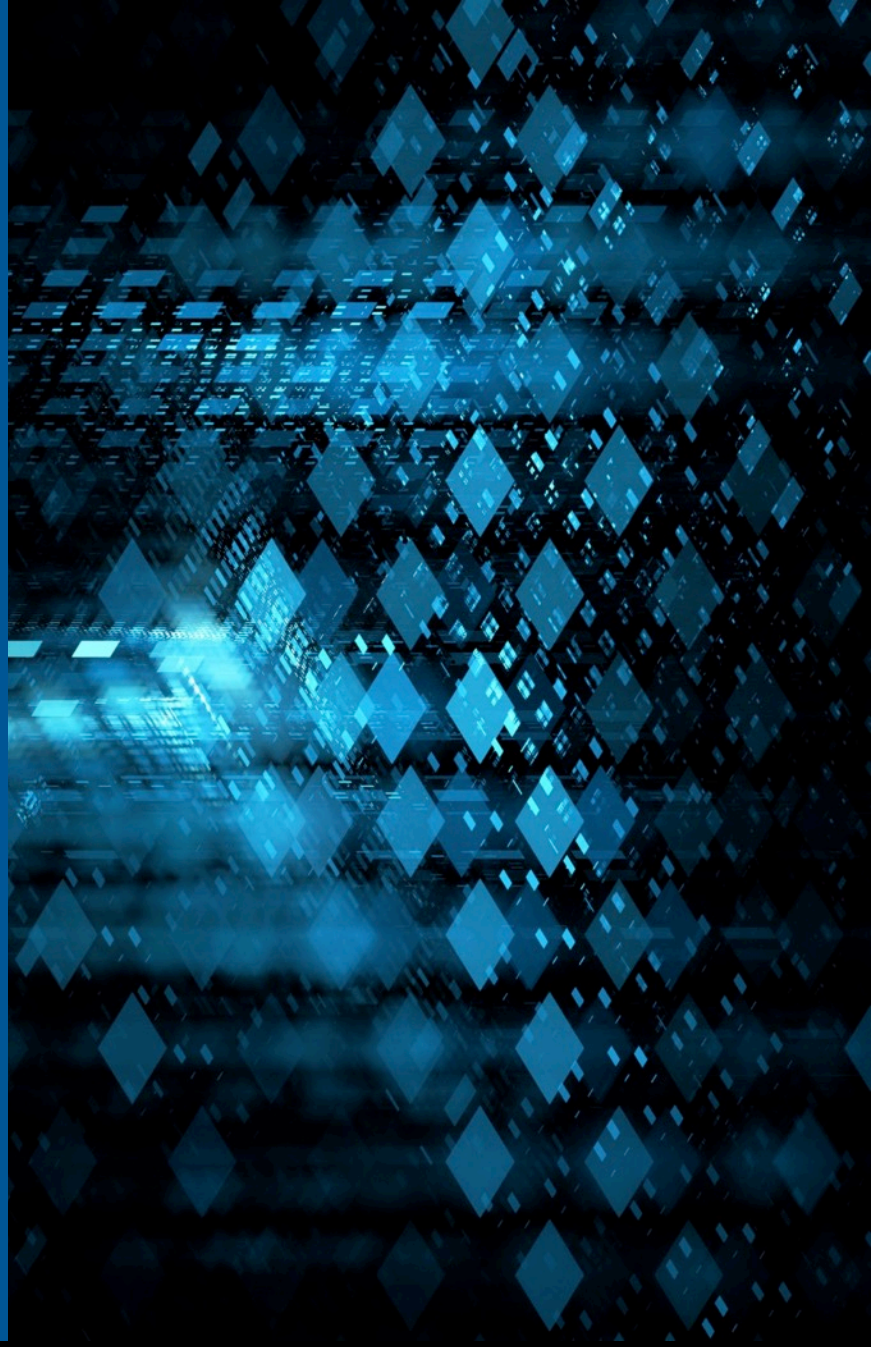
Introduction to Autonomy Software

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213

Introduction to MADARA and GAMS

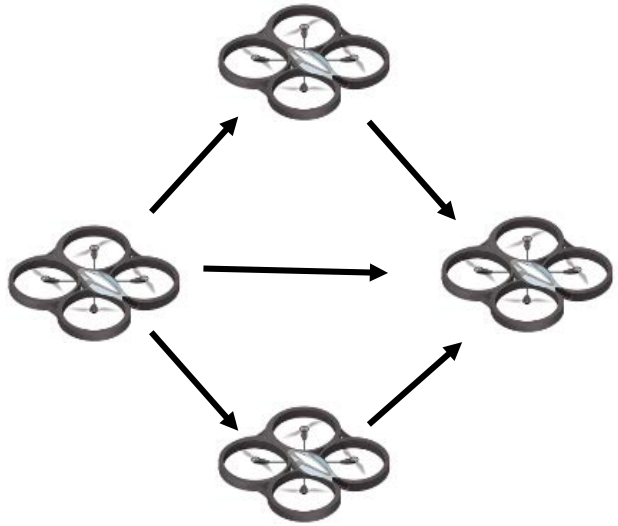
Module 1

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213



This material is based upon work funded and supported by the Department of Defense under Contract No. FA8702-15-D-0002 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center.

DM17-0380

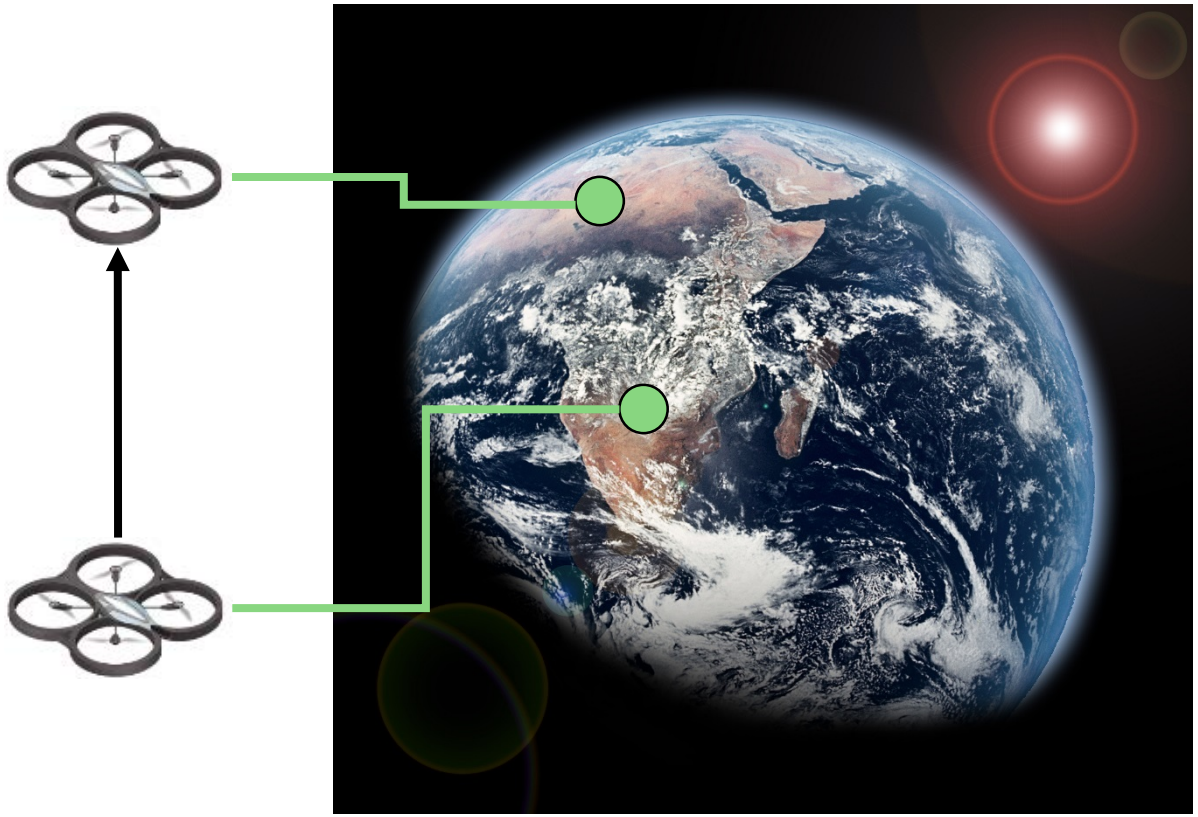


- In this talk, **we will discuss** the capabilities, history and focus of the **SEI-developed, open-sourced** Multi-Agent Distributed Adaptive Resource Allocation (**MADARA**) **and** Group Autonomy for Mobile Systems (**GAMS**) open source projects

- **MADARA** and **GAMS** have been consistent technologies used by the CPS-ULS group to **provide predictable execution and knowledge sharing** between a **group of distributed autonomous systems**

Understanding MADARA

- MADARA provides a thread-safe **world model** that is **automatically updated** to and from other autonomous agents **in your network** when you make changes to C++ or Java variables
- Information like robot location, orientation, etc.



MADARA First Started Development in 2009

- Initially developed to support portable **knowledge and reasoning**, **boolean logic** and **reinforcement learning** for a pub-sub-based (DDS) distributed artificial intelligence

- Over the years, it has **grown to** include **support** for
  - **Generic distributed systems**
  - **Threading**
  - **Advanced quality-of-service**
  - **UDP-based transports** (unicast, broadcast, multicast)
  - Ports to **Java, Android**, basic **python support**, etc.

- **Focus of development** since inception
  - **Quality-of-service** with especial focus on **timing and control**
  - **Scalability, speed, and expressiveness**
  - **Documentation (**Wikis and doxygen documentation for **all library calls)**
  - **Open-source release** (BSD-licensed, no cost, freely available)
  - **Extensibility** for **threading, transports** and **programming models**
    (e.g., iterative versus reactive)

MADARA is the Larger of the Two Projects and has been Developed for Longer

http://cloc.sourceforge.net v 1.64 T=2.93 s (279.2 files/s, 63610.9 lines/s)

Language	files	blank	comment	code
C++	370	20143	12781	81409
C/C++ Header	300	7444	25604	18126
Java	66	1542	6430	6410
HTML	15	245	174	1537
Python	8	498	648	1121
XML	41	0	4	1119
MSBuild script	1	0	0	416
Bourne Shell	2	61	32	208
Windows Resource File	1	24	33	69
IDL	3	5	0	50
C	1	28	14	40
Ant	1	6	2	28
Perl	2	10	13	25
CSS	1	8	7	14
Windows Module Definition	1	1	1	5
DOS Batch	5	0	0	5
SUM:	818	30015	45743	110582

Understanding GAMS

- GAMS provides **portable logic and algorithms** that can be used on any supported GAMS platform
- GAMS is an **interface for portable AI**

GAMS First Started Development in 2012 and Built on Top of MADARA

- Initially started as SMASH, a project at SEI **intended to support distributed algorithms for quadcopters in outdoor environments**
- Over the years, it has **grown to** include **support** for
  - **Extensible algorithms**
  - **Extensible platforms**
  - **Advanced quality-of-service**
  - **Generic simulation environment support**
  - **Multi-step and dynamic algorithms** (algorithms that execute other algorithms)
  - Ports to **Java and Android**
- **Focus of development** since inception
  - **Quality-of-service** with especial focus on **timing and control**
  - **Scalability, speed, and expressiveness**
  - **Open-source release** (BSD-licensed, no cost, freely available)
  - **Documentation (**Wikis and doxygen documentation for **all library calls)**
  - **Extensibility** for **new platforms and algorithms**

