GIANT
INNOVATION CAMPUS

# Givy, a software DSM runtime using raw pointers

François Gindraud*

*UJF/Inria

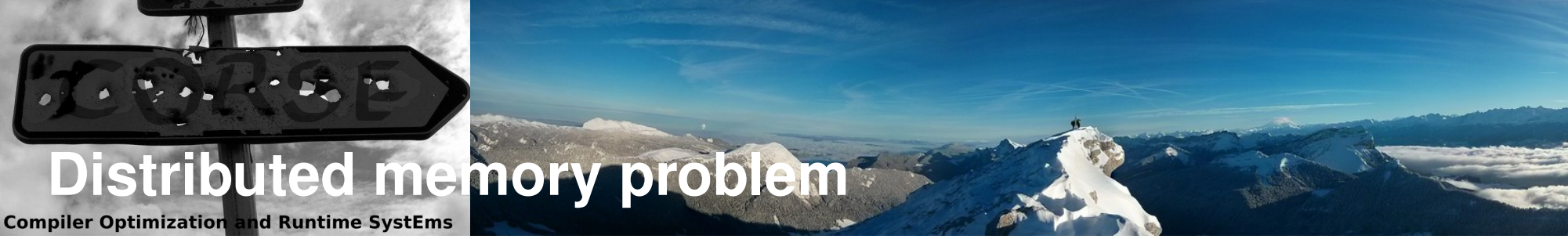
Compilation days (21/09/2015)

Inria informatics mathematics


# Outline

Compiler Optimization and Runtime SystEms

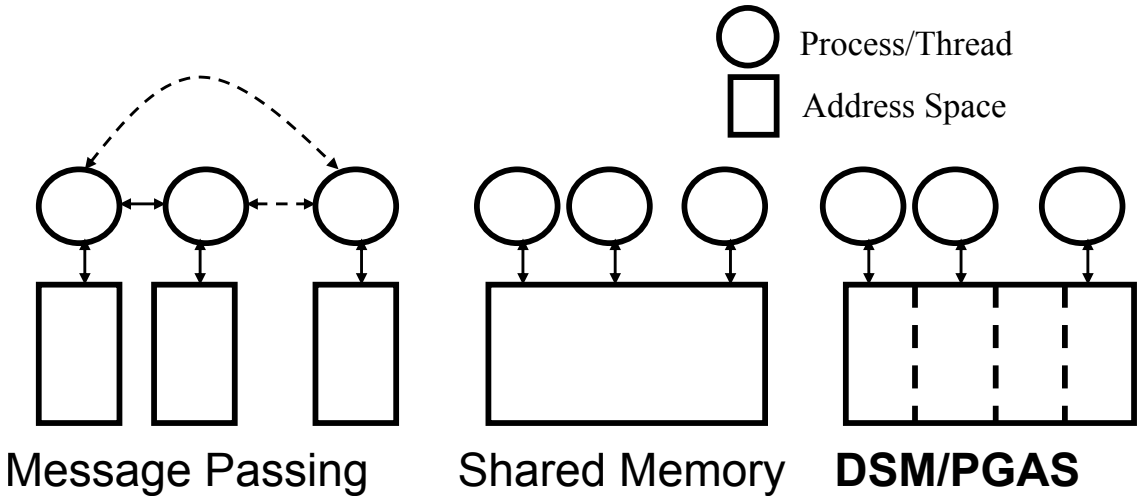

1. Distributed shared memory systems
   - Examples
   - Categories
2. Givy
   - Data model
   - Cache coherence protocol
   - Implementation
   - Dataflow task execution model



