SCALING AND OPTIMIZING STOCHASTIC TUPLE-SPACE COMMUNICATION IN THE DISTRIBUTIVE INTEROPERABLE EXECUTIVE LIBRARY

Zaire Ali (Morehouse College)

Jason Coan (Maryville College)

Mentors: Kwai Wong and David White

LOOSELY COUPLED SYSTEMS

- Modular program design
- The problem is broken down into sub-problems that are computed independently
- Interaction between modules occurs along a set of specifically designated shared boundary points
- Reduces the complexity of the system, and makes it easier to debug
- Can be solved efficiently on a parallel computer
- Results in code that is highly reusable
- Generally a good idea for program design

THE DISTRIBUTIVE INTEROPERABLE EXECUTIVE LIBRARY (DIEL)

- Designed to make it easier to build loosely coupled systems for high-performance computers

- A lightweight integrator of modules, managing distribution of data and coordinating communication among processes

STRUCTURE OF THE DIEL

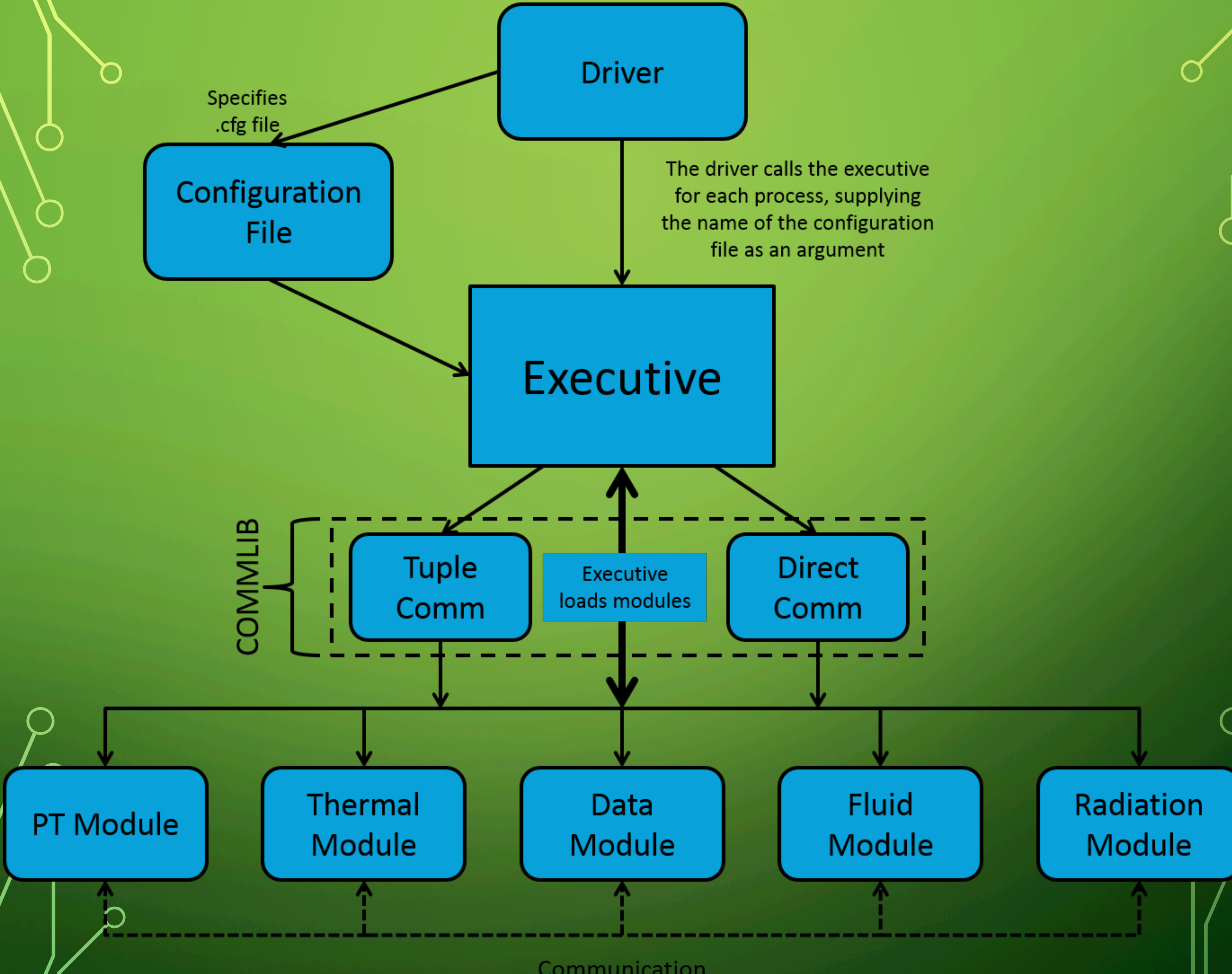
- Consists of the “Executive” and a communication library
- Executive reads a simple configuration file to execute desired modules and define the shared boundary points between them
- Communication library consists of two parts:
  - **Direct communication –** wrappers for MPI_Send() and MPI_Recv() that enforce shared boundary conditions
  - **Indirect communication –** global “tuple space” used to store data until it is needed

```
# 2 modules, 2 processes per module, 4 points per
shared boundary condition

shared_bc_sizes = [4,4];

modules = (
{
    function="Radiosity_Module_V3";
    library="librad.so";
    size=2;
    points=(
      ( [0,1,2,3], [] ),
      ( [], [0,1,2,3] )
    );
},
{
    function="Thermal_Module_V1";
    library="libtherm.so";
    size=2;
    points=(
      ( [0,1,2,3], [] ),
      ( [], [0,1,2,3] )
    );

}
);
```