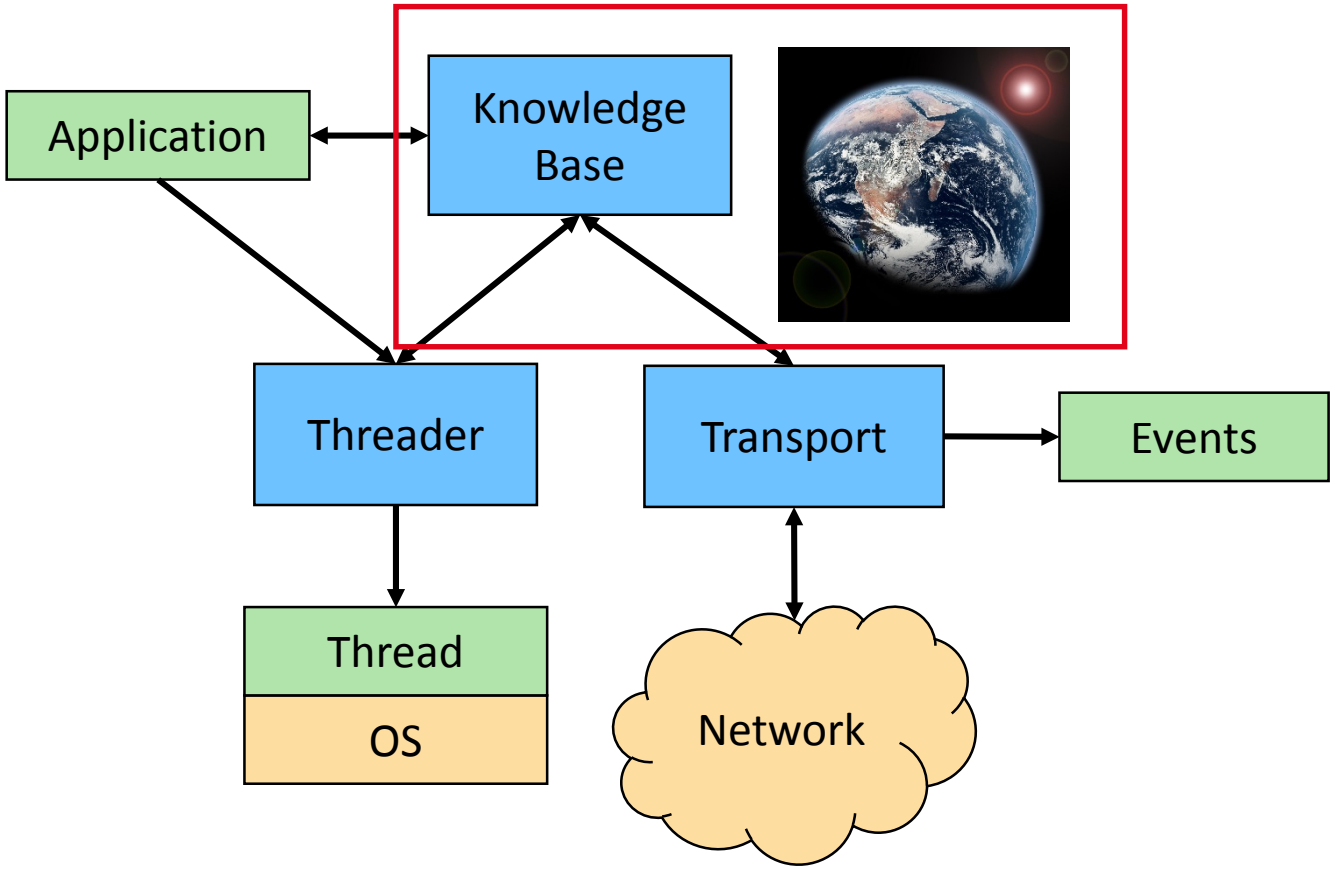
GAMS is the Smaller of the Two Projects and has been Developed for a Few Years

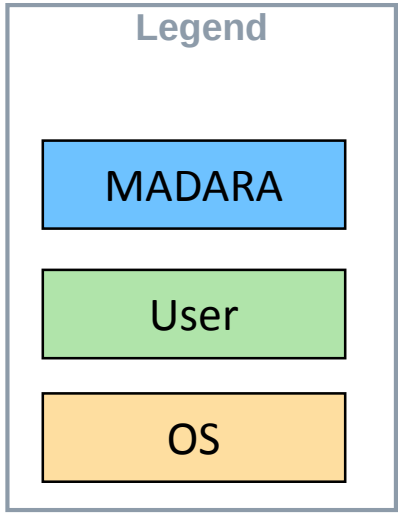
http://cloc.sourceforge.net v 1.64 T=2.00 s (387.8 files/s, 49611.3 lines/s)

Language	files	blank	comment	code
C++	132	4253	6966	21548
XML	383	1142	2600	12607
Perl	41	1430	422	7692
C/C++ Header	130	2806	14095	6627
Java	49	1159	7851	5927
Bourne Shell	21	166	218	841
DOS Batch	4	83	2	272
Bourne Again Shell	1	20	21	123
Groovy	4	10	3	68
YAML	5	10	20	39
make	2	13	13	30
Prolog	1	2	0	15
CMake	1	7	20	12
Lua	1	0	1	2
SUM:	775	11101	32232	55803

MADARA Overview

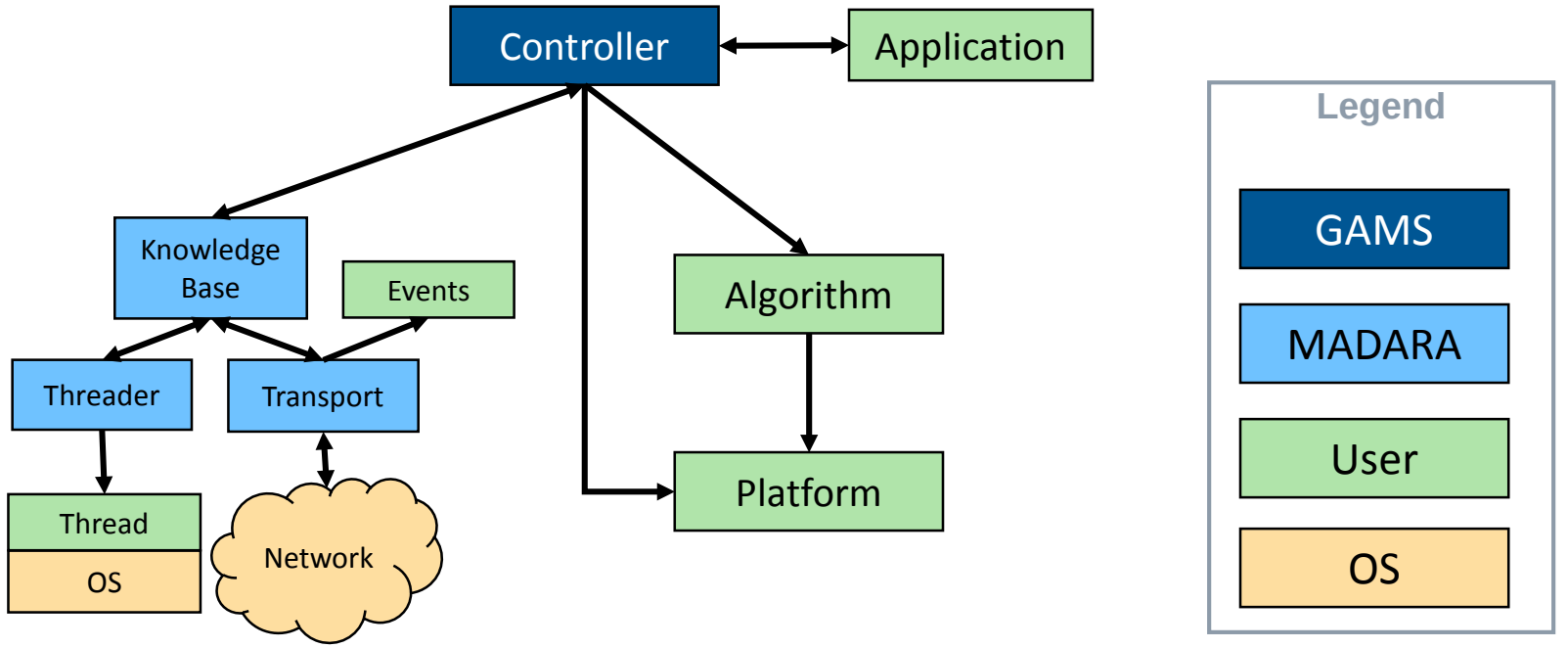
- **User applications interact with** a **Knowledge Base**
- The **Transport** layer transfers **knowledge across a network**
- **The Threader controls portable, knowledge-enabled threads**





GAMS Overview

- **GAMS** is **built on** top of **MADARA**, so it inherits all of its features
- **User application** features an **autonomous system controller**
- **Controller executes a distributed algorithm in a MAPE Loop**
- **Algorithm interacts with a hardware or simulation platform**



Platforms and Uses We have been Targeting for GAMS

- Distributed algorithms and platforms for **quadcopters in outdoor environments (**SMASH **2013,** GAMS **2014)**

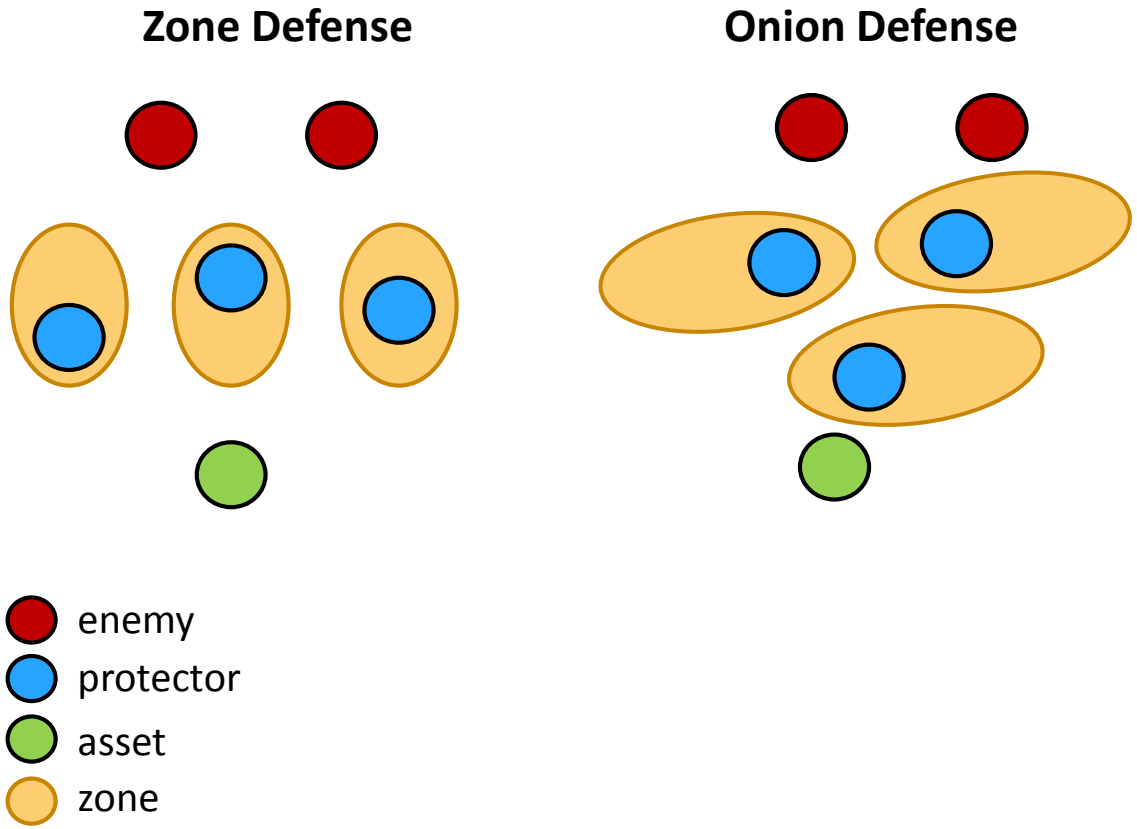
- **Scalable and adaptive algorithms** and platforms for **Platypus Lutra boats in outdoor environments (**ELASTIC **2015)**

ARM processor performance (lower powered)

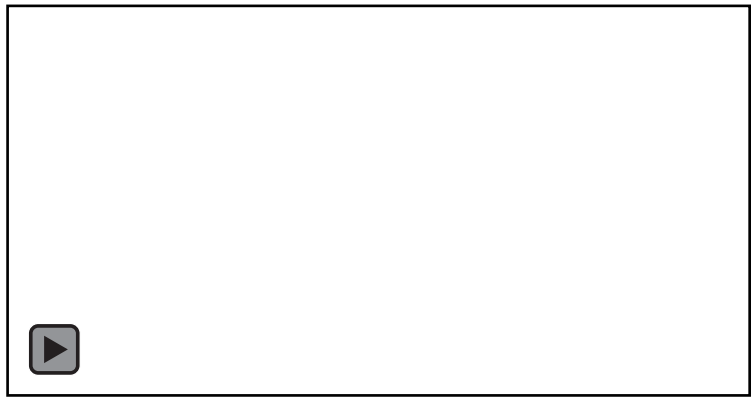
	Min Hz	Max Hz	Avg Hz
Per Node Publish	500.00	517.30	507.46
Per Node Receive	432.00	490.00	468.12
Per Node Total Receive	8,820.00	9,082.00	**8,921.38**
Total Throughput			133,868.00
Platypus Node Publish	5.00	35.00	31.00
Platypus Node Receive	5.00	35.00	31.00
Platypus Per Node Total Receive	100.00	700.00	**620.00**
Platypus Total Throughput			12,400.00
Estimated boat scaling	1764	240	**280**

