

---

# Condorを用いた グリッドスケジューリング導入実習

産業技術総合研究所

中田秀基

ここにいろいろおいてあります



</share/apps/condor-tutorial>



# Condorとは

---

- 遊休計算機を利用して独立したジョブを大量に処理するシステム

  - ▶ c.f. Grid MP 、 SETI@HOME

- 並列ジョブにも対応したバッチキューイングシステム

  - ▶ c.f. PBS Professional, Sun One Grid Engine, LSF

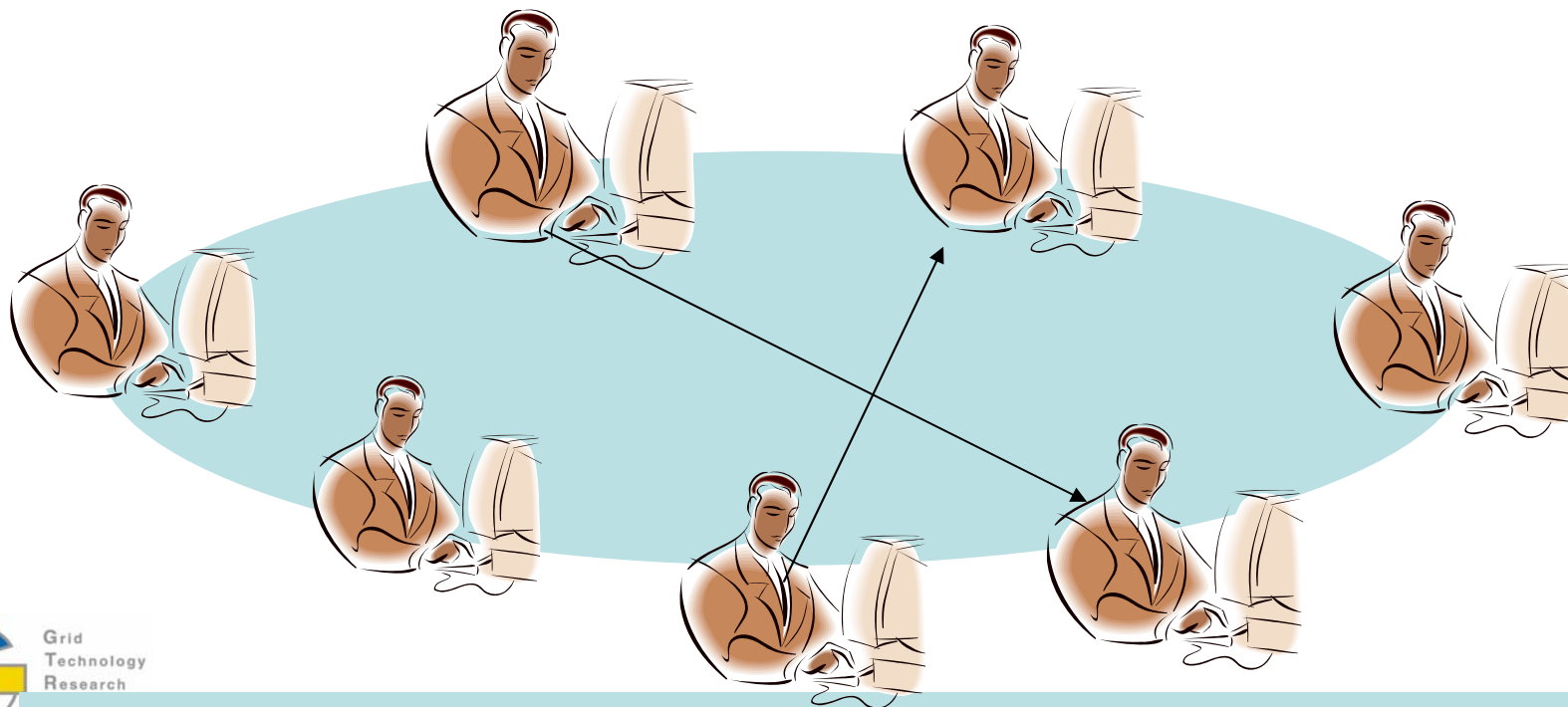
- メタスケジューラ

  - ▶ c.f. Community Scheduler Framework (Platform computing)

# Condorでできること

## ● 社内でお互いのPCを共有

- ▶ 自分のジョブを他人の計算機で実行
- ▶ 自分の計算機を他人に提供
- ▶ 「フェア」なスケジューリング
- ▶ 仕事を妨げない



# Condor の歴史

---

- 1983年 Prof. Miron Livny のPh.D Thesis
  - ▶ “The Study of Load Balancing Algorithms for Decentralized Distributed Processing Systems”
- 1985年 ウィスコンシン大学において、Condor Project 開始
- 1986年 最初のCondor システムが実装される
  
- 2005年で20周年(!)
- 予算： DoE、NASA、NIH、NSF、EU、INTEL、Micron、Microsoft、UW Graduate School