Driver
Specifies .cfg file
Configuration File
The driver calls the executive for each process, supplying the name of the configuration file as an argument
Executive
COMMLIB
Tuple Comm
Executive loads modules
Direct Comm
PT Module
Thermal Module
Data Module
Fluid Module
Radiation Module
Communication

TRADITIONAL CODE USING C

- Can be executed on most machines that have a C compiler
- Traditional libraries do not properly accommodate supercomputer resources

C TRADITIONAL CODE AND DIEL

- Needs to be properly called from the configuration file
- Function names and definitions need to be modified accordingly
- Very time consuming and potentially confusing to convert

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

int main()
{
    printf("Hello from FirstModule\n");

    return EXIT_SUCCESS;
}
```

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

#include <mpi.h>                    /*these*/
#include "IEL.h"                    /*are*/
#include "IEL_exec_info.h"          /*required to use the IEL functions*/


int FirstModule(IEL_exec_info_t *exec_info)
{
    printf("Hello from FirstModule\n");

    return EXIT_SUCCESS;
}
```

NON-C TRADITIONAL CODE AND DIEL

- Allows users larger access to previously written code
- Allows users the benefits of other languages while still providing the benefits of using the DIEL
- Successfully developed methods that allow Fortran and JAVA based codes to be executed using DIEL

REPETITION AND SERIAL DIEL CODE

- Useful for collecting simulation data
- Successfully developed methods to execute code multiple times simultaneously across several processors

SCALABILITY

- Ability of a system to expand to accommodate a given work load
- By using repetition on serial code we can receive a benchmark on how a system adapts to a huge work load

FUTURE OBJECTIVES

- Perform scalability tests on more systems
- Create universal scripts
- Give more DIEL access to Fortran and JAVA users
- Possibly develop a method to break down existing code and parallelize sections
- Possibly incorporate more languages

WHAT IS A TUPLE SPACE?

