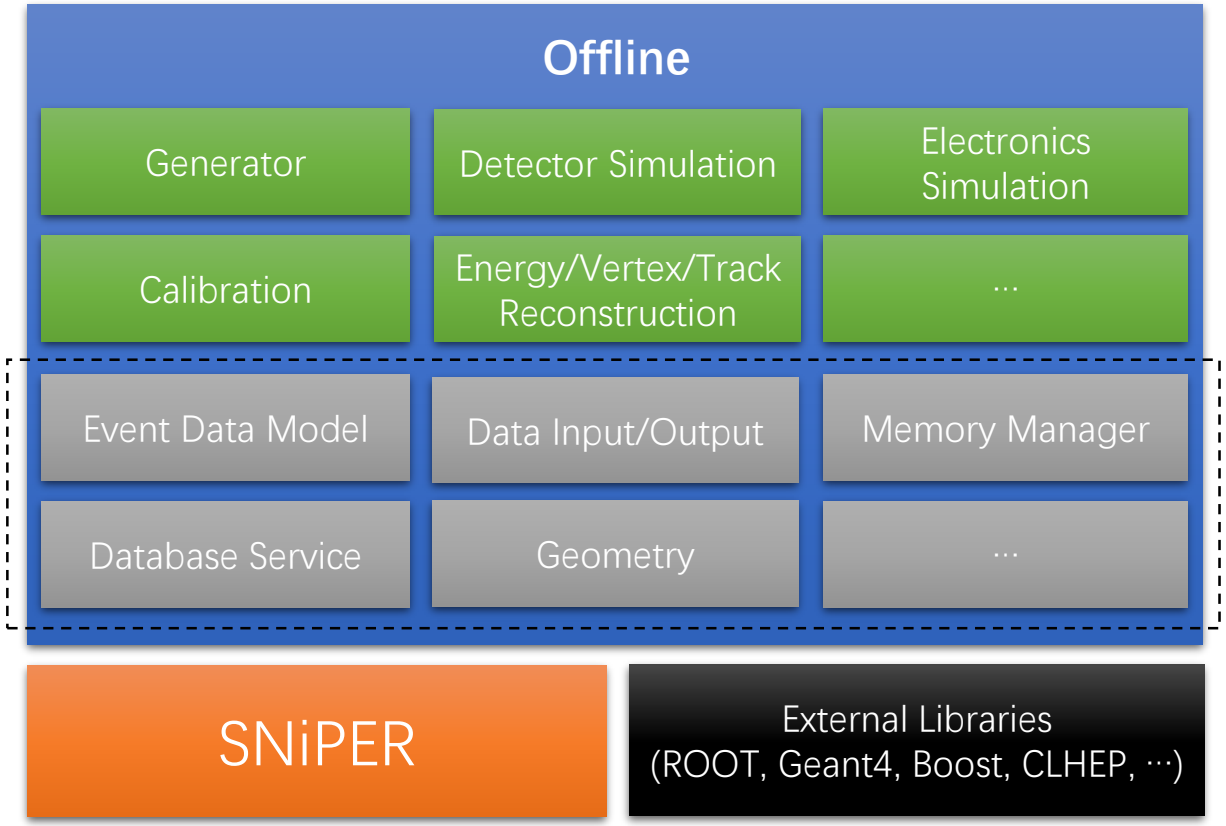
实验介绍

- **江门中微子实验（Jiangmen Underground Neutrino Observation, JUNO）**
  - 物理目标：测定中微子质量顺序
  - 预计于2020年开始取数
  - 数据量：~2 PB/Year



