energie atomique • energies alternatives

MPC Allocator, developement status

Sébastien Valat

PhD student

Directeur de thèse : William Jalby

Encadrant CEA : Marc Pérache

Plan

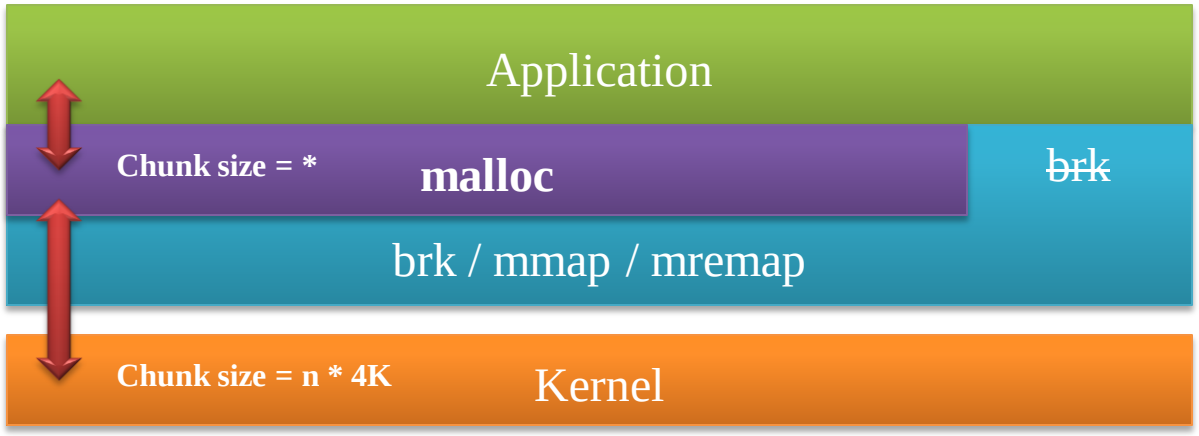
- **Remind malloc goal and constrains**
- **General design**
- **Focus on some points**
- **Debugging & profiling allocator**
- **Some results**
- **Current status**

energie atomique • energies alternatives

Background

Reminder on malloc

- **Kernel manage memory at page level (4K, 2M, 1G)**
- **Application need allocation of non restricted sizes.**
- **Malloc, an interface to hide pages :**
  - Request memory pages to the kernel.
  - Keep track of used and non used chunks.
  - Return the memory to the system when possible.



Subtle balance

energie atomique • energies alternatives

- **Malloc must :**
  - be as fast as possible, especially for C++
    (eg : 75M allocations on Hera)
  - save as much memory as possible
  - take care of the cache if possible

- **With the standard interface, malloc can't :**
  - predict the usage of allocated memory segments.
  - move an allocated segment

- **It must relies on heuristics for decisions**

Today issues

- **Becoming heavily multi-thread.**
  - Malloc must be parallel, not only thread-safe.
  - NUMA architecture
- **Number of cores tend to grow faster than memory.**
  - Memory per core will decrease.
  - Must take care of memory consumption.
- **Usage of some specific segments out of malloc :**
  - Shared memory
  - Locked pages (GPU, IB…)