# Distributed memory problem

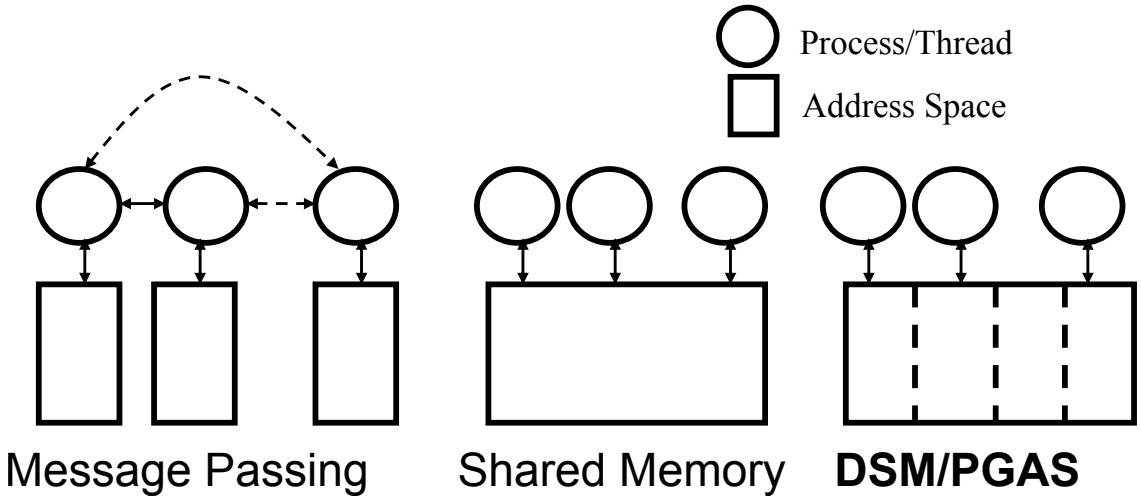
Compiler Optimization and Runtime SystEms

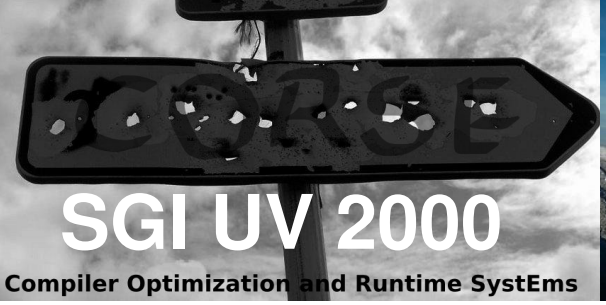
- Message passing is hard
  - Irregular communication patterns
  - Moving structured data in different address spaces
- Shared memory is easier
  - Implicit communications
  - Same address space for everyone



