Walk over 8 years of research in HPC

- Memory managent (tools)
- LHCb 40 Tb/s DAQ R&D

SÉBASTIEN VALAT – ATOS – 04 APRIL 2019

Plan

- Quick reminder: memory management performance impact
- MALT : A Malloc Tracer
- NUMAPROF : A NUMA memory profiler
- LHCb 40 Tb/s DAQ R&D effort for 2020 upgrade

UNIVERSITÉ DE VERSAILLES ST-QUENTIN-EN-YVELINES

Exascale computing research

Quick reminder memory managment impact

PHD. WORK

A little bit of bibliography

- **Interesting to read :**
  - What every programmer should know about memory (Urlich Drepper) https://people.freebsd.org/~lstewart/articles/cpumemory.pdf
- For all details from this presentation : look on my PhD. thesis :
  - PhD: https://hal.archives-ouvertes.fr/tel-01253537
  - PhD slides: https://svalat.github.io/docs/2014-07-phd-defense.pdf
  - Webcast of similar talk (FR): https://youtu.be/wp8cCTD7XDI

Context : HPC

July 2014

- **Supercomputers** for numerical simulations
- At CEA, **Tera 100 :**
  - **6e** from TOP 500 in 2010
  - **140 000** cores, **1.05** Pflops.
- **Massively parallel** machines
  - Now **million cores**
- More and **more cores** !

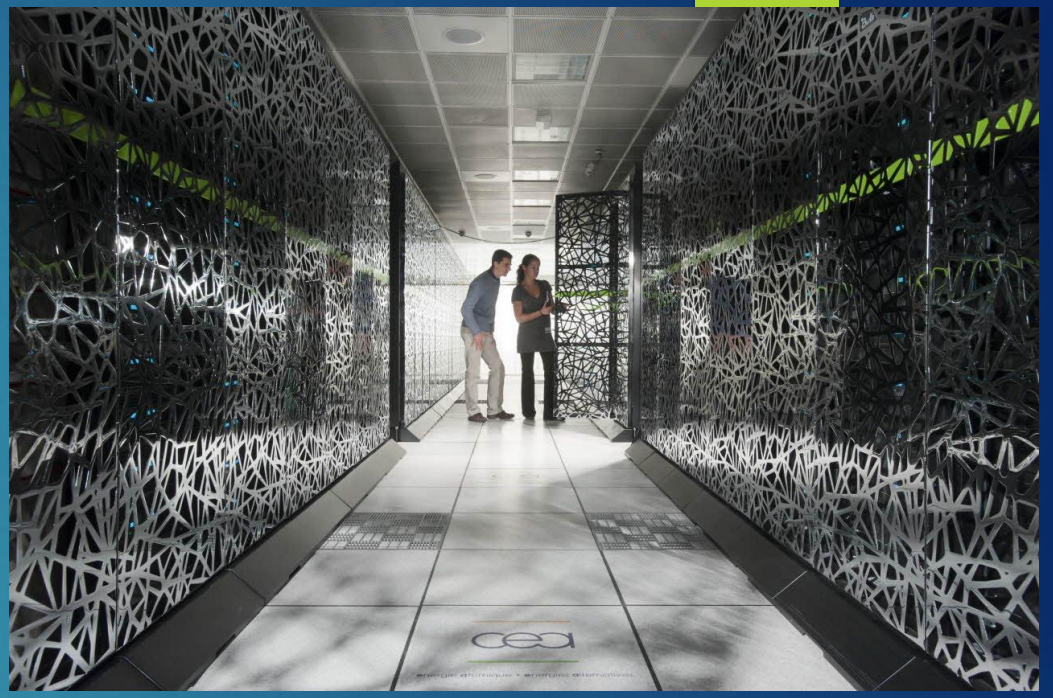
http://www.cea.fr/multimedia/Pages/galeries/defense/Tera-100.aspx

Architecture	Proc.	Cores	Threads
Tera 100 thin nodes	**4**	**32**	32
Tera 100 large nodes	**16**	**128**	128
Intel KNL	1	64	256
PEZY-SC2	1	**2048**	**16 384**

Context : HPC

July 2014

- **Memory** becomes a **critical resource**
- Growing impact on **performance**
- **Data movements :** speed gap CPU / RAM, **memory wall**.
- **Management** : now have to handle close to **TB** of memory
- Decreasing **memory per core**

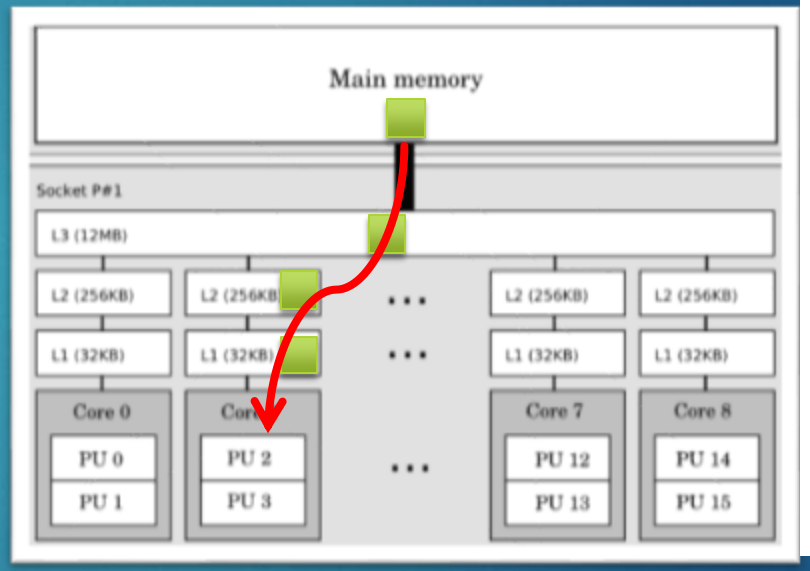


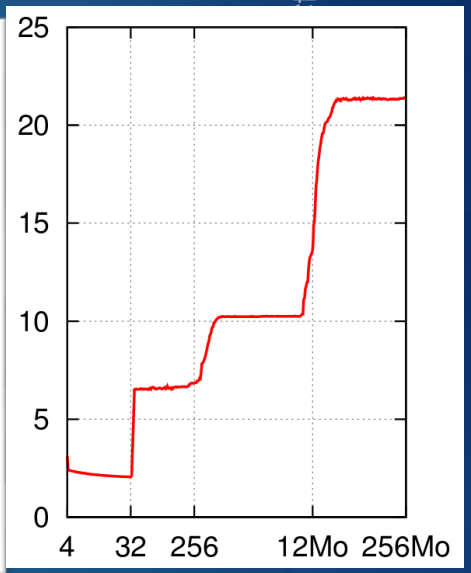
http://www.cea.fr/multimedia/Pages/galeries/defense/Tera-100.aspx

Architecture

- Computer science : **operations** & **data**
- Multiple **memory levels**
- Hierarchical **caches**
- *Pre-fetcher*

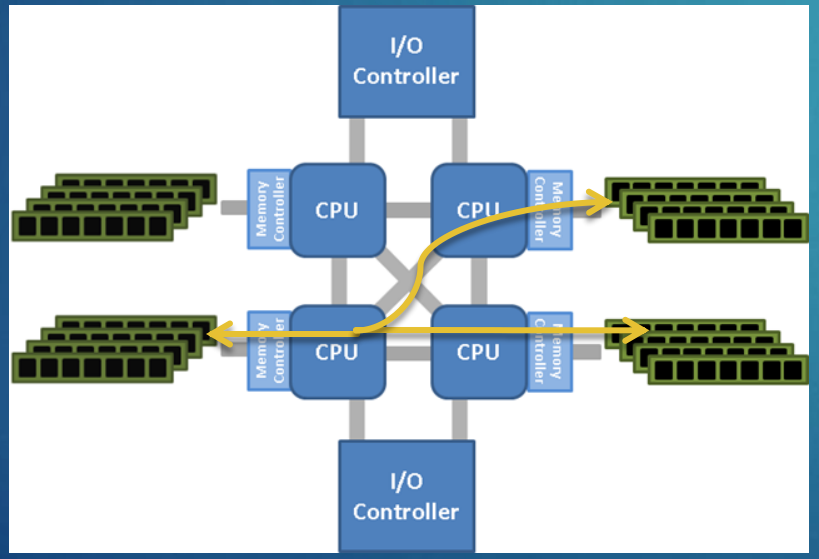
Processor : 8 cores

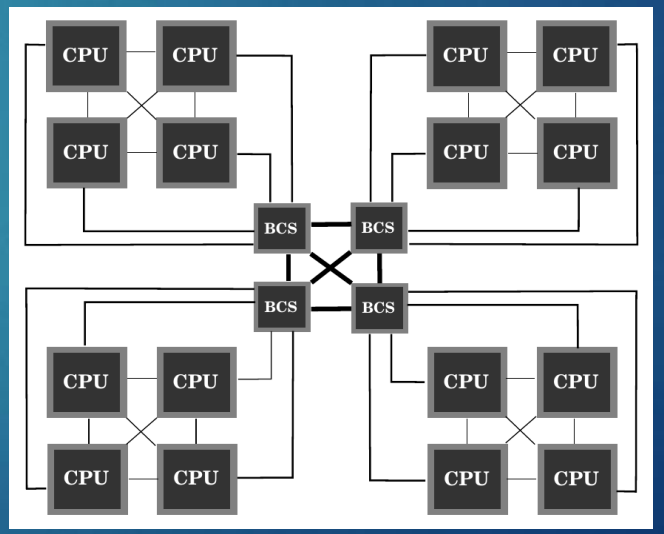




Architecture

- Hierarchical **memory**
- **Remote** / **local** memories **(NUMA : Non Uniform Memory Access)**

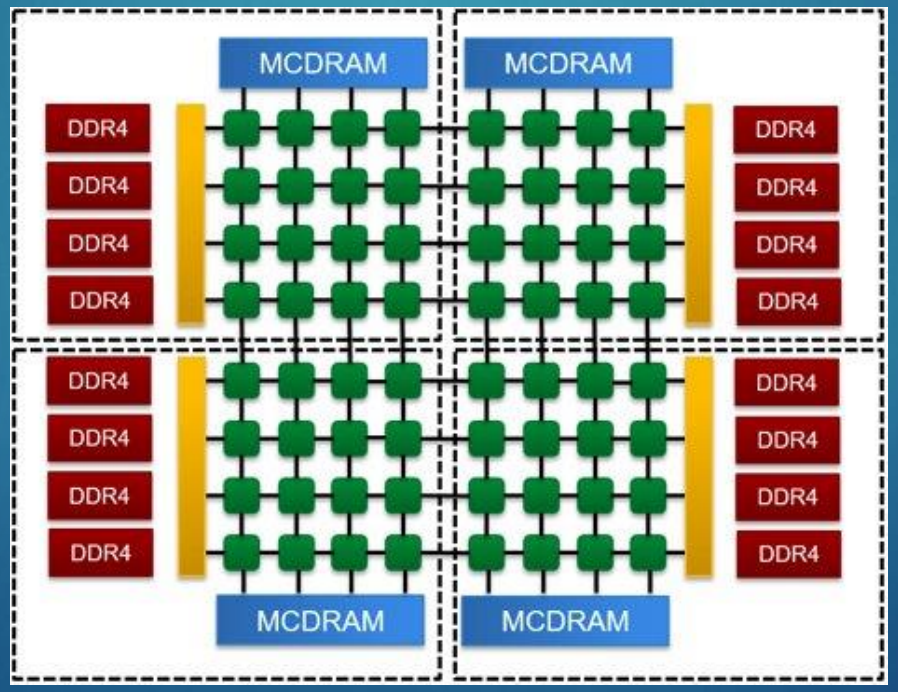




Large nodes : 128 cores (BCS)

Now also inside the CPU – Intel KNL

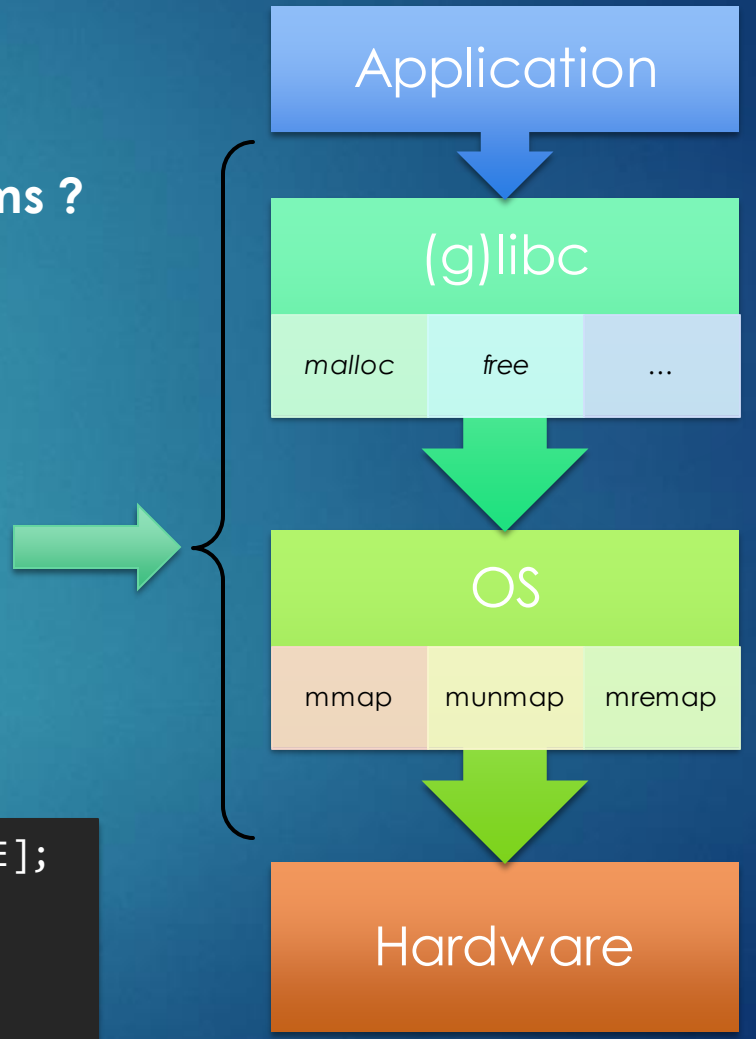
- Intel KNL (64 cores) can be configured in **2 or 4 NUMA domains**
- Also add **MCDRAM** (similar idea than GPU GDDR5) **viewed** as a **N**



- Or on **AMD** CPUs

Software memory management layer

- **Impact of memory management mechanisms ?**
- Involving **two components** :
  - User space ***memory allocator*** (malloc)
  - **Operating System** (OS)
- Focus on :
  - Impact on **allocation time**
  - Impact on **access efficiency** (placement)
- Malloc C or C++ interface :




```c
float * ptr = malloc(SIZE);
…
ptr = realloc(ptr,NEW_SIZE);
…
free(ptr);
```

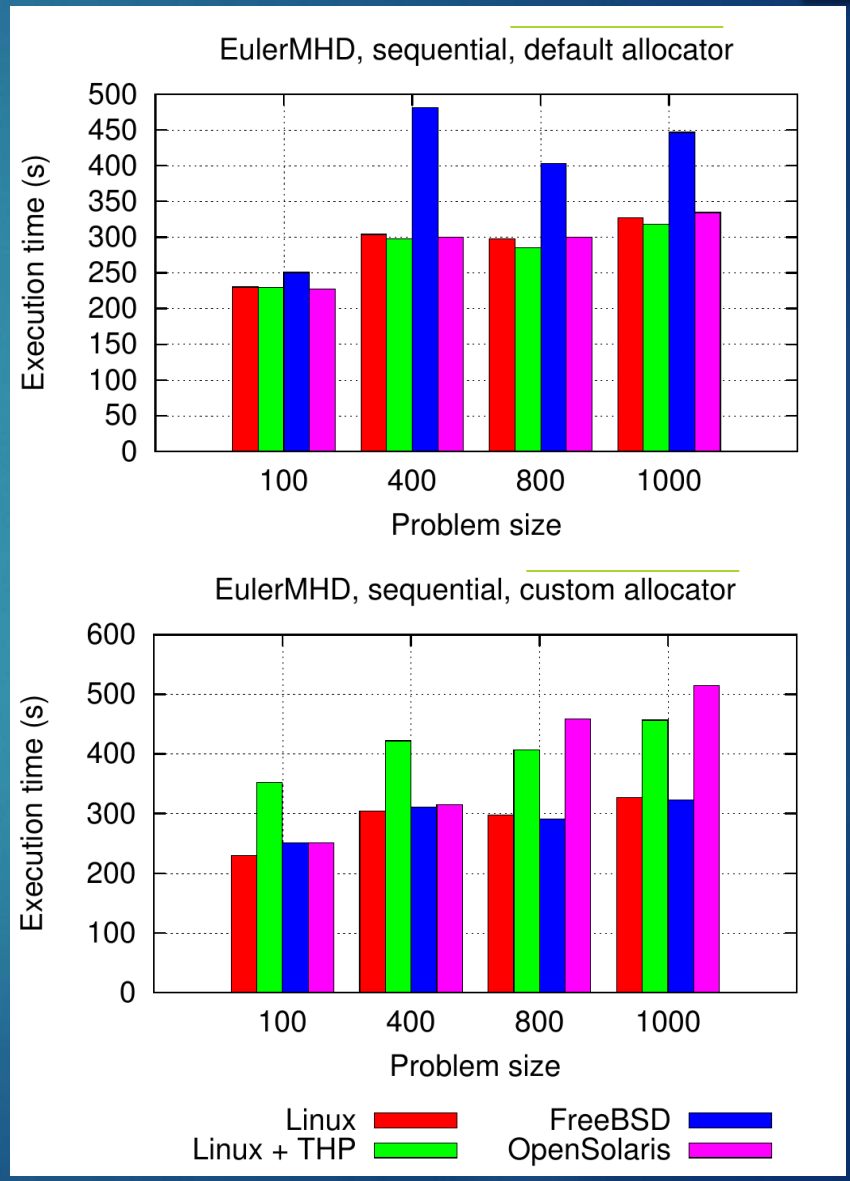
```c
float * ptr = new float[SIZE];
…
…
…
delete [] ptr;
```

EulerMHD issue

- **EulerMHD (CEA) :**
  - **C++ /MPI**
  - Magnéto-hydrodynamic **stencil code**
- **FreeBSD :** slowdown of **1.5x**, up to **3x** in **parallel**
- Impacted function only do compute.
- Function with **9 arrays pre-allocated** at init. :

```
for (i = 0 ; i < SIZE ; i++)
        x1[i] = x2[i] + x3[i] … + x9[i]
```

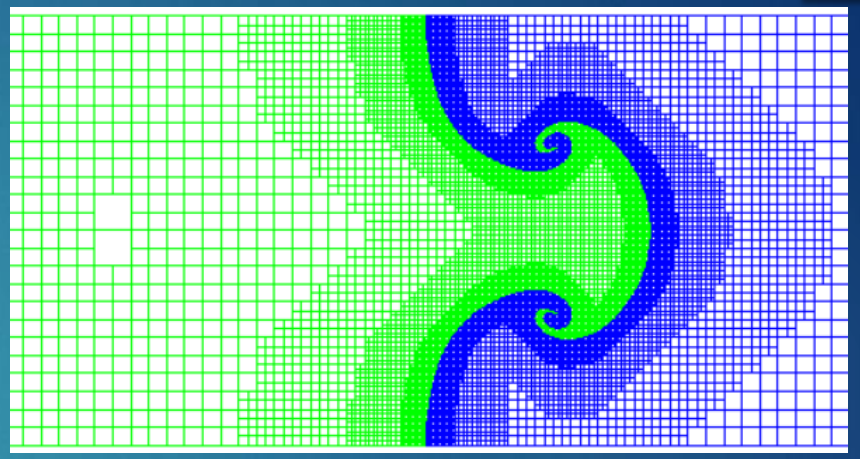
- Change between OS's :
  - **User space memory allocator** (malloc).
  - **OS paging policy**
  - ~~*(Scheduler)*~~
- Effect can be controlled by **changing the allocator.**

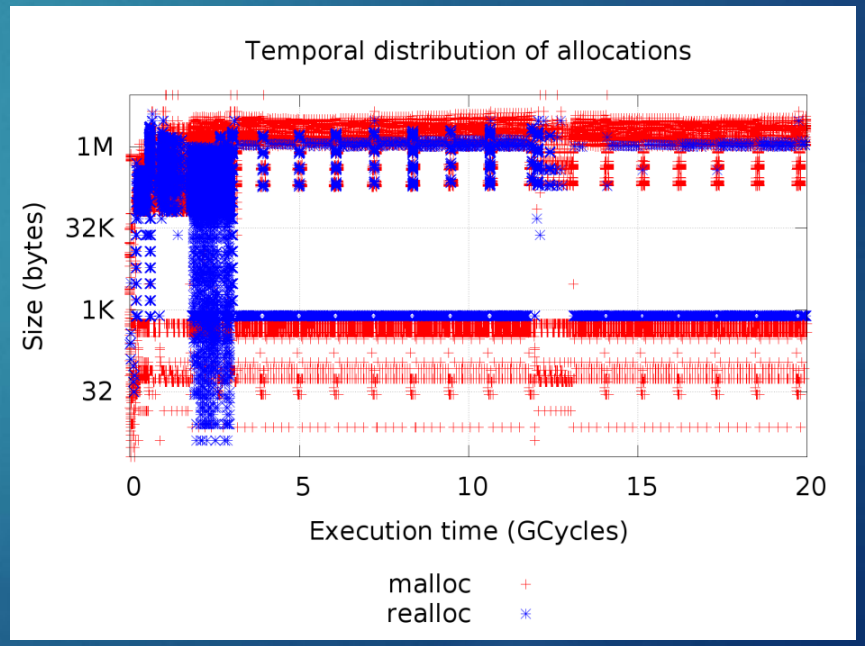


Allocator performance on HPC applications

July 2014

- Main interest : **malloc time cost**
- Test case : **Hera (CEA)**
  - **Adaptive Mesh Refinement** (AMR)
  - **Massive C++/MPI code** (~1 million lines).
- **Large number** of **memory allocations** (~75 millions / 5 minutes on 12 cores)
- **Large number** of **alloc/realloc** around **~20 MB**
- **Available allocators :**
  - **Doug Lea** / **PTMalloc** : libc Linux
  - **Jemalloc** : FreeBSD / Firefox / Facebook
  - **TCMalloc** : Google
  - *Hoard*

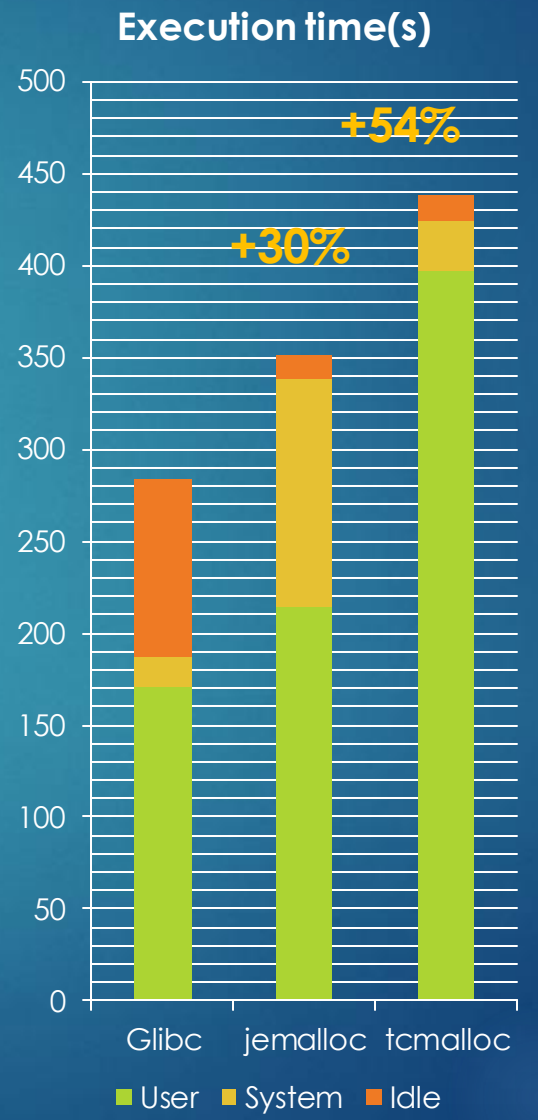


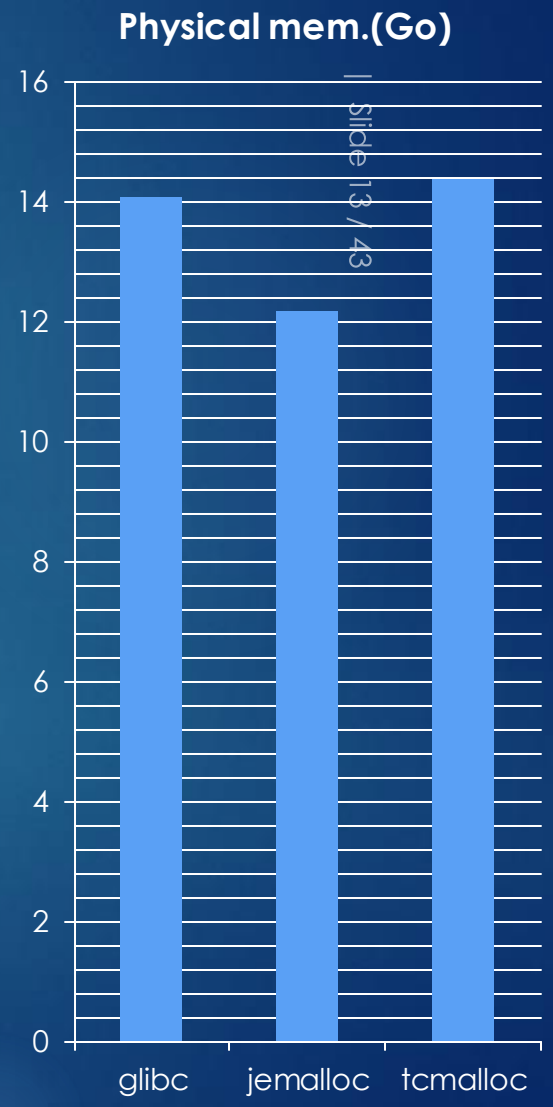


Hera preliminary results

12 cores

128 cores





How to measure malloc time

- Measurement method :

```
T0 = clock_start();
ptr = malloc(SIZE);
T1 = clock_end();
```

- Ok for **small blocks**, but not for **large** one :

```
T0 = clock_start();
ptr = malloc(SIZE);
for ( i = 0 ; i < SIZE ; i += PAGE_SIZE)
          ptr[i] = 0;
T1 = clock_end();
```

- **Lazy page allocation**.
- **Page faults** on first access.

