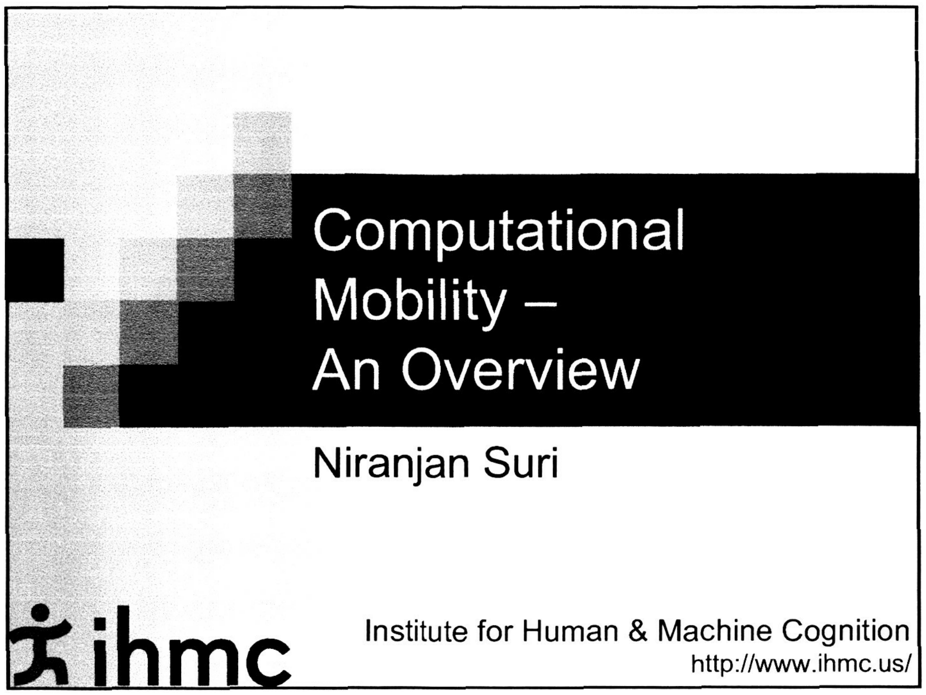
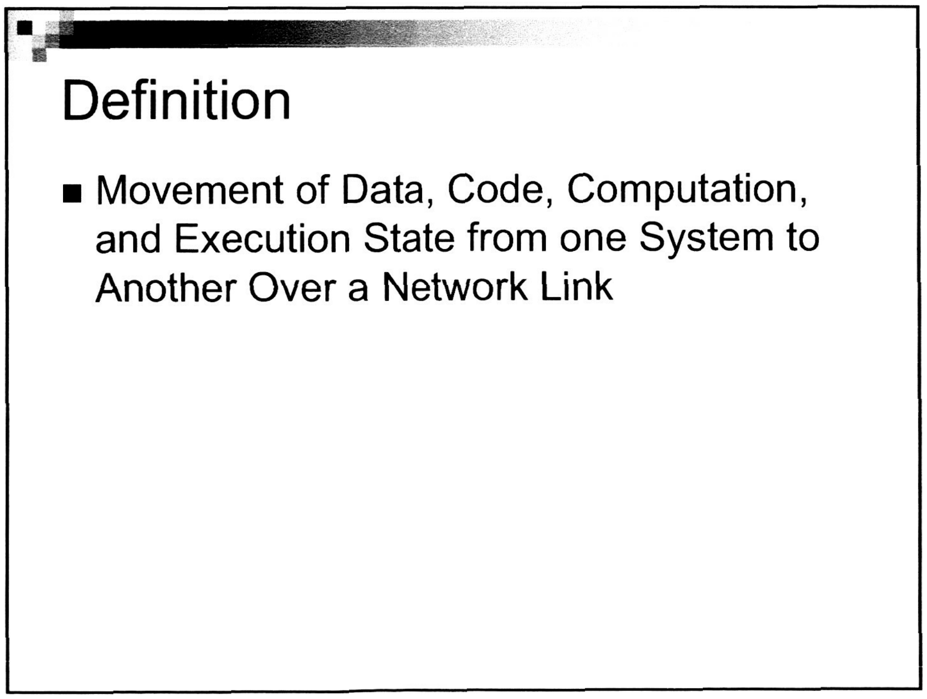


Computational Mobility – An Overview

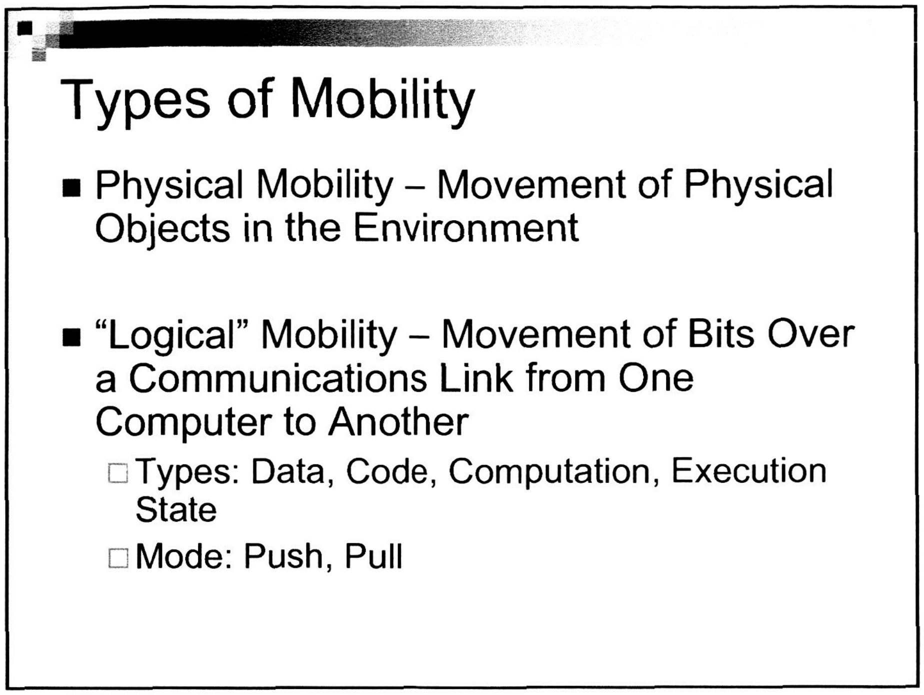
Niranjan Suri

ihmc Institute for Human & Machine Cognition
http://www.ihmc.us/



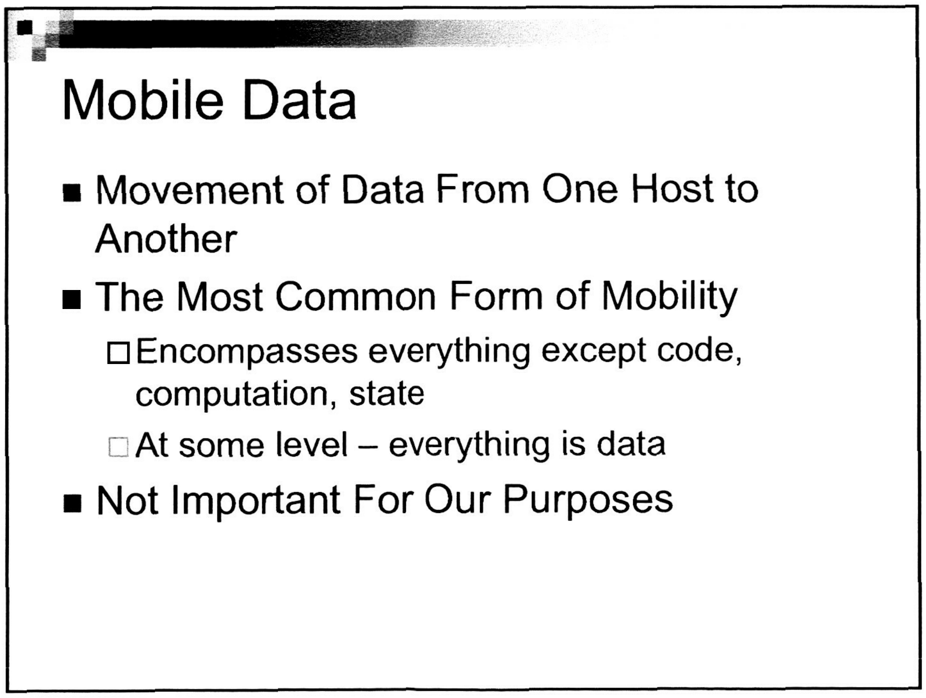
Definition

- Movement of Data, Code, Computation, and Execution State from one System to Another Over a Network Link



Types of Mobility

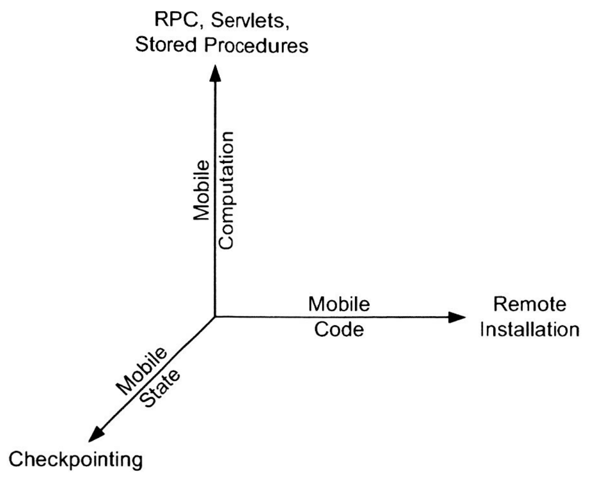
- Physical Mobility – Movement of Physical Objects in the Environment
- "Logical" Mobility – Movement of Bits Over a Communications Link from One Computer to Another
  - Types: Data, Code, Computation, Execution State
  - Mode: Push, Pull