</share/apps/condor-tutorial>



# サポートアーキテクチャ

---

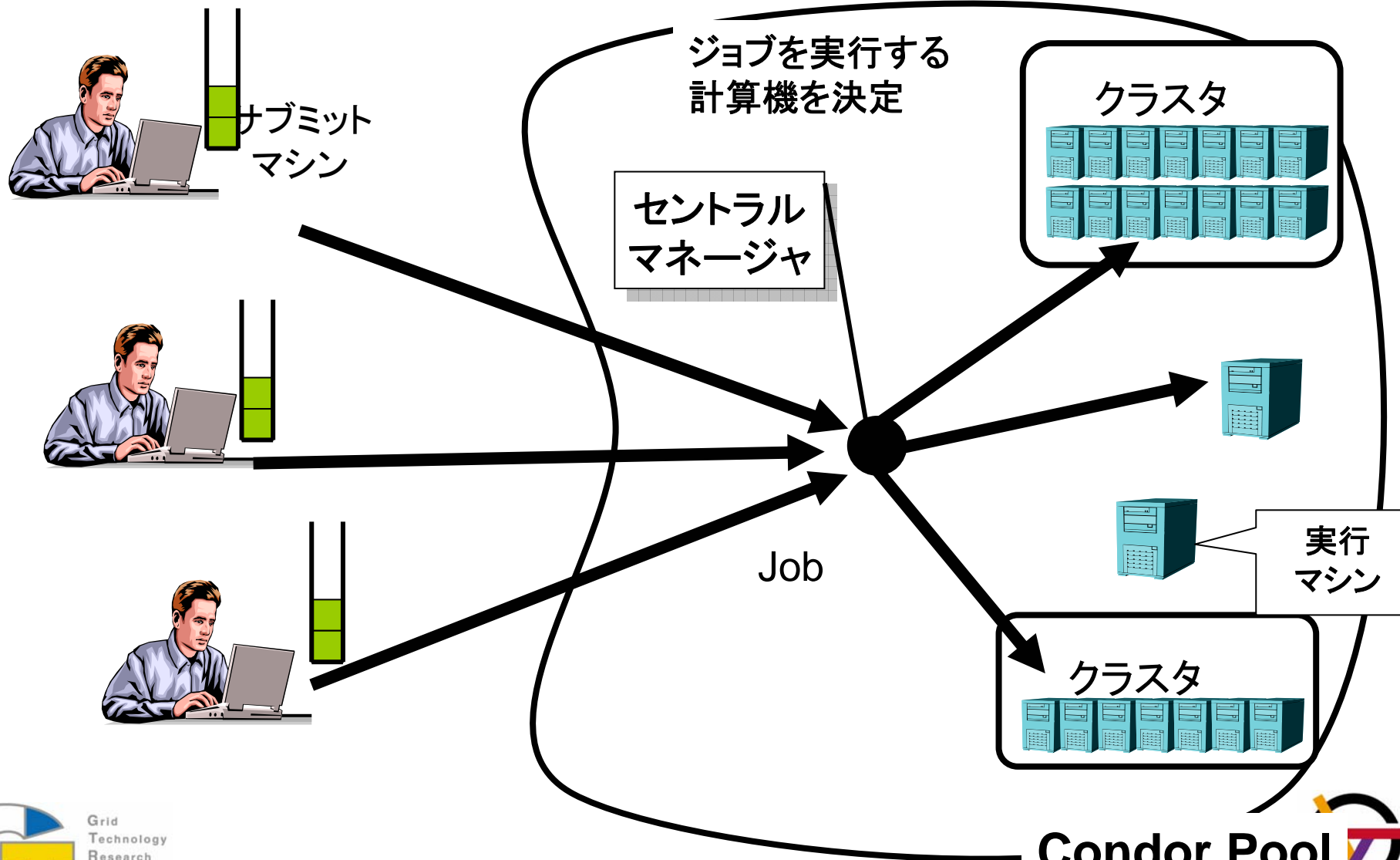
- Sun SPARC Sun4m, Sun4c, Sun UltraSPARC
  - ▶ Solaris 8, 9
- Silicon Graphics MIPS (R5000, R8000, R10000)
  - ▶ IRIX 6.5 (clipped)
- Intel x86
  - ▶ Red Hat Linux 7.1, 7.2, 7.3, 8.0, 9, Enterprise Linux 3, Debian Linux 3.1 (sarge)
  - ▶ Fedora Core 1, 2, 3
  - ▶ Windows 2000 Professional and Server, 2003 Server (Win NT 5.0), Windows XP Professional (Win NT 5.1) (clipped)
- ALPHA
  - ▶ Tru64 5.1 (clipped)
  - ▶ Red Hat Linux 7.1, 7.2, 7.3 (clipped)
- PowerPC
  - ▶ Macintosh OS X (clipped)
  - ▶ AIX 5.2 (clipped)
- Itanium (IA64)
  - ▶ Red Hat Linux 7.1, 7.2, 7.3 (clipped)
  - ▶ SuSE Linux Enterprise Server 8.1 (clipped)



[/share/apps/condor-tutorial](#)