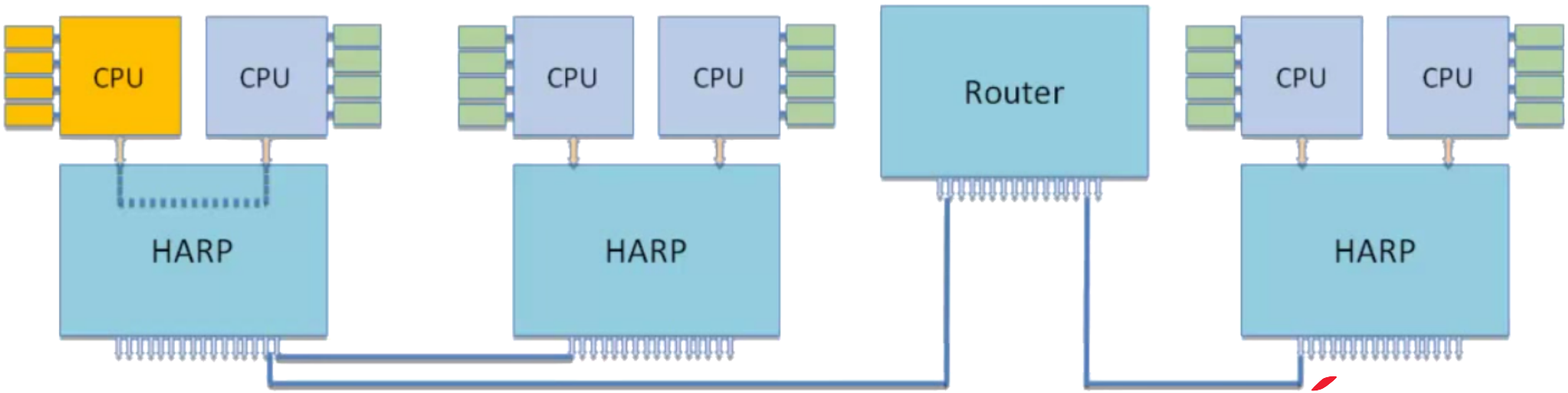
# Distributed shared memory

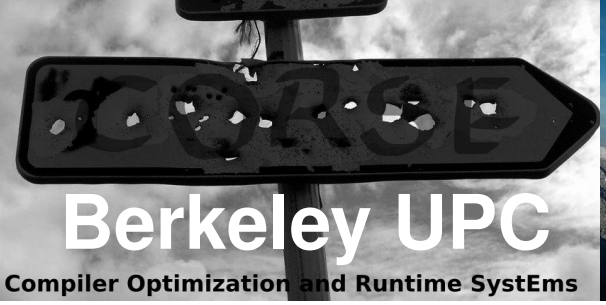
- Idea: simulate a shared memory over the distributed memory
- PGAS: DSM with explicit difference between local and remote accesses






- NUMA architecture (hardware transparent GAS)
- Transparent communications (triggered at cache line access)
- Additionnal library API for block transfer
- No data migration (placement on first requesting node)

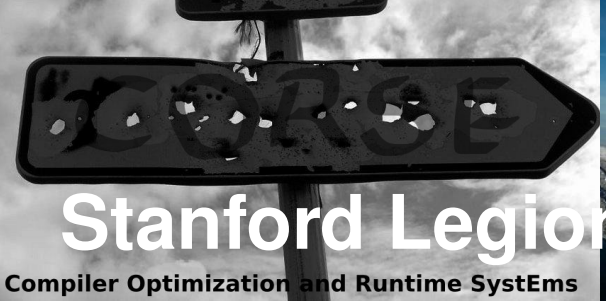




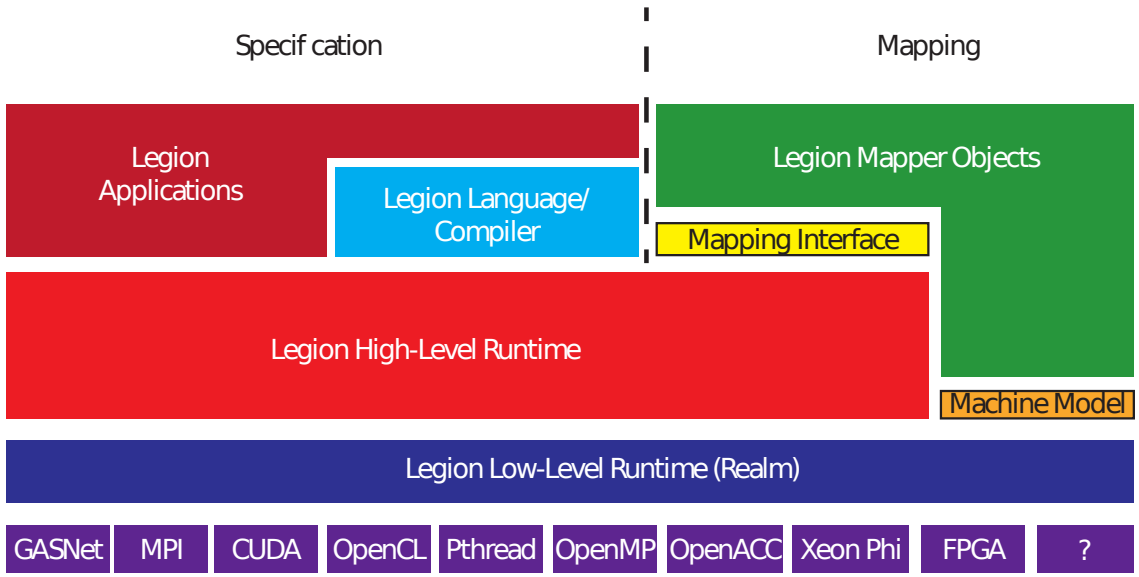

- PGAS C extension (requires compiler support)
- SPMD execution model
- No data migration
- Built on the GASnet runtime