Mobile Data

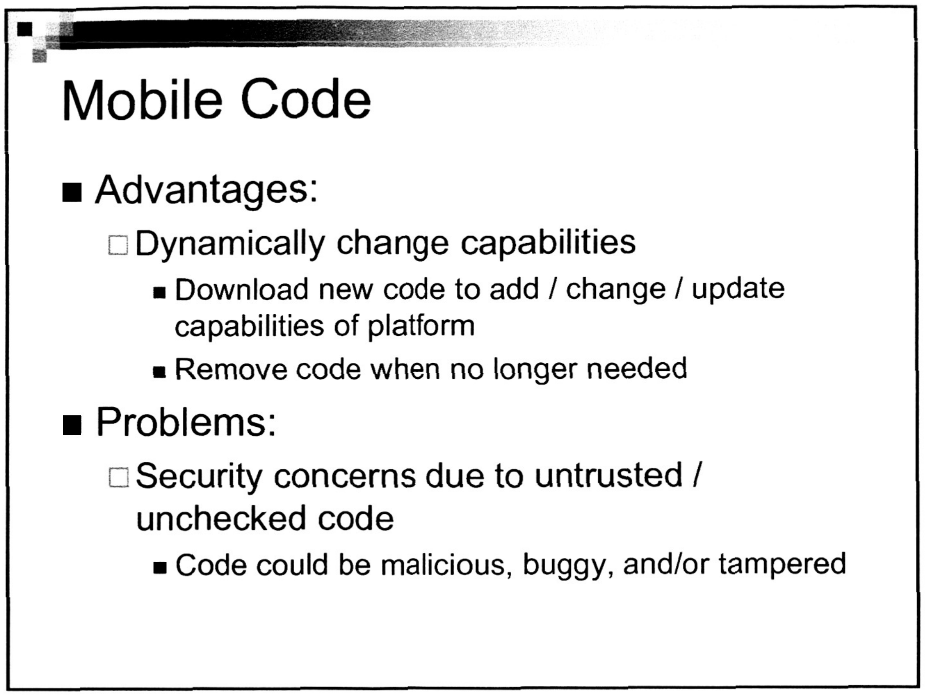
- Movement of Data From One Host to Another
- The Most Common Form of Mobility
  - Encompasses everything except code, computation, state
  - At some level – everything is data
- Not Important For Our Purposes

Mobility – Another Perspective



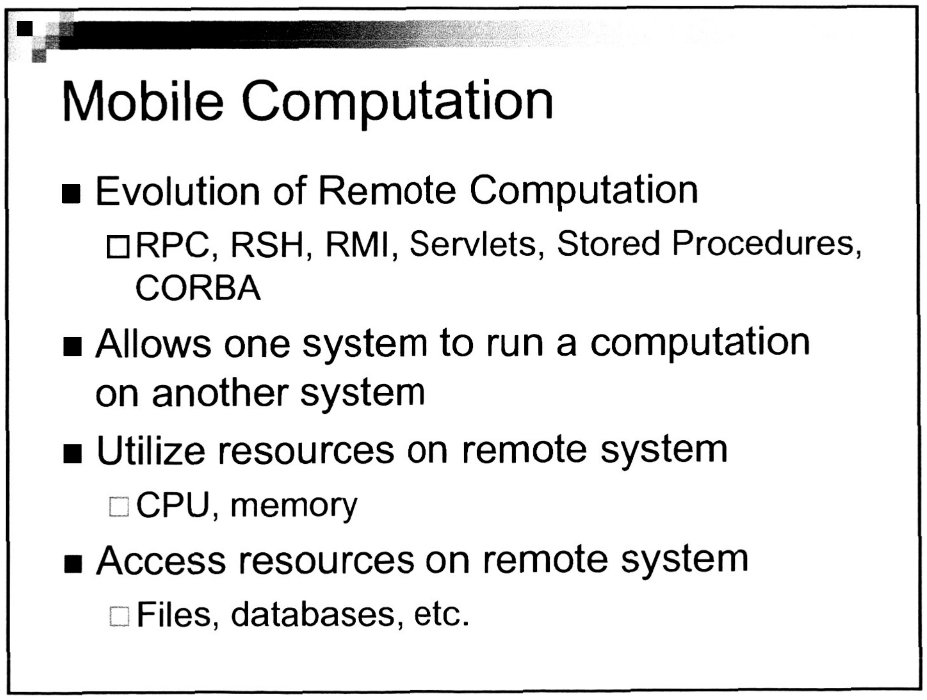
Mobile Code

- Allows executable code to be moved to a new host
- May use the push or pull model
  - Pull: Applets
  - Push: Remote Installation
- Code may be binary (intermediate or native) or source



Mobile Code

- Advantages:
  - Dynamically change capabilities
    - Download new code to add / change / update capabilities of platform
    - Remove code when no longer needed
- Problems:
  - Security concerns due to untrusted / unchecked code
    - Code could be malicious, buggy, and/or tampered



Mobile Computation

- Evolution of Remote Computation
  - RPC, RSH, RMI, Servlets, Stored Procedures, CORBA
- Allows one system to run a computation on another system
- Utilize resources on remote system
  - CPU, memory
- Access resources on remote system
  - Files, databases, etc.

Mobile State

- Evolution of State Capture
  - Checkpointing
- Allows execution state of a process to be captured and moved
- State may be machine specific or machine independent
- May contain
  - State of single or multiple threads
  - Code

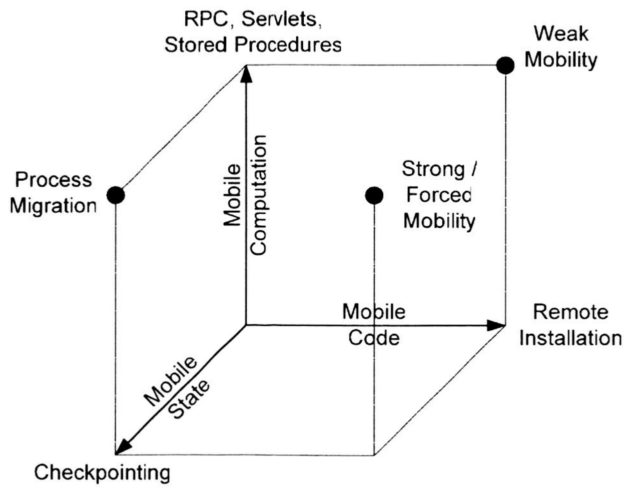
Mobility Matrix

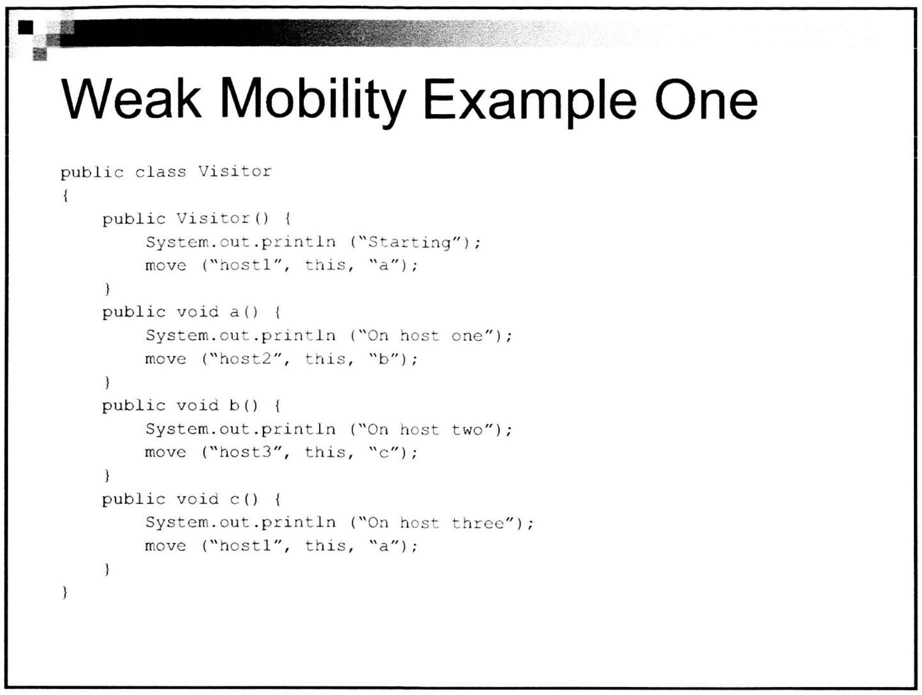
	Data	Code	Computation	Execution State
Pull	Web Browsing, SETI@home	Applets, JavaScript, Jini	While You're Away (WYA)	While You're Away (WYA)
Push	FTP Upload	Remote Installation, Mobile Agents	RPC, RMI, Grid Computing	Process Migration

Weak -vs- Strong -vs- Forced

- Weak Mobility
  - Computing entity requests movement
  - Entity “restarts” execution after move operation
  - Combines Mobile Code and Mobile Computation
- Strong Mobility
  - Computing entity requests movement
  - Execution continues after movement
  - Combines Mobile Code, Mobile Computation, and Mobile State
- Forced Mobility
  - External, asynchronous request for movement
  - Execution continues after movement
  - Computing entity may not be aware of movement
  - Combines Mobile Code, Mobile Computation, and Mobile State

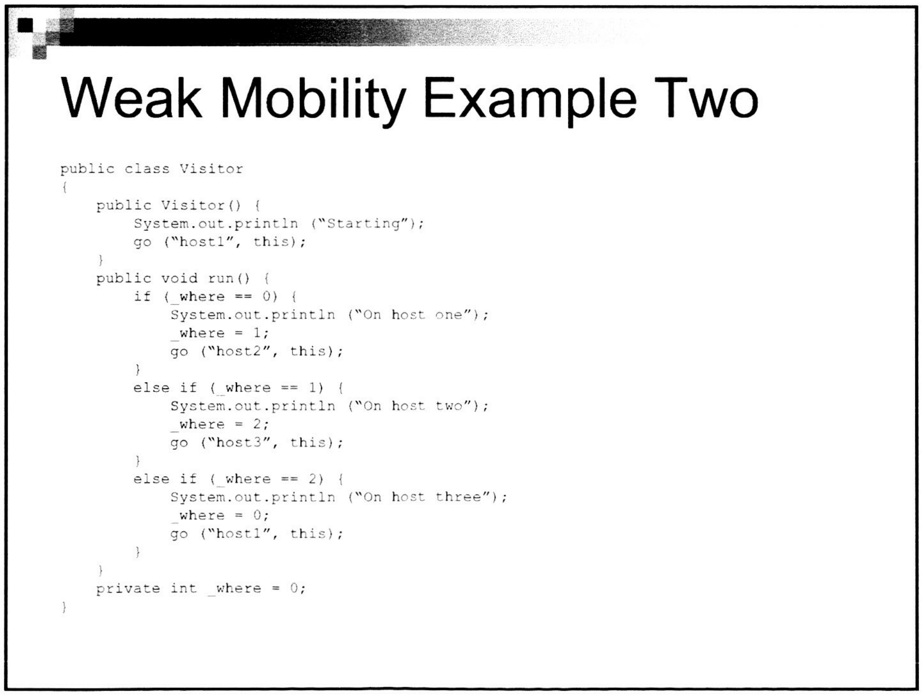
Weak -vs- Strong -vs- Forced





Weak Mobility Example One

```
public class Visitor
{
    public Visitor() {
        System.out.println ("Starting");
        move ("host1", this, "a");
    }
    public void a() {
        System.out.println ("On host one");
        move ("host2", this, "b");
    }
    public void b() {
        System.out.println ("On host two");
        move ("host3", this, "c");
    }
    public void c() {
        System.out.println ("On host three");
        move ("host1", this, "a");
    }
}
```