离线软件系统

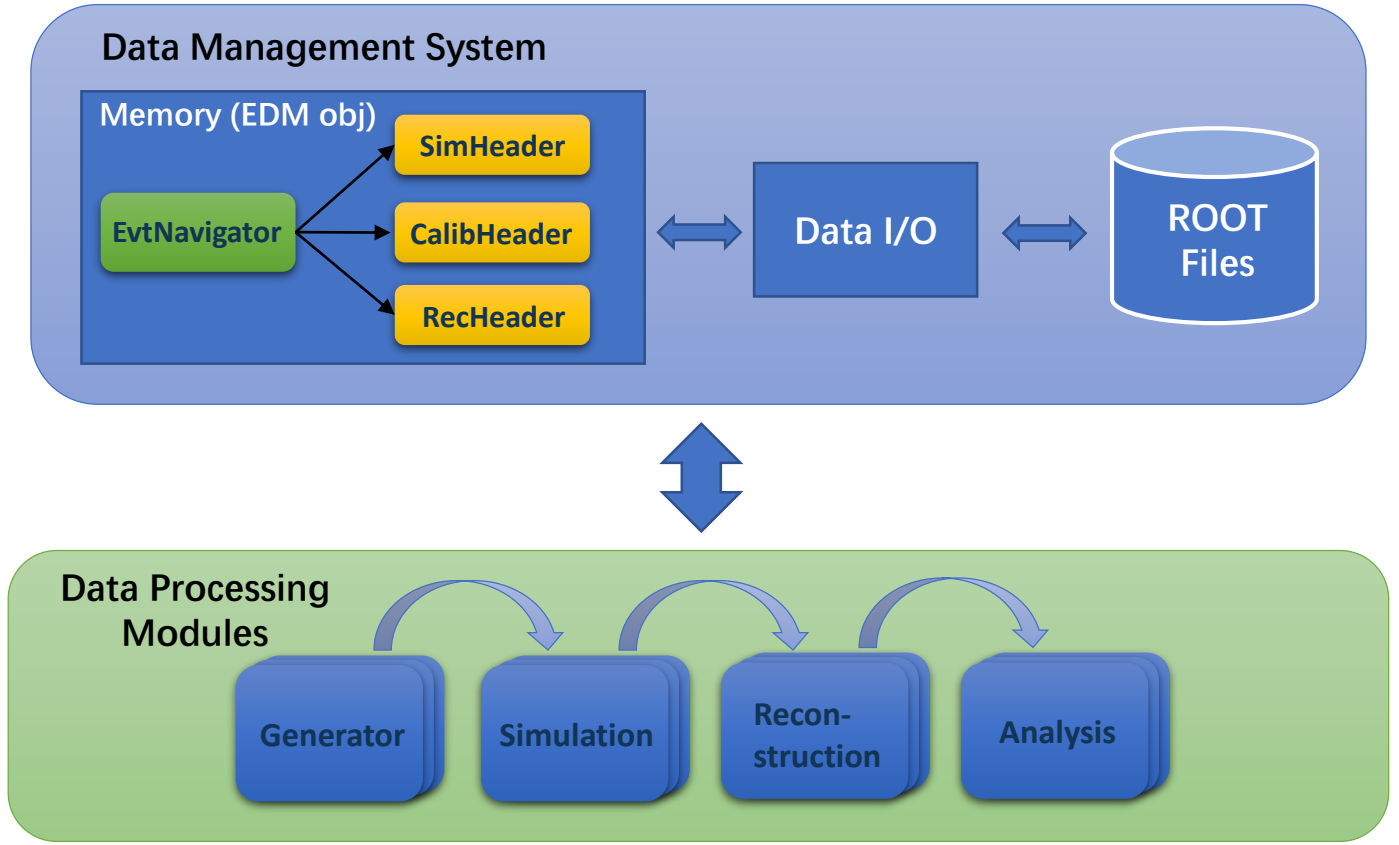
- **功能**
  - MC数据的产生（物理产生子、探测器模拟和电子学触发模拟等）
  - 真实数据的收集与处理（刻度与重建等）
  - 数据分析
- **构成**
  - **SNiPER：**
    - **底层框架**
  - **External Libraries:**
    - **第三方软件包和工具**
  - **Offline：**
    - **中间层，如事例模型、数据输入输出、内存管理和几何等**
    - **数据处理相关软件包，如产生子、探测器模拟、电子学模拟和重建等**