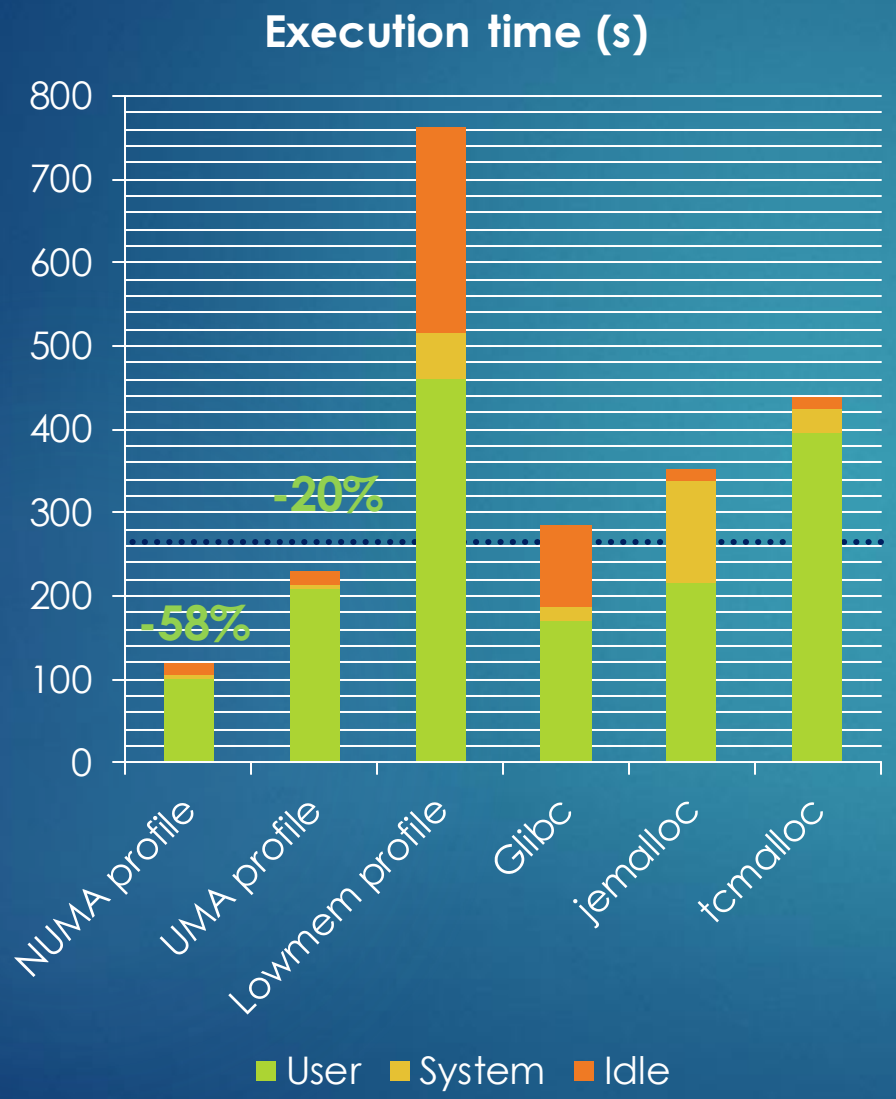
For 4GB	Malloc	First access
Time (M cycles)	0,008	1 217

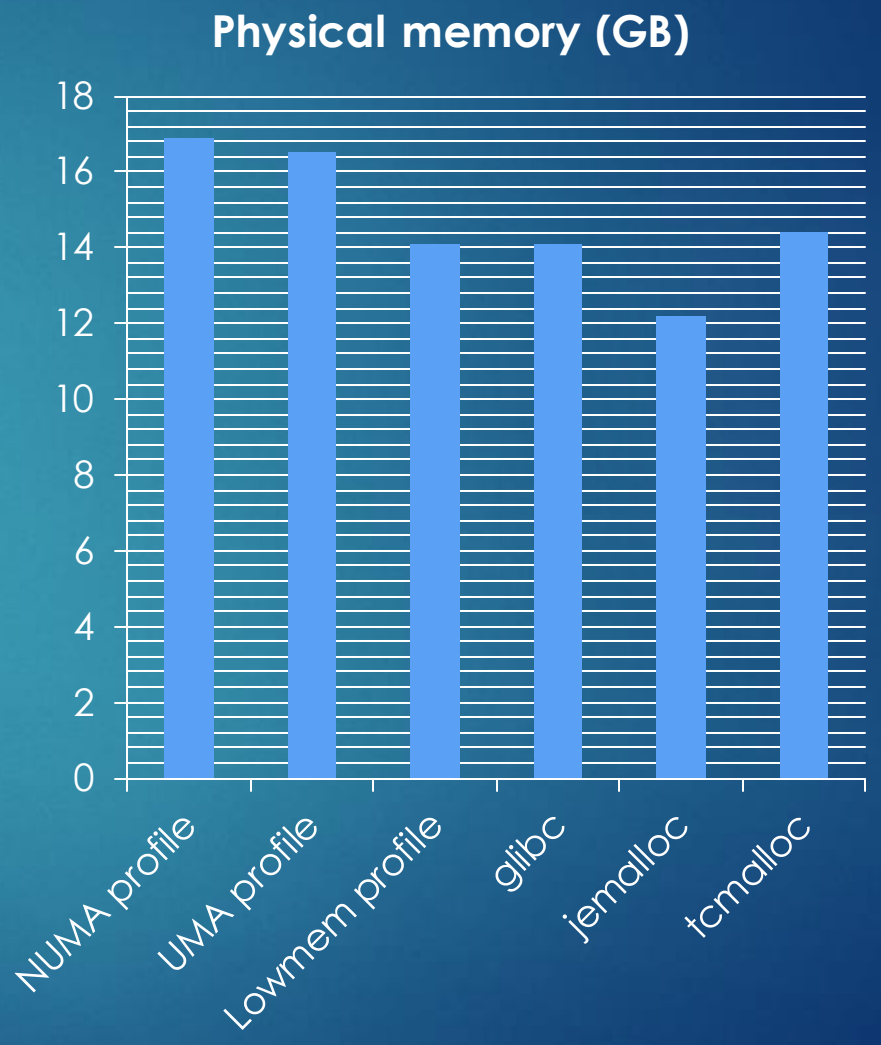
Large allocations

- Small allocation **well handled** by most allocators, **best is jemalloc / tcmalloc**.
- Cost for **large allocation : page faults.**
- **Commonly neglected**, literature mainly discuss small allocations
- Direct call to **mmap/munmap**
- **HPC applications** (expected to) use **large arrays**

Hera on Nehalem-EP (128 : 4*4*8 cores)

July 2014





Sébastien Valat
15/02/2018

Exascale computing research

UNIVERSITÉ DE VERSAILLES ST-QUENTIN-EN-YVELINES

cea

intel

GENCI

MALT

A MALLOC TRACKER

Questions

- We have **good profiling tool** for **timings** (eg. Valgrind or vtune)
- But for what **memory profiling** ?
- Memory can be an issue :
  - **Availability** of the resource
  - **Performance**
- Three main questions :
  - How to reduce **memory footprint** ?
  - How to improve overhead of **memory management** ?
  - How to improve **memory usage** ?

Some issue examples

- We want to point :
  - **Where** memory is allocated.
  - **Properties** of allocated chunks.
  - **Bad** allocation **patterns** for performance.

```
__thread Int gblVar[SIZE];
int * func(int size)
{
        child_func_with_allocs();
        void * ptr = new char[size];
        double * ret = new double[size*size*size];
        for (auto it : iter_items)
        {
                double * buffer = new double[size];
                //short and quick do stuff
                delete [] buffer;
        }
        return ret;
}
```

Global variables and TLS

Indirect allocations

Leak

Might lead to swap for large size

Auto missing reference

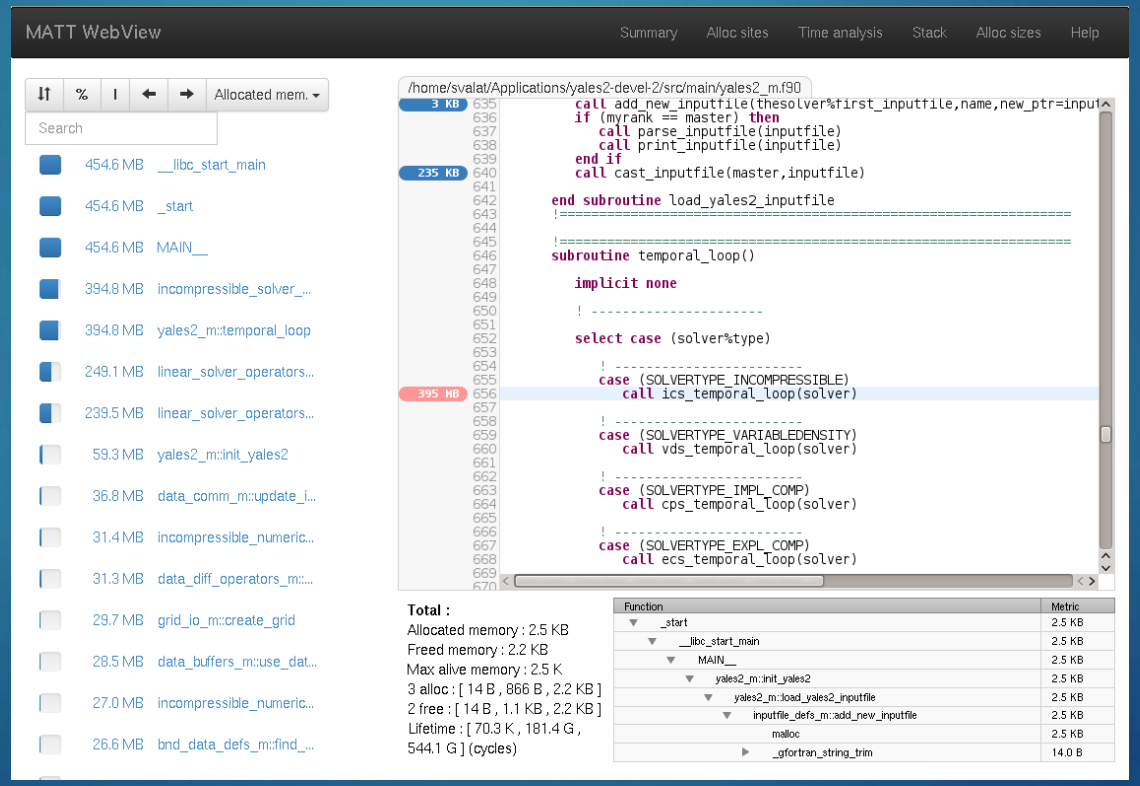
Short life allocations

What I want to provide

- Same **approach** than **valgrind/kcachgind**
- **Mapped** allocations on **sources lines** and **call stacks**
  - **Using profile** approach, **not snapshots**
- For **memory resource usage** :
  - Memory **leaks**
  - **Memory on peak**
- For **performance** :
  - Allocation **count** and **cumulated size**
  - Allocation **sizes** (min/average/max)
  - Chunk **lifetime** (min/average/max)

Our GUI : based on web tech.

- Web technology (NodeJS, D3JS, Jquery, AngularS)
- Easier for remote usage



Global summary

- Provide a small summary
- Provide some warnings

Show all details | Show help

Physical memory peak	66.7 MB
Virtual memory peak	158.1 MB
Requested memory peak	6.1 MB
Cumulated memory allocations	11.5 MB
Allocation count	172.2 K
Recycling ratio	1.9
Leaked memory	743.7 KB
Largest stack	0 B
Global variables	**10.0 MB** ⚠
TLS variables	48 B
Global variable count	**421.0 K** ⚠
Peak allocation rate	37.8 MB/s

Global summary : top 5 functions

- Summarize **top functions** for some metrics
- Points to check
- Examples on YALES2

Alloc count

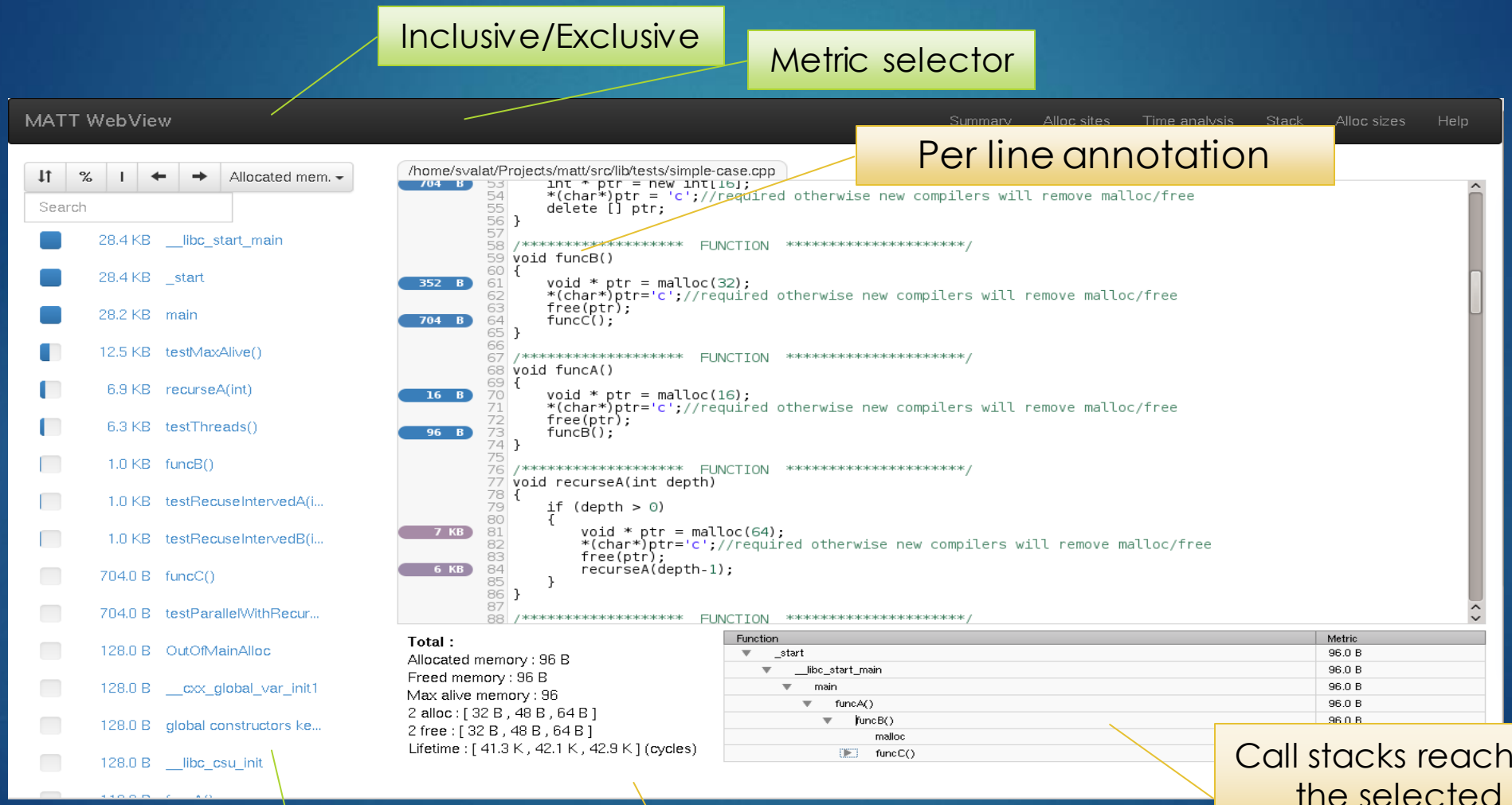
Ratio	Allocs	Function
	911.9 K	data_comm_m::copy_int_comm_to_data
	896.4 K	data_comm_m::copy_data_to_int_comm
	853.2 K	data_comm_m::update_int_comm
	484.9 K	sponge_layer_m::calc_sponge_layer_mask
	296.0 K	incompressible_numerics_m::ics_diffuse_velocity_rk_4th

Allocated memory

Ratio	Allocs	Function
	202.4 MB	linear_solver_operators_m::solve_linear_system_deflated_pcg
	26.6 MB	bnd_data_defs_m::find_bnd_data
	21.8 MB	linear_solver_operators_m::solve_el_grp_pcg
	19.0 MB	data_comm_m::copy_int_comm_to_data
	18.1 MB	data_comm_m::update_int_comm