Weak Mobility Example Two

```
public class Visitor
{
    public Visitor() {
        System.out.println ("Starting");
        go ("host1", this);
    }
    public void run() {
        if (_where == 0) {
            System.out.println ("On host one");
            _where = 1;
            go ("host2", this);
        }
        else if (_where == 1) {
            System.out.println ("On host two");
            _where = 2;
            go ("host3", this);
        }
        else if (_where == 2) {
            System.out.println ("On host three");
            _where = 0;
            go ("host1", this);
        }
    }
    private int _where = 0;
}
```

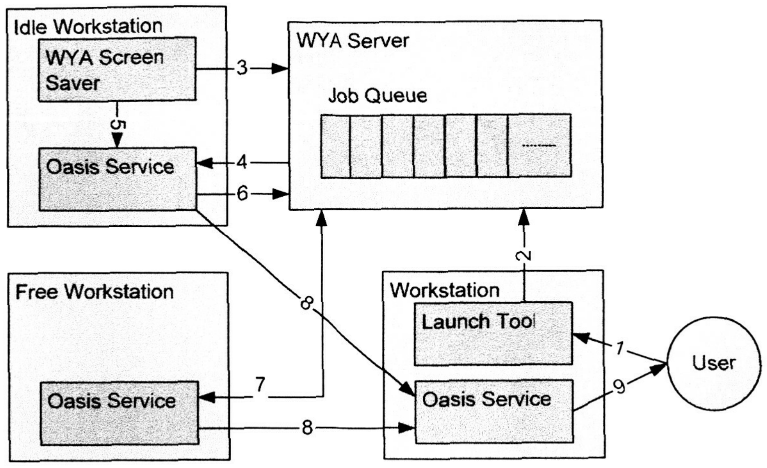
Strong Mobility Example

```
public class Visitor
{
    public Visitor()
    {
        System.out.println ("Starting");
        while (1) {
            go ("host1");
            System.out.println ("On host one");
            go ("host2");
            System.out.println ("On host two");
            go ("host3");
            System.out.println ("On host three");
        }
    }
}
```

Forced Mobility Example

- Visitor Not Appropriate
  - Mobility is dictated by external entity
- Examples:
  - Survivability
  - Load-balancing
- Concrete Example – While You're Away (WYA)
  - System for utilizing idle workstations
  - Abstraction – roaming computations

WYA Design



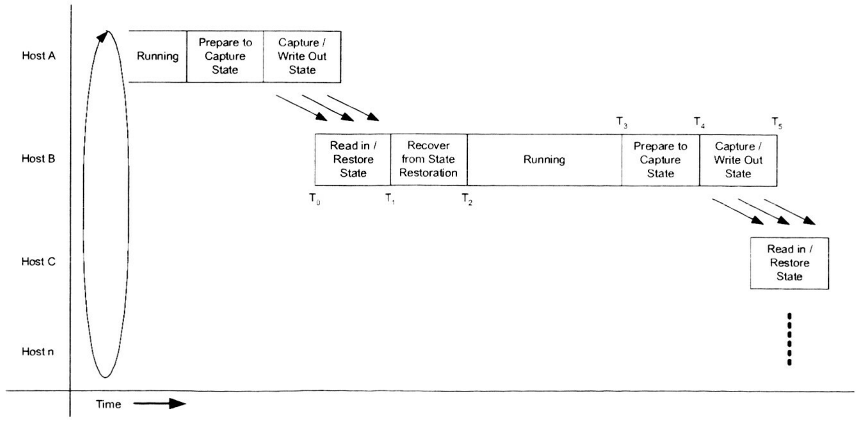
WYA Programming Abstraction

```
public class MyComputation extends RoamingComputation
{
    public void init (String args[])
    {
        // Perform any initialization required here
    }

    public void compute()
    {
        // Actual computations go here
    }

    public void reportResults()
    {
        // Report results back to the user here
    }
}
```

Forced Mobility Example Two

```
public class Jumper
{
    public Jumper() {
        System.out.println ("Starting");
        new Mover().start();
        while (1) {
            System.out.println ("hello, world");
        }
    }

    public class Mover extends Thread
    {
        public void run() {
            for (int i = 0; i < hosts.length; i++) {
                go (hosts[i]);
                Thread.sleep (100);
            }
        }
    }
}
```

Process Cycle



Mobility Abstraction

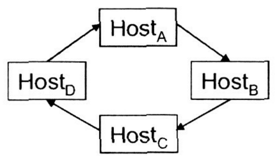
- Process is Continuously Moving
- Code Has no Knowledge of Current Host
- Code Prefixes Operation with a Scope that Identifies the Host
- Operation Gets Performed when Process is on that Host

Visitor Example Revisited

```
public class Visitor
{
    public Visitor()
    {
        System.out.println ("Starting");
        while (1) {
            h1.System.out.println ("On host one");
            h2.System.out.println ("On host two");
            h3.System.out.println ("On host three");
        }
    }
}
```

One Possible Realization...

- Hosts Form a Logical Ring
- Process is Created on one Host
- At Fixed Intervals (Timeslices?), Process is Migrated from Host$_n$ to Host$_{n+1}$
- Generic Operations May be Performed on Any Host
- Operations Qualified by a Host will be Performed only on that Host
  - Runtime system blocks until process is on required host
  - Runtime system possibly leaves process on required host until operation is completed
  - A form of critical section


$$\begin{tikzcd} & \text{Host}_A \arrow[dr] & \\ \text{Host}_D \arrow[ur] & & \text{Host}_B \arrow[dl] \\ & \text{Host}_C \arrow[ul] & \end{tikzcd}$$

Another Example

```
public class WasteTime
{
    public WasteTime()
    {
        System.out.println ("Starting");
        while (1) {
            float a = h1.readValue();
            float b = Math.sin (a);
            float c = Math.cos (b);
            h2.writeValue (c);
            float d = Math.acos (c);
            float e = Math.asin (d);
            h3.writeValue (e);
        }
    }
}
```

Variation on the Theme

- Process Migration Path is Determined by Operation to be Performed
  - If program wants to do something on $Host_p$, migrate directly to $Host_p$
- Could Result in Certain Hosts being Ignored
  - Undesirable if hosts deliver asynchronous events to process

Interesting "Performance" Questions

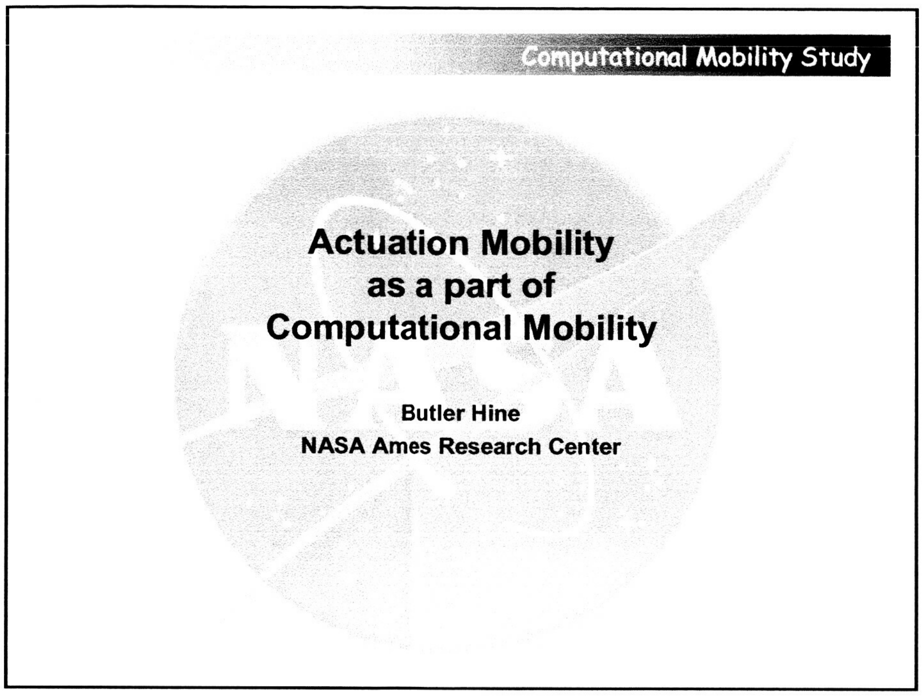
- What is a Good Timeslice?
- What is the Maximum Number of Hosts?
- When do you Start Thrashing?
- Answers Depend on Current State of the Art in Implementation
- What can we Project about the Future?

Interesting "Abstraction" Questions

- What is the Best Abstraction?
  - Is mobility dictated by the program?
  - Is program dictated by the mobility?
- What about Time?
  - Can `System.currentTimeMillis()` run anywhere?
  - Will cause clock synchronization problems
- Division Between Higher-order Functions and Lower-order Functions
- Splitting / Joining Groups
  - Equivalent of a fork() / join()?

Available Resource – Aroma VM

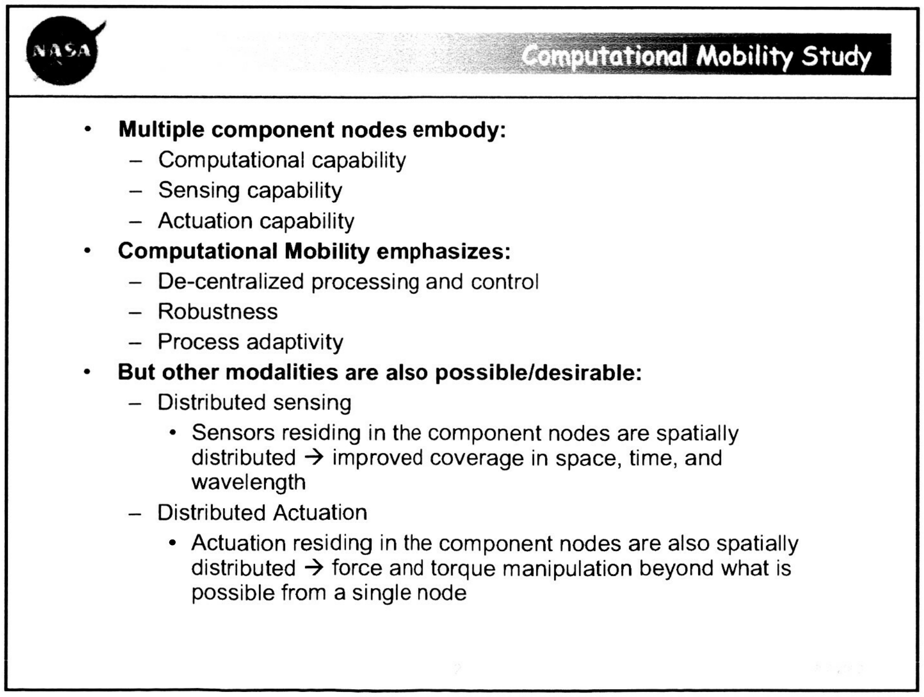
- Clean-room implementation
- State capture mechanism
- Dynamic, fine-grained resource control
  - Disk, Network, CPU
- JDK 1.2.2 compatible
  - Uses Java Platform API from JRE 1.2.2
  - No AWT / Swing
- Ported to Win32 (x86), Linux (x86), Solaris (SPARC)
- No Just-In-Time compilation (in progress)