JUNO离线软件

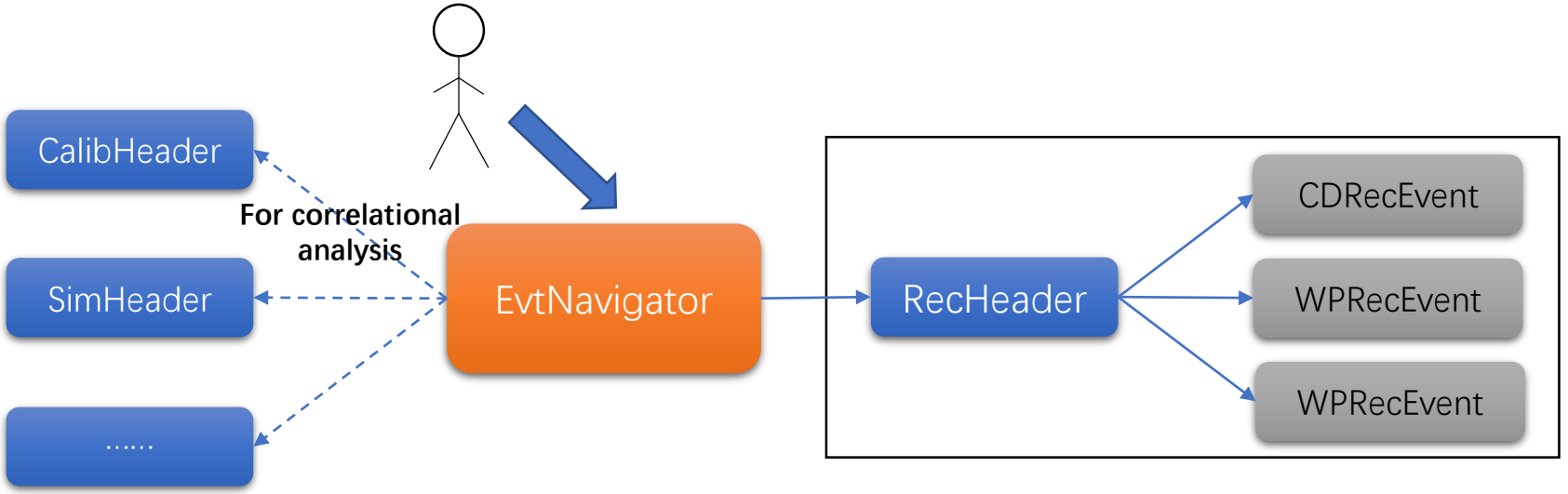
JUNO离线软件应用数据与算法分离设计

- 数据处理算法
  - 仅关心算法实现，不关心数据管理
- 数据管理系统
  - 数据输入输出系统
  - 内存管理系统
- 事例数据模型（EDM）
  - 定义数据（事例）在内存中组织的格式
  - 以C++类对象的形式保存数据



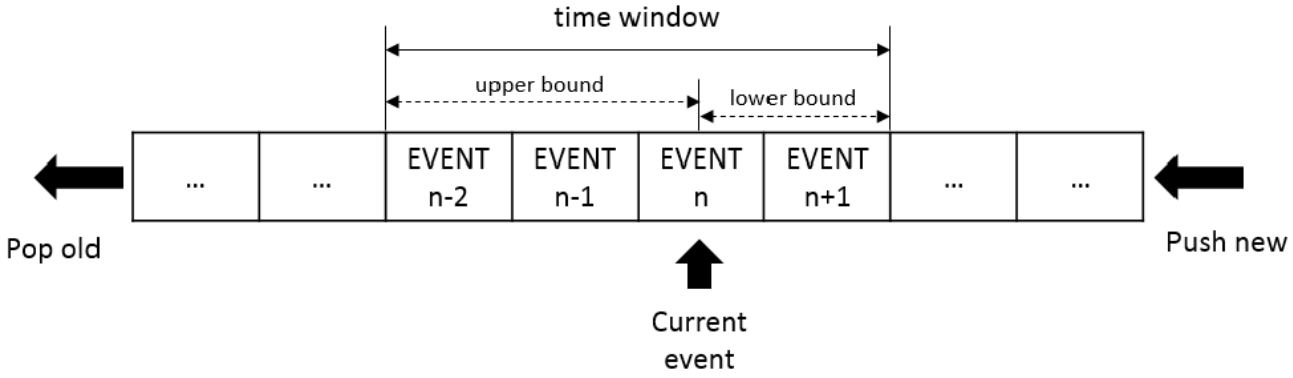
事例数据模型（EDM）

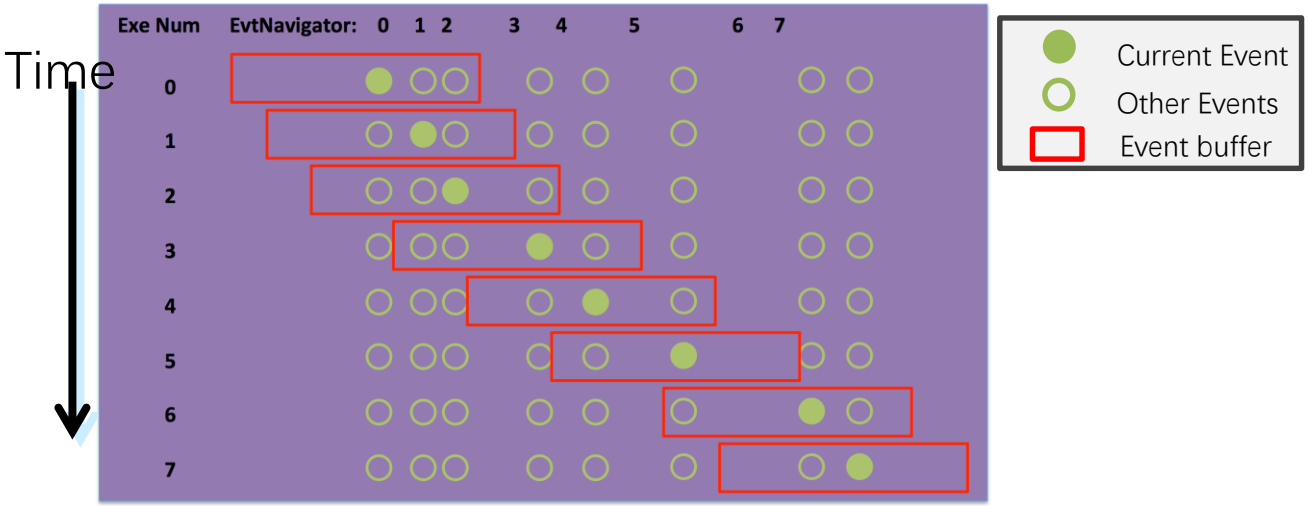
- **JUNO EDM的设计**
  - EvtNavigator作为数据的索引和入口
  - Header-Event双层设计
  - EvtNavigator-Header-Event的关联由**SmartRef**实现
    - 在I/O过程保持关联，实现延时（惰性）加载



