

# **Debugging Tools for Charm++ Applications**

Filippo Gioachin

University of Illinois at Urbana-Champaign

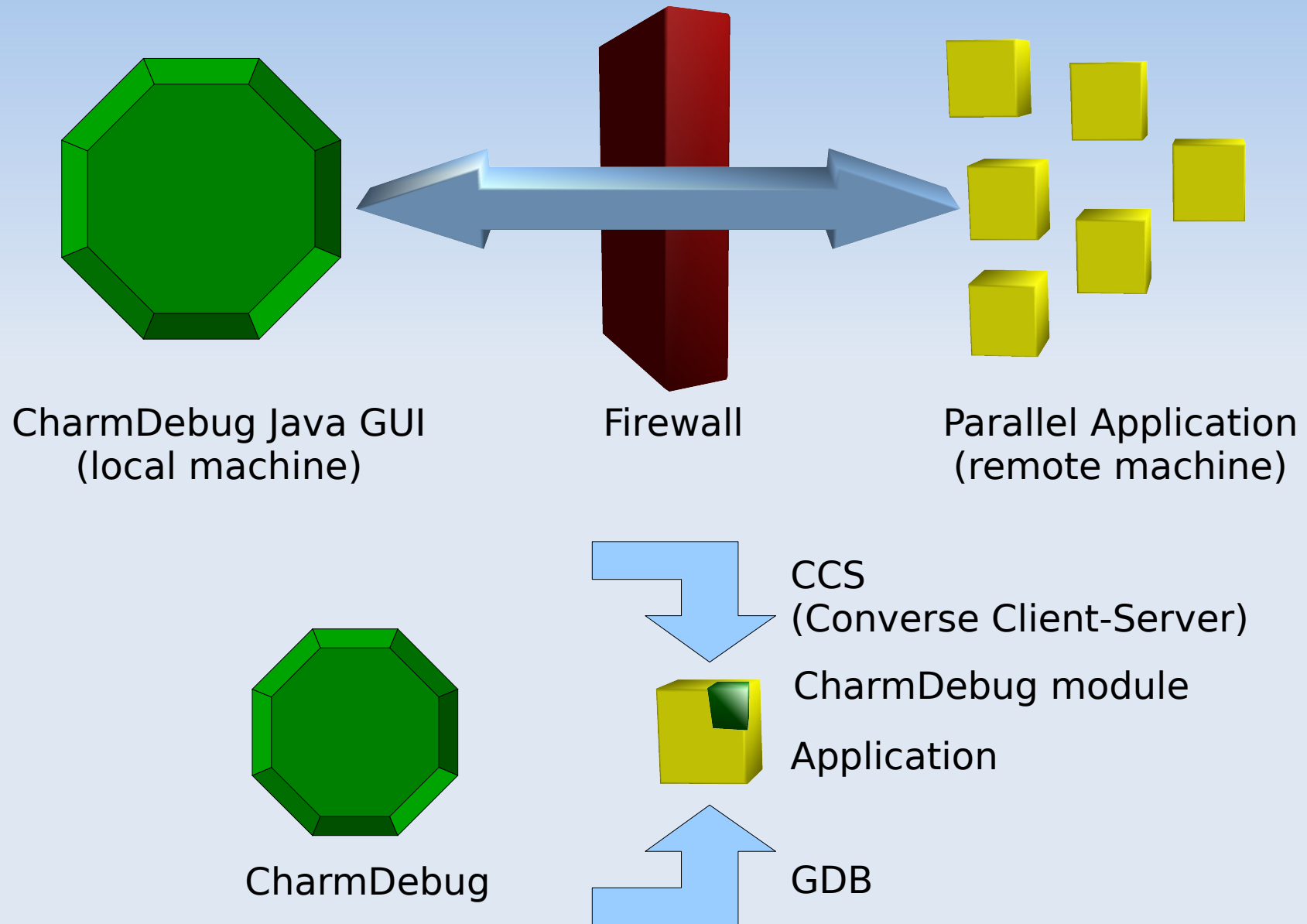
# Outline

- Overview of CharmDebug
- Introspection
- Memory Debugging
  - Leak detection
  - Cross-charre corruption detection