Computational Mobility Study

Actuation Mobility as a part of Computational Mobility

Butler Hine
NASA Ames Research Center

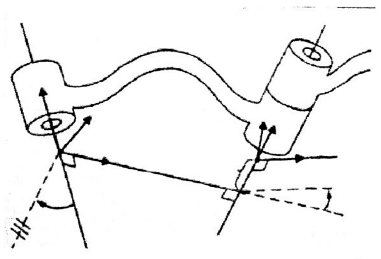


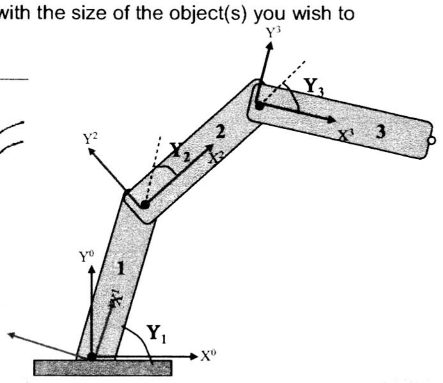
Computational Mobility Study

- **Multiple component nodes embody:**
  - Computational capability
  - Sensing capability
  - Actuation capability
- **Computational Mobility emphasizes:**
  - De-centralized processing and control
  - Robustness
  - Process adaptivity
- **But other modalities are also possible/desirable:**
  - Distributed sensing
    - Sensors residing in the component nodes are spatially distributed → improved coverage in space, time, and wavelength
  - Distributed Actuation
    - Actuation residing in the component nodes are also spatially distributed → force and torque manipulation beyond what is possible from a single node

Traditional Robotics

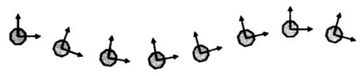
- **Kinematic chains**
  - Forces and torques are transmitted through *mechanical* linkages to the end effector
  - The system is limited in the external forces and torques it can exert through the end effector by the kinematic chain
  - System mass and size scales with the size of the object(s) you wish to manipulate





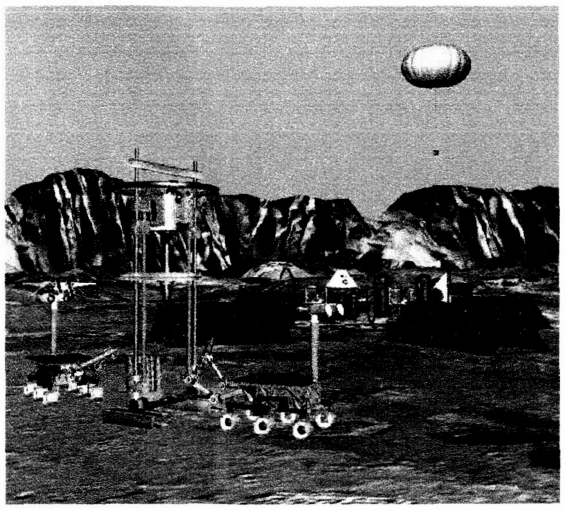
Distributed Actuation

- **Distributed Computational/Sensing/Actuation Nodes**
  - Forces and torques are transmitted by each unconnected node
  - External forces and torques are possible that are not limited by any mechanical connection
  - System mass becomes independent of the size of the object(s) you wish to manipulate



  - Possibilities:
    - Conformal Forces
      - Lifting/positioning large objects
      - Lifting/positioning delicate object
      - Multi-component assembly
      - Large size-scale compressive forces
      - Large size-scale expansion forces

Robot Team Scenario



Robotic exploration of Mars

- Mobile robots will serve as the remote sensors and data collectors for scientists.
- To create an outpost for such long-term exploration, robots need to
  - assemble solar power generation stations,
  - map sites and collect science data,
  - communicate with Earth on a regular basis.
- In one scenario, a large number of robots (20-30) are sent, many with different capabilities. Some of the robots specialize in heavy moving and lifting, some in science data collection, some in drilling and coring, and some in communication. The rovers have different, but overlapping, capabilities – different sensors, different resolutions and fields of view, even different mobility, such as both wheeled and aerial vehicles.

Robotic exploration of Mars

- Upon landing, the rovers search for a location suitable in size and terrain for a base station.
- Once such a location is found, rovers with appropriate capabilities form several teams to construct the base station capable of housing supplies and generating energy.
  - Two rovers carry parts, such as solar panels, that are too large for a single rover.
  - Complementary capabilities are exploited – for example, to align and fasten trusses, rovers with manipulators receive assistance from camera-bearing rovers that position themselves for advantageous viewing angles.

Robotic exploration of Mars

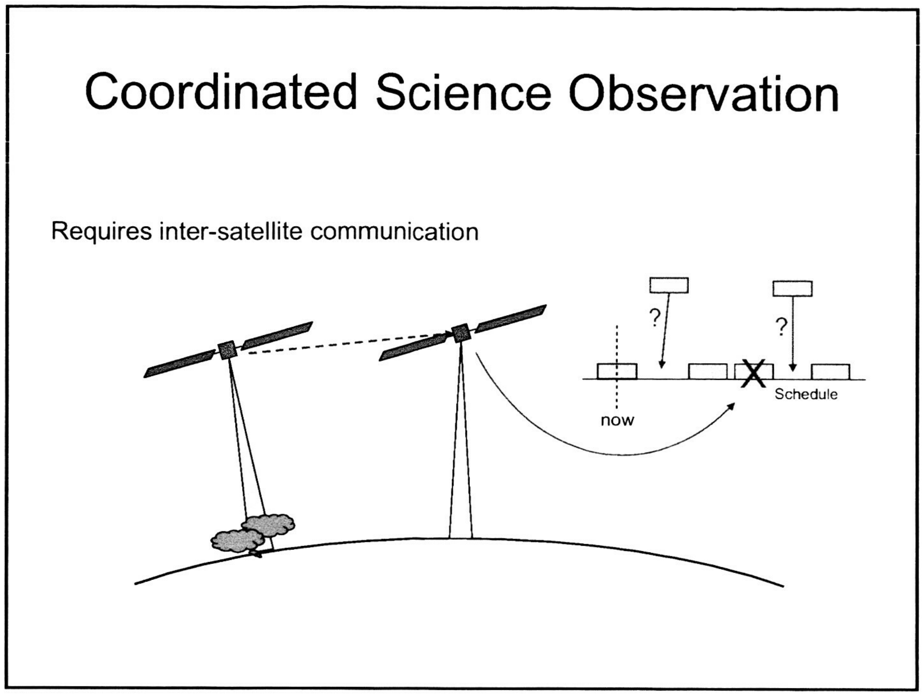
- Rover failures are addressed by dispatching a rover with diagnostic capabilities. The diagnostic rover can use its cameras to view the failed robot to see if it can be aided in the field (e.g., if it has a stuck wheel or is high-centered), or it may drag the rover back to the base station to be repaired by replacement of failed modules. In the meantime, another robot with the same (or similar) capabilities can be substituted, so as to complete the original task with minimal interruptions.

Robotic exploration of Mars

- At any given time, different teams of rovers may be involved in exploration, base-station construction/maintenance, and rover diagnosis/repair.
- Many tasks will be time critical, requiring execution within hard deadlines (e.g., repair of a failed power generation station) or synchronization with external events (communication satellite visibility, periods of sunlight).
- The teams form dynamically, depending on the task, environment, and capabilities and availability of the various robots to best meet mission requirements over time.
- The rovers negotiate their individual roles, ensure safety of the group and themselves, and coordinate their precise actions, attempting as a group to avoid unnecessary travel time, to minimize reconfiguration and wait time, and to prefer more reliable alternatives in cases of overlapping capabilities.
- The challenge is to keep all the robots healthy and busy in appropriate tasks, in order to maximize the scientific data collected.

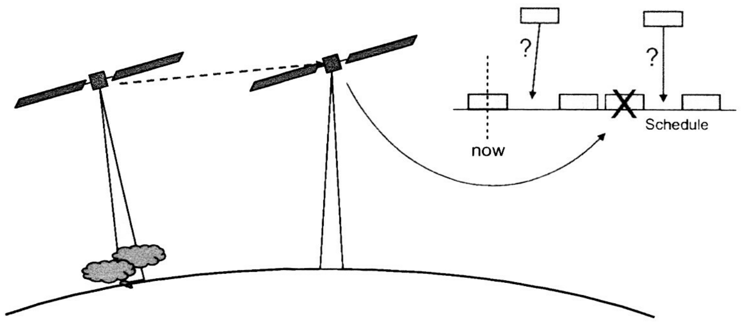
Robotic exploration of Mars

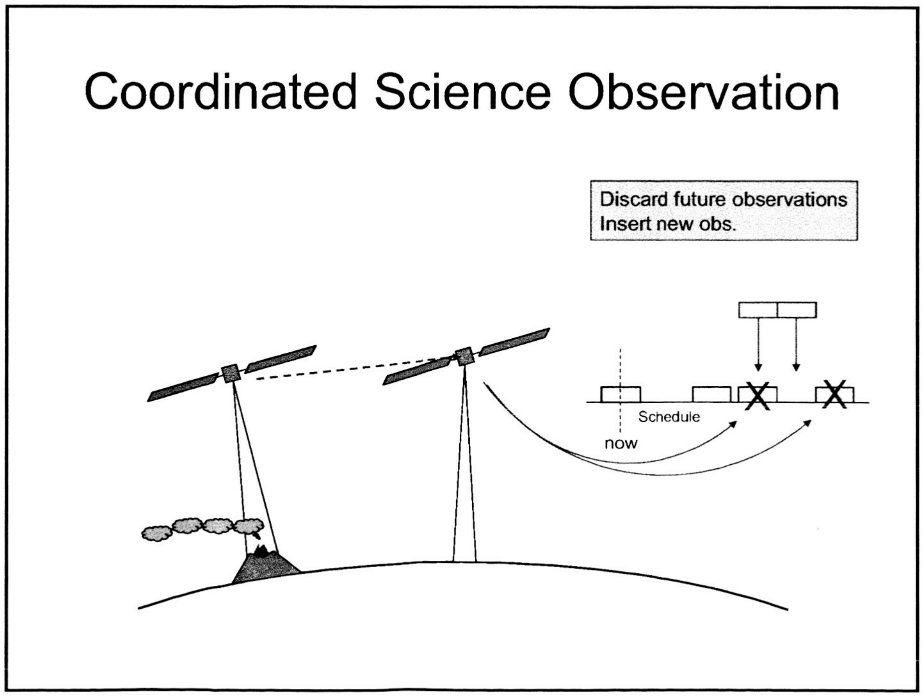
- Similar scenarios exist for domains such as habitat construction, space solar power construction and maintenance, and Space Station maintenance.
  - For instance, consider an inspection robot that has identified a failed component on the Space Station. It tries to assemble a team of robots to replace the failed component. After negotiation, a courier robot (capable of retrieving the necessary replacement part) and a repair robot (capable of swapping-out the failed device) take responsibility for the repair task, leaving the inspection robot free to continue inspection. While the courier collects the replacement part, the repair robot evaluates the problem and plans its course of action, possibly seeking additional aid if it encounters unexpected difficulties it is unable to resolve. Upon arrival with the replacement part, the courier and repair robot tightly coordinate their actions, turning themselves into what is effectively a single high degree-of-freedom robot.



