



**中山大學**  
SUN YAT-SEN UNIVERSITY



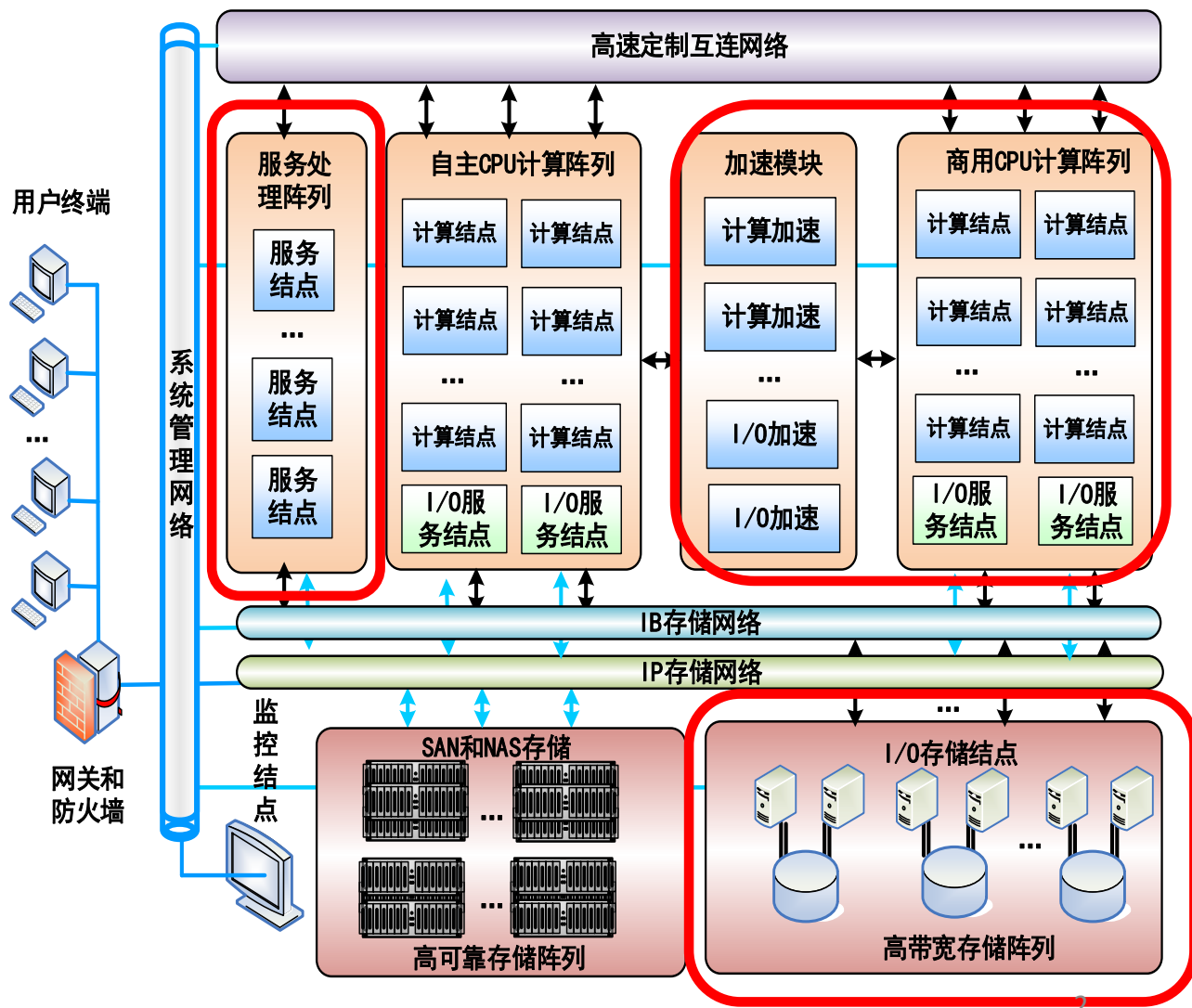
**国家超级计算广州中心**  
NATIONAL SUPERCOMPUTER CENTER IN GUANGZHOU

# 天河二号超级计算机 资源管理系统

**任课教师：吴迪、杜云飞**

# 天河二号逻辑架构

- 计算阵列
- 服务阵列
- 存储子系统
- 互联通信子系统
- 诊断监控子系统



# 天河二号逻辑结构

- **管理节点**

- mn0,mn1,.....
- 运行系统管理进程与支撑服务

- **登录节点**

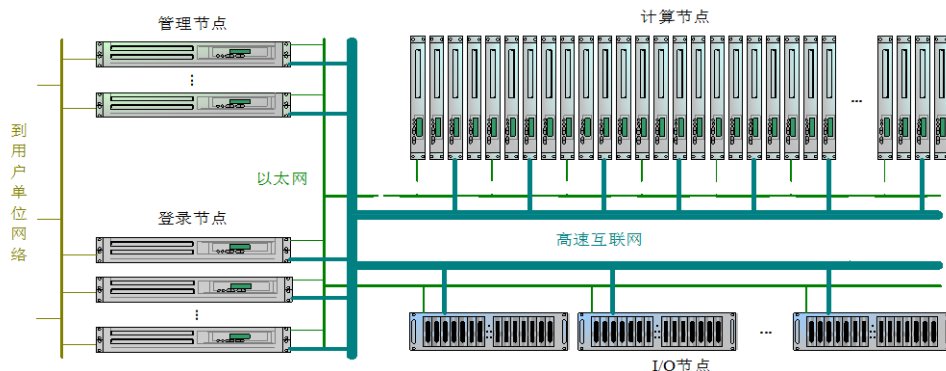
- ln0,ln1,.....
- 用户登录、编辑、编译、提交作业、结果分析

- **计算节点**

- cn0,cn1,.....
- 主要的计算资源，执行用户程序

- **I/O节点**

- 元数据服务器：mds0,mds1,..
- 对象存储：ost0,ost1,.....
- I/O增强：ion0,ion1,.....
- 提供存储服务
- 表现为全局共享文件系统



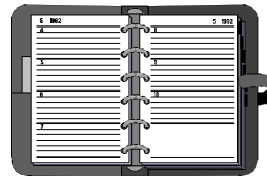
# Slurm资源管理系统概述

# 天河二号资源管理系统

- 基于开源软件SLURM定制
  - Simple Linux Utility for Resource Management from LLNL
  - 最大支持管理65,536个节点, 作业吞吐率120,000Job/Hour
- 提供高效的资源与作业管理
  - 状态监控
  - 资源管理
  - 作业调度
  - 用量记账
- 是用户使用计算资源的接口
  - 作业提交 / 运行
  - 任务加载
  - 作业控制
  - 状态查看