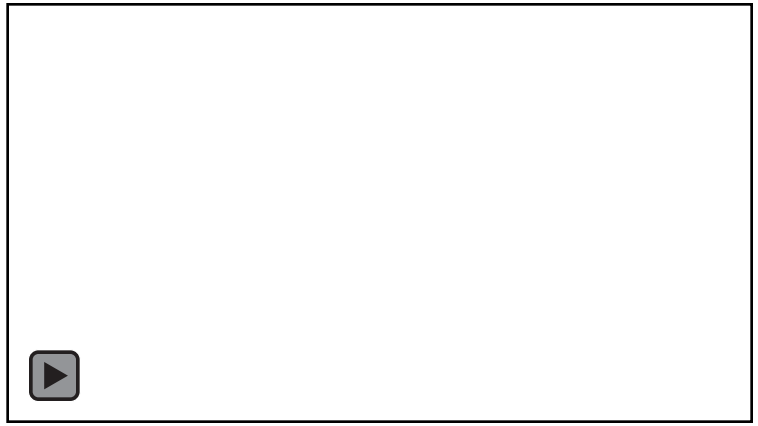
Platforms and Uses We have been Targeting for GAMS

Adversarial algorithms for Platypus Lutra boats in **outdoor environments (**MADPARTS **2016)**



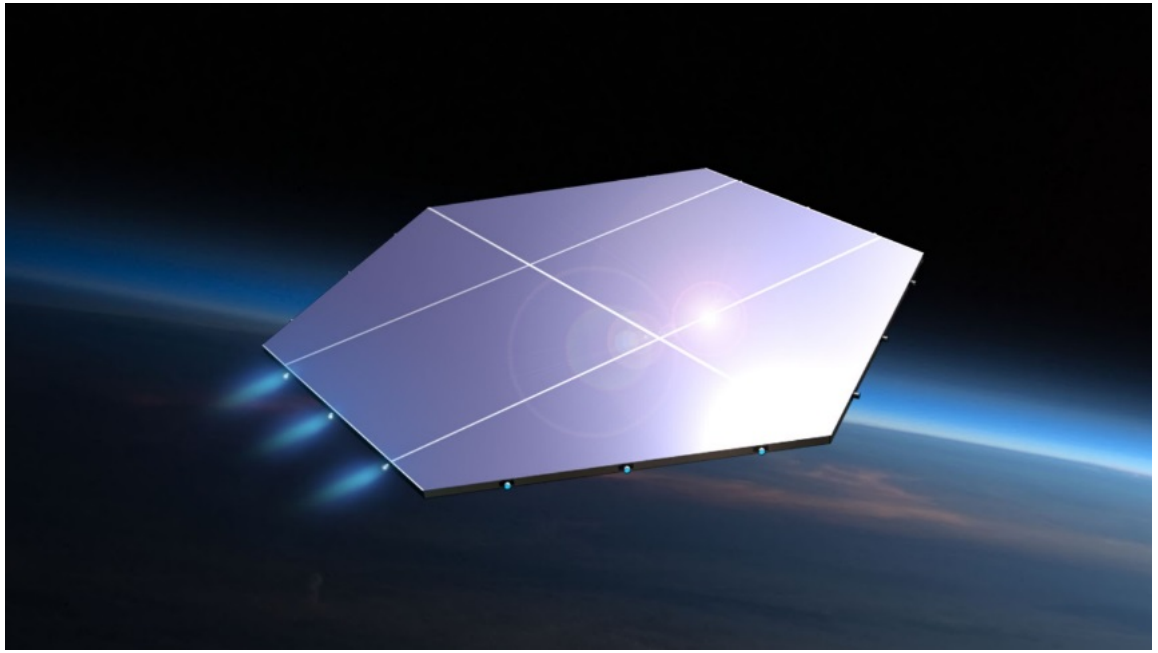
Simulation infrastructure for verification techniques like **DART** and **DEMETER**

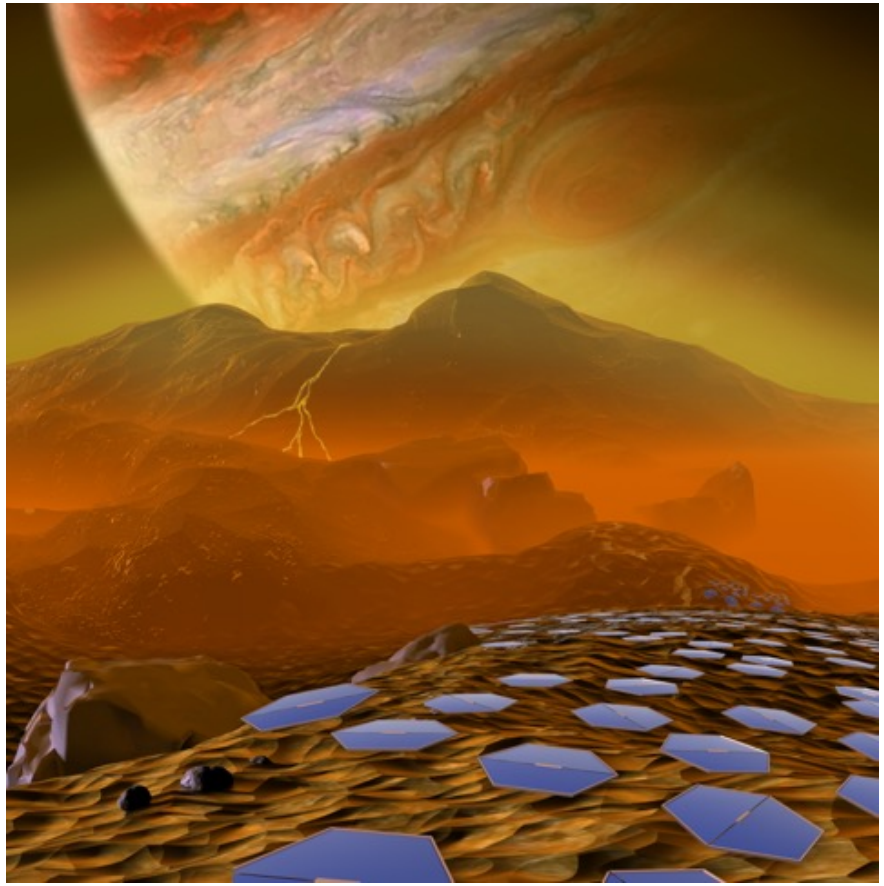




Platforms and Uses We have been Targeting for GAMS

Simulation and real-world infrastructure for platforms and algorithms for **Multi-Planetary Smart Tile (2016-future), currently in development at GE GRC**





What Else Could I Use Instead of GAMS and MADARA?

- The major industry player in this space is ROS (Robot Operating System)
- Pros of ROS:
  - **Also open source (http://www.ros.org/)**
  - **Commercial support**
  - **Active development community**
  - **Lots of available tools**
- Cons of ROS:
  - **Forces message passing as main programming interface (less intuitive)**
  - **Does not scale well in processing hertz or number of agents**
  - **Synchronous interactions that can block indefinitely**
  - **Requires reliable network transport that may fail spectacularly when networking is unreliable or offline for long periods of time (natural environment problem or adversarial jamming)**
  - **Lack of verification tools**
    - Not designed for verification
- We will be using GAMS and MADARA in this tutorial series
- Most high level autonomy programming concepts translate to programming any autonomous system (including ROS)

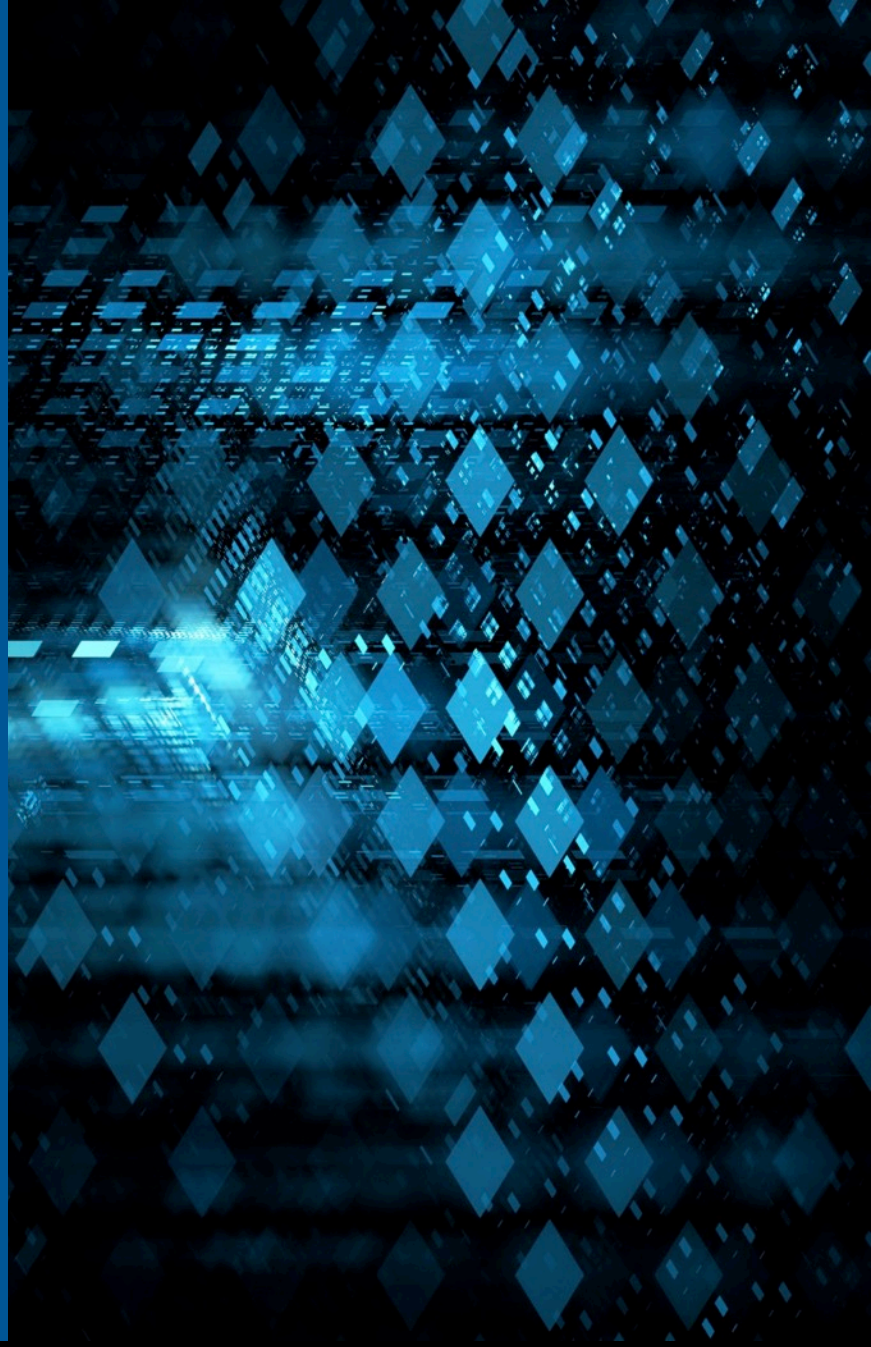
Next Steps in this Tutorial Series

- Installing MADARA and GAMS
- Using the GAMS project configuration tool
- Creating distributed algorithms

Installation of MADARA and GAMS

Module 2

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213



This material is based upon work funded and supported by the Department of Defense under Contract No. FA8702-15-D-0002 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center.