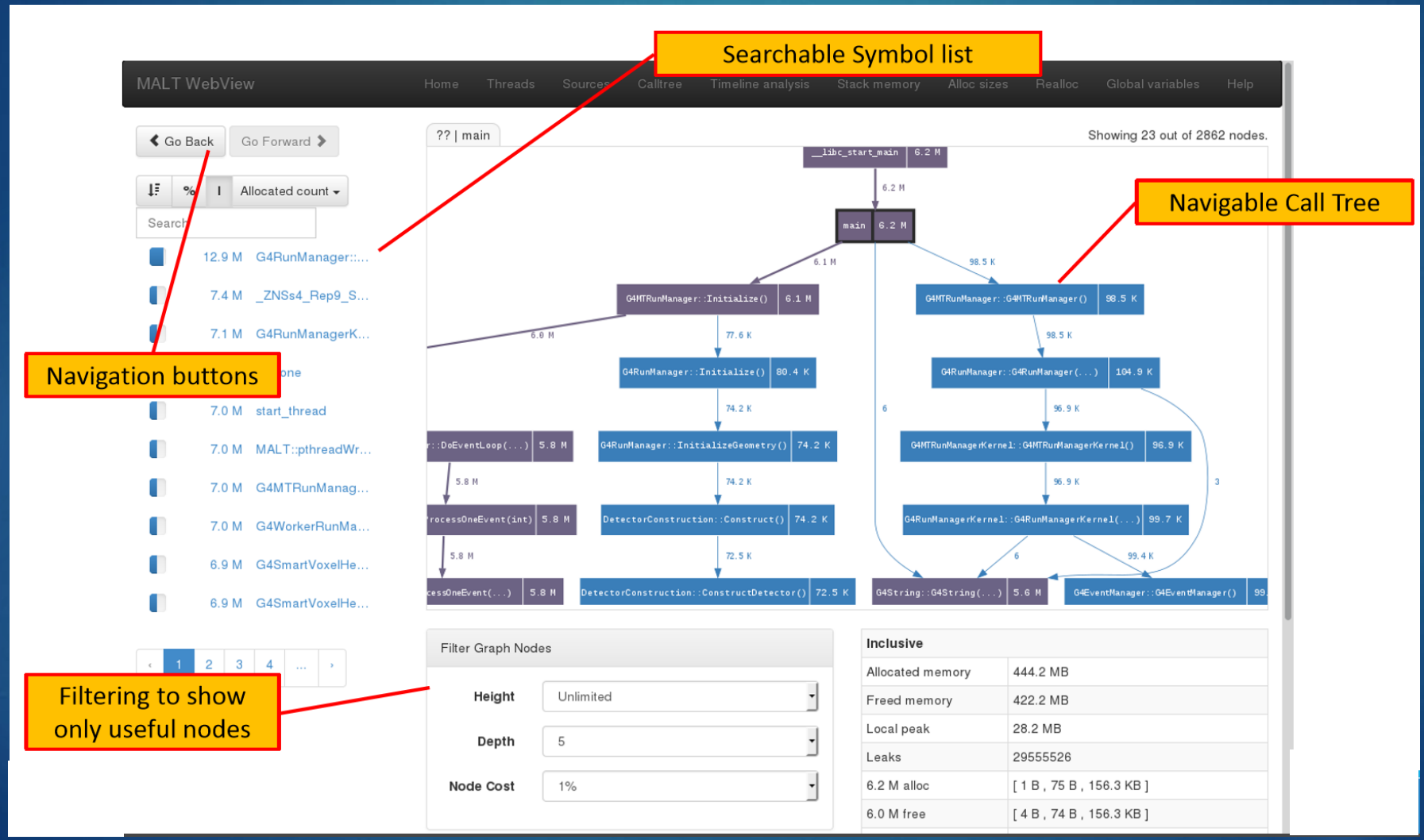
Peak memory

Source annotations

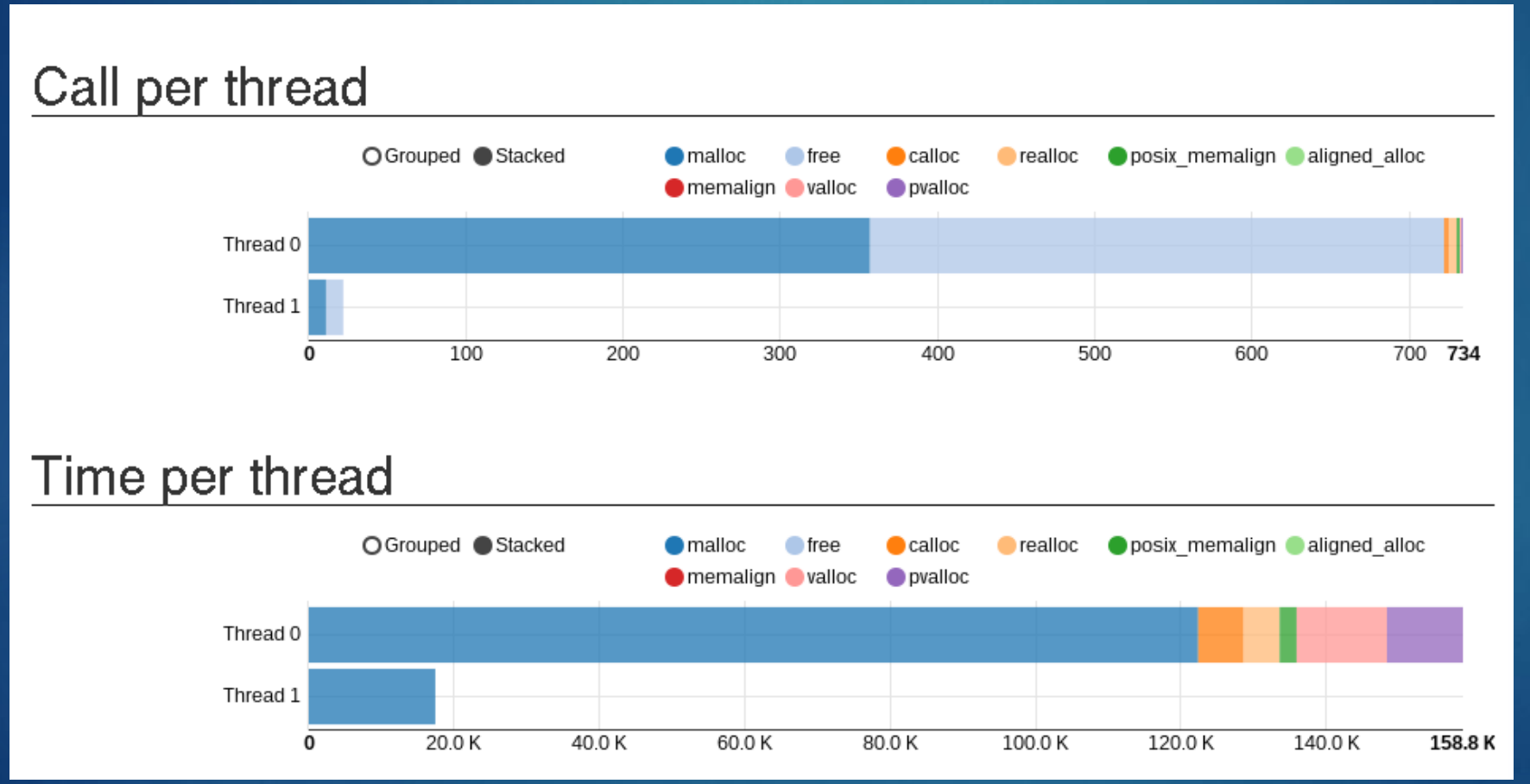


Call tree view

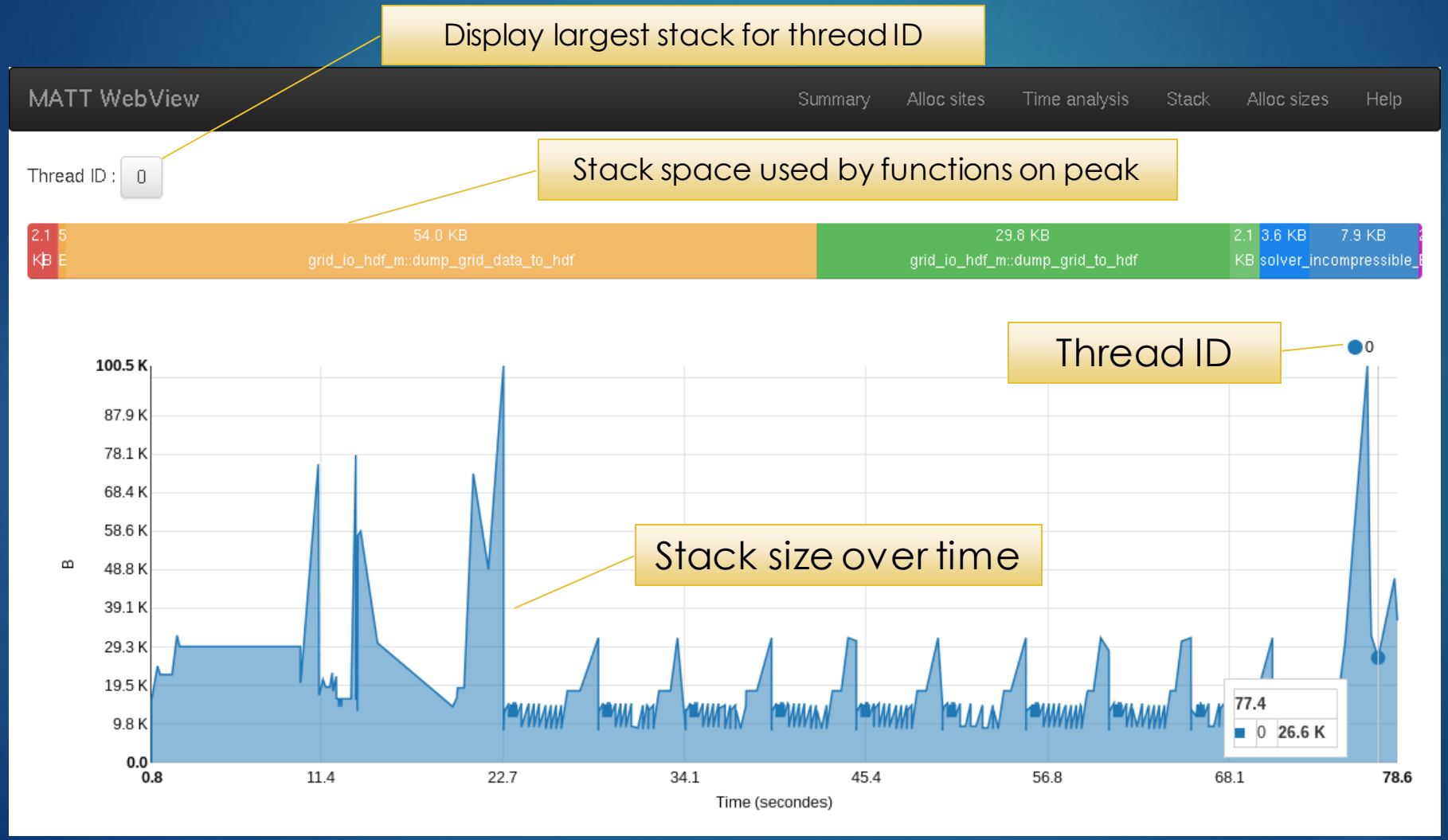
Work by CERN/Openlab summer student



Per thread statistics

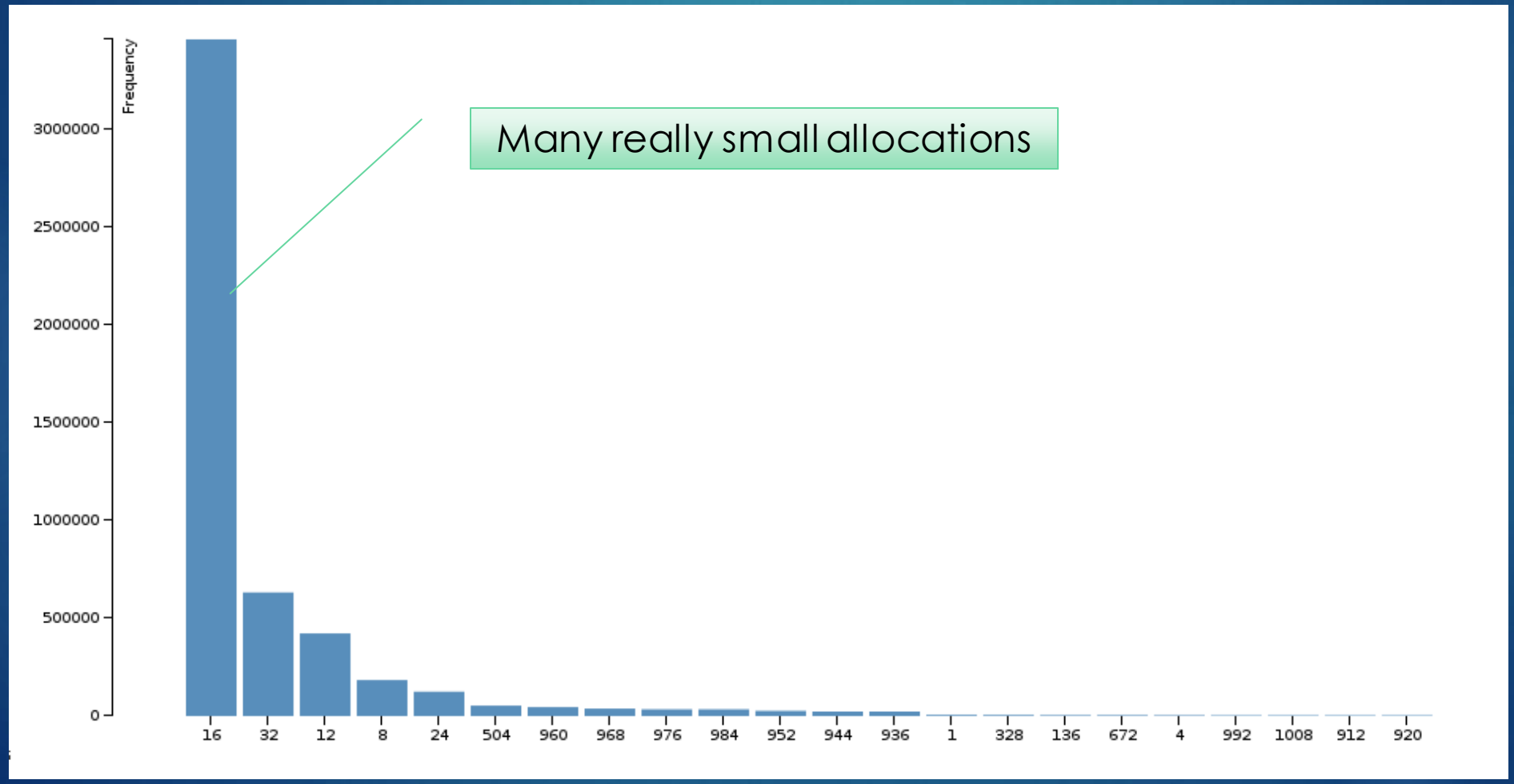


Tracking stack memory

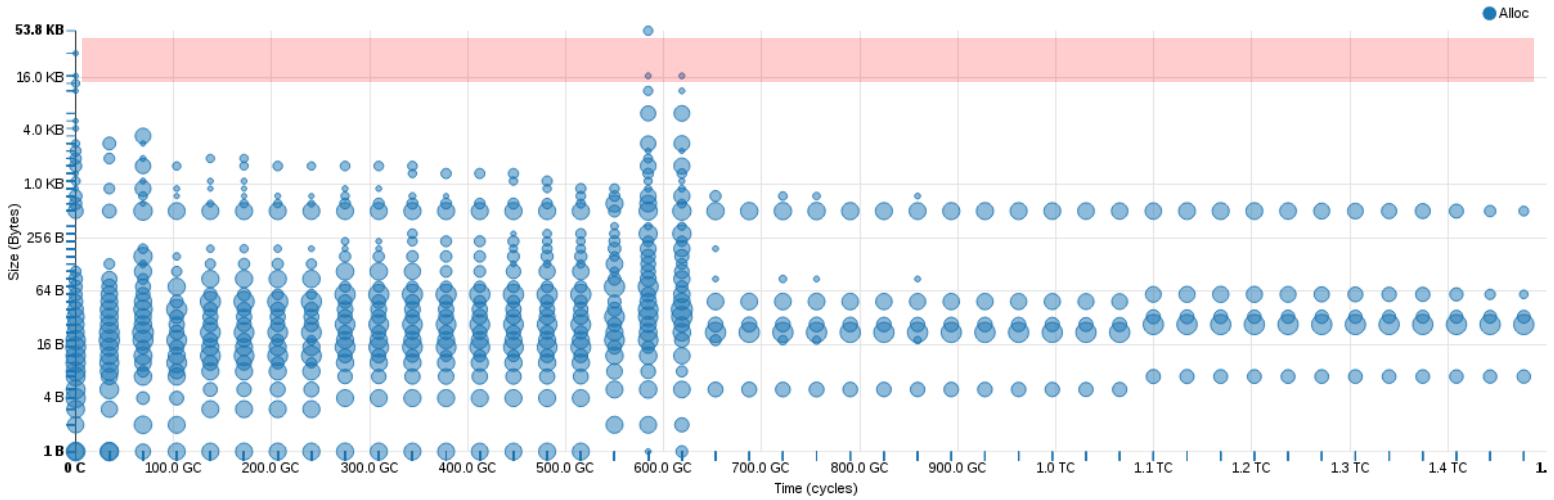


Chunk size distribution

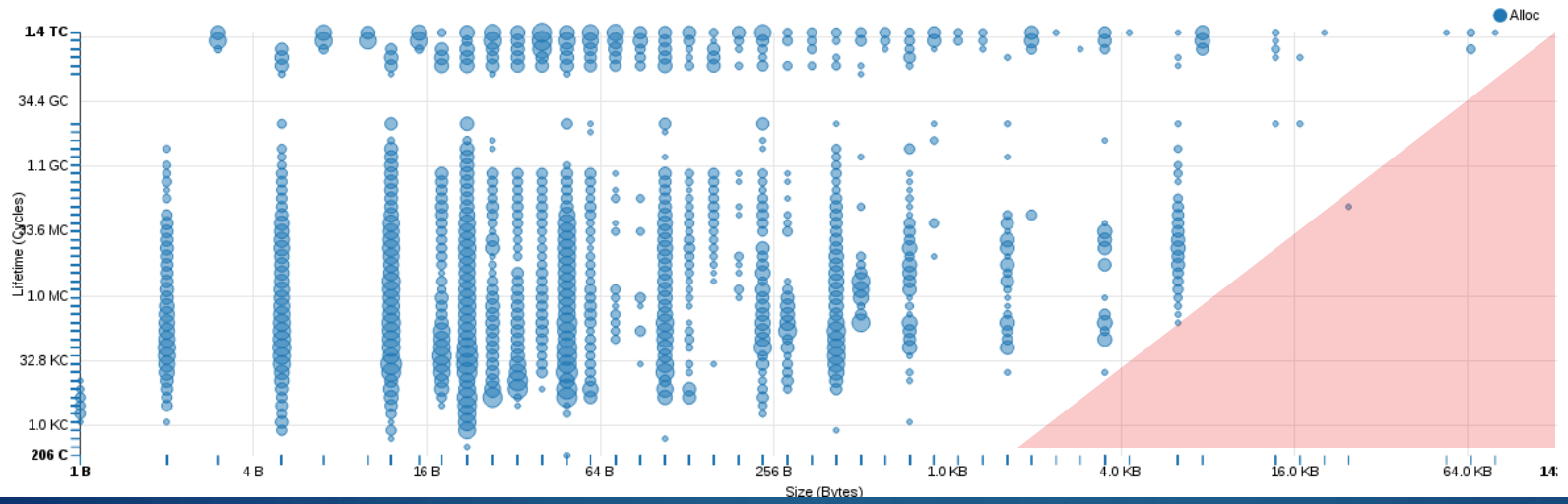
Example from YALES2



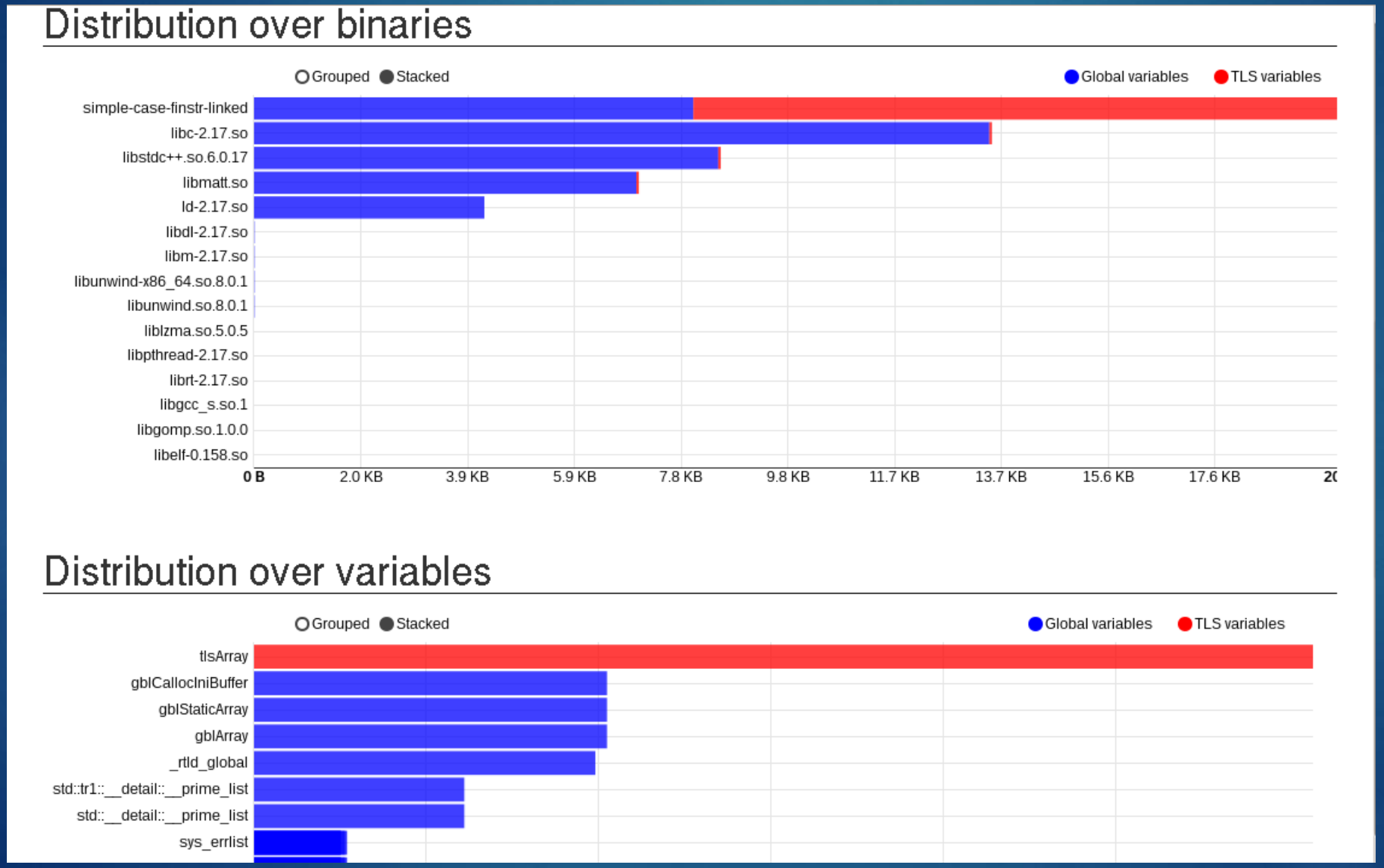
Size over time



Lifetime over size

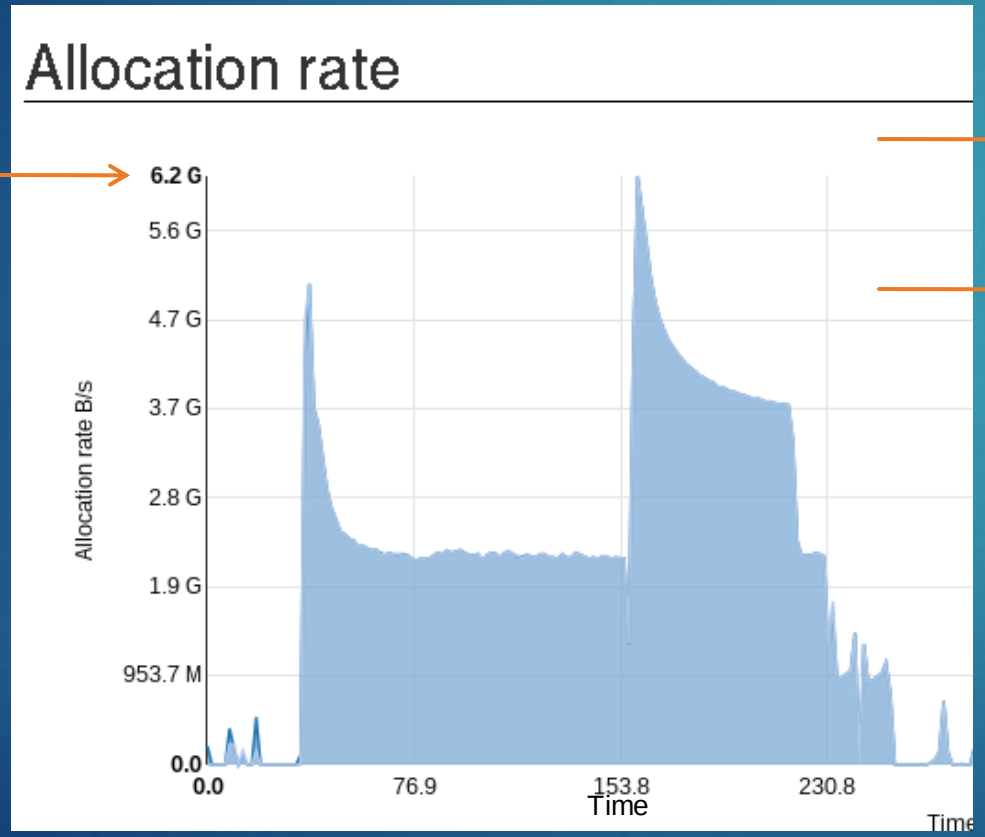


Global variables



Example on AVBP init phase

- Issue with **reallocation** on init
- Detected with **allocation rate** & **cumulated allocatated mem.**




```
99          CALL assert(capacity==size(array),&
100               'array and capacity variable are not
101
102         IF (needed_size>capacity) THEN
103           IF (ALLOCATED ( temp) ) DEALLOCATE(temp)
104           ALLOCATE ( temp(capacity))
105
106           DO i=1,capacity
107             temp(i)=array(i)
108           END DO
109
110           DEALLOCATE ( array)
111           ALLOCATE ( array(new_cap))
112
113           DO i=1,capacity
114             array(i)=temp(i)
115           END DO
116
117           capacity=new_cap
118         END IF
119
```

57 GB

57 GB

Function

_start

Total :

Allocated memory : 56.8 GB

Max alive memory : 135.7 M

3.5 K alloc : [16.0 KB , 16.3 MB , 33.7 MB]

Lifetime : [107.8 K , 26.7 M , 476.7 M] (cycles)

Own :

Allocated memory : 56.8 GB

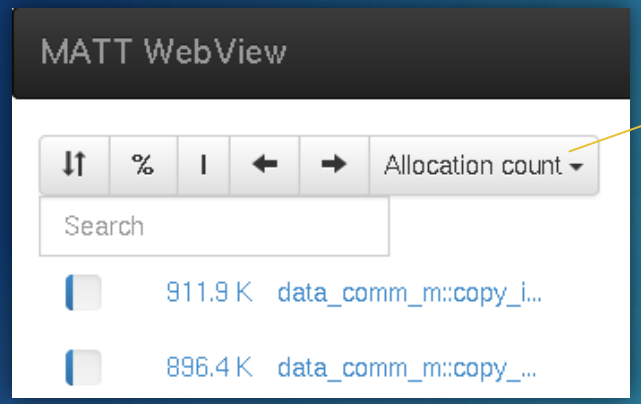
Max alive memory : 135.7 M

3.5 K alloc : [16.0 KB , 16.3 MB , 33.7 MB]

Lifetime : [107.8 K , 26.7 M , 476.7 M] (cycles)

Allocatable arrays on YALES2

- Issue only occur with **gfortran**, ifort uses stack arrays.



Search intensive alloc functions

Huge number of allocation for a line programmer think it doesn't do any !

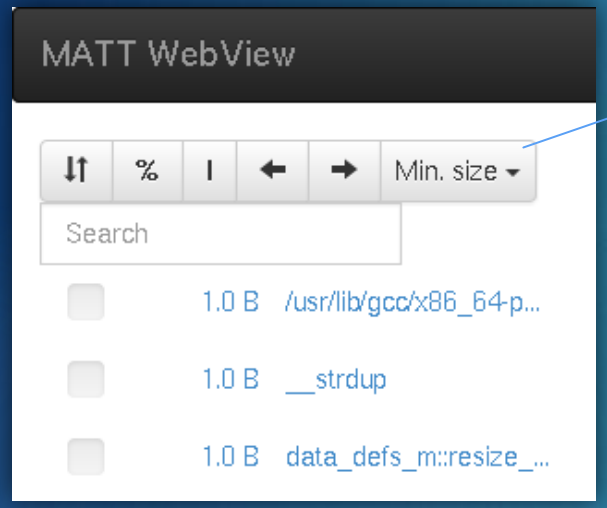
```
892                 do i=1,nitem_el_grp
893                    el_grp_ind = el_grp_index2int_comm_index%val(1,i)
894                    int_comm_ind = el_grp_index2int_comm_index%val(2,i)
608 K 895              el_grp_r2%val(1:dim1,el_grp_ind) = int_comm_r2%val(1:dim1,int_comm_ind)
896                 end do
```

Total :
Allocated memory : 9.5 MB
Freed memory : 9.5 MB
Max alive memory : 432
608.0 K alloc : [16 B , 16 B , 16 B]
608.0 K free : [16 B , 16 B , 16 B]
Lifetime : [24.5 K , 39.9 K , 37.8 M] (cycles)
Own :
Allocated memory : 9.5 MB

And mostly really small allocations !

We can found allocs of 1B !