## Code

```
int a[16]; // local variable
shared int b[16]; // gas variable
shared [4] int c[16]; // with explicit layout
b[3] += 42; // generates communication code
upc_memcpy(b, c, 16 * sizeof (int)); // gas version of memcpy
shared int * gas_ptr; // fake pointer (node, vaddr, phase)
```



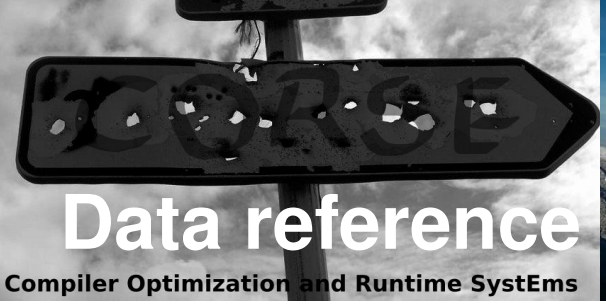

- C++ runtime library (with optionnal DSL)
- Structured data, referenced through a tuple space
- Access is done by replication ; communications and placement are implicit in the DSL, explicit in the runtime




# Support type

Compiler Optimization and Runtime SystEms

- Hardware (NUMA machines) / OS (software NUMA)
  - Transparent
  - Not portable, high performance still requires programmer action
- Compiler
  - Can generate communication code
  - New language to learn, need to port the compiler
- Library
  - Most portable, easy to adopt
  - Manual API calls




# Data reference

Compiler Optimization and Runtime SystEms

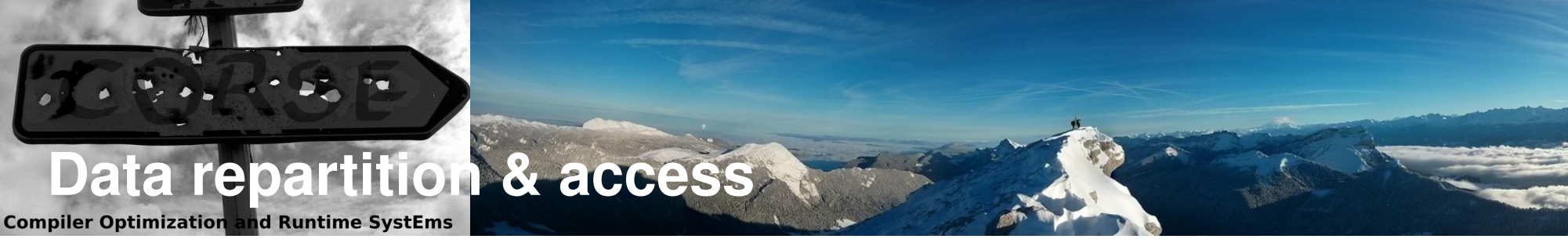
- Real raw pointers
  - Must be valid on every machine or translated
- Tuples
  - Array with integer offsets (UPC)
  - Tuples as keys in tables (Legion, CnC)
- Abstract objects
  - C++ fat pointers
  - Object-based DSM (CORBA)


# Data granularity

Compiler Optimization and Runtime SystEms

Memory size at which the DSM operates:

- Fixed
  - Usually page or cache line level
  - Waste of bandwidth and possible false-sharing if data is smaller than granularity
  - Reconciliation can eliminate false sharing, at the cost of computation
- Dynamic
  - No false sharing
  - Metadata layout is more complex (multiple sizes)
  - DSM knows the data size exactly