Coordinated Science Observation

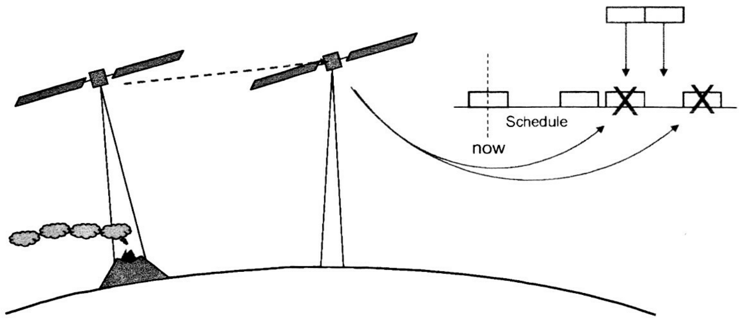
Requires inter-satellite communication



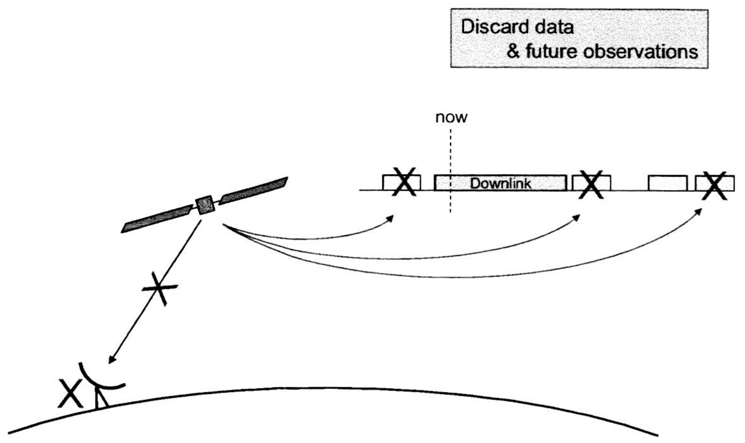


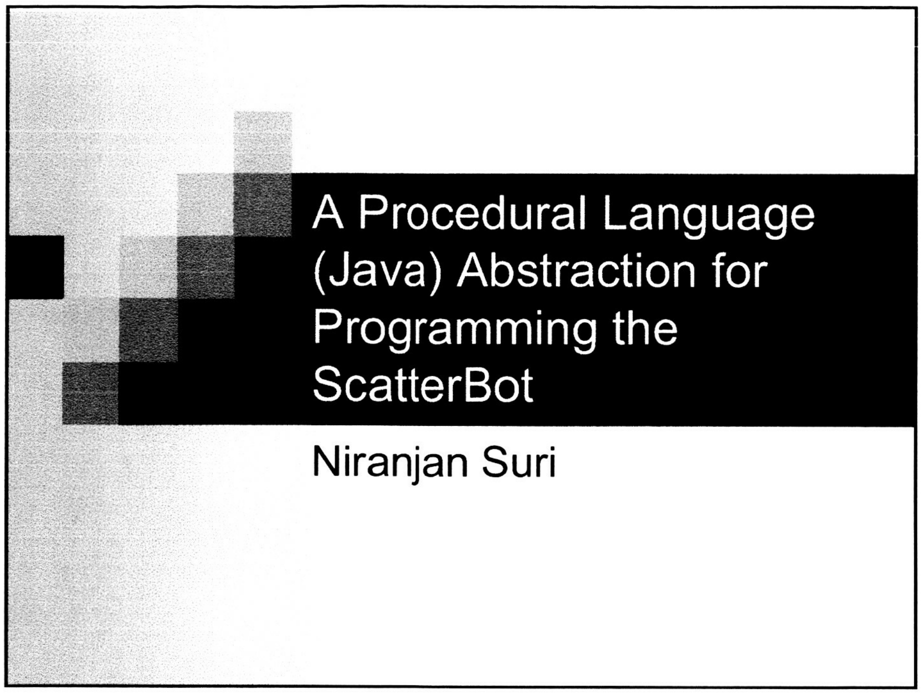
Coordinated Science Observation

Discard future observations
Insert new obs.



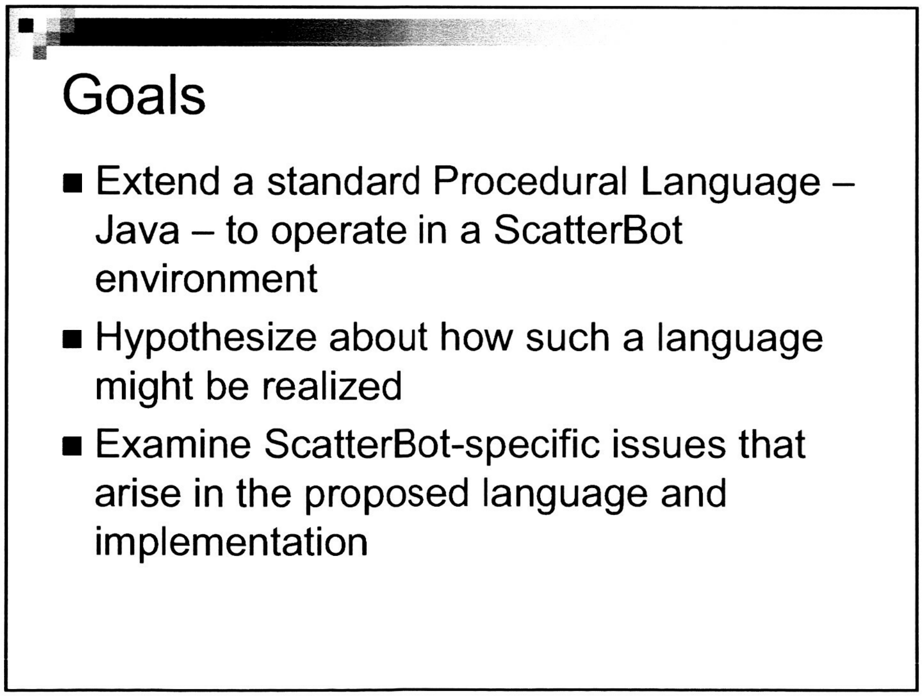
Coordinated Science Observation





A Procedural Language (Java) Abstraction for Programming the ScatterBot

Niranjan Suri



Goals

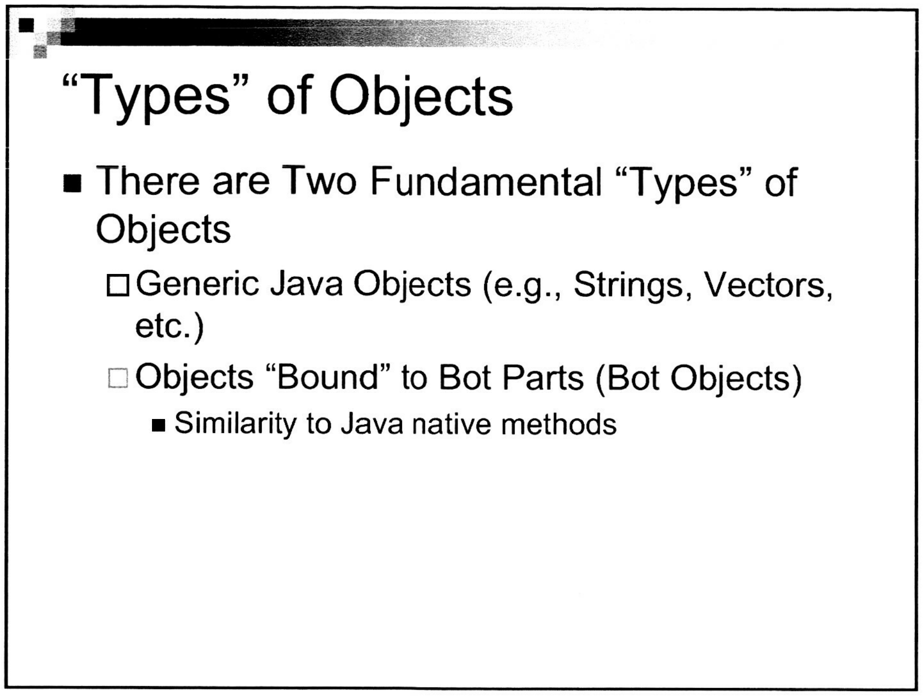
- Extend a standard Procedural Language – Java – to operate in a ScatterBot environment
- Hypothesize about how such a language might be realized
- Examine ScatterBot-specific issues that arise in the proposed language and implementation

Questions

- Is it right to call Java a procedural language?