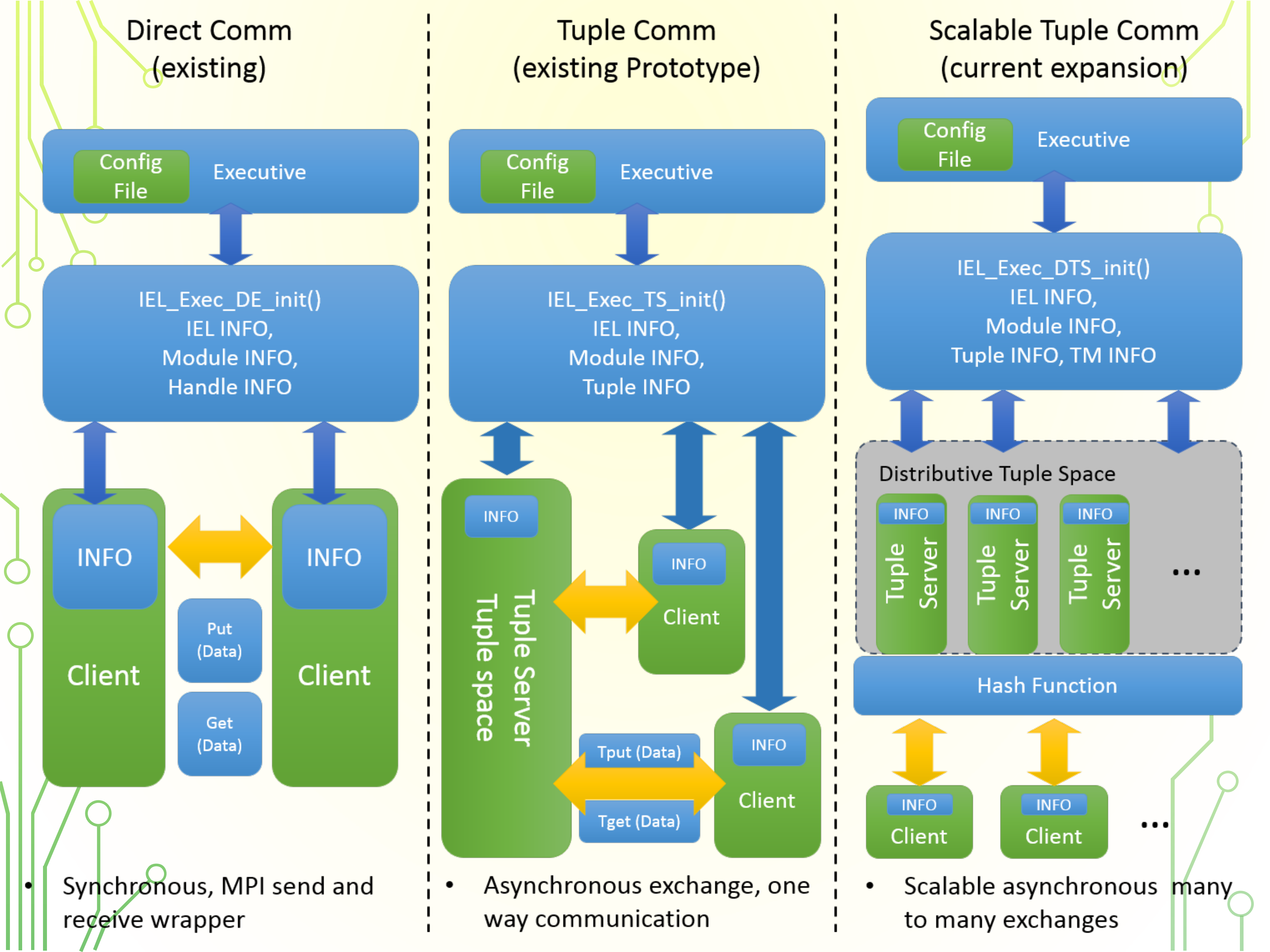
- Basically, it is **associative memory** that can be accessed concurrently.
- **Associative memory** means that the pieces of data, or "tuples", are indexed according to whichever abstract, human-intuitive idea they represent.
- Tuple spaces have multiple uses. The DIEL uses one to achieve **asynchronous, stochastic** inter-process communication.

THE EXISTING PROTOTYPE

- Tuple-space communication consisted of a single server process processing “put” and “get” requests in sequence
- The server was a special function that was called on rank 0 by the executive
- Not concurrent, therefore not a true tuple space
- Associativity was implemented, but it was completely arbitrary. The user would simply choose any “tag” for each tuple when putting it to the server

DESIRED END

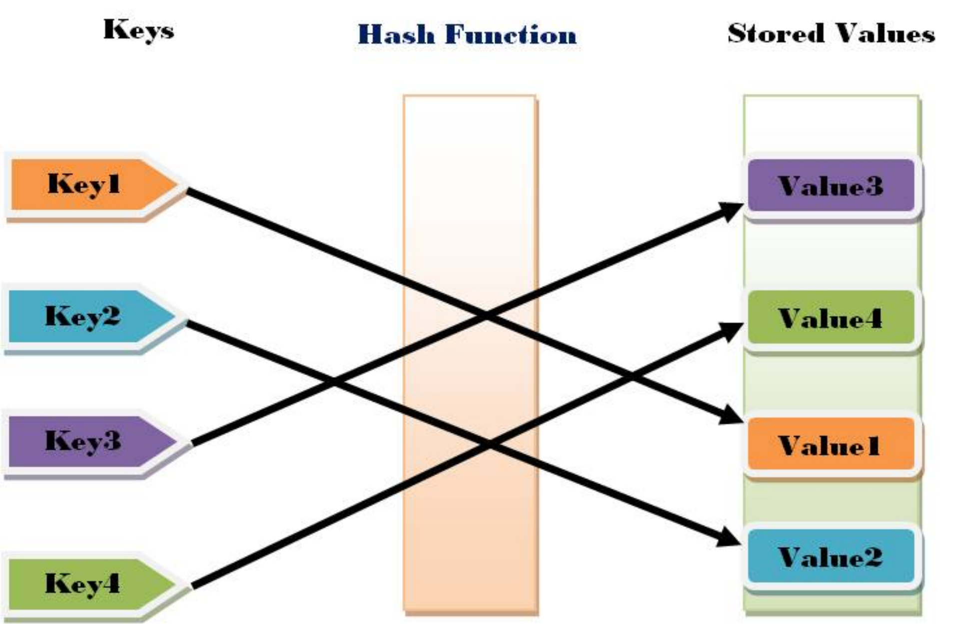
- The tuple server is a DIEL module, like any other
- Multiple servers running in parallel
- Each server controls an equal portion of the overall tuple space
- Data are indexed according to the same shared boundary conditions used for direct communication
- The data structure used is a **distributed hash table**


