IHEP School of Computing 2020

高能物理计算概述

高能所計算中心
IHEP Computing Center

程耀东

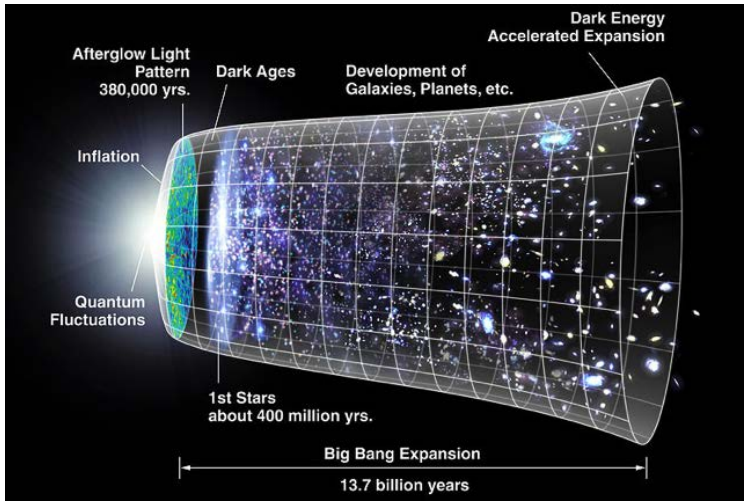
chyd@ihep.ac.cn

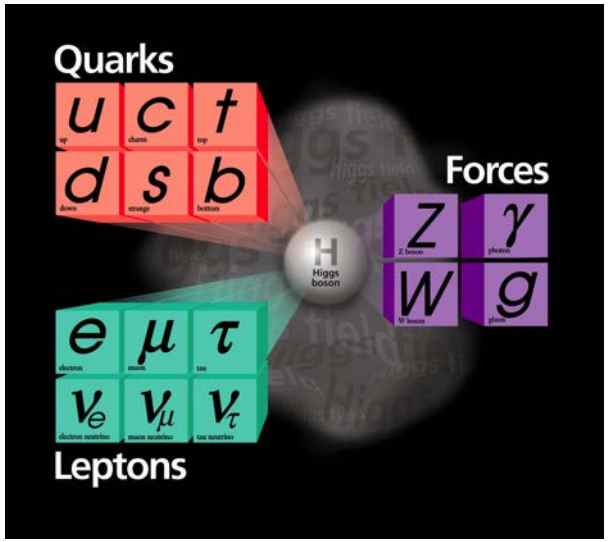
主要内容

- 高能物理数据处理的挑战
- 高能物理数据处理平台
- 分布式系统使用与优化
- 新技术展望

高能物理的目标

- 探索物质微观结构、宇宙起源等自然规律、新物理寻找
  - 为什么物质有质量？标准模型不能解释W,Z玻色子为什么有质量
  - 为什么宇宙中观测到的物质只有理论预言的4%?



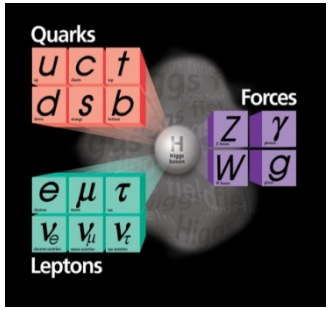


- 反物质在哪里？为什么自然界的正反物质是不对称的？
- 宇宙大爆炸刚发生时的物质形态是什么样的？

高能物理科学研究

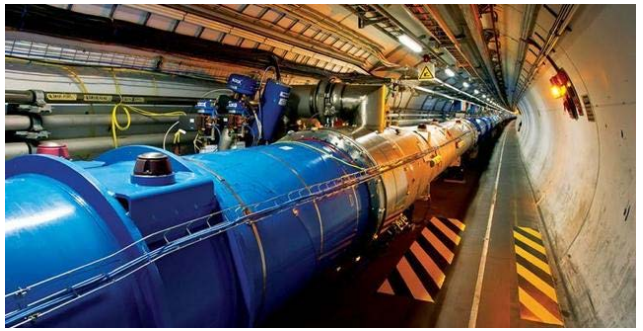
物质结构组成（理论）

- 夸克、轻子、玻色子
- 强力、弱力、电磁力、万有引力



粒子加速器（装置）

- 粒子物理研究的重要手段之一
- BEPCII，LHC，CEPC等等



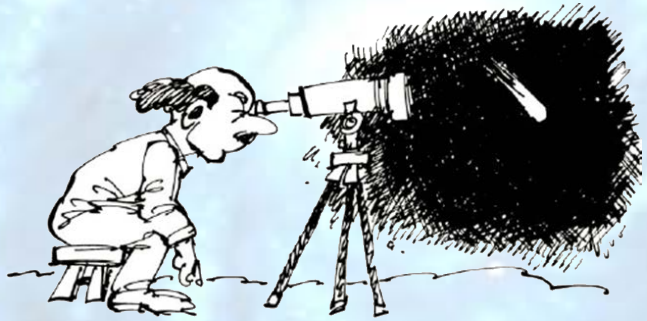
探测器 （实验）

- 探测各类粒子，用于科学研究
- BESIII, JUNO, LHAASO, ATLAS, CMS …

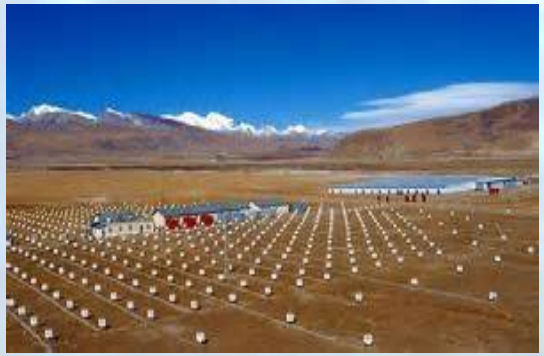
数据分析（科学发现）

- 暗物质/暗能量
- 宇宙起源、…

探索世界

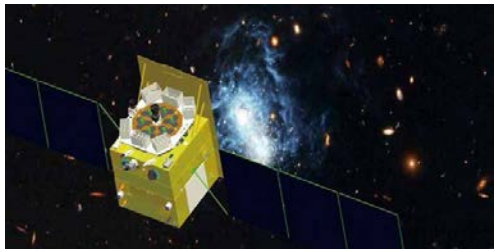


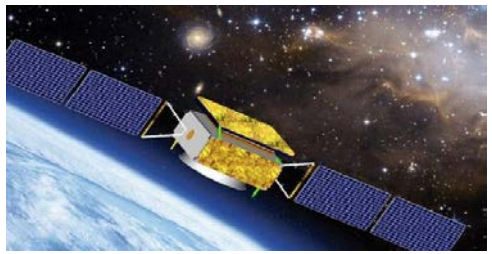
现代的大科学装置



中国的高能物理实验

太空





~550KM

空间天文卫星
(HXMT，DAMPE等)

高原

~4400M

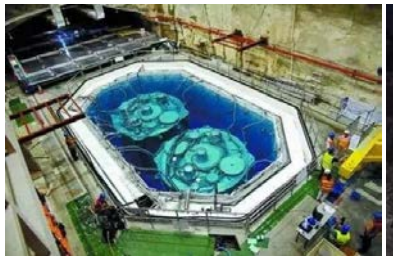
高海拔宇宙线观测站
(LHAASO，YBJ等)

地面

~-5M

粒子对撞机
(BEPCII，CSNS等)

地下

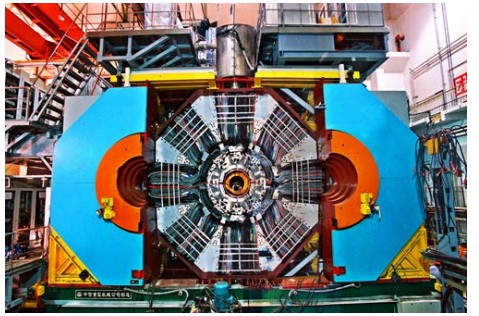


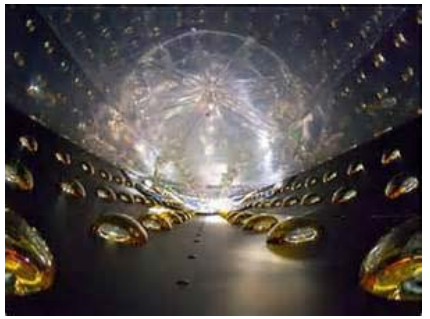
-2500M
~-300M

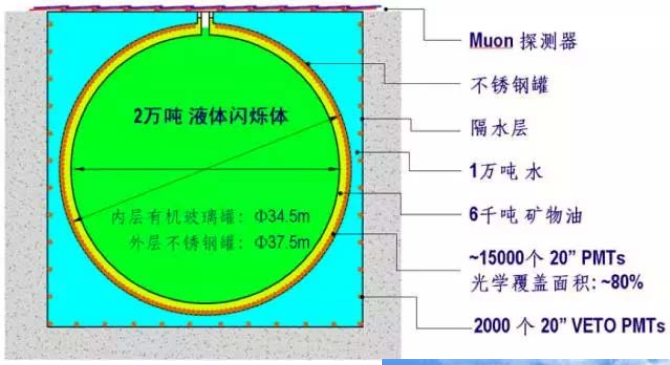
地下中微子实验
(JUNO，dayabay，JPL等)

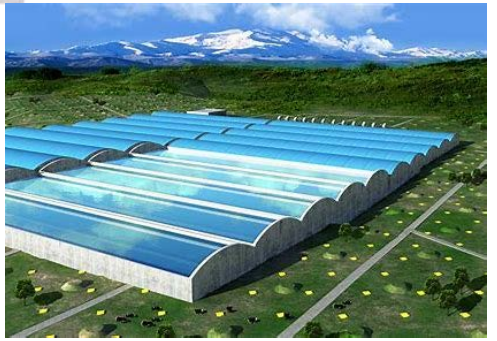
高能物理实验数据挑战

- 北京正负电子对撞机BECPII/BESIII
  - 每年~1PB raw data
  - 已经积累10PB+
- 大亚湾中微子实验
  - 每年200TB原始数据
  - 已经积累2PB以上原始数据
- 江门中微子实验
  - 地下500米实验大厅
  - 2022年运行，每年将产生2PB数据
- 高海拔宇宙线实验LHAASO
  - 位于四川稻城海子山，海拔4400米
  - 目前1/2规模运行，未来每年将产生6PB以上原始数据
- 其它：HXMT、 CSNS 、eXTP、HERD、HEPS、…

