

Memory Tagging in Charm++

Filippo Gioachin
Laxmikant V. Kalé

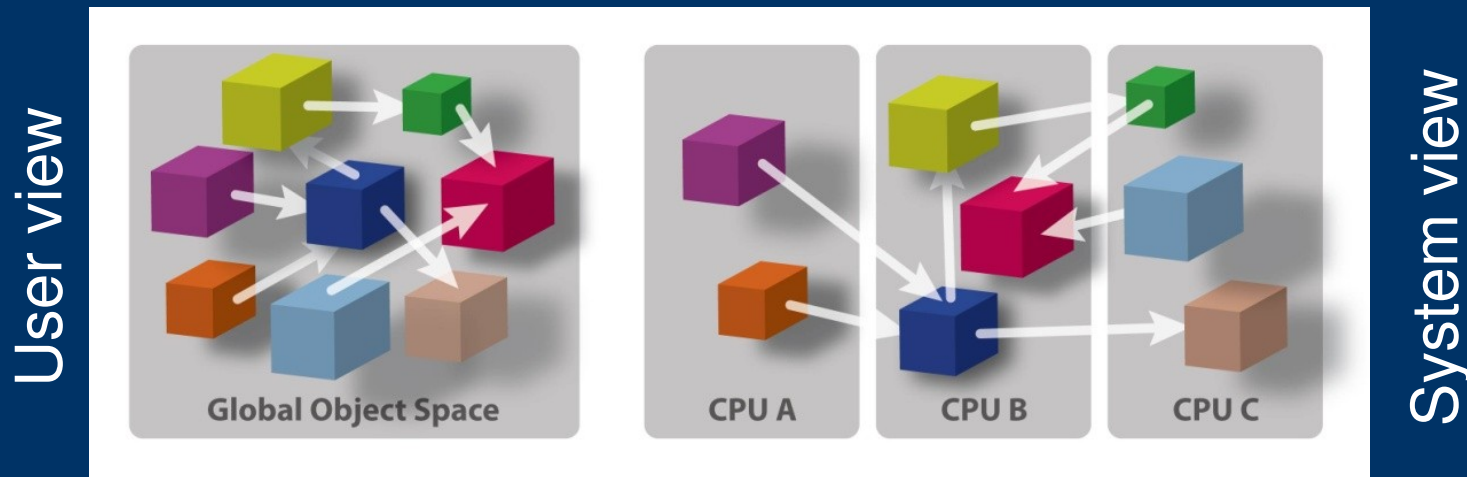
Department of Computer Science
University of Illinois at Urbana-Champaign

Outline

- Overview
 - Charm++ RTS
 - CharmDebug
- Memory tagging
 - Charm++ memory subsystem
 - Detecting memory violations
- Future work