# CharmDebug Overview

- Developed specifically with Charm++ in mind
  - Provides information at the Charm++ abstraction level
- Composed of two modules:
  - Java GUI (client)
  - Plugin inside Charm++

# CharmDebug Architecture



# Introspection

- The capability of a system or application to inspect its own state.
- For an application this includes the capability to
  - Identify the type of a variable;

```
MyType var;
```

- Identify the layout of a type, and browse it.

```
class MyType {  
    int a;  
    double b;  
}
```

# Introspection in Charm++

- Interactivity
  - Want to run introspection code on demand
  - “Are the values in my array getting too big?”
- Use CharmDebug's connection to the app.
- Charm++ Python interface

# Charm++ Python Interface

- Send Python code through a CCS channel
- Execute the code on the parallel application

```
group [python] myGroup {  
    entry [python] void getValue();  
}
```

- Code is bound to a charm element (myGroup)
- Python can use method getValue

```
def check(self):  
    value = charm.getValue(data, 10)
```

# Browsing variables

```
class MyClass {  
    int length;  
    double * data;  
    MyExtraData * extra;  
}
```

- C++ is not reflective
  - Cannot access data structure layout
- We use the information already collected in CharmDebug to provide the extra information needed at runtime



# Introspection with CharmDebug

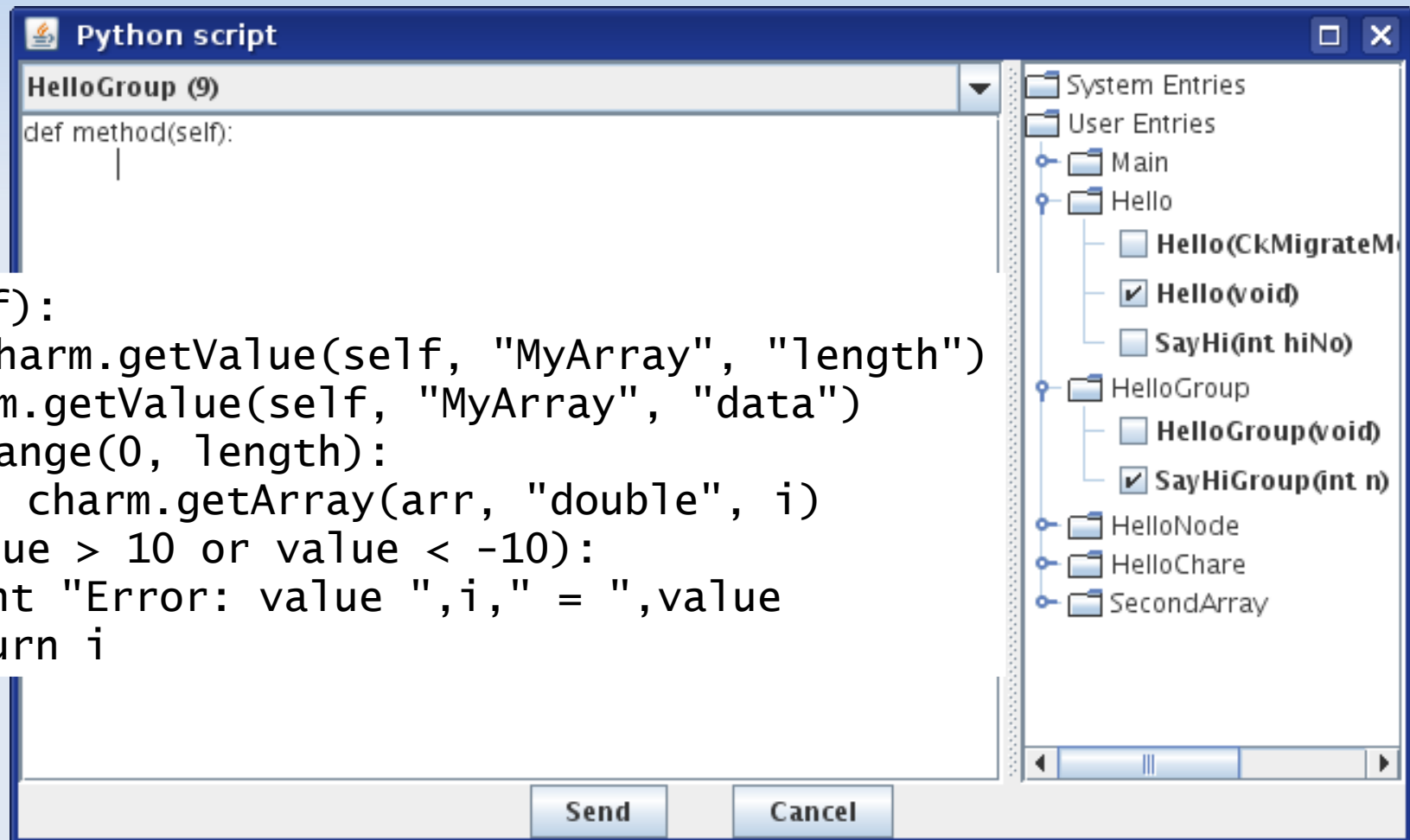
- A *group* part of CharmDebug plugin is bound to Python
  - No need to modify user code
- Set of functions exported:

```
group [python] CpdPythonGroup {  
    entry [python] void getValue();  
    entry [python] void getArray();  
    entry [python] void getCast();  
    entry [python] void getStatic();  
}
```