内存管理

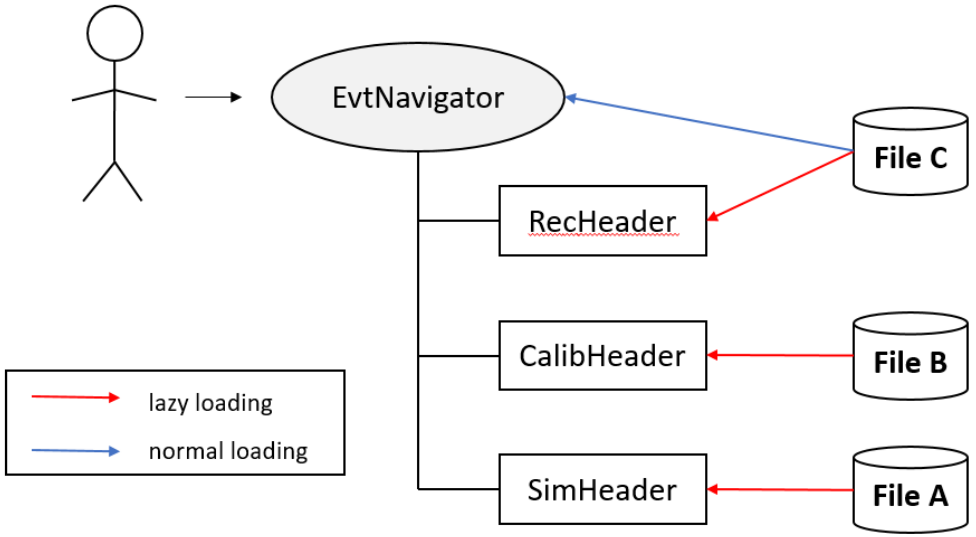
- **JUNO内存管理（DataBuffer）**
  - 为满足中微子信号的关联分析，DataBuffer设计为内置时间窗的双端队列
    - 按触发时间顺序，缓存多个物理事例
    - 旧事例从头部删除，新事例自尾部读入
    - 用户可配置时间窗的大小
  - 运行时，始终保持时间窗填满状态，用户可获取时间相邻的多个事例
  - DataBuffer内保存EvtNavigator