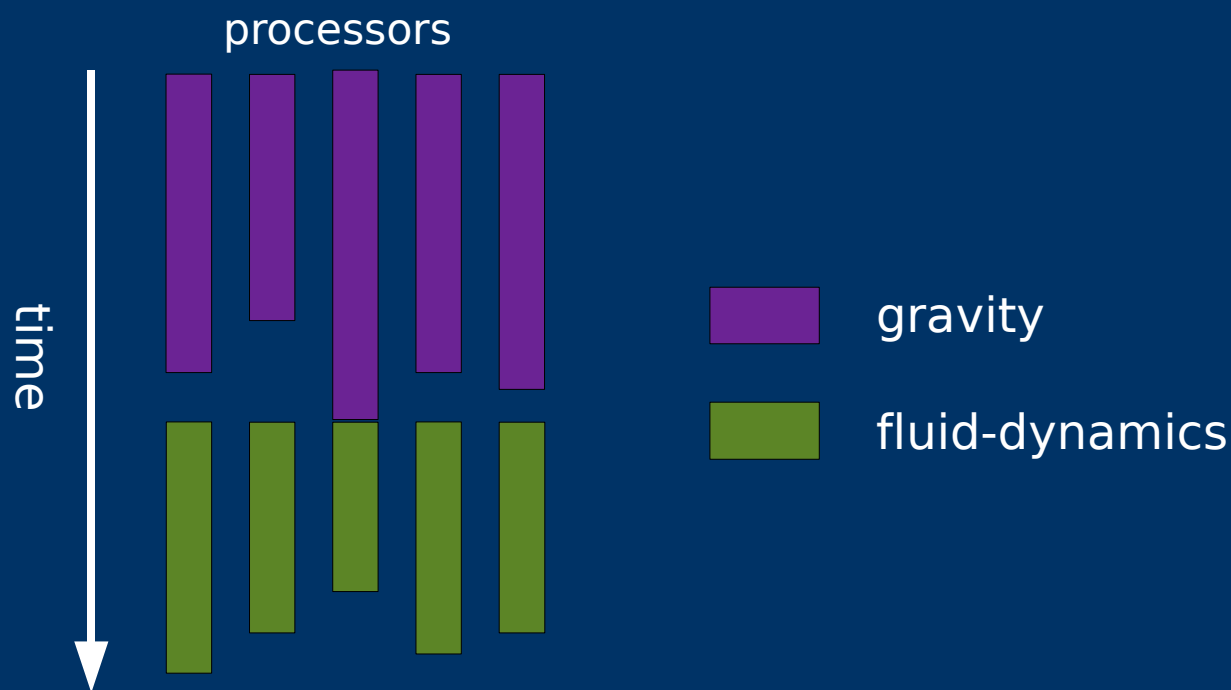
Charm++ Overview

- Middleware written in C++
- User decomposes work among objects (*chares*)
- System maps chares to processors
 - automatic load balancing
 - communication optimizations
- Communicate through asynchronous messages