energie atomique • energies alternatives

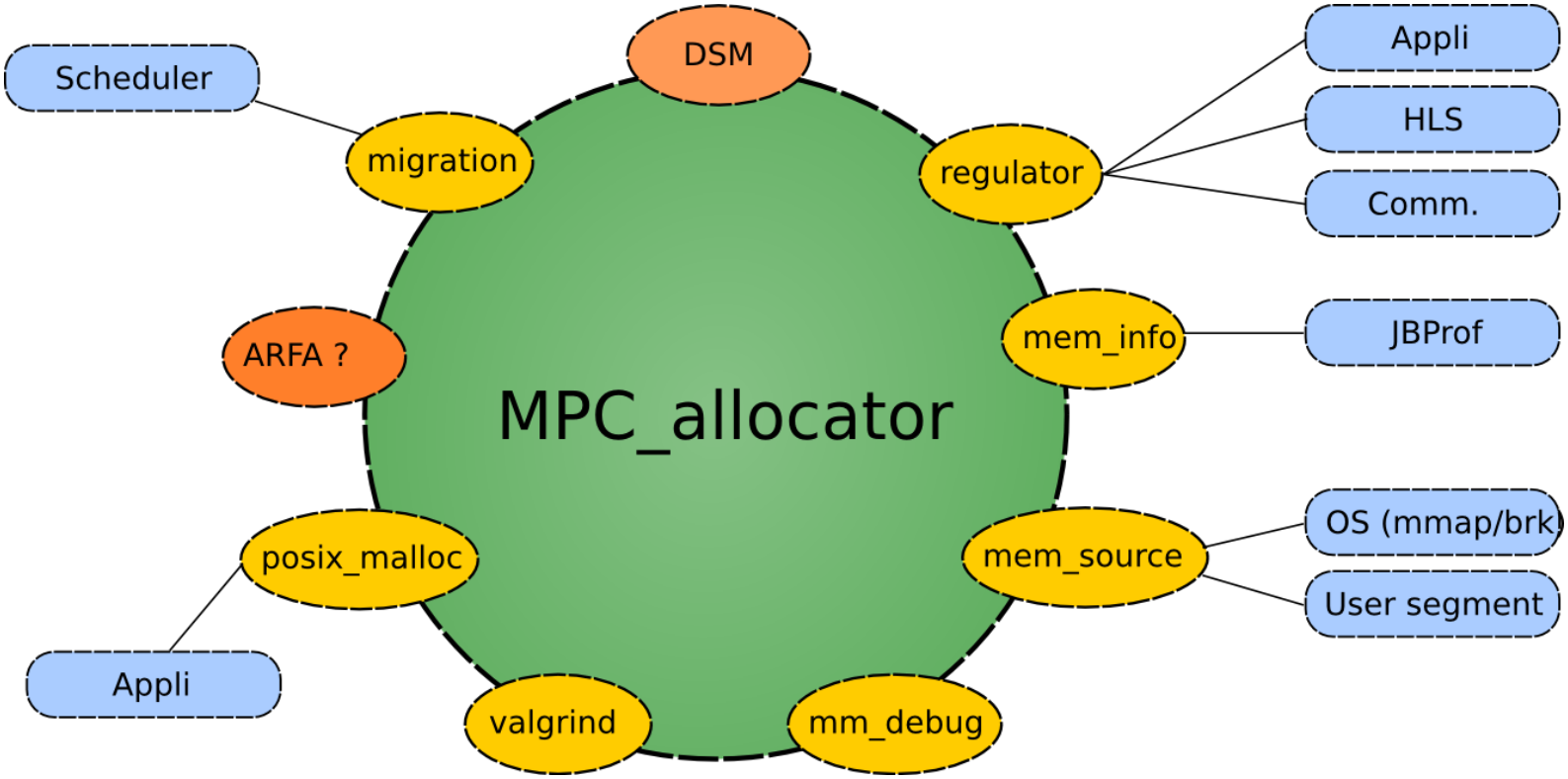
General design

Main objectives

energie atomique • energies alternatives

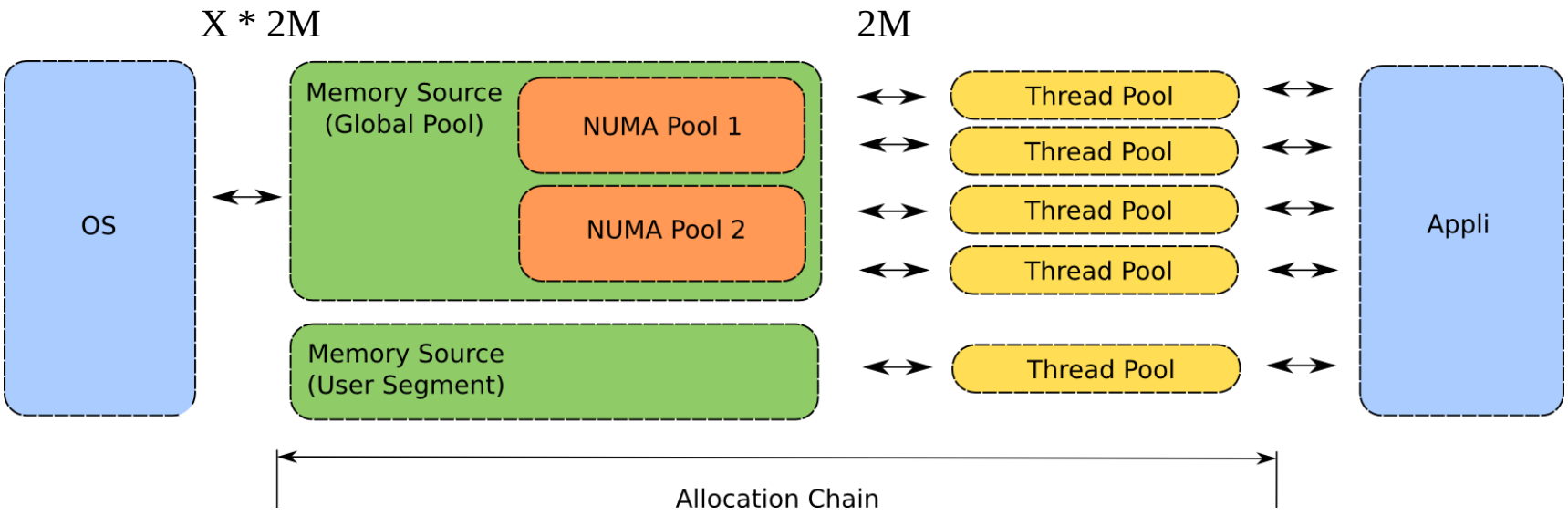
- **Must be integrated into MPC**
- **Minimal properties to maintain :**
  - Multi-thread
  - Support checkpoint restart
  - NUMA aware
  - Limit calls on *mmap* to keep performance.
- **New features :**
  - Manage user segments.
  - Grouped allocation / de-allocation
  - Adaptive memory consumption

Malloc and related components



General design

- **Define an allocation chain :**
  - A « Thread Pool » to manage non used chunks.
  - A memory source
- **An allocation chain per thread.**
- **Exchange by macros-blocs of 2M.**