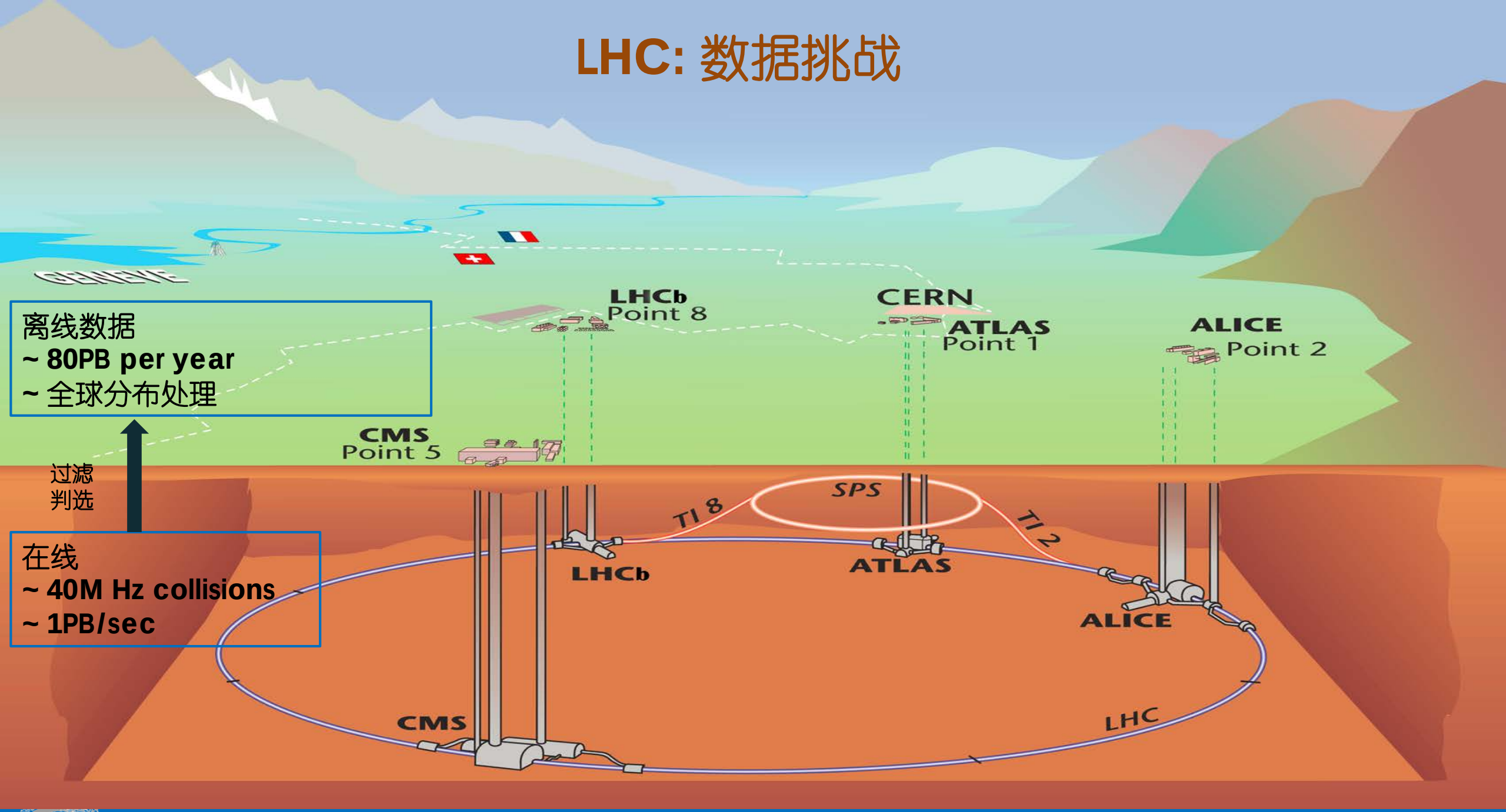








LHC: 数据挑战

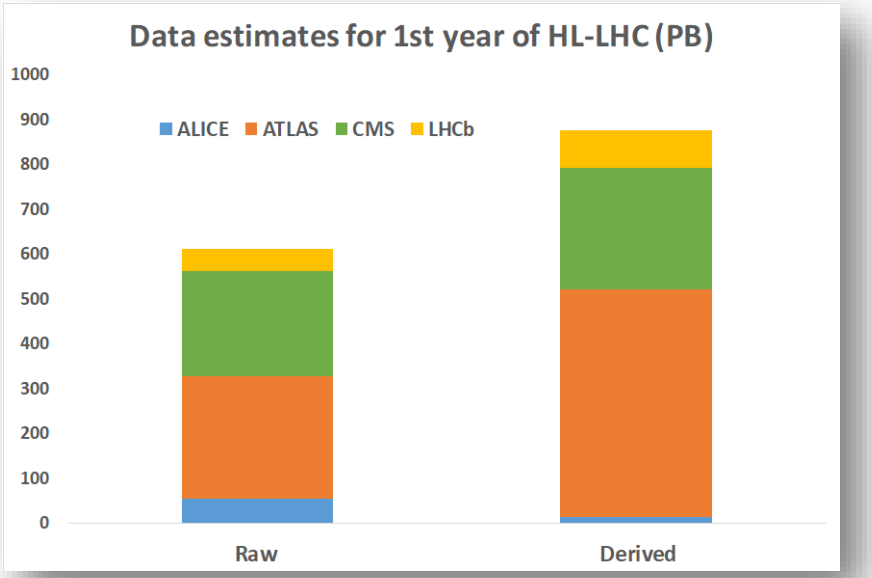


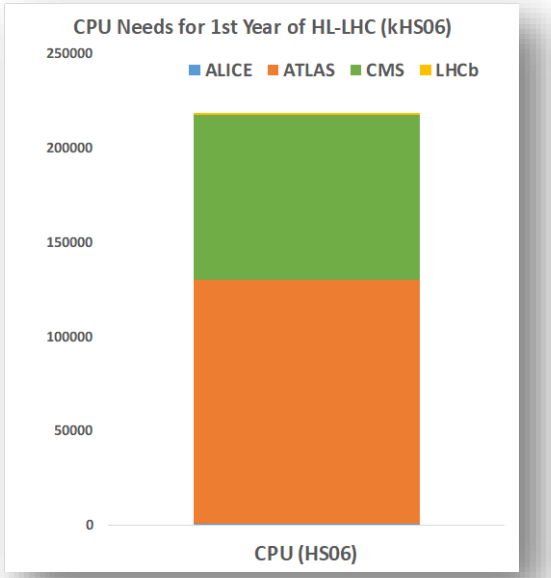
离线数据
~ 80PB per year
~ 全球分布处理

过滤
判选

在线
~ 40M Hz collisions
~ 1PB/sec

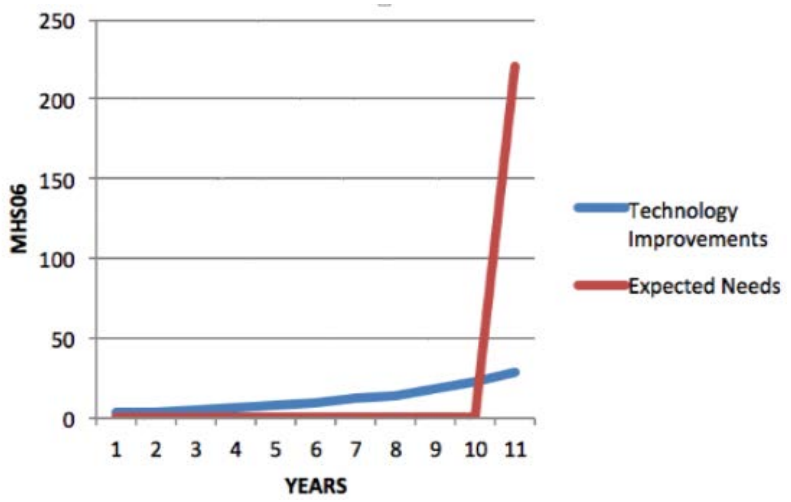
HL-LHC的计算需求





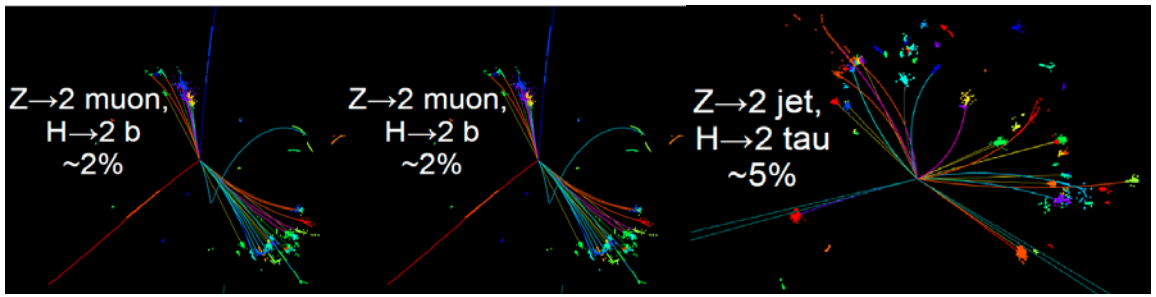
数据与计算挑战:

- 原始数据 2016: 50 PB → 2027: 600 PB
- 处理后的数据 (1 copy): 2016: 80 PB → 2027: 900 PB
- ↓
- 10倍以上的存储需求
- 60倍以上的计算需求
  - 假设未来计算机技术平均每年有20%的提升，10年将有6倍的技术提升，我们仍需要在计算资源方面增加越10倍左右的投入



高能物理实验计算

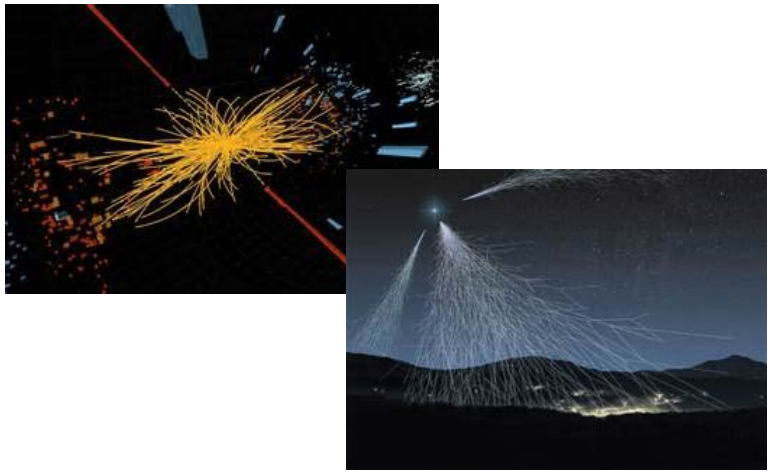
- 大数据：多次测量的随机过程（多次独立实验）
  - 随机变量空间很大：产生的末态粒子极其丰富；
  - 精确测量需要大样本：大数据
- 大计算：末态的模式复杂（随机变量）
  - 参数估计：拟合及误差估计；
  - 物理图像还原非常复杂：图像处理、模式识别技术；等等