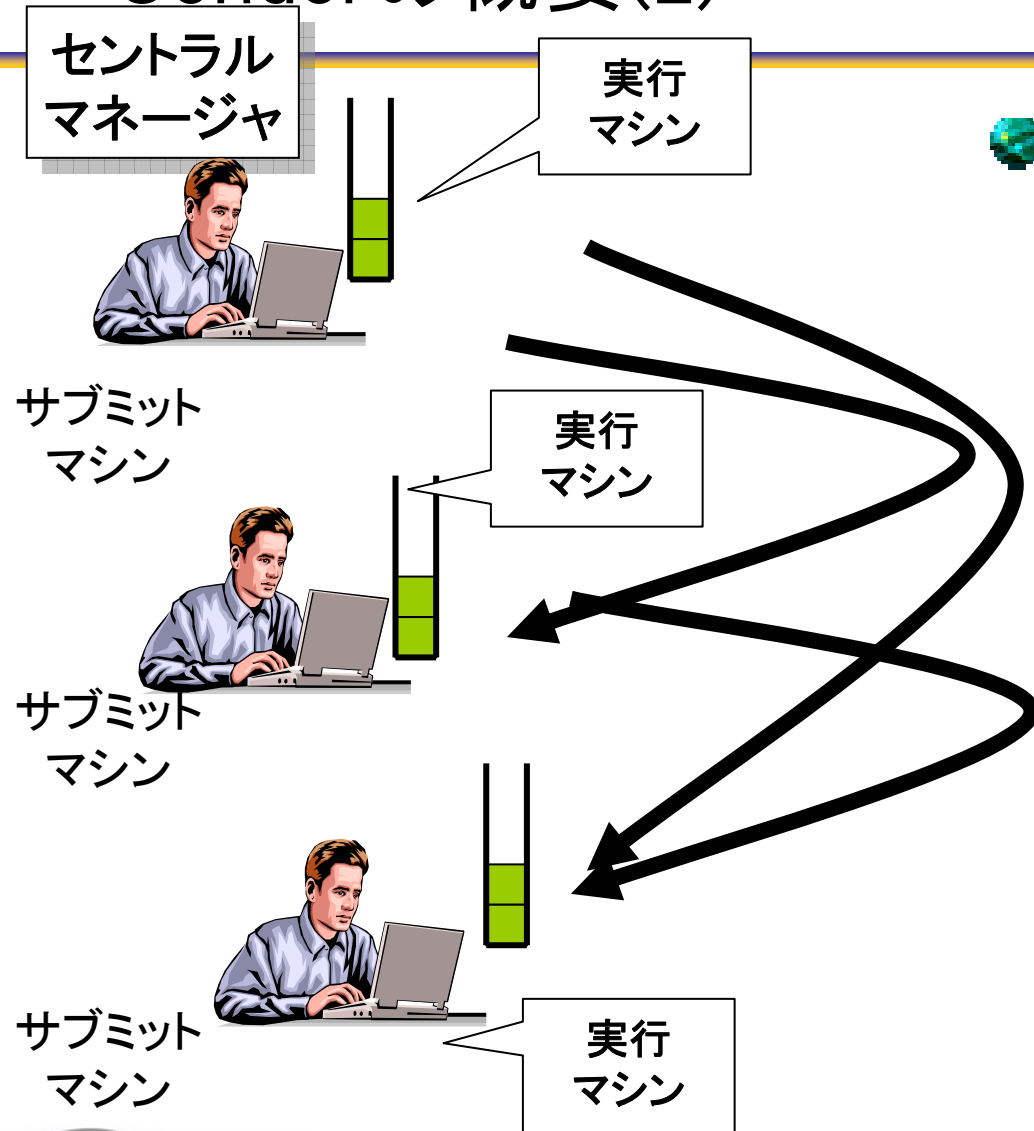


# Condorの概要



</share/apps/condor-tutorial>

## Condorの概要(2)



● 一つの計算機が複数の役割を担うことも可能

- ▶ サブミットマシン 兼 実行マシン
- ▶ サブミットマシン 兼 実行マシン 兼 セントラルマネージャ

# Condorの特徴

---

## ● スケーラブル

- ▶ 1プールで1000ノード程度まで稼動実績あり
- ▶ 複数のプールを組み合わせればさらに

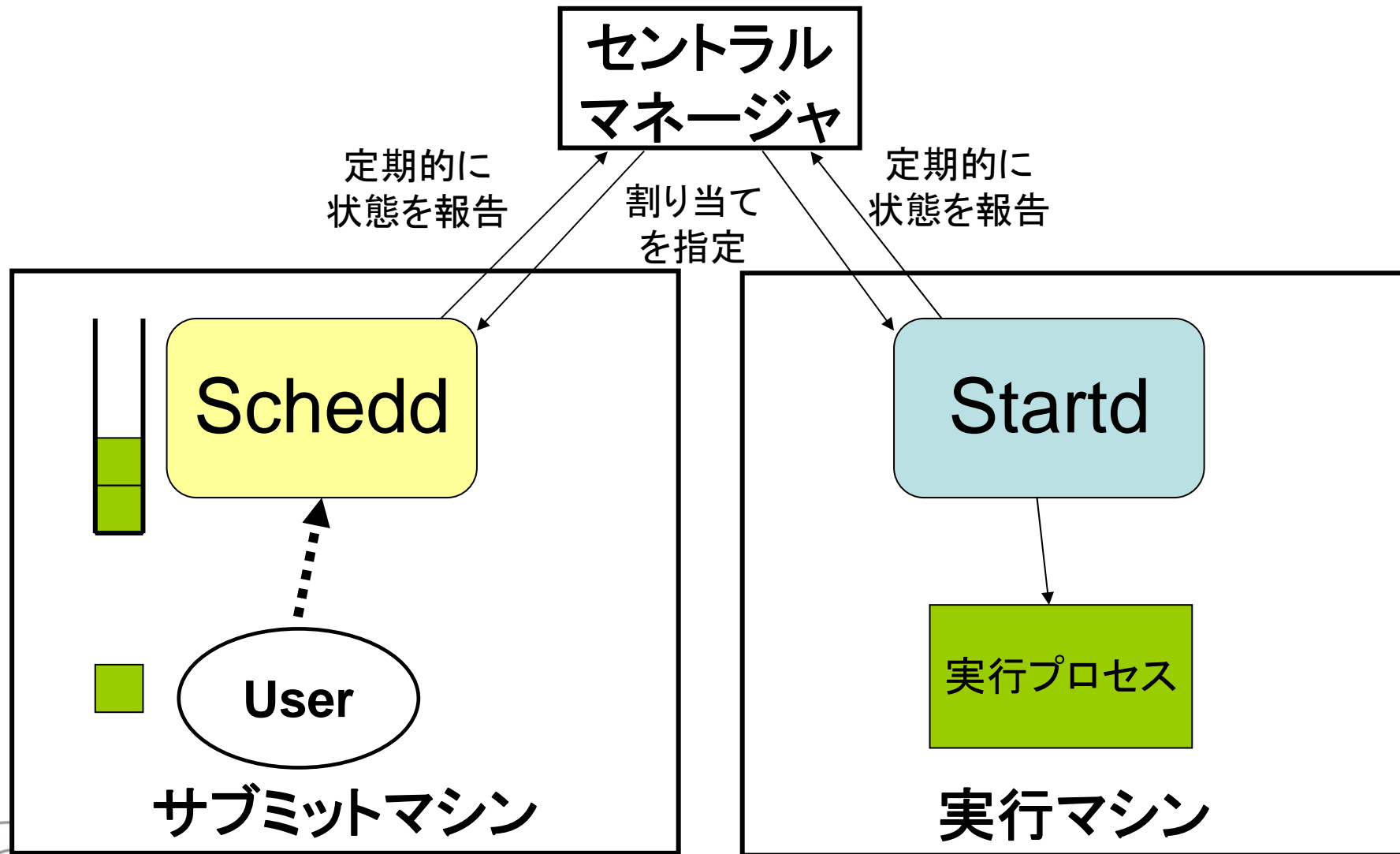
## ● スтейブル

- ▶ 自動復旧機構が組み込まれている

## ● フレキシブル

- ▶ ClassAdとマッチメイキングによる柔軟なスケジューリング
- ▶ ユーザのプライオリティによるフェアシェア

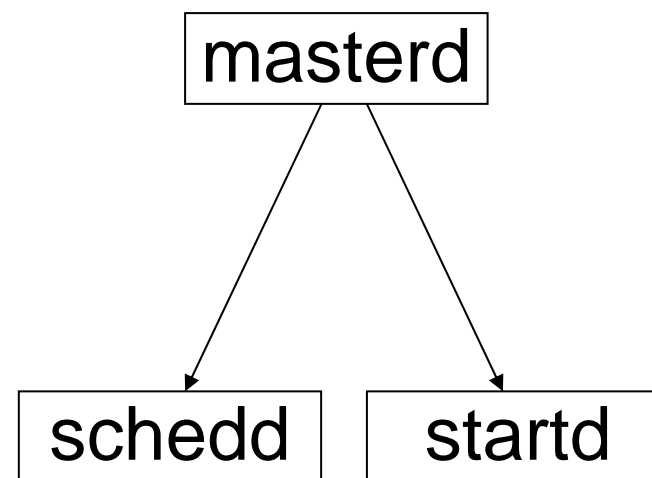
# Condorの構造



# スタビリティ

## ● マスターデーモン

- ▶ 他のデーモン群を起動・監視
- ▶ 落ちていたら再起動
- ▶ 単機能なのでこのデーモンが落ちることはほとんどない



# Condor ClassAd

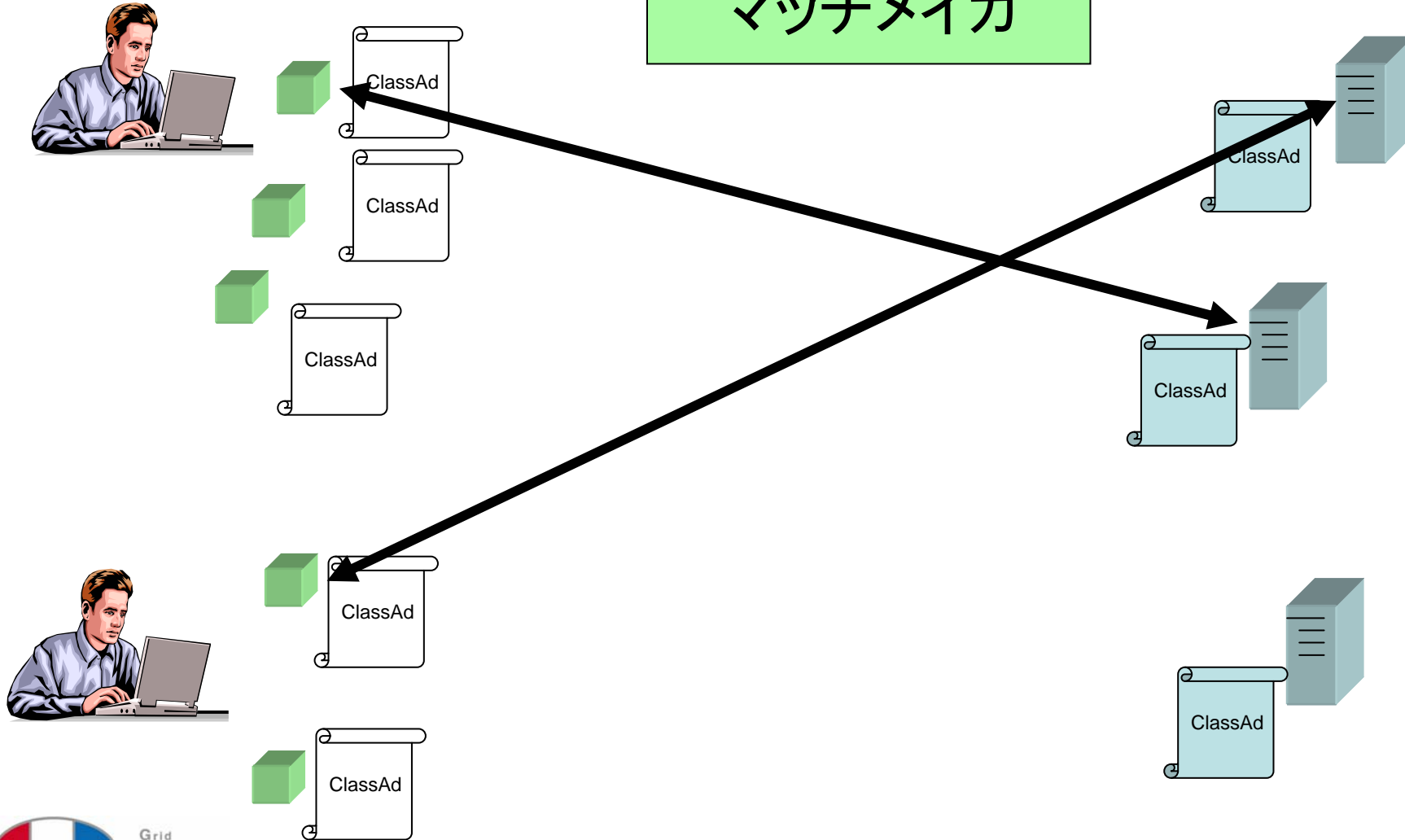
---

- Condorでやり取りする情報の表現形式
  - ▶ 汎用性、拡張性に富む
  - ▶ サブミットマシン、実行マシンの両方が利用
- 基本的には属性名と属性値の集合
  - ▶ 比較式を定義することも可能
- セントラルマネージャはサブミットマシン、実行マシンから得たClassAdを用いてマッチメイキングを行う

```
MyType = "Job"  
TargetType = "Machine"  
ClusterId = 292  
User = "nakada@a02.aist.go.jp"  
In = "/dev/null"  
Out = "A.out"  
Err = "/dev/null"
```

# マッチメイキング

マッチメイカ



</share/apps/condor-tutorial>



# 本日のハンズオン

- 個々のPC上にPersonal Condor Pool を構成
  - ▶ サブミット 兼 実行マシン 兼 セントラルマネージャ
- すべてのPCでひとつのCondor Pool を構成
  - ▶ 皆さんのPC - サブミット兼 実行マシン
  - ▶ 正面のPC - サブミット兼実行マシン 兼 セントラルマネージャ
- メニュー
  - ▶ 簡単なジョブのサブミット
  - ▶ Condor管理の基礎
  - ▶ Standard Universe
  - ▶ DAGManを用いたワークフロージョブの実行



セントラル  
マネージャ

サブミット兼  
実行マシン

サブミット兼  
実行マシン

サブミット兼  
実行マシン

サブミット兼  
実行マシン

サブミット兼  
実行マシン



[/share/apps/condor-tutorial](https://share/apps/condor-tutorial)

# 実習用PCの環境

---

● compute-0-X

▶ X は 0-20

● それぞれ griduserX を利用してください



</share/apps/condor-tutorial>



---

# インストール



</share/apps/condor-tutorial>



# インストールしていただくもの

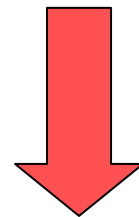
---

## ● Condor 6.8.5

- ▶ Condor 本体, Linux RHL4 用RPM

## ● テスト用パッケージ

- ▶ テストプログラム, サブミットファイル, バッチファイルなど



</share/apps/condor-tutorial>



# テストパッケージ

---

## helloWorld : 単純なジョブのテスト

- ▶ helloWorld.c : テストプログラム
- ▶ Makefile :
- ▶ helloWorld.sub : サブミットファイル

## chkpt\_test: スタンダードユニバーズのテスト

- ▶ chkpt\_test.c : テストプログラム
- ▶ chkpt\_test.sub : サブミットファイル
- ▶ Makefile :

## dagTest: Dagmanを用いたワークフロー テスト

- ▶ pi.dag : DAGファイル
- ▶ pi\_trial.sub, pi\_sumup.sub : サブミットファイル
- ▶ programs/{pi\_trial.c, pi\_sumup.c} : Cプログラム
- ▶ programs/concat.sh : 前処理用スクリプト
- ▶ programs/Makefile:



# パッケージのコピー

---

- > su - griduserX
- > cd
- > cp -r /share/apps/condor-tutorial/condor\_handson .

# RPM からのインストール

---

## 🌐 condor ユーザの作成 (済み)

## 🌐 以下 root で

▶ `cd /share/apps/condor-tutorial`

▶ `rpm -i condor-6.8.5-linux-x86-rhel3-dynamic-1. rpm`

▶ `/opt/condor-6.8.5` 以下にインストールされる

🌀 `--prefix` で変更することも可能.

## 🌐 設定ファイル

▶ マスタ

🌀 `/opt/condor-6.8.5/etc/condor_config`

▶ ローカル

🌀 `/opt/condor-6.8.5/local.compute-0-X/condor_config.local`



**`/share/apps/condor-tutorial`**



# 設定と起動

● export CONDOR\_CONFIG=/opt/condor-6.8.5/etc/condor\_config

● viなどで /opt/condor-6.8.5/etc/condor\_config を編集

▶ HOSTALLOW\_WRITE = YOU\_MUST\_CHANGE\_THIS\_INVALID\_CONDOR\_CONFIGURATION\_VALUE

を

▶ HOSTALLOW\_WRITE=\*

に変更

● 設定スクリプトを起動 (以下一行で)

▶ /opt/condor-6.8.5/condor\_configure  
--install-dir=/opt/condor-6.8.5  
--make-personal-condor

● 起動

▶ /opt/condor-6.8.5/sbin/condor\_master

● おまけ

▶ cp /share/apps/condor-tutorial/condor.sh /etc/profile.d/

▶ path, 環境変数を設定



[/share/apps/condor-tutorial](#)



# 動作確認

- master, collector, negotiator, schedd, startd の5つのデーモンが動作しているはず

```
[root@compute-0-0 /]# ps -ef | grep condor
condor    25327      1  0 17:09 ?           00:00:00 condor_master
condor    25328  25327  0 17:09 ?           00:00:00 condor_collector -f
condor    25329  25327  0 17:09 ?           00:00:00 condor_negotiator -f
condor    25330  25327  0 17:09 ?           00:00:00 condor_schedd -f
condor    25331  25327 16 17:09 ?           00:00:06 condor_startd -f
root      25356  20243  0 17:09 pts/0      00:00:00 grep condor
```

# 動作確認

## condor\_q

- ▶ queue 内のジョブの状態を表示

```
[griduser0@compute-0-0 condor-tutorial]$ condor_q
-- Submitter: compute-0-0.local : <10.255.255.254:34130> : compute-0-0.local
ID   OWNER      SUBMITTED  RUN_TIME ST PRI SIZE CMD
0 jobs; 0 idle, 0 running, 0 held
```

## condor\_status

- ▶ Condor プール内のノード状態を表示

```
[nakada@compute-0-0 condor-tutorial]$ condor_status
Name      OpSys  Arch  State  Activity  LoadAv  Mem  ActvtyTime
compute-0-0.l LINUX  INTEL Unclaimed Idle    0.000  502  0+00:07:27
          Total Owner Claimed Unclaimed Matched Preempting Backfill
          INTEL/LINUX  1  0  0  1  0  0  0

          Total  1  0  0  1  0  0  0
```