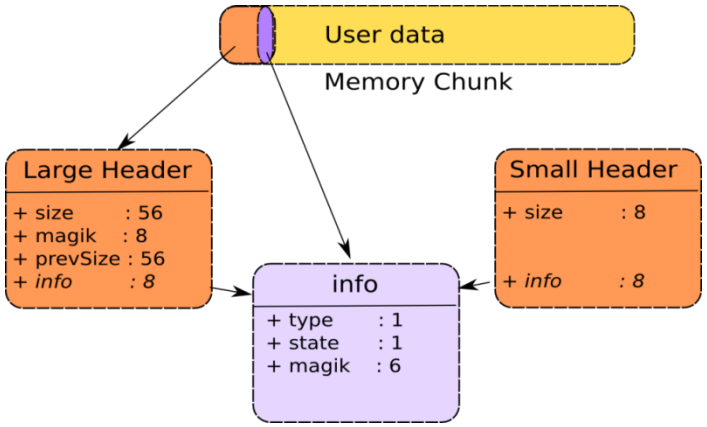


energie atomique • energies alternatives

Focus on some points

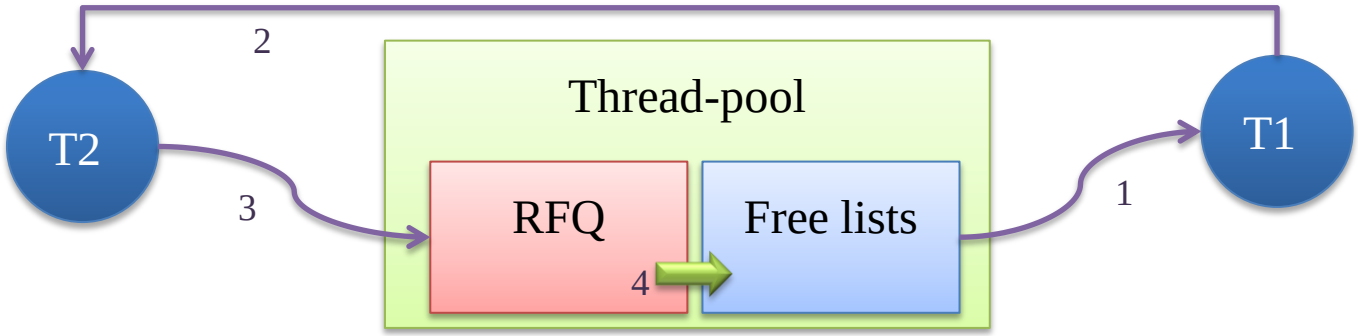
Thread-pools

- **Maintain lists of free chunks :**
  - 43 sub-lists for each size of 32B à 2MB
- **Up to now, partial definition of two class of blocs :**
  - Smaller than 64B *(not fully implemented up to now)*
  - Standard, larger than 64B
- **Thread-safety depend on a flag**
  - No locks for per thread chain
  - Thread safe for user segments