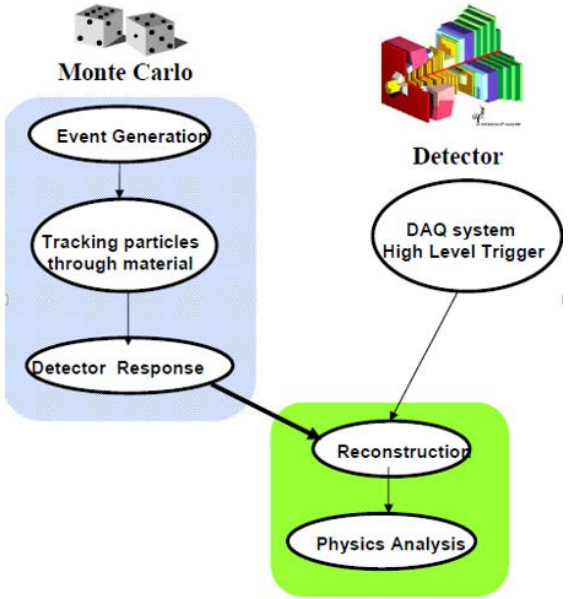


高能物理领域已经步入EB级的大数据时代

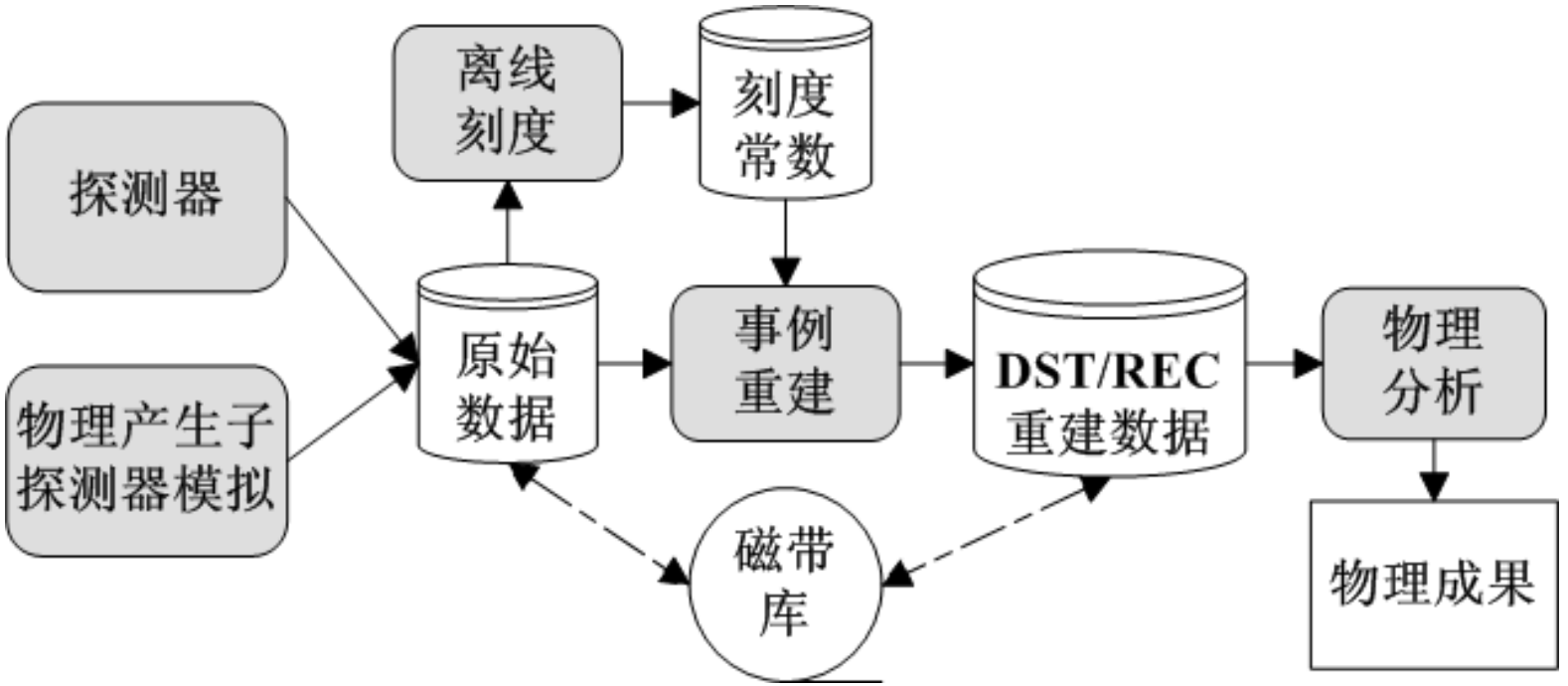
数据处理过程

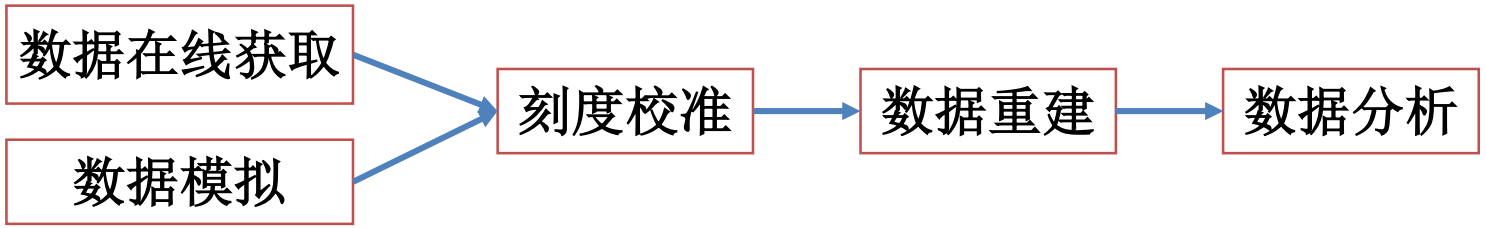
- **事例：**一次粒子对撞或者一次粒子间的相互作用
  - 粒子物理研究的基本对象
- 探测器记录事例，产生原始数据
  - 以二进制格式记录的探测器信号
  - 由计算机产生模拟实验的蒙特卡罗模拟数据，数字化
- 事例重建
  - 读出Raw/MC Raw数据，处理后产生相关物理信息，如动量、对撞顶点等；
- 数据分析
  - 由上千个属性组成的Event文件，提供物理学家进行分析，并最后产生物理结果



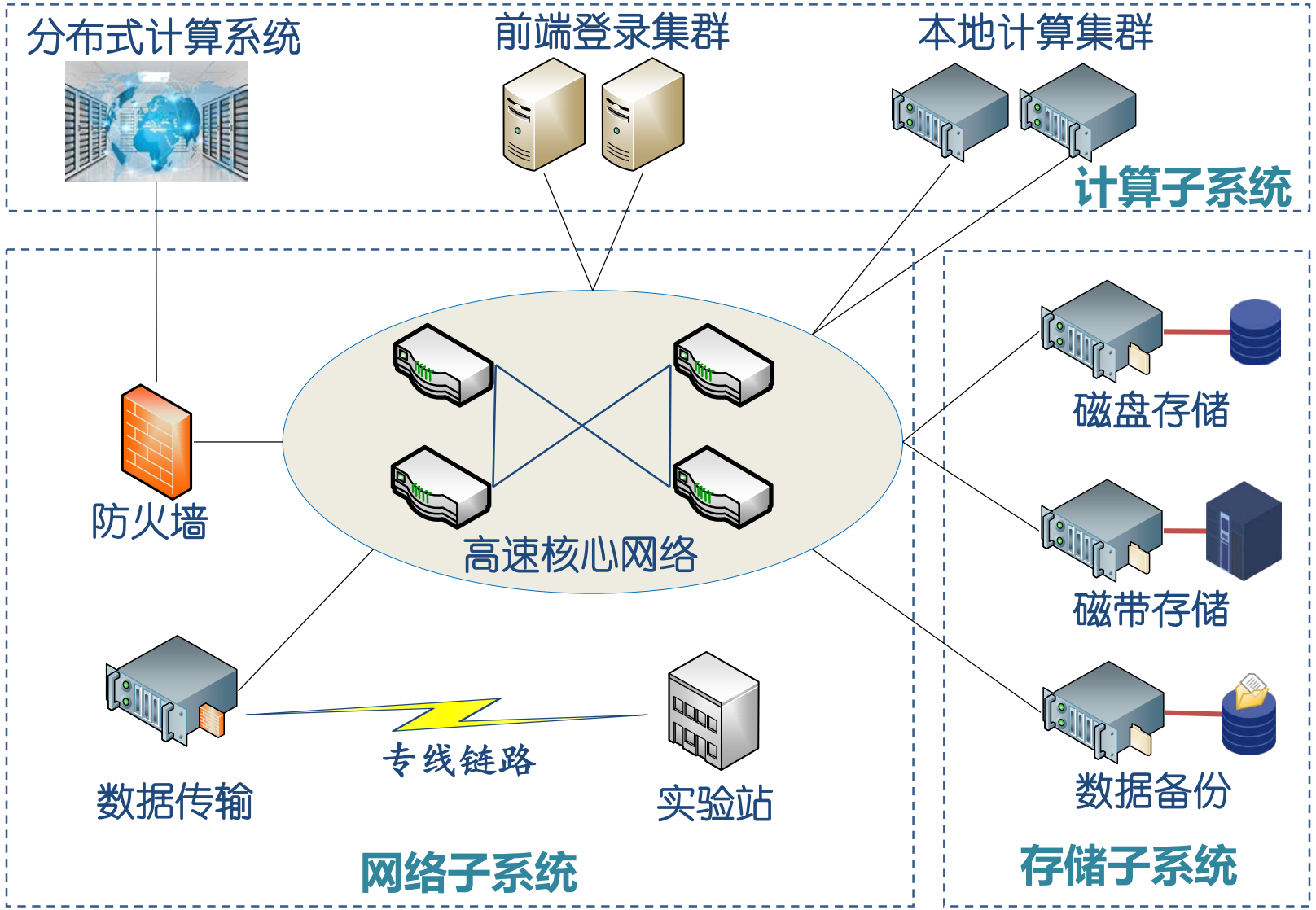


离线数据处理流程


探测器
物理产生子
探测器模拟
离线
刻度
刻度
常数
原始
数据
事例
重建
DST/REC
重建数据
物理
分析
物理成果
磁带
库

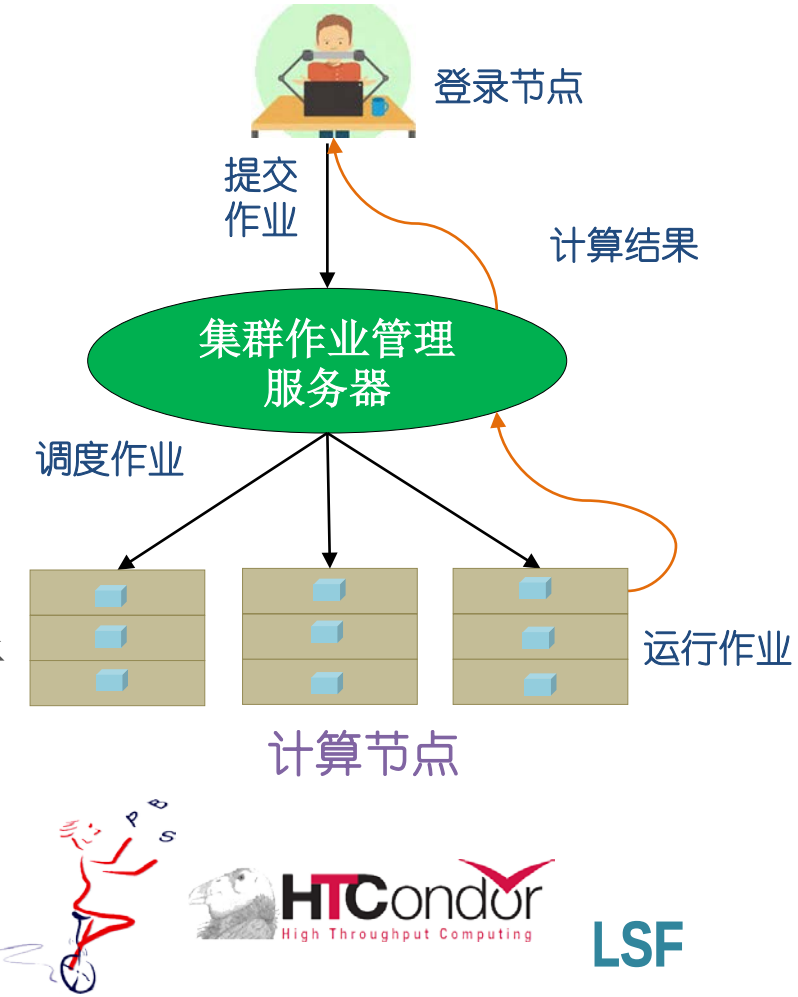

数据在线获取
数据模拟
刻度校准
数据重建
数据分析

典型的高能物理计算平台



本地计算集群

- 管理计算节点，调度作业
- 提供用户提交作业接口
- PBS
  - 开源，简单，历史悠久
  - OpenPBS，PBS Pro，Torque
  - IHEP在2016年以前的调度系统
- HTCondor
  - HTC：High Throughput Computing
  - 开源，更好的性能，更多的功能，调度算法更为公平
  - **IHEP现有调度系统**
- SLURM：高性能计算调度
  - HPC：High Performance Computing
  - GPU、MPI等作业调度
- LSF：商业调度软件