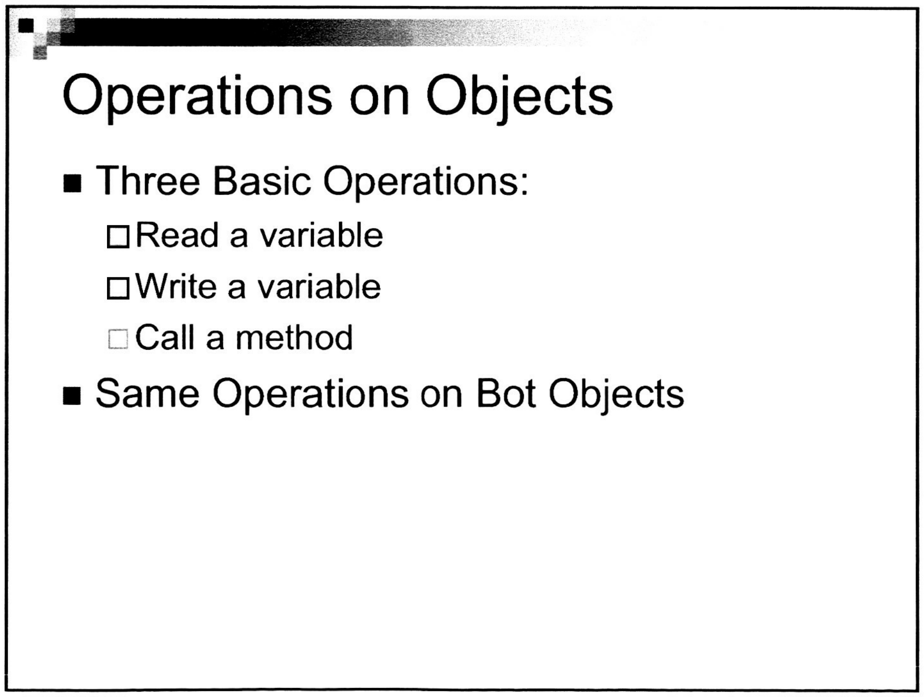
Basics

- Assume that each bot part is represented by an object (an instance of some class)
    - The type of the object (i.e., the class) represents the type of the bot-part
    - We can leverage object-oriented notions of subclassing (is-a relationships) and containment (part-of relationships) to model bot-parts



"Types" of Objects

- There are Two Fundamental "Types" of Objects
  - Generic Java Objects (e.g., Strings, Vectors, etc.)
  - Objects "Bound" to Bot Parts (Bot Objects)
    - Similarity to Java native methods



Operations on Objects

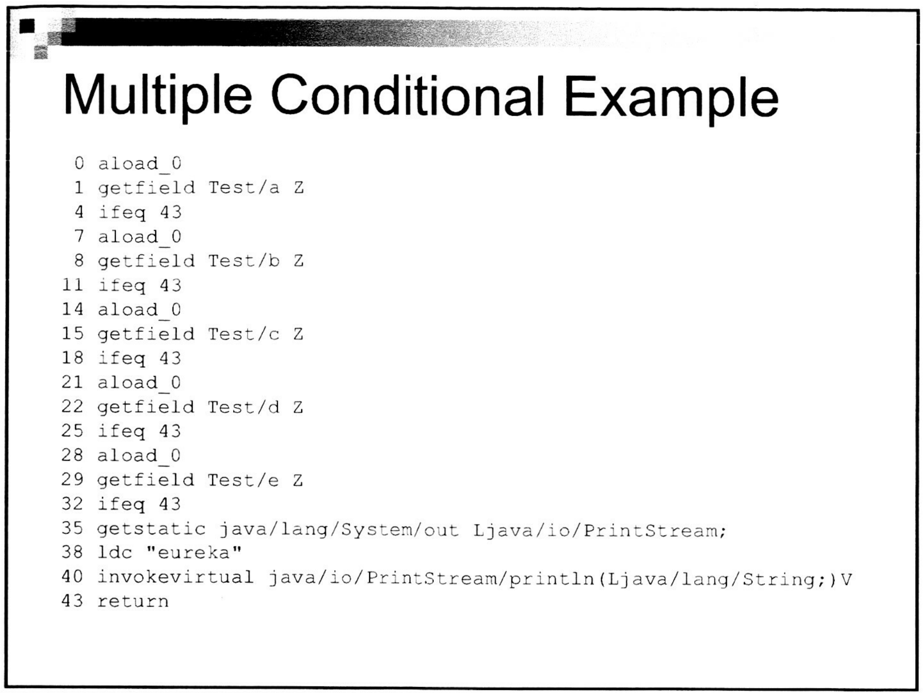
- Three Basic Operations:
  - Read a variable
  - Write a variable
  - Call a method
- Same Operations on Bot Objects

Implementation Thoughts

- At the Java VM level, most bytecodes manipulate the operand stack and local variables - these can execute anywhere
- There are 3 types of bytecodes to worry about:
  - putstatic, getstatic
  - putfield, getfield
  - Invokevirtual, invokestatic, invokeinterface, invokespecial
- If any of these are executed on a Bot Object, the VM must execute the resultant operation only on the corresponding Bot part

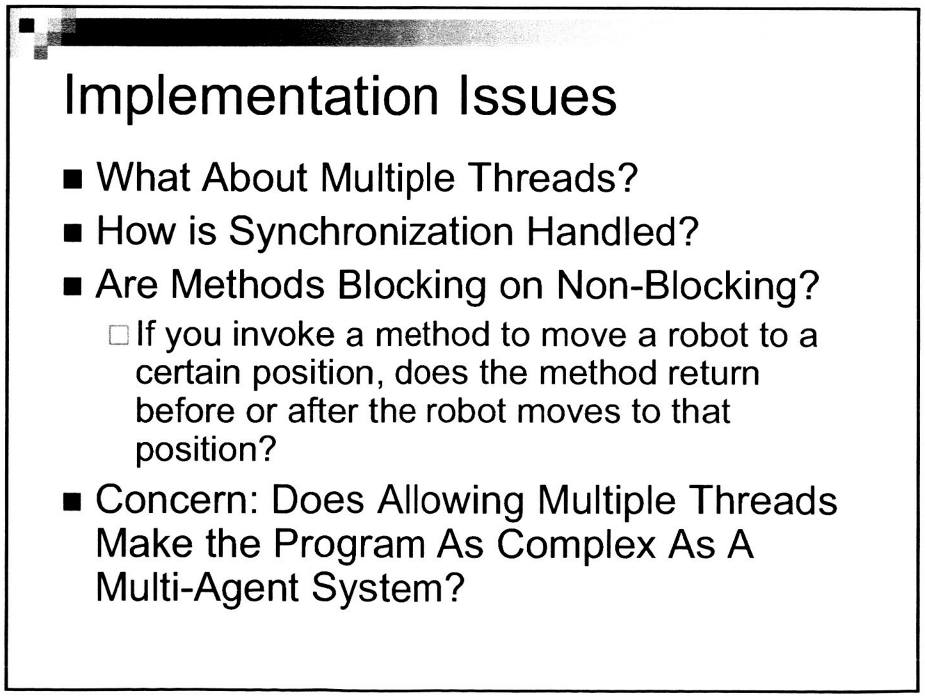
Multiple Conditional Example

```
public class Test
{
    public void doSomething()
    {
        if (a && b && c && d && e) {
            System.out.println ("eureka");
        }
    }

    private boolean a;
    private boolean b;
    private boolean c;
    private boolean d;
    private boolean e;
}
```



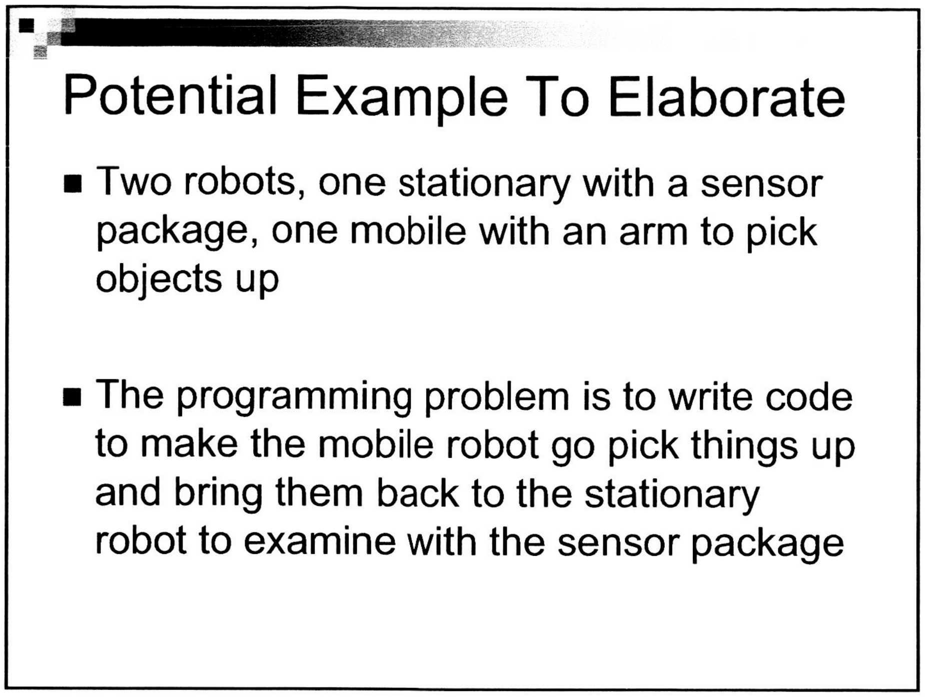
Multiple Conditional Example

```
 0 aload_0
 1 getfield Test/a Z
 4 ifeq 43
 7 aload_0
 8 getfield Test/b Z
11 ifeq 43
14 aload_0
15 getfield Test/c Z
18 ifeq 43
21 aload_0
22 getfield Test/d Z
25 ifeq 43
28 aload_0
29 getfield Test/e Z
32 ifeq 43
35 getstatic java/lang/System/out Ljava/io/PrintStream;
38 ldc "eureka"
40 invokevirtual java/io/PrintStream/println(Ljava/lang/String;)V
43 return
```



Implementation Issues

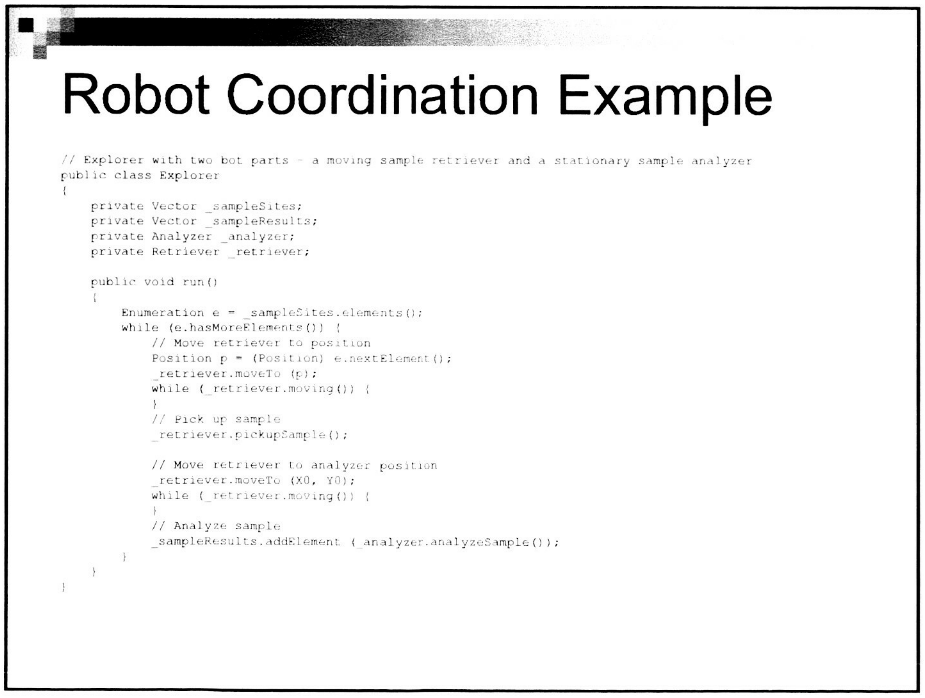
- What About Multiple Threads?
- How is Synchronization Handled?
- Are Methods Blocking on Non-Blocking?
  - If you invoke a method to move a robot to a certain position, does the method return before or after the robot moves to that position?
- Concern: Does Allowing Multiple Threads Make the Program As Complex As A Multi-Agent System?