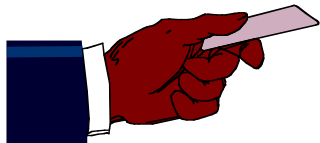
# Slurm资源管理系统

External scheduler manages the queue



SLURM allocates nodes, starts and manages the jobs

Job 1
Job 2
Job 3
Job 4



Users submit work

Node 0
Node 1
Node 2
Node 3
Node 4
Node 5
Node 6
Node 7

# Slurm资源管理系统架构

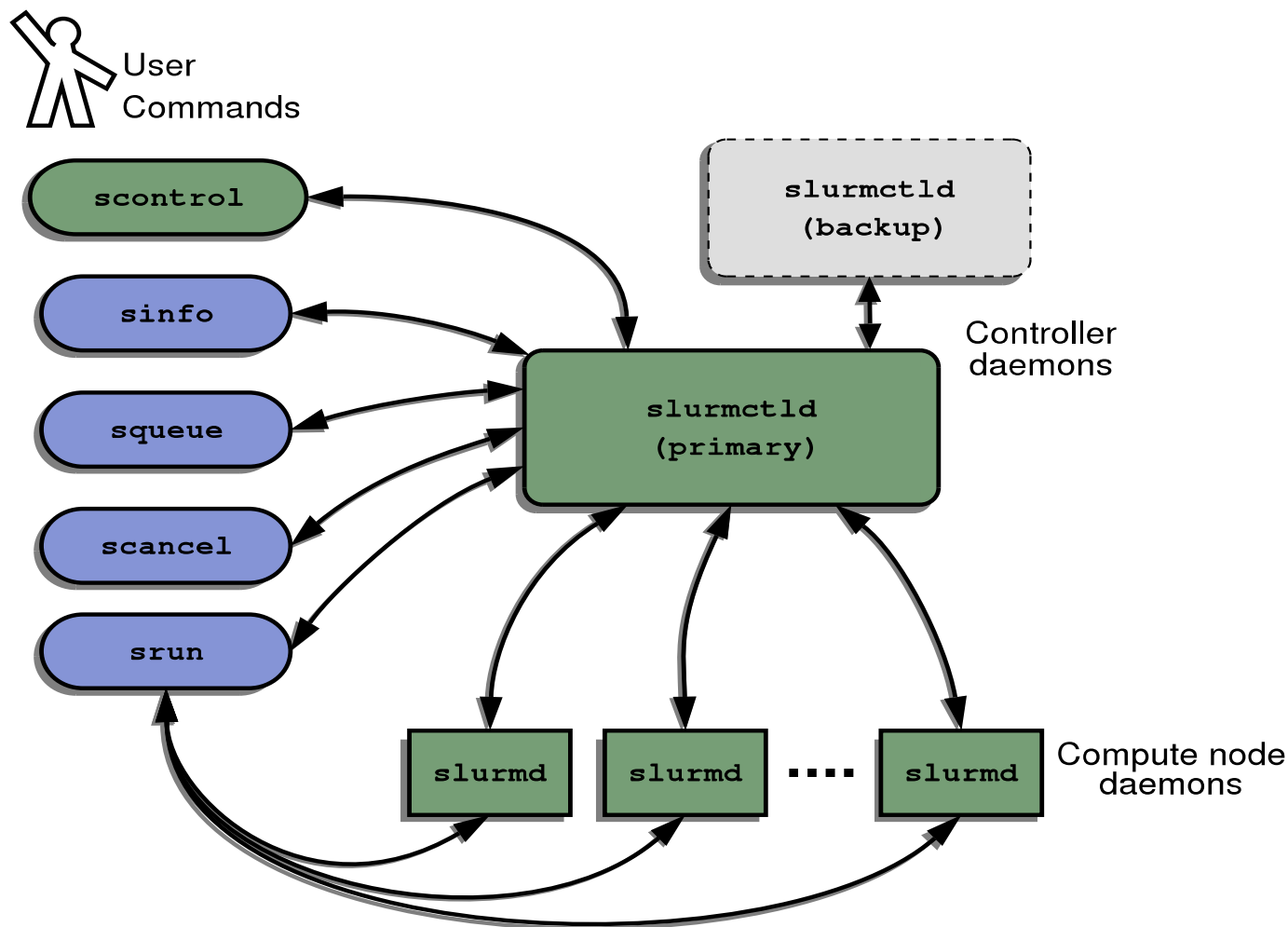
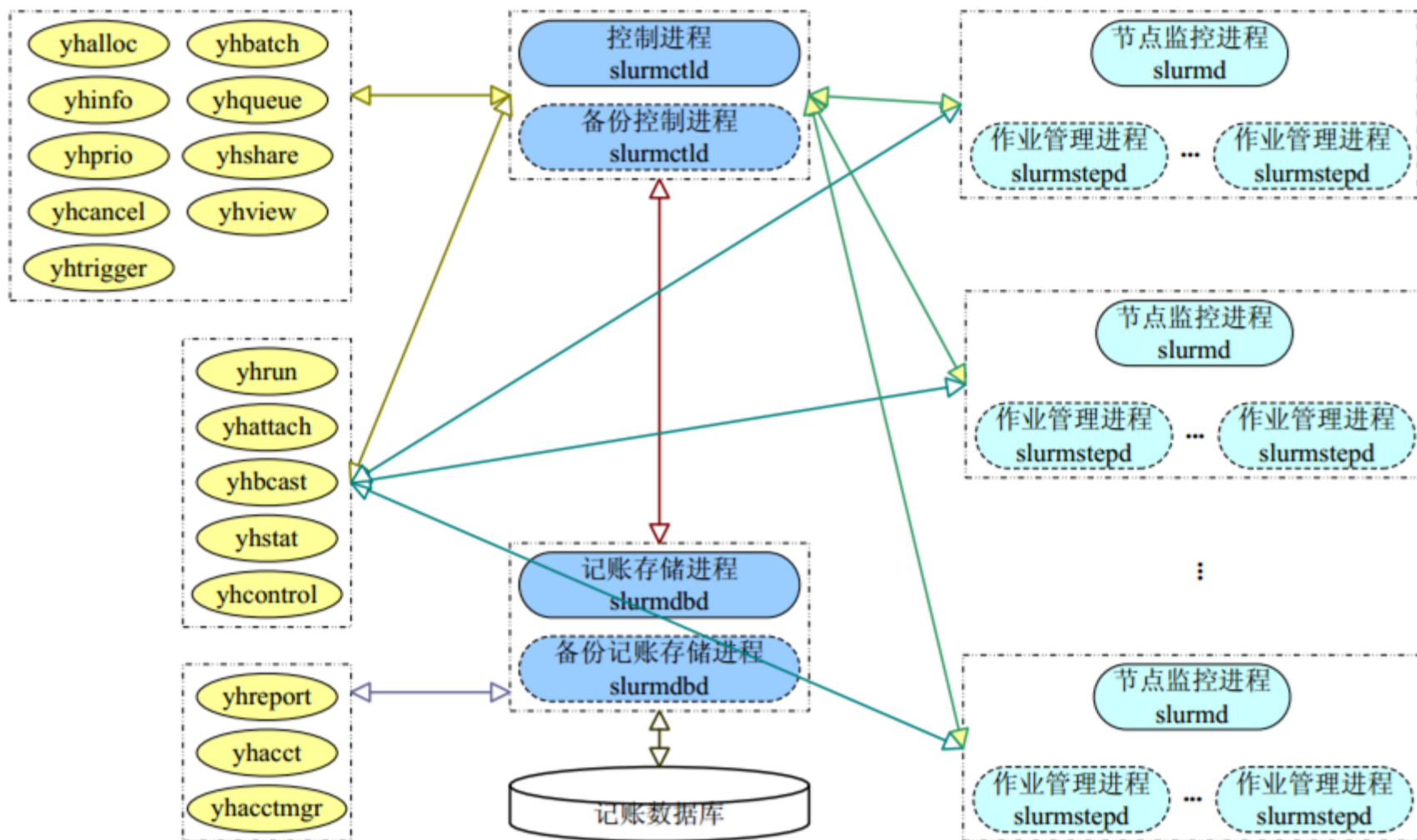