DM17-0380

In this talk, **we will discuss** the installation of the **SEI-developed, open-sourced** Multi-Agent Distributed Adaptive Resource Allocation (**MADARA**) **and** Group Autonomy for Mobile Systems (**GAMS**) open source projects

- To install MADARA and GAMS, you will need:

- **Linux users**
  - A Virtual Machine or Host OS with administrative privileges (sudo access)
  - A functioning internet connection
  - We will be installing perl, build-essential, and a few other prerequisites

Installation for Linux Users

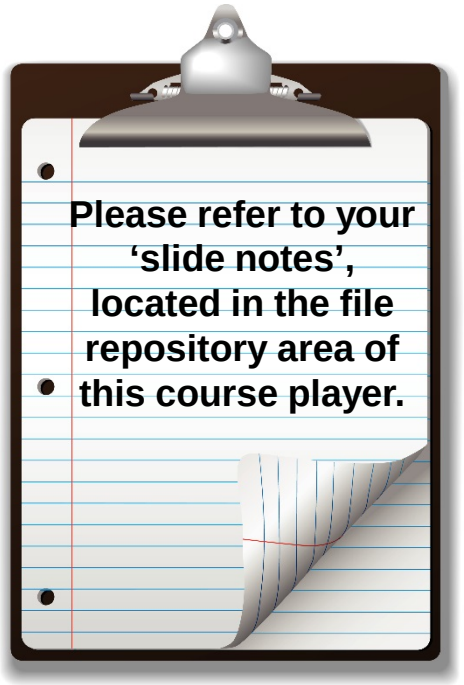
1. We need to **open a terminal** and install git if it hasn't been installed
2. We need to figure out **where** we want **GAMS to be installed**
3. We need to **download the GAMS repo** to gain access to the build scripts
4. We need to **run the build script**

Installation for Linux Users

Linux Install (should be easy, open a terminal)

```
sudo apt-get install git-core
export GAMS_ROOT=<directory to install to>
export CORES=<number of CPU cores you have>
git clone -b master --single-branch https://github.com/jredmondson/gams $GAMS_ROOT
$GAMS_ROOT/scripts/linux/build_c++.sh prereqs tests
```

It is **highly recommended** that you **export** the **variables** mentioned by the installer **in your .bashrc file** to have them loaded for each new terminal session

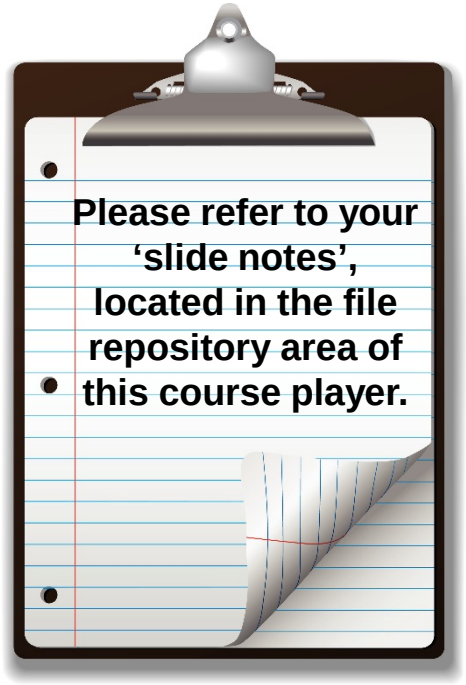


Updating the repo for Linux Users

1. We need to **open a terminal**
2. We need to **run the build script**

Linux update (should be easy, open a terminal)

```
$GAMS_ROOT/scripts/linux/build_c++.sh tests
```

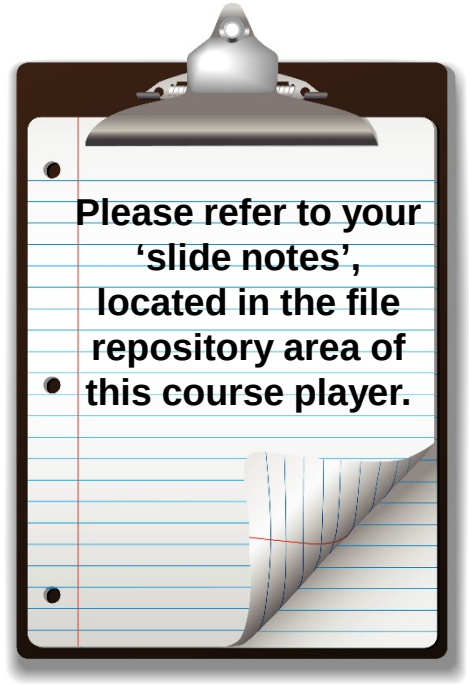


Updating the repo for Linux Users

Configuring non-default modules

1. Build_c++ is a convenience script for arguments to base_build.sh
2. We simply pass in the options that make sense for the options we want
3. **Most options are completely optional**

```
$GAMS_ROOT/scripts/linux/base_build.sh madara gams tests vrep
```



It’s a Good Idea to Update your .bashrc

In that same terminal, open your ~/.bashrc file and add the exports that are listed after your GAMS and MADARA installation is completed (example below):

- Modifying your .bashrc will make these environment variables available to you and also make GAMS and MADARA dynamic libraries available to you from any bash terminal
- You may want to do the exact same thing in .profile for non-bash terminals and certain development IDEs (e.g., Netbeans)

It’s a Good Idea to Test your Installation

Open terminal

```
$MADARA_ROOT/bin/test_reasoning_throughput
```

You should see performance information from MADARA reasoning operations

Next Steps in this Tutorial Series

- Using the GAMS project configuration tool
- Creating distributed algorithms

Creating Distributed Autonomy Projects

Module 3

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213

This material is based upon work funded and supported by the Department of Defense under Contract No. FA8702-15-D-0002 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center.

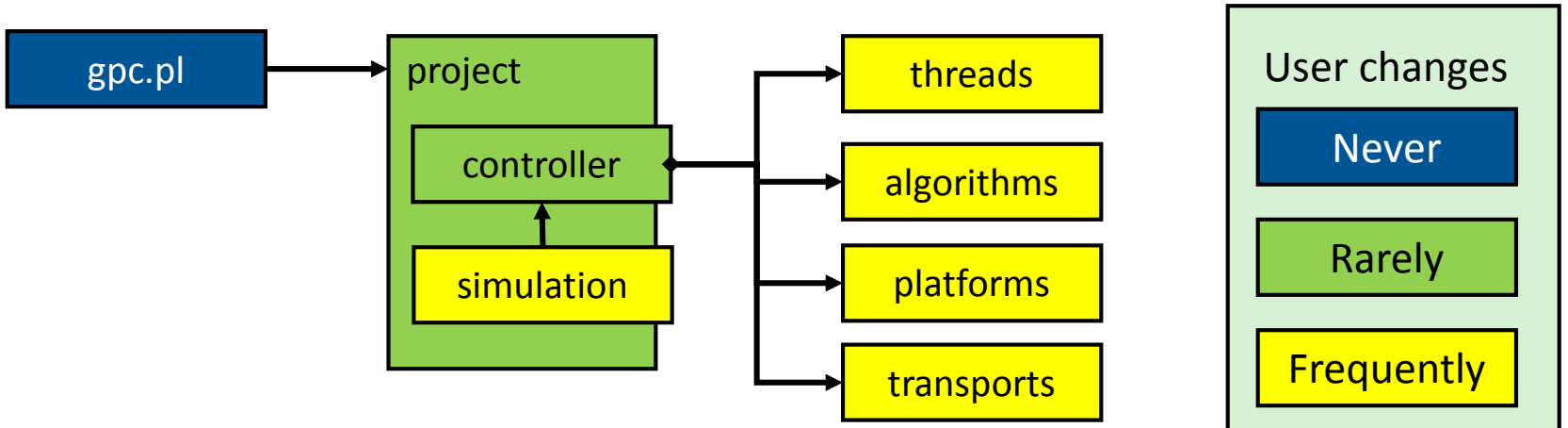
DM17-0380

In this talk, **we will discuss** the usage of the GAMS Project Configurator (gpc.pl) to create and manage portable GAMS and MADARA projects

- This talk covers what to do after GAMS and MADARA are installed in order to create your own GAMS-enabled projects

- In this talk, we will learn
  - **How to create an initial project**
  - **How to compile a GAMS project**
  - **How to generate simulations for quadcopters, boats, and other platforms in VREP**
  - **How to generate custom threading, algorithm, platform and networking transports**

Understanding the GAMS Project Configurator (gpc.pl) Process



- A user generates a project which may contain a controller and simulation
- The gpc can tweak a controller to include custom threads, algorithms, platforms and transports

Generating a New GAMS Project

Linux terminal

```
$GAMS_ROOT/scripts/projects/gpc.pl --path $PROJECT_ROOT/tutorial1
```

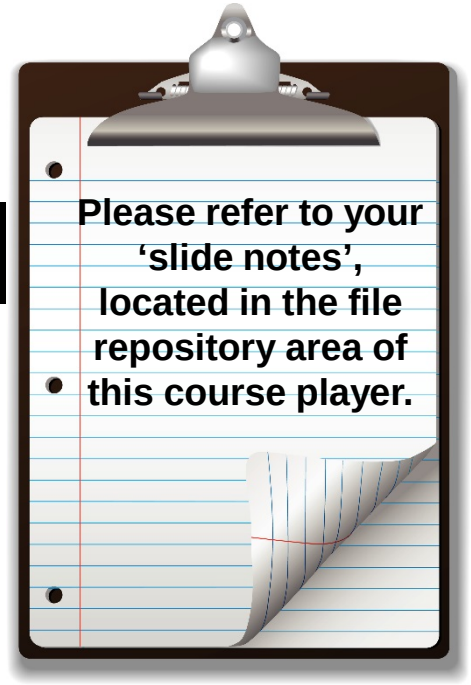
- PROJECT_ROOT should be set to where you want your projects

- Providing only path information is the bare minimum project creation
  - bin directory for a future controller
  - sim directory with a basic sim
  - src directory with folders for algorithms, platforms, threads and transports

Running the Basic Sim

Linux terminal

```
$PROJECT_ROOT/tutorial1/action.sh vrep sim
```

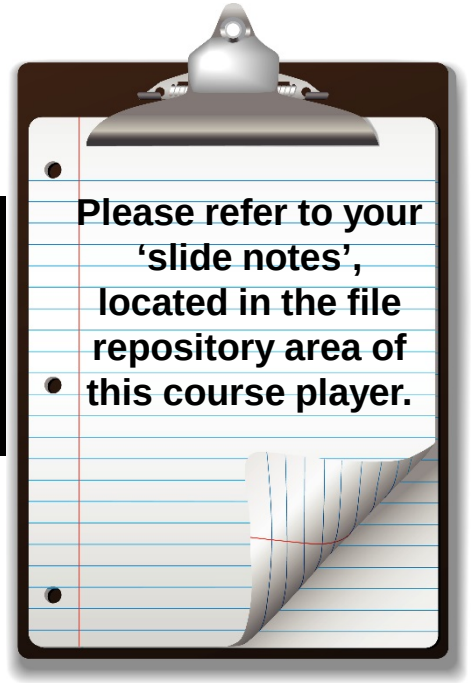


- The sim will just have one stationary quadcopter at the center of the map
- You should see lots of logging in the output
- A rotated log file will appear in the sim directory
- On Linux, you should be able to Ctrl+C to exit VREP

Change the Simulation to have Five Random Agents and Minimal Logging

Linux terminal

```
cd $PROJECT_ROOT/tutorial1
$GAMS_ROOT/scripts/projects/gpc.pl --madara-debug 1 --gams-debug 1
$GAMS_ROOT/scripts/projects/gpc.pl --agents 5 --randomize
./action.sh vrep sim
```

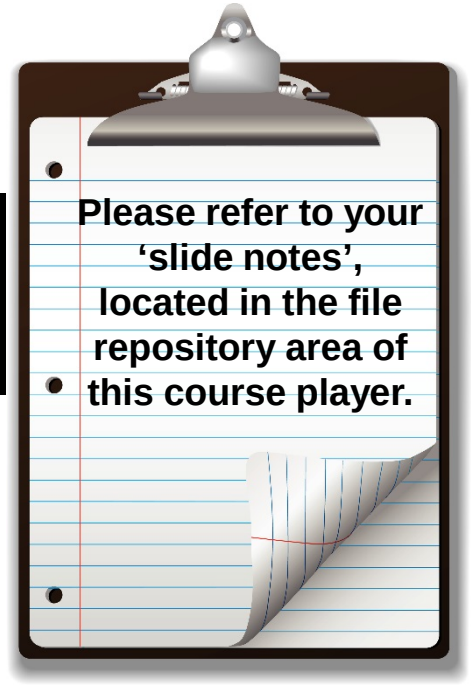


- You should see no logging (only errors or "always" level logging messages)
- The old log will be saved to agent_
- The sim will have five randomly placed quadcopters
- The quadcopters communicate together using multicast (default)

Change the Simulation to have Agents Distributed Evenly

Linux terminal

```
cd $PROJECT_ROOT/tutorial1
$GAMS_ROOT/scripts/projects/gpc.pl --distributed
./action.sh vrep sim
```