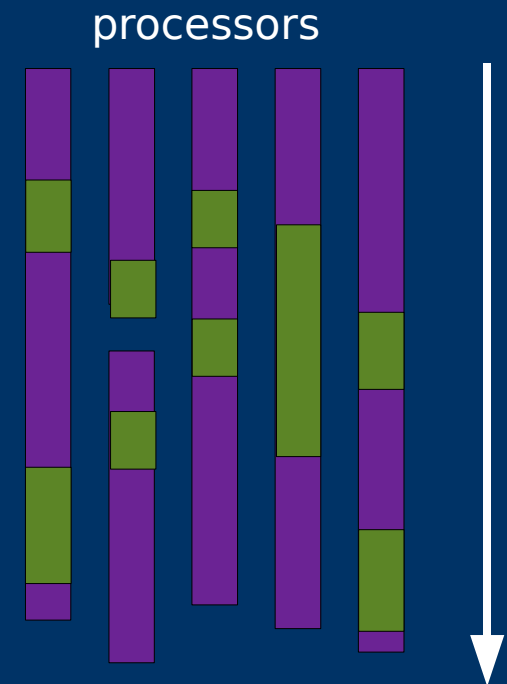


Example: cosmology

Separate modules



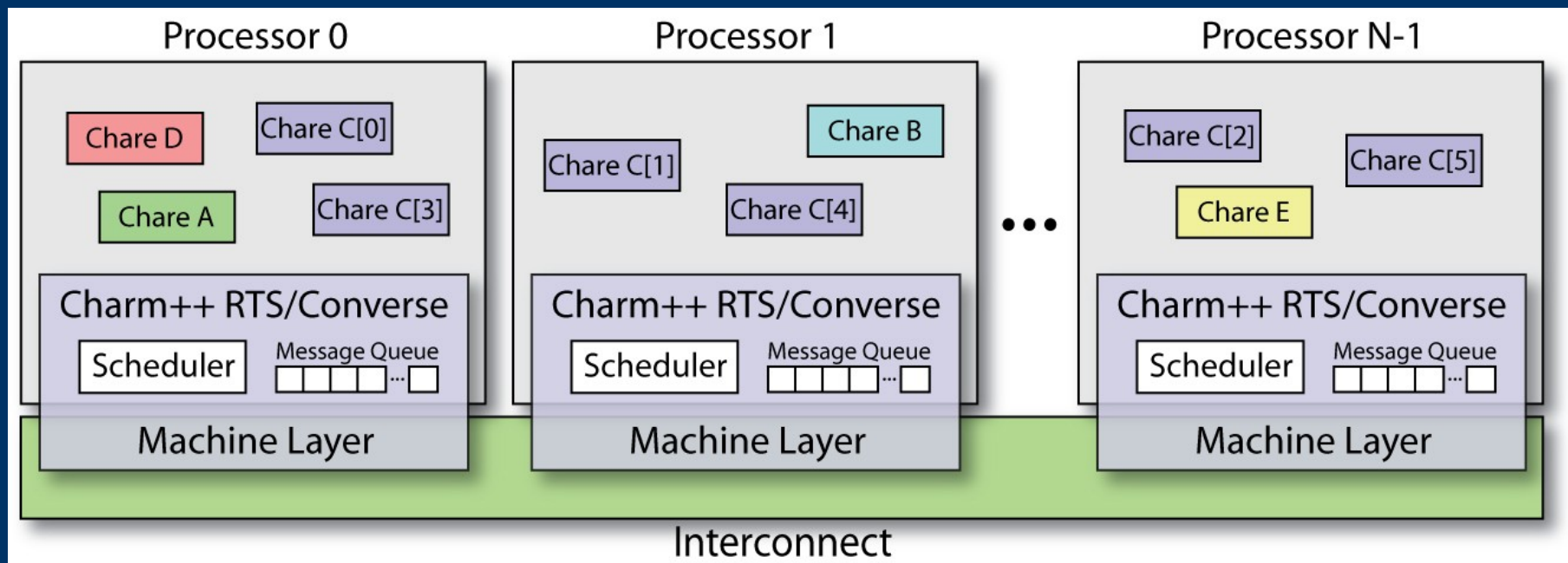
Interleaved modules



- Both sequential and parallel