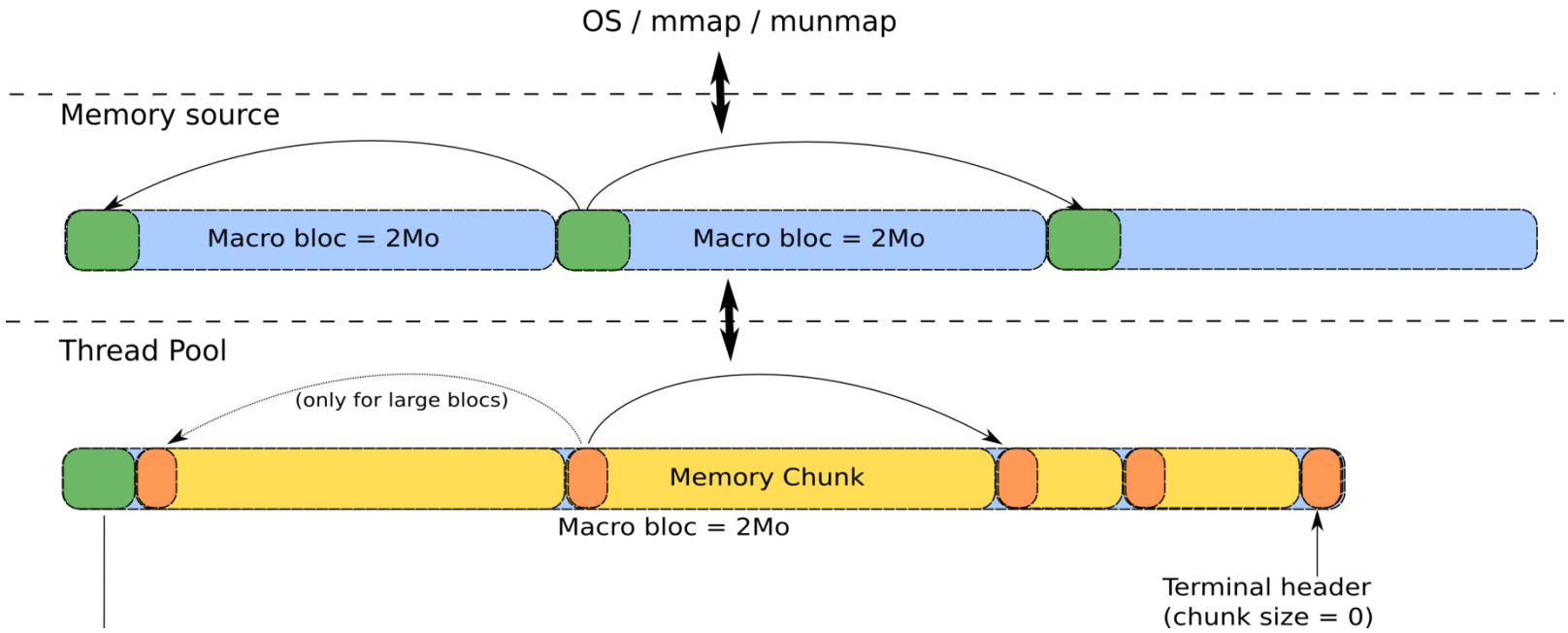
Remote Free Queue

- **Distant free :**
  - Chunk allocated by a thread.
  - Freed by another thread.
- **Main free lists didn’t be locked**
- **Provide a Remote Free Queue (RFQ)**
  - Distant thread register on it
  - Local thread freed the segment at next allocation/free.
- **Need specific management on thread exit (not done)**



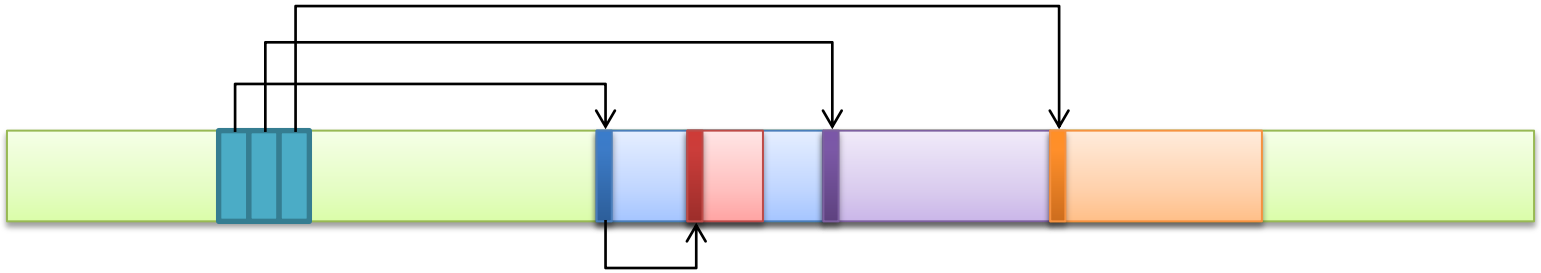
Memory source - Macros blocs

- **Manage by same code than thread-pools**
- **Bloc header contain the related allocation chain to know :**
  - Parent thread
  - NUMA node (not finish)



How to find macro-blocs in *free* method

- **For standard, be aligned to 2M limit, so easy**
- **But not for user segments**
- **Currently defined by an indirection map**



- **The map require 4MB per 1TB used**
- **Split into sub-maps of 2MB**

Why user segments ?

- **Special segments**
  - User allocate it and set some system properties
  - But need to manually allocate objects in this segment
  - Locked pages, shared memory…
- **Quickly instantiate a buffer**
  - Local allocator to reuse segments in list implementation
- **Avoid to re-implement allocation mechanisms everywhere**
- **Keep centralized information on memory usage in apps.**

Usage of allocation chain

energie atomique • energies alternatives

- **To manage a user segment :**

```
struct sctk_alloc_chain chain;
void * buffer = mmap(……);
sctk_alloc_chain_user_init( &chain , buffer , BUF_SIZE);

void * ptr = sctk_alloc_chain_alloc( &chain , 42 );

sctk_alloc_chain_free( &chain , ptr );
```

- **Add a « Memory source » to the chain :**

```
struct sctk_alloc_mm_source_default source;
sctk_alloc_mm_source_default_init(&source);
chain.source = &source;
```

Select default allocator

- **We can temporary change the default allocation chain :**
  - Default allocation on a user segment
  - Aggregate some particular allocations (same lifetime…)
  - Isolate buggy allocation sequences in external functions
  - Replace a buggy allocation chain ?
  - Thanks to TLS it is per-thread definition

```
sctk_set_chain_as_default( &chaine );

//malloc, new, new ……

sctk_restore_default_chaine();
```

energie atomique • energies alternatives

Debugging / profiling

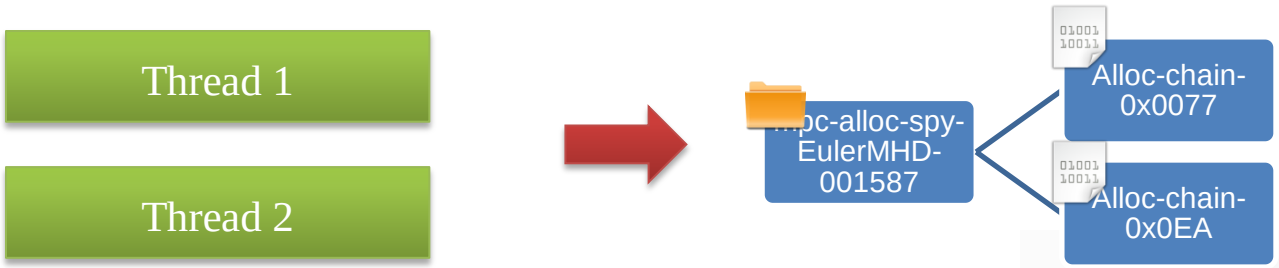
How to debug allocation implementation ?