- Examples on YALES 2, small allocations :



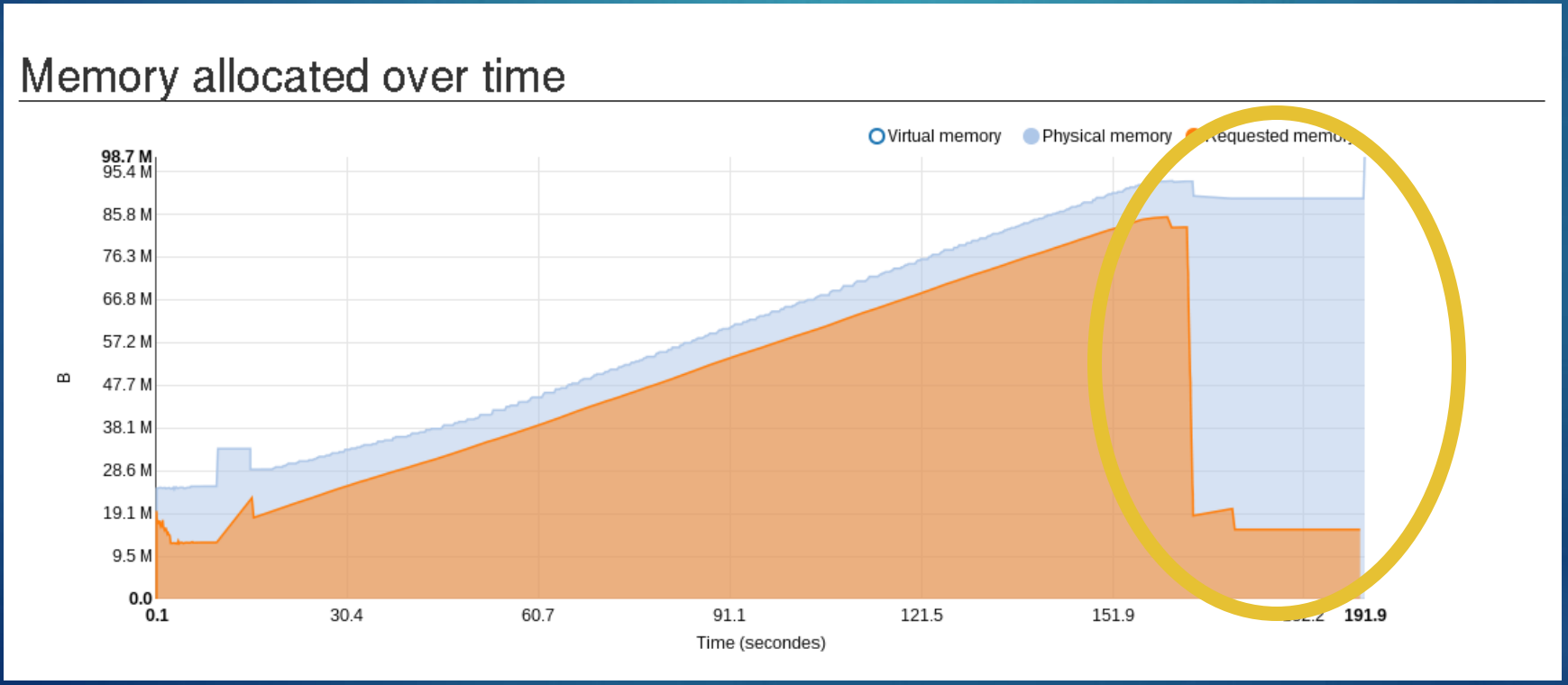
Search for the minimal chunk size.

Many codes produce allocations of 1B.
OK with moderation.

```
530              case (DATATYPE_REAL_NODE_VECTOR,DATATYPE_REAL_ELEM_VECTOR, &
531                    DATATYPE_REAL_FACE_VECTOR,DATATYPE_REAL_PAIR_VECTOR)
532                 if (associated(data_ptr%r2_ptrs)) then
533                    deallocate(data_ptr%r2_ptrs)
534                 end if
1 B 535             allocate(data_ptr%r2_ptrs(nel_grps))
536                 do n=1,nel_grps
537                    NULLIFY(data_ptr%r2_ptrs(n)%ptr)
538                 end do
```

Fragmentation issue

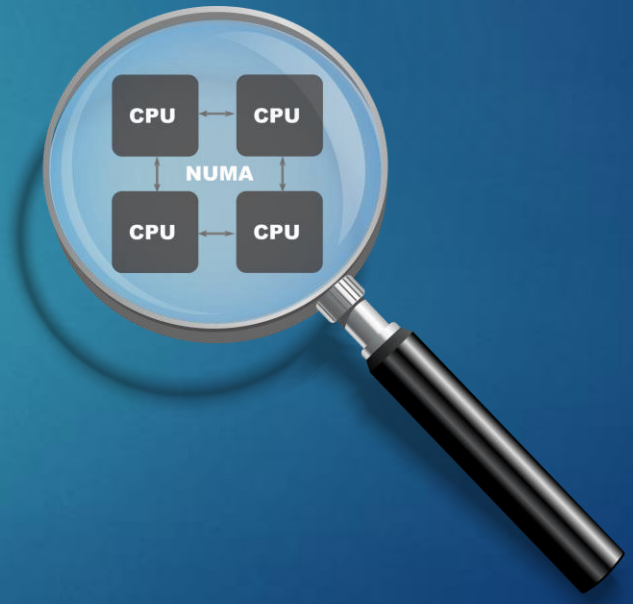
- Example of **fragmentation** detection
- Using the time chart with **physical**, **virtual** and **requested memory**
- **Solution** : **avoid interleaved** allocation of chunks with **different lifetime**.
- Looking on **source annotation** : most of them **can be avoided**.



Sébastien Valat
15/02/2018

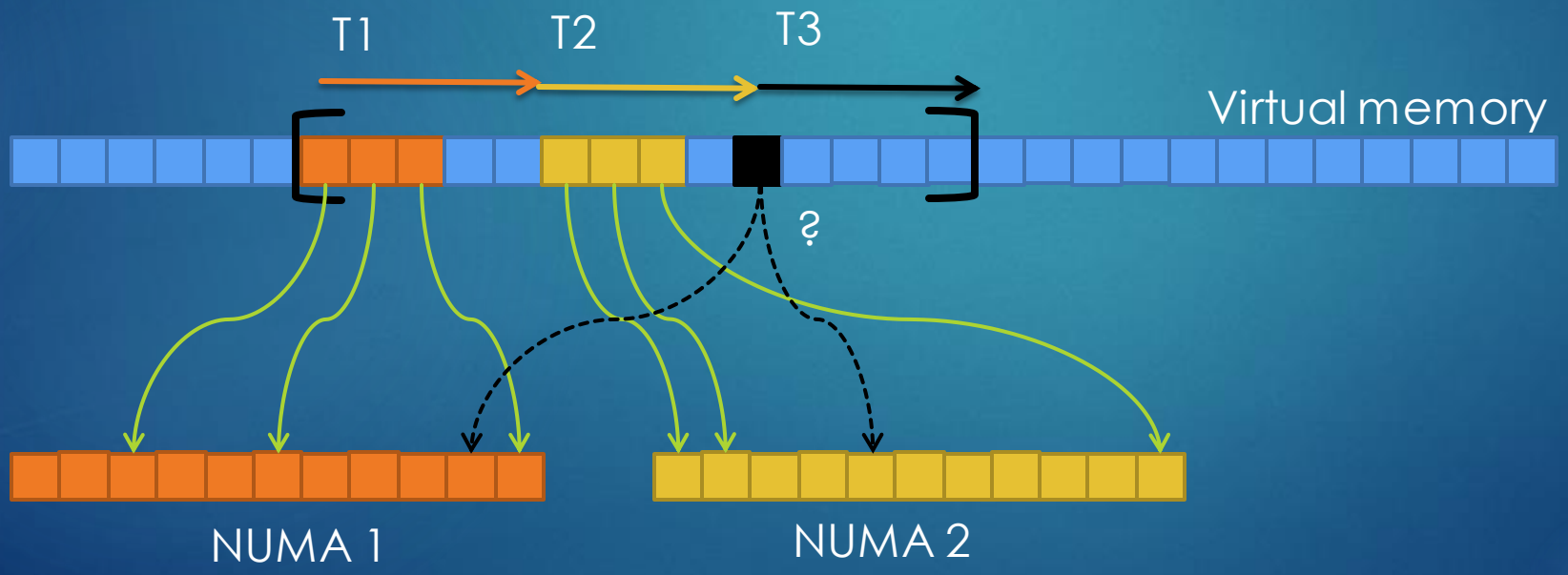
NUMAPROF

A NUMA MEMORY PROFILING TOOL



Implicit binding : first touch

- New allocated segments are **physically empty**
- They are filled on **first touch**
- Page selection **depend** of the **thread position**



Typical OpenMP mistake

- Make first **init outside of OpenMP** (in thread 1)
- So **each pages** will be first touched **on NUMA 1**

```
#pragma omp parallel for
for (int i = 0 ; i < SIZE ; i++)
        array[i] = 0;
```

- Then access

```
#pragma omp parallel for
for (int i = 0 ; i < SIZE ; i++)
        array[i]++;
```

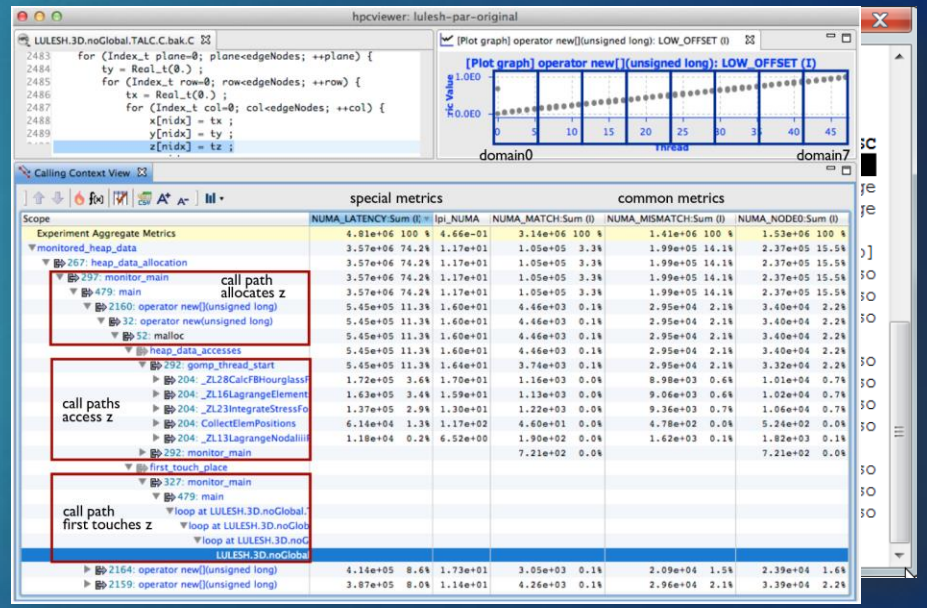
- **Bad performance** due to remote accesses !

Wish list for a profiling tool…

- We want to know if we make **remote accesses**
- Ideally we need to know **where**…
- We can dream, we want to know **which allocation contain issues**
- We want to know **where** the **first touch** has been done
- On KNL we want to check **MCRAM accesses**

Existing tools

- **MemProf** [1], a research paper from Grenoble
  - Hardware feature provided by AMD & kernel module, No GUI
- **SNPERF** [2], again a research project
  - Link utilization on time chart
- **NUMAgrind** [3]
  - Looks nice but not available
- **Numatop**
  - Similar to top, global profiling
- **HPCToolkit** [4]
  - Also have a nice interface, but hw counters & sampling



[1] Renaud Lachaize and Al. MemProf: A Memory Profiler for NUMA Multicore Systems.
[2] U. Prestor and Al., "An application-centric ccNUMA memory profiler,"
[3] Profiling Directed NUMA Optimization on Linux Systems: A Case Study of the Gaussian Computational Chemistry Code
[4] A Tool to Analyze the Performance of Multithreaded Programs on NUMA architectures

NUMAPROF

- Take back the idea from **MALT**
  - **Web interface**
  - **Source annotation**
  - **Global metrics**

- Use intel **Pin**
  - Permit to **instrument** all **memory accesses**
  - **Parallel** opposite to valgrind
  - **Difficulty**: we cannot easily use libs inside the tool
  - I would have used hwloc and libnuma.....

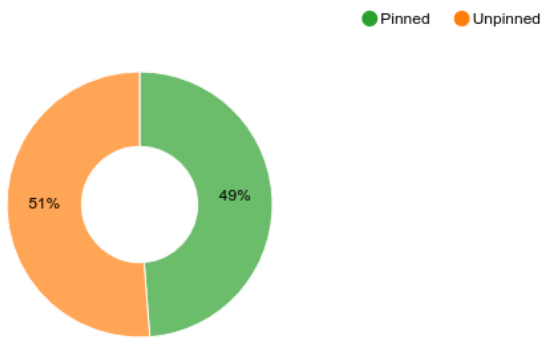
On access we need…

- On each access we want to know if it is
  - **Remote** access
  - **Local** access
  - **MCDRAM** access
  - **Page is pinned**
  - **Thread is pinned**
- So, we need to know
  - Where is **the page**
  - Where is **the current thread**
- We can **skip** accesses to **local stack** (overhead 80x -> 40x)