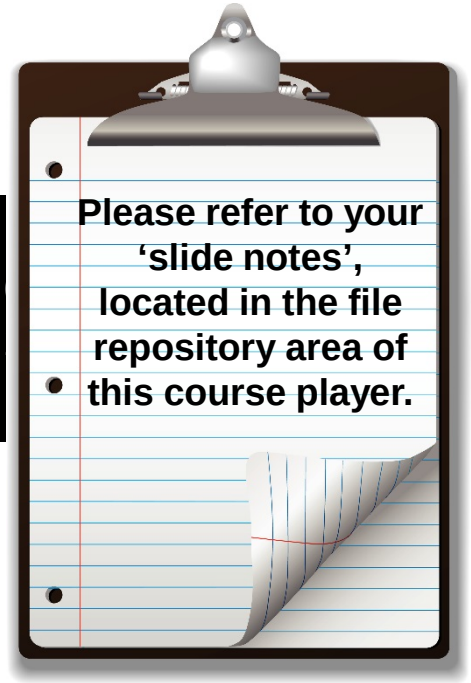


- The sim will have 5 quadcopters distributed across the simulator map
- You can use the --rotate

Change the Simulation to Include Two Groups of Agents

Linux terminal

```
cd $PROJECT_ROOT/tutorial1
$GAMS_ROOT/scripts/projects/gpc.pl --group group.attackers --first
$GAMS_ROOT/scripts/projects/gpc.pl --group group.defenders --first
./action.sh vrep sim
```



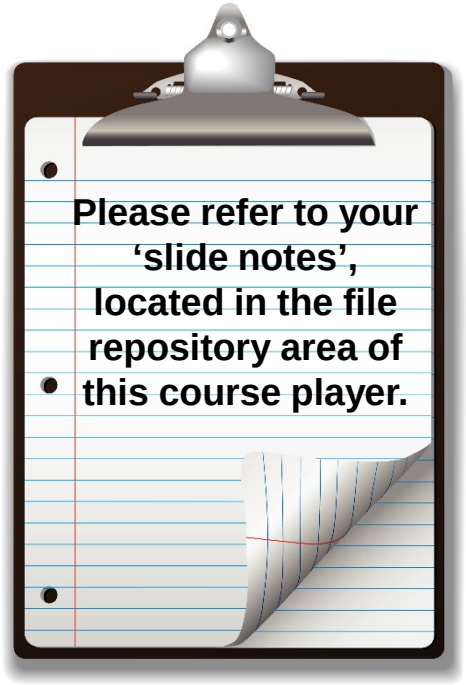
- These groups and their memberships will be available to any algorithms you design later
- Memberships are placed in the sim/common.mf file
- Groups can be fixed or dynamic and can change at runtime

Isolate your Simulation from Others

Linux terminal

```
cd $PROJECT_ROOT/tutorial1
$GAMS_ROOT/scripts/projects/gpc.pl --domain jamese
./action.sh vrep sim
```

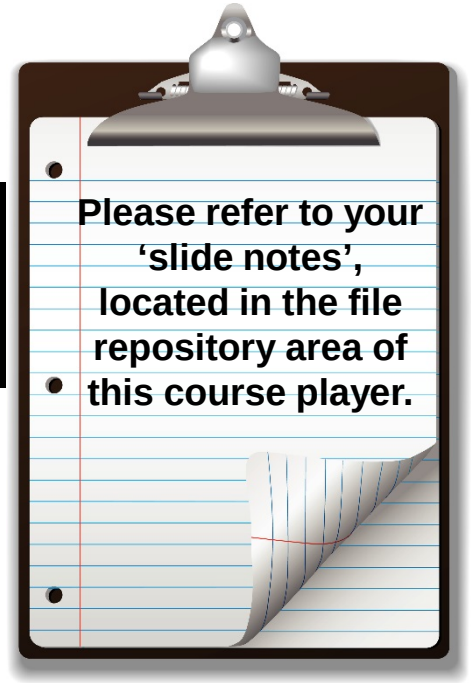
- Simulations default to using the same multicast ip
- A domain is a logical partition of a network transport
- Traffic from other simulations still arrive, but everyone else's knowledge is discarded



More Effectively Isolate your Simulation from Others

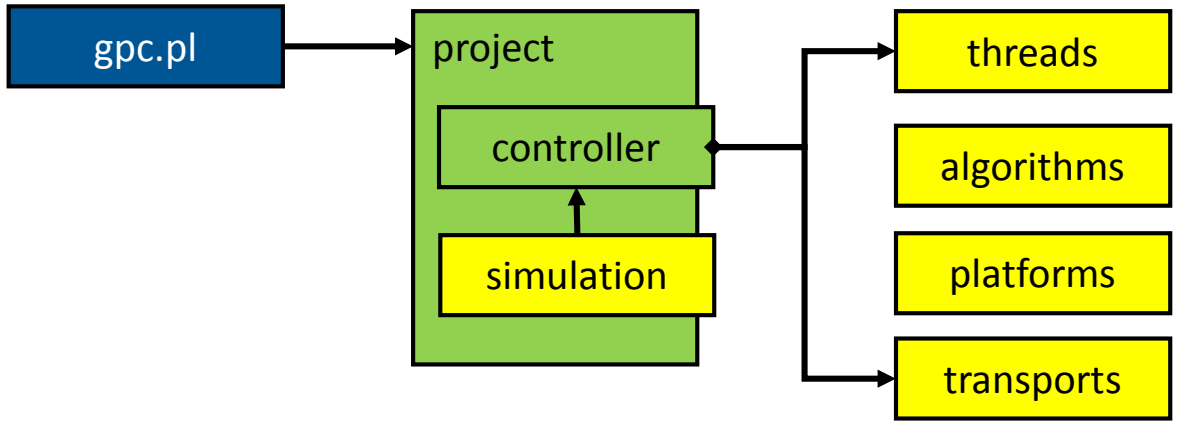
Linux terminal

```
cd $PROJECT_ROOT/tutorial1
$GAMS_ROOT/scripts/projects/gpc.pl --multicast "239.100.5.40:47000"
./action.sh vrep sim
```



- Simulations default to multicast endpoint "239.255.0.1:4150"
- Choose a random multicast ip to more effectively isolate yourself
- You can also switch to using --udp "localhost:47000" --udp "localhost:47001", etc. to keep all traffic on your local computer, using unicast sockets

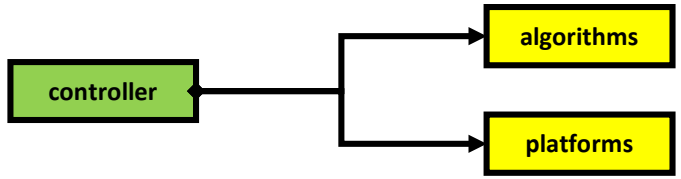
Understanding the Controller Extension Process



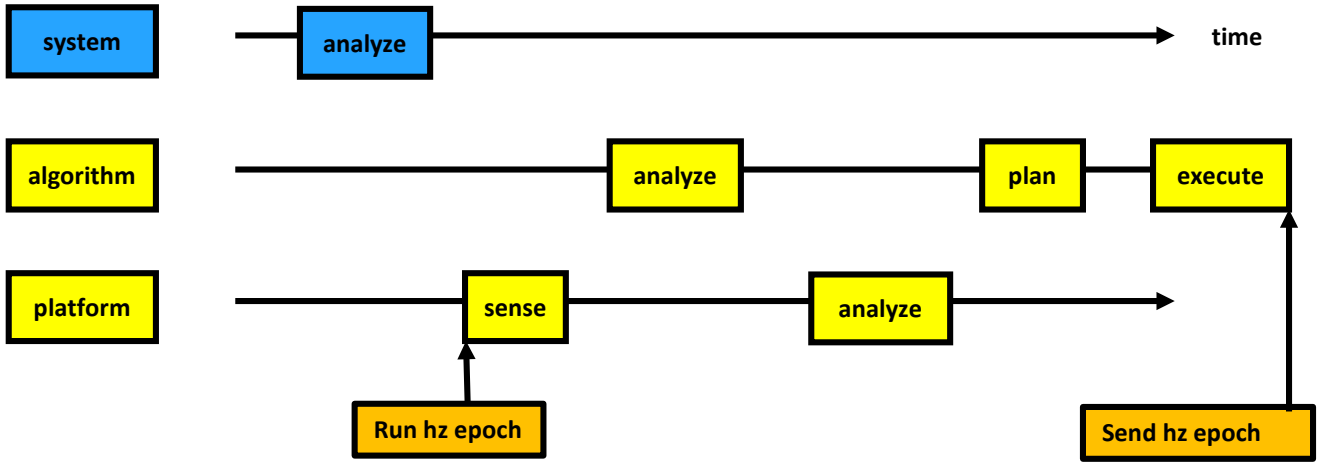
- **Controllers are regenerated whenever you create new threads, algorithms, platforms or transports**

- **Projects will also need to be regenerated whenever new files or libraries are added or removed**

Understanding how the Controller Interacts with Algorithms and Platforms



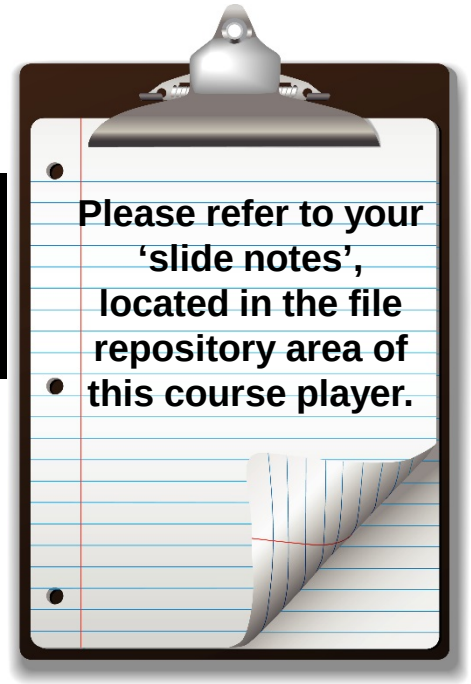
Controller executes algorithms and platform functions in a MAPE (Monitor Analyze Plan Execute**) process**



Creating a New Controller by Adding Threads and Other Components

Linux terminal

```
cd $PROJECT_ROOT/tutorial1
$GAMS_ROOT/scripts/projects/gpc.pl --new-thread sense --new-thread analyze
$GAMS_ROOT/scripts/projects/gpc.pl --new-algorithm intelligent_isr
```

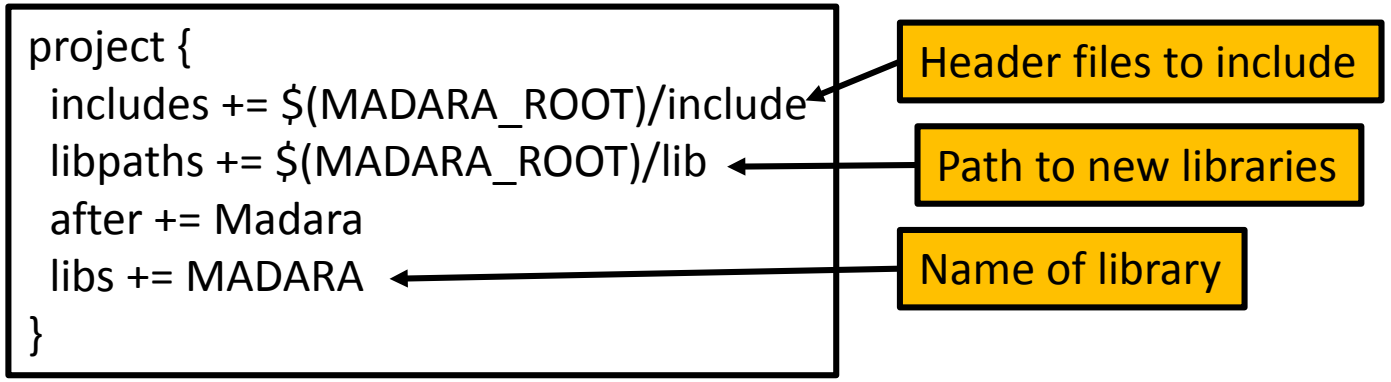


- We have created two new threads in src/threads called “sense” and “analyze”
- We have created one new algorithm called “intelligent_isr”
- These new generated threads and algorithms have null implementations that can be implemented however you like
- We’ll talk more about algorithms in the next tutorials

Linking Other Libraries into the Controller

- The action scripts are using the MakeFileProjectCreator open-source project underneath the hood to automate the project generation, compilation and linking of source files

- See the using_madara.mpb file for an example of how to link to external libraries



- Modifying this project file is usually portable to all other operating systems (Linux, Windows, MacOS, Android, etc.)