---

# Condorの使い方



[/share/apps/condor-tutorial](#)



# 最低限必要なコマンド群

---

## condor\_q

- ▶ ジョブキューの状態を表示
- ▶ 各ジョブの状態を詳しく見ることも可能

## condor\_status

- ▶ 参加マシンの状態を表示

## condor\_submit SUBMIT\_FILE\_NAME

- ▶ ジョブをサブミット

## condor\_rm JOB\_ID

- ▶ ジョブを削除

# helloWorld

---

## ● テストプログラムのビルド

> cd ~/condor\_handson/helloWorld

> make

## ● テストプログラムのテスト

> ./helloWorld 10

## ● サブミット

> condor\_submit helloWorld.sub

# helloWorld.c

---

```
#include <stdio.h>
#include <stdlib.h>
main(int argc, char ** argv){
    printf("Hello World!¥n");
    sleep(atoi(argv[1])); // sleeps specified sec
    exit(0);
}
```

# helloWorld.sub

```
# ユニバース指定 Windows 上では vanilla と java のみ可能
universe = vanilla

# 実行ファイル名指定
executable = helloWorld
arguments = 10

# 標準出力
output = helloWorld.$(Cluster).$(Process).out
# 標準エラー
error = helloWorld.$(Cluster).$(Process).err
# ログファイル
log = helloWorld.log

#ジョブのキューイング
queue 1
```



# ユーザログファイル

サブミットした  
マシン

実行しているマ  
シン

```
000 (030.000.000) 12/09 16:03:54 Job submitted from host:
<192.50.74.200:1432>
...
001 (030.000.000) 12/09 16:04:02 Job executing on host:
<192.50.74.175:1043>
...
006 (030.000.000) 12/09 16:04:10 Image size of job updated: 588
...
005 (030.000.000) 12/09 16:04:12 Job terminated.
(1) Normal termination (return value 0)
    Usr 0 00:00:00, Sys 0 00:00:00 - Run Remote Usage
    Usr 0 00:00:00, Sys 0 00:00:00 - Run Local Usage
    Usr 0 00:00:00, Sys 0 00:00:00 - Total Remote Usage
    Usr 0 00:00:00, Sys 0 00:00:00 - Total Local Usage
14 - Run Bytes Sent By Job
36864 - Run Bytes Received By Job
14 - Total Bytes Sent By Job
36864 - Total Bytes Received By Job
...
```

ジョブの終了  
コード



# ジョブが実行されている様子を見る

---

● ジョブはcondor\_exec.xxxxという名前で実行される

```
> ps -ef | grep condor
```

# 小技

---

- condor\_q をなんども実行するのは面倒.
- ひとつウィンドウを開いておいて下記を実行
  - > while /bin/true ; do condor\_q; sleep 2; clear; done

# やってみよう

---

🌐 queue 1 となっている部分を変更し, queue 10 としてサブミットしてみる

▶ condor\_q でたくさんサブミットされていることを確認

🌐 サブミットしすぎたジョブを condor\_rm で消してみる

▶ condor\_rm JOB\_ID

▶ condor\_rm USER\_NAME

👤 自分のジョブをすべて消去

# Condorの管理

---

## ● 設定ファイル

- ▶ `/opt/condor-6.8.5/etc/condor_config`
  - ⊗ マスター設定ファイル
  - ⊗ Unix環境ではこのファイルをすべてのノードでNFSで共有し、ノード固有情報をローカル設定ファイルに書くのが一般的
- ▶ `/opt/condor-6.8.5/local.compute-0-X/condor_config.local`
  - ⊗ ローカル設定ファイル
  - ⊗ マスター設定ファイルから参照される

# Condorの管理

---

- 設定ファイルを書き換えたら `condor_reconfig` で設定を読み込ませる必要がある
  - ▶ Condorサービス全体を再起動してもよい
  - ▶ 再起動した場合, そのノードで実行中のジョブがあれば一度実行が破棄されてしまうが, ジョブが消えてなくなることはない.

# やってみよう

## ● 設定ファイルを書き換えてサービスを再起動してみよう

▶ root に

▶ viで

`/opt/condor-6.8.5/local.compute-0-X/condor_config.local`  
を開く

▶ ファイルの最後に下記のように書いてみる.

`NUM_CPUS=2`

▶ `/opt/condor-6.8.5/sbin/condor_restart -startd`

## ● `condor_status` で確認

▶ ひとつだった計算機が2つになっているはず

▶ VMと呼ばれる概念. デュアルCPU・コアなどSMPを活用するためのもの

◎ いわゆる「仮想計算機」ではない

◎ 6.9 以降では slot と呼ばれる



Grid  
Technology  
Research

[/share/apps/condor-tutorial](#)



