# *Trick Simulation Environment:*
# *Monte Carlo*

***Donna Panter(L-3Com/ER7)***

# Agenda/Schedule

- **Topics**
  1. **Monte Carlo Overview**
  2. **Input File Requirements (using spring dampening example)**
  3. **Monte Carlo Execution**
  4. **Monte Carlo Slaves**
  5. **Monte Carlo Jobs**
  6. **Monte Carlo Example (land the cannon ball in a target)**
  7. **Final notes**

# *Overview*

- **What is Monte Carlo?**
  - **A technique to solve mathematical problems by using random numbers and probability statistics.**
    - **For Trick – Run the simulation repeatedly varying values of user-chosen variables**

# *Overview*

- First, we look at a spring mass damper system simulation (`SIM_spring`, which has now been copied as `SIM_spring_mc`) and allow Trick to perform Monte Carlo for two specific examples
  - Hard-coded input
  - Distribution formula to generate input

- Second, we will look at how to use Monte Carlo jobs.
  - In Chapter 11 of the Trick Tutorial, it was shown how to use Trick to vary jet firing sequences for the cannon jet control problem, both using ‘hard-coded’ inline data and Gaussian randomly generated data.
  - We will modify the simulation to determine the jet firing sequence to hit a target.

# *Monte Carlo Input Variables*

- **The following classes are used to specify which input variables are available for changing from run to run.**
  - **MonteVarFile**
    - Pulls values from an input file.
  - **MonteVarRandom**
    - Auto-generate the input values using a distribution formula
      - Gaussian
      - Poisson
      - Flat
  - **MonteVarFixed**
    - Specifies a constant value
  - **MonteVarCalculated**
    - Calculates the values in user-created jobs.

# *Monte Carlo Input Variables (inline)*

## Go to the following directory

```
% cd $HOME/trick_sims/SIM_spring_mc/RUN_test.inline
```

## Open the input.py file

```
% [vi|nedit|kate] input.py
```

```
var0 = trick.MonteVarFile(“smd.spring.input.damping”, “M_spring_inline”, 1)

trick_sys.sched.add_variable(var0)
```

# *Monte Carlo Input Variable (inline)*

**Let’s view the input file**

```
% cd ..
% [vi|nedit|kate] M_spring_inline
```

```
0.0000        3      3.4
2.0000        4      3.5
4.0000        5      3.6
8.0000        6      3.7
16.0000       7      3.8
32.0000       8      3.9
64.0000       9      4.0
128.0000      10     4.1
256.0000      11     4.2
512.0000      12     4.3
```

# Example 2 – Varying M, K, C (Gaussian)

## Now let's view the gaussian input file

```
% [vi|nedit|kate] RUN_test.gauss/input.py
```

```
var2 = trick.MonteVarRandom("smd.spring.input.damping", trick.MonteVarRandom.GAUSSIAN)
var2.set_seed(3)
var2.set_sigma(0.6862)
var2.set_mu(8.0)
var2.set_min(-4.0)
var2.set_min_is_relative(1)
var2.set_max(48.0)
var2.set_max_is_relative(1)
trick_sys.sched.add_variable(var2)
```

- **Here we use syntax to set up a Gaussian distribution of mass, stiffness, and damping (notice seed (initializes random number generator), sigma (std dev), mu (mean), rel_min and rel_max)**

- **For this example, Trick randomly generates the run data through an interface to the GNU Scientific Library (`trick_gsl_rand.c`)**

# *Monte Carlo Execution*

- **To execute either of these examples, two variables must be set in the input file:**
  - `trick.mc_set_enabled(1)`
  - `trick.mc_set_num_runs(50)`

- **CP the simulation**

```
% CP
```

- **Run the sim for the first example:**

```
% S_main_* RUN_monte.inline/input.py
```

- **Notice the new `RUN_MONTE_monte.inline` directory which contains the output data (can visualize multiple curves through trick_dp)**