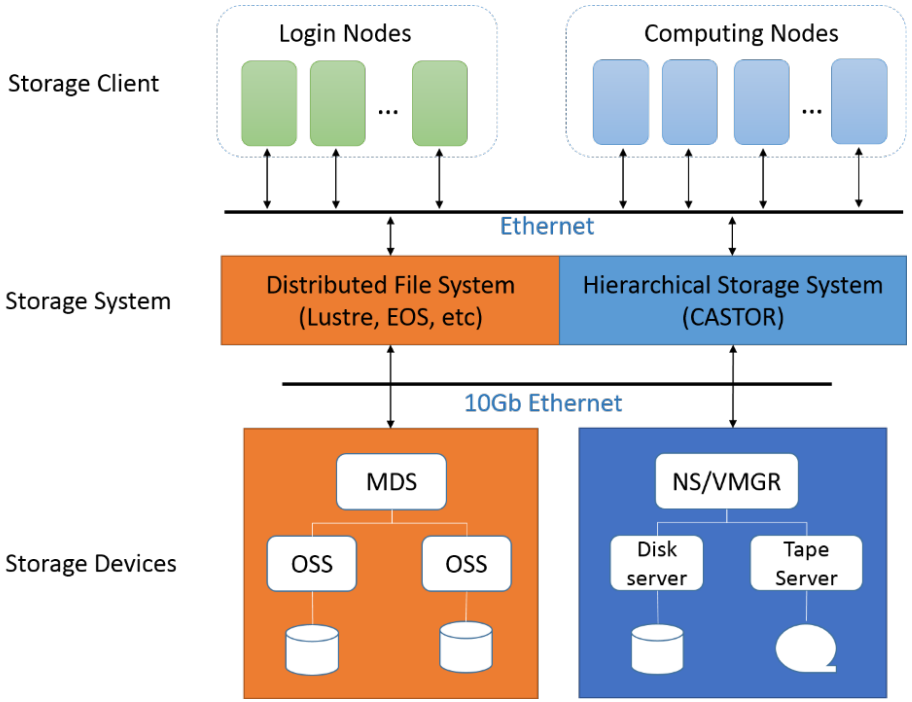


存储系统

- 磁带存储系统
  - 将顺序设备映射成类似于存储系统的树形目录
  - CERN CASTOR, enstore等开源软件
  - TSM等商业软件
- 磁盘存储系统
  - 分布式文件系统

    Lustre、**EOS**、BeeGFS, GPFS、…
  - 应用层存储系统

    dCache、HDFS、EOS、…
- 用户目录
  - AFS
- 软件库共享系统
  - AFS, CVMFS



高能所计算集群

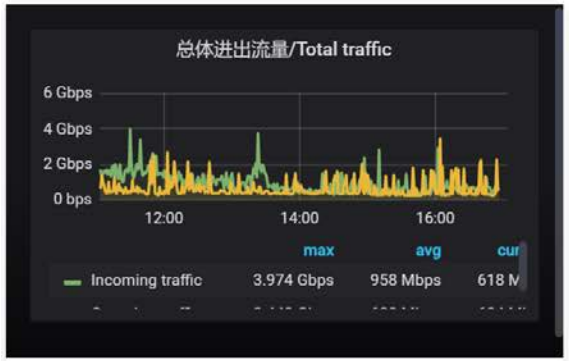

~25000CPU核
HTCondor, Slurm


~30PB磁盘
Lustre, EOS等

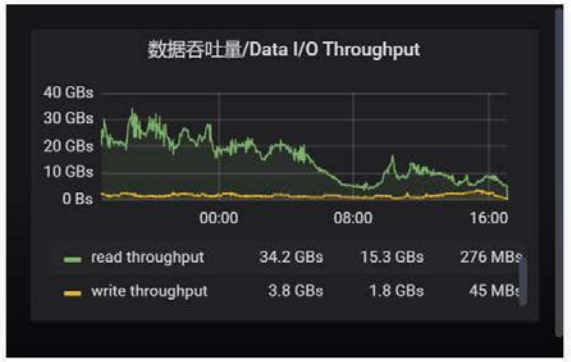

~50PB磁带库
CASTOR


2Tbps计算网络
10Gbps国际带宽
IPV4/V6双链路

运行监视



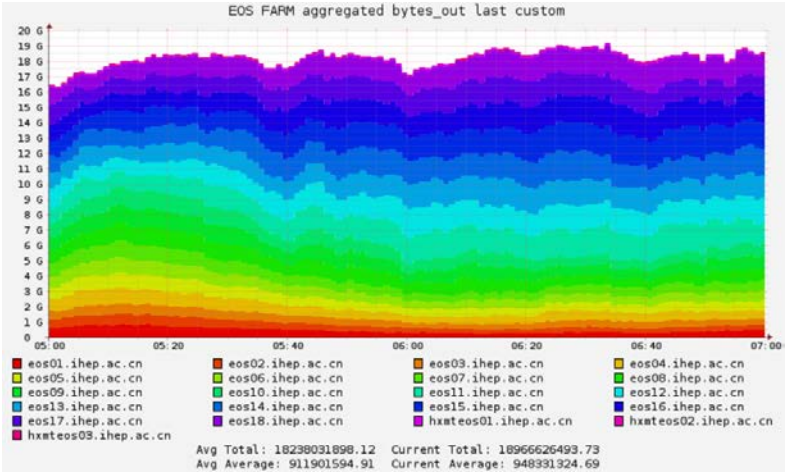
广域网流量



数据存储读写速率

计算作业执行概况

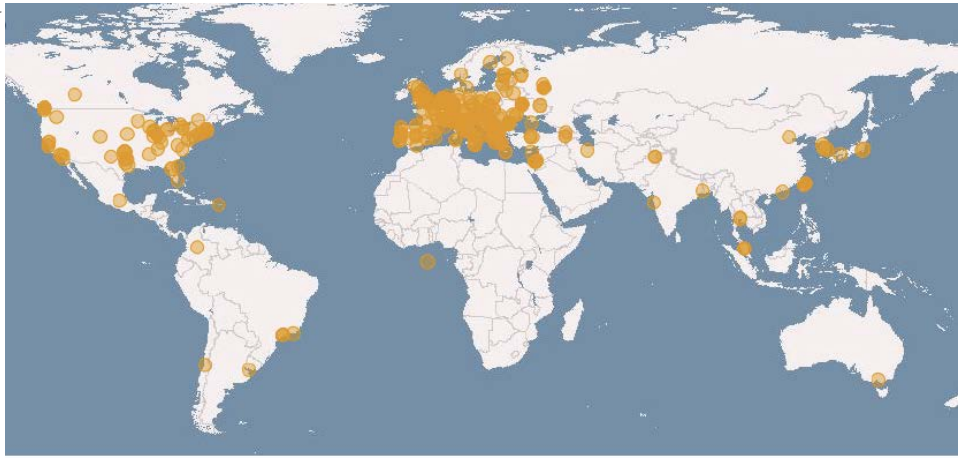
公共集群作业运行情况

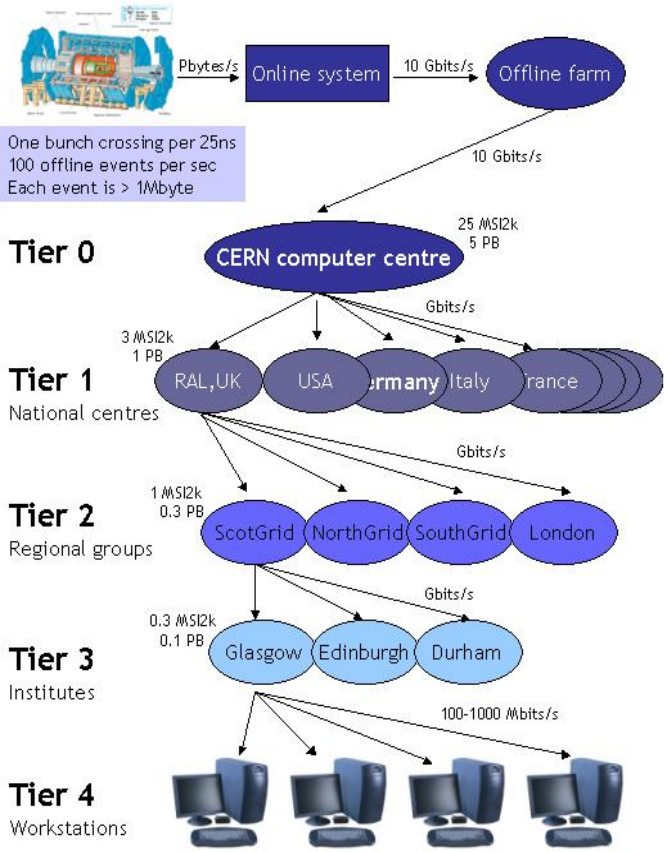


EOS聚合带宽

WLCG网格

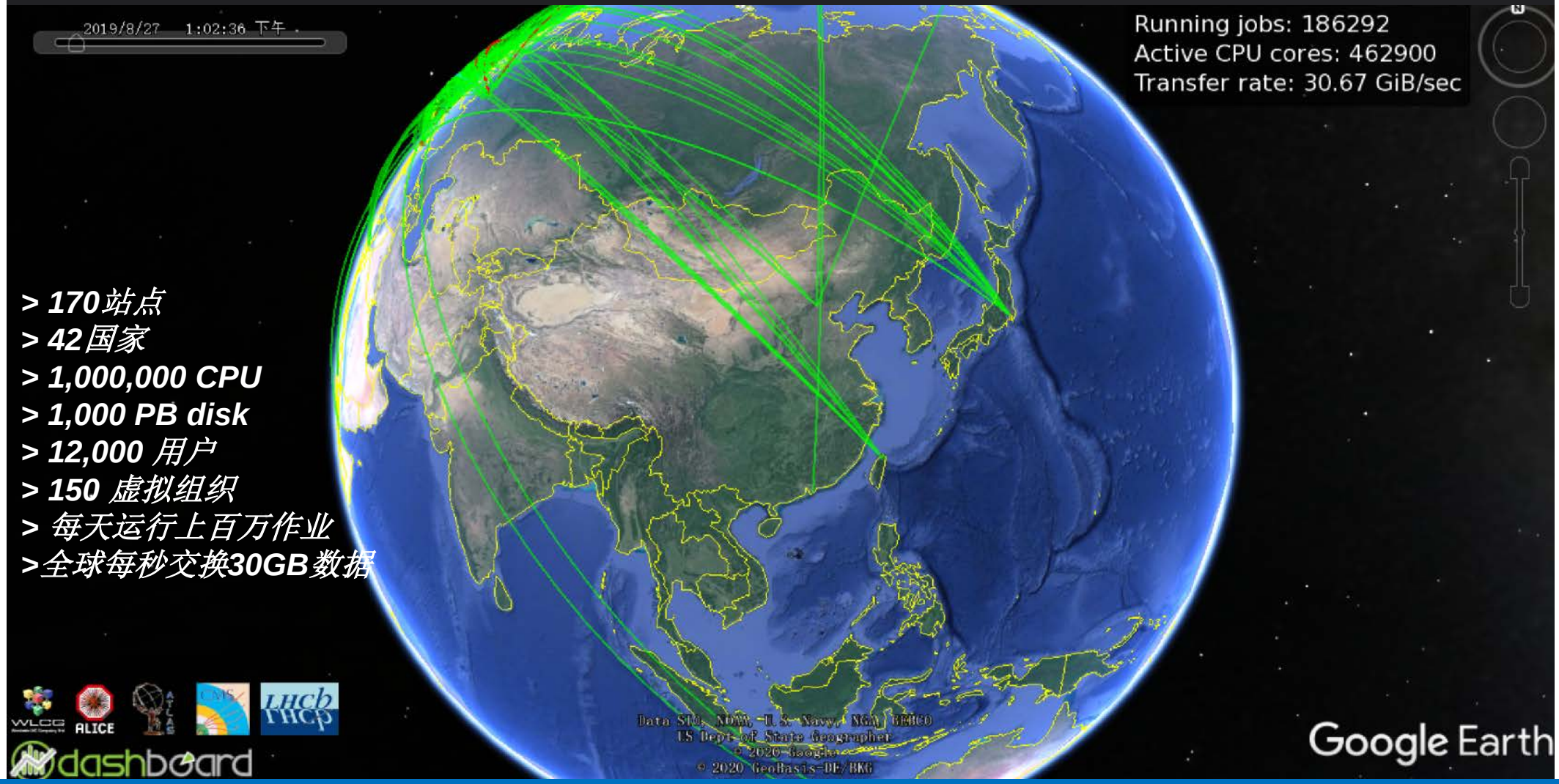
- WLCG: WorldWide LHC Computing Grid
- Tier 0: CERN
  - 接收原始数据，保存在磁带系统中，并进行第一遍数据重建
  - 向Tier1分发数据
- Tier1: 15个
  - 提供原始数据备份
  - 执行数据重建、分析等任务
  - 提供数据分发等网格服务
- Tier2: 149个
  - 执行模拟、数据分析等任务
- Tier3：本地系统