- **About allocator debugging :**
  - Hard to find bugs in real applications
  - Hera done 75M of allocations what if the last one is buggy due to the 1st one ?
  - May fail later than the real bug and elsewhere.
  - printf also call malloc, so forbidden.
  - Gdb ? How to interpret when pointers are corrupted ?

- **How develop such code ?**
  - Done unit test to remove errors in controlled environment (~160 tests).
  - Provide a way to quickly replay a buggy sequence in vitro.
  - Dump allocator state in readable format for manual analysis.

Allocator spy system

- **To optimize, we need to measure**
- **Provide events trace of internals :**
  - Thread pool events
  - Memory source events
  - Posix interface



- **In future, may use MPC trace system**
  - But need to solve the initialization issue (*egg_allocator*)
  - Convert the trace reader.

Trace usage

- **Provide an interactive command line tool :**

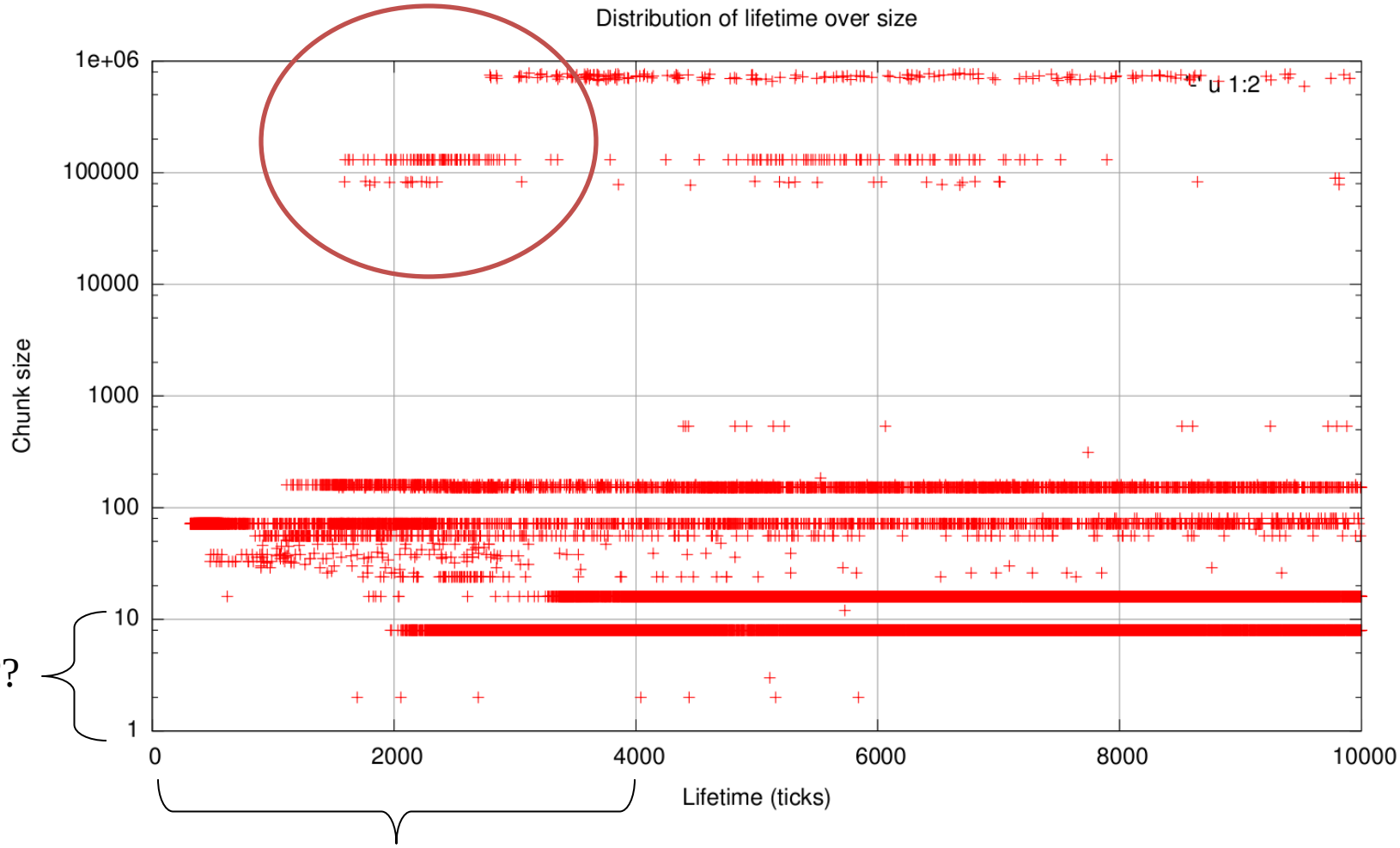
```
test > chain --all -stat
======================================== STAT MAX ============================================
chain           | requested_memory        | Real_memory             | Virtual_memory
----------------+-------------------------+-------------------------+-------------------------
0xc0000020      | 831 Mo 768 ko 196 o     | 1 Go 144 Mo 952 ko      | 1 Go 196 Mo
0xc001f020      | 372 Mo 922 ko 153 o     | 604 Mo 784 ko 656 o     | 630 Mo
...

chain           | nb_alloc                | nb_realloc              | macro_bloc_header
----------------+-------------------------+-------------------------+-------------------------
```

- **May help to debug and optimize allocator memory usage**
  - Permit to know where the memory stay.
  - Which size is loss by meta-data.
  - Get stats on free list usage.
  - Get details on fragmentation.
  - Trace history of a specific address.
  - May permit to replay to get deeper analysis (not done).