Fig. 1. SLURM architecture

# 天河二号slurm资源管理系统架构





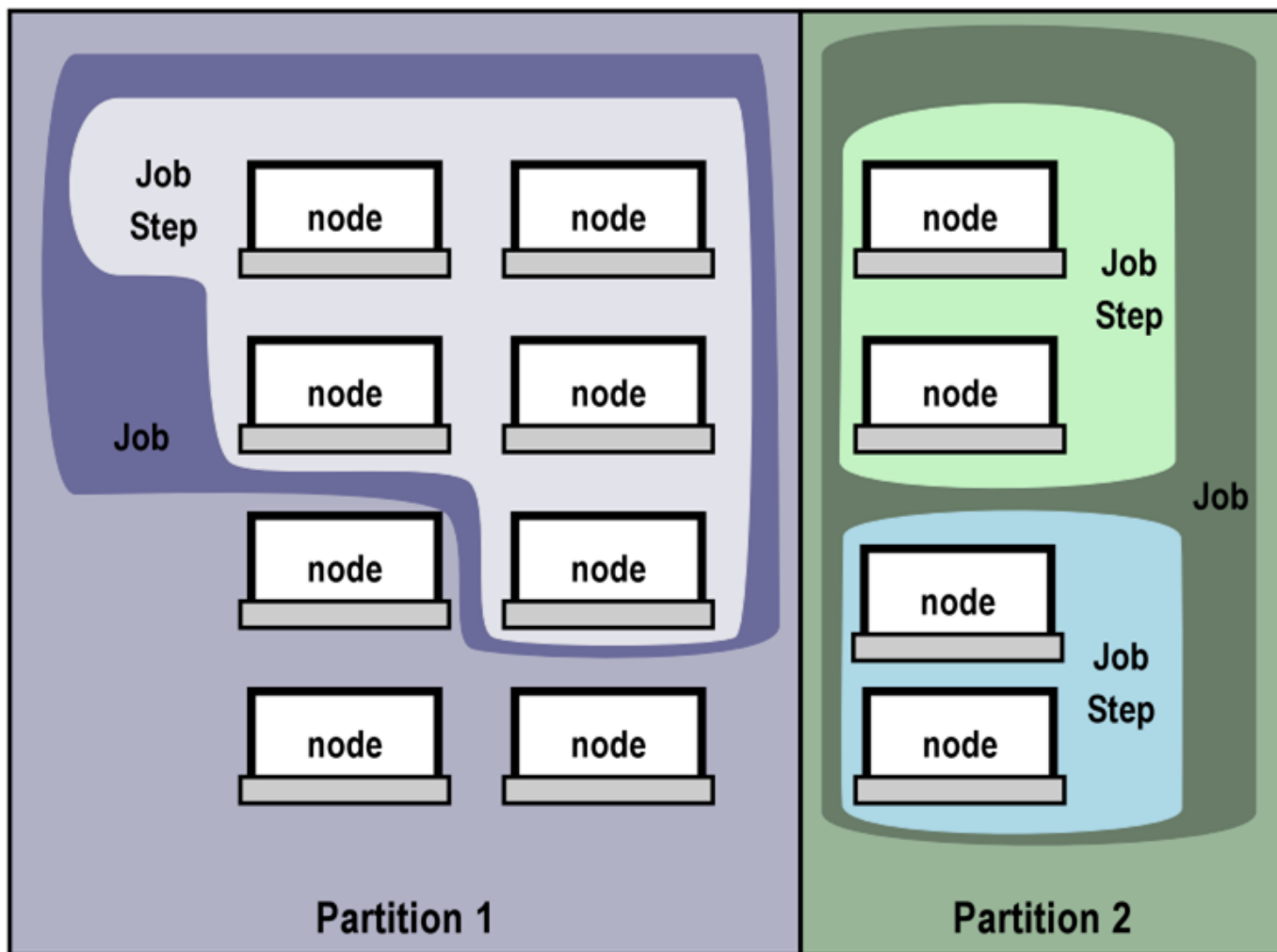
# Slurm管理进程介绍

- 控制进程：Slurmctld
  - 运行在管理节点
  - 是资源管理系统的控制中枢
  - 记录节点状态
  - 进行分区管理
  - 进行作业管理、作业调度、资源分配
- 记账存储进程：Slurmdbd
  - 运行在管理节点
  - 将作业信息保存到数据库
  - 记录用户、帐号、资源限制、 QOS 等信息
  - 用户认证和安全隔离

# Slurm管理进程 (2)

- 节点监控进程: Slurmd
  - 运行在每个计算节点
  - 监控节点状态, 并向控制进程注册
  - 接收来自控制进程与用户的请求并进行处理
- 作业管理进程: Slurmstepd
  - 加载计算任务时由节点监控进程启动
  - 管理一个作业步的所有任务
    - 启动计算任务进程
    - 标准 I/O 转发
    - 信号传递
    - 任务控制
    - 资源使用信息收集

# 资源管理系统实体概念



# 资源管理系统实体

- **节点 (Node) : 即指计算节点**
  - 包含处理器、内存、磁盘空间等资源
  - 具有空闲、分配、故障等状态
  - 使用节点名字标识
- **分区 (Partition) : 节点的逻辑分组**
  - 提供一种管理机制, 可设置资源限制、访问权限、优先级等
  - 分区可重叠, 提供类似于队列的功能
  - 系统有一个默认分区
  - 使用分区名字标识

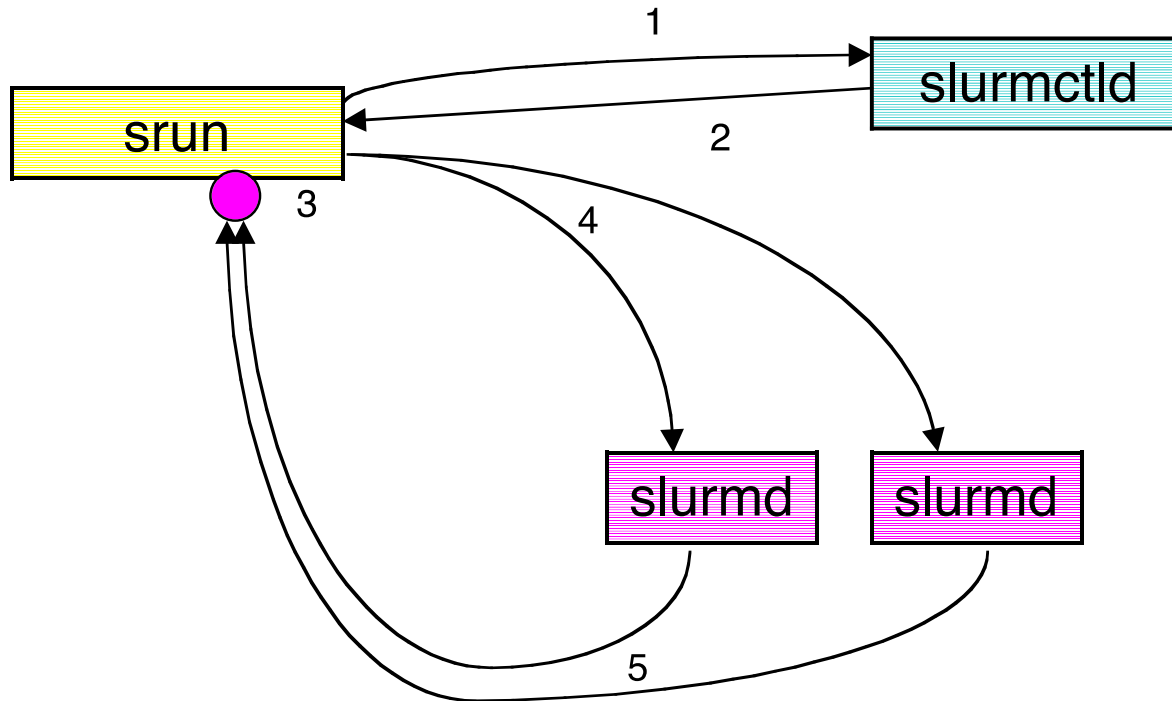
# 资源管理系统实体

- **作业 (Job) : 一次资源分配**
  - 位于一个分区中, 作业不能跨分区
  - 排队调度后分配资源运行
  - 通过作业 ID 标识, 如 123
- **作业步 (Jobstep) : 一个作业可包含多个作业步**
  - 通过 yhrun 进行的任务加载
  - 作业步可只使用作业中的部分节点
  - 可并发运行
  - 在作业内通过作业步 ID 标识, 如 123.0

# 资源管理系统关联

- 关联 (Association) : 系统实施资源限制的一个基础概念
  - 由<cluster,account,user,partition>构成的四元组
  - 每个作业都有对应的关联, 因为作业都是由用户使用某计费帐号提交到系统的一个分区中
- 帐号、用户的资源限制, 在实现上最终以关联进行记录
  - 节点数量
  - 作业数量
  - 时间限制

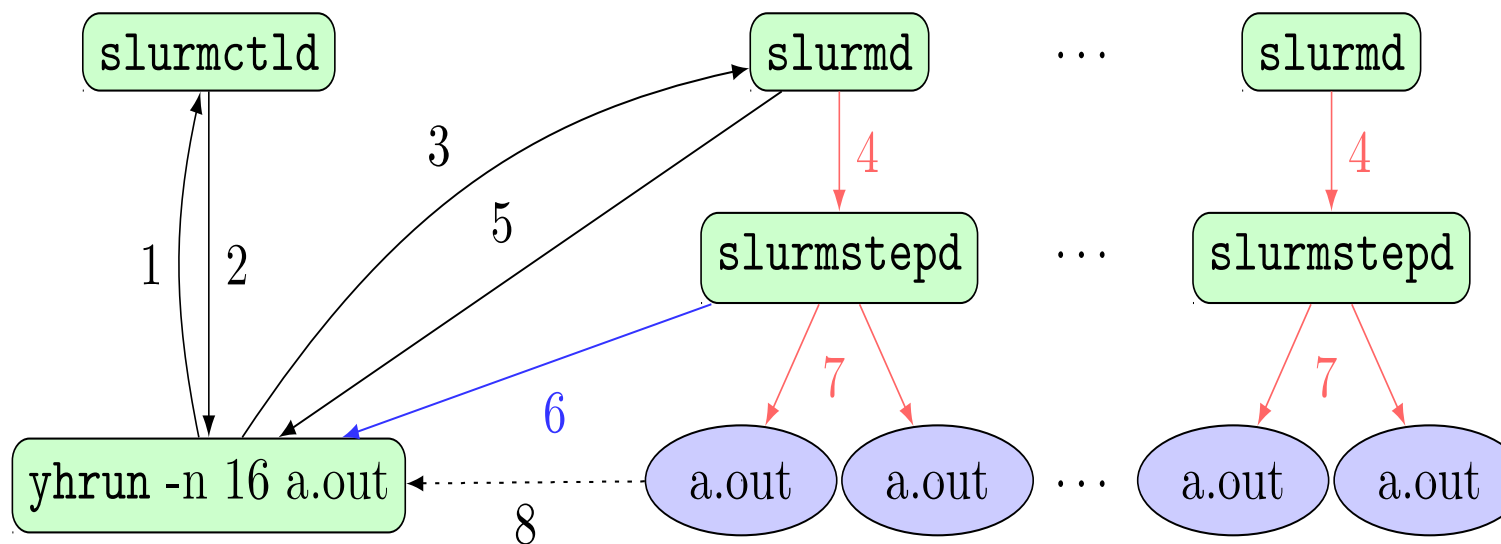
# Slurm工作流程



- 1: srun connects to slurmctld requesting resource allocation and step creation
- 2: slurmctld responds with node list and job step credential
- 3: srun opens I/O connection (ephemeral port)
- 4: srun sends job step requests to slurmd daemons
- 5: slurmd initiates job step and makes I/O connections to srun