# An Example

- Are the values in my array getting too big?

```
def check(self):  
    length = charm.getValue(self, "MyArray", "length")  
    arr = charm.getValue(self, "MyArray", "data")  
    for i in range(0, length):  
        value = charm.getArray(arr, "double", i)  
        if (value > 10 or value < -10):  
            print "Error: value ",i," = ",value  
        return i
```



# Memory Debugging

- Memory problems are typically subtle and hard to trace
- In Charm++ multiple chares reside on the same processor and share the same address space
- Focus
  - Memory leak
  - Cross-charge corruption

# Main View

- link with “-memory charmdebug”

The screenshot shows the Memory Processor 0 application window. The title bar reads "Memory Processor 0". The interface includes a control panel at the top with the following elements:

- Action** and **Info** tabs.
- Input fields for "Number of lines" (set to 50) and "Horizontal pixels" (set to 1400).
- Input fields for "Line size" (set to 12) and "Bytes per pixel" (set to 71).
- An "Update" button.

The main area displays a memory dump with a grid of colored cells (red, blue, black, pink) representing data. The bottom section, titled "Information", contains the following text:

```
Memory type: chare object
Slot at position 0x870118 of size 144 bytes. Belonging to chare 13. Backtrace:
function CkArray::insertElement(CkMessage*) (0x4b0b9e) at ckarray.C:639
function CkArray::insertInitial(CkArrayIndex const&, void*, int) (0x4b0ede) at ckarray.C:694
function CkArrayMap::populateInitial(int, CkArrayIndexMax&, void*, CkArrMgr*) (0x4a2637) at cklocation.C:204
function CkLocMgr::populateInitial(CkArrayIndexMax&, void*, CkArrMgr*) (0x4b3a6c) at cklocation.h:521
function CkArray::CkArray(CkArrayOptions&, CkMarshaledMessage&, _ckGroupID) (0x4b1867) at ckarray.C:530
function CkIndex_CkArray::_call_CkArray_marshall1(void*, CkArray*) (0x4b1b7e) at CkArray.def.h:178
```