# ログファイル

---

● Condorでは各デーモンが個別のログファイルを持つ

▶ /opt/condor-6.8.5/local.compute-0-X/log

▶ MasterLog

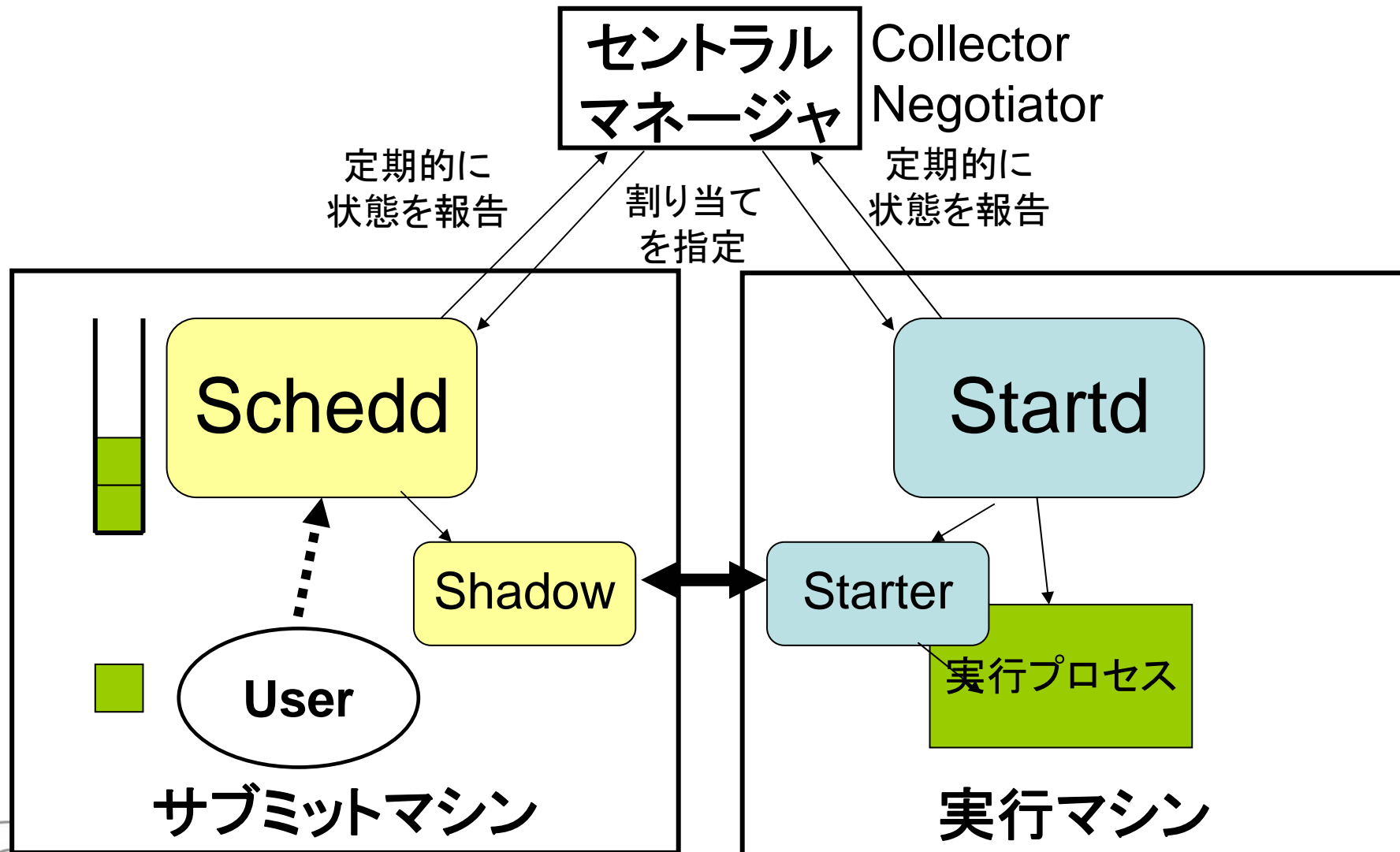
▶ SchedLog

▶ StartLog

▶ ShadowLog

▶ StarterLog

# Condorの構造 (詳しく)



# やってみよう

## StarterLog を見てみよう

```
12/9 16:04:02 *****
12/9 16:04:02 ** condor_starter (CONDOR_STARTER) STARTING UP
12/9 16:04:02 ** C:¥Condor¥bin¥condor_starter.exe
12/9 16:04:02 ** $CondorVersion: 6.7.13 Nov 7 2005 $
12/9 16:04:02 ** $CondorPlatform: INTEL-WINNT50 $
12/9 16:04:02 ** PID = 1308
12/9 16:04:02 *****
12/9 16:04:02 Using config file: C:¥Condor¥condor_config
12/9 16:04:02 Using local config files: C:¥Condor/condor_config.local
12/9 16:04:02 DaemonCore: Command Socket at <192.50.74.200:2723>
12/9 16:04:02 Setting resource limits not implemented!
12/9 16:04:02 Communicating with shadow <192.50.74.200:2721>
12/9 16:04:02 Submitting machine is "SHELL"
12/9 16:04:02 File transfer completed successfully.
12/9 16:04:03 Starting a VANILLA universe job with ID: 30.1
12/9 16:04:03 IWD: C:¥Condor/execute¥dir_1308
12/9 16:04:03 Output file: C:¥Condor/execute¥dir_1308¥helloWorld.30.1.out
12/9 16:04:03 Error file: C:¥Condor/execute¥dir_1308¥helloWorld.30.1.err
12/9 16:04:04 Renice expr "10" evaluated to 10
12/9 16:04:04 About to exec C:¥Condor¥execute¥dir_1308¥condor_exec.exe
12/9 16:04:04 Create_Process succeeded, pid=2020
12/9 16:04:14 Process exited, pid=2020, status=0
12/9 16:04:14 Got SIGQUIT. Performing fast shutdown.
12/9 16:04:14 ShutdownFast all jobs.
12/9 16:04:14 **** condor_starter (condor_STARTER) EXITING WITH STATUS 0
```

---

# 簡易ビューアの実行



</share/apps/condor-tutorial>



# python で書かれた 簡易ビューア

---

● condor\_q, condor\_status をラップしただけ.

● コピー, 実行

▶ ウィンドウを新しく開く

> cp -r /share/apps/condor-tutorial/pythonviewer ~

> cd ~/pythonviewer

> ./condor\_view.py    – CGI version

または

> ./startcherry      – CherryPy version

● 使用

> firefox localhost:8080



</share/apps/condor-tutorial>



---

# Standard Universe とチェックポイント



</share/apps/condor-tutorial>



# Standard Universe

---

## 🌐 チェックポイントをサポート

- ▶ 実行の途中で自由に停止, 再開ができる

- ▶ 特定のアーキテクチャ・OSでのみ可能

  - 🌐 Solaris と Linuxのみ

## 🌐 長期間実行するジョブには非常に有効

## 🌐 フェアシェアの実現にも寄与

- ▶ 「公平」な使用を実現するために実行中のジョブを停止しても計算が無駄にならない。

# チェックポイントの実現

---

## ● 特殊なライブラリをリンク

- ▶ シグナルをフックして、特定のシグナルを受け取るとメモリ領域をダンプする
- ▶ コンパイルドライバcondor\_compileが提供される
- ▶ ディスクI/Oをフックし、サブミットマシン上のデーモン(shadow)で処理。つまり、サブミットマシン上のファイルを直接読み書きできる。

## ● 制約

- ▶ ネットワーク接続不可
- ▶ プロセスフォーク不可
- ▶ スレッド不可

# テストプログラム

```
#include <stdio.h>
#include <stdlib.h>

main(){
    int i;
    FILE * fp;
    fp = fopen("/tmp/chkpt_test.tmp", "w");
    for (i = 0; i < 100; i++){
        fprintf(fp, "%03d¥n", i);
        fflush(fp);
        sleep(1);
    }
    exit(0);
}
```

100回なにか  
出力して1秒休み

全体で100秒



# サブミットファイル

```
universe = standard
executable = chkpt_test
output = chkpt_test.$(Cluster).$(Process).out
error = chkpt_test.$(Cluster).$(Process).err
log = chkpt_test.log
queue 1
```

# サンプルのコンパイルと実行

---

 > `cd ~/condor_handson/chkpt`

 > `make`

 > `ls -al`

▶ サイズが異常に大きいことがわかる.

▶ Condorのライブラリがスタティックにリンクされているため.

 > `condor_submit chkpt_test.sub`

## 途中で実行を止めてみる

- condor\_q で job が R(running)になるのを確認
- condor\_vacate を実行
  - ▶ 実行したマシンからジョブを追い出す
  - ▶ condor\_q で I (idle) になることを確認
  - ▶ log ファイルで停止を確認
  - ▶ /tmp/chkpt\_test.tmp を見てみる.
- condor\_reschedule を実行
  - ▶ ほうっておいても自動的に再実行されるが時間がかかるので、強制的に再度割り当てを行う.
- 実行が終わったら log ファイルで実行時間を確認
  - ▶ 2度の実行時間の和が100秒になっているはず



---

すべてのPCでひとつのプールを作成



</share/apps/condor-tutorial>



# 前においてあるPC (gtrain.local) をセントラルに

- 以下 root で
- Condor を停止
  - ▶ `> condor_off -master`
- 再コンフィグ (以下一行で)
  - ▶ `/opt/condor-6.8.5/condor_configure`  
`--type=execute,submit`  
`--central-manager=gtrain.local`  
`--install-dir=/opt/condor-6.8.5/`
- 下記のように `/opt/condor-6.8.5/local.compute-0-X/condor_config.local` を修正

```
## When is this machine willing to start a job?
```

```
START = True
```

```
## When to suspend a job?
```

```
SUSPEND = False
```