# 天河二号作业步任务加载流程

● `yhrun [options] program [args]`



1. 创建作业步
2. 响应：任务布局/证书
3. 加载任务
4. 派生作业管理进程
5. 加载响应
6. 建立 I/O 连接
7. 派生任务
8. PMI 连接



# Slurm资源管理系统使用

# 天河二号命令工具

- 命令工具

- yhacct: 查看历史作业信息
- yhacctmgr: 记账管理
- yhalloc: 资源分配
- yhbatch: 提交批处理作业
- yhcancel: 取消作业
- yhcontrol: 系统控制
- yhinfo: 节点与分区状态查看
- yhqueue: 队列状态查看
- yhrun: 任务加载

# 系统状态查看

节点状态监控机制	命令工具
节点状态	yhinfo
分区状态	yhqueue
作业状态	yhcontrol
作业步状态	yhacct

# 节点状态监控机制

- 节点状态由控制进程维护
- 控制进程使用三种机制检查节点状态
  - ping: 仅检查通信状态
  - register: 报告资源状态
    - slurmd 启动时主动进行
    - 节点多时, 周期较长
  - health check: 管理员定制脚本
- 命令工具从控制进程获取节点状态

# 节点状态查看

- 使用yinfo命令查看节点状态
  - 别名: yhi

```
$ yinfo
PARTITION  AVAIL  TIMELIMIT  NODES  STATE  NODELIST
work*      up      infinite   1110  down*  cn[0-451,494-1151]
work*      up      infinite    42   idle  cn[452-493]
test       up      infinite    7   down*  cn[1178-1179,1224-1225,1244-1245,1259]
test       up      infinite   121   idle  cn[1152-1177,1180-1223,1226-1243,1246]
```

- 使用yhcontrol命令查看节点详细信息

```
$ yhcontrol show node cn0
NodeName=cn0 Arch=x86_64 CoresPerSocket=12
CPUAlloc=0 CPUErr=0 CPUTot=24 Features=(null)
OS=Linux RealMemory=64000 Sockets=2
State=DOWN* ThreadsPerCore=1 TmpDisk=0 Weight=1
Reason=Not responding [slurm@2015-03-15T15:17:11]
```

# 节点基本状态

- 节点基本状态值

- UNKNOWN: 未知, **unk**
- IDLE: 空闲, **idle**
- ALLOCATED: 已分配, **alloc**
- DOWN: 故障, **down**

- 状态标识

- DRAIN: 不再分配, **drng/drain**
- COMPLETING: 有作业正在退出, **comp**
- NO\_RESPOND: 无响应, **\***

- 指定查看特定状态的节点:

- `yhi -t 状态值`

# 分区状态查看

- 使用yhinfor命令查看分区状态
  - 与查看节点状态一致

```
$ yhinfor
```

PARTITION	AVAIL	TIMELIMIT	NODES	STATE	NODELIST
work*	up	infinite	1110	down*	cn[0-451,494-1151]
work*	up	infinite	42	idle	cn[452-493]
test	up	infinite	7	down*	cn[1178-1179,1224-1225,1244-1245,1259]
test	up	infinite	121	idle	cn[1152-1177,1180-1223,1226-1243,1246]

- 分区名
- 分区状态
  - UP、DOWN、DRAIN、INACTIVE
  - DEFAULT/\*
- 运行时间限制
- 查看指定分区
  - yhi -p 分区名
- 节点数量
- 节点状态
- 节点列表

# 分区属性查看

- 使用yhcontrol命令查看分区属性

```
$ yhcontrol show partition work
```

```
PartitionName=work
```

```
AllocNodes=ALL AllowGroups=ALL Default=NO
```

```
DefaultTime=NONE DisableRootJobs=NO Hidden=NO
```

```
MaxNodes=UNLIMITED MaxTime=UNLIMITED MinNodes=1
```

```
Nodes=cn[0-1151]
```

```
Priority=1 RootOnly=NO Shared=NO
```

```
State=UP TotalCPUs=27648 TotalNodes=1152
```

- |               |        |
|---------------|--------|
| – 节点列表        | – 资源限制 |
| – 状态： UP/DOWN | – 节点范围 |
| – 隐藏分区        | – 运行时间 |
| – 访问权限        | – 优先级  |
| – RootOnly    | – 共享节点 |
| – AllowGroups | – 默认分区 |



# 作业状态查看

- 使用`yhqueue`命令查看作业队列状态
  - 别名: `yhq`

```
$ yhqueue
JOBID PARTITION  NAME USER ST TIME  NODES NODELIST(REASON)
1463      test   sbatch root  R 1:06     12 cn[1246-1257]
1465      work    tjob  test PD 0:00     66 (PartitionNodeLimit)
1464      work    myjob root  R 0:32     23 cn[452-474]
```

- 作业结束一段时间后，信息将从`slurmctld`中清除
- 常用查询选项
  - `-p`: 指定分区
  - `-u`: 指定用户
  - `-t`: 指定状态
  - `-w`: 指定包含的节点
  - `-j`: 指定作业id号
  - `-J`: 指定作业名

# 作业详细信息查看

- 使用yhcontrol命令查看作业详细信息

```
$ yhcontrol show job 1464
JobId=1464 Name=myjog
UserId=root(0) GroupId=root(0)
Priority=2 Account=(null) QOS=normal
JobState=RUNNING Reason=None Dependency=(null)
TimeLimit=UNLIMITED Requeue=1 Restarts=0 BatchFlag=1 ExitCode=0:0
SubmitTime=2015-03-16T08:24:34 EligibleTime=2015-03-16T08:24:34
StartTime=2015-03-16T08:24:34 EndTime=NONE
SuspendTime=None SecsPreSuspend=0
Partition=work AllocNode:Sid=ln0:8116
ReqNodeList=(null) ExcNodeList=(null)
NodeList=cn[452-474]
NumNodes=23 NumCPUs=23 CPUs/Task=1 ReqS:C:T=1:1:1
MinCPUsNode=1 MinMemoryNode=0 MinTmpDiskNode=0
Features=(null) Reservation=(null)
Shared=OK Contiguous=0 Licenses=(null) Network=(null)
Command=(null)
WorkDir=/WORK
```