Global stats of allocations chains


```
test > chain --all -stat
 ======================================== STAT MAX ==========================================

chain           | requested_memory        | Real_memory             | Virtual_memory
----------------+-------------------------+-------------------------+-------------------------
0xc0000020      | 831 Mo 768 ko 196 o     | 1 Go 144 Mo 952 ko      | 1 Go 196 Mo
0xc001f020      | 372 Mo 922 ko 153 o     | 604 Mo 784 ko 656 o     | 630 Mo
0xc0022e20      | 408 Mo 927 ko 195 o     | 709 Mo 158 ko 272 o     | 718 Mo

chain           | overhead                | small_header            | large_header
----------------+-------------------------+-------------------------+-------------------------
0xc0000020      | 337 Mo 184 ko 652 o     | 1 Mo 62 ko 336 o        | 6 ko 48 o
0xc001f020      | 231 Mo 886 ko 503 o     | 1 Mo 11 ko 224 o        | 5 ko 240 o
0xc0022e20      | 300 Mo 255 ko 77 o      | 1 Mo 1 ko 752 o         | 5 ko 96 o
----------------+-------------------------+-------------------------+-------------------------
Total           | 3 Go 981 Mo 824 ko      | 13 Mo 842 ko 672 o      | 82 ko 240 o

chain           | rfq_cnt                 | rfq_requested           | rfq_real
----------------+-------------------------+-------------------------+-------------------------
0xc0000020      | 37                      | 176 o                   | 224 o
0xc001f020      | 25                      | 16 ko 256 o             | 16 ko 416 o
0xc0022e20      | 25                      | 16 ko 256 o             | 16 ko 416 o

chain           | nb_alloc                | nb_realloc              | macro_bloc_header
----------------+-------------------------+-------------------------+-------------------------
0xc0000020      | 5760834                 | 1575                    | 3 ko 128 o
0xc001f020      | 5782332                 | 1810                    | 1 ko 800 o
0xc0022e20      | 5065304                 | 1102                    | 2 ko 416 o
```

Example : extract lifetime per size



Size <= 8B ??

Lifetime <= 4000 cycles ??

energie atomique • energies alternatives

Some results on Hera

Hera with its internal allocator

- **MPC run with 16 threads – 1 process**
- **Let its internal allocator for small blocs enabled**

Allocator	Total time (s)	System time (s)	Physical Mem. (GB)
Jemalloc	136.3	9.2	2.6
MPC v99	143.2	10	2.7
MPC v2.2.0	139.8	7.7	3.6
Glibc	131.6	5.8	4.1
Tcmalloc	130.9	1.1	8.1
Hoard	Crash	Crash	OOM (>24GB)

- **Get large gaps in memory usage (up to 4 times)**
- **Gaps on system time with only 16 threads.**

Hera without its internal allocator

energie atomique • energies alternatives

- **MPC run with 16 threads – 1 process**
- **Remove internal allocator (increase to 75M allocations)**

Allocator	Total time (s)	System time (s)	Physical Mem. (GB)
Jemalloc	140.9	12.4	2.2
MPC v99	165.9	12.3	2.7
MPC v2.2.0	153.6	4.5	4.4
Glibc	147.4	4.2	3.7
Tcmalloc	137.6	2.1	3.7
Hoard	492.7	182.1	2.8

- **Previous OOM was due to bad interaction between the two allocators**
- **HOARD didn't work really better on this.**

To improve

- **General remarks**

Allocator	Remark
Glibc	Not parallel, only thread-safe
HOARD	Not enough robust
Tcmalloc	Fast, but not bound in memory
Jemalloc	Work well !!!!! Damn.

- **Were to improve**
  - None of them have explicit NUMA support.
  - Jemalloc be aggressive on *mmap* & *munmap*.

    it may lead to performance issue on large-scale systems.
  - Adaptive memory consumption.

    Why to free when we have memory ?

    But must be more aggressive when needed.
- **Others features (checkpoint, user segments…)**

What to improve in current code

energie atomique • energies alternatives

- **Currently use large headers for all bloc size**
  - Not an issue for memory consumption (~13MB on Hera)
  - But could impact caches
  - Solution : support small block allocation (not done)
  - Reduce virtual memory on huge blocs (not done)
- **Add a decision component at memory source level to keep memory or free it**
  - Manually test extreme policies and observe positive reactions.
- **Remove large / small header distinction or optimize current inefficient implementation.**
- **Alignment of huge blocks (currently aligned to 2M)**

energie atomique • energies alternatives

Developement status

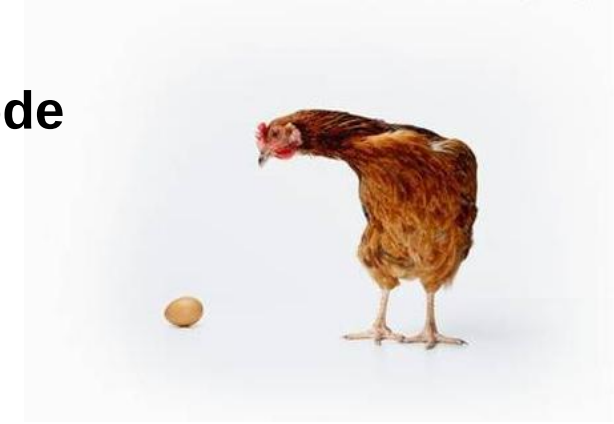
Progression

- **Used by LD_PRELOAD as not merged into MPC up to now :**

```
LD_PRELOAD=./liballoc.so startx
```