Modifying the Project File for Non-Generated Code Inclusion

The project.mpc file defines the controller compilation task

```
project (custom_controller) : using_gams, using_madara, using_ace, using_vrep {
  exeout = bin
  exename = custom_controller
  …
  Header_Files {
    src/
    src/algorithms
    src/platforms
    src/threads
    src/transports
  }

  Source_Files {
    src
    src/algorithms
    src/platforms
    src/threads
    src/transports
  }
}
```

Custom libraries

Header files, can include directories

Source files, can include directories

Compile the New Controller and Run the Sim

Linux terminal

```
cd $PROJECT_ROOT/tutorial1
./action.sh compile vrep sim
```

The action script automates the compilation, launch of VREP, and the start of the simulation for you

Other Compilation Notes

- The project file can be modified to produce libraries instead of or in addition to the controller
- See the documentation for the MakefileProjectCreator (https://www.openhub.net/p/mpc) to see other options for this project and workspace system
- Users also have the option of creating their own make systems, but care should be taken to include the ACE, MADARA, and GAMS libraries

Next Steps in this Tutorial Series

- **Creating Distributed Algorithms**

Creating Distributed Algorithms

Module 4

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213

This material is based upon work funded and supported by the Department of Defense under Contract No. FA8702-15-D-0002 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center.

DM17-0380

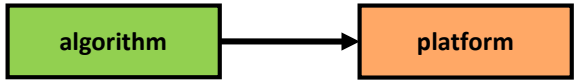
In this talk, **we will discuss** the creation of distributed algorithms

- In this talk, we will learn
  - **How to plan an algorithm**
  - **How to generate GAMS algorithms**
  - **Why GAMS algorithms are usually better than threads**
  - **How to control knowledge dissemination**
  - **How to change algorithms at runtime**

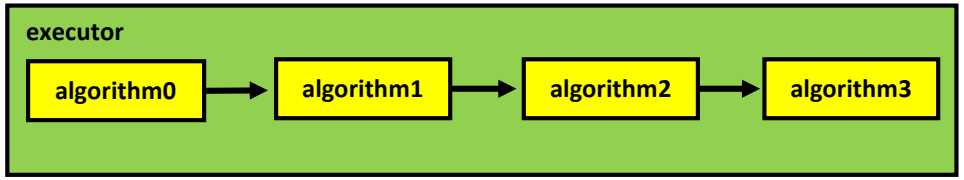
Understanding Algorithm Concepts

An **algorithm** is a **logic** that **governs behavior** for an autonomous system

- In **GAMS**, an **algorithm** is **also** a class that **controls a platform**



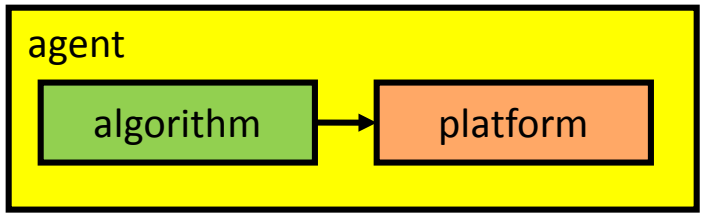
- **Algorithms can be composed of other algorithms**



- **Algorithms can** also **be orthogonal** and **unrelated** to each other
  - For instance, **a robotic arm** may be **perform**ing **behaviors** and actions **independ**ently of the **movement** of the vehicle
  - **We call** these **orthogonal algorithms "accents"** (accentuating algorithms)

Understanding Algorithm Concepts

- An **agent** is an **autonomous entity** that **contains** an **algorithm and platform**



- Some **agent algorithms are only interested in** their **own agency (**often called **“self-interested”)**
- However, **most** algorithms **interact** with **other agents** whether **friend or foe**
- **Multiple interacting agents** are often called **multi-agent systems**
- When discussing **multi-agent systems,** most researchers **assume** the **interaction is collaborative toward a common goal**

Understanding Algorithms in GAMS

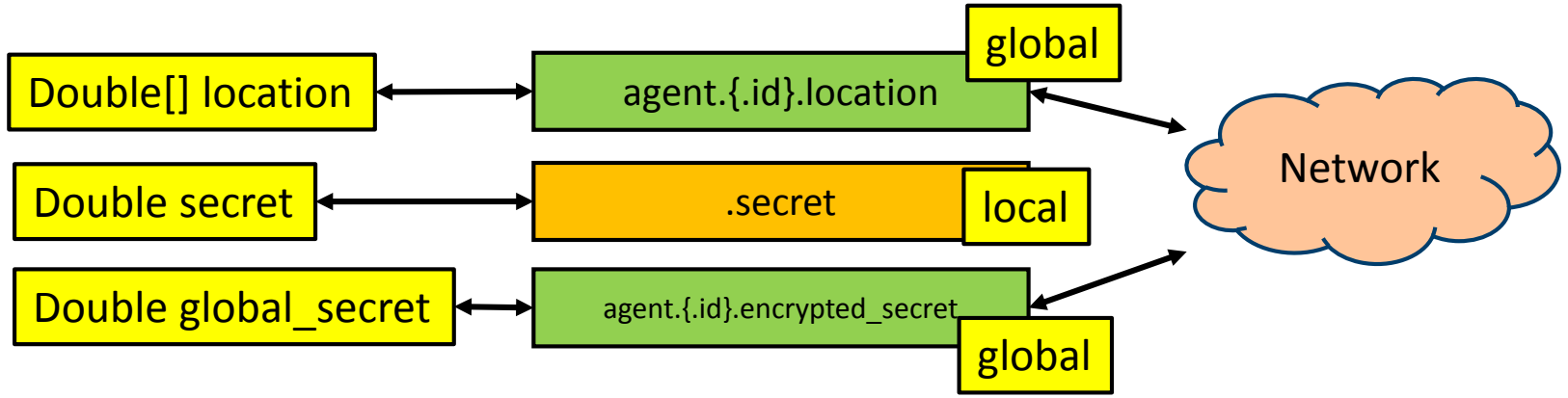
- **GAMS includes** over a dozen **algorithms** for **general usage**

Formation Coverage
Prioritized Region Coverage
Minimum Time Coverage
Serpentine Coverage
Waypoints
Formation Follow
Synchronized Formations
Convoy Shielding
Line Defense
Arc Defense
Onion Defense
Executor

- Collection has **both self-interested** algorithms **and multi-agent** algorithms
- **Algorithm definitions** contain both an **algorithm implementation** and a **factory method** for creating the algorithm
- **Factory methods** are **required so** an **algorithm** can be **changed/created at runtime**

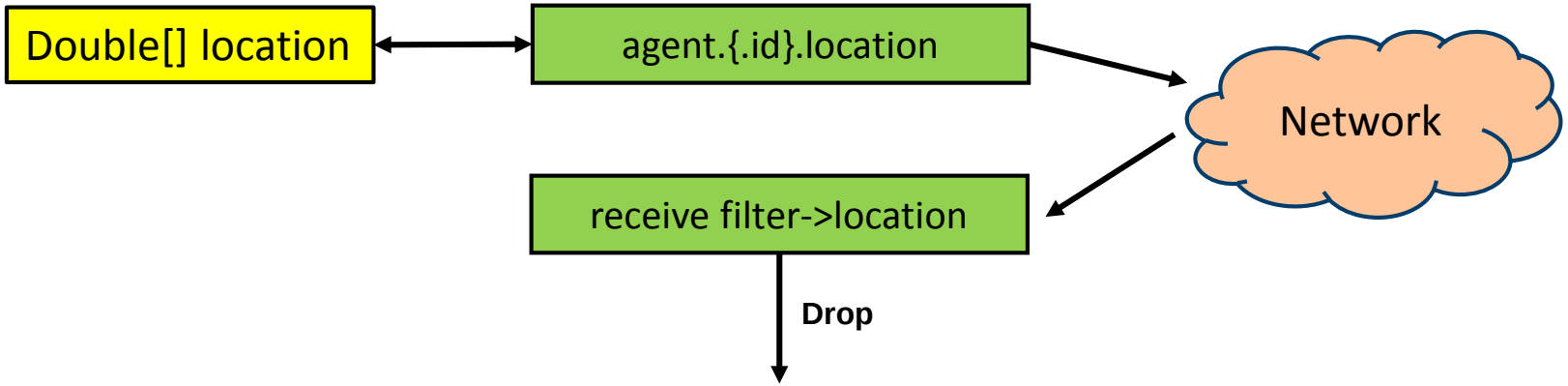
Planning Your Algorithm

- Is your **algorithm self-interested** or **multi-agent** and **collaborative**?
- Does your **algorithm need to share** information**?**



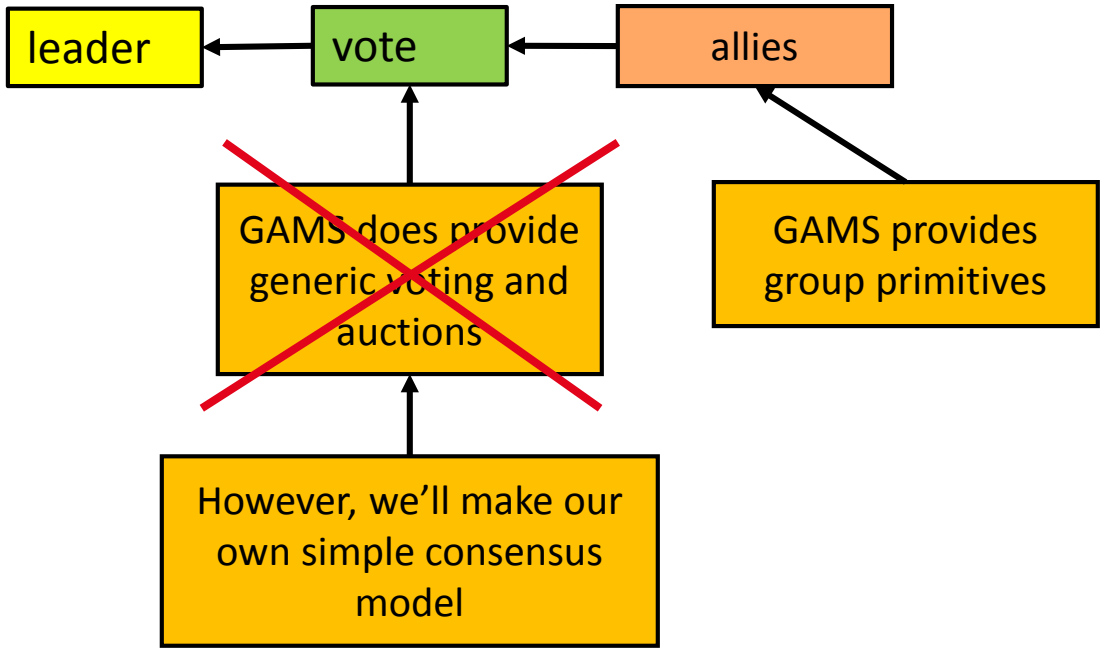
Remember the world model in MADARA?