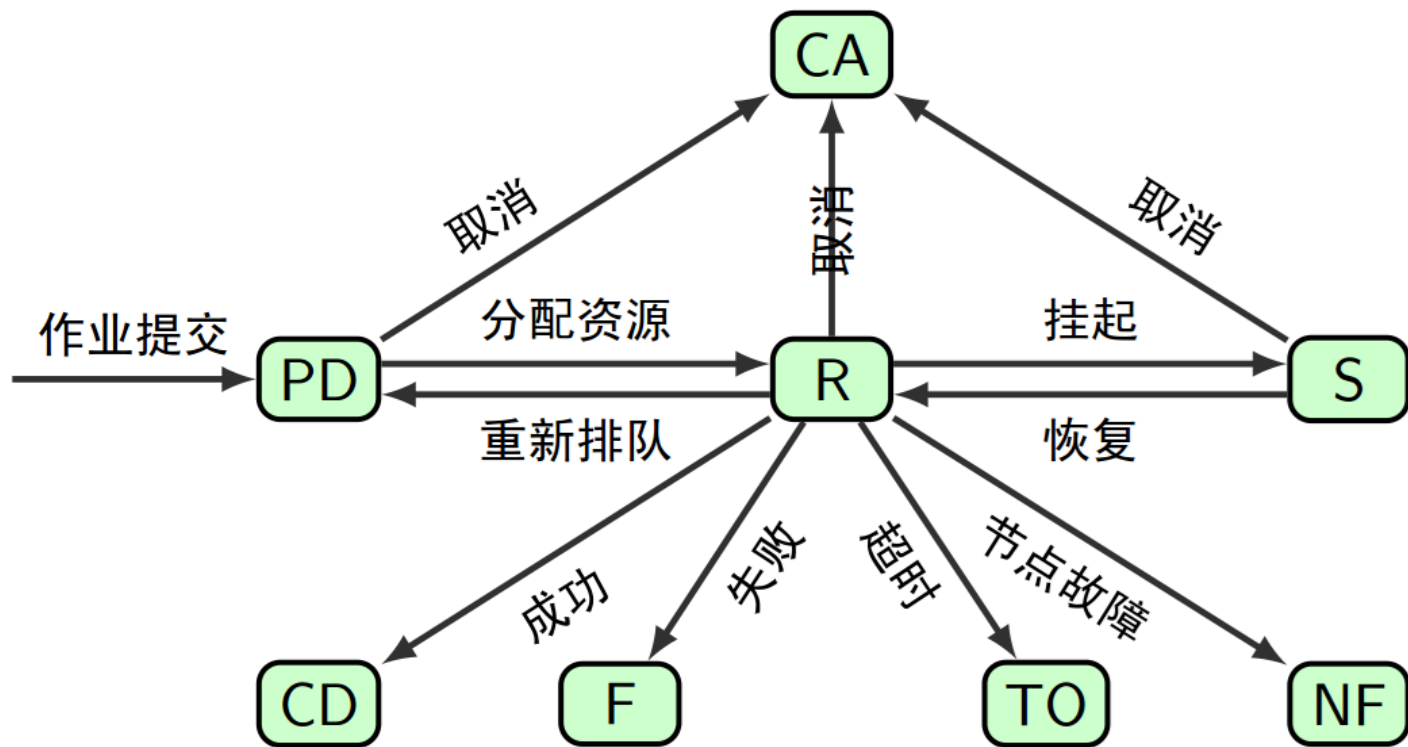
# 历史作业信息查看

- 使用yhacct命令查看历史作业信息

```
$ yhacct
```

JobID	JobName	Partition	Account	AllocCPUS	State	ExitCode
-----	-----	-----	-----	-----	-----	-----
18	xhpl	work	test	0	CANCELLED+	0:0
30	xhpl	work	test	0	CANCELLED+	0:0
32	xhpl	work	test	0	CANCELLED+	0:0
34	test.sh	work	test	256	COMPLETED	0:0
37	test.sh	work	test	256	FAILED	0:0
40	test.sh	work	test	2048	COMPLETED	0:0
40.0	xhpl		test	2048	COMPLETED	0:0
41	test.sh	work	test	2048	CANCELLED+	0:0
41.0	xhpl		test	2048	CANCELLED+	0:0
42	test.sh	work	test	2048	CANCELLED+	0:0
45	test.sh	work	test	2048	CANCELLED+	0:0
46	test.sh	work	test	2048	RUNNING	0:0
46.0	xhpl		test	2048	RUNNING	0:0
48	test.sh	work	test	0	PENDING	0:0

# 作业状态转换



## 状态标识

- **COMPLETING**: 正在退出, CG
- **CONFIGURING**: 分配给作业的节点正在启动, CF
- **PENDING**: 排队, PD
- **RUNNING**: 运行, R
- **SUSPENDED**: 挂起, S

- **COMPLETED**: 成功结束, CD
- **FAILED**: 失败结束, F
- **CANCELLED**: 被取消, CA
- **TIMEOUT**: 超时, TO
- **NODE\_FAIL**: 因节点故障而运行失败, NF

# 排队状态原因

- **Priority**: 优先级不够高
- **Dependency**: 作业的依赖关系未满足
- **Resources**: 当前可用资源不能满足作业需求
- **PartitionNodeLimit**: 作业请求的节点数超过分区的作业节点数限制
- **PartitionTimeLimit**: 作业请求的运行时间超过分区作业运行时间限制
- **PartitionDown**: 作业所在的分区处于 DOWN 状态
- **JobHeld**: 作业被阻止调度
- **BeginTime**: 作业请求的启动时间还未到达
- **AssociationJobLimit**: 关联的作业限制已满
- **AssociationResourceLimit**: 关联的资源限制已满
- **AssociationTimeLimit**: 关联的运行时间限制已满
- **Reservation**: 作业请求的预约时间未到
- **ReqNodeNotAvail**: 作业请求的节点不可用

# 结束状态原因

- **PartitionDown**: 作业所在的分区被删除
- **NodeDown**: 分配给作业节点进入 DOWN 状态
- **BadConstraints**: 作业的资源约束无效
- **SystemFailure**: 系统故障
- **JobLaunchFailure**: 作业加载故障
- **NonZeroExitCode**: 作业的退出代码非 0
- **TimeLimit**: 作业超出运行时间限制
- **InactiveLimit**: 作业超出不活跃时间限制
- **InvalidBankAccount**: 作业的计费帐号无效

# 作业步状态查看

- 使用yhqueue与yhcontrol命令查看作业步
  - 类似于查看作业队列

```
$ yhqueue -s
```

STEPID	NAME	PARTITION	USER	TIME	NODELIST
45.0	xhpl	work	test605	2:00:54	cn[0-255,512-1023]
46.0	xhpl	work	test605	1:57:49	cn[256-511]

```
$ yhcontrol show steps 46.0
```

StepId=46.0 UserId=1000 StartTime=2014-05-20T08:19:27 TimeLimit=6-22:40:00  
Partition=work Nodes=cn[256-511] Tasks=256 Name=xhpl Network=(null)  
ResvPorts=(null) Checkpoint=0 CheckpointDir=/WORK/home/test605/hpl/goto/256-2

- 作业步仅在运行时存在，运行结束后从系统中删除，并记录到记账数据库中
- 使用yhacct 命令可查看历史作业步信息

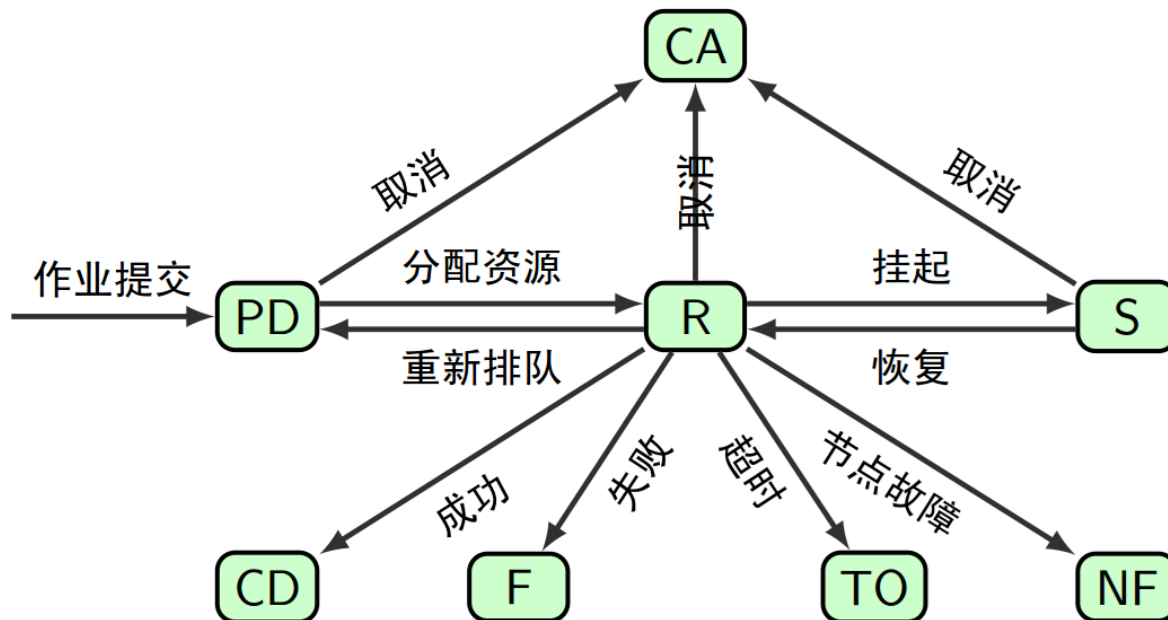
# 作业与资源分配

- 作业运行模式
- 作业的资源需求
- 作业的运行参数
- 作业的环境变量





# 作业 = 资源分配请求



- **提交：** 申请资源
- **排队：** 等待资源
- **运行：** 分配资源（无论是否执行程序）
- **挂起：** 暂时释放资源
- **结束：** 释放资源