# *Monte Carlo Execution*

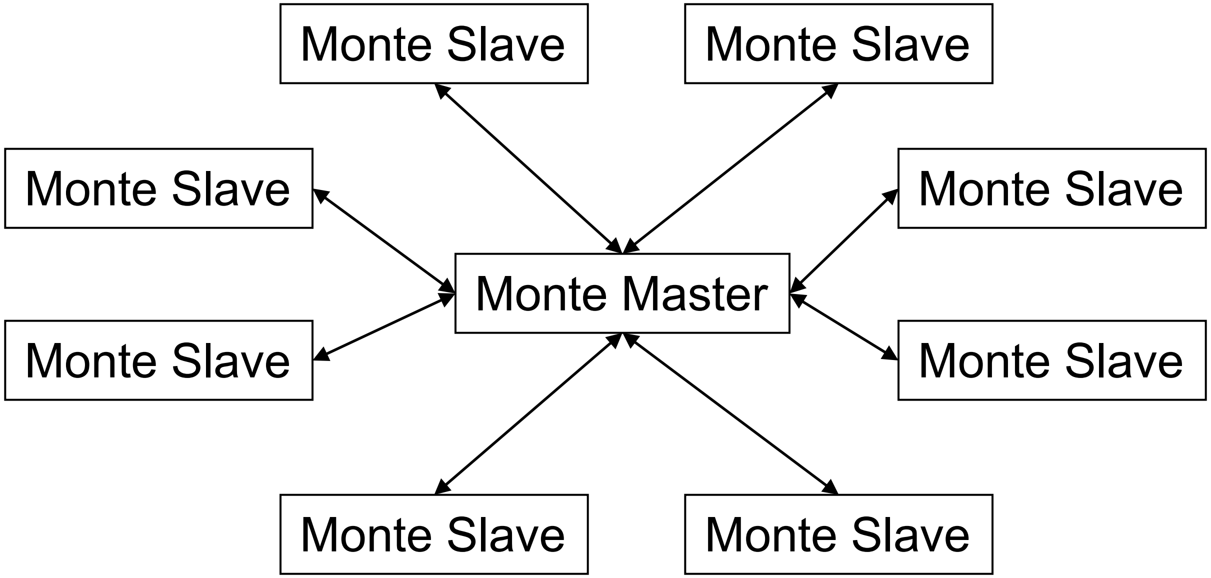
- **Now run the sim for the second example:**

```
% S_main_* RUN_monte.gauss/input
```

- **Notice the new `RUN_MONTE_monte.gauss` directory which contains the output data (again, multiple curves can then be visualized through trick_dp)**

# *Monte Carlo Slaves*

- **Previous examples used only a single worker**
- **Trick's Monte Carlo capability optimized for multiple workers**



# *Monte Carlo Slaves*

- **To add slaves,**
  - **Unlimited number of slaves can be specified**

```
slave0 = trick.MonteSlave(“localhost”)
trick_sys.sched.add_slave(slave0)
slave1 = trick.MonteSlave(“WonderWoman”)
trick_sys.sched.add_slave(slave1)
slave2 = trick.MonteSlave(“CatWoman”)
trick_sys.sched.add_slave(slave2)
```