- Extendable outside Charm++

Charm++ RTS

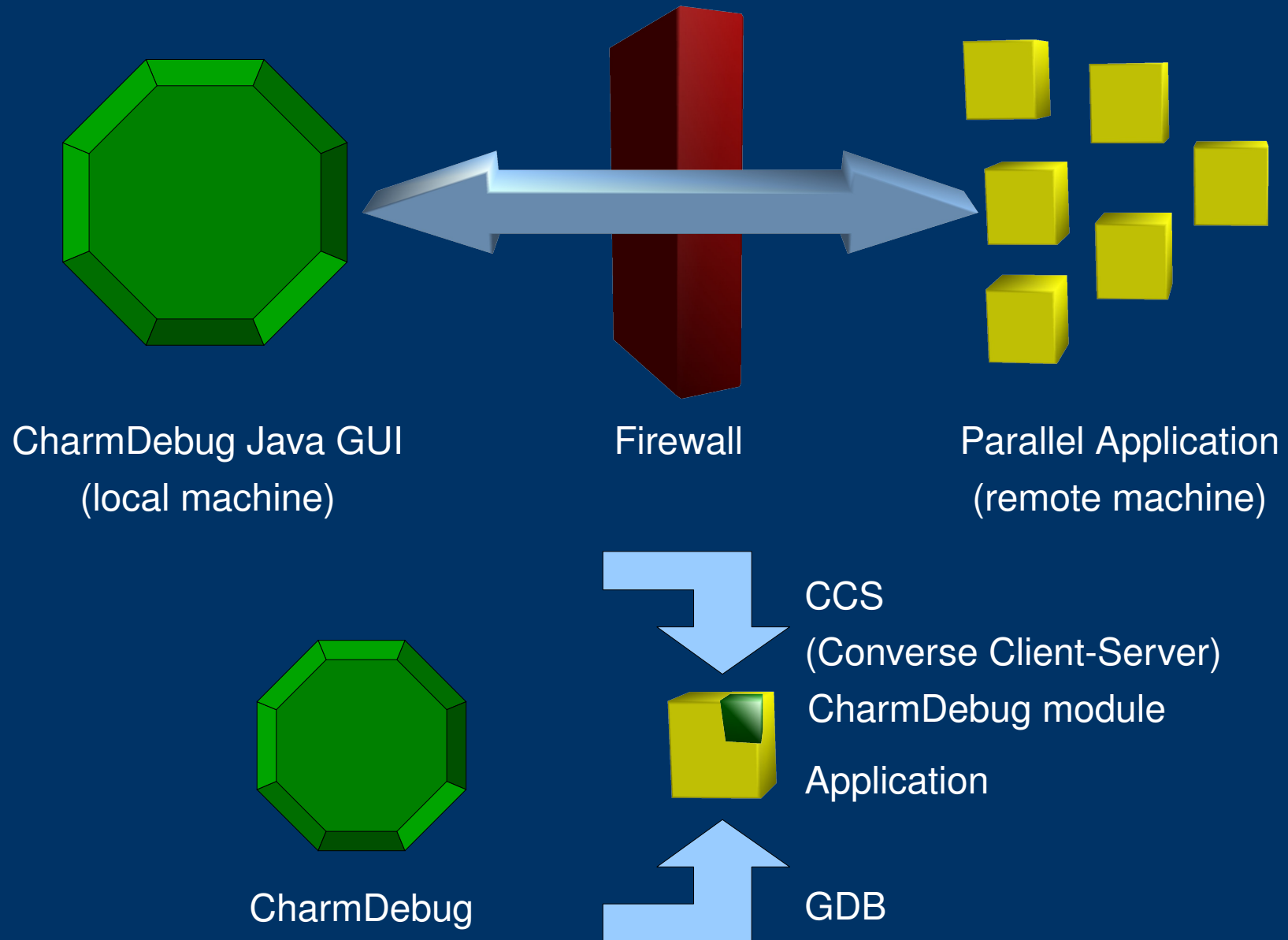
- Message contains:
 - destination chare ID
 - method to be invoked