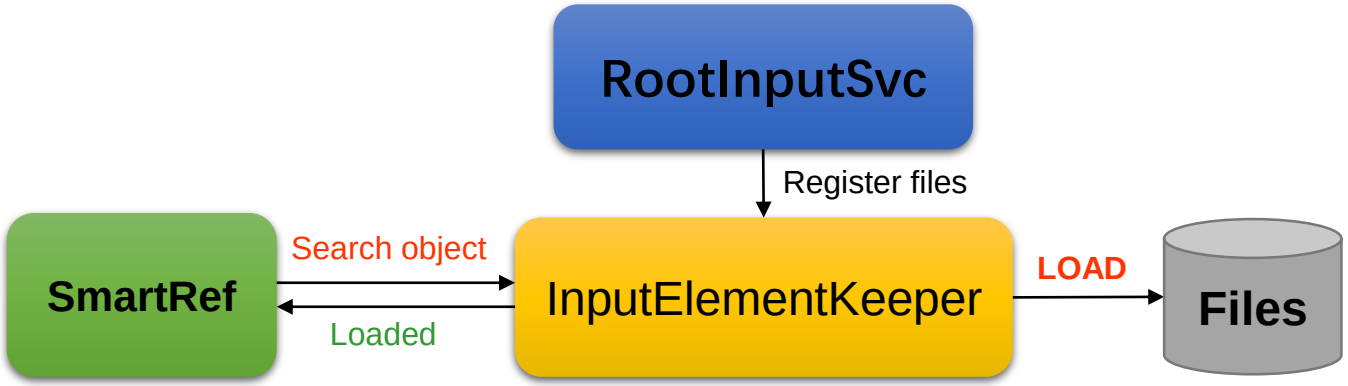


ROOT输入系统

- **功能：**
  - 输入EvtNavigator作为数据索引
  - 辅助SmartRef，实现延时加载

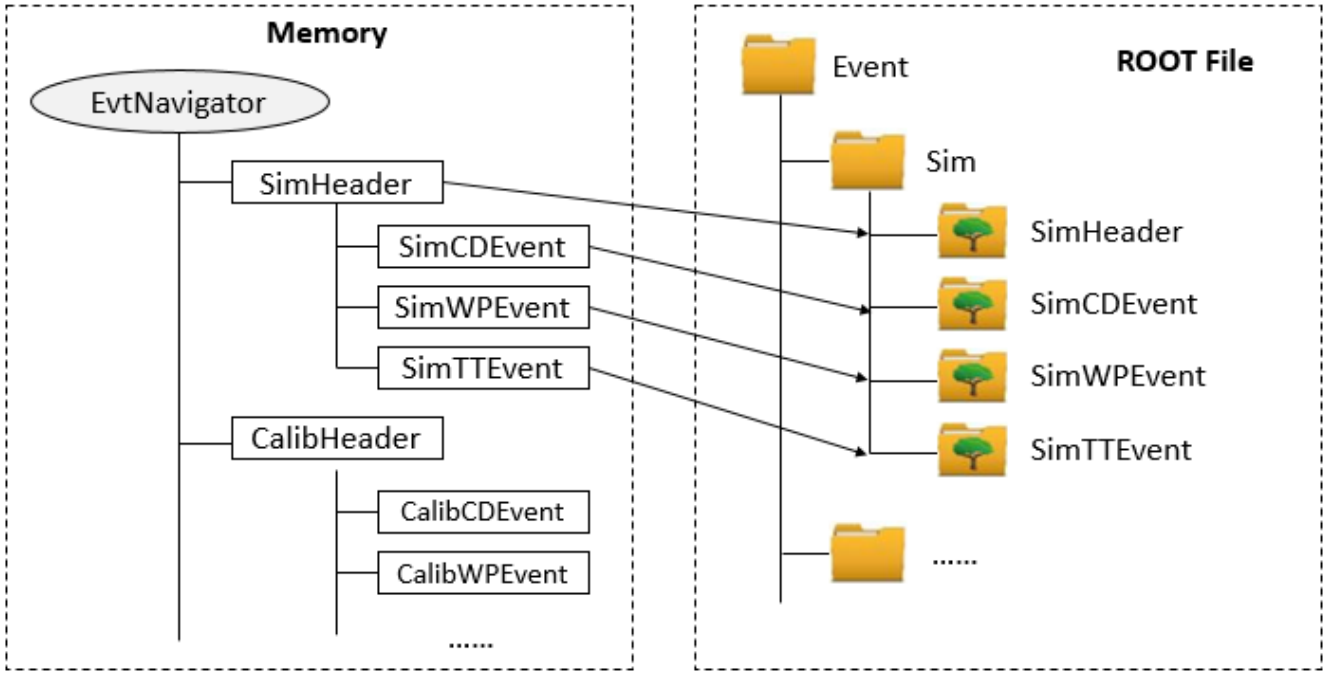


- **延时加载：**
  - 输入服务仅主动输入EvtNavigator，其所关联EDM对象仅在被访问时才载入内存
  - 由SmartRef配合输入服务（InputElementKeeper模块）实现
  - InputElementKeeper：单例类，由输入服务创建并初始化，保存输入文件元信息