- **Can successfully run :**
  - In multi-node : Hera et EulerMHD on 4 nodes, 32 cores
  - Gimp, OpenOffice, VirtualBox, kernel compiling …
  - Full KDE 4 session

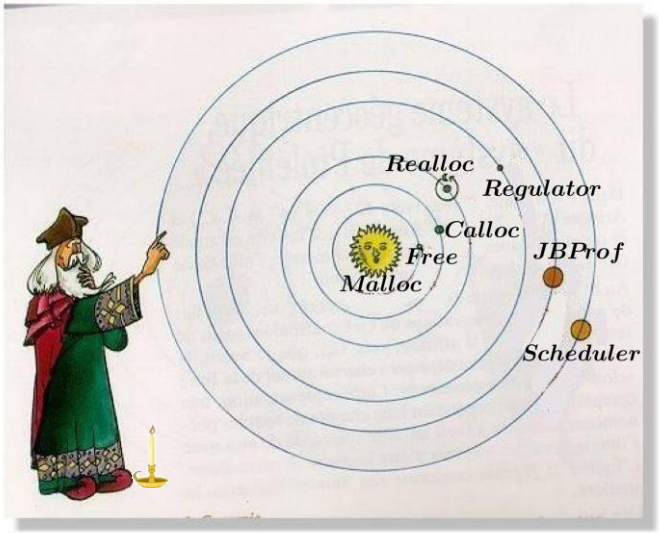
- **Can run with MPC in LD_PRELOAD mode**
  - Use a specific initialization routine
  - Use spinlocks as much as possible (mutex not available at startup)



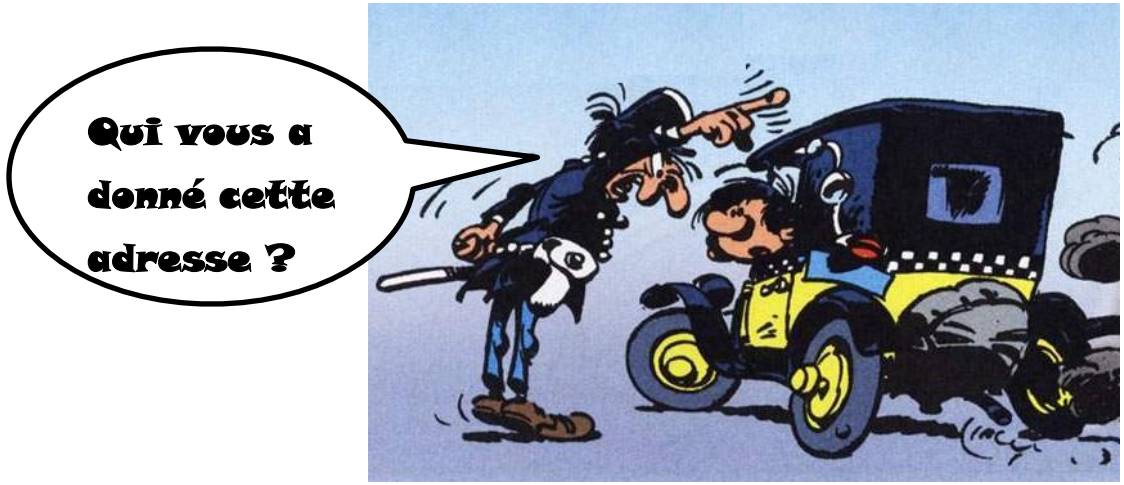
Work to do

- **Finish to optimize current implementation**
  - Small blocs support.
  - Padding of large arrays.
  - NUMA support.
- **Extend**
  - Adaptive mode (already manually played with parameters)
  - Integration in MPC and support checkpoint / restart
  - Valgrind support
  - Auto-tuning of free list size ?
- **Improve trace analysis system**
- **…**

THANKS


Realloc
Regulator
Calloc
Free
JBProf
Malloc
Scheduler


M'ENFIN !
COMMENT VEUX-TU ?!?
Il y en a qui attendent 0x3A478 comme réponse à malloc(0) !!! ??