Potential Example To Elaborate

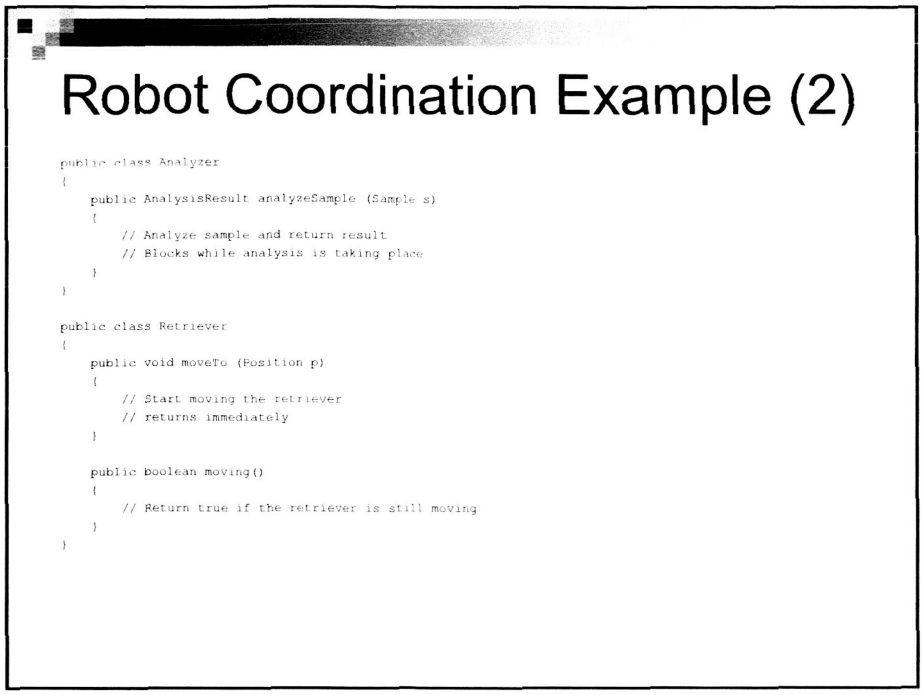
- Two robots, one stationary with a sensor package, one mobile with an arm to pick objects up

- The programming problem is to write code to make the mobile robot go pick things up and bring them back to the stationary robot to examine with the sensor package



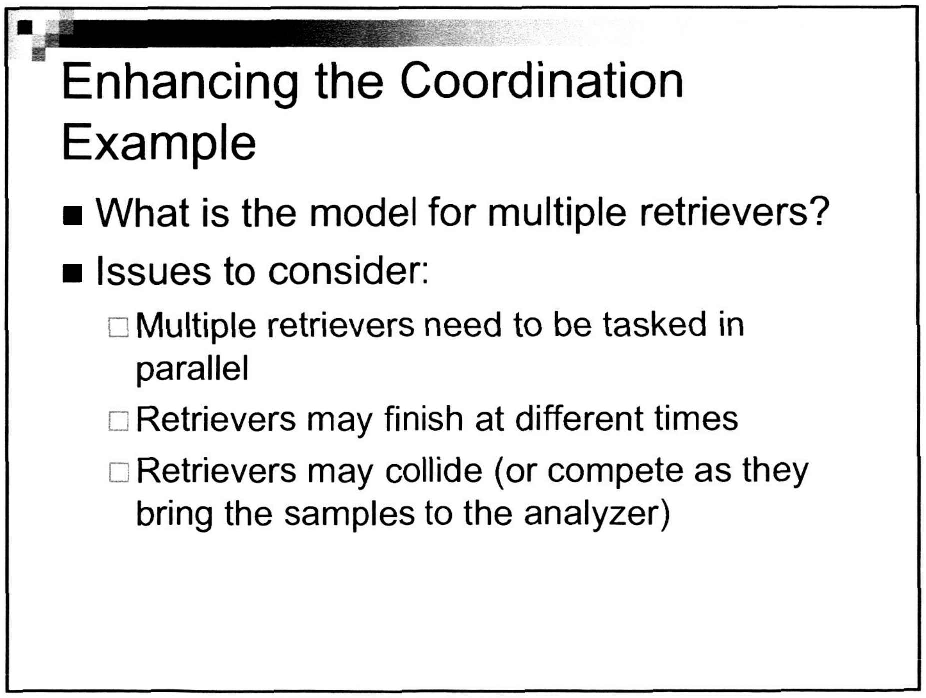
Robot Coordination Example

```
// Explorer with two bot parts - a moving sample retriever and a stationary sample analyzer
public class Explorer
{
    private Vector _sampleSites;
    private Vector _sampleResults;
    private Analyzer _analyzer;
    private Retriever _retriever;

    public void run()
    {
        Enumeration e = _sampleSites.elements();
        while (e.hasMoreElements()) {
            // Move retriever to position
            Position p = (Position) e.nextElement();
            _retriever.moveTo (p);
            while (_retriever.moving()) {
            }
            // Pick up sample
            _retriever.pickupSample();

            // Move retriever to analyzer position
            _retriever.moveTo (X0, Y0);
            while (_retriever.moving()) {
            }
            // Analyze sample
            _sampleResults.addElement (_analyzer.analyzeSample());
        }
    }
}
```



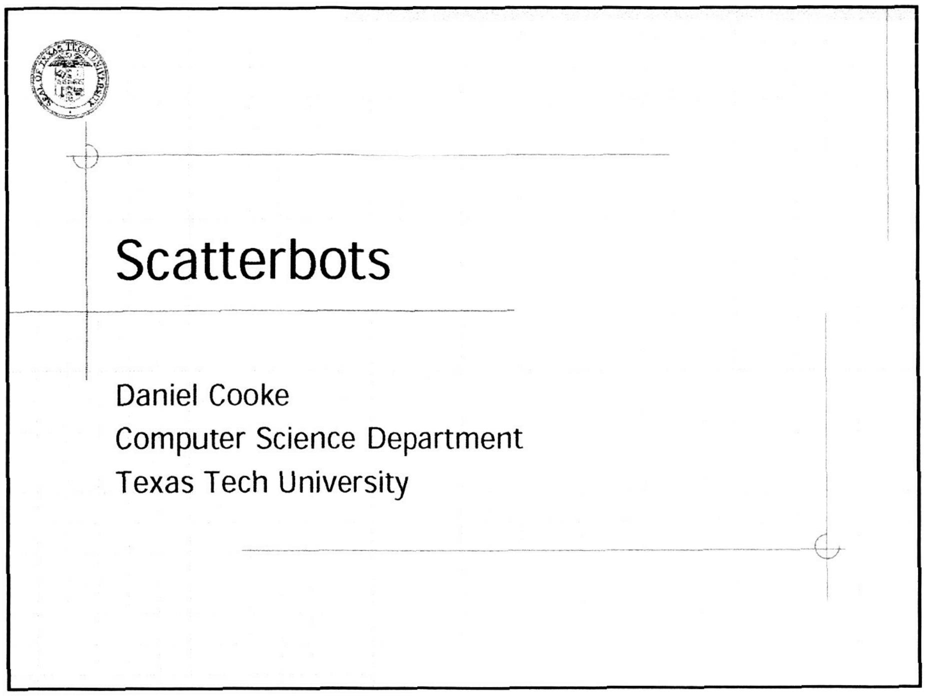
Robot Coordination Example (2)

```
public class Analyzer
{
    public AnalysisResult analyzeSample (Sample s)
    {
        // Analyze sample and return result
        // Blocks while analysis is taking place
    }
}

public class Retriever
{
    public void moveTo (Position p)
    {
        // Start moving the retriever
        // returns immediately
    }

    public boolean moving()
    {
        // Return true if the retriever is still moving
    }
}
```



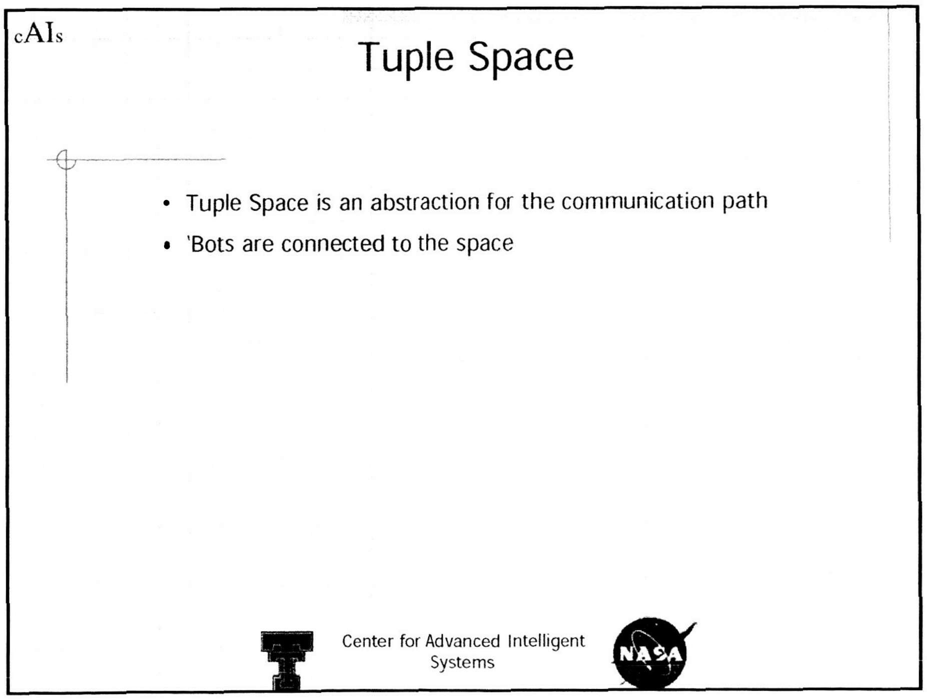
Enhancing the Coordination Example

- What is the model for multiple retrievers?
- Issues to consider:
  - Multiple retrievers need to be tasked in parallel
  - Retrievers may finish at different times
  - Retrievers may collide (or compete as they bring the samples to the analyzer)