- **Global variables can be one-way** by adding filters to the transport layer



Planning Your Algorithm

- Do you understand what your algorithm is supposed to do?
- **Can you visualize it? Can you draw it?**

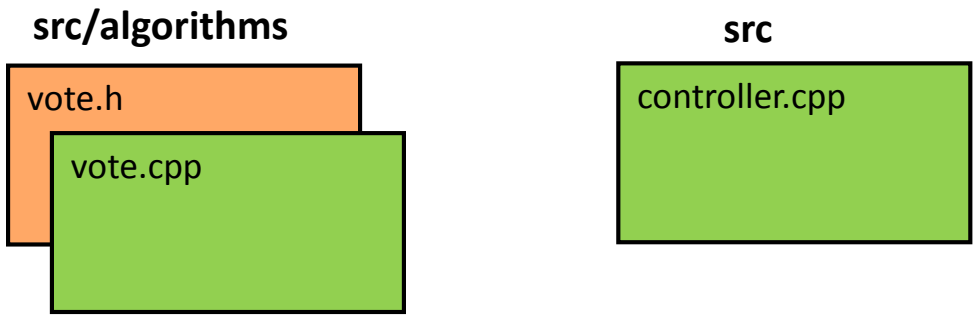


Generating Your Algorithm

- Let's **generate a leader election algorithm** called "vote" in a project called "leader_elect"

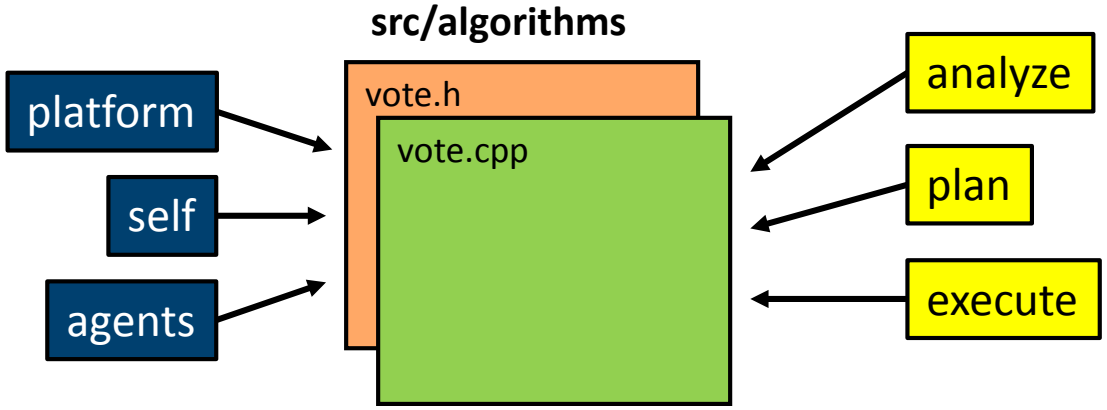
Linux terminal

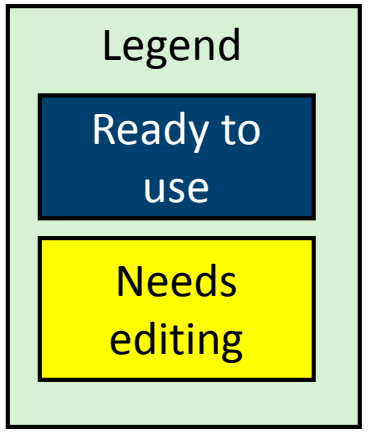
```
$GAMS_ROOT/scripts/projects/gpc.pl --path $PROJECT_ROOT/leader_elect
cd $PROJECT_ROOT/leader_elect
$GAMS_ROOT/scripts/projects/gpc.pl --new-algorithm vote
```



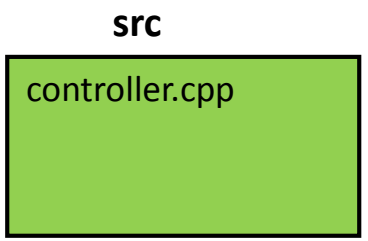
Understand What has been Generated

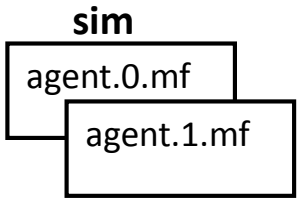
- **Algorithms** have **platforms** and **self information** that are **ready-to-use**
- **Algorithms** have **methods** that **need to be implemented**





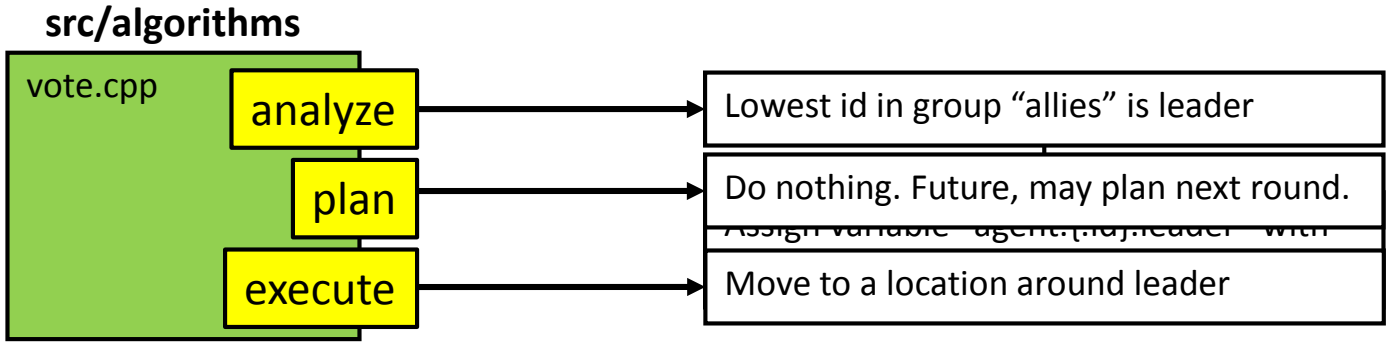
- The **controller adds** the **"vote" algorithm factory**
- **User** can **call "vote" algorithm remotely**
  - **agent.{.id}.algorithm = "vote"**
  - **swarm.algorithm = "vote"**
- **User** can **specify algorithm in sim folder**
  - **Simply edit the agent.*.mf files**
  - **Can also specify "--algorithm vote" to gpc.pl**

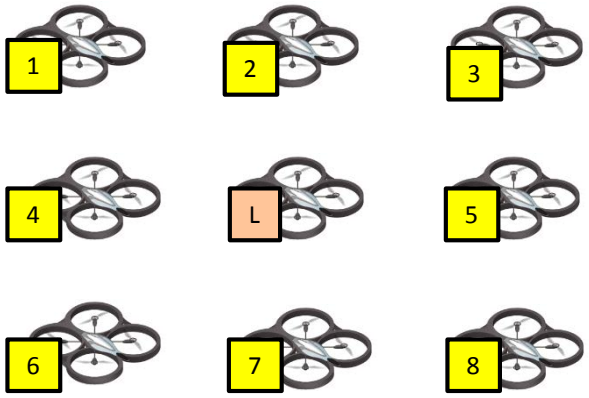




Changing Your Algorithm

- Let's fill in the blanks of our algorithm by **outlining what we want**





- **Location based on sorted ranking** in group
- **0th ranking is leader at center position**
- **1-8th ranking surrounds leader**
- **Any other member stays at current location**

Changing Your Algorithm

- Let's write the code for each method: **vote.h**

src/algorithms

vote.h

includes

Add protected

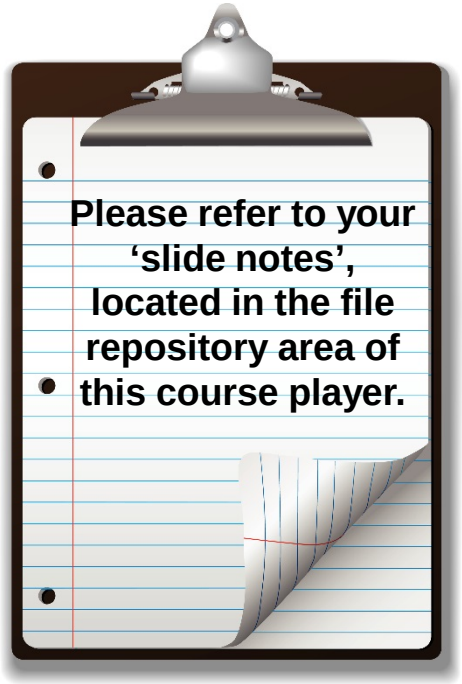
Add two include files to top of document

```
#include "gams/groups/GroupBase.h"
#include "madara/knowledge/containers/String.h"
```

```
class vote : …
{
```

Add a protected section and the following vars

```
  protected:

  // the group of allies
  gams::groups::GroupBase * group;

  // the member list for allies
  gams::groups::AgentVector members;

  // indicate our vote for the leader
  madara::knowledge::containers::String my_leader;
```



Changing Your Algorithm

- Let's write the code for each method: **vote.cpp**

src/algorithms

vote.cpp

includes

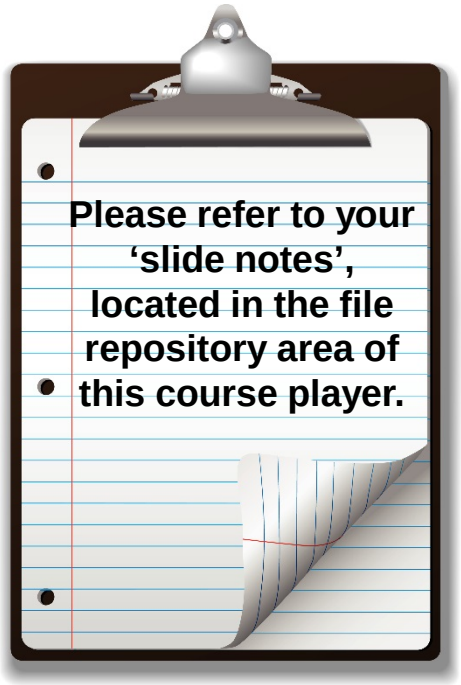
Add five include files to top of document

```
#include "gams/groups/GroupFactoryRepository.h"
#include "gams/pose/Position.h"
#include "gams/pose/CartesianFrame.h"
#include "gams/pose/GPSFrame.h"
#include <algorithm>
```

constructor

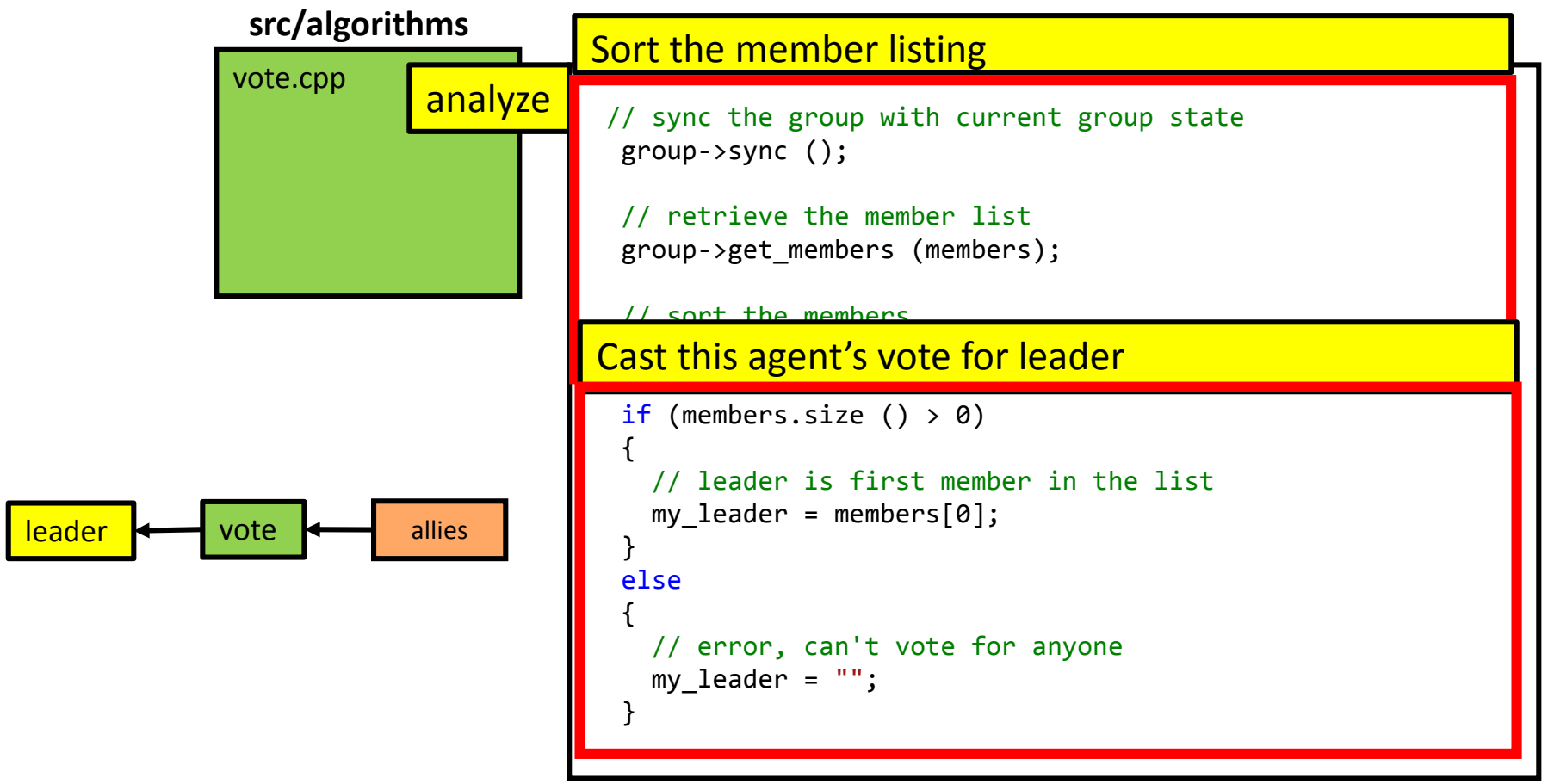
Setup leader vote location and group location