- 再起動



```
▶ /opt/condor-6.8.5/sbin/condor_master
```

</share/apps/condor-tutorial>



# 動作確認

- master, schedd, startd の3つのデーモンが動作しているはず

```
[nakada@compute-0-0 dagtest]$ ps -ef | grep condor
condor    21050      1  0 09:45 ?        00:00:04 /opt/condor-6.8.5/sbin/condor_master
condor    21051 21050  0 09:45 ?        00:00:09 condor_schedd -f
condor    21052 21050  0 09:45 ?        00:00:12 condor_startd -f
nakada    24247 21535  0 15:11 pts/0    00:00:00 grep condor
```

# 確認

---

- condor\_status でみんなのPCが見えるか？
- condor\_q -global でみんなのqueue が見えるか？
  
- ローカルホストで実行している簡易ビューアを停止
- gtrain.local 上の ビューアを利用してください
  - ▶ 個別のビューアスクリプトがcondor\_q を行うと負荷がかかるため
  - ▶ > firefox gtrain.local:8080



# やってみよう

---

- Hello World をqueue 10 程度で投入し, どこで実行されるかを condor\_q, condor\_status で監視してみよう.

---

# DAGman を用いたワークフロー実行

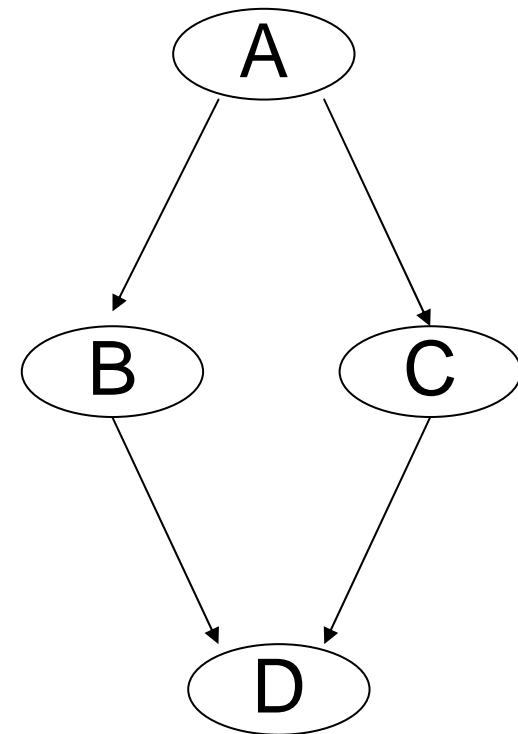


</share/apps/condor-tutorial>



# Condor DAGMan

- 複数の依存関係を持つジョブ群 (ワークフロー) を依存関係の順番に実行
- 各ジョブごとにCondorのサブミッションファイルを記述
- ジョブ間の依存関係を簡単なテキストで記述

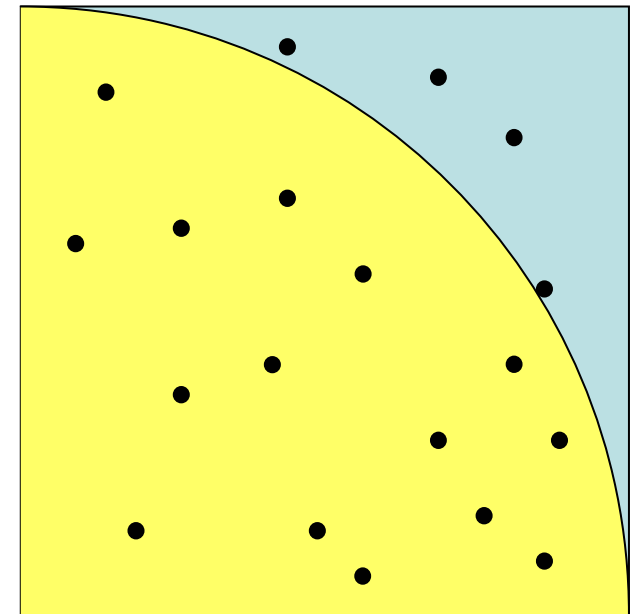


```
JOB A A.condor  
JOB B B.condor  
JOB C C.condor  
JOB D D.condor
```

```
PARENT A CHILD B C  
PARENT B C CHILD D
```

# サンプルプログラム

- ランダムに点をうって $\pi$ を求める
  - ▶ 正方形の中に大量に点を打つ
  - ▶ 4分円の中の点の数を数える
  - ▶ この確率から $\pi$ を求める



$$\begin{aligned} \text{PI} &\approx 4 * \frac{\text{no. points in the quadrant}}{\text{no. of whole points}} \\ &= 4 * \frac{15}{19} \end{aligned}$$

$$= 3.1579\dots$$



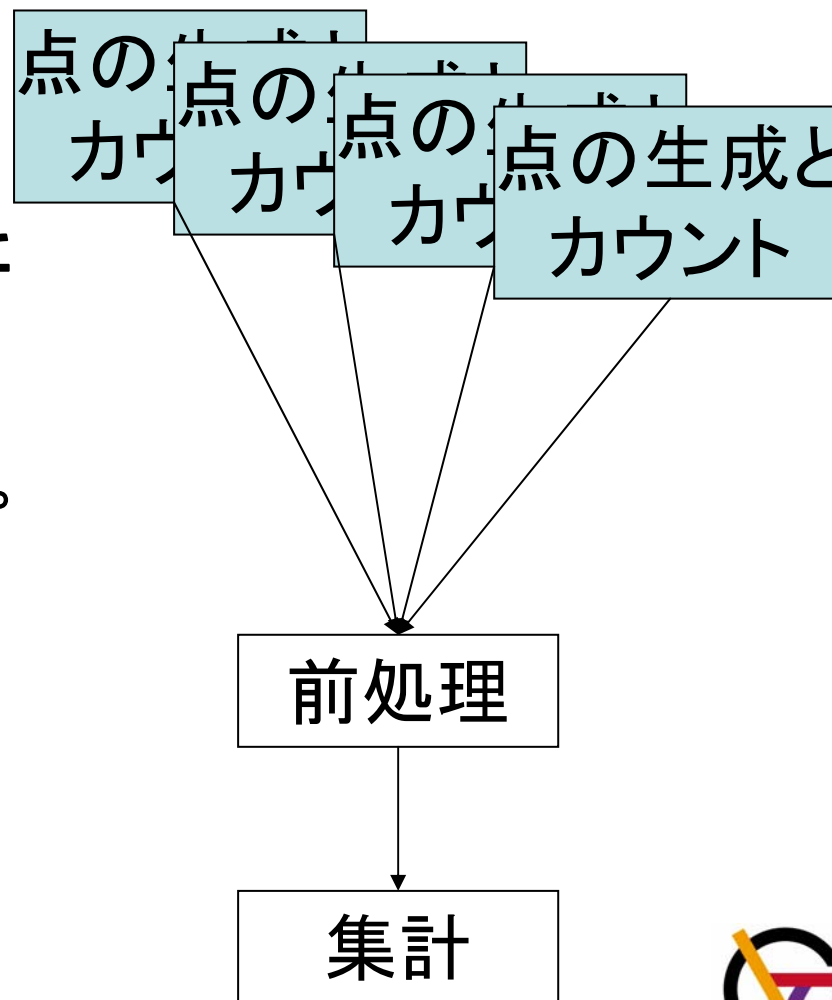
Grid  
Technology  
Research

</share/apps/condor-tutorial>



# DAGman による実装

- 点を打つ部分は乱数のシードを変更すれば並列に実行することができる
- 点を生成し、四分円に落ちたかどうかを判定するプログラムを複数実行
- 結果のファイルを積算するプログラムを別ジョブで実装