# View by Chare ID

**Memory Processor 0**

Action Info

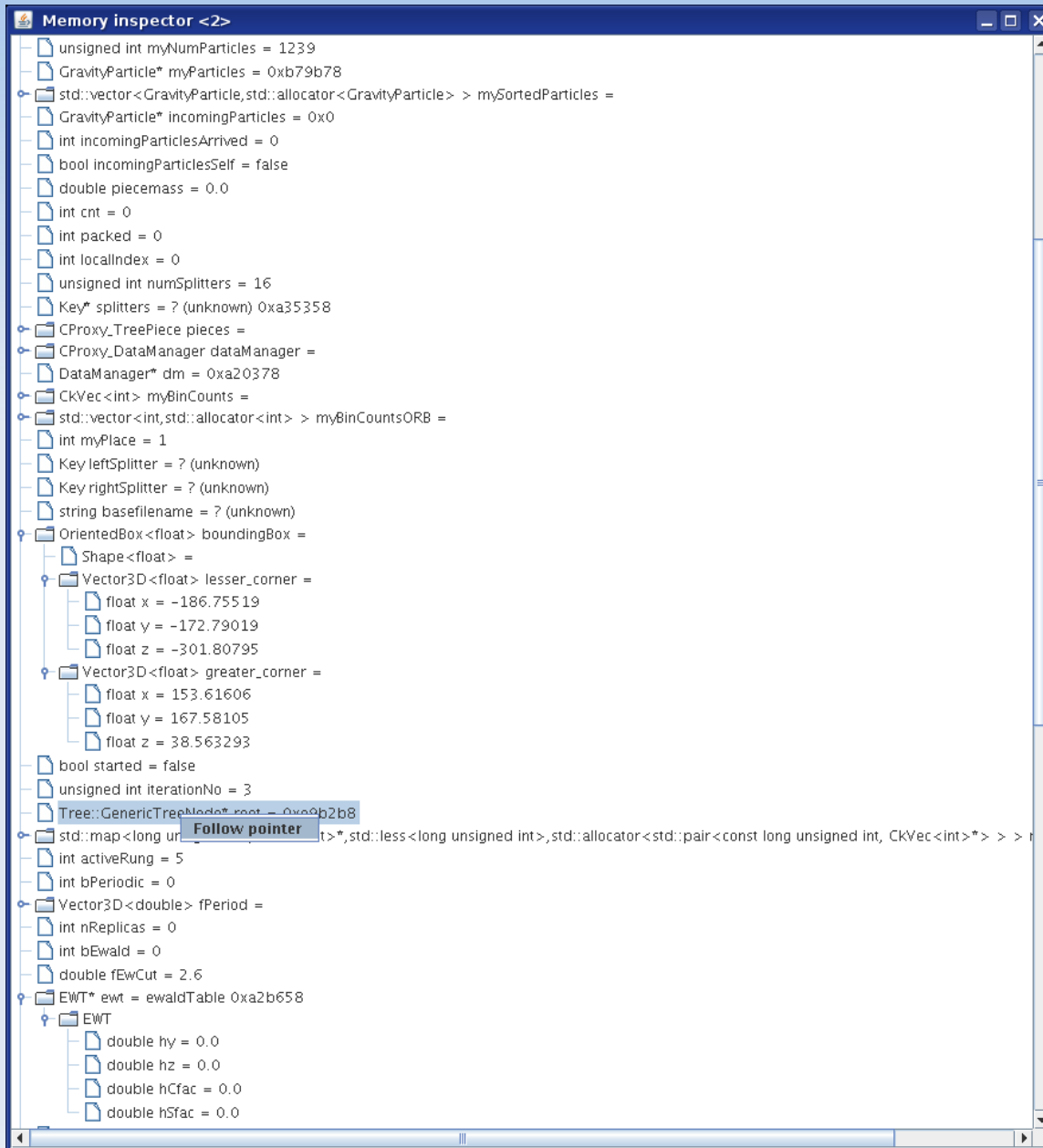
Number of lines: 50  
Horizontal pixels: 1400  
Line size: 12  
Bytes per pixel: 71

Update

**Information**

Memory type: user  
Slot at position 0x966518 of size 1616 bytes. Belonging to chare 15. Backtrace:  
function ?? (0x2aaaab308dd9) at ??:0  
function Jacobi::begin\_iteration() (0x473ab1) at jacobi2d.C:154  
function CkIndex\_Jacobi::call\_begin\_iteration\_void(void\*, Jacobi\*) (0x470d17) at jacobi2d.def.h:453  
function CkDeliverMessageReadOnly (0x494e50) at ck.C:414  
function CkLocRec\_local::invokeEntry(CkMigratable\*, void\*, int, bool) (0x4a71e8) at cklocation.C:1025  
function CkMigratable::ckInvokeEntry(int, void\*, bool) (0x4b4f5e) at cklocation.h:306

# Inspecting Memory



- Example from ChaNGa
- Inspect memory
- Data is loaded in the client
- Still evolving

# Searching for Leaks...

- ChaNGa
  - Cosmological simulator
  - Runs in a step-wise fashion
- Observed effect: the allocated memory keeps increasing
- We ran the application for a few steps, then froze it

# Searching for Leaks... (2)

The screenshot displays the Charm Parallel Debugger interface. The window title is "Charm Parallel Debugger".