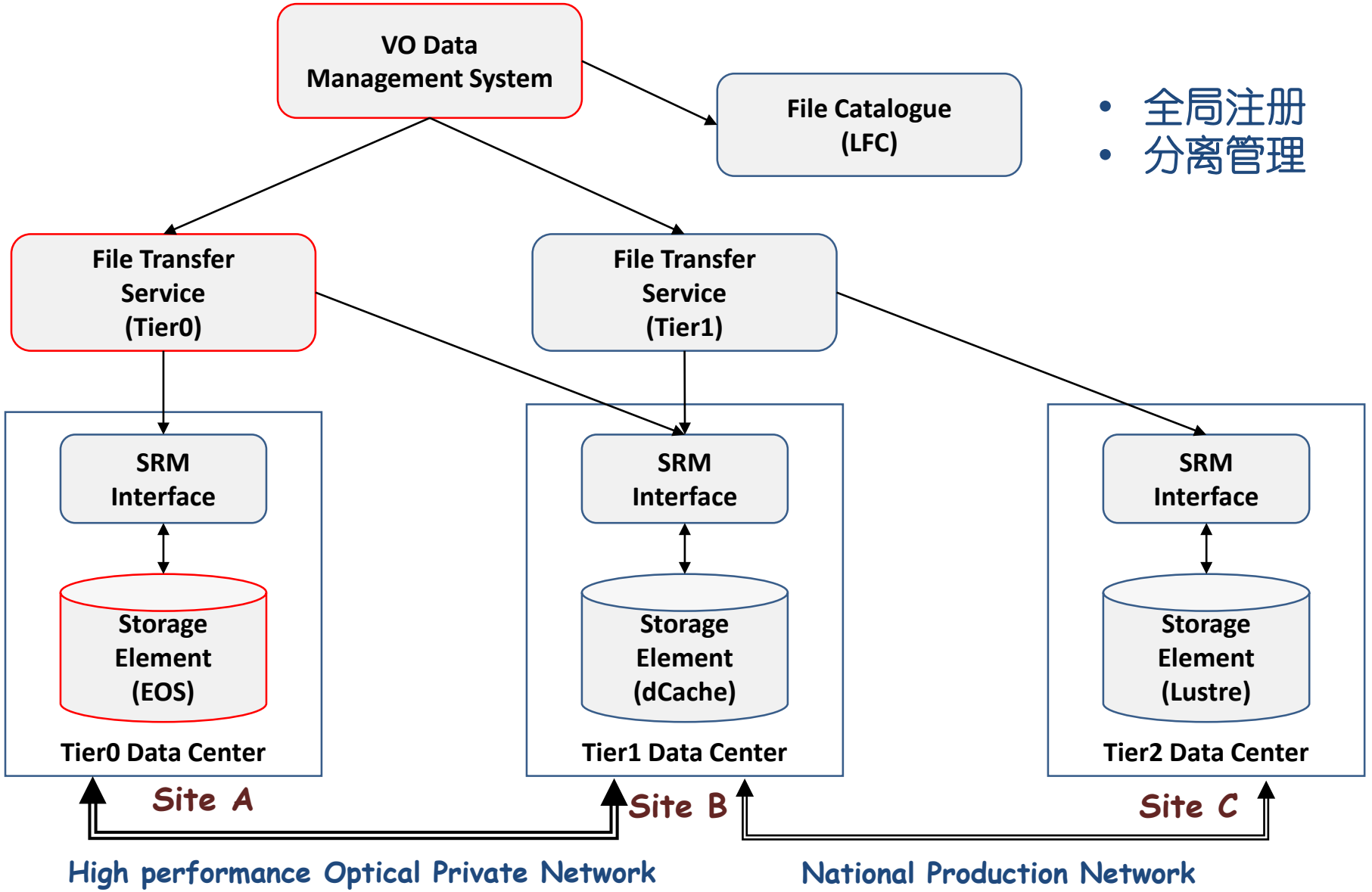
WLCG网格站点

vo	Used	Free	Total
ALICE	151.9 PB	28.8 PB	180.7 PB
ATLAS	488.4 PB	64.7 PB	553.1 PB
CMS	235.1 PB	30.9 PB	266.0 PB
LHCb	102.8 PB	22.5 PB	125.3 PB



- *> 170站点*
- *> 42国家*
- *> 1,000,000 CPU*
- *> 1,000 PB disk*
- *> 12,000 用户*
- *> 150 虚拟组织*
- *> 每天运行上百万作业*
- *>全球每秒交换30GB数据*

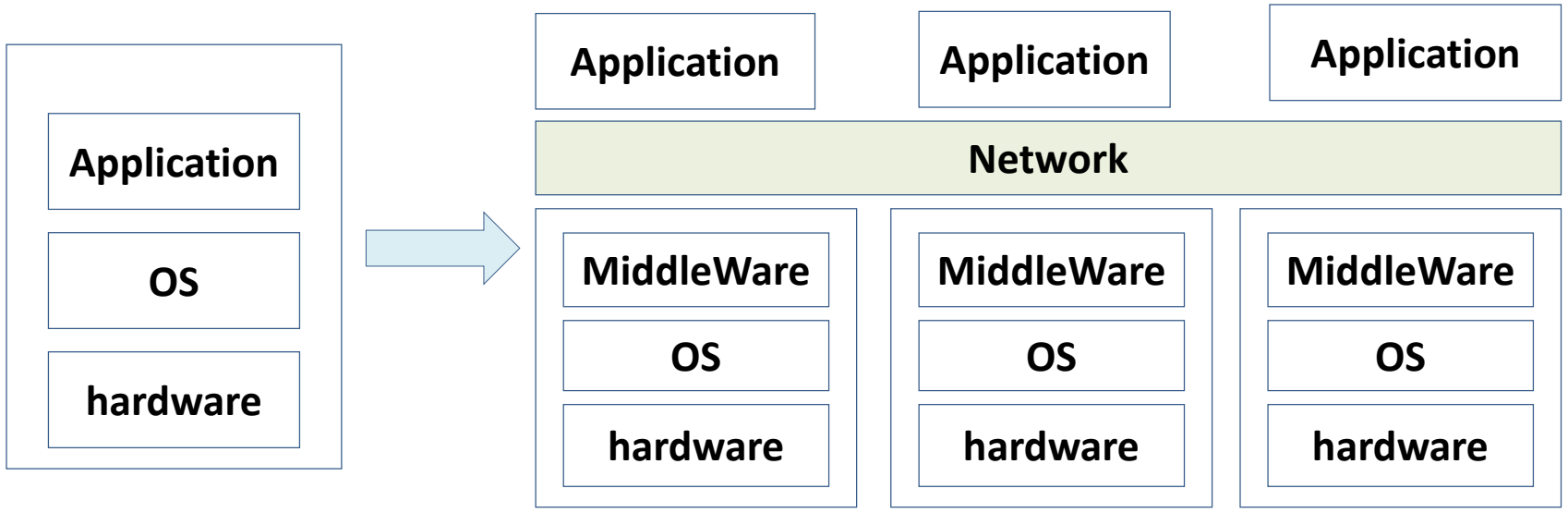
网格数据管理



- 全局注册
- 分离管理

分布式系统

- 网络存储、集群、网格等都是分布式系统
- 什么是分布式系统
  - 由一组通过网络进行通信、为了完成共同的任务而协调工作的计算机节点组成的系统
  - 为了用单个计算机无法完成的计算、存储任务
  - 其目的是**利用更多的机器，处理更多的数据**



分布式系统透明性

- 单一系统映像，期待像单机一样使用

访问透明性	位置透明性	迁移透明性
重定位透明性	复制透明性	并发透明性
故障透明性	持久化透明性	安全透明性

一些错误认识

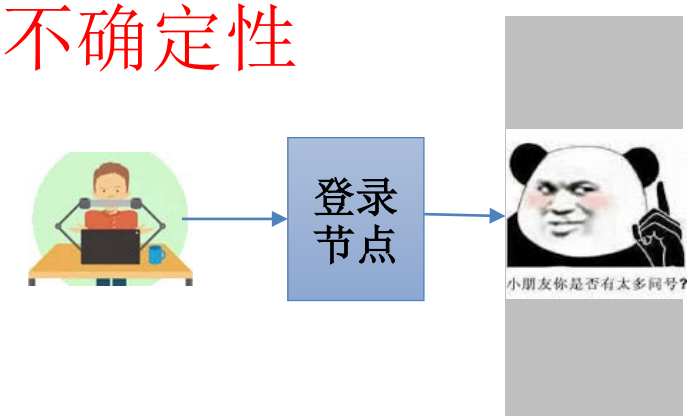
Fallacies of distributed computing

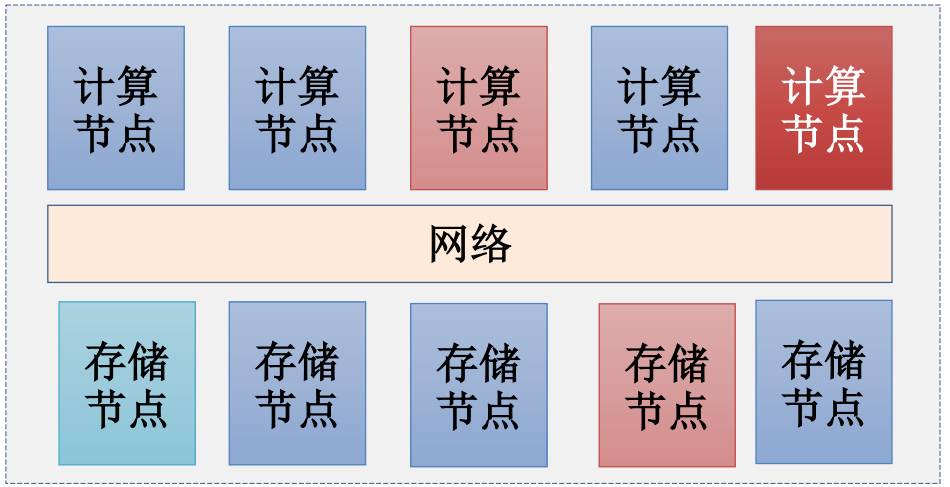
- The network is reliable;
- Latency is zero;
- Bandwidth is infinite;
- The network is secure;
- Topology doesn't change;
- There is one administrator;
- Transport cost is zero;
- The network is homogeneous.