Qui vous a donné cette adresse ?

energie atomique • energies alternatives

BAKUPS

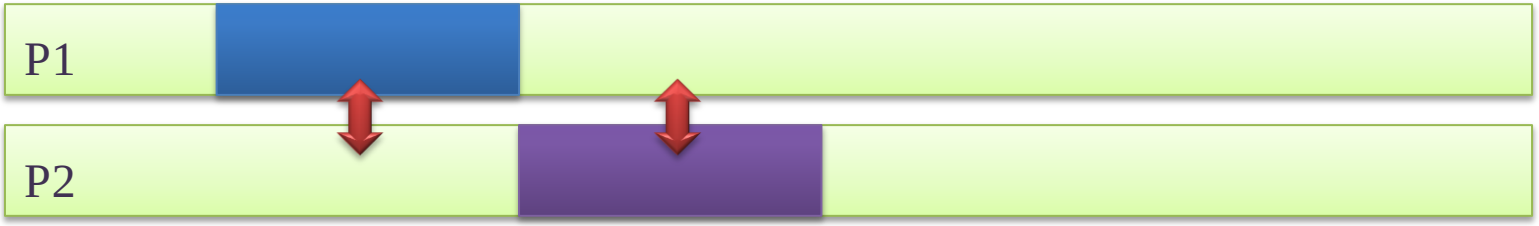
Tested allocators

energie atomique • energies alternativ

Allocator	Source	Particularity
GLIBC	Linux default one	Only thread-safe, production
Jemalloc	FreeBSD default one	Parallel, production
HOARD	PhD of Berger E.	Research parallel allocator
TCMalloc	Google allocator	Parallel, production in some critical apps (mysql).
MPC 2.2.0	Current MPC allocator	Parallel
MPC v99	My allocator	Parallel (in development)

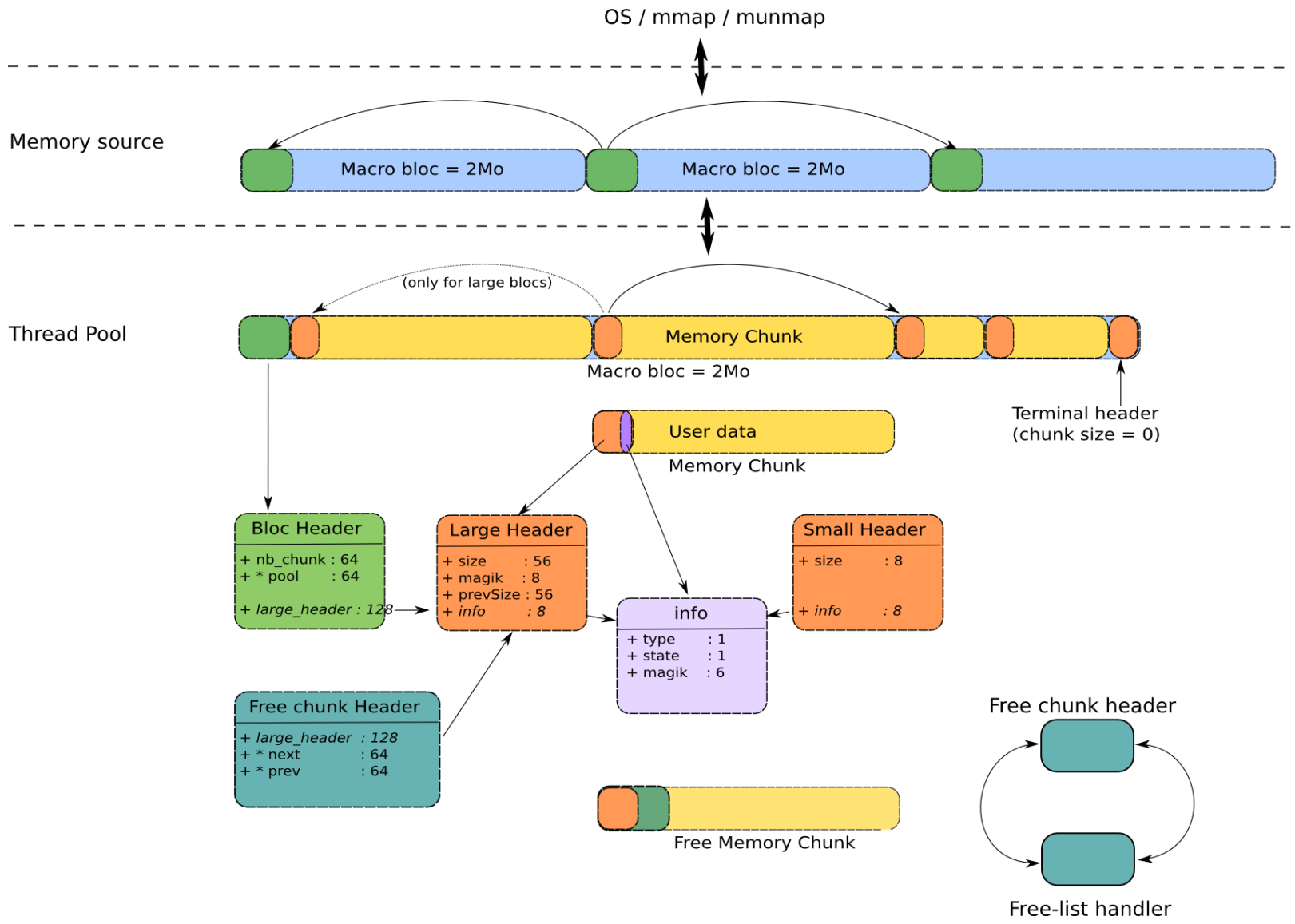
Migration constrain

- **Migration require unique addresses on all nodes**
  - Be at limit of virtual address space on Tera 100
  - So must prohibit migration between some nodes in future
- **Must force addresses returned by *mmap***



- **Need to provide *mmap* wrapper to avoid collision with libs**
  - Eg : New CUDA driver force *mmap* addresses
  - What about the loader and dynamic libraries ?

Les structures manipulées



Hera with internal allocator