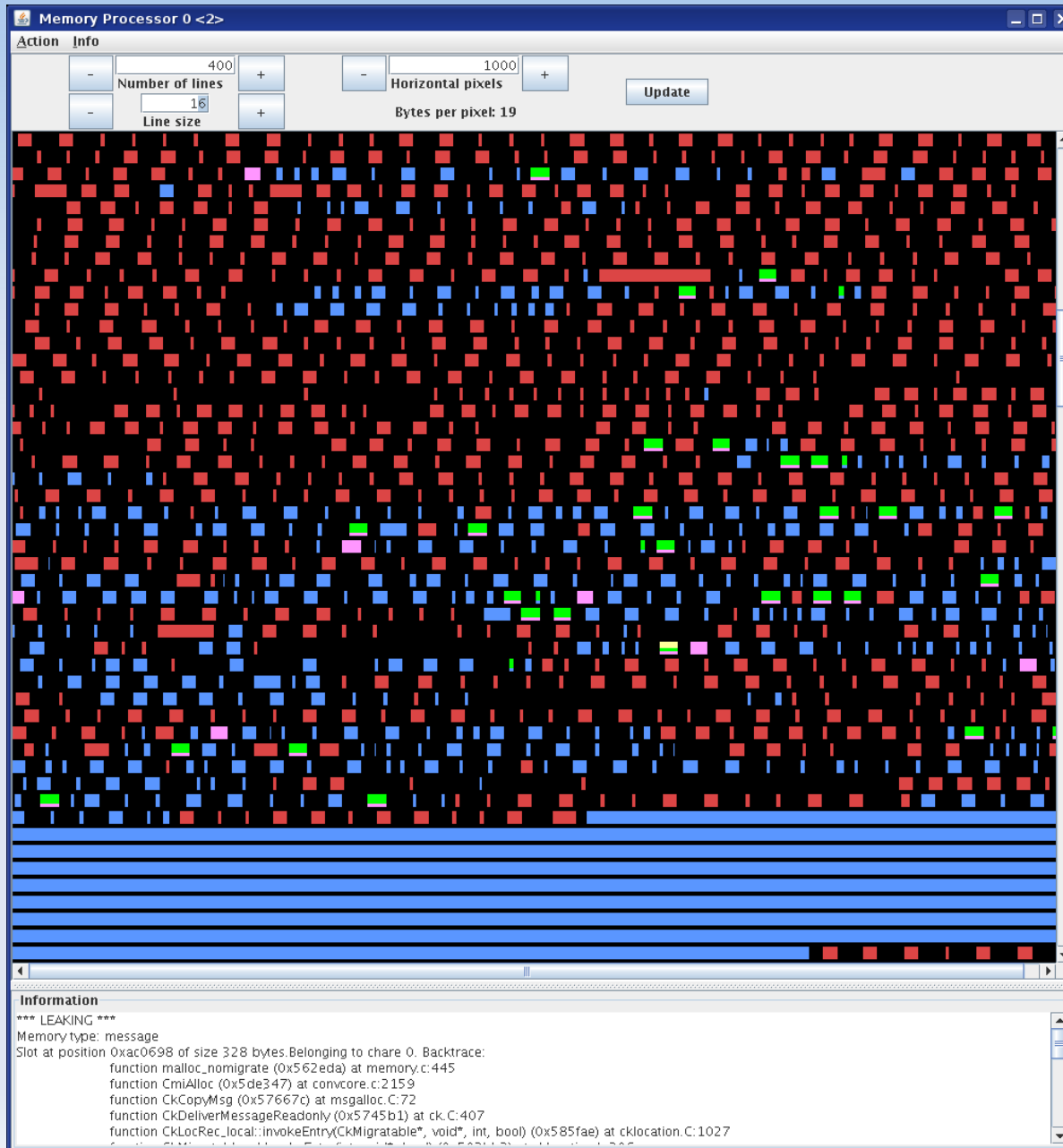
- Set Break Points:** A tree view on the left shows the project structure. Under "CacheManager", the function `cacheSync(double theta, int activeRung, double dEwhCut, const CkCallback &cb)` is selected and checked.
- Control Buttons:** A row of buttons includes "Start", "Step", "Continue", "Freeze", "Quit", and "Start GDB".
- Program Output:** A text area showing the execution log. It includes system information like "Server IP = 128.174.236.225" and a series of memory usage reports: `[0]: CmiMaxMemoryUsage: 7.179459 M` through `7.685104 M`. A message at the bottom states: "Breakpoint is set for function cacheSync(double theta, int activeRung, double dEwhCut, const CkCallback &cb) with an epldx = 135".
- View Entities on PE:** A dropdown menu shows "Messages in Queue" with a value of "0".
- Entities:** A list box at the bottom left shows "CacheManager::cacheSync(double theta, int activeRung, double dEwhCut, const CkCallback &cb)" selected.
- Details:** A text area at the bottom right shows the details of the selected entity: "Sender processor: 0", "Destination: CacheManager::cacheSync(double theta, int activeRung, double dEwhCut, const CkCallback &cb)", "Size: 72", and "User data: data={theta=0.7, activeRung=5, dEwhCut=2.8, cb=...}".