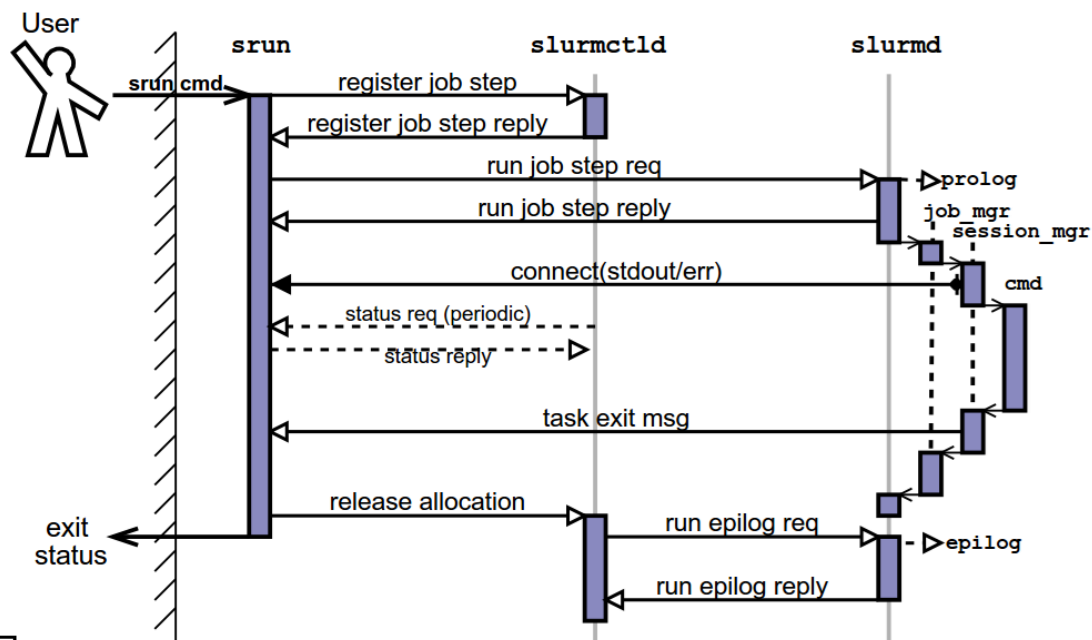
# 作业运行模式

- 交互模式 `yhrun`
- 批处理模式 `yhbatch`
- 分配模式 `yhalloc`
- 只是用户使用方式区别
- 管理、调度、记账时同等对待

# 交互模式

## 交互模式作业

- 1.在终端提交资源分配请求
- 2.等待资源分配
- 3.加载计算任务
- 4.运行中，任务I/O传递到终端
- 5.可与任务进行交互：I/O，信号
- 6.任务执行结束后，资源被释放



- 一个作业（一次资源分配）包含一个作业步（一次任务加载）

# 交互模式

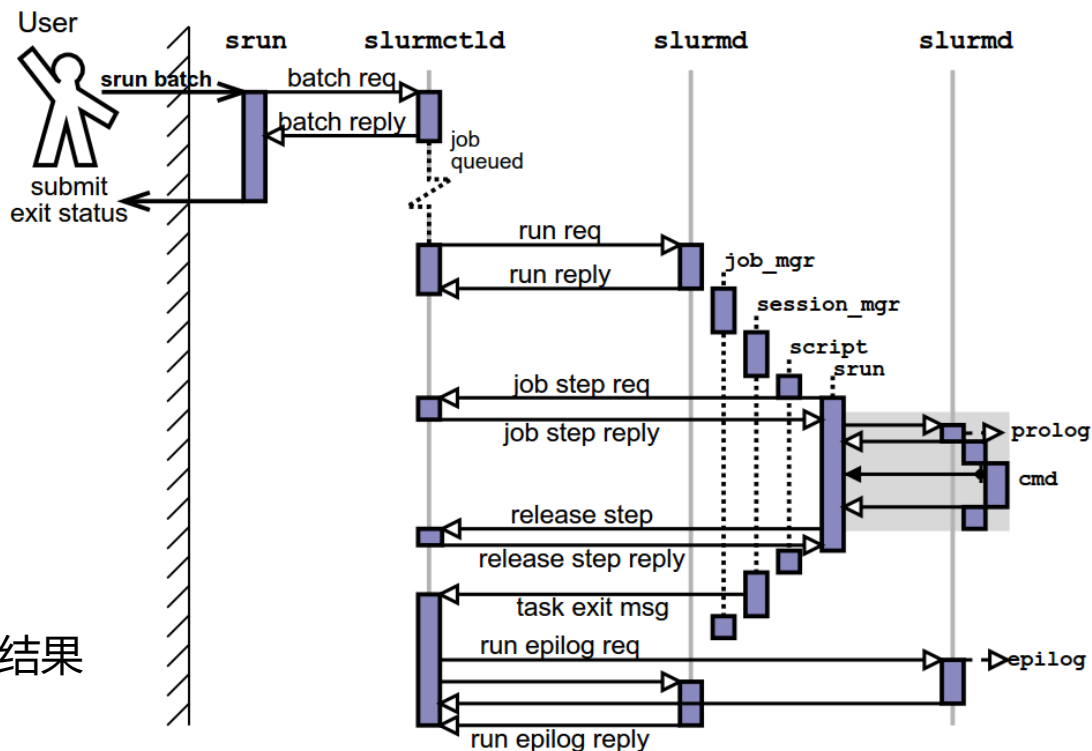
- 通过yhrun命令运行

```
$ yhrun -n 4 cpi
yhrun: job 52 queued and waiting for resources
yhrun: job 52 has been allocated resources
Enter the number of intervals: (0 quits) 1
pi is approximately 3.2000000000000002, Error is 0.0584073464102071
wall clock time = 0.000047
Enter the number of intervals: (0 quits) 2
pi is approximately 3.1623529411764704, Error is 0.0207602875866773
wall clock time = 0.000022
Enter the number of intervals: (0 quits) 3
pi is approximately 3.1508492098656031, Error is 0.0092565562758100
wall clock time = 0.000014
Enter the number of intervals: (0 quits) 0
```

# 批处理模式

## 批处理模式

- 1.用户编写作业脚本
- 2.提交作业
- 3.作业排队等待资源分配
- 4.在首节点加载执行作业脚本
- 5.脚本执行结束，释放资源
- 6.用户在输出文件中查看运行结果



- 脚本中可通过yhrun加载计算任务
  - 一个作业可包含多个作业步

# 批处理模式

- **脚本文件**：第一行应以“**#!/**”开头，指定脚本文件的解释程序；在脚本中，如果一行以“**#SBATCH**”开头，则该行中的其余部分被当作命令行选项，被**yhbatch**处理

```
$ cat job.sh
#!/bin/sh
#SBATCH -N 16 -t 100 -n 16 -c 4
yhrun -n 16 hostname
```

- **通过yhbatch命令运行**

```
$ yhbatch job.sh
Submitted batch job 53
$ yhqueue
```

JOBID	PARTITION	NAME	USER	ST	TIME	NODES	NODELIST(REASON)
53	work	job.sh	test605	PD	0:00	16	(Priority)
52	work	test.sh	test605	R	7:04:01	256	cn[256-511]

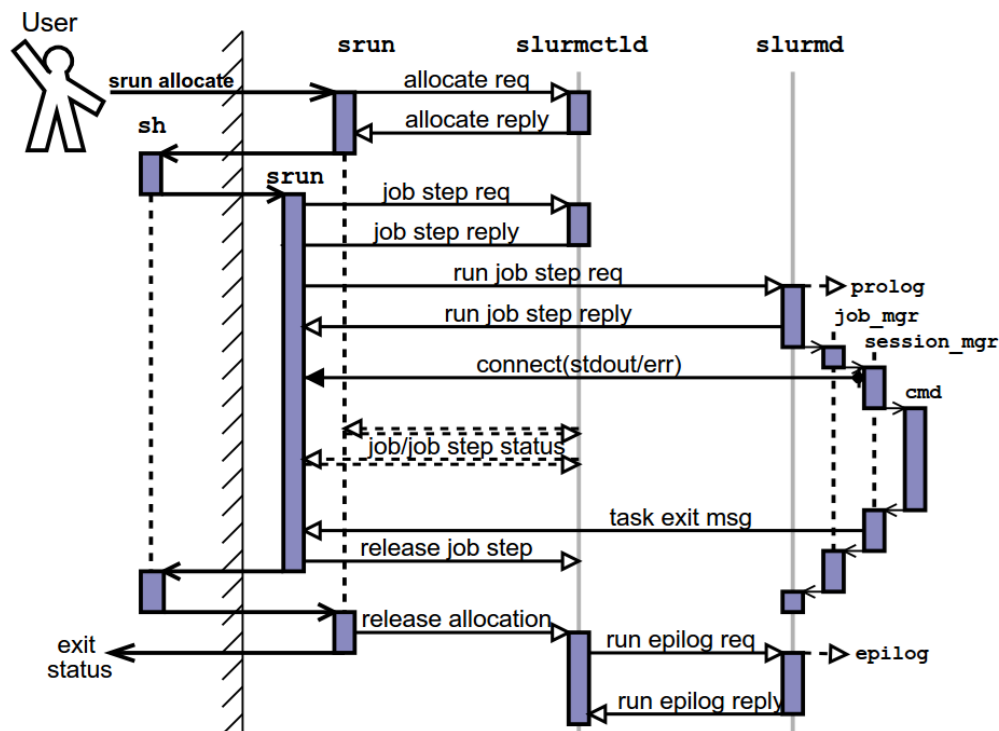
- **运行后，生成输出文件**

```
$ ls
hpl-2.0  iotest  job.sh  NPB3.3-MPI  slurm-53.out  test.sh
```