# 実装の詳細

---

## pi\_trial SEED NUM\_POINTS

- ▶ SEEDを種にした乱数を用いてNUM\_POINTS 個の点を生成し, 判定
- ▶ 出力: NUM\_POINTS NUM\_IN

## concat.sh

- ▶ 複数のpi\_trial から出力されたファイルをマージして一つのファイル concatenated に

## pi\_sumup

- ▶ 入力ファイルの内容を積算して,  $\pi$  を導出

# pi.dag

```
JOB A pi_trial.sub  
JOB B pi_trial.sub  
JOB C pi_trial.sub  
JOB D pi_trial.sub  
JOB E pi_trial.sub
```

```
JOB Z pi_sumup.sub
```

```
VARs A seed="0"  
VARs B seed="1"  
VARs C seed="2"  
VARs D seed="3"  
VARs E seed="4"
```

```
SCRIPT pre Z programs/concat.sh  
PARENT A B C D E CHILD Z
```

実行前  
スクリプト



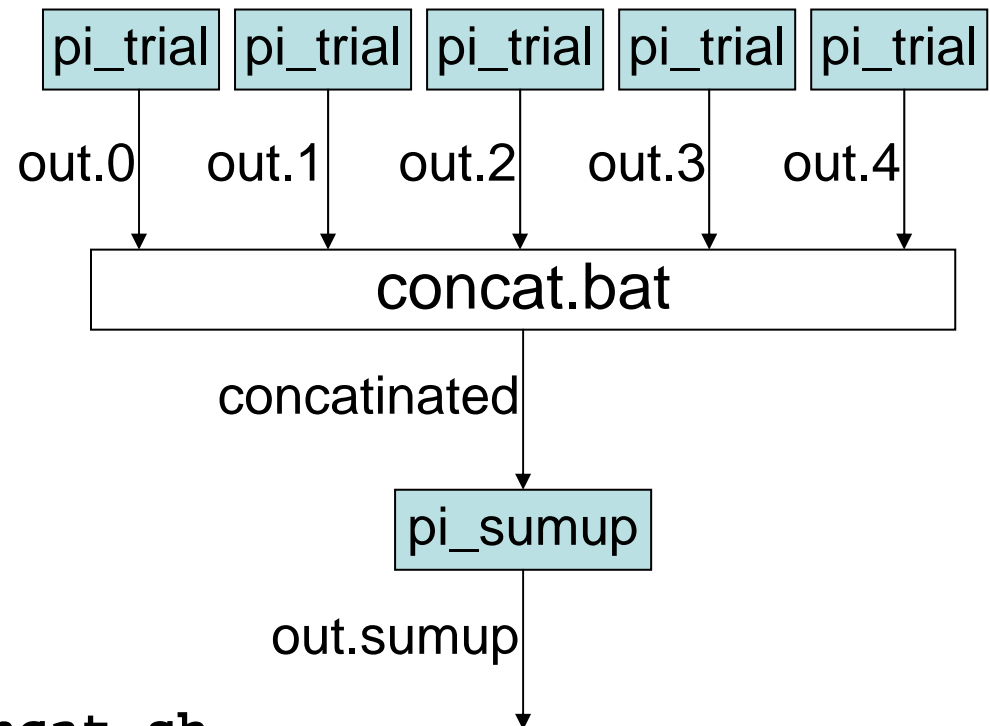
# pi.dag

```
JOB A pi_trial.sub  
JOB B pi_trial.sub  
JOB C pi_trial.sub  
JOB D pi_trial.sub  
JOB E pi_trial.sub
```

```
JOB Z pi_sumup.sub
```

```
VARs A seed="0"  
VARs B seed="1"  
VARs C seed="2"  
VARs D seed="3"  
VARs E seed="4"
```

```
SCRIPT pre Z programs/concat.sh  
PARENT A B C D E CHILD Z
```



# pi\_trial.sub

VARsで  
置換される

```
universe = vanilla
executable = programs/pi_trial
arguments = $(seed) 10000000
output = pi_out.$(seed)
error = pi_err.$(seed)
should_transfer_files = true
when_to_transfer_output = ON_EXIT_OR_EVICT
log = pi.log

queue
```

# pi\_trial.c

```
#include <stdio.h>
#include <stdlib.h>
#define MAX_LEN 256

main(int argc, char ** argv){
    int seed, i;
    long times, inside = 0;

    if (argc < 3){
        fprintf(stderr, "USAGE: %s SEED TIMES\n", argv[0]);
        exit(2);
    }
    seed = atoi(argv[1]);
    times = atol(argv[2]);
    srand(seed);
    for (i = 0; i < times; i++){
        double x, y;
        x = (double)(rand()) / RAND_MAX;
        y = (double)(rand()) / RAND_MAX ;

        if (x*x + y*y <= 1.0)
            inside++;
    }
    printf("%ld %ld\n", times, inside);
    exit(0);
}
```



# concat.sh

---

```
cat out.* > concatenated
```



</share/apps/condor-tutorial>



# pi\_sumup.sub

---

```
universe = vanilla
executable = programs/pi_sumup
input = concatenated
output = out.sumup
error = err.sumup
log = pi.log
queue
```



# pi\_sumup.

```
#include <stdio.h>
#include <stdlib.h>
#define MAX_LEN 256

main(){
    char buf0[MAX_LEN], buf1[MAX_LEN];
    long times = 0;
    long sum = 0;

    while (scanf("%s %s", &buf0, &buf1) == 2){
        times += atol(buf0);
        sum += atol(buf1);
    }
    printf("%lf\n", (((double)sum) / times) * 4);
    exit(0);
}
```



# DAGman job のサブミット

---

● ~/condor\_handson/dagtest を /tmp にコピー

▶ cp -r ~/condor\_handson/dagtest /tmp

▶ cd /tmp/dagtest

● プログラムをコンパイル

▶ cd programs; make

● > condor\_submit\_dag pi.dag

● > condor\_q で状況を確認

● すべて終了したら結果を確認

▶ > cat out.\*

▶ > cat concatenated

▶ > cat out.sumup



# やってみよう

---

● pi.dag を変更して 計算するジョブを増やしてみよう

▶ ジョブA-E に加えて 同様にF,G を作成

▶ 最後の行の依存関係のところにも, F, Gを追加

● 注意:

▶ いろいろと生成されたファイルが残っていると再実行できないので, 一度 `make clean` でクリアしてください

# おわりに

---

- Condorを用いるとキャンパスグリッドが簡単に構築できる
  - ▶ Mac OS X, Windows も
  - ▶ 混在も可能(バイナリを複数用意する)
- ワークフロー管理は便利
- より高度な利用法
  - ▶ Globus等の広域グリッドとの融合も可能
  - ▶ Ninf-G のバイナリをCondorでスケジュールすることも可能
  - ▶ MasterWorker フレームワークを使用したパラメータサーベイ

# 参考リンク

---

 <http://www.cs.wisc.edu/condor/>

▶ダウンロードもここから

 <http://www.cs.wisc.edu/condor/manual/v6.8/>

▶ドキュメンテーション