Direct Comm
(existing)
Config File
Executive
IEL_Exec_DE_init()
IEL INFO,
Module INFO,
Handle INFO
INFO
Client
INFO
Client
Put (Data)
Get (Data)
Synchronous, MPI send and receive wrapper
Tuple Comm
(existing Prototype)
Config File
Executive
IEL_Exec_TS_init()
IEL INFO,
Module INFO,
Tuple INFO
INFO
Tuple Server
Tuple space
INFO
Client
Tput (Data)
Tget (Data)
INFO
Client
Asynchronous exchange, one way communication
Scalable Tuple Comm
(current expansion)
Config File
Executive
IEL_Exec_DTS_init()
IEL INFO,
Module INFO,
Tuple INFO, TM INFO
Distributive Tuple Space
INFO
Tuple Server
INFO
Tuple Server
INFO
Tuple Server
...
Hash Function
INFO
Client
INFO
Client
...
Scalable asynchronous many to many exchanges

DISTRIBUTED HASH TABLES

- In a hash table, a **hash function** calculates the proper index for data element based on its associated key
- In a distributive hash table, the hash function returns the proper node as well as the index on the node

- This means we do not need to pass messages between multiple processes just to find out where our data element is located



HOW AND WHY IT WORKS

- Each of the shared boundary conditions in the configuration file is assigned an integer-value ID
- The hash function uses modulus to determine the correct tuple server, and again to determine the correct index

SBC_ID mod NUM_SERV = server

SBC_ID mod NUM_IDX = index

HOW AND WHY IT WORKS (CONT.)

- DIEL modules have two functions for interacting with the tuple space:

  Producer: IEL_tput(&data, size, sbc)

  Consumer: IEL_tget(&data, &size, sbc)

- **Since the hash function always returns the same values for the same input, if IEL_tput and IEL_tget both call the hash function, they will get back the same location**
- So they will look in the same place without directly communicating with each other!

ANTICIPATING A STOCHASTIC PROCESS

- A major challenge with most parallel systems is that they are, from the programmer's point of view, nondeterministic
- The actual sequence of events will usually be different every time the program is run because every process is individually subject to a large number of uncontrollable variables
- A robust tuple server algorithm must be able to anticipate and handle all possible sequences short of a catastrophic hardware failure

ANTICIPATING A STOCHASTIC PROCESS (CONT.)

For example, consider having a producer module and a consumer module. The producer module is delayed by the operating system, and the consumer calls IEL_tget on the relevant data before the producer calls IEL_tput. So the tuple server is faced with being asked for data that it does not have.

When I started development, the existing tuple server algorithm could not handle this case. The system would become deadlocked and never complete.

A RANDOMIZED STRESS TEST

- Do this ten times, with ITER starting at 0:
  - Send your rank id to the tuple space using your rank ID plus ITER as the input to the hash function
  - Do this until you are done:
    - Based on the number of module processes, pick a rank ID at random
    - Request that ID from the tuple space, using the ID plus ITER as the input to the hash function
    - Repeat until you have received every rank ID in the system, including your own, at which point you are done
  - Increment ITER and repeat

RESULTS OF TEST

- Due to the randomized nature of the test, we should run it many times and then look at the distribution of completion times.
- 16 tuple servers, 256 module processes on Darter
- After 40 trials, the tuple servers collectively fulfill an average of 9.6 million tget/tput requests per trial
- It takes an average of 7.5 seconds to complete