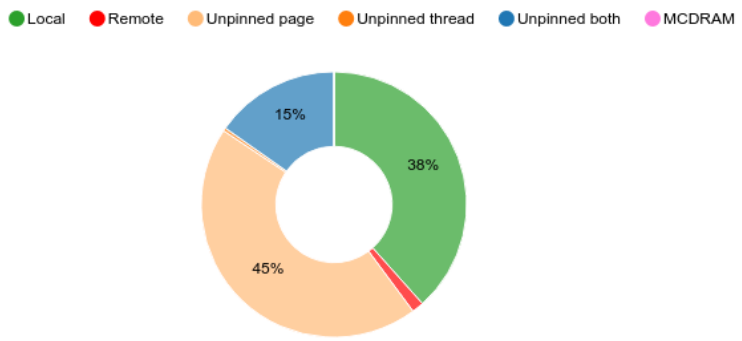
Global summary

Sébastien Valat
15/02/2018

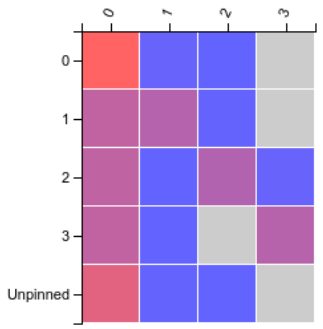
First touch



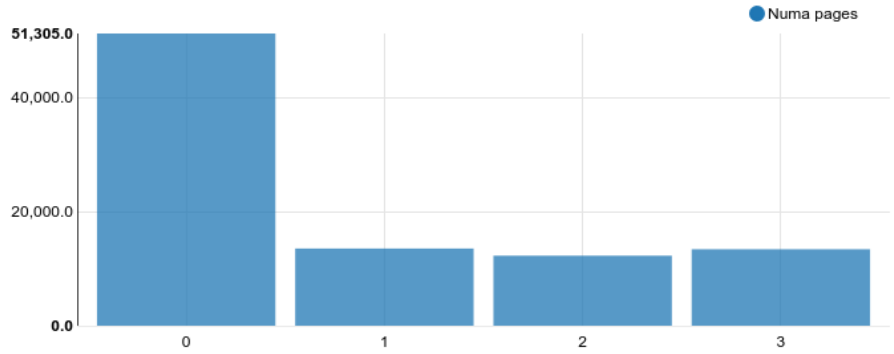
Memory access



Access matrix

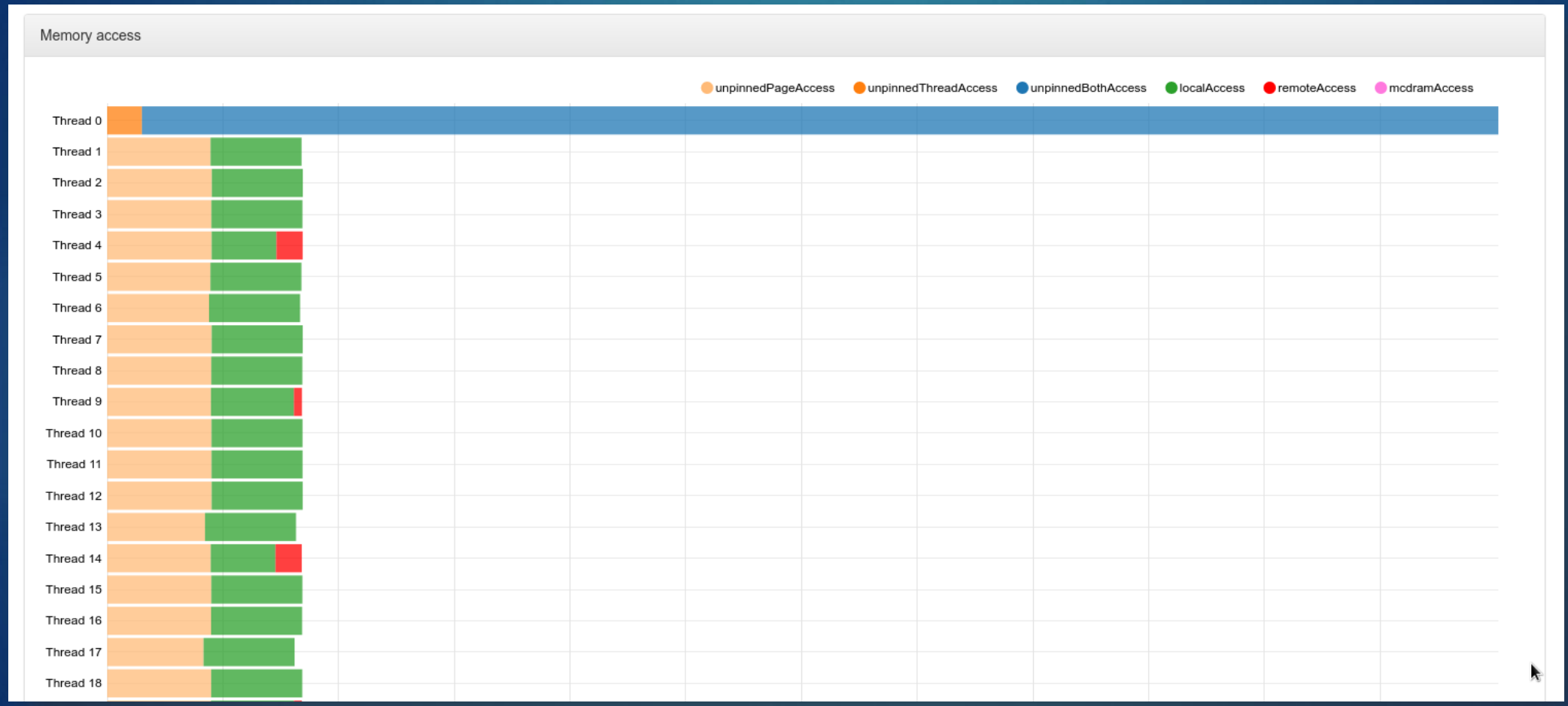


Peak allocated numa pages

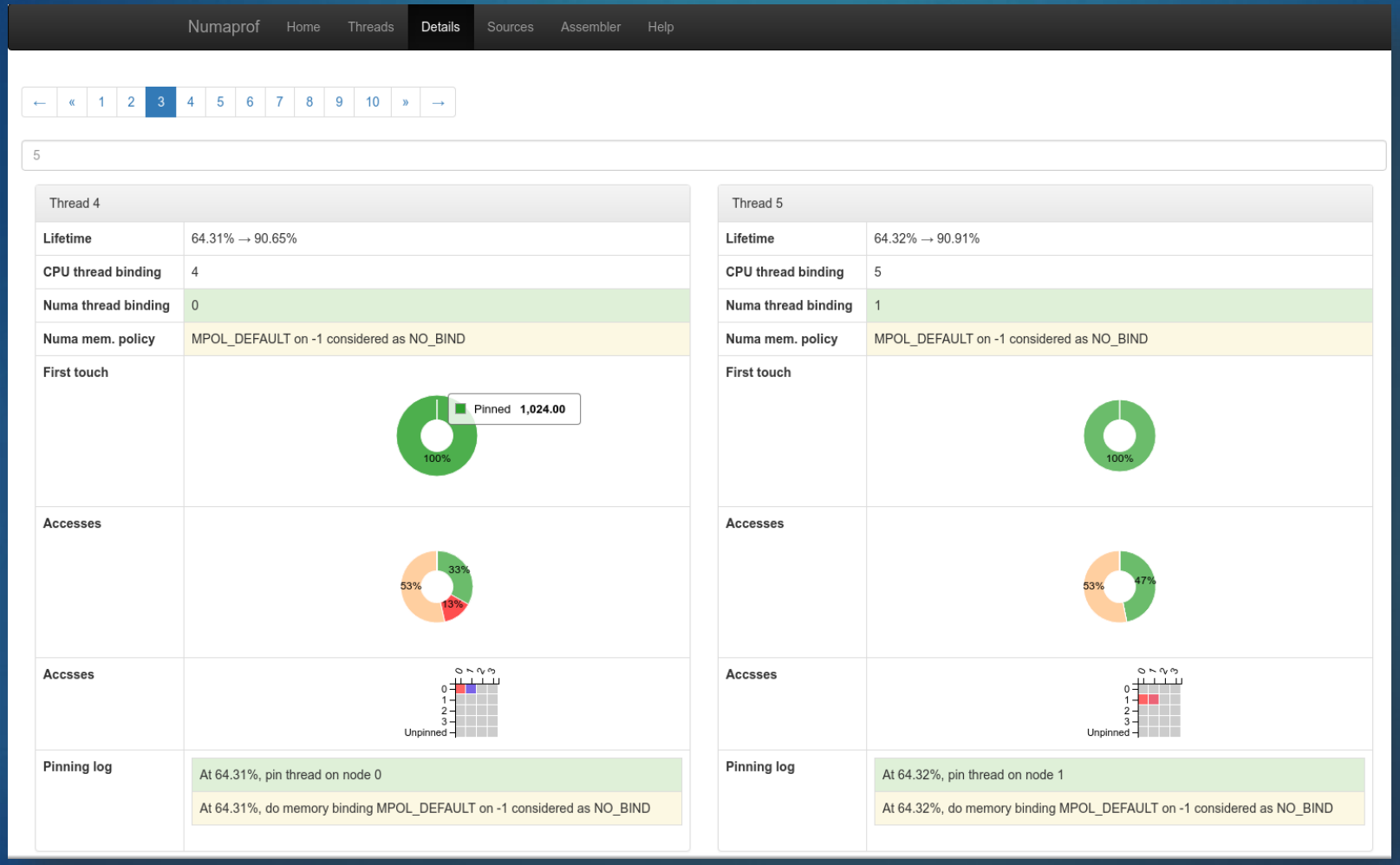


Statistics per thread

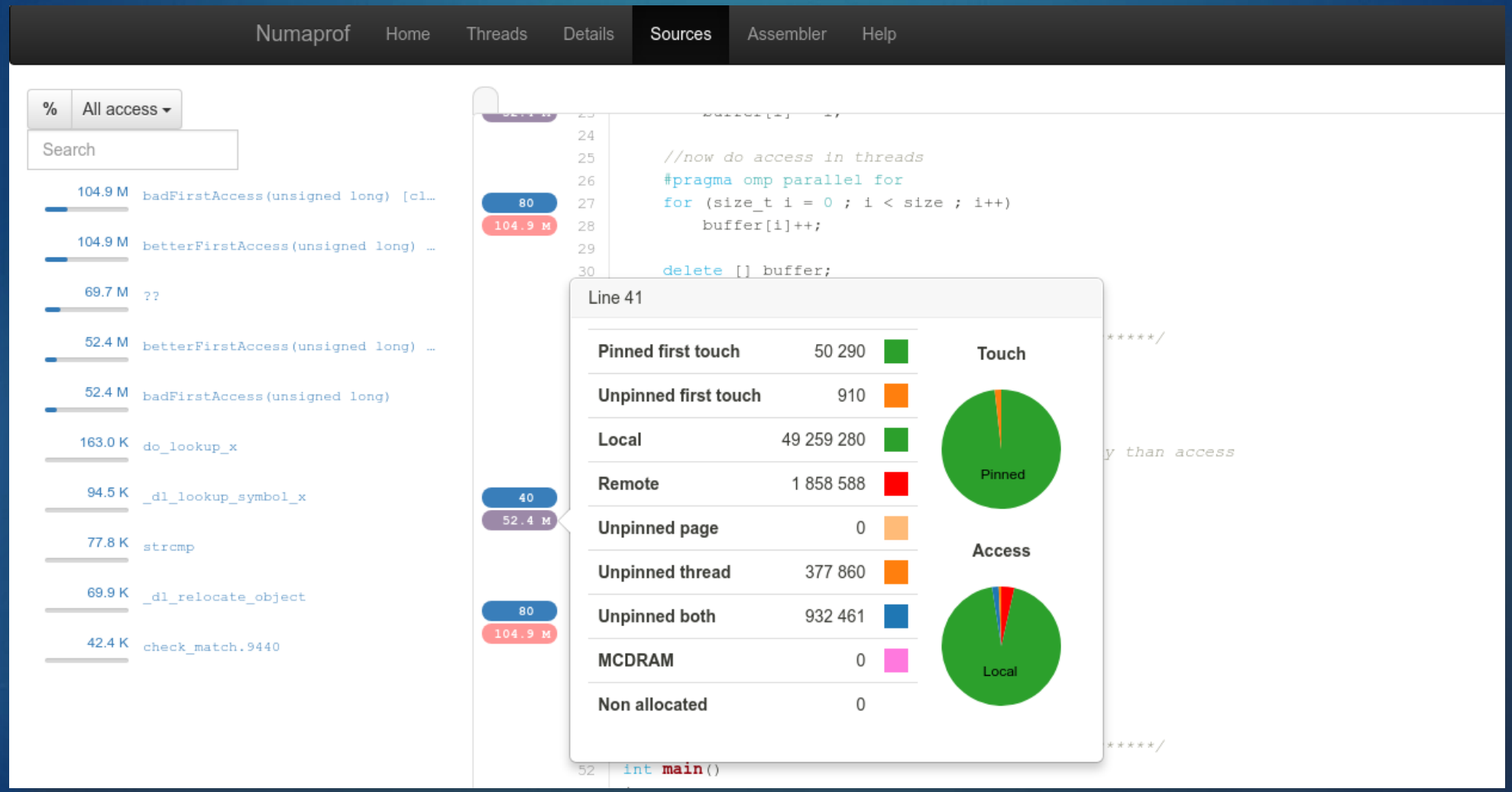
Sébastien Valat
15/02/2018



Details per thread

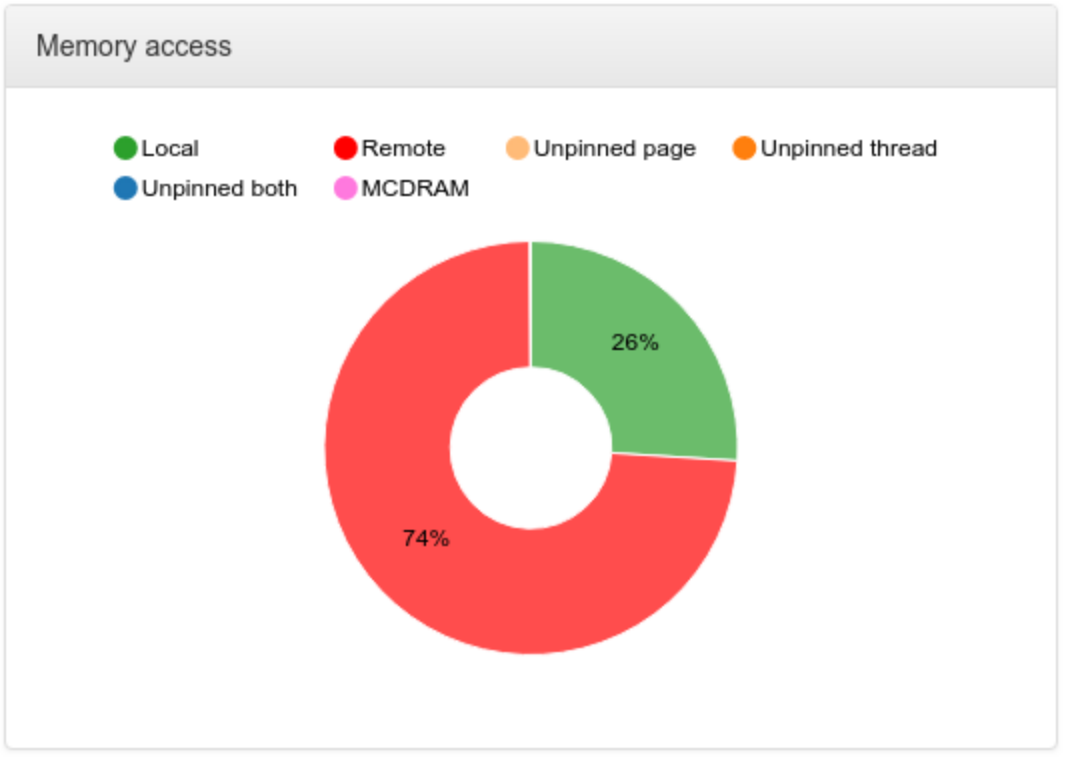


Source & asm annotations

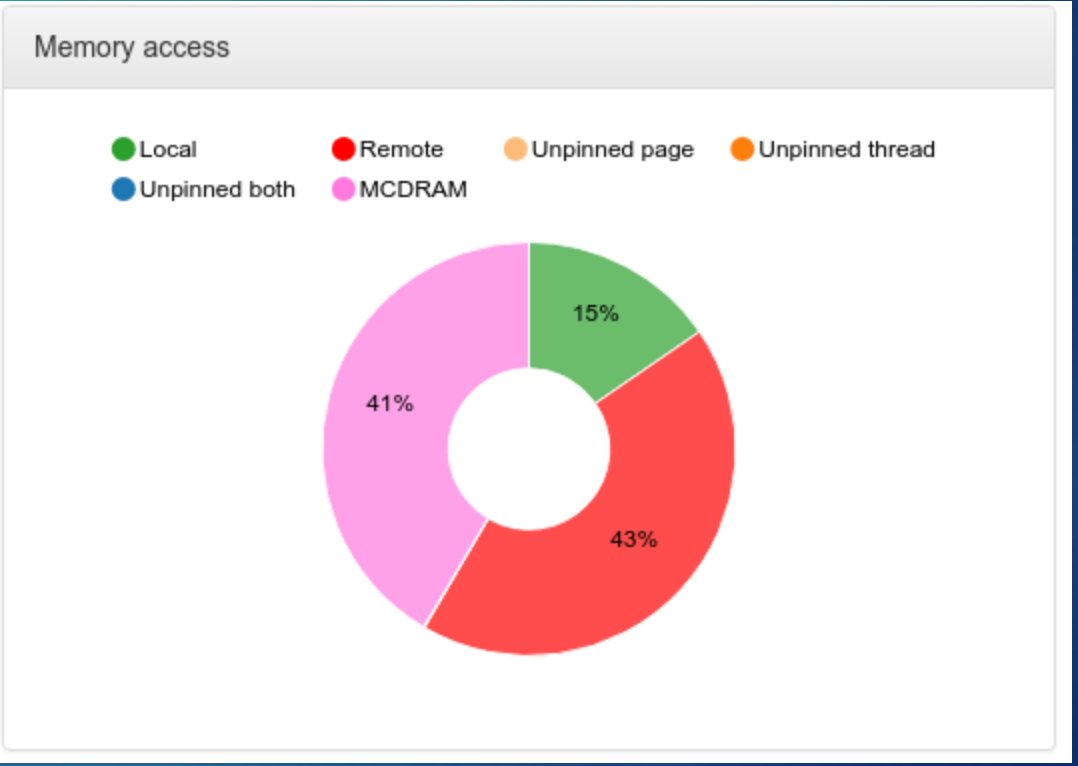


Code Hydro

- KNL Without HBM

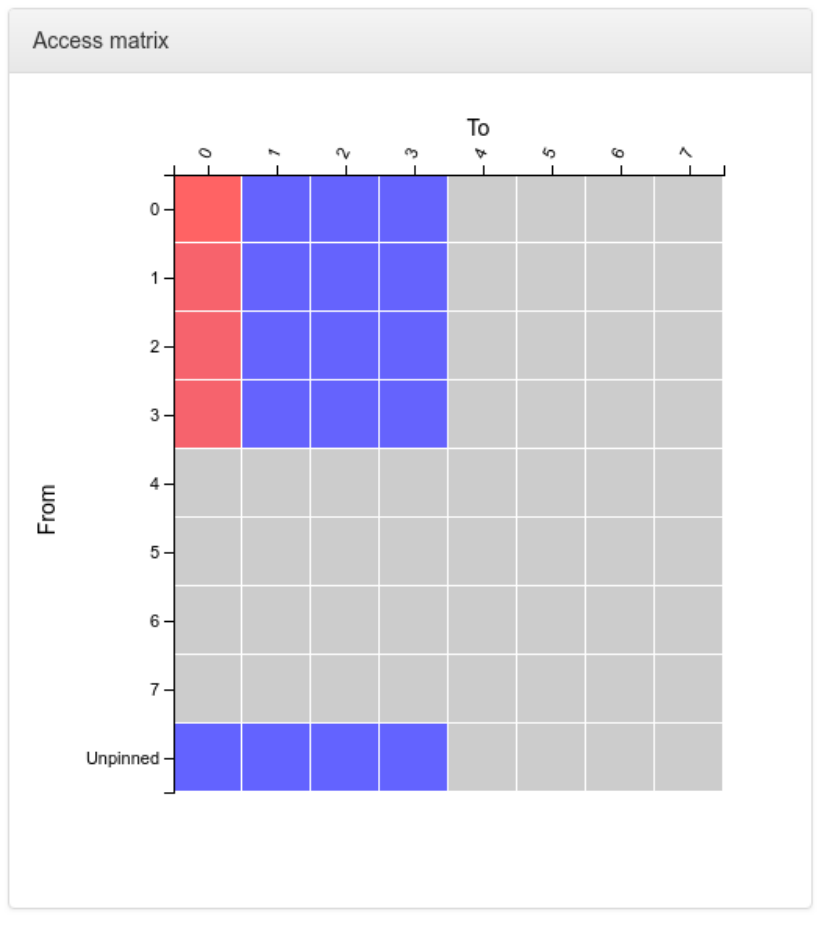


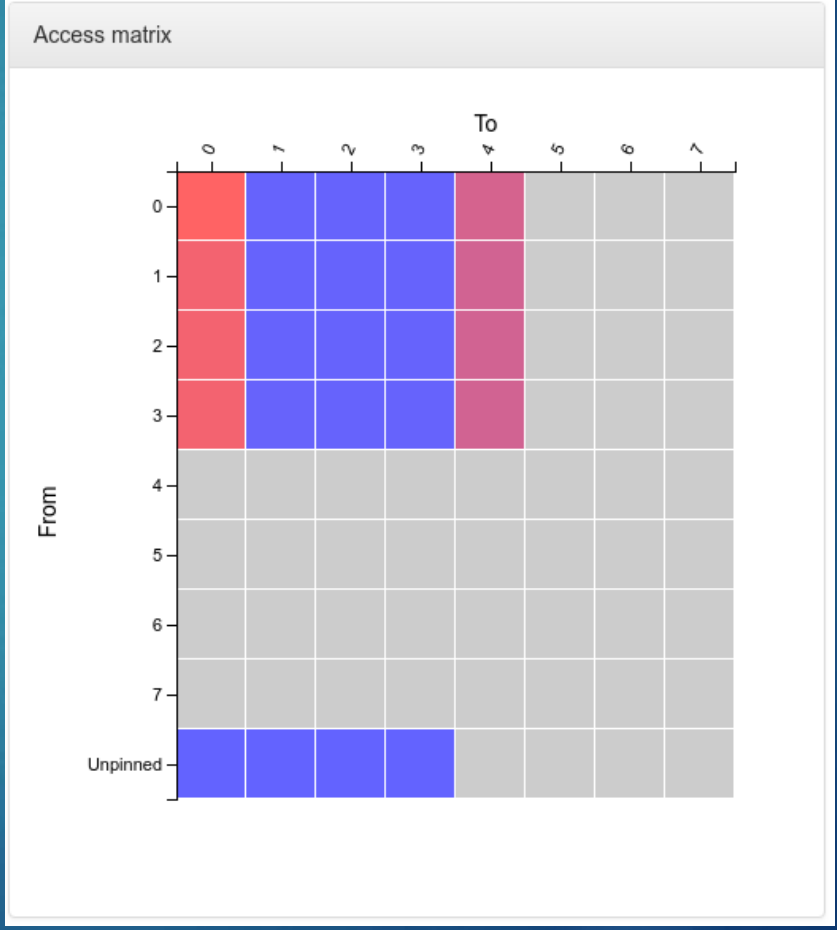
WITH HBM



Original Hydro access matrix

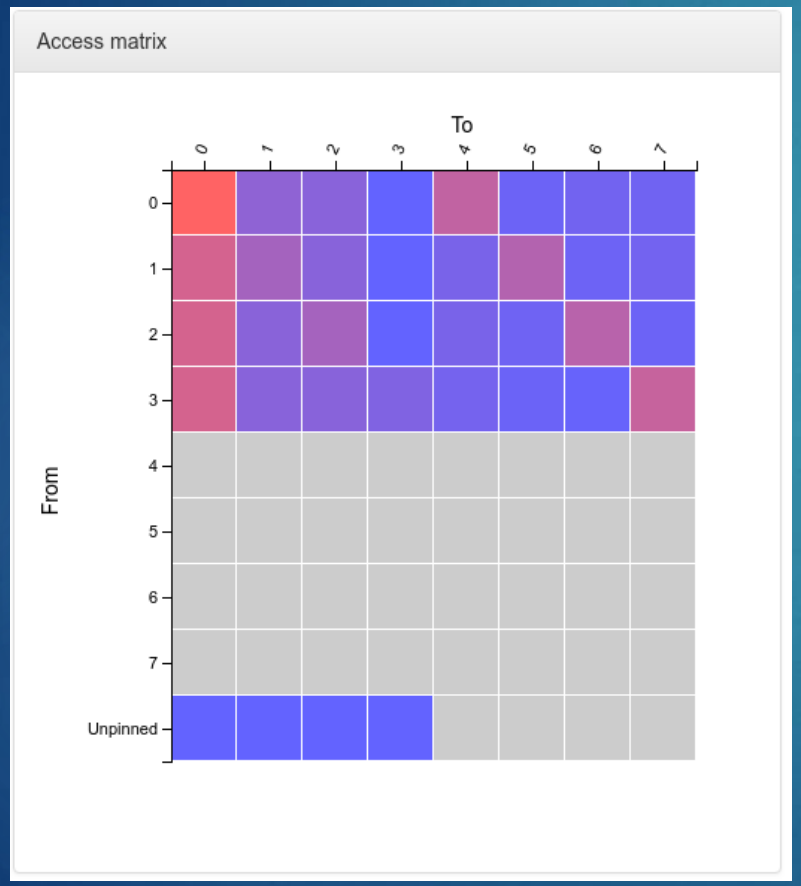
Sébastien Valat
15/02/2018

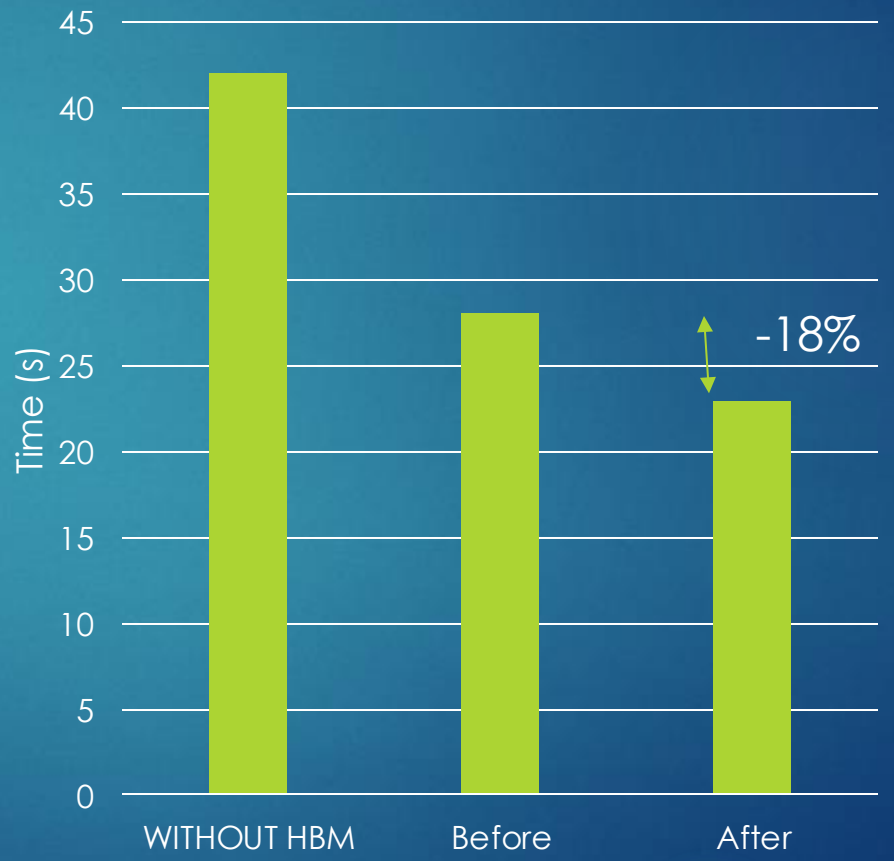




Speed up obtained on Hydro

Sébastien Valat
15/02/2018





Conclusion on memory

- Need to undestand mempory behavior of apps
- Tools are need
- I provide two on http://memtt.github.io/ un CeCILL-C (LPLC-Like)
  - MALT
  - NUMAPROF
- Still work needed in OS layers to progress

Questions ?

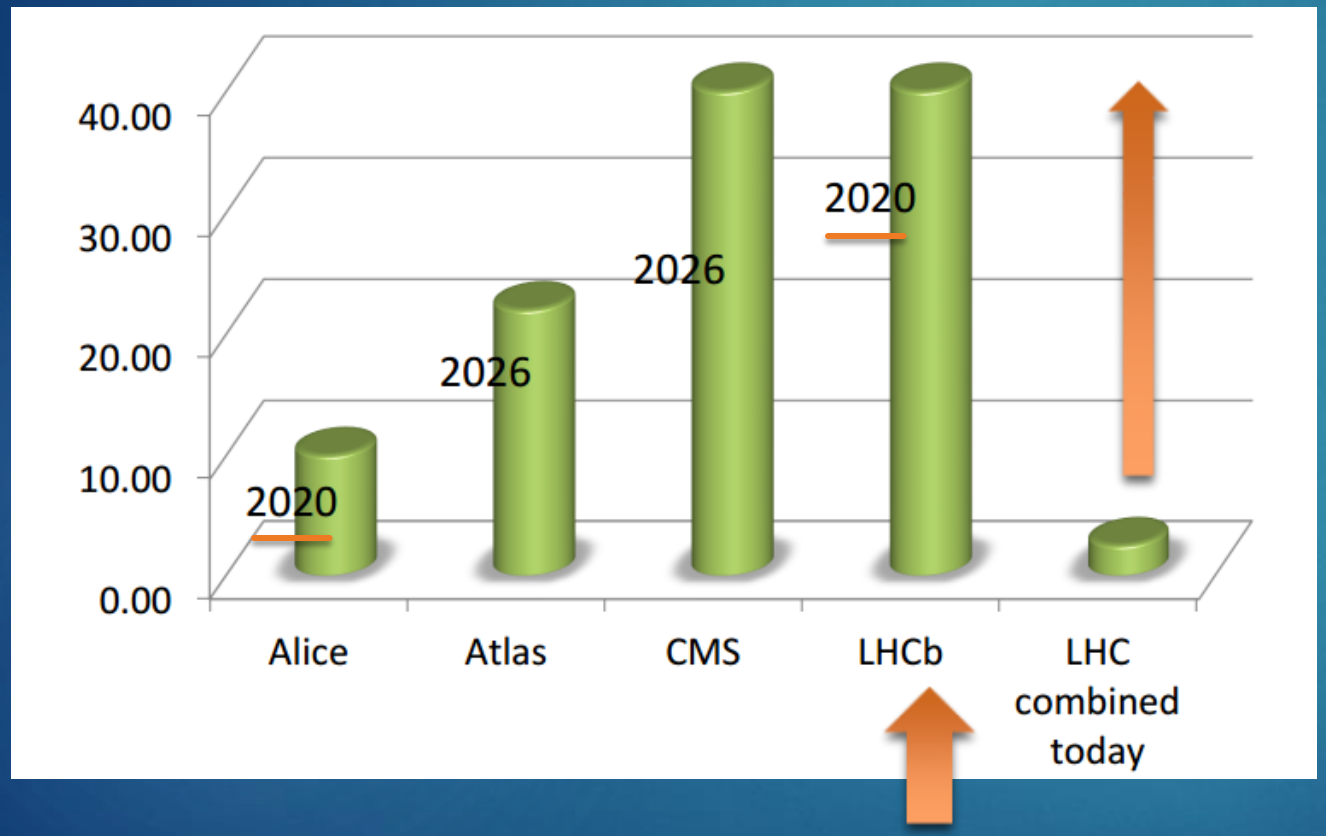
THANK YOU.

Studying a 40 Tb/s data acquisition system for the LHCb experiment

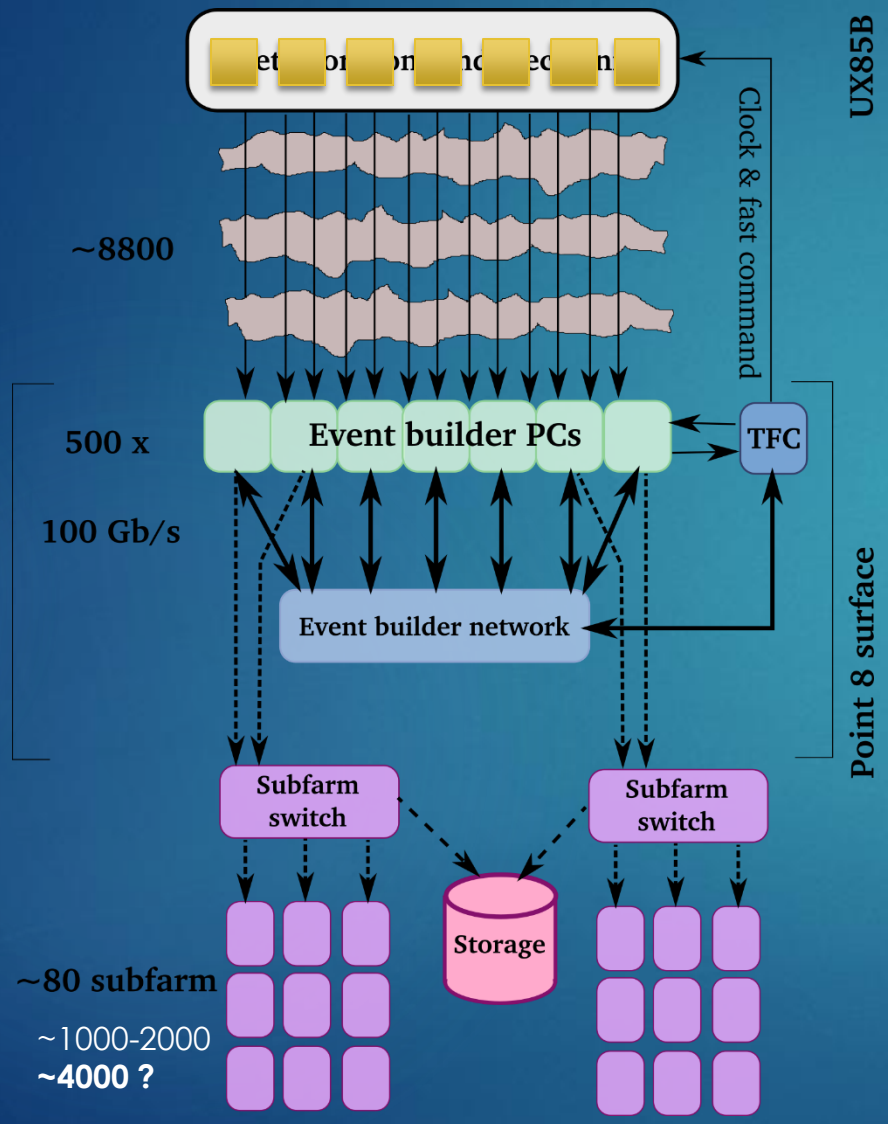
LHCb, an upgrade for 2018-2020

- Update of sub-detectors
- Removal of hardware trigger
  - Currently in **FPGA** and **analogic** components
  - Have to answer in **real time**
  - **Hard to maintain** and update
- New status :
  - Larger **event rate** (1 Mhz to 40 Mhz)
  - Larger **event size** (50 KB to ~100 KB)
- Much more data for DAQ & Trigger (x80 => 40 Tb/s)