- **Trick will automatically start each slave simulation with ssh**

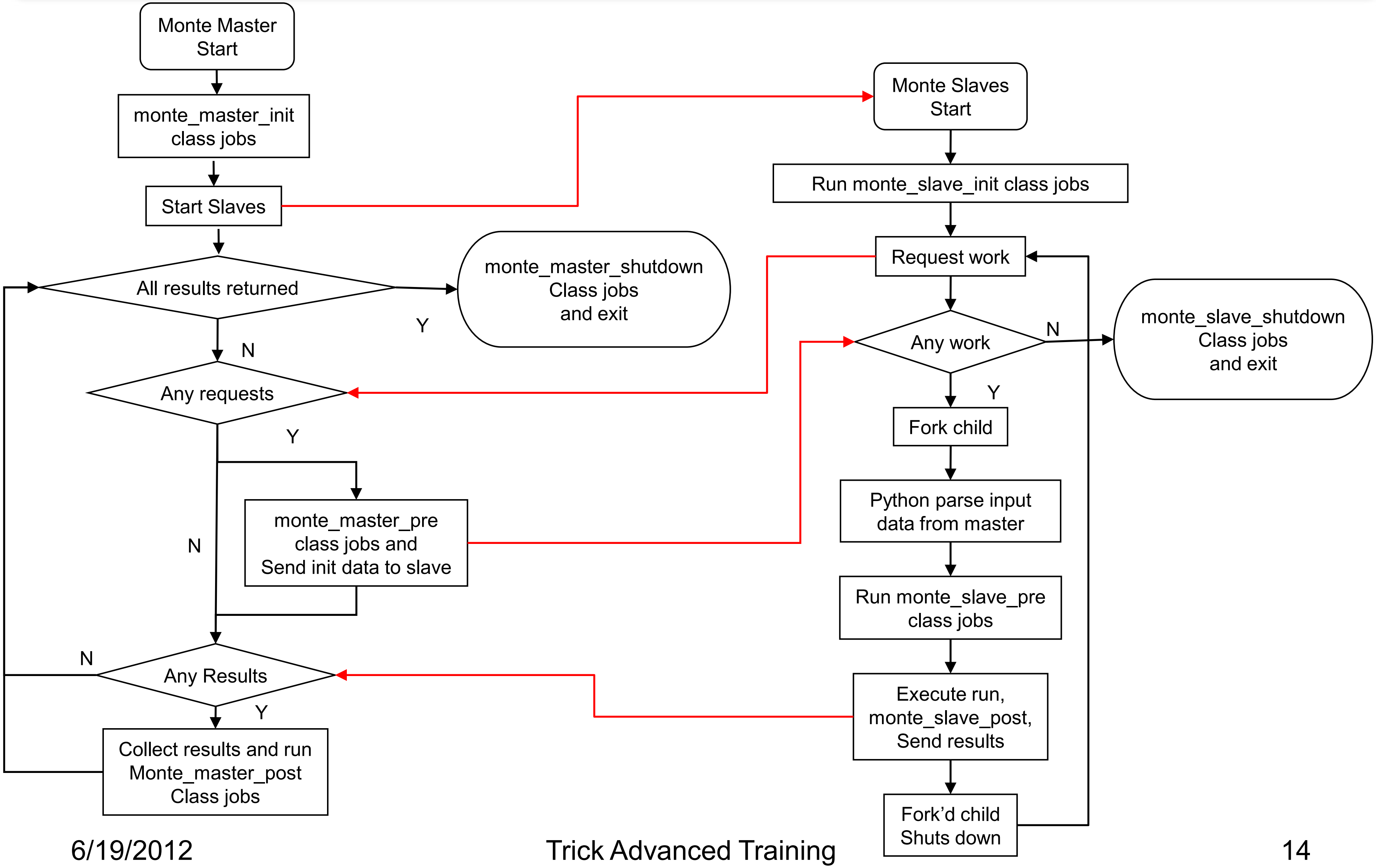
- **Slaves ask the master for work when they are ready for work**
  - **Faster slave machines will do more work**

- **You can start multiple slaves on the same machine**
  - **Useful for machines with multiple processors**

# *Job Classes*

- **Monte Carlo specific job classes to handle master/slave interations**
  - **Monte_Master_Init**
    - Runs when master sim is initialized
  - **Monte_Master_Pre**
    - Runs before new data is dispatched to slave sim
    - Useful for calculating/optimizing next run values if desired
  - **Monte_Master_Post**
    - Runs after result is returned from slave
    - Useful for calculating statistics for returning results
  - **Monte_Master_Shutdown**
    - Runs when master shuts down
  - **Monte_Slave_Init**
    - Runs when slave sim is initialized
  - **Monte_Slave_Pre**
    - Runs after new data is received from master
  - **Monte_Slave_Post**
    - Runs after slave sim is completed (sends result to master)
  - **Monte_Slave_Shutdown**
    - Runs when monte carlo master comm is lost and slave shuts down

# Monte Carlo Master/Slave Interaction



Monte Master Start
monte_master_init class jobs
Start Slaves
All results returned
Y
monte_master_shutdown Class jobs and exit
N
Any requests
Y
monte_master_pre class jobs and Send init data to slave
N
N
Any Results
Y
Collect results and run Monte_master_post Class jobs
Monte Slaves Start
Run monte_slave_init class jobs
Request work
Any work
N
monte_slave_shutdown Class jobs and exit
Y
Fork child
Python parse input data from master
Run monte_slave_pre class jobs
Execute run, monte_slave_post, Send results
Fork'd child Shuts down

# *Monte Slaves*

- The master sets a timeout value
  - Default timeout is 120 seconds
  - User may change the value in the input file with the following function: trick.mc_set_timeout(double)
- Each slave must return a result within its individually timed timeout period
  - If no result is returned, the slave is assumed dead and the run's initial data is re-dispatched to the next available slave
  - Slaves can be "killed" and no results will be lost

# Cannon Ball in Target Example

- This is the cannonball simulation example used in the tutorial with to demonstrate Monte Carlo.
- Create a monte_master_post job and a monte_slave_post job.
  - The monte_master_post job will read the CANNON struct information from the slave. Check if the cannon landed in the target area. Shutdown if it did, otherwise continue.
  - The monte_slave_post job will write the CANNON struct information to the master.