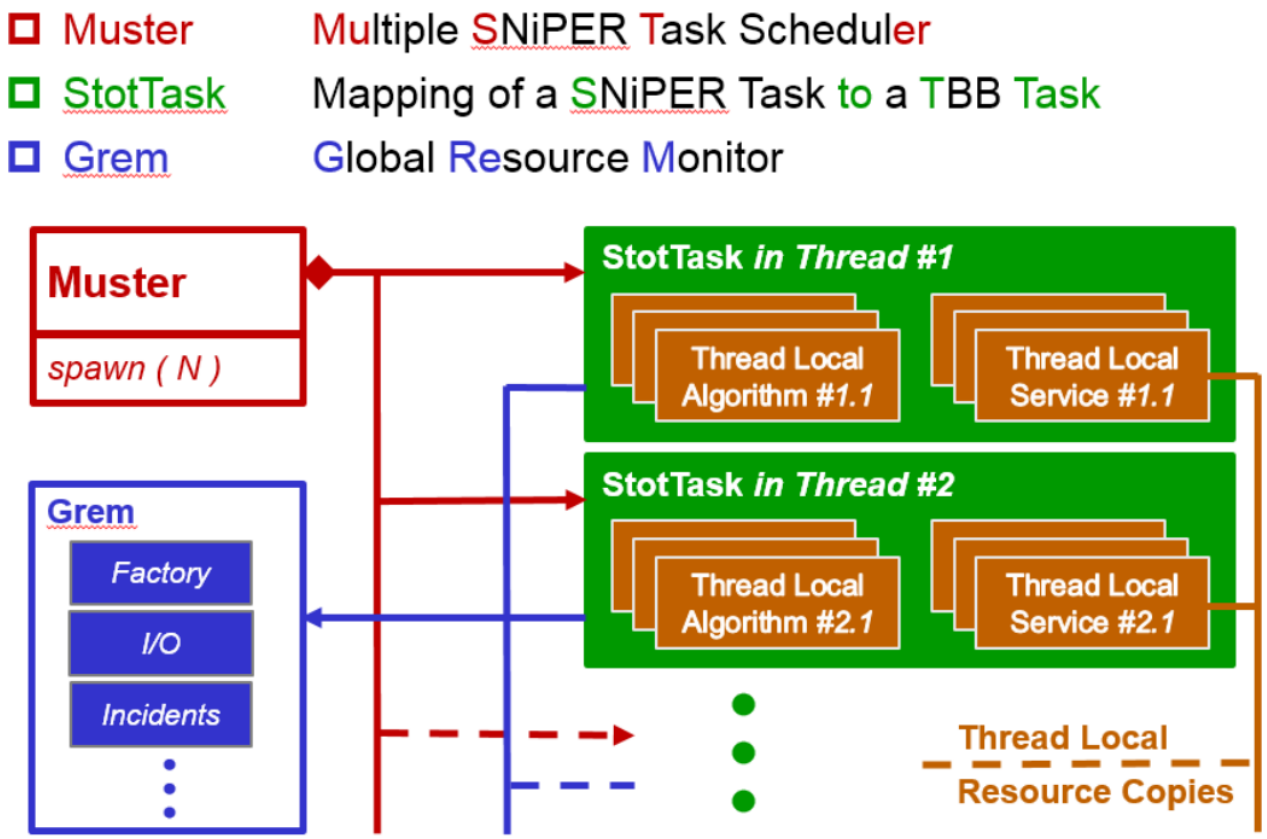
ROOT输出系统

- **功能**
  - 输出内存中的EDM对象
    - 直接写入TTree
  - “冗余”输出EvtNavigator
    - 写入TTree，作文文件索引保存
  - 生成文件元数据
    - 作业级元数据：几何、作业配置信息
    - 延时加载所需元数据



JUNO并行离线软件原型

- 并行化的JUNO离线软件
  - 合理利用多核CPU，提升整体性能
  - 以SniperMT作为底层框架
    - 基于Intel TBB开发
    - 映射TBB Task与SNiPER Task
    - 利用TBB task scheduler，获得合理的并行度和高效的线程调度策略
  - 目前正处于设计阶段



SniperMT设计

JUNO并行离线软件原型

- EDM及数据管理系统的并行化是重点及难点之一，需解决的问题包括：
  - EDM

    SmartRef：SmartRef通过全局的TProcessID对象访问关联对象，需保证其线程安全
  - 数据管理系统

    整体设计：在串行环境下，各SNiPER Task拥有独立的数据管理系统；并行环境下需对整体重新设计，以满足高效地数据处理
    - 减少线程的空闲时间
    - 满足JUNO特殊数据处理需求：数据关联分析、本底混合等