40 Tb/s for LHCb in 2020

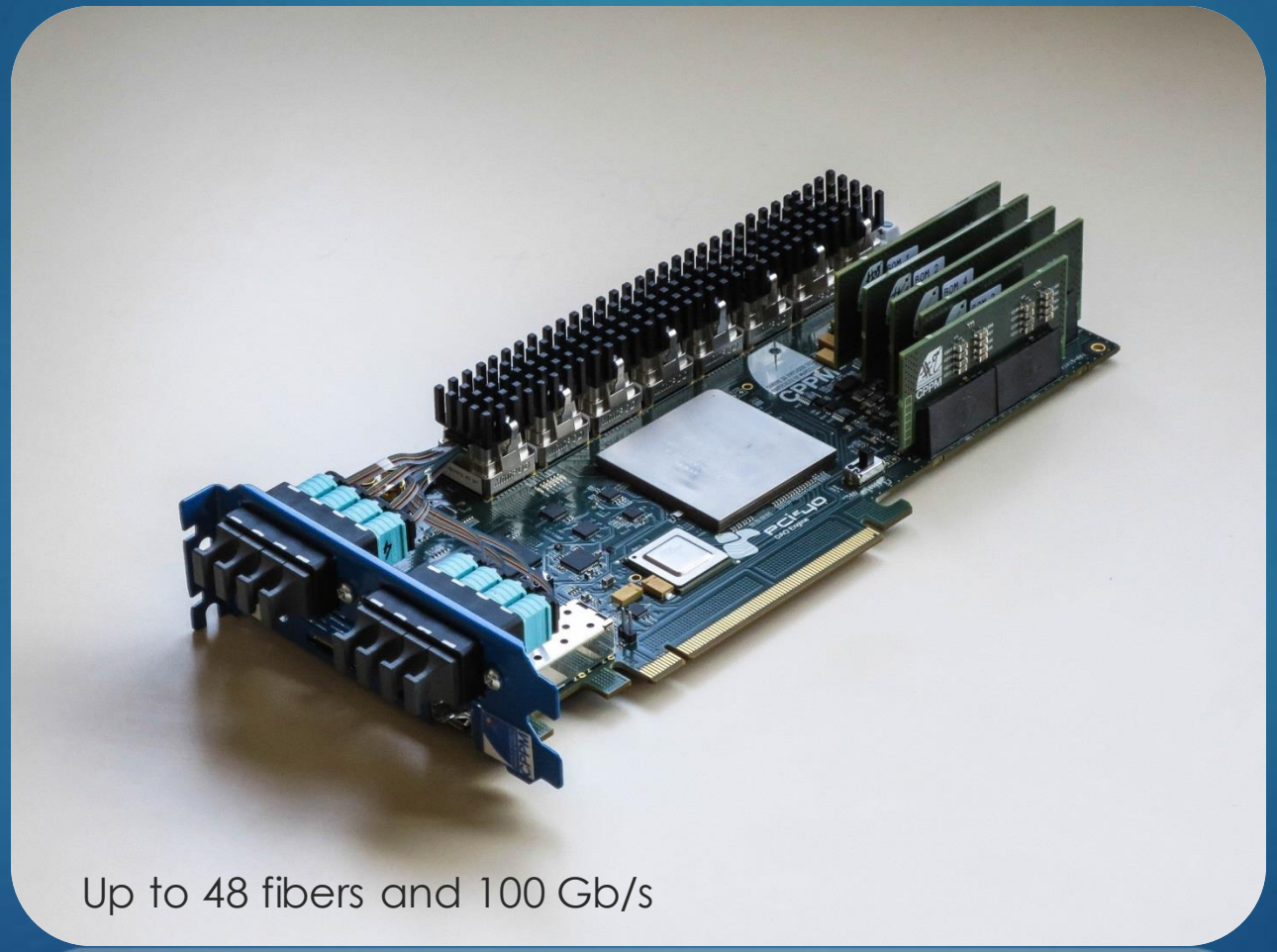


Data flow



- Numbers
  - **~8800** optical links going out from detector to the surface (**~100 m)** and up to **~4.8 Gb/s** each.
  - **~500** readout nodes (up to 24 input links each)
  - Filter farm of **O(1000)** nodes
- 40 Tb/s to transport
  - Can run over **512 nodes**
  - Need a **100 Gb/s** network
  - Considering net. load at **80%**
  - So **80 Gb/s**
  - Current estimation is **70 Gb/s**

Starting point : the acquisition board

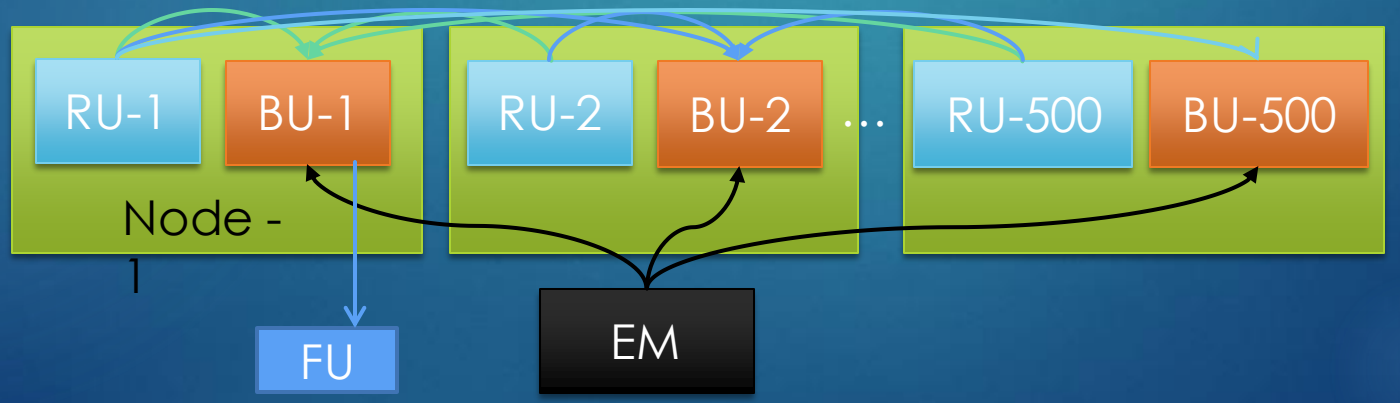


Up to 48 fibers and 100 Gb/s

Event building network topology

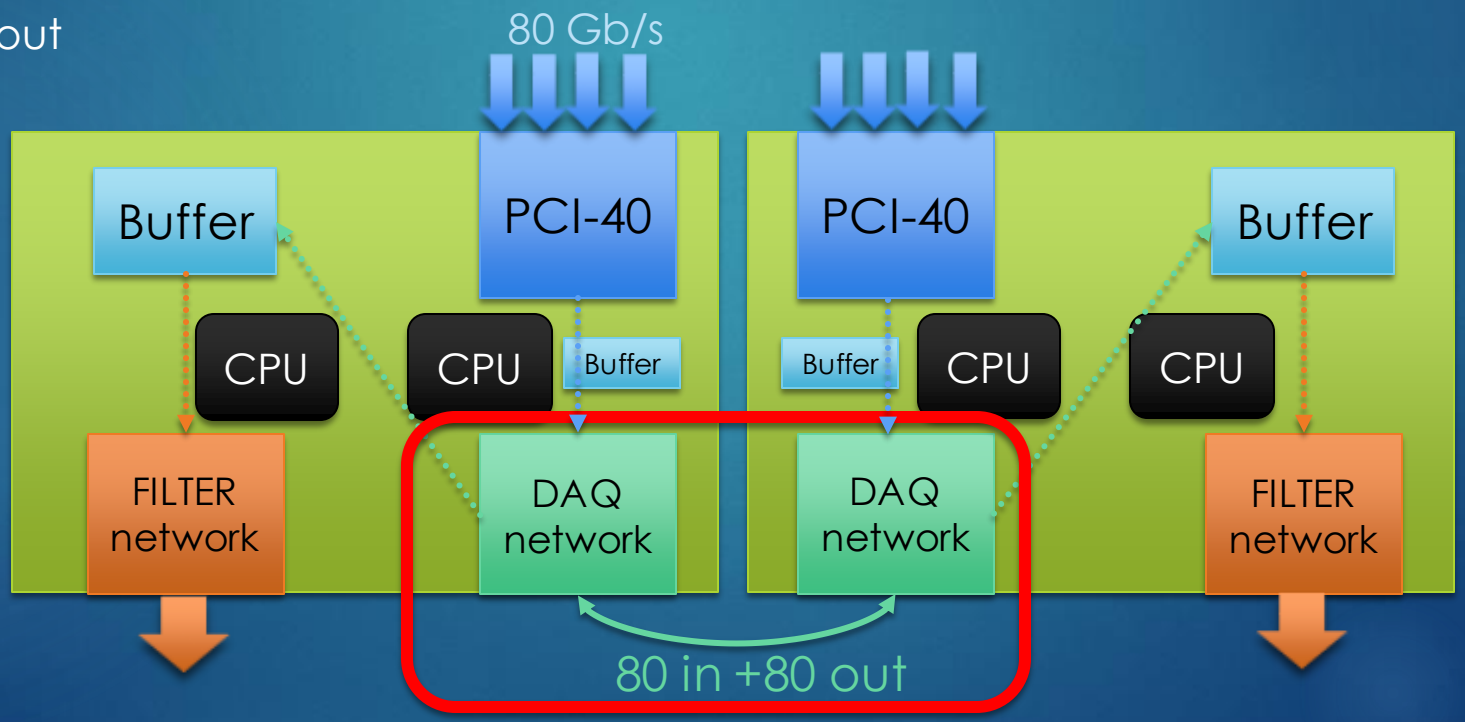
- Work with 4 units:
  - **Readout unit (RU)** to **read** data from PCI-E readout board
  - **Builder unit (BU)** to **merge** data from **Readout unit** and send to **Filter unit.**
  - **Filter unit (FU)** to select the interesting data (**not considered here**)
  - **Event manager (EM)** to dispatch the work over **Builder unit** (credits)

- RU/BU mostly does a "all-to-all"
  - To **aggregate** the data chunks from each collision



IO nodes hardware

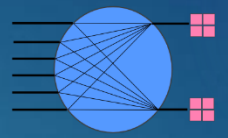
- Three IO boards at 100 Gb/s per node:
  - PCI-40 for fiber input
  - Event building network
  - Output to filter farm
- Also stress the memory (320 Gb/s of total traffic)
  - 80 in + 80 out



The DAQ network technologies

- We expect to use HPC networks
  - 100 Gb/s (expect running at 80 Gb/s)
- Technologies we looked on:
  - **Infiniband** EDR (HDR 200 Gb/s ?)
  - Intel **Omni-path** (~~version 2, 200 Gb/s ?~~)
  - 100 Gb/s **Ethernet**
- Not as HPC apps
  - We need to **fully fill the network continuously**
  - In **full duplex**
  - Bad pattern : **many gathers** (all-to-all) !
- Think using a fat-tree topology

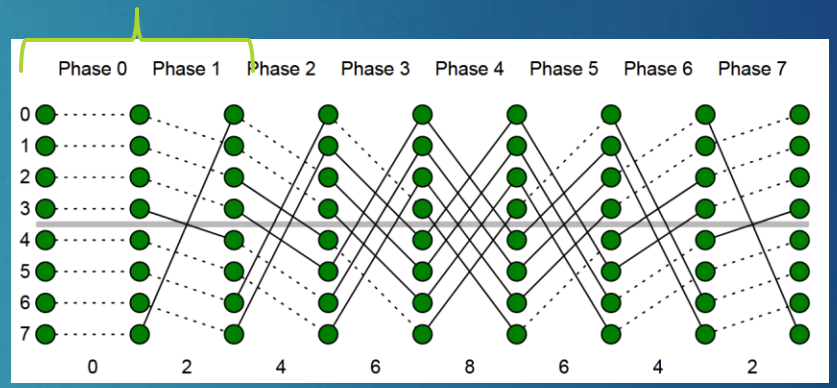
DAQPIPE



- A benchmark to evaluate event building solutions
  - DAQ Protocol Independent Performance Evaluator
  - Provides EM/RU/BU units
- Support various APIs
  - **MPI**
  - Libfabric
  - PSM2
  - Verbs
  - TCP / UDP
  - RapidIO
- Multiple protocols
  - PUSH
  - **PULL**

Communication control parameters

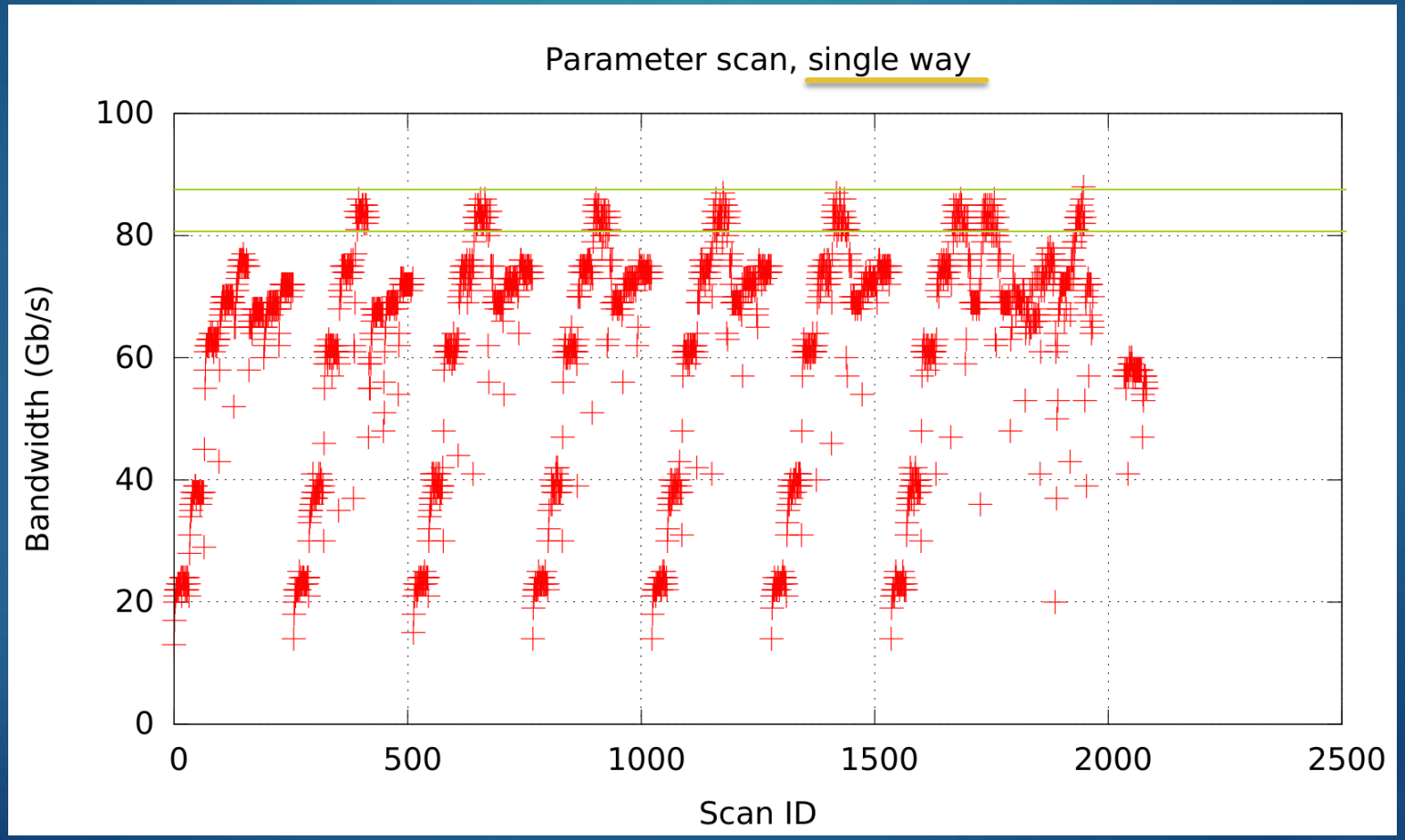
- Manage communication scheduling models
  - **Barrel shift** ordering (with N on-fly)
  - **Random** ordering (with N on-fly)
  - Send **all in one go**
- Messages size
  - best **~512 KB**
- Parallel gathers (credits)
  - best **~2**
- Parallel communication per gather
  - With how many nodes we communicate
  - best **~8**
  - So in total **2*8** communication in flight
- Process per nodes, 1 or 2
  - best **2**

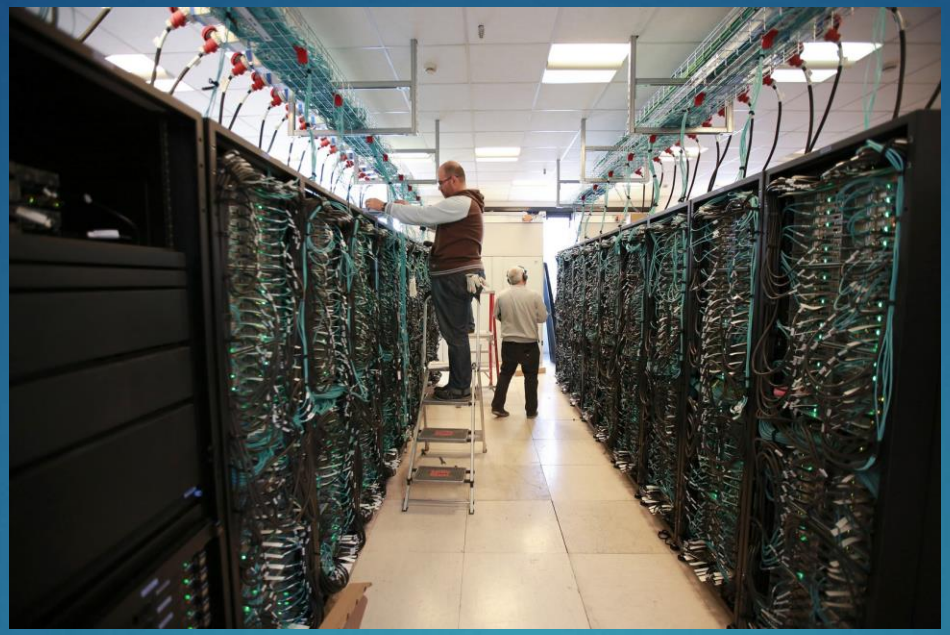


* From Bandwidth-optimal all-to-all exchanges in fat tree networks

What we have seen here on our cluster with 16 nodes OPA

87 Gb/s = **2%** of the points

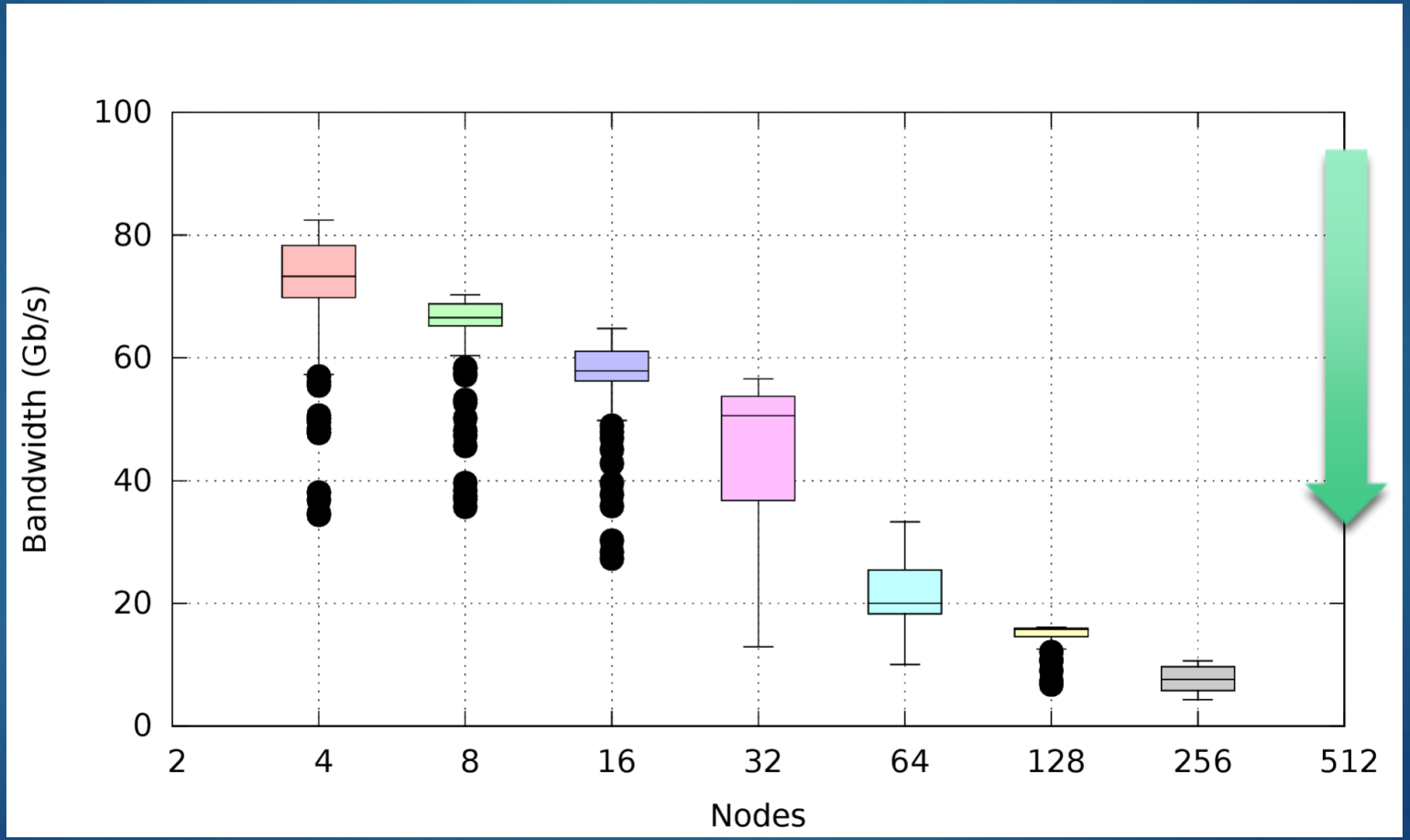




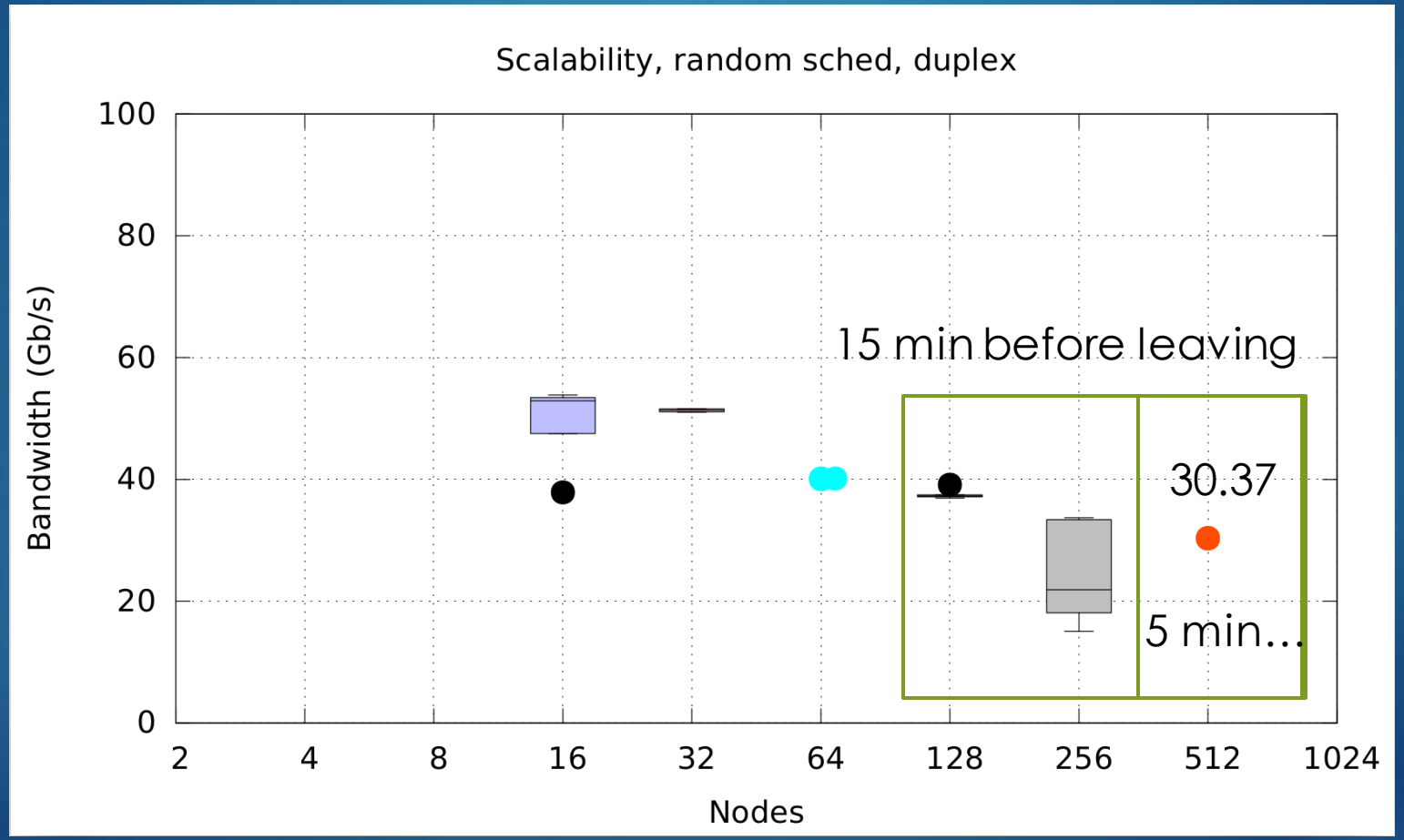
From http://www.flickriver.com/photos/tags/supercomputer/interesting/