# Data repartition & access

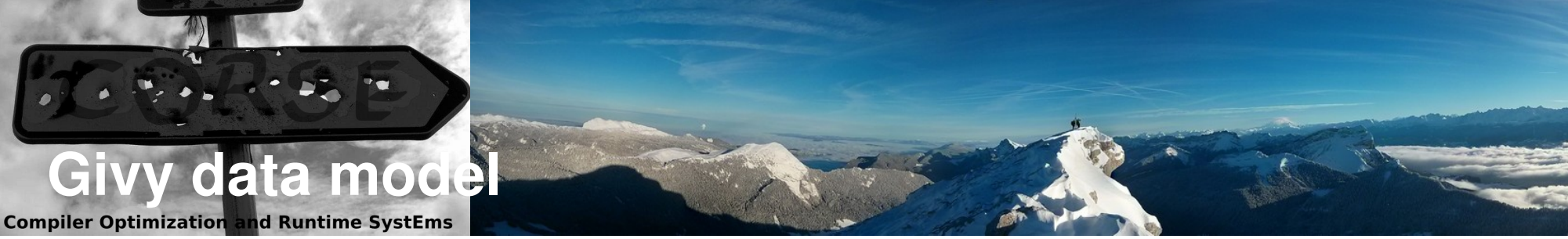
Compiler Optimization and Runtime SystEms

- Replication & local access
  - Manual copy management
  - Cache coherence protocol
- Partitionning & remote access
  - Explicit vs Implicit partitionning
  - Static placement vs migration


# Design choices

Compiler Optimization and Runtime SystEms

- Library support
- Real pointers
  - No translation
  - Compatible with C code
  - Requires virtual memory support
  - Gives programmer control of actual memory layout
- Dynamic granularity
- Data replicated
  - Copies managed by a software cache coherence protocol



# Givy data model

Compiler Optimization and Runtime SystEms

## Allocation/deallocation

```
void * malloc (size_t size);
void free (void * ptr);
int posix_memalign (void ** ptr, size_t align, size_t size);
```

## Access

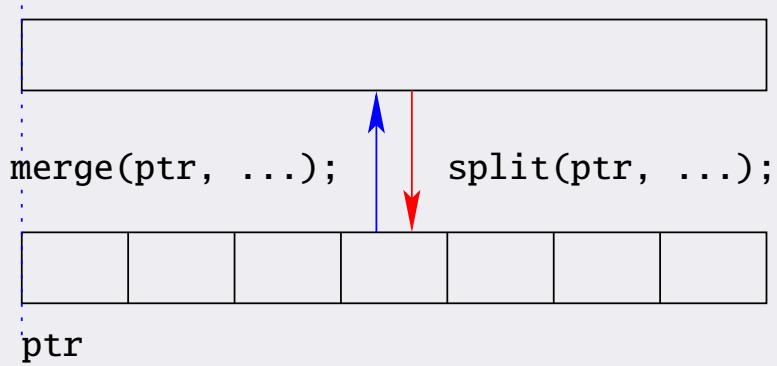
```
void require_access_read (void * ptr); // can read
void require_access_rw (void * ptr); // can read and write
/* after calling require_access_* (ptr), the region of ptr is
 ↪  available in local memory at ptr */
```