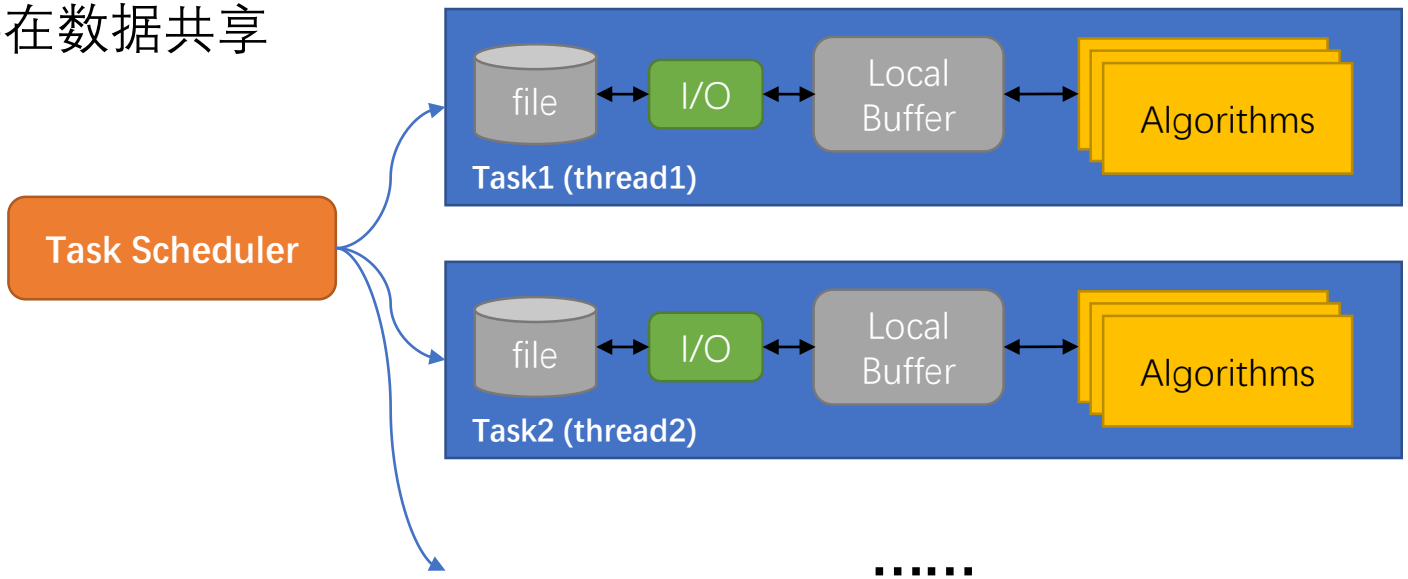
    内存管理：设计可由多线程共享的内存管理模块

    ROOT输入服务：保证InputElementKeeper模块（延时加载）的高效和线程安全

    ROOT输出服务：保证RootOutputFileManager、EDMManager等单例类的线程安全

设计方案讨论（一）

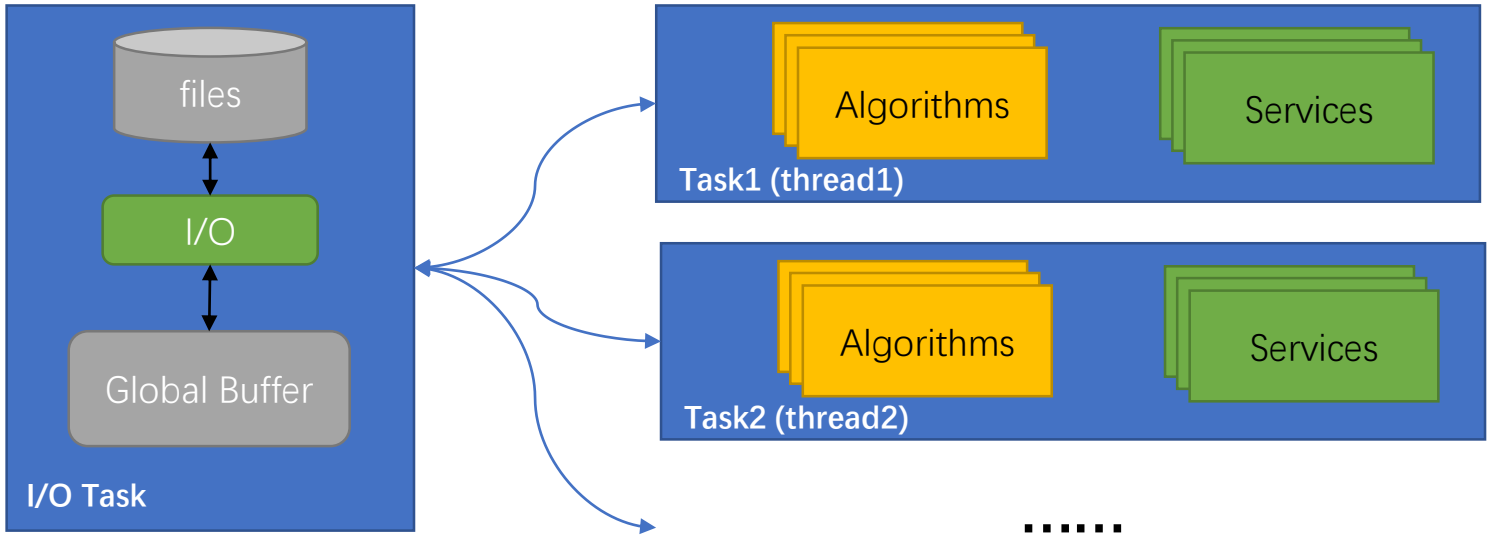
- 方案一：简单方案
  - 线程（Task）各自拥有独立的数据管理系统（输入输出文件、输入输出流和内存管理）
  - 线程间不存在数据共享