First test at scale (512): hitting the wall ☹

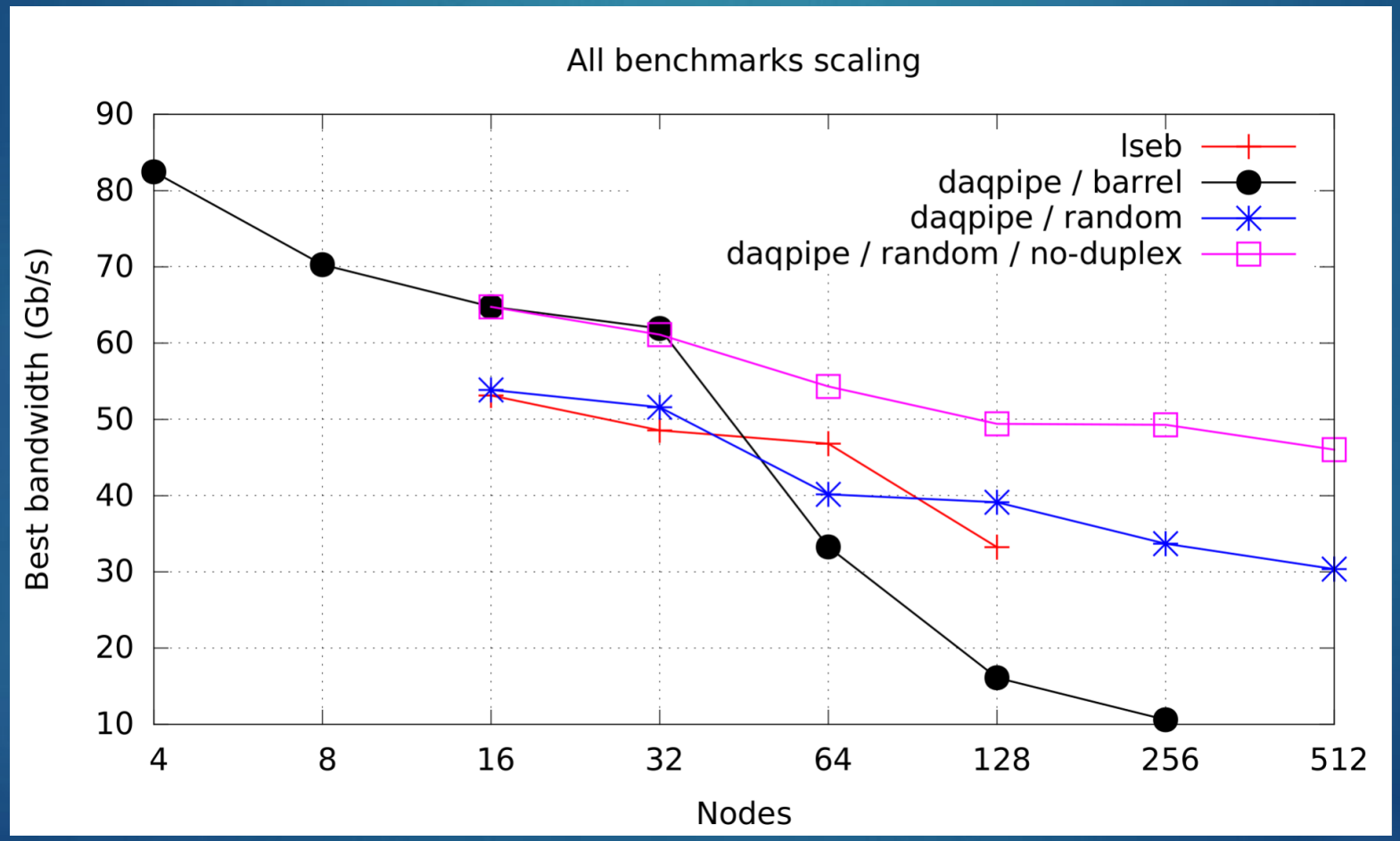
Summary scalability on Day-1 …. ☹



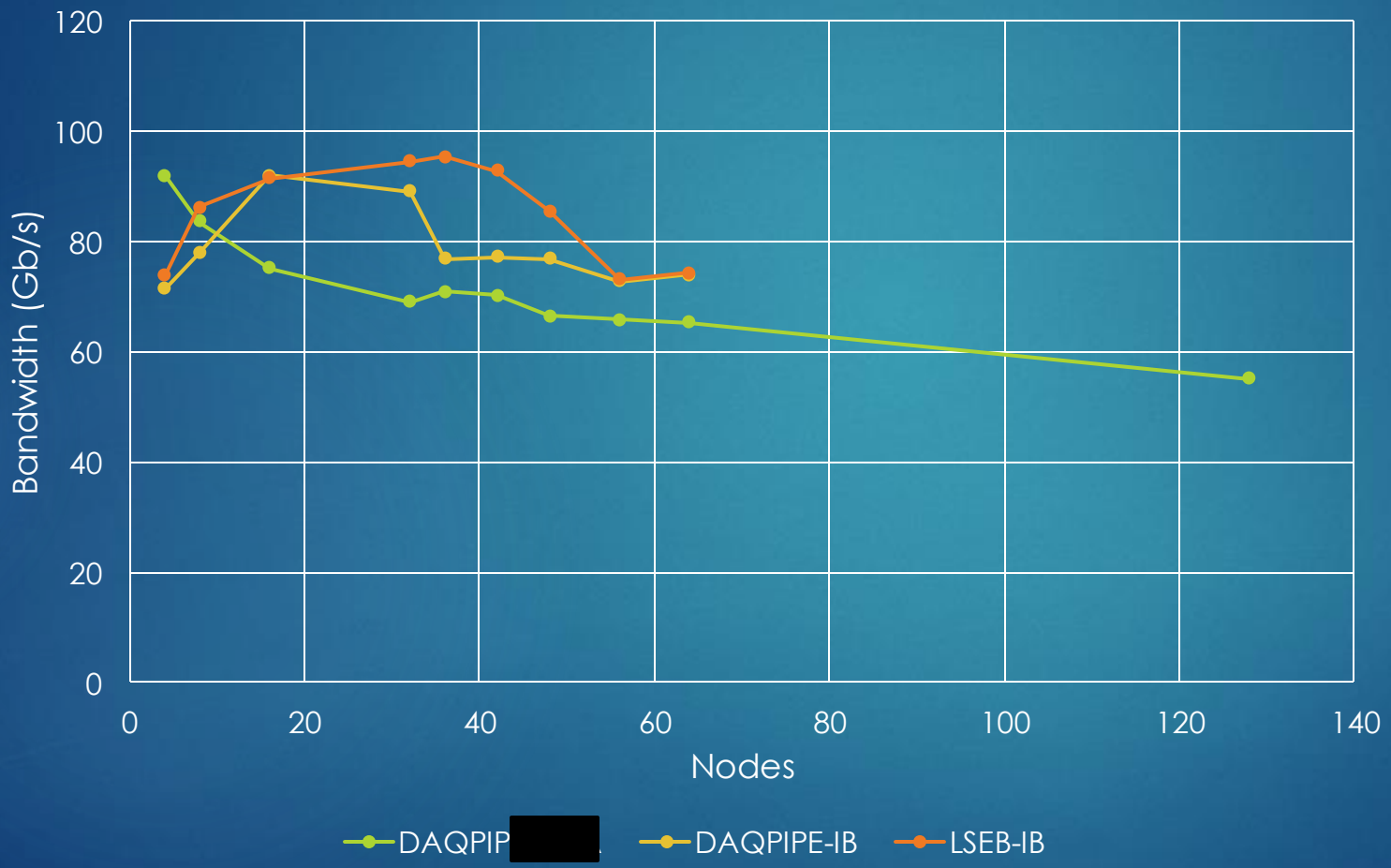
Day-2, 2 hours before leaving try random scheduling



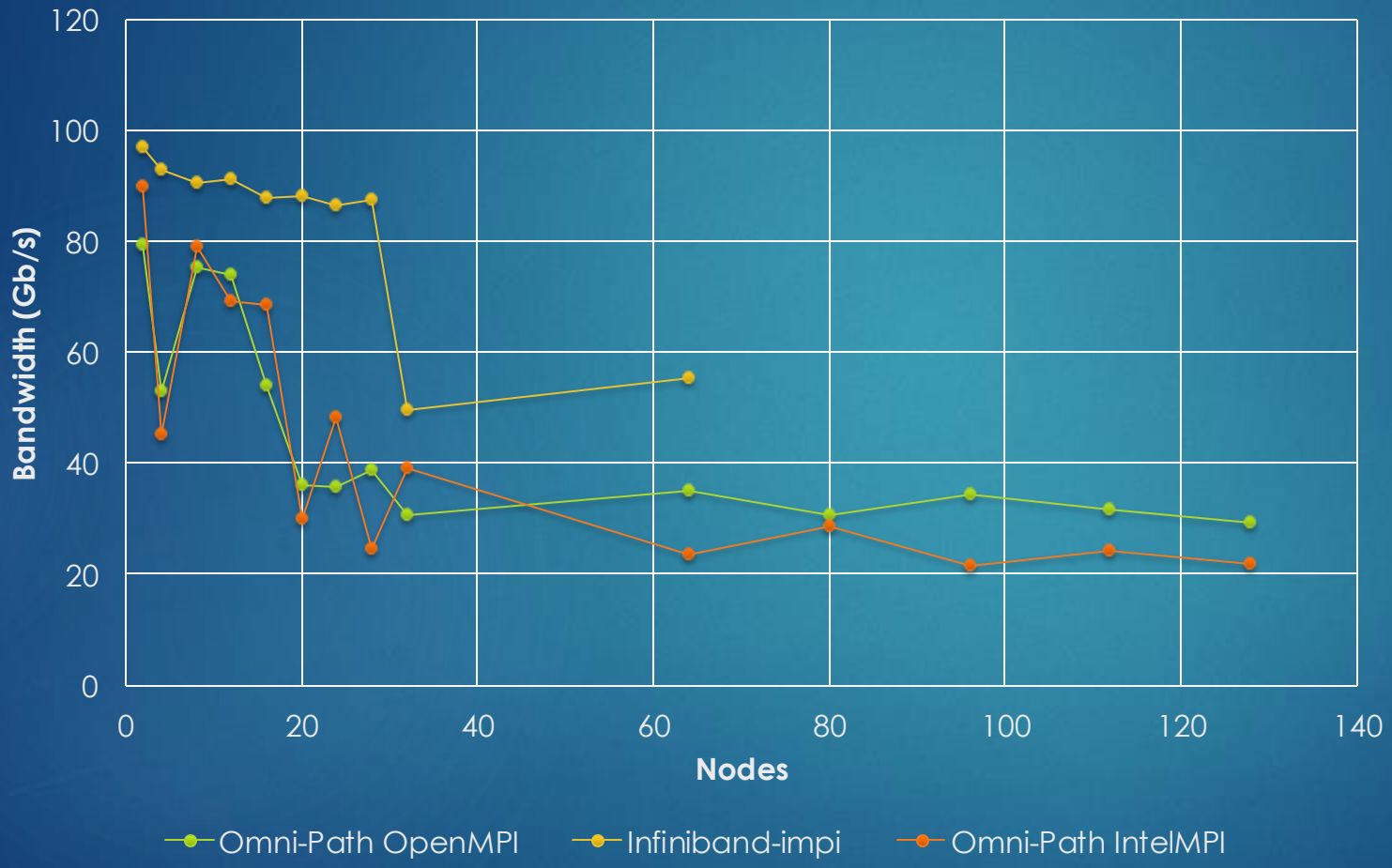
Final summary



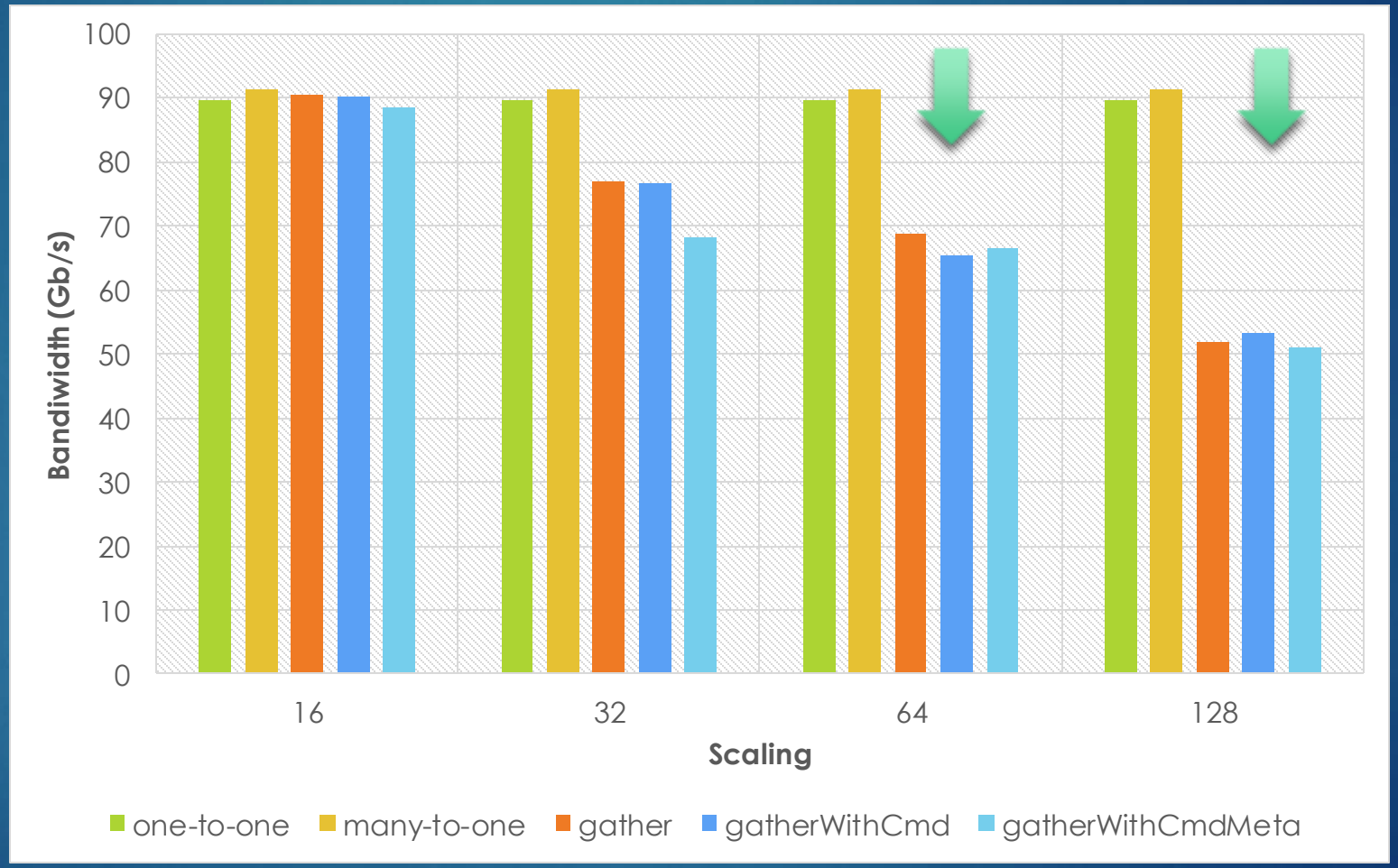
Another test on two clusters



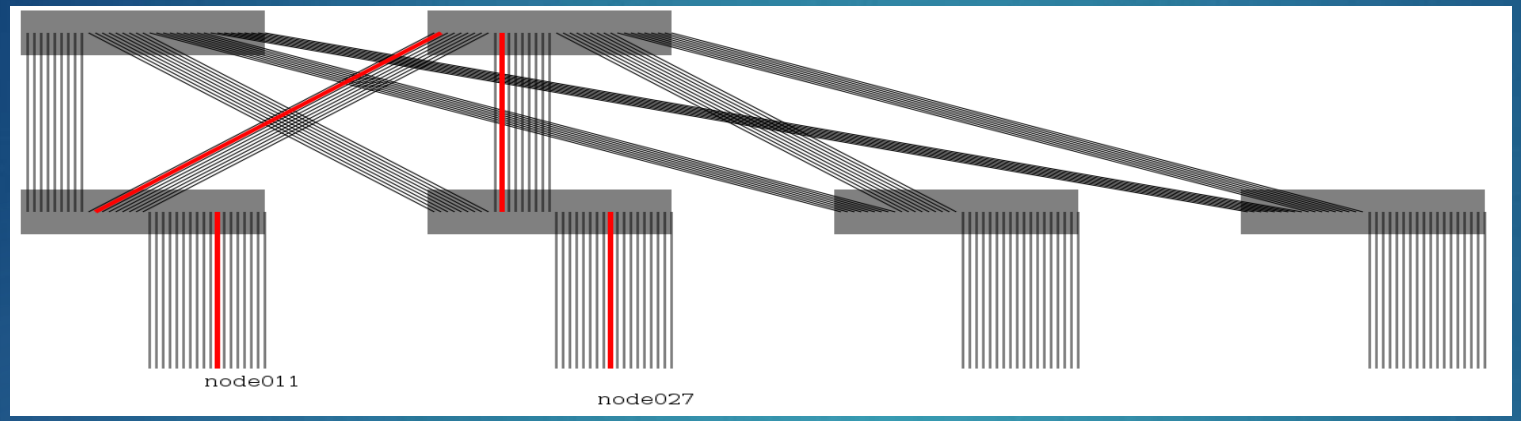
MPI_Alltoall

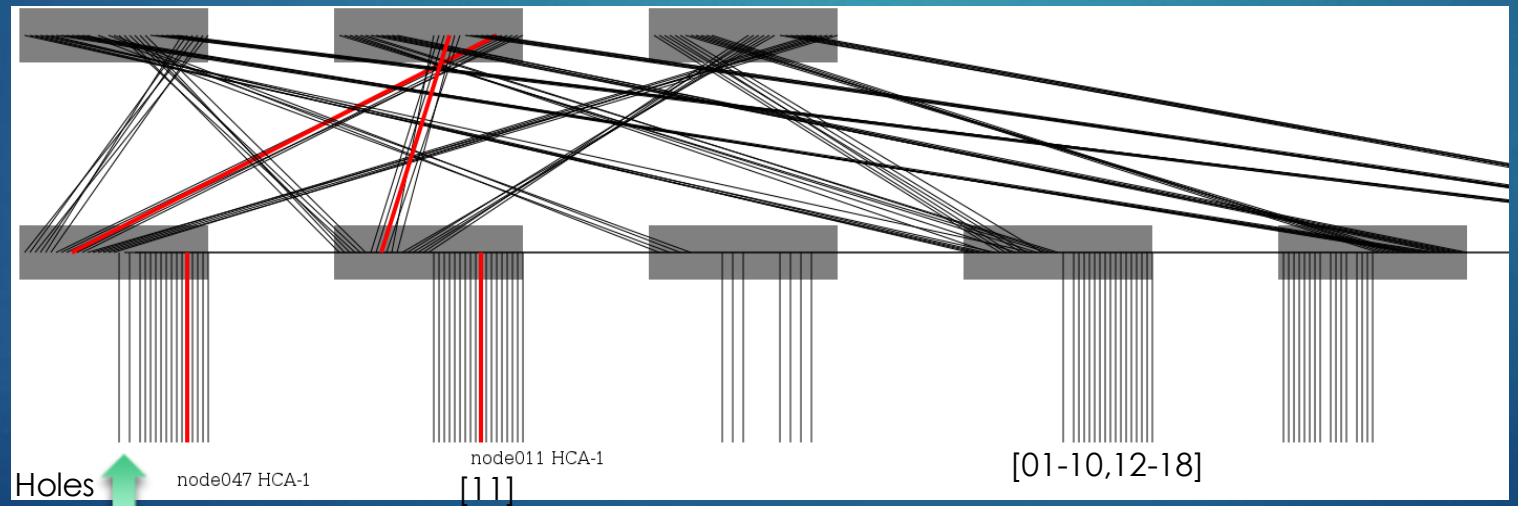


Micro-benchmarks on OPA



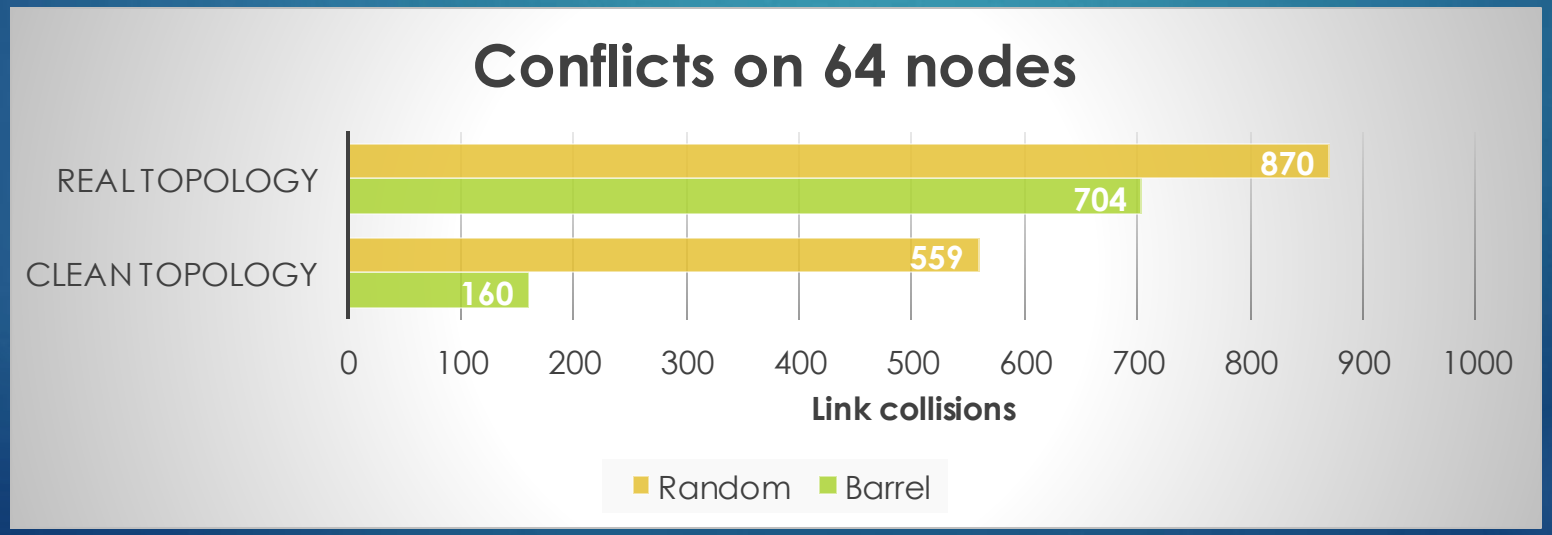
Ideal and real topologies





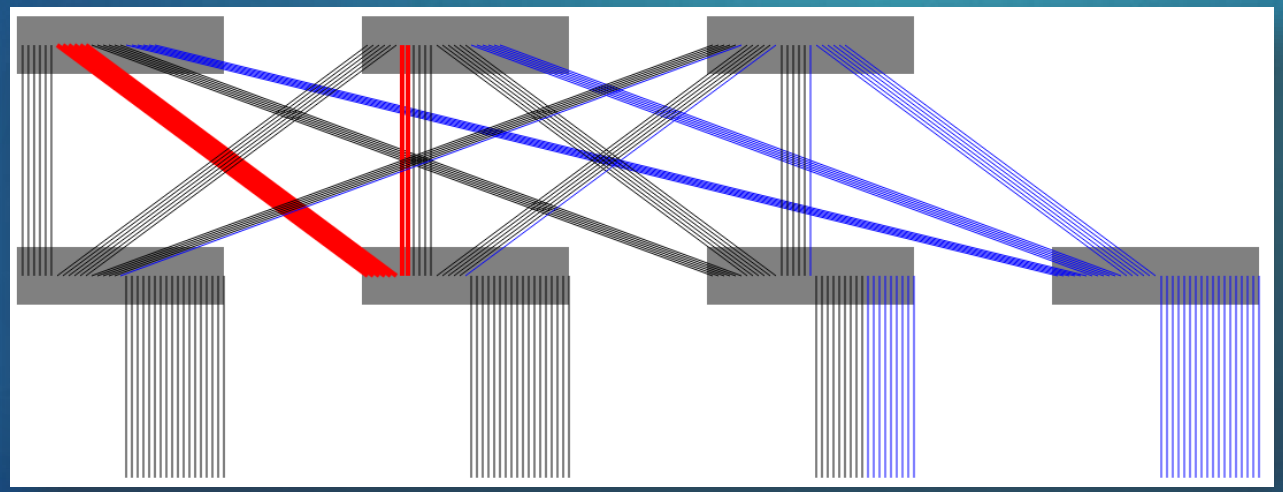
Conflicts

- The real topology implies many conflicts
- Due to :
  - routing table
  - missing nodes in leaf switches
- 64 nodes imply 4096 communications
  - 870 conflicts => **21%**, 704 conflicts => **17%**, 160 conflicts => **3%**