分布式系统异常

- 节点异常：宕机、高负载、无响应等
- 网络异常：消息丢失；消息乱序；数据错误
- 分布式三态：Remote Procedure Call

  成功、失败、未知
- 存储节点故障：硬盘损坏
- 节点亚健康等其它异常
- 最大的挑战：不确定性





分布式系统的指标

- 性能
  - 吞吐能力 → HTC
  - 响应延迟 → HPC
  - 并发能力 → High QPS

相互制约

- 可用性
  - 停机时间/正常时间
  - 失败次数/成功次数

→ 衡量系统鲁棒性

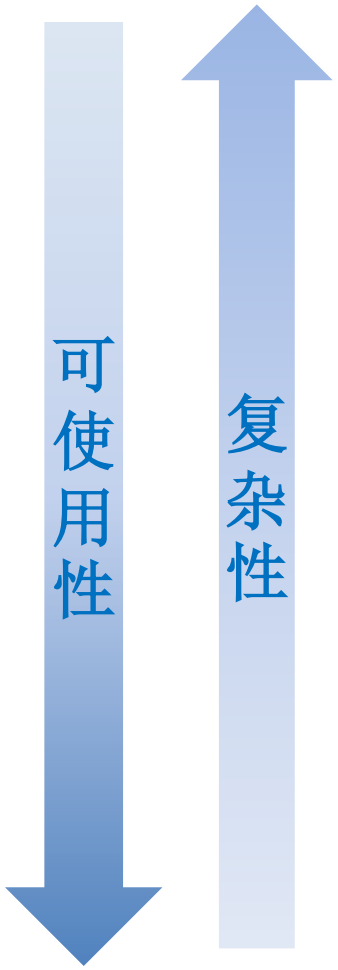
- 可扩展性
  - 性能（吞吐、延迟、并发）
  - 容量、计算能力等

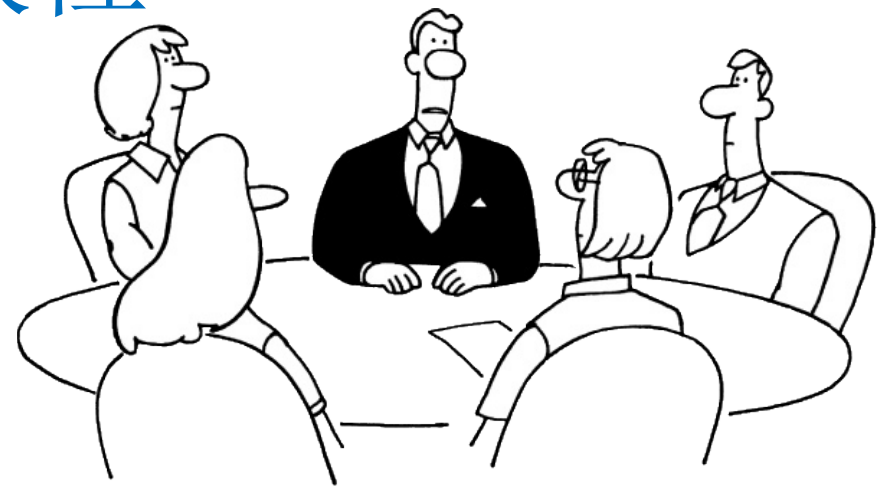
→ 追求线性扩展

- 一致性
  - 副本、缓存、分片任务等
  - 一致性越强，系统实现越复杂，用户使用越简单

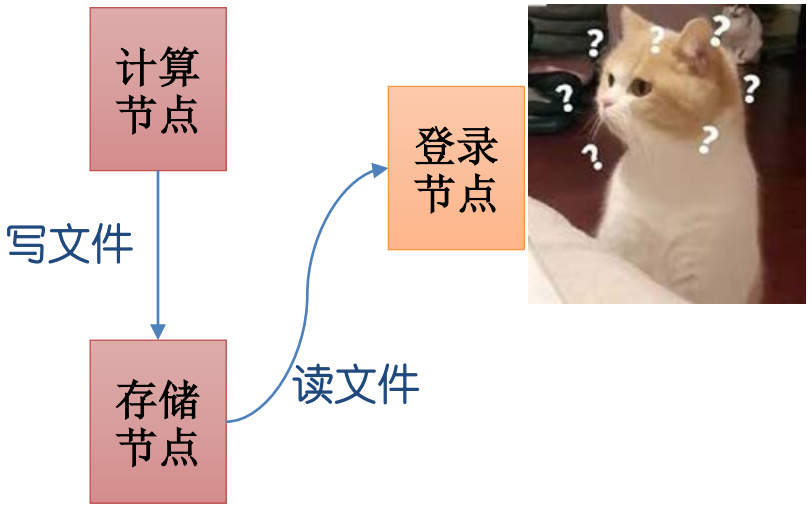
副本一致性

- 强一致性
- 单调一致性
- 会话一致性
- 最终一致性
- 弱一致性





"Whew! That was close!
We almost decided something!"



分布式系统的应用优化

- 使用时牢记：分布式系统不是单机，不确定性
- 内存分配：计算节点“非我独占”，还有好多竞争者
- 文件管理：所有操作均通过网络，不是在本地
- IO性能：内存缓存，异步IO，并行IO，…
- 错误处理：所有操作尽可能判断结果，防止crash

C/C++内存管理

- 一个由C/C++编译的程序占用的内存分为以下几个部分

  1、栈区（stack）： 由编译器自动分配释放，存放函数的参数值，局部变量的值等

  2、堆区（heap） ： 一般由程序员分配释放，若程序员不释放，程序结束时可能由OS（操作系统）回收。

  3、全局区（静态区）（static）：全局变量和静态变量的存储是放在一块的，初始化的全局变量和静态变量在一块区域，未初始化的全局变量和未初始化的静态变量在相邻的另一块区域。程序结束后由系统释放。

  4、文字常量区 ：常量字符串就是放在这里的。程序结束后由系统释放。该区不可写。

  5、程序代码区：存放函数体的二进制代码。

栈和堆

栈

- 由编译器自动分配和释放
- 存放函数的入参，局部变量
- 速度较快
- 由相对较高的地址向相对低的地址增长
- 栈的大小受限

堆

- 由程序员分配和释放，若程序不释放，程序结束时系统收回
- 速度慢，容易产生碎片
- 增长方向与栈区相反
- 谁分配，谁释放（内存泄漏）
- 建议使用tcmalloc或者jemalloc

一个例子

```
int  a=0; //全局初始化区
char *p1; //全局未初始化区
int  main()
{
  int  b; //栈
  char  s[]="abc"; //栈
  char  *p2; //栈
  char  *p3="123456";//123456在常量区，p3在栈
  static int c = 0;//全局（静态）初始化区
  char  *p4; //栈

  p1 = (char  *)malloc(10);  //堆区
  p2 = (char  *)malloc(20);
  p4 = strdup(p3);

  free(p1);
  free(p2);

  //free(p3);

  free(p4);
}
```

```
*** Error in `./a.out': free(): invalid pointer: 0x00000000004006c0 ***
======= Backtrace: =========
/lib64/libc.so.6(+0x81489)[0x7f124ce9b489]
./a.out[0x40061a]
```

Strdup调用malloc，数据也在堆区，必须要释放

栈大小

- linux下默认的堆栈空间大小是8M或10M，不同发行版本可能不一样
- 分配过大的栈空间（比如数组），会导致段错误
- 两个函数互相调用，局部变量的栈空间不能释放，会导致程序crash

```
[root@lhmteos03 chyd]# ulimit -a
core file size          (blocks, -c) unlimited
data seg size           (kbytes, -d) unlimited
scheduling priority             (-e) 0
file size               (blocks, -f) unlimited
pending signals                 (-i) 511770
max locked memory       (kbytes, -l) 64
max memory size         (kbytes, -m) unlimited
open files                      (-n) 1024
pipe size            (512 bytes, -p) 8
POSIX message queues     (bytes, -q) 819200
real-time priority              (-r) 0
stack size              (kbytes, -s) 8192
cpu time               (seconds, -t) unlimited
max user processes              (-u) 511770
virtual memory          (kbytes, -v) unlimited
file locks                      (-x) unlimited
```

```
int  main()
{
    char a[8388608] = {0};
}

[root@lhmteos03 chyd]# ./a.out
Segmentation fault (core dumped)
```

（1）主线程和子线程可以获得的stack size不相同

（2）静态区内存不释放

查看程序内存占用

```
  PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
380936 root      20   0   18.0g   3.2g    396 R  96.7   2.5   0:12.45 a.out

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
380263 root      20   0   18.0g   8.0g    396 S   0.0   6.4   0:31.54 a.out

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
383584 root      20   0 8392816   8.0g    352 S   0.0   6.4   0:29.33 a.out
```

- VIRT：virtual memory usage 虚拟内存
- RES：resident memory usage 常驻内存
- SHR：shared memory 共享内存

```
char a[8589934592] = {0}; //8GB
int main(void)
{
    int64_t i;
    char *p = (char *)malloc(10737418240); //10GB
    for (i=0;i<8589934592;i++) {
        a[i] = 1;
    }
    free(p);
    sleep(30);
    return 0;
}
```

静态区，申请8G

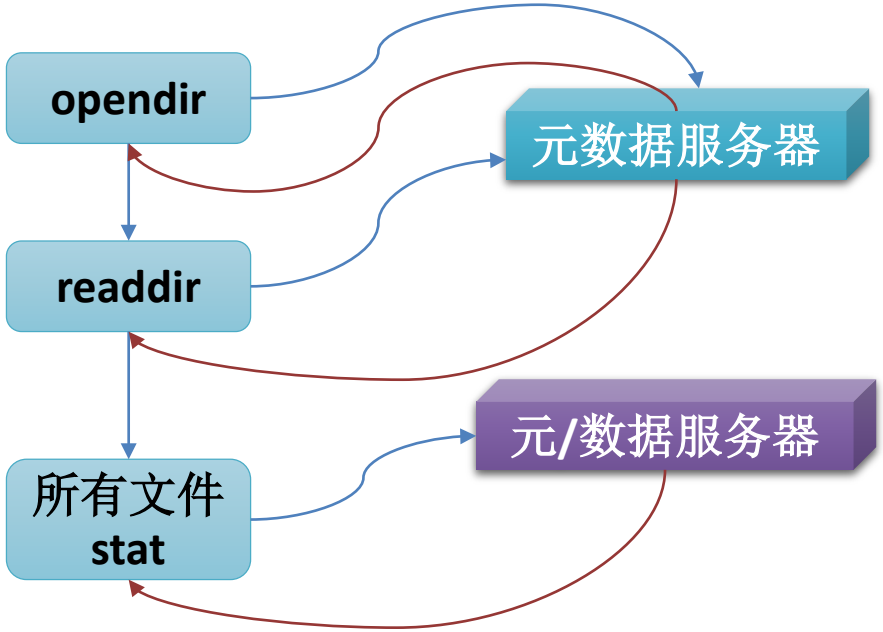
堆区，申请10G

使用

释放10G

元数据管理

- 显示目录或者文件属性




```
[chyd@lxslc702 bigdir]$ stat gen.sh
  File: 'gen.sh'
  Size: 119             Blocks: 1          IO Block: 512
Device: 43h/67d Inode: 44260157139451904  Links: 1
Access: (0755/-rwxr-xr-x)  Uid: (60003/    chyd)   Gid:
Access: 2019-04-10 11:10:55.843875333 +0800
Modify: 2019-04-10 11:10:55.843875333 +0800
Change: 2019-04-10 11:10:55.841617304 +0800
 Birth: -
[chyd@lxslc702 bigdir]$ ls -l
total 1
drwxr-xr-x 1 chyd u07    66977792 Dec 13  2019 100
drwx------ 1 chyd u07  9816637561 Dec 13  2019 15k
drwx------ 1 chyd u07 19677184121 Aug 12 14:46 30k
drwxr-xr-x 1 chyd u07   309067776 Dec 13  2019 500
-rwxr-xr-x 1 chyd u07         119 Apr 10  2019 gen.sh
```

本地

```
real    0m7.492s
user    0m0.245s
sys     0m0.487s
```

#time ls -l 500

```
real    0m0.683s
user    0m0.017s
sys     0m0.027s

real    0m2.700s
user    0m0.015s
sys     0m0.048s
```

#time ls -l 15k

```
real    0m31.513s
user    0m0.337s
sys     0m0.834s

real    1m15.315s
user    0m0.351s
sys     0m1.288s
```

#time ls -l 30k

```
real    0m49.856s
user    0m0.683s
sys     0m1.589s

real    2m23.301s
user    0m0.708s
sys     0m2.385s
```