# Data races

Compiler Optimization and Runtime SystEms

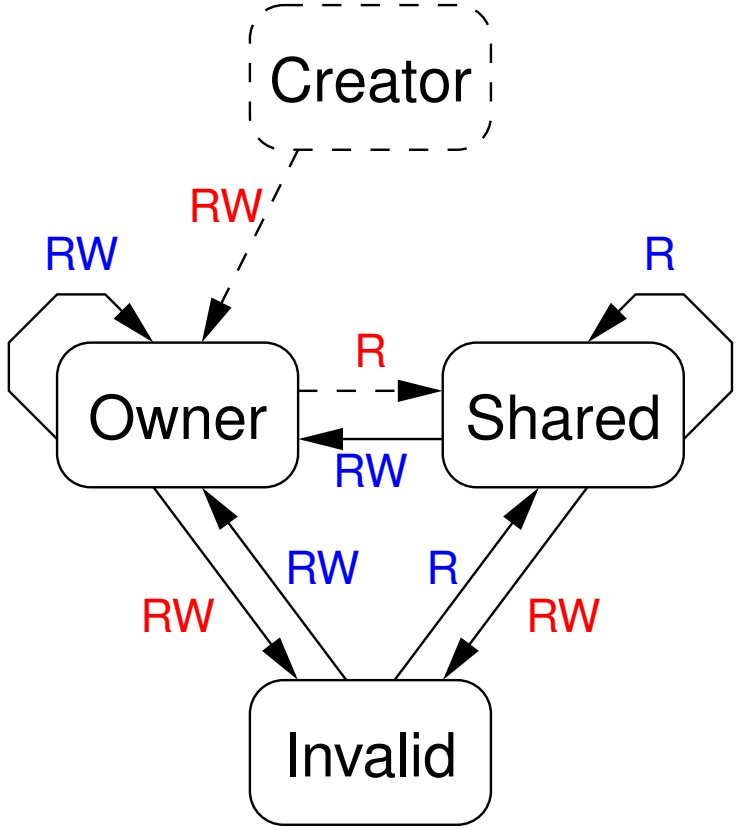
- Accesses to a region must be data race free
  - Each pair of accesses (RW,RW) or (RW,R) must be ordered
  - Need of external synchronisation (barriers, control message passing, task runtime)
- Useful data races:
  - Mutexes : extension ?
  - Multiple writers (merge of results) : region splitting

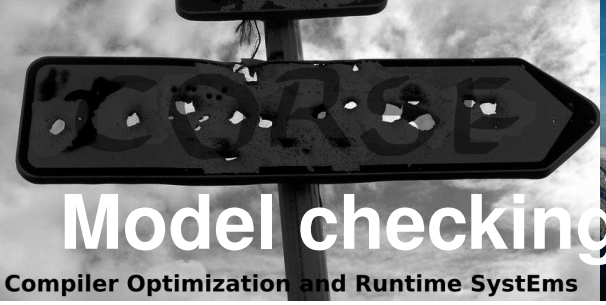
## Region splitting




- Split and merge semantically destroy their input regions and generate new ones (keeping the data intact)
- Requires RW access for all inputs
- Splitting position is a runtime choice

- Each region tag its node copies with a state:
  - Owner: always one, has exclusive write access
  - Shared: has valid copy (but no owner)
  - Invalid: no valid copies
- Directory
  - Owner stores the set of Shared nodes (invalidations)
  - Creator (constant) points to current Owner






# Model checking

Compiler Optimization and Runtime SystEms

## Cubicle

- Model checker for parametric problems
- Tries to generate invariants (to have a finite checking)

## Property

For every set of data race free accesses, every task reads from the last writer in synchronisation order

## Result

- >100hrs (not finished)
- → try to guide the model checker towards the data race free property


# Libc `malloc()` problem

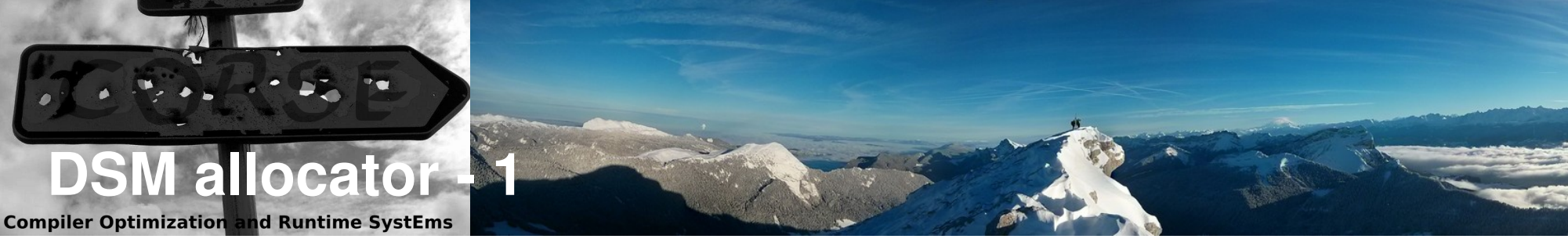
Compiler Optimization and Runtime SystEms