At the bottom of the window, a status bar reads: "Break point reached for Function = cacheSync(double theta, int activeRung, double dEwhCut, const CkCallback &cb)".





# Searching for Leaks... (3)



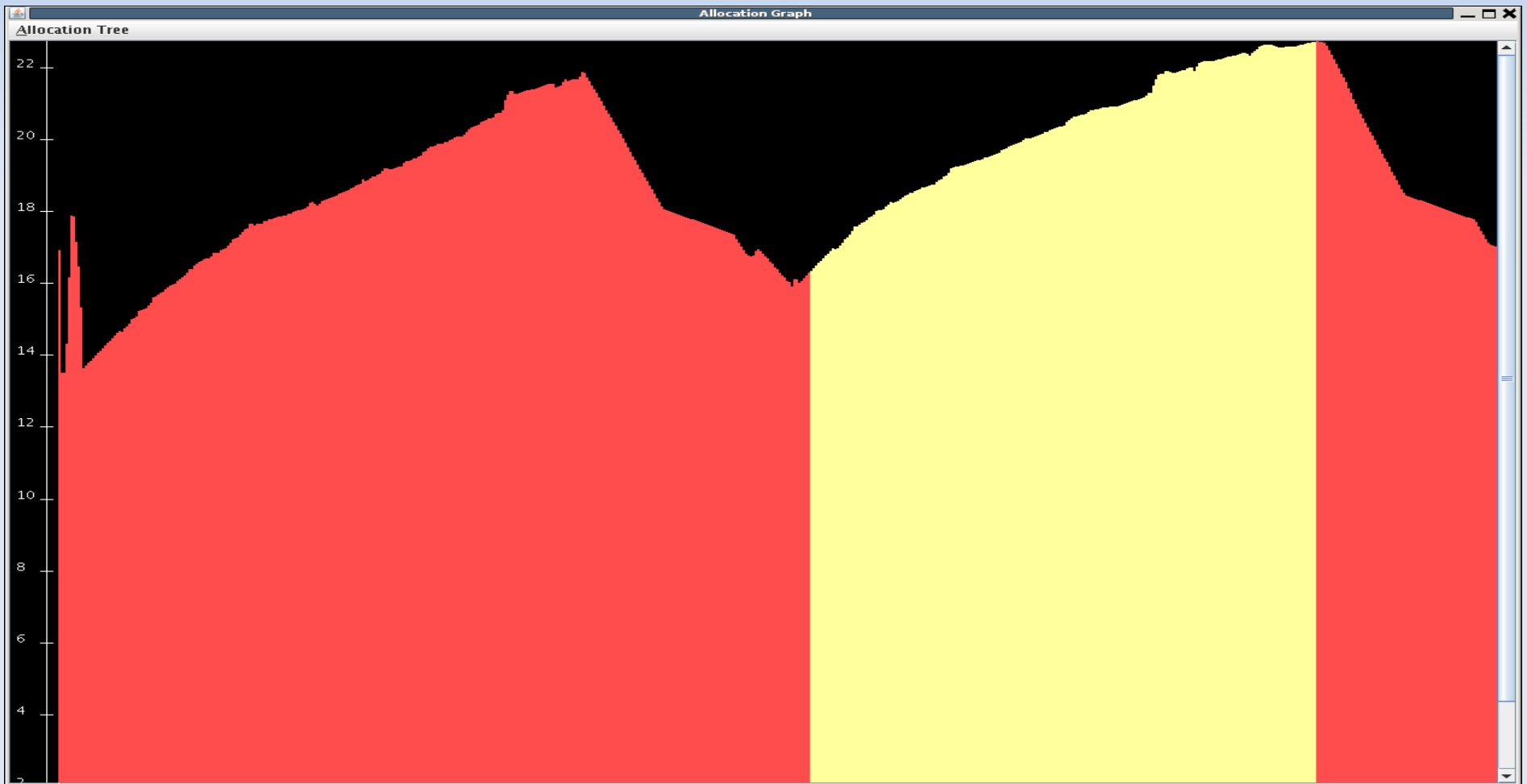
- Leaks are in green
- Messages are leaking (pink)