# 分配模式

## 分配模式

- 1.提交资源分配请求
- 2.作业排队等待资源分配
- 3.执行用户指定的命令
- 4.命令执行结束，释放资源



- 交互模式作业与批处理模式作业的结合
  - 一个作业可包含多个作业步
  - 可通过yhrun加载计算任务
  - 可与任务进行交互
  - 命令在用户提交作业的节点上执行

# 分配模式

- 通过yhallocc命令运行

```
$ yhallocc -N 2 -n 4 -c 2 -t 100 /bin/sh
yhallocc: Granted job allocation 56
sh-4.1$ yhrun -n 4 hostname
cn1
cn1
cn0
cn0
sh-4.1$ ssh cn0 ls
hpl-2.0  iotest  job.sh  NPB3.3-MPI  slurm-53.out  test.sh
sh-4.1$ yhrun -n 2 date
Fri Mar 20 15:46:05 CST 2015
Fri Mar 20 15:46:05 CST 2015
sh-4.1$ exit
exit
yhallocc: Relinquishing job allocation 56
$
```



# 作业的资源需求

- **节点数量:** `-N,--nodes min[-max]`
  - 如未指定, 则根据其他需求, 分配足够的节点
- **处理器数量:** 由几个参数决定
  - 作业要加载的任务数 `-n,--ntasks`, 默认每个节点一个
  - 每个任务需要的处理器数 `-c,--cpus-per-task`, 默认为 1
  - 系统将根据参数计算, 分配足够处理器数目的节点
- **节点与处理器数目约束**

```
$ yhrun -N 4 -n 8 -l hostname
0: cn1246
2: cn1247
4: cn1248
6: cn1249
1: cn1246
3: cn1247
5: cn1248
7: cn1249
```

# 作业的资源需求

- **运行时间：** `-t,--time`
  - 单位为分钟
  - 超出时间限制的作业将被终止
  - 应尽可能准确估计：调度时用此估计时间进行backfill判断与优先级设置

```
$ yhrun -N 4 -n 8 -t 100 a.out
```

- **分区：** `-p,--partition`
  - 从指定分区中分配节点
  - 使用指定分区的资源限制 / 访问权限进行检查
  - 作业必须位于一个分区中，不能跨分区
- **节点：**
  - `-w,--nodelist`：指定分配给作业的资源中**至少**要包含的节点
  - `-F,--nodefile`：指定分配给作业的资源中**至少**要包含的节点，但节点写在文件中，而不是在命令行直接给出（**此选项仅对yhalloc和yhbatch可用**）
  - `-x,--exclude`：指定分配给作业的资源中不要包含的节点
  - `--exclusive`：指定作业不能与其它作业共享节点（**默认作业不共享节点**）
  - `--contiguous`：表示作业需要被分配连续的节点

# 作业的运行参数

- 作业名字

- 默认：加载的程序 / 批处理脚本文件名 / 执行的命令
- -J,--job-name: 指定名字

```
$ yhbatch -N 4 job.sh
$ yhbatch -N 8 -J myjob job.sh
$ yhqueue
```

JOBID	PARTITION	NAME	USER	ST	TIME	NODES
NODELIST(REASON)						
1234	work	job.sh	test	R	00:05	4 cn[0-3]
1235	work	myjob	test	R	00:03	8 cn[12-19]

```
$ yhcanceled -n myjob
```

- 工作目录

- -D,--chdir: 指定任务/脚本/命令的工作目录
- 默认：yhrun/yhbatch/yhallocc 的工作目录

```
$ yhbatch -N 4 -D /WORK/test/devel/bin job.sh
```

# 作业的运行参数

- 启动时间

- --begin: 作业在指定时间之后才能运行

- 依赖关系

- -d,--dependency: 指定作业的依赖关系
  - after:jobid: 在指定作业开始之后
  - afterok:jobid: 在指定作业成功结束之后
  - afternotok:jobid: 在指定作业不成功结束之后
  - afterany:jobid: 在指定作业结束之后
- 满足依赖关系的作业才能运行
- 不可能满足依赖关系的作业将被取消

- 节点故障容忍

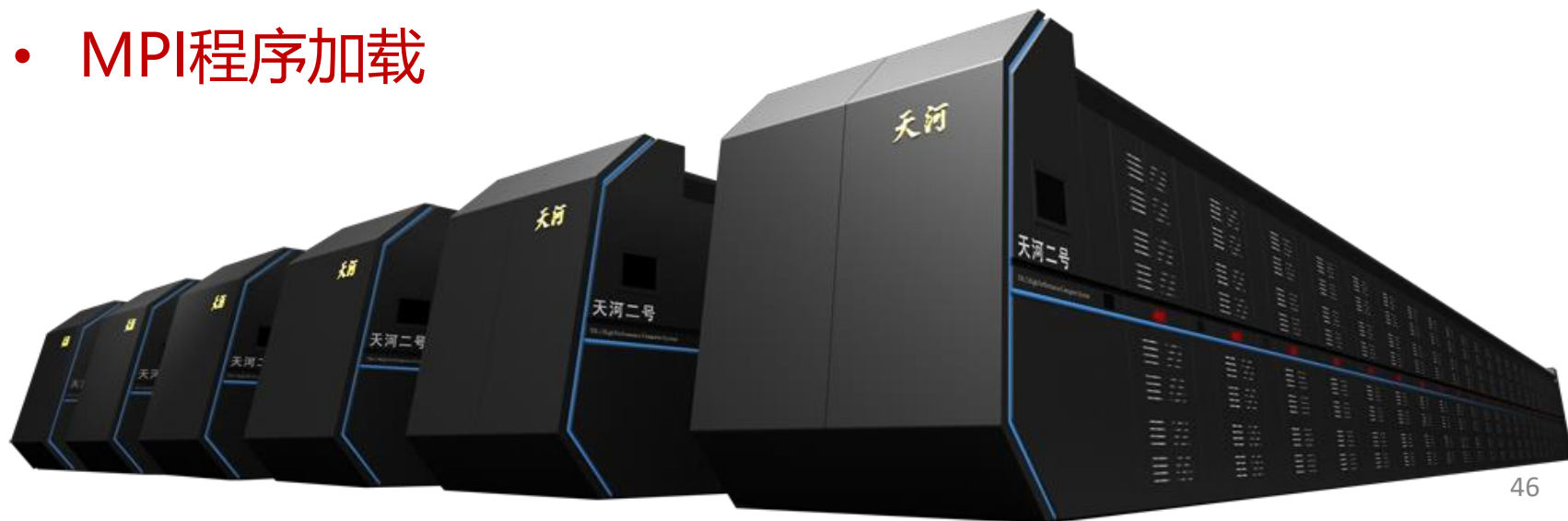
- 默认: 节点失效时将终止作业
  - 失效指节点变为 DOWN 状态
  - 主要针对 MPI 程序的执行, 及时释放资源
- -k,--no-kill: 容忍节点故障
  - 程序自身容错
  - 正在执行的作业步失败后, 继续运行后续作业步

# 作业的环境变量

- 系统在运行计算任务/作业脚本/命令时，会为其设置一些环境变量，以反映其资源分配情况
- 在批处理和分配模式作业中，可根据环境变量获得资源分配情况
  - SLURM\_NPROCS: 要加载的任务数
  - SLURM\_TASKS\_PER\_NODE: 每节点要加载的任务数
  - SLURM\_JOB\_ID: 作业的 JobID
  - SLURM\_SUBMIT\_DIR: 提交作业时的工作目录
  - SLURM\_JOB\_NODELIST: 作业分配的节点列表
  - SLURM\_JOB\_CPUS\_PER\_NODE: 每个节点上分配给作业的 CPU 数
  - SLURM\_JOB\_NUM\_NODES: 作业分配的节点数
  - HOSTNAME: 对于批处理作业，此变量被设置为批处理脚本所执行节点的节点名

# 作业步与任务加载

- 作业步任务布局
- 作业步任务I/O 传递
- 登录计算节点
- MPI程序加载



# 任务布局

- 任务布局指任务在所分配节点上的分布
  - 任务数分布
  - 任务号分布
- 支持多种布局方式: -m,--distribution
  - 循环布局
  - 块布局
  - 基于面的布局
  - 任意布局
- 可用于性能调优
  - 节点的负载
  - 任务间通信

# 循环布局

- -m cyclic
- 在分配的处理器数目范围内，尽可能将任务在节点间平均分配
- 示例：设作业步使用 4 个节点，cn[0,3] 四个处理器，cn[1-2] 两个处理器

cn0	cn1	cn2	cn3
0	1	2	3
4	5	6	7
8			9
10			11
12	13	14	15

先按可用处理器在节点间轮转分配任务

处理器全部分配，按节点轮转分配任务

...