## Problem

- Real pointers as DSM references → Local memory is not private
- → Libc `malloc()` allocates in the GAS, and may generate collisions with other node's `malloc()`
- Libc `malloc()` could collide with any custom allocator for Givy, unless this allocator preallocates all the GAS virtual memory (not desirable)

## Solution

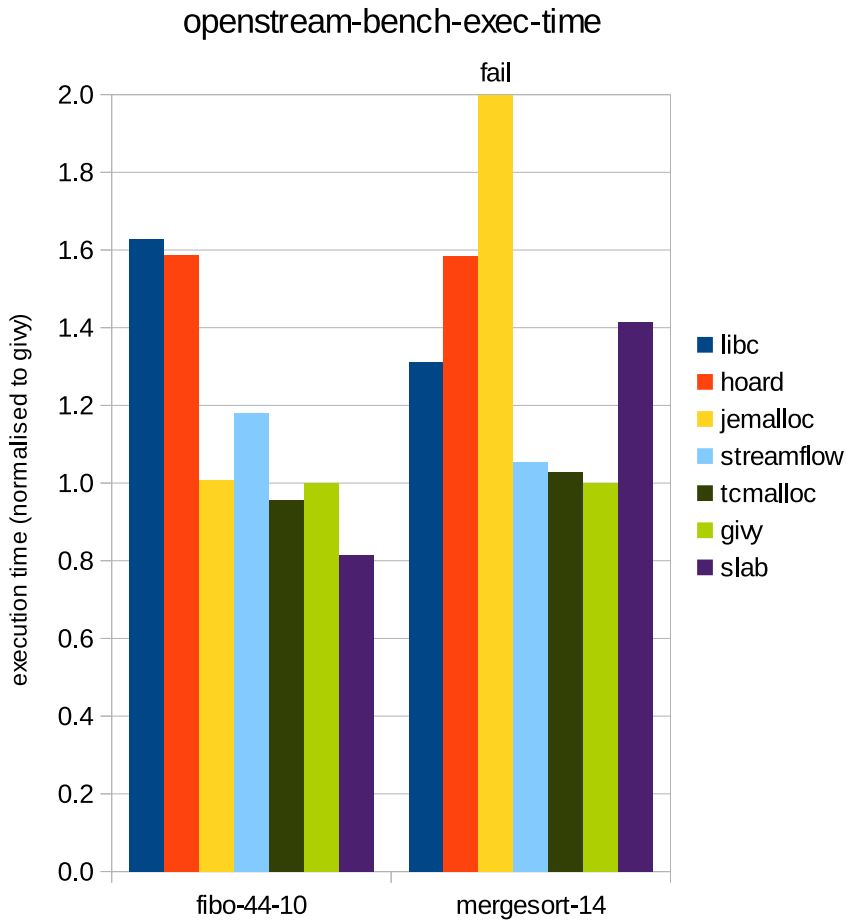
Replace the libc `malloc()` by an allocator that prevent collisions at the DSM level, and allocate everything in GAS




# DSM allocator - 1

Compiler Optimization and Runtime SystEms

- GAS-level collision-free property
  - Cut the GAS into *nb_nodes* segments
  - Have the n-th node `malloc()` allocate from the n-th segment exclusively
- The segment constraint has little effect on performance




# DSM allocator - 2

Compiler Optimization and Runtime SystEms

- Advantages to a single allocator
  - Simpler interface (UPC has 3 different malloc interfaces internally)
  - Better memory ressource control
  - Avoid wasting memory sitting in multiple allocator's caches.
- Coherence metadata integration
  - Must be stored somewhere and referenced quickly from the pointer alone
  - Must be flexible enough to support split and merge
  - Allocator already have structures to find metadata (to perform `free(ptr)`)


- Combining a task runtime and a DSM...
  - Hiding latency while coherence fetches remote data
  - Synchronisation
  - Natural interfaces with explicit memory dependencies
  - Moving computation to data
  - Memory aware scheduling


Thank you for your attention.