# Cross-chare Corruption

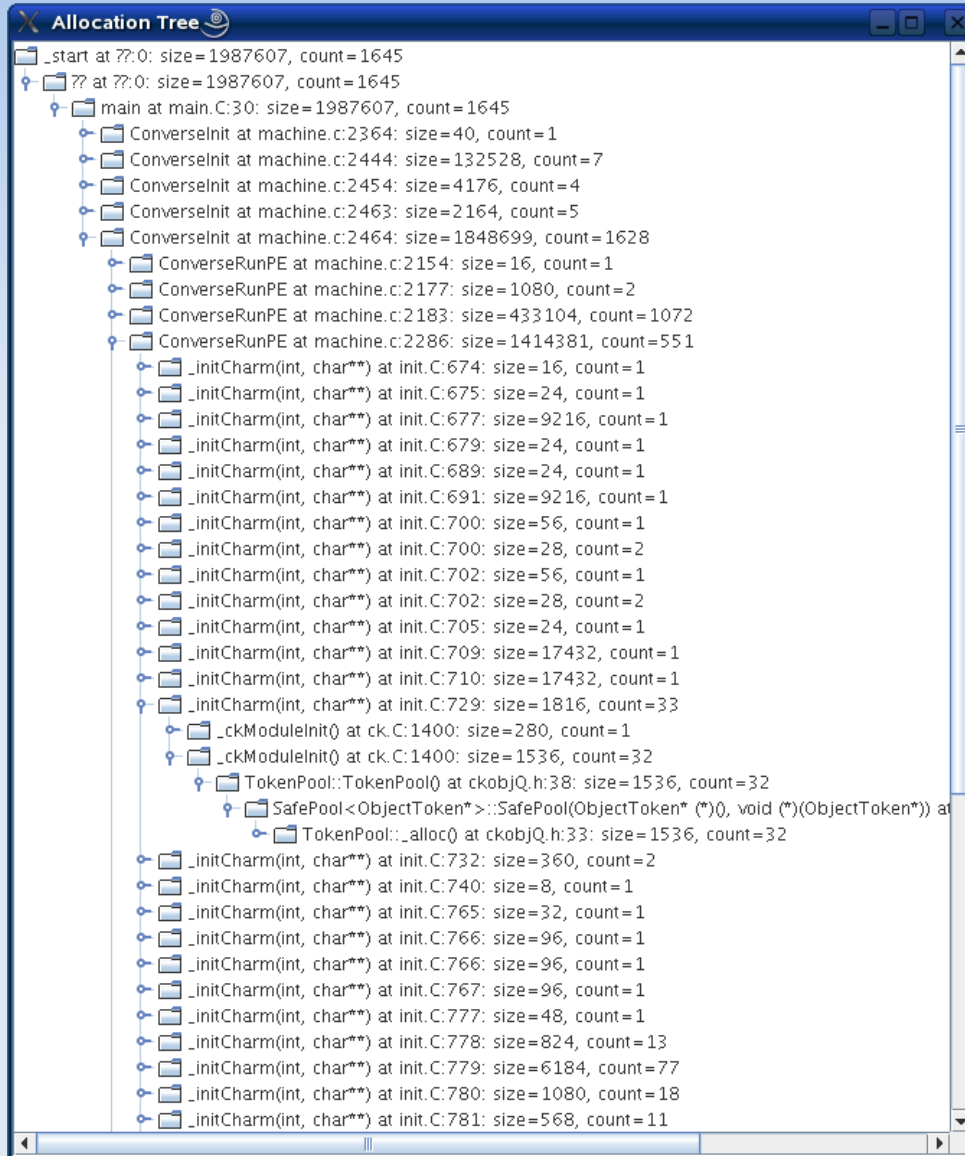
- A chare should access only to its data structures
  - Inside a processor the address space is shared
  - A chare can write some other chare's data
- Associate each chare an ID, and mark all its memory with that ID
- After an entry method, check if the chare modified some memory not belonging to it