# Cannon Ball in Target Example

```
% cd $HOME/trick_models/cannon
% mkdir -p monte/src
% mkdir -p monte/include
% cd monte/src
%[vi|nedit|kate] cannon_slave_post.c
```

```
/************************* TRICK HEADER *******************
PURPOSE:                    (Kaboom!!!)
***********************************************************/
#include "cannon/aero/include/cannon_aero.h"
#include "sim_services/MonteCarlo/include/montecarlo_c_intf.h"

int cannon_slave_post(CANNON_AERO* C)
{
    mc_write((char*) C, sizeof(CANNON_AERO) );

    return(0) ;
}
```

# Cannon Ball in Target Example

```
%[vi|nedit|kate] cannon_master_post.c
```

```
/************************** TRICK HEADER ********************
PURPOSE:                     (Kaboom!!!)
*************************************************************/
#include "cannon/aero/include/cannon_aero.h"
#include "sim_services/MonteCarlo/include/montecarlo_c_intf.h“
int cannon_master_post()
{
    CANNON_AERO C_curr ;
    mc_read((char*) &C_curr, sizeof(CANNON_AERO) ) ;
    if ((C_curr.pos[0] > 152) & (C_curr.pos[0] < 153)) {
        exec_terminate(“cannon_master_post”,
                       “Cannon landed in the target!”);
    }
    return(0) ;
}
```

# Cannon Ball in Target Example

```
% cd ../include
% [vi|nedit|kate] cannon_monte_proto.h
```

```
/************************** TRICK HEADER ********************
PURPOSE:                     (Kaboom!!!)
*************************************************************/
#ifndef _cannon_monte_proto_h_
#define _cannon_monte_proto_h_
#include "cannon/aero/include/cannon_aero.h"
#ifdef __cplusplus
extern "C" {
#endif
int cannon_master_post();
int cannon_slave_post(CANNON_AERO*);
#ifdef __cplusplus
}
#endif
#endif
```

# Modify S_define

```
% cd $HOME/trick_sims/SIM_monte
% [vi|nedit|kate] S_define
```

- Add the two new jobs to LIBRARY DEPENDENCIES

  ```
  (cannon/monte/src/cannon_master_post.c)
  (cannon/monte/src/cannon_slave_post.c)
  ```

- Add the new prototype header file at the end of the ##include list

  ```
  ##include “cannon/monte/include/cannon_monte_protot.h”
  ```

# *Modify S_define*

```
    .
    .
class MonteSimObject : public Trick::SimObject {
    public:
        CANNON_AERO *cannon_ptr;
        MonteSimObject() {
            ("monte_master_post") cannon_master_post();
            ("monte_slave_post")  cannon_slave_post(cannon_ptr);
        }
};

MonteSimObject optimizer;

void create_connections() {
    optimizer.cannon_ptr = &dyn.baseball;
}
```

# CP and Run Simulation

## Compile and Execute the simulation

```
% CP
% S_*exe RUN_test.gauss/input.py
```

```
     .
     .
|L 1|2011/08/08,10:09:09|WonderWoman| |T 0|0.00| Monte [Master] Receiving results for run 8 from WonderWoman:1.
|L 1|2011/08/08,10:09:09|WonderWoman| |T 0|0.00| Monte [Master] Dispatching run 9 to WonderWoman:1.
|L 1|2011/08/08,10:09:09|WonderWoman| |T 0|0.00| SIMULATION TERMINATED IN
|L 1|2011/08/08,10:09:09|WonderWoman| |T 0|0.00|    PROCESS: 0
|L 1|2011/08/08,10:09:09|WonderWoman| |T 0|0.00|    ROUTINE: cannon_master_post
|L 1|2011/08/08,10:09:09|WonderWoman| |T 0|0.00| DIAGNOSTIC: Cannon landed in the target

|L 1|2011/08/08,10:09:09|WonderWoman| |T 0|0.00| Monte [WonderWoman:1] : Shutdown command received from Master.
Shutting down.
```

# *Monte Carlo Notes*

- A dry run flag is available: `trick.mc_set_dry_run(int)`
  - Useful for generating random distributions without actually doing the runs
  - See monte_runs file in the MONTE_<run _directory> directory
- It is also possible to run a subset of runs by using
  - `trick.mc_add_range(<run num>)`
  - `trick.mc_add_range(<first run num>, <last run num>)`
- All data recording for all runs is saved.
  - Large data sets can generate enormous amounts of data.
  - Take care on what to data record

# Monte Carlo Notes

- A monte_input file is created in each RUN_* directory
  - Allows a user to execute a single monte carlo run by simply including the file in the input.py file.
- Almost too easy to add slaves
  - Tendency to add machines which seem unused
  - Monte Carlo slaves tend to use 99.9% of CPU
  - Don’t use too many machines in your lab!