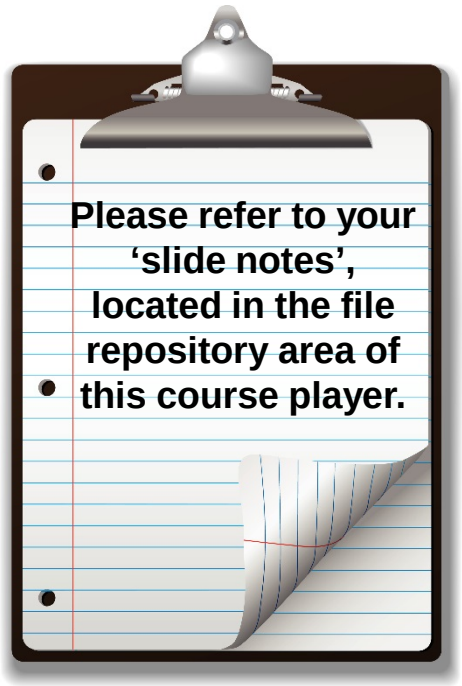
```
if (knowledge)
{
  my_leader.set_name (self_->agent.prefix + ".leader", *knowledge);

  gams::groups::GroupFactoryRepository factory (knowledge);
  group = factory.create ("group.allies");
}
```



Changing Your Algorithm

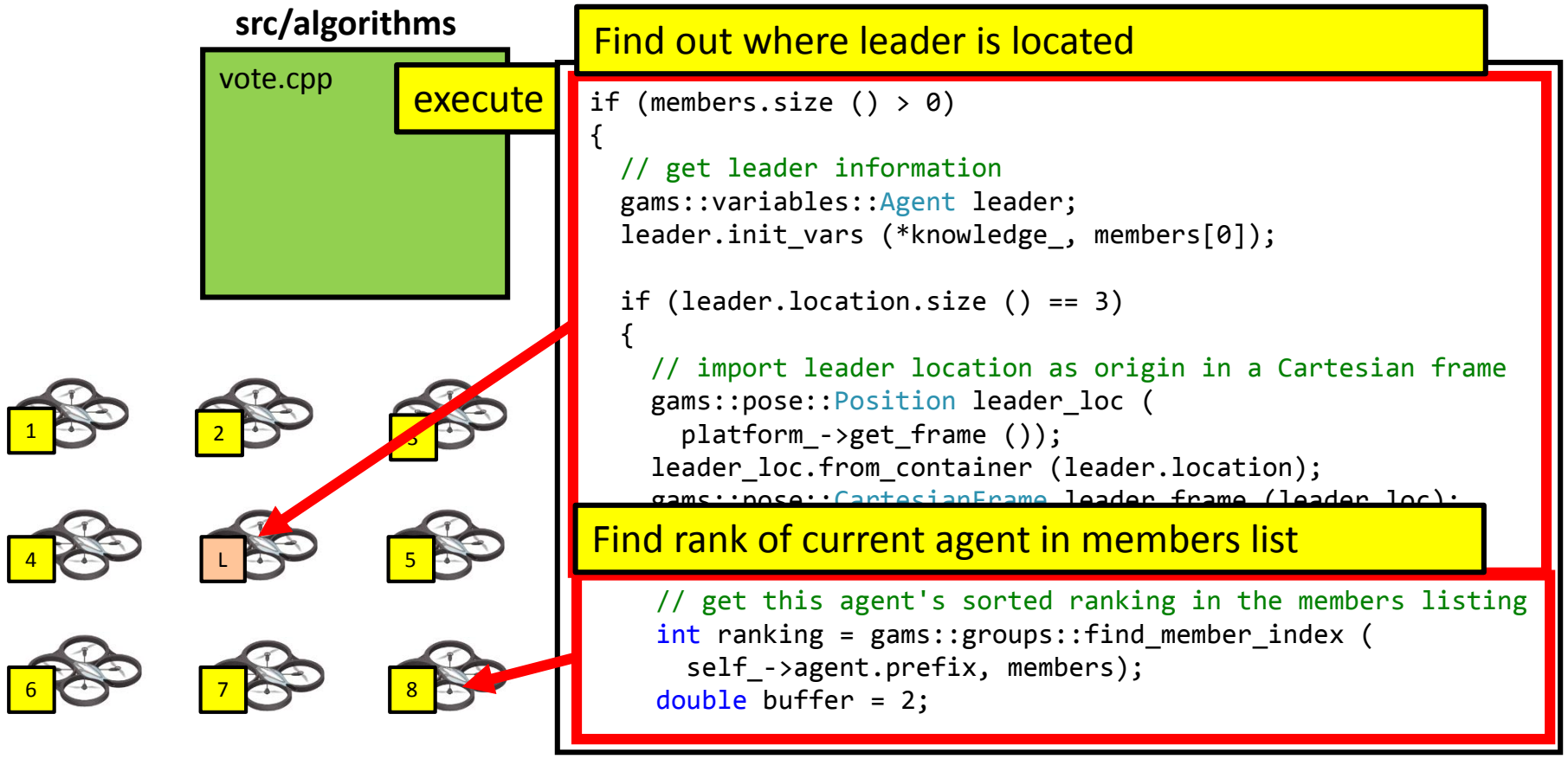
- Let's write the code for each method: **vote.cpp::analyze**

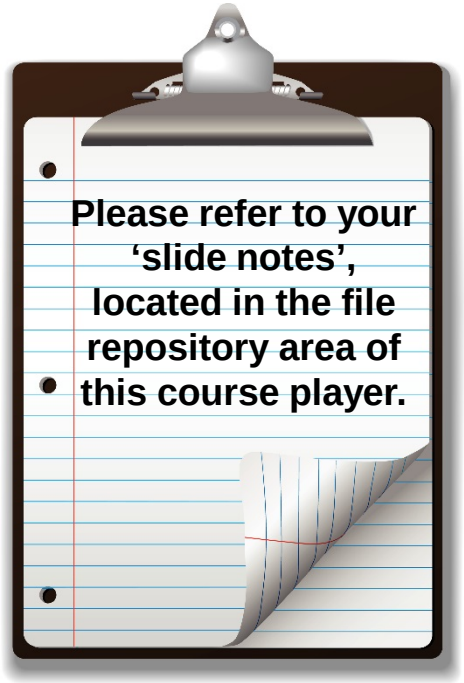




Changing Your Algorithm

- Let's write the code for each method: **vote.cpp::execute**





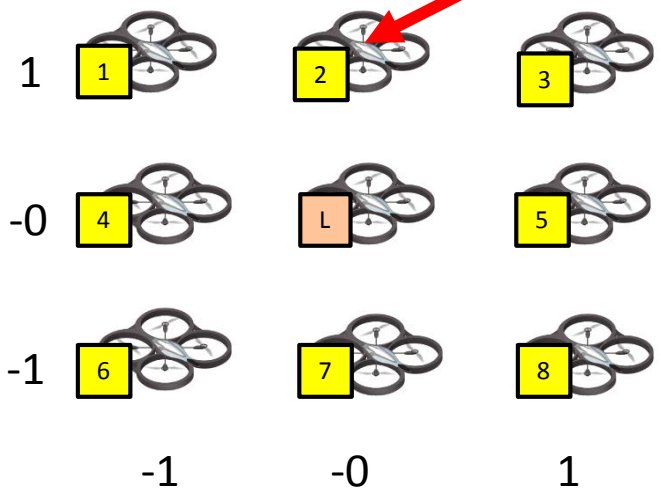
Changing Your Algorithm

- Let's write the code for each method: **vote.cpp::execute**

src/algorithms

vote.cpp

execute



Determine row and column in formation

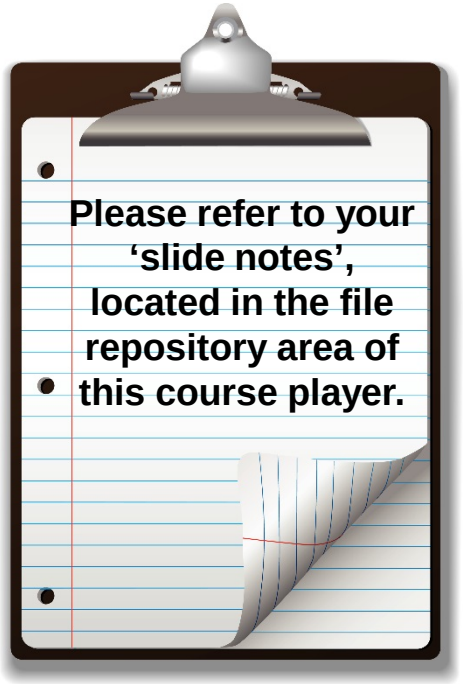
```
    if (ranking > 0 && ranking < 9)
    {
      int position = ranking;

      if (ranking <= 4)
```

ranking = 2, position = 1, row = 1, col = 0

```
      int row = ((position / 3) - 1);
```

Determine location in leader frame and move

```
        gams::pose::Position destination (
          leader_frame, col * buffer, row * buffer);

        platform_->move (destination);
      } // end ranking check
    } // end leader has a valid location
  } // end group has enough members check
```

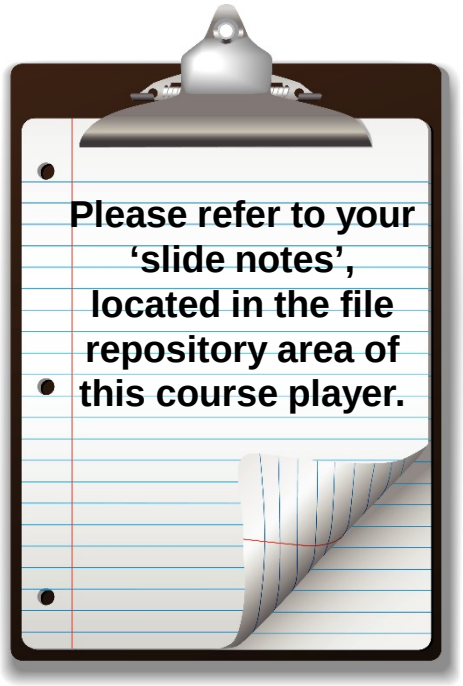


Configuring Your Simulation

Let’s configure the sim with 9 agents, a custom group, unique heights and set the logging level

Linux terminal

```
cd $PROJECT_ROOT/leader_elect
$GAMS_ROOT/scripts/projects/gpc.pl --agents 9 --distributed
$GAMS_ROOT/scripts/projects/gpc.pl --algorithm vote
$GAMS_ROOT/scripts/projects/gpc.pl --min-height 4 --unique
$GAMS_ROOT/scripts/projects/gpc.pl --group group.allies
$GAMS_ROOT/scripts/projects/gpc.pl --gams-debug 3 --madara-debug 1
```



Compiling and Running Your Algorithm

Compilation and simulation can be done through the **action script**

Linux terminal

```
cd $PROJECT_ROOT/leader_elect
./action.sh compile-vrep vrep sim
```

You should see your new algorithm running in a vrep window!

Compiling and Running Your Algorithm

This is what your algorithm looks like!

What You've Learned in this Tutorial Series

- What GAMS and MADARA are
- How GAMS and MADARA are different from ROS
- How to install GAMS and MADARA
- How to create and configure distributed autonomy projects
- How to add external libraries to distributed autonomy projects
- How to simulate GAMS algorithms in VREP
- How to plan an algorithm
- How to generate GAMS algorithms
- How to control knowledge dissemination
- How to change algorithms at runtime

Future Tutorials

- If these topics interest you, please contact us as we can focus new tutorials on what the community wants to know more about
  - **On-site tutorials with real drones in our drone lab**
  - **Formally verifying distributed autonomous systems for correctness and safety**
  - **Real-time scheduling with MADARA and GAMS**
  - **Creating GAMS platforms for new robotic systems**
  - **More advanced mission-focused algorithm creation**
  - **Accent algorithms for manipulating robotic actuators or sensors while performing other core autonomy**