- 优缺点：
  - 设计、实现简单，用户易于使用
  - 性能存在明显瓶颈（输入文件数少于线程数时）

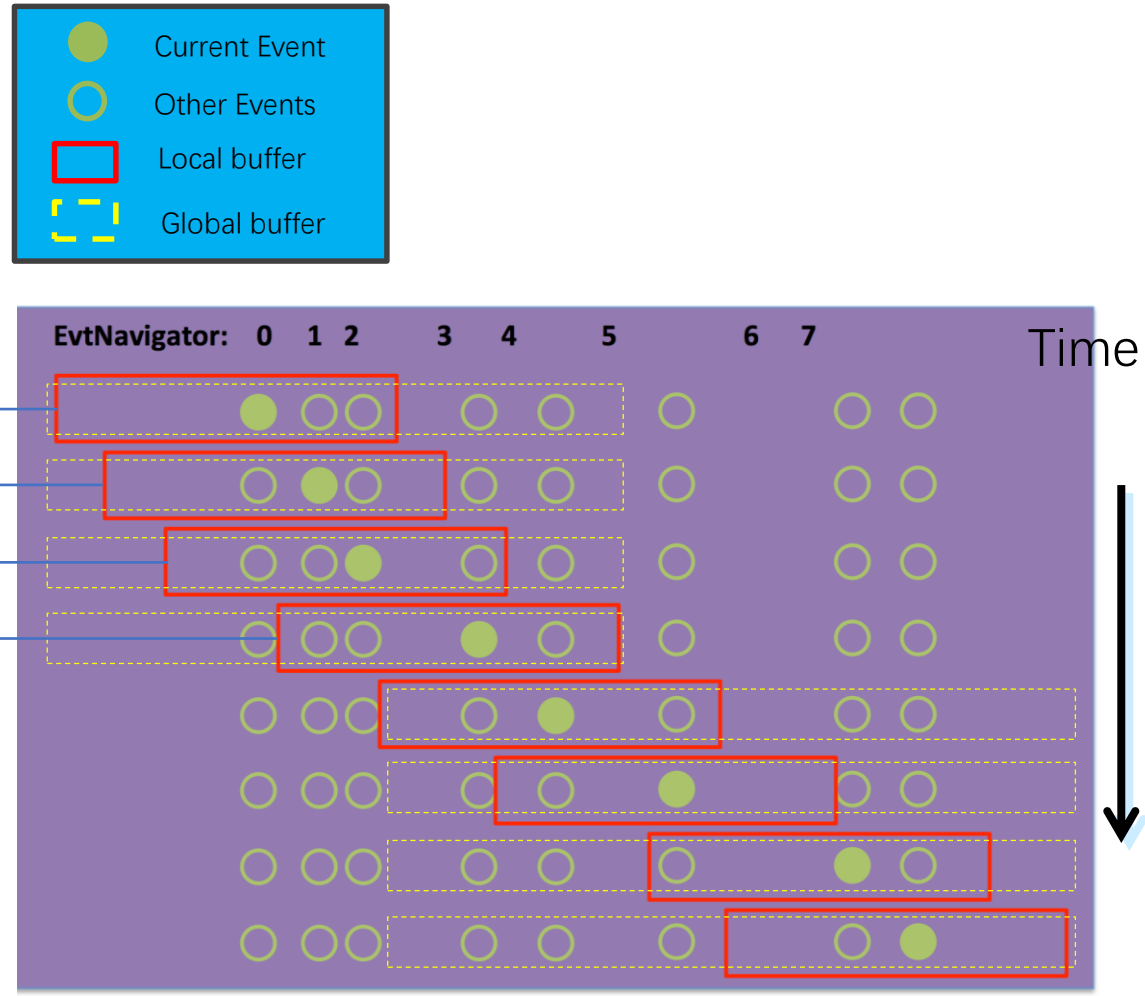
设计方案讨论（二）

- 方案二：
  - 由全局提供数据管理功能（输入输出文件、输入输出流和内存管理）
  - 全局内存管理模块向各个线程转发事例
  - 各线程处理后将输出数据归还统一输出



设计方案讨论（三）

- Global Buffer的设计（Proposed by Jiaheng）
  - Global Buffer向各个线程分发数据
  - 线程各自持有一个逻辑上的Buffer
  - 线程“拥有”当前事例，“共享”时间窗内其它事例
  - Global Buffer联合输入输出系统，保证事例的输入输出顺序
    - 设置合理的高低水位保证性能
- 一些问题：
  - 实现方案？std::list/tbb::concurrent_queue/ Boost.Circular Buffer
  - 一次性分发的事例数？



开发状态总结

- EDM & ROOT I/O
  - 使用互斥锁保护了可能产生竞争的函数
    - SmartRef
    - InputElementKeeper
    - EDMManager
    - RootOutputFileManager
  - 重构了ROOT I/O的部分模块，消除竞争
  - 基本可满足方案一要求
- 内存管理
  - 方案二的设计正处于讨论中

未来计划

- 近期完成方案一的实现与测试
- 完善EDM及ROOT I/O，尽量实现lock-free的设计
- 完成Global Buffer的设计，实现方案二

谢谢！