优化方法

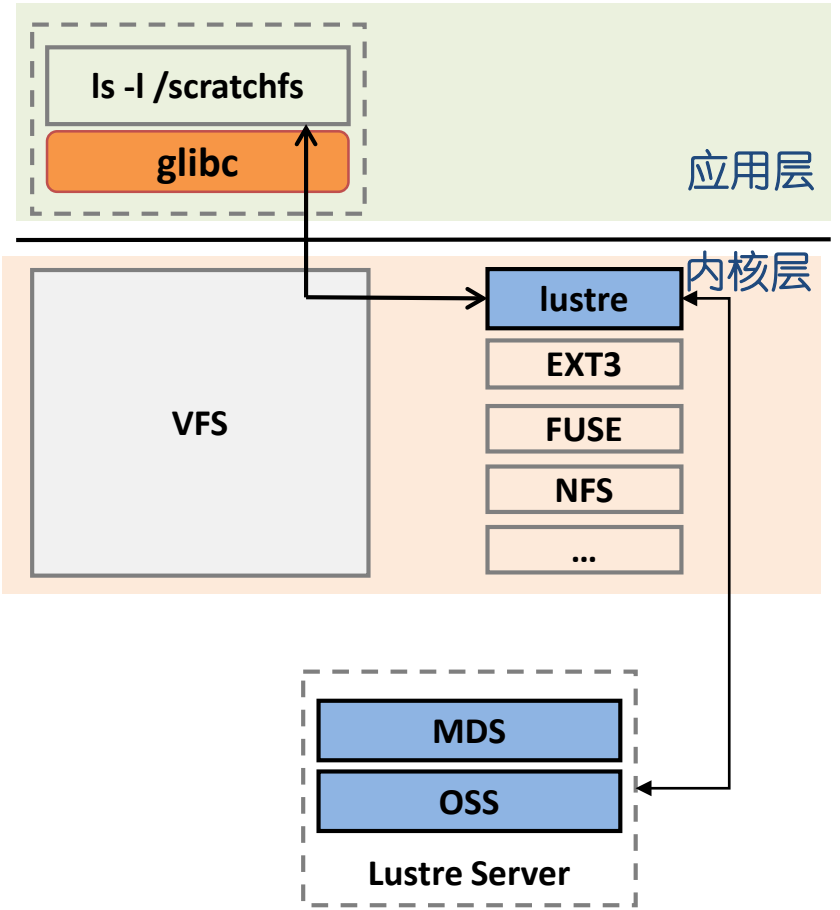
- 不要做：
  - 1）在一个目录下放置大量文件
  - 2）遍历整个目录
  - 3）使用通配符
- 建议
  - 使用离线数据处理框架，比如BOSS、SNiPER等
  - 将需要处理的文件整理成列表，针对列表进行处理
  - 将大目录分成多个小目录
  - 如果确实需要了解某个文件的信息，可以对单个文件ls或者stat

```
[chyd@lxslc702 bigdir]$ time ls -l 30k/ff.122
-rw-r--r-- 1 chyd u07 1048576 Aug 12 14:36 30k/ff.122

real    0m0.009s
user    0m0.003s
sys     0m0.003s
```

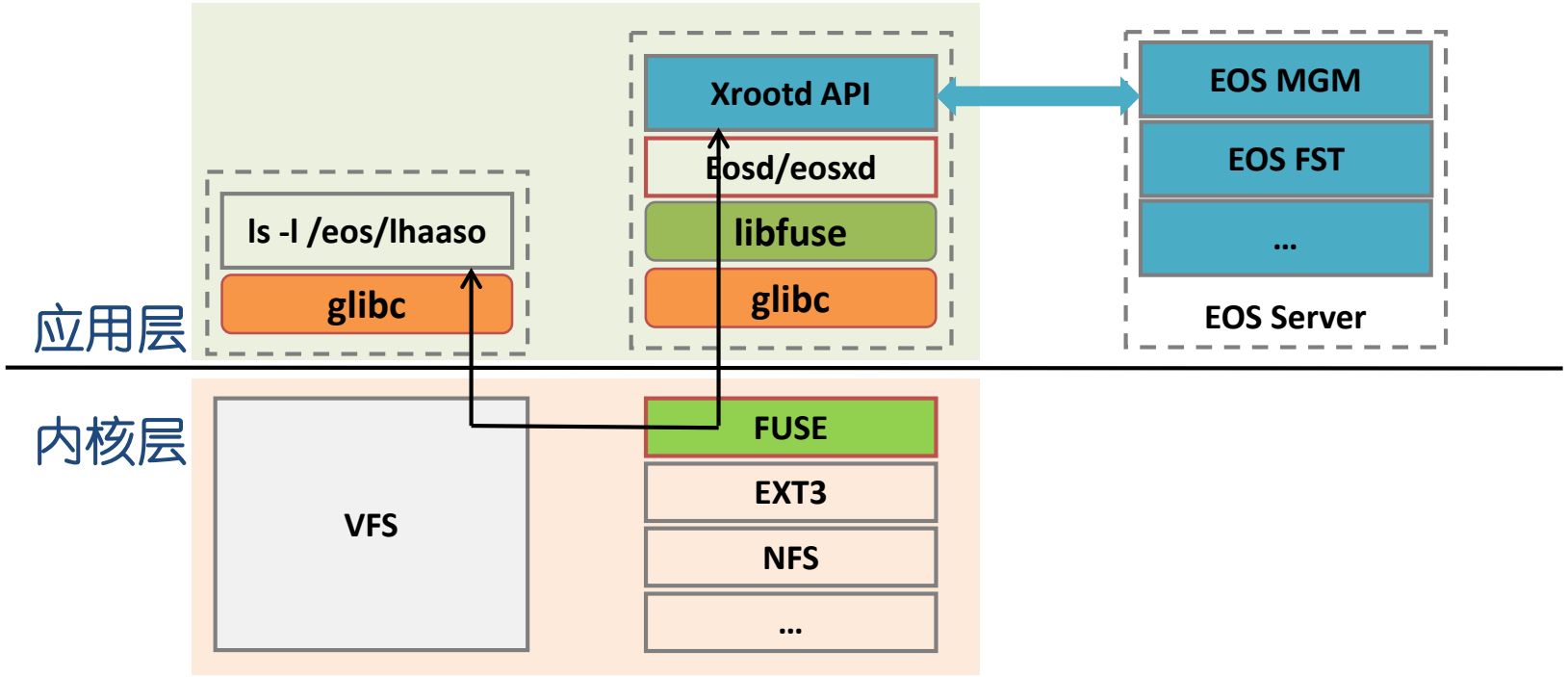
数据访问协议

- 内核级文件系统：Lustre
  - 并行性好，文件系统语义支持好
  - 内核依赖，管理复杂
- 应用级文件系统：EOS Fuse
  - 提供文件系统语义
  - 并行性支持差
  - 目前稳定性差
- 应用级数据访问：Xrootd
  - 基于文件访问API，不提供文件系统语义
  - 稳定性好，不受文件系统限制
  - 使用不太方便



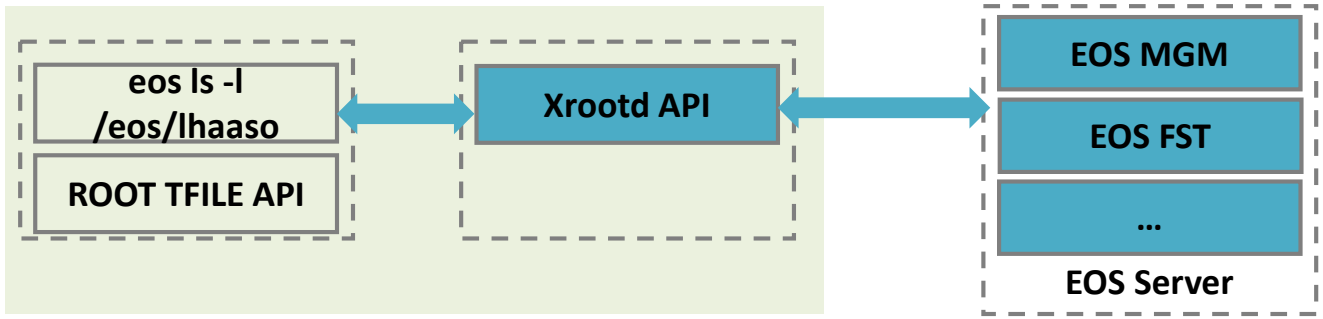
EOS文件系统

- 在XROOTD基础上开发文件系统接口，类似于本地文件系统
- 对于物理分析来说并不高效，但是比较灵活
- 基于FUSE（ Filesystem in Userspace ）实现，FUSE是Linux内核标准模块
  - 比内核级文件系统（eg. Lustre）实现简单
  - 但是并不是最高效的
  - 任何FUSE模块或者eosd的失败都会导致作业的失败



应用层访问接口

- 命令行方式，比如eos ls，直接调用xrootd API来访问EOS服务器，绕过任何内核模块
- 调用ROOT TFILE类的应用软件，也可以直接调用xrootd API
- 这种方式完全工作在应用层，不受文件系统及内核的影响，稳定好
- 用户使用不太灵活，没有本地文件系统的接口，cat等命令无法工作




```
[chyd@lxslc702 ~]$ eos ls -lh /eos/lhaaso
drwxr-sr-+   1 lhaasore lhaaso      419.26 G Dec 15  2019 cal
drwxr-sr-+   1 lhaasore lhaaso      673.86 T Dec 15  2019 decode
drwxr-sr-+   1 root     root         51.41 T Dec 15  2019 experiment
drwxr-sr-+   1 lhaasore lhaaso      526.88 G Dec 15  2019 monitor
drwxr-sr-+   1 root     root        747.41 T Dec 15  2019 raw
drwxr-sr-+   1 lhaasore lhaaso      219.03 T Dec 15  2019 rec
drwxr-sr-+   1 root     root        906.68 T Dec 15  2019 simulation
```

Xrootd使用

- 首先，物理软件（比如BOSS或者SNiPER ）调用ROOT库 TFile:: Open，比如：

```
TFile*  inputFiles[m_fileNum] =  TFile::Open(m_fileNames[m_fileNum].c_str(),"READ");
```

  - 注意：以下两种调用方式不支持

（1）简单声明：TFile file(fn.c_str());

（2）New方法： TFile* inputFile = new TFile(m.c_str(),"READ");

- 其次，将输入输出文件采用ROOT的命令方式，比如：

```
root://eos01.ihep.ac.cn///eos/user/c/chyd/703/mc/KKpi/phsp2/KKpi_phsp_0001_boss703.rtraw
```

协议　服务器名　绝对路径

- 由于没有本地文件系统接口，脚本中不能出现通配符，不能采用相对路径，不能使用操作系统命令来遍历目录，比如for f in `ls /eos/lhaaso/raw/wcda`之类的语句。但是可以使用eos的命令行工具，比如：

```
[chyd@lxslc702 ~]$  for f in `eos ls /eos/lhaaso/raw/wcda`;do  echo $f;done
1970
2019
2020
test
```