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



Main View

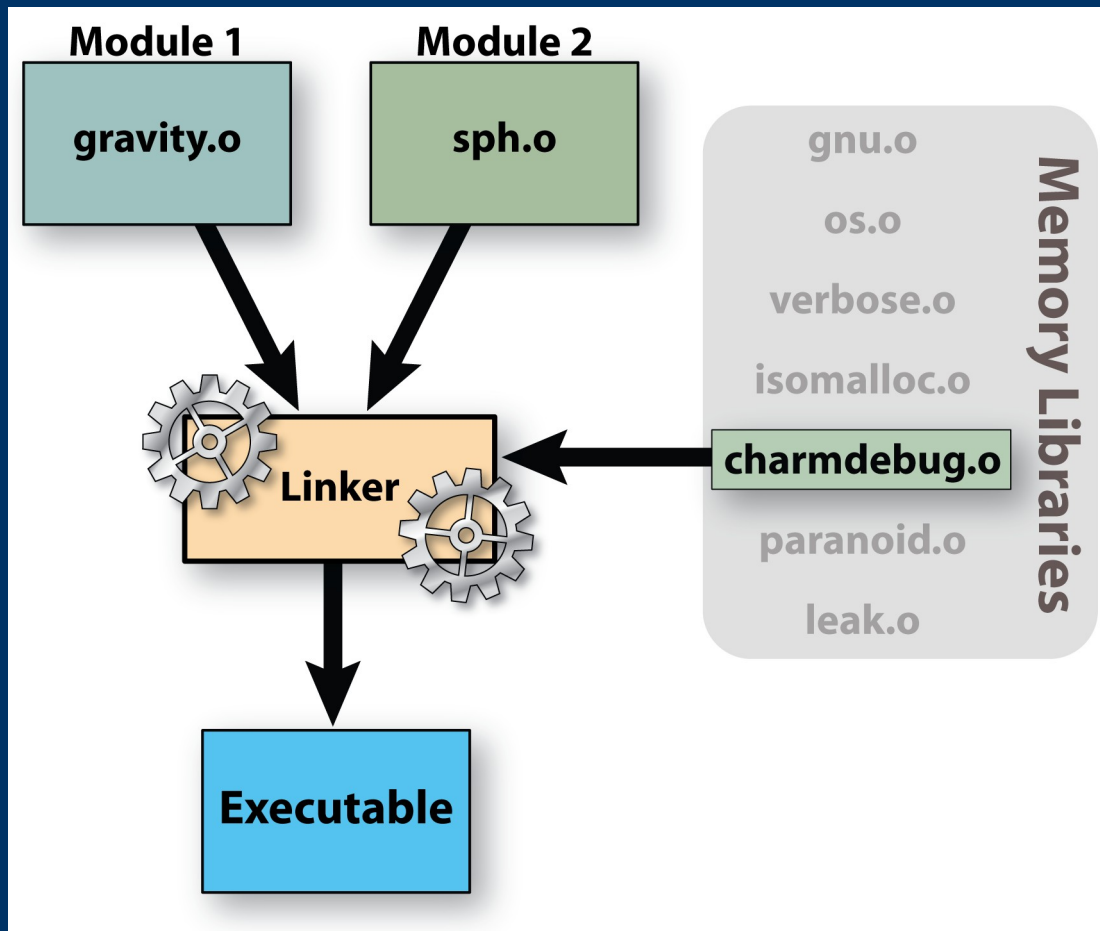
entry
methods

The screenshot shows the Charm Parallel Debugger interface with the following components:

- File Action:** A menu bar at the top left.
- Set Break Points:** A tree view on the left showing a hierarchy of folders: System Entries, User Entries, Main, Hello, HelloGroup, HelloNode, HelloChare, and SecondArray. Under the Hello folder, methods are listed: Hello(CkMigrate), Hello(void), and SayHi(int hiNo) (which is checked).
- Control Buttons:** A row of buttons: Start, Step, Continue, Freeze, Quit, and Start GDB.
- Program Output:** A central text area displaying a list of messages such as "Hello 1 created", "Hello 7 created", etc. A red "output" annotation is placed over this area.
- Pes:** A panel on the right with radio buttons for "all" and "even".
- View Entities on PE:** A dropdown menu showing "Messages in Queue" and a numeric value "0".
- Entities:** A list of entity names: Hello::SayHi(int hiNo), Hello::SayHi(int hiNo), and HelloChare::SayHi(int hiNo). A red "messages queued" annotation is placed over this list.
- Details:** A panel showing message details: "Sender processor: 0", "Destination: Hello::SayHi(int hiNo) (type 16)", "Size: 16", and "User data: data= {hiNo=27}". A red "message details" annotation is placed over this panel.
- Status Bar:** At the bottom, it says "Frozen processor 0".

processor
subsets

Charm++ Memory Subsystem



- At link time compile a memory library in
- Re-implement malloc, free, etc.
- Extend debugger capabilities