# 块布局

- -m block
- 各节点任务数与循环布局相同，仅任务编号不同
  - 由横着排变成竖着排

16 个任务

cn0	cn1	cn2	cn3
0	5	8	11
1	6	9	12
2			13
3			14
4	7	10	15

10 个任务

cn0	cn1	cn2	cn3
0	3	5	7
1	4	6	8
2			9

# 基于面的布局

- -m plane=size
- 是一种块循环布局，块大小为 size
- 不考虑负载

示例：4 节点×4 处理器，加载 16 任务

循环布局

cn0	cn1	cn2	cn3
0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

块布局

cn0	cn1	cn2	cn3
0	4	8	12
1	5	9	13
2	6	10	14
3	7	11	15

面布局，块大小为 3

cn0	cn1	cn2	cn3
0	3	6	9
1	4	7	10
2	5	8	11
12	15		
13			
14			

# 任意布局

- **-m arbitrary**
- **-F, --nodefile** 指定节点列表文件
  - 每个节点一行
  - 行数与要加载的任务数相同

```
$ cat job.sh
#!/bin/sh
echo "This is process $SLURM_PROCID of $SLURM_NPROCS running on
`hostname`"
$ cat hostlist
cn2
cn1
cn0
$ yhrun -N 3 -n 3 -w ./hostlist -m arbitrary ./job.sh
This is process 0 of 3 running on cn2
This is process 1 of 3 running on cn1
This is process 2 of 3 running on cn0
```

# 查看任务布局

- 对于已经加载运行的作业步，可以通过 “`yhattach --layout`” 命令查看其任务布局

```
$ yhqueue -s
STEPID      NAME PARTITION  USER    TIME NODELIST
45453.0      xhpl      all    root     7:46  cn[0-63]
45454.0      xhpl      all    root     7:46  cn[64-127]
45455.0      xhpl      all    root     7:33  cn[128-142,144-149,151-193]
...
$ yhattach --layout 45453.0
Job step layout:
64 tasks, 64 nodes (cn[0-63])
Node 0 (cn0), 1 task(s): 0
Node 1 (cn1), 1 task(s): 1
Node 2 (cn2), 1 task(s): 2
...
Node 62 (cn62), 1 task(s): 62
Node 63 (cn63), 1 task(s): 63
```

# 多程序作业步

- 支持 MPMD 程序的运行，即不同任务号执行不同程序
  - --multi-prog 选项
  - yhrun 最后跟配置文件，而不是可执行程序命令
- 配置文件格式
  - 按行组织，每行分为若干空白分隔的字段
  - 第一字段：任务号部分
    - 逗号分隔的任务号列表
    - 可包含范围 min-max
    - "\*" 表示其余所有任务
  - 第二字段：要执行的程序
  - 其余字段：执行程序的参数

# 多程序作业步

- 配置文件格式

- 在程序和参数部分，支持变量替换
- %t: 任务号    %o: 在本行的偏移

```
$ cat mp.conf
0          a.out abc
1          b.out %t
2,7-9     c.out %o
*          d.out
$ yhrun -n 16 --multi-prog mp.conf
```

- 任务布局

```
0: a.out abc
1: b.out 1
2: c.out 0
3-6,10-15: d.out
7: c.out 1
8: c.out 2
9: c.out 3
```

# 作业步任务I/O 传递

- **I/O 传递**: 可通过 `-i,--input/-o,--output/-e,--error` 选项控制
  - `all`: 默认, 传递所有任务 I/O
  - `none`: 不传递
  - `taskid`: 仅传递指定任务的 I/O
  - `filename`: 将任务的 I/O 写入文件
  - 支持变量替换, 节点名 / 任务号 / 局部任务号等
- **将任务的标准输出 (和标准错误) 写入文件 result:**

```
$ yhrun -n 16 -o result a.out
```

- **将每个任务的标准输出 (和标准错误) 写入不同文件中:**

```
$ yhrun -n 16 -o result-%t a.out
```

- **任务从文件 input 读取标准输入:**

```
$ yhrun -n 16 -i input a.out
```

# 登录计算节点

- 用户分配资源后，可使用 SSH 登录所分配的节点
- 用户只能登录自己的已经有作业分配的节点
- 可用于查看节点上进程状态、运行程序等
- 用户自行运行的程序不会由资源管理系统自动进行CPU绑定
- 当用户不再有作业分配计算节点后，其在相应节点上的所有进程将被终止



# MPI 程序加载

- MPICH2 及派生 MPI

- 资源管理系统中集成了PMI实现，对使用PMI接口进行进程管理的MPI实现，可利用yhrun直接加载，包括 MPICH2、MVAPICH2、YH-MPI 等
- 编译配置MPI时需指定 --with-pmi=slurm

- Open MPI

- Open MPI支持使用yhrun加载任务，编译配置时需指定 --with-slurm
- 加载的是 orted，而不是直接加载任务，产生作业步，但名字为 orted

- 通用加载方法

- 利用资源管理系统分配资源
  - yhallocc
  - yhbatch
- 使用 MPI 自带的 mpiexec 加载计算任务
  - 进行修改或包装
  - 从环境变量中获取分配的节点、要加载的任务等

# 作业控制

- 取消作业
- 挂起与恢复
- 修改作业

# 取消作业

- **yhcancel 命令取消作业 / 作业步**
  - 排队作业：标记为 CANCELLED 状态
  - 运行 / 挂起作业：终止所有作业步；标记为 CANCELLED 状态；回收资源
  - 使用yhcancel之后，系统将定期重复发送 SIGKILL 到作业步任务，直到其退出
  - 显示为 CG 状态的作业已经结束，不用再取消

```
$ yhcancel 123 456
$ yhcancel 789.1
# yhcancel -u test
# yhcancel -p debug -t pd
```

# 作业挂起与恢复

- **yhcontrol 命令挂起 / 恢复作业 / 作业步**
  - 挂起：暂时释放处理器资源
    - 向任务发送 SIGTSTP 和 SIGSTOP 信号
    - 节点被释放，可以分配给其它作业
  - 恢复：节点再次分配给作业；向任务发送 SIGCONT 信号

```
$ yhqueue
JOBID PARTITION      NAME      USER  ST      TIME  NODES NODELIST(REASON)
  123      work      job.sh    root   R       0:02      2  cn[0-1]
$ yhcontrol suspend 123
$ yhqueue
JOBID PARTITION      NAME      USER  ST      TIME  NODES NODELIST(REASON)
  123      work      job.sh    root   S       0:02      2  cn[0-1]
$ yhcontrol resume 123
$ yhqueue
JOBID PARTITION      NAME      USER  ST      TIME  NODES NODELIST(REASON)
  123      work      job.sh    root   R       0:05      2  cn[0-1]
```

# 修改作业

- 改变作业排队的分区与作业名字

```
$ yhqueue
JOBID PARTITION      NAME      USER  ST      TIME  NODES NODELIST(REASON)
  123      work      job.sh    root  PD      0:00      2 (Resource)
$ yhcontrol update jobid=123 partition=debug name=myjob
$ yhqueue
JOBID PARTITION      NAME      USER  ST      TIME  NODES NODELIST(REASON)
  123      debug      myjob     root  PD      0:00      2 (Resource)
```

- 改变已运行作业的运行时间限制

```
$ yhqueue --format '%i %P %l %t %.5M %N'
JOBID PARTITION TIMELIMIT ST  TIME  NODELIST
  123      work      60:00  R  52:12  cn[0-1]
...
# yhcontrol update jobid=123 timelimit=100
$ yhqueue --format '%i %P %l %t %.5M %N'
JOBID PARTITION TIMELIMIT ST  TIME  NODELIST
  123      work      01:40:00  R  52:25  cn[0-1]
```

# 小结

## 1.天河二号系统概述

## 2.资源管理系统概述

- 系统组成
- 系统实体

## 3.资源管理系统使用

- 状态查看
- 作业与资源分配
- 作业步与任务加载
- 作业控制