IO性能与延迟

- 高能物理计算涉及大量数据，高性能的IO有利于提高数据处理效率
- IO延迟来自于多个方面

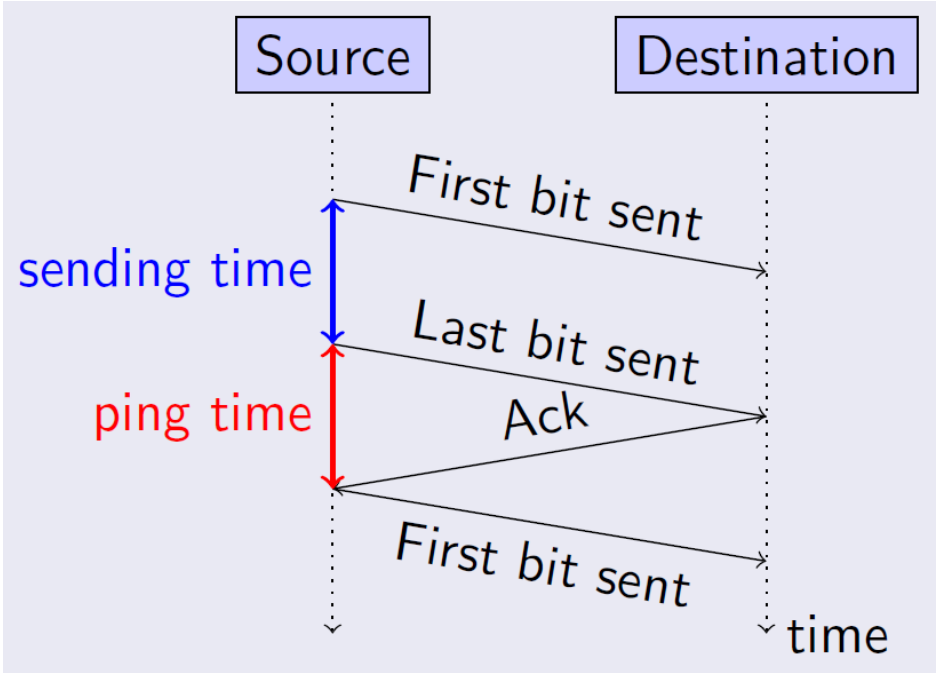
存储设备	网络	光速
• SSD (~100us) • Disk (~10ms) • Tape (~1min)	• 传输路径上路由器的数量 • ~200us per switch/router	• 光在光纤中传输更慢(折射率1.47) • ~200m us^{-1}或者20cm ns^{-1} • ~150ms between CERN and IHEP

IO延迟对性能的影响

典型的IO模式

1. 向目标发送一个数据包
2. 等待确认
3. 进入下一轮处理

$$efficiency = \frac{sending\ time}{sending\ time + ping\ time}$$



效率计算

Definitions

$$efficiency = \frac{sending\ \ time}{sending\ \ time + ping\ \ time} \tag{1}$$

$$sending\ \ time = \frac{data\ \ size}{speed} \tag{2}$$

$$ping\ \ size = speed * ping\ \ time \tag{3}$$

Gives

$$efficiency = \frac{1}{1 + \frac{ping\ \ size}{data\ \ size}} \tag{4}$$

$$data\ \ size = \frac{efficiency}{1 - efficiency} * ping\ \ size \tag{5}$$

效率计算（2）

Consequences for 10KB blocks

Usage	Speed	Latency	Ping Size	Efficiency
CC	$1\,\mathrm{GB\,s^{-1}}$	100 µs	10 kB	50%
CC	$10\,\mathrm{GB\,s^{-1}}$	100 µs	100 kB	9%
WAN	$1\,\mathrm{GB\,s^{-1}}$	10 ms	1 MB	1%
WAN	$10\,\mathrm{GB\,s^{-1}}$	10 ms	10 MB	1‰
UK-JP	$10\,\mathrm{GB\,s^{-1}}$	250 ms	250 MB	0.04‰

Data size for decent efficiency

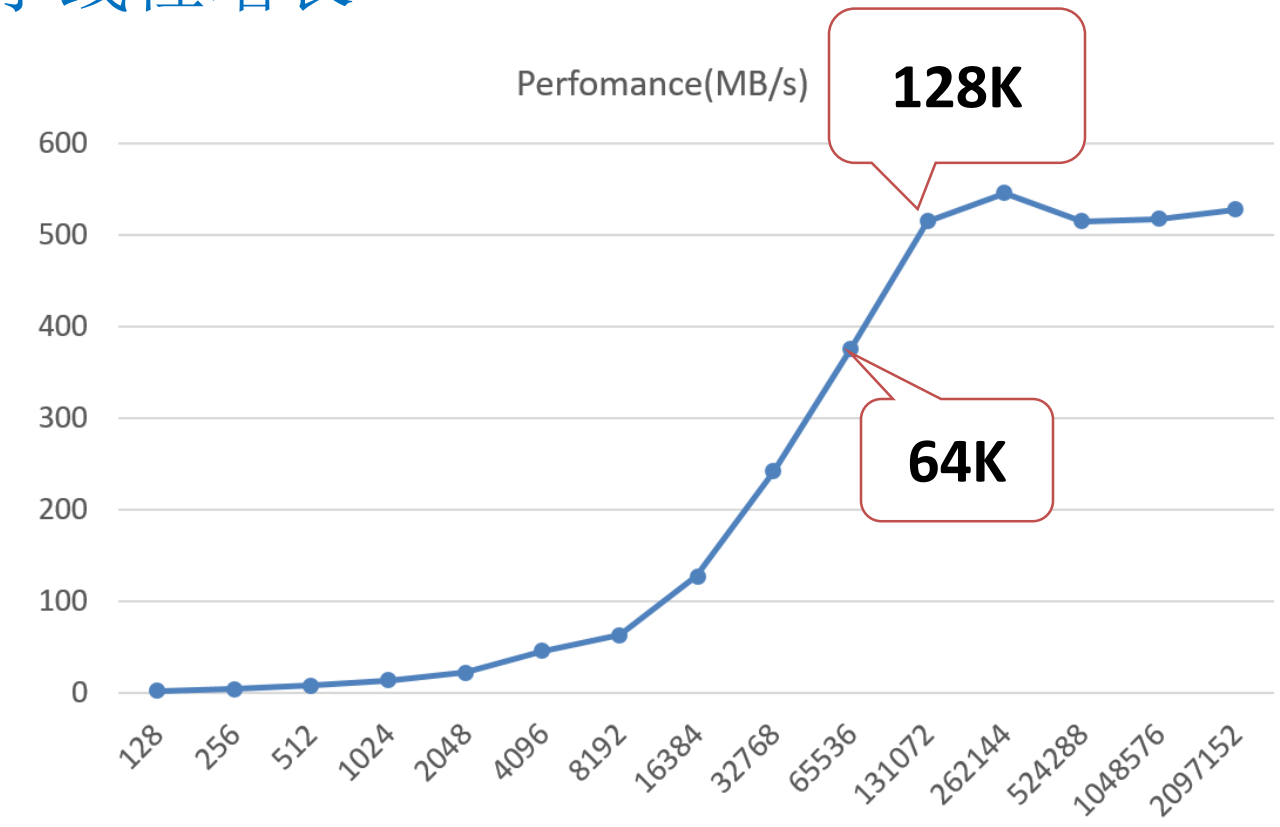
Usage	Speed	Latency	50% efficiency	91% efficiency
CC	$1\,\mathrm{GB\,s^{-1}}$	100 µs	10 kB	100 kB
CC	$10\,\mathrm{GB\,s^{-1}}$	100 µs	100 kB	1 MB
WAN	$1\,\mathrm{GB\,s^{-1}}$	10 ms	1 MB	10 MB
WAN	$10\,\mathrm{GB\,s^{-1}}$	10 ms	10 MB	100 MB
UK-JP	$10\,\mathrm{GB\,s^{-1}}$	250 ms	250 MB	2.5 GB

Remember maximum TCP packet size is 64 KiB

一个例子

- 用dd向EOS写100M数据，每次改变blocksize的大小
- Blocksize小于128K时几乎线性增长

	A	B	C	D	E	F
1	dd if=/dev/zero of=f1 bs=128 count=1048576 oflag=direct					
2						
3	block size	Perf. MB/s		count		
4	128	1.9		1048576		
5	256	3.6		524288		
6	512	7.4		262144		
7	1024	13.3		131072		
8	2048	21.8		65536		
9	4096	45.4		32768		
10	8192	62.6		16384		
11	16384	127		8192		
12	32768	242		4096		
13	65536	376		2048		
14	131072	515		1024		
15	262144	546		512		
16	524288	515		256		
17	1048576	518		128		
18	2097152	528		64		



解决方案

- 异步IO
  - 多线程处理：IO线程与数据处理线程
  - 异步IO系统调用：aio_read, aio_write, …
- 优化传输
  - 异步网络传输：设置socket连接为NONBLOCK，后续调用select, epoll等机制
  - 本地数据传输：设置较大的Block size；预读；利用内核缓存等
- 数据结构与算法
  - 列表、树、图等
  - 优化数据放置
  - 使用C++ STXXL库
  - 使用ROOT等分析框架
  - 使用SNiPER等框架！

N number of items in the problem

M number of items fitting in memory

B number of items fitting in a disk block

Fundamental Bounds of I/O complexity

- scanning : $O(\frac{N}{B})$
- searching : $O(log_B N)$
- sorting : $O(\frac{N}{B} log_{\frac{M}{B}} \frac{N}{B})$

Importance of B

- N = 256 10^6, B = 8000, 1ms disk access time
- N I/Os take 256 10^3 sec = 71 h
- $\frac{N}{B}$ I/Os take 32 s

错误处理

- 前提：分布式系统中，故障是常态
- 所有的系统调用都要做好判断，并做错误处理，比如

```
if ((s = fopen (infile, "r")) == NULL) {
    perror (infile);
    return (1);
}
if ((o = fopen (outfile, "w")) == NULL) {
    perror (outfile);
    return (1);
}

while ((c = fread (buf, 1, sizeof(buf), s)) > 0) {
    if ((rc = fwrite (buf, 1, c, o)) < c) {
        fprintf (stderr, "write error: %s\n", strerror(errno));
        fclose (s);
        return (1);
    }
}
fclose (s);
fclose(o);
```

判断输入文件是否正常打开

判断输出文件是否正常打开

判断读文件是否正常

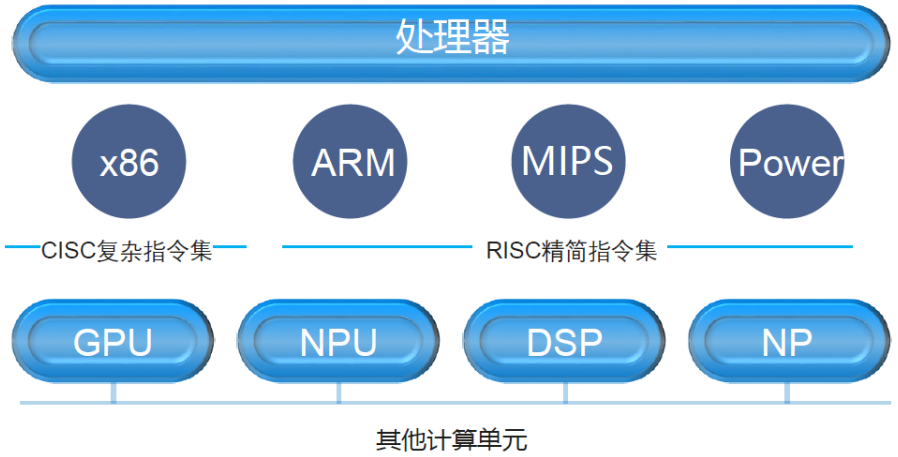
判断写文件是否正常

关闭文件，非常重要！！！

新技术展望

- 多核/众核CPU架构
  - X86（Intel、AMD等），ARM（kunpeng，phytium），MIPS，Power，…
  - 国产CPU：龙芯，申威，鲲鹏，飞腾，海光，…
- 异构加速技术
  - GPU，FPGA，NPU，DSP等
- 高性能计算
  - 多线程（OpenMP，Intel TBB等）
  - 大规模并行（OpenMPI等）
- 数据处理技术
  - 机器学习

多种计算架构并存的组合是最优的解决路径



总结

- 数据处理在高能物理科研中不可或缺，被称为“第四种科学研究范式”
- 高能物理计算涉及到计算机体系结构、计算机软件与理论、科学数据处理等多个学科方向
- 计算、存储、网络是基础设施，数据是核心，软件框架是中间件，数据分析是顶层应用，科学发现是最终目标
- 本次培训讲授离线计算、数据存储与管理、网络与安全、软件框架、软件开发工具、数据分析软件、机器学习以及GPU等技术
- 采用先进的技术，进行规范的使用，以提高数据处理的效率


我们的征途是星辰大海
仰望星空，脚踏实地
2020.5.2凌晨 拍摄于张北草原-天鹅湖