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

Memory view

Memory Processor 0

Action Info

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

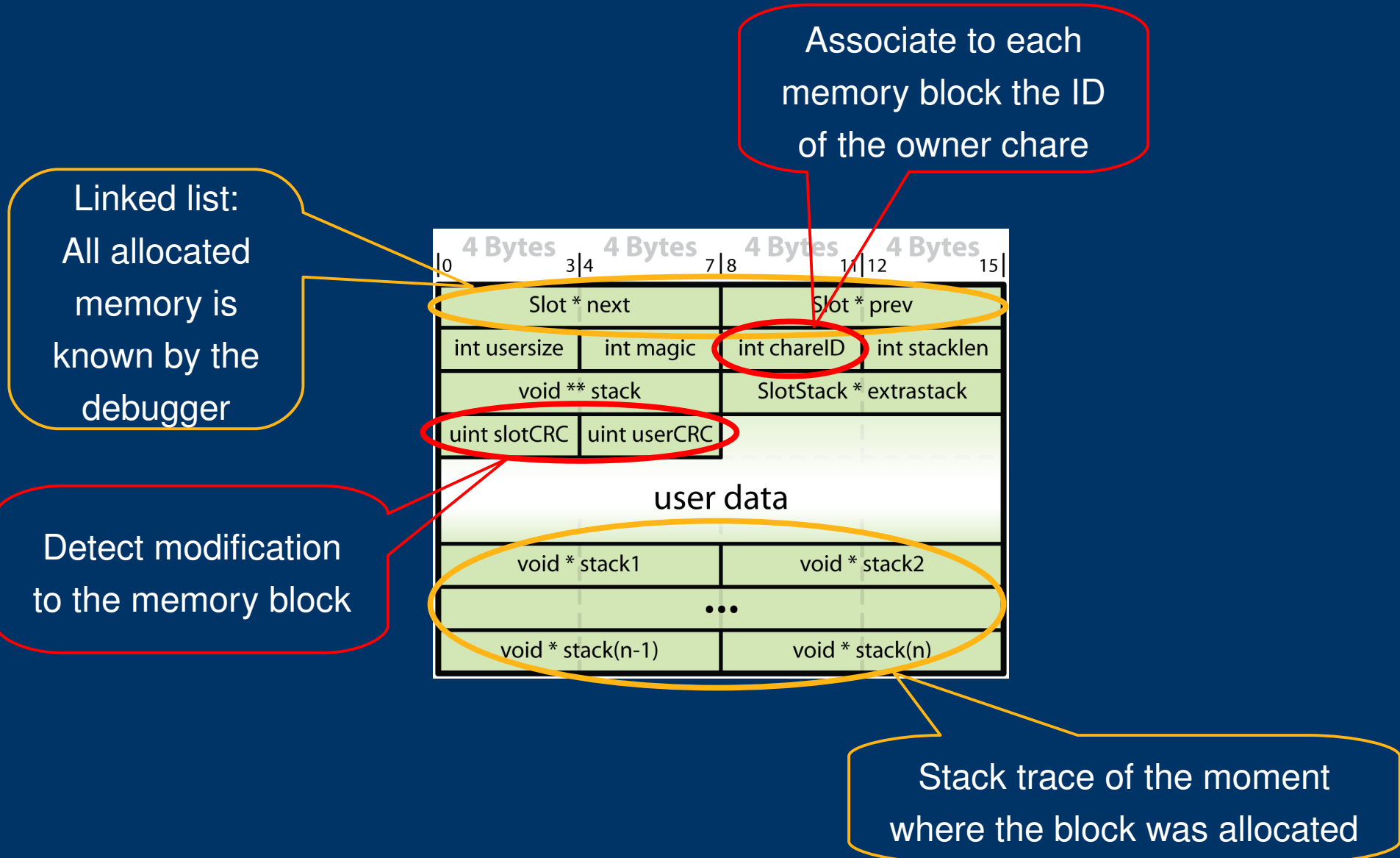
Update

Information

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

Memory Tagging



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

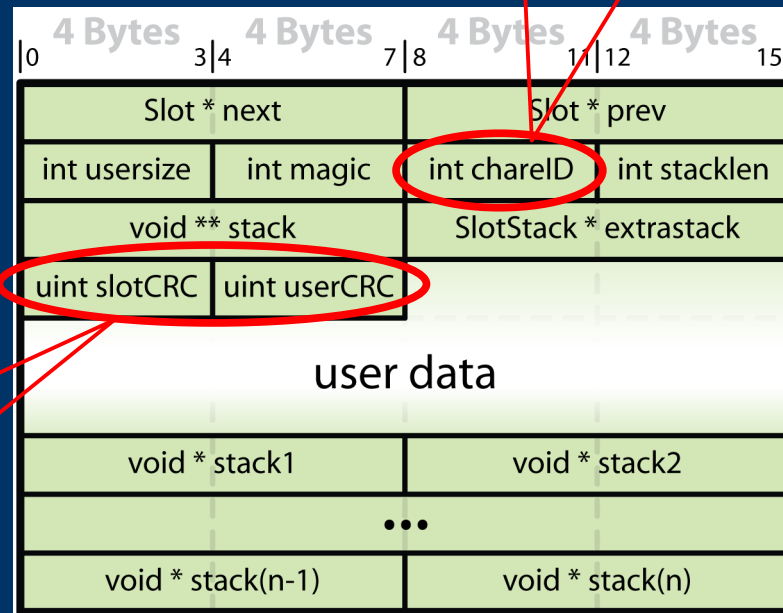
Memory type: user
Slot at position 0x966518 of size 1616 bytes. Belonging to chare 15. Backtrace:
function ?? (0x2aaaab308dd9) at ??:0
function Jacobi::begin_iteration() (0x473ab1) at jacobi2d.C:154
function CkIndex_Jacobi::_call_begin_iteration_void(void*, Jacobi*) (0x470d17) at jacobi2d.def.h:453
function 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

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

Detecting violations

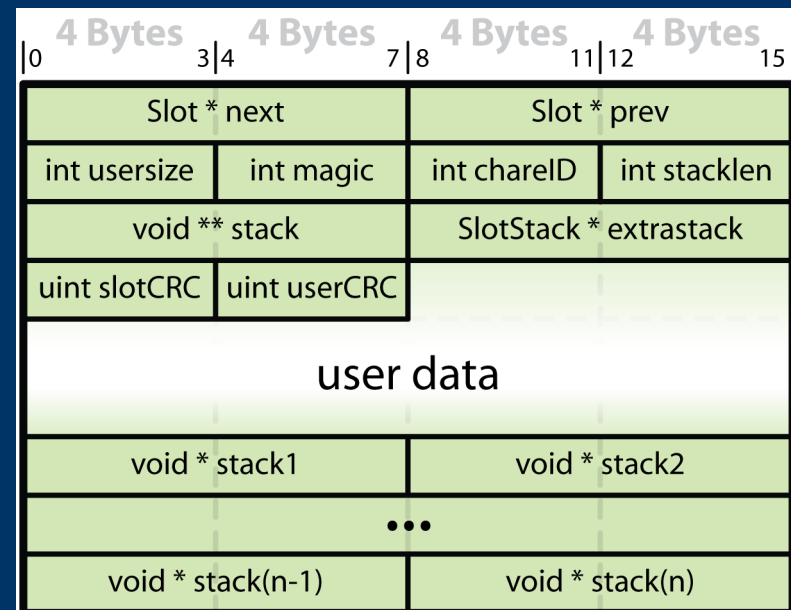
Associate to each memory block the ID of the owner char