Scatterbots

Daniel Cooke
Computer Science Department
Texas Tech University

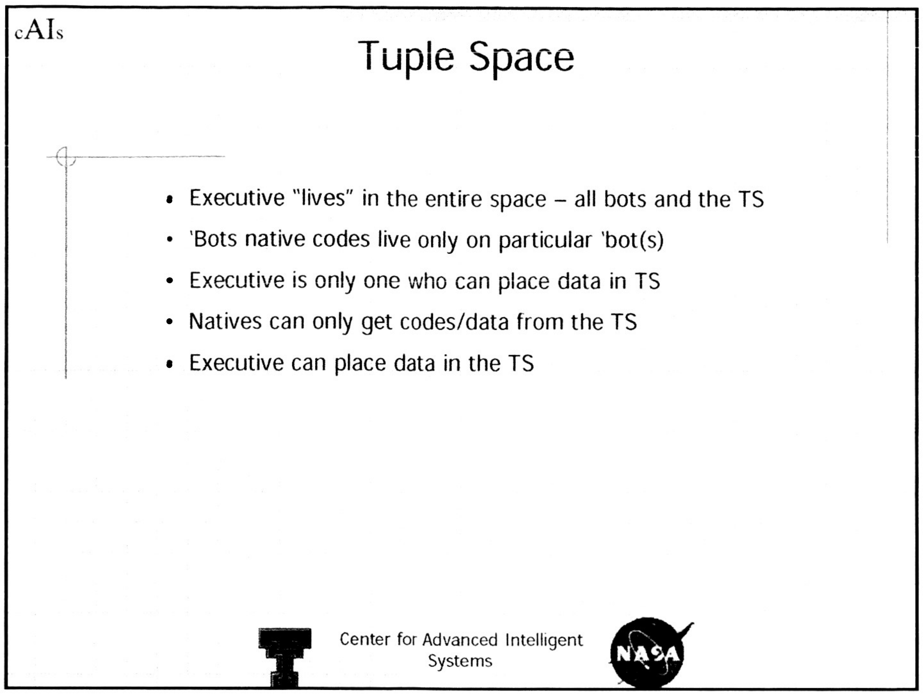


cAIs

Tuple Space

- Tuple Space is an abstraction for the communication path
- 'Bots are connected to the space

Center for Advanced Intelligent Systems

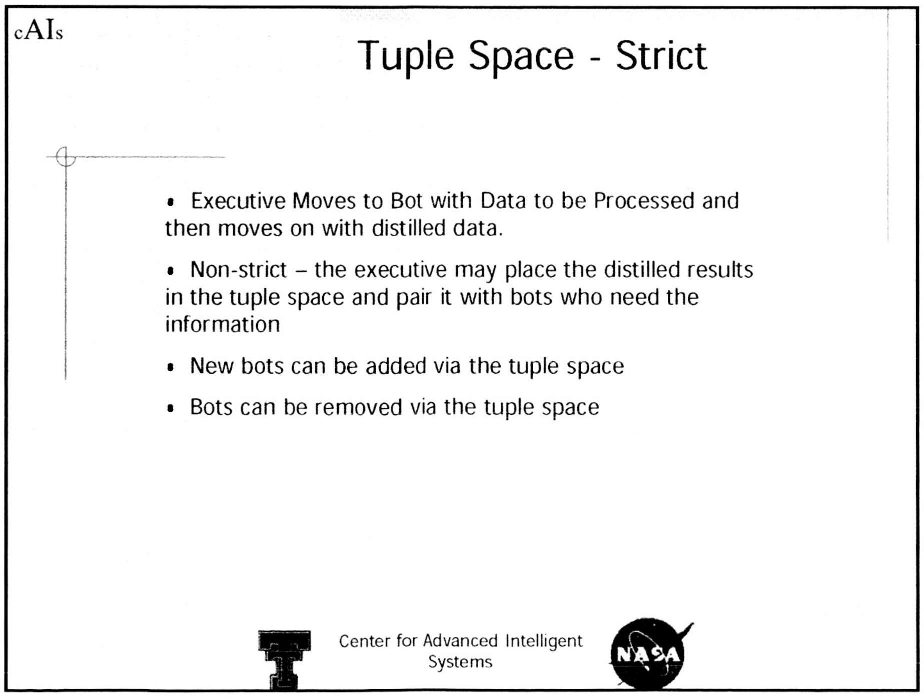


cAIs

Tuple Space

- Executive "lives" in the entire space – all bots and the TS
- 'Bots native codes live only on particular 'bot(s)
- Executive is only one who can place data in TS
- Natives can only get codes/data from the TS
- Executive can place data in the TS

Center for Advanced Intelligent Systems



cAIs

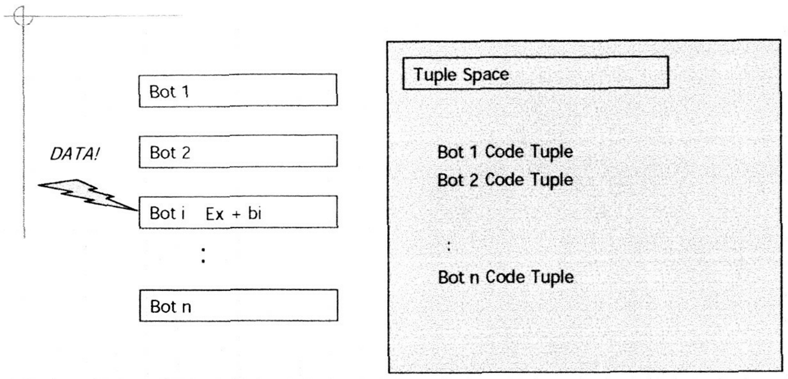
Tuple Space - Strict

- Executive Moves to Bot with Data to be Processed and then moves on with distilled data.
- Non-strict – the executive may place the distilled results in the tuple space and pair it with bots who need the information
- New bots can be added via the tuple space
- Bots can be removed via the tuple space

Center for Advanced Intelligent Systems

cAIs

Tuple Space - Strict

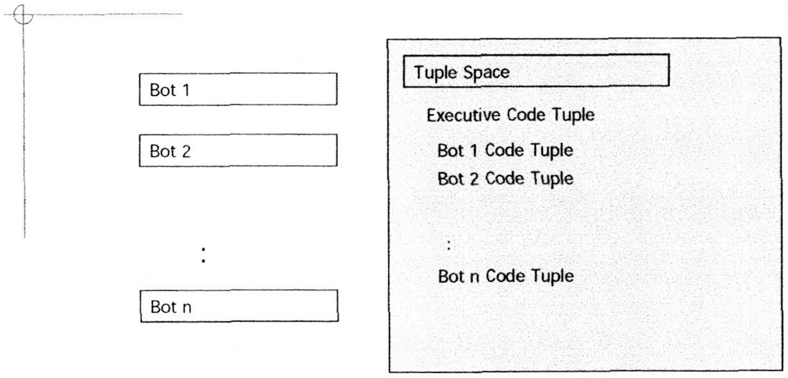


When a Bot(i) has important data the executive (Ex) moves with appropriate bot (bi) code to process data – minimizing movement of data

Center for Advanced Intelligent Systems

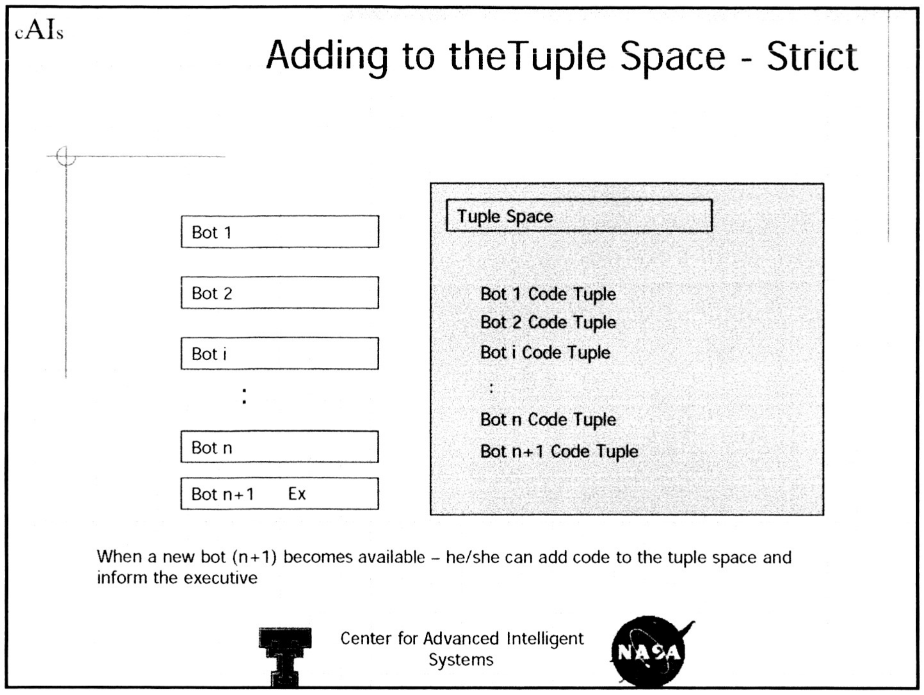
cAIs

Tuple Space - Strict



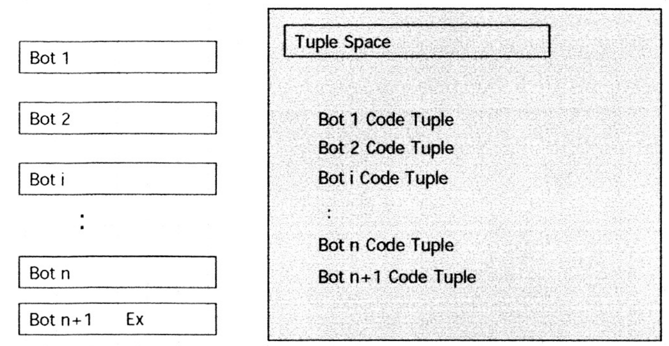
When a Bot(i) is no longer needed, he/she can inform the executive and remove code from TS

Center for Advanced Intelligent Systems



cAIs

Adding to theTuple Space - Strict



When a new bot (n+1) becomes available – he/she can add code to the tuple space and inform the executive

Center for Advanced Intelligent Systems