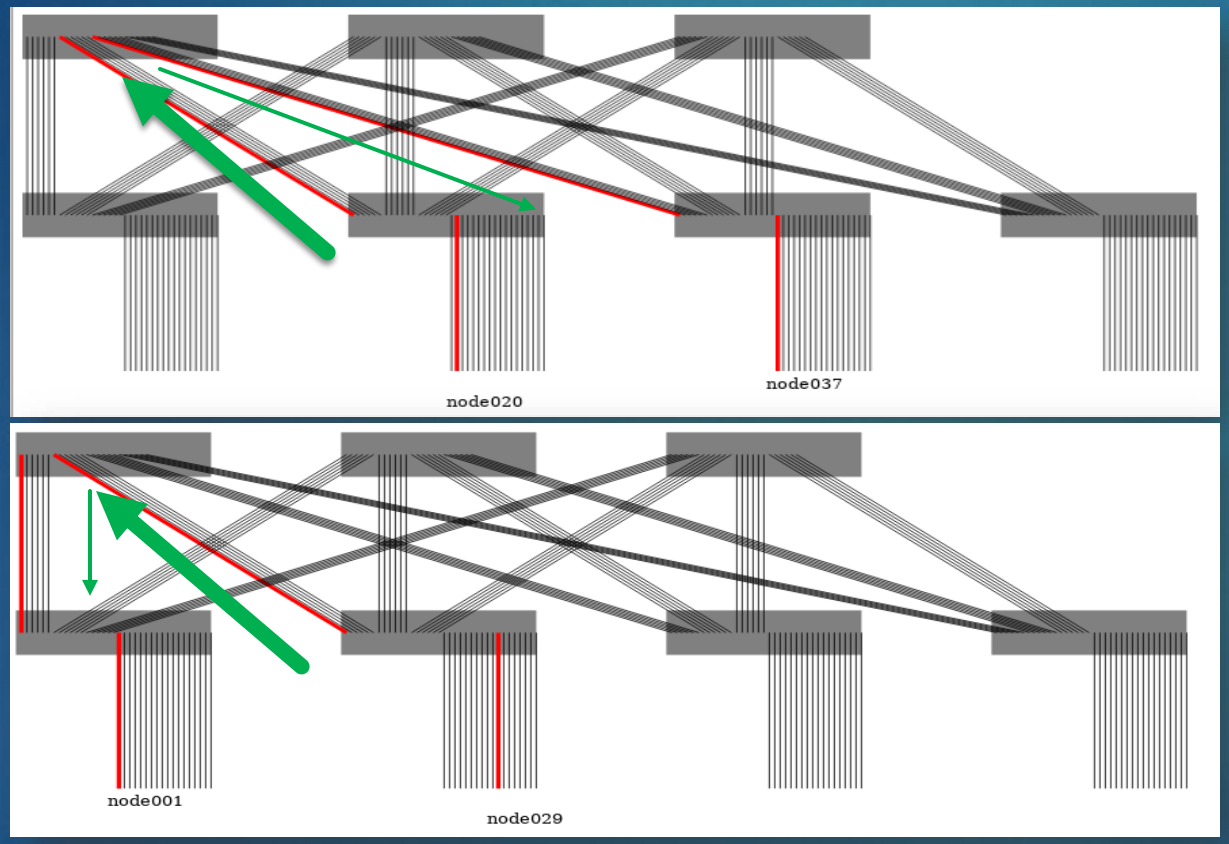


Example on clean topology

- Switch **radix : 36**
- Run on **45 nodes** (2 * 18 **+ 9**)
- Look on **step 28** => **9 conflicts**



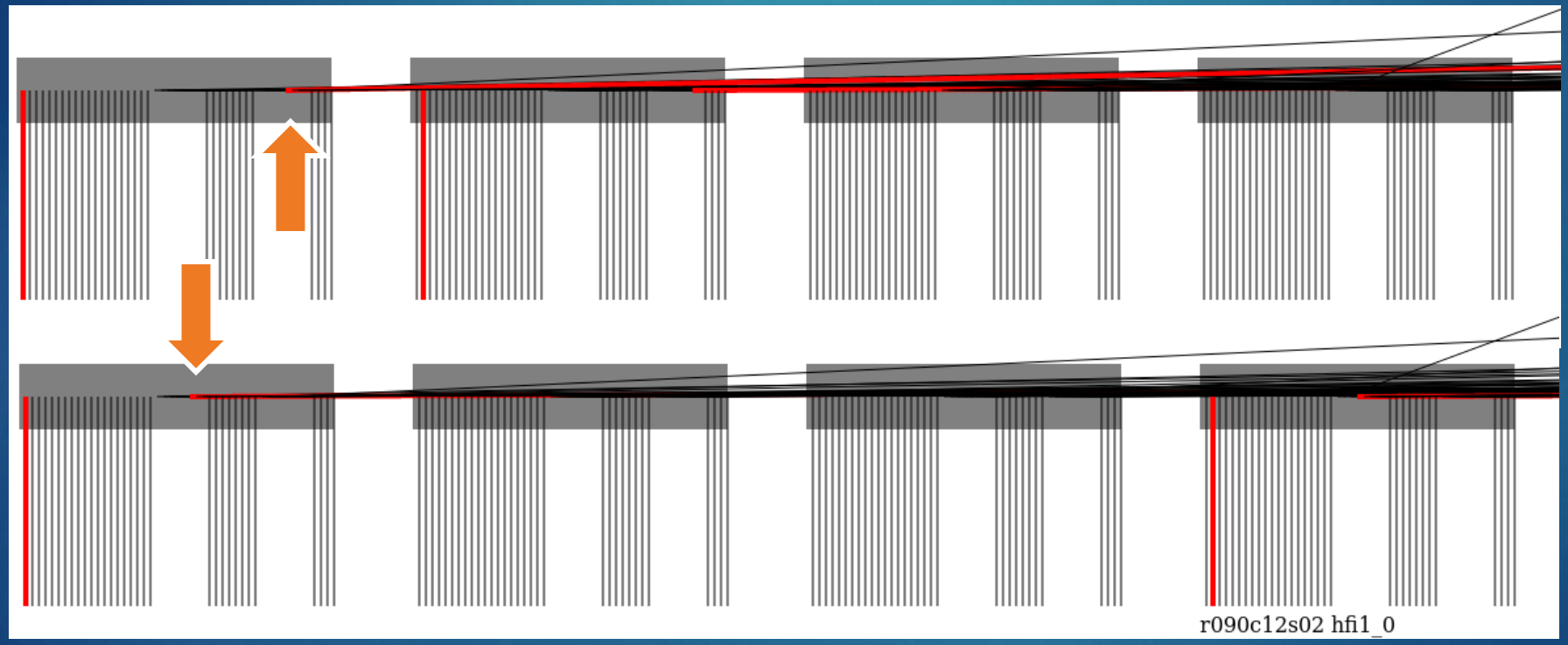
Extract two conflicts to understand



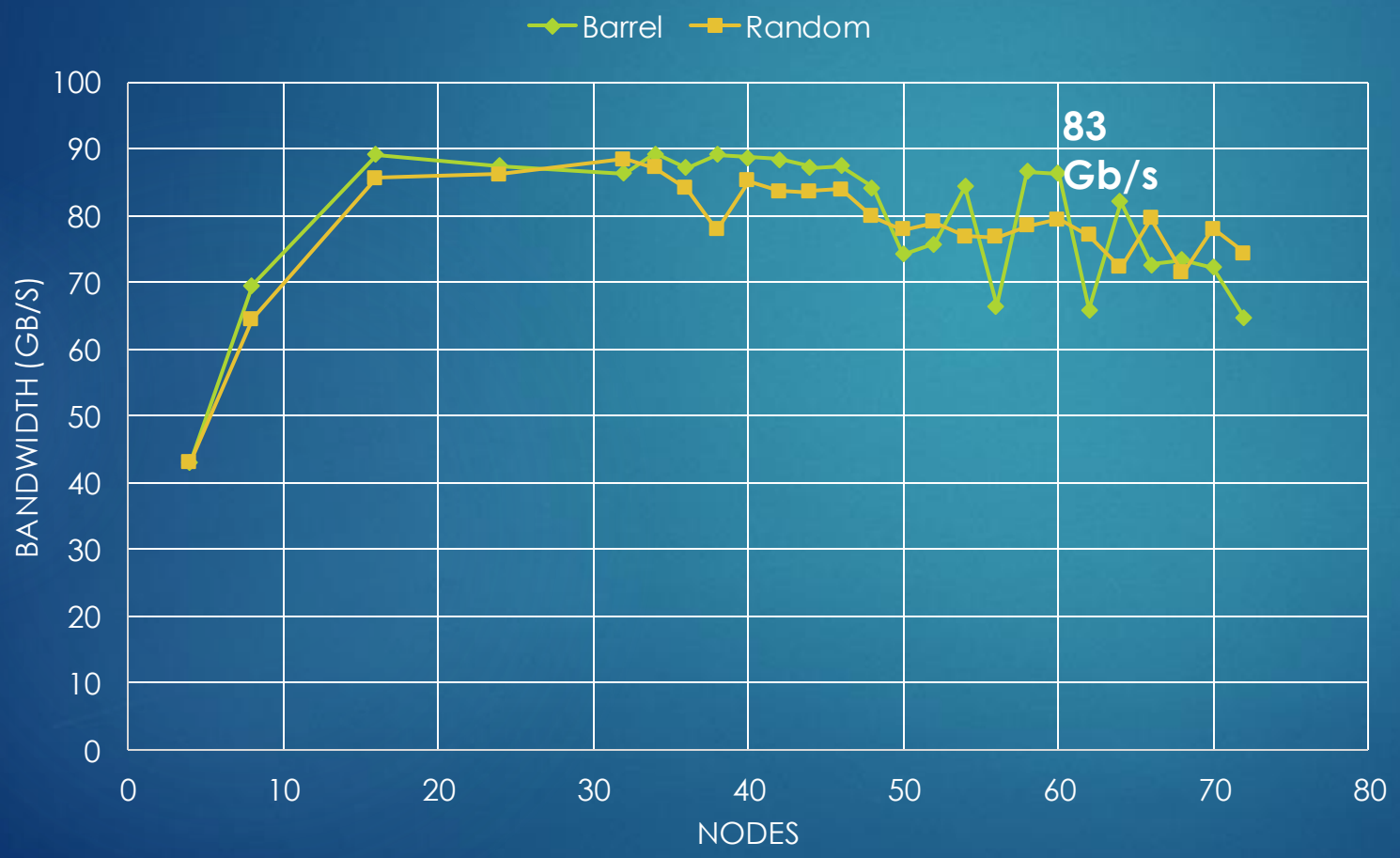
- Example, **2 conflicts** of **step 28**
  - **20 -> 37**
  - **29 -> 1**
- They use the same **UP** link

Default routing tables OPA

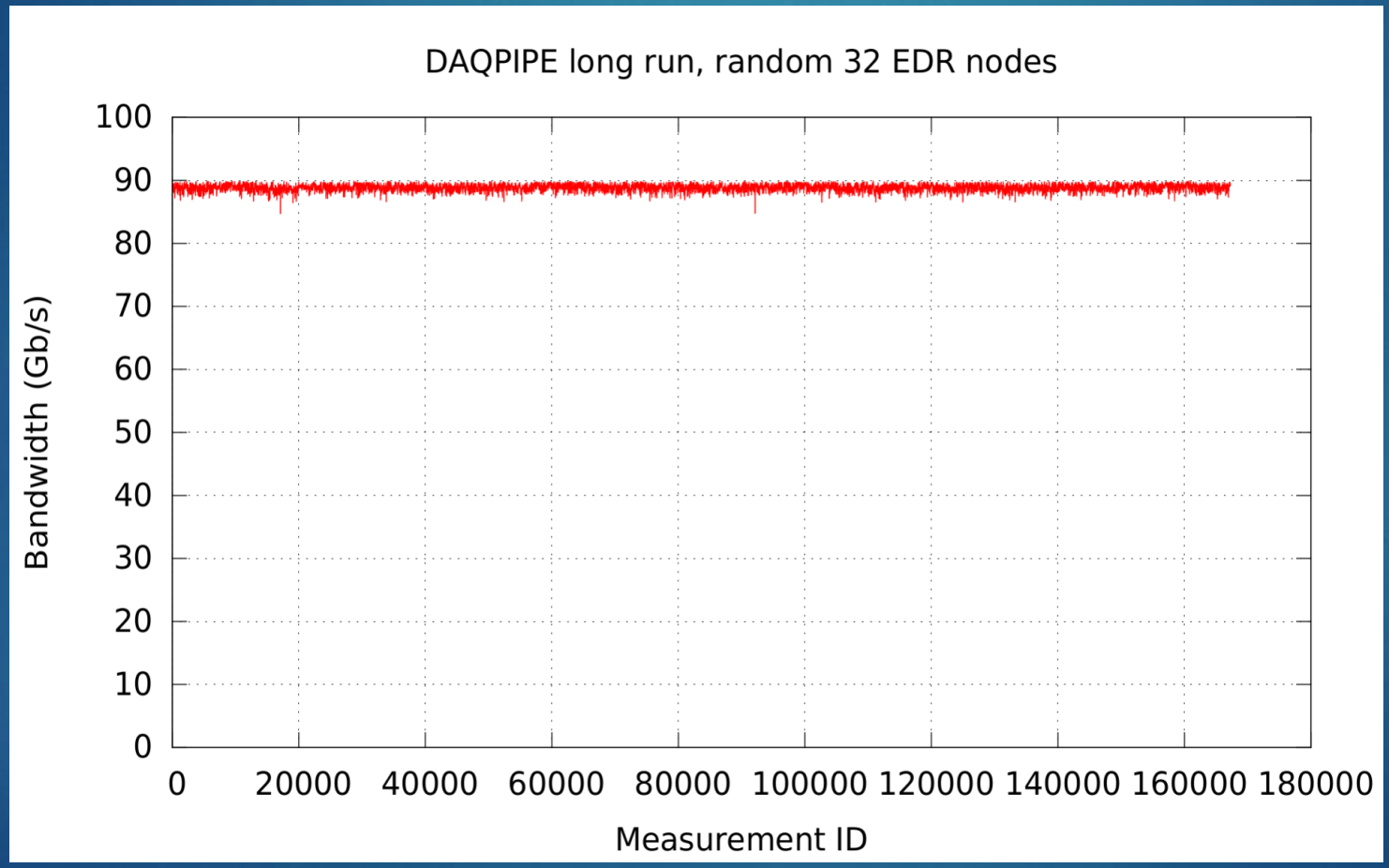
Not same up-link used to reach N-eme node on leaf switches



Last results on Tech B (LENOVO)
This time with close to perfect cabling

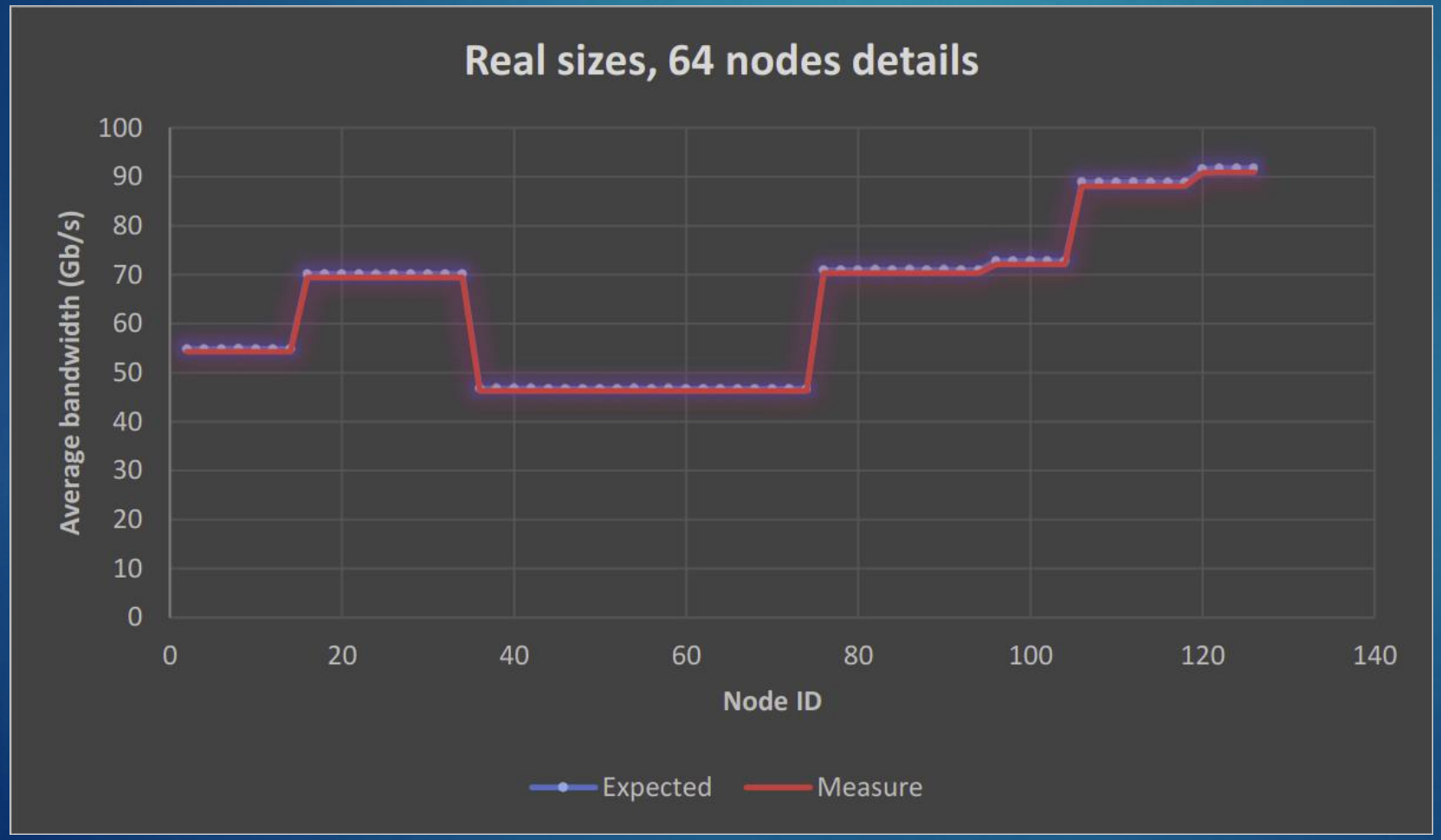


Stability over time (3 hours)



Real unbalanced **sub-detectors** bandwidth

Finally that's should be OK with IB/EDR.... but not so much margins



Conclusion

- That was a long road

- Finally IB/EDR should be OK
- Would sleep better if IB/HDR (200 Gb/s) will be available in time
- OPA need work on routing (intel did it, need scale tests)

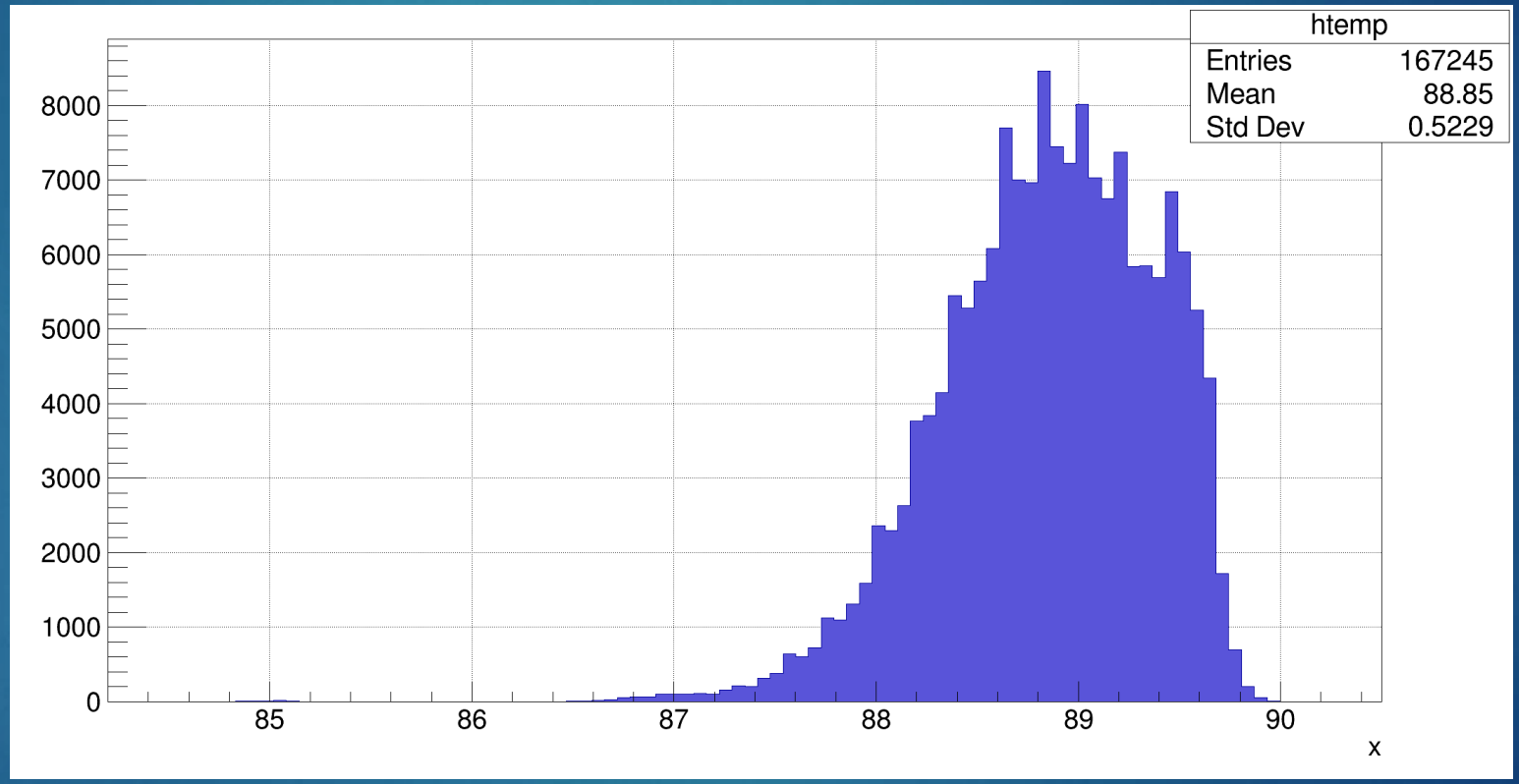
- We also ave failure recovery
- Is needed ? MPI restart ~30s ?

- Current prospective about 100Gb/s ethernet (via iwap/RoCE)

Backup

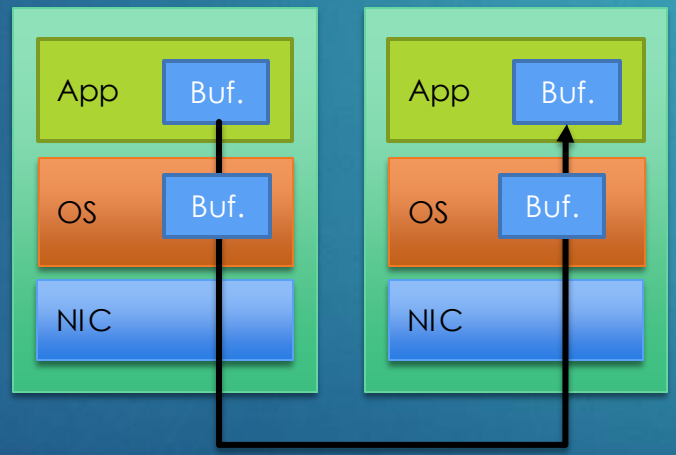
Htopml integration
Communication scheduling

Stability over time (3 hours)



RDMA
Remote Direct Memory Access

- **Communication with Ethernet**
  - Traverse all **IP stack** in **OS**
  - Internal **memory copies** (use mem. **bandwidth** & **CPU**)
  - Higher latency



- **OS by pass in InfiniBand or Omni-Path**
  - **RDMA** : Remote Direct Memory Access
  - No memory copies
  - **Lower latency**
  - **CPU not involved**