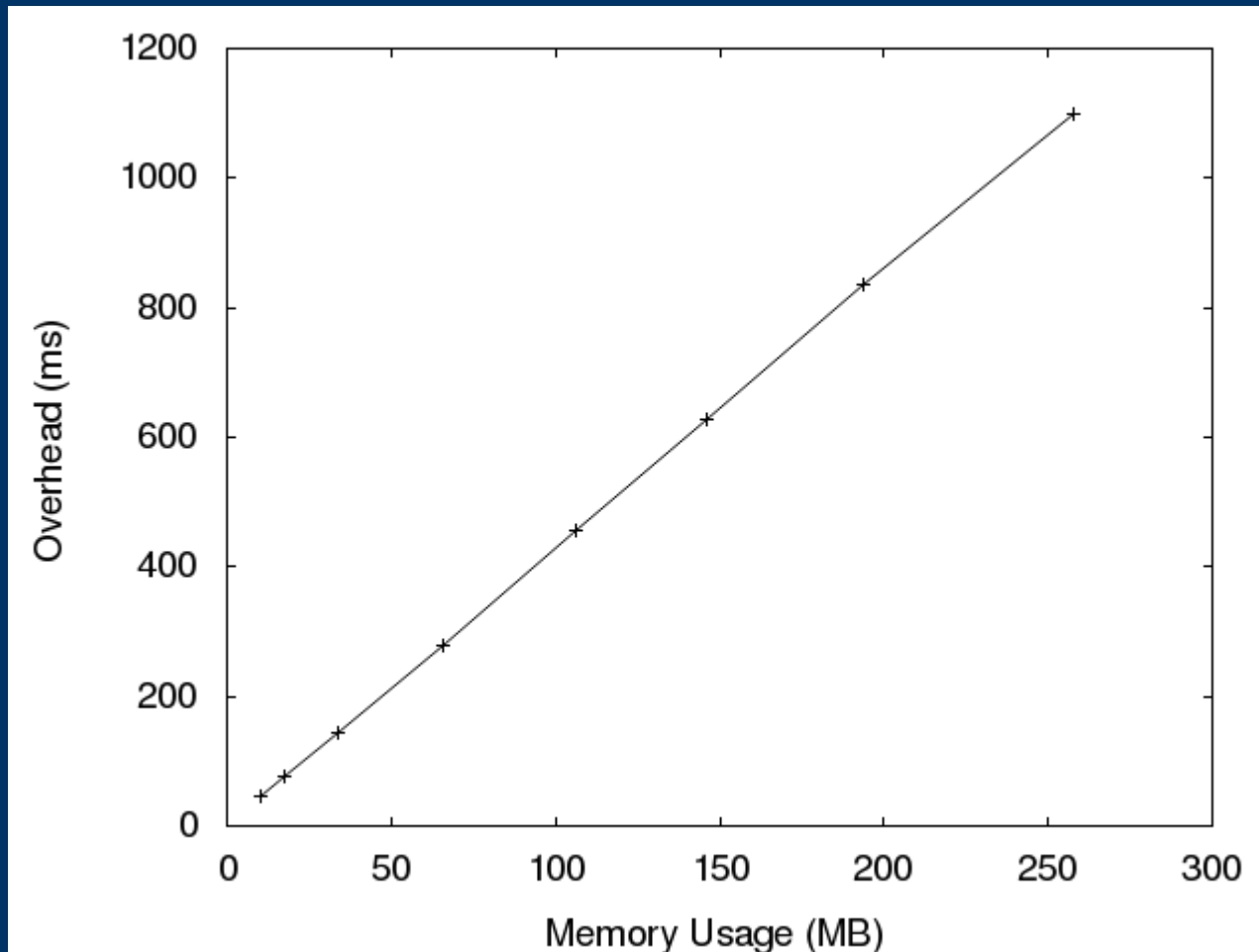
Detect modification to the memory block

Buffer overflow violations

- Utilize the other CRC field to protect the system data laying around the user data
- Detect violations similarly to cross-chare violations



Performance



$$1 + \frac{4.3 * M}{t}$$

M: allocated
memory

t: interval
between
methods

Future directions

- Reduce checking time
 - Reduce frequency of checking
 - Check only some entry methods
 - Reduce amount of memory scanned
 - Check only some memory
 - Reduce time to scan
 - Faster error detection codes
 - Memory shadowing
 - mmap and mprotect
 - allows detection of reads as well as writes

Future directions (2)

- Applying technique to real applications
- Presenting results of test to the user
 - Allow multiple ownership of data blocks
 - Filter errors (simple warnings)
- Re-execute erroneous code
 - Roll-back to state before message delivered
 - Re-run under more detailed debugging
 - Connected to live record-replay

Questions?

Thank you

<http://charm.cs.uiuc.edu/>