# Allocation Graph

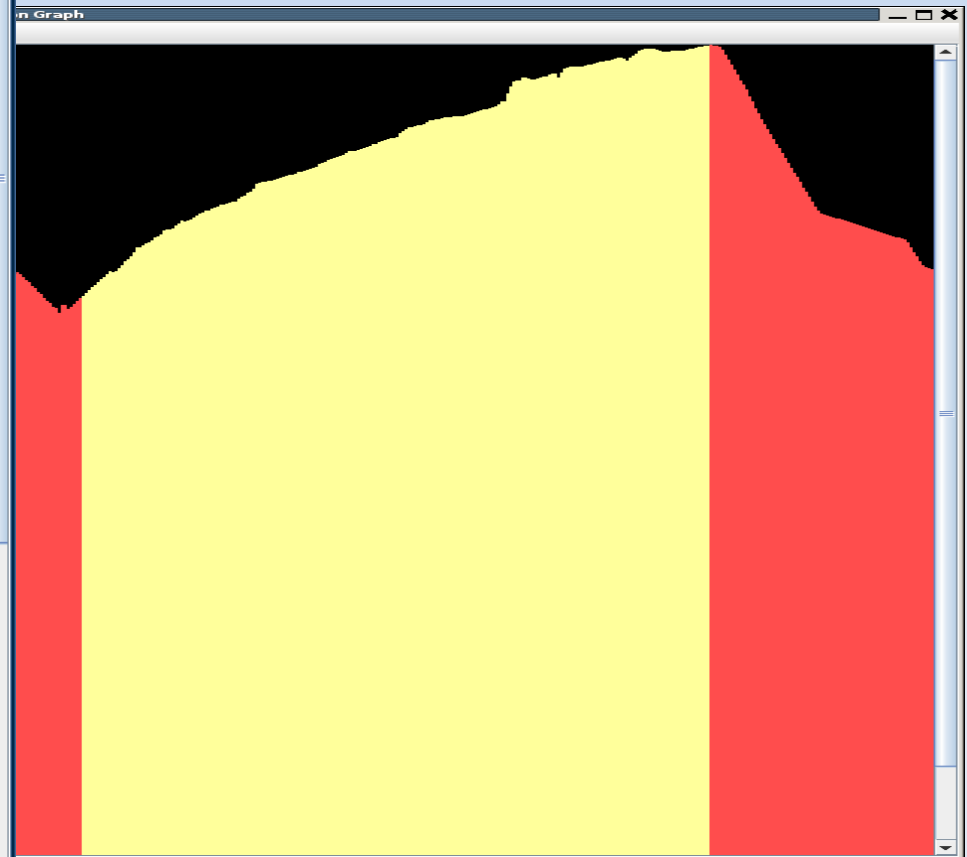
- link with “-tracemode memory”



# Allocation Graph & Allocation Tree



memory”



# Questions?

**Thank you**