arm

Efficient HPC Development with Allinea Forge

DKRZ, Hamburg

10/10/2017

Florent Lebeau
Florent.Lebeau@arm.com

Agenda

- Debugging and profiling MPI applications at DKRZ
- Analysing memory issues
- Detecting deadlocks
- MPMD applications
- Best practices

About Allinea Tools

- Allinea Tools: leading toolkit for HPC application developers
  - Available on 65% of the top 100 HPC systems
  - Help maximise application efficiency with Performance Reports
  - Help the HPC community design the best applications with Forge
    - Available at DKRZ: 1024 tokens
- As of December 2016 Allinea is now part of ARM
  - Allinea objective: continue to be the trusted HPC Tools leader in tools across every platform
- This means:
  - The same team will continue to work with you, our customers and partners, and the wider HPC community
  - Being part of ARM gives us strength to deliver on our roadmap faster
  - We remain 100% committed to providing cross-platform tools for HPC
  - Our engineering roadmap is aligned with upcoming architectures from every vendor

ARM HPC Tools

The mission:

Enable the software ecosystem for large-scale ARM systems.

Based in Manchester and Warwick, UK.

Research Compilers	ARM Performance Libraries	Userspace Performance Tools	Open Source HPC	Allinea Tools
New compiler technology to support and evaluate next-generation ARM architecture.	Commercially-supported BLAS, LAPACK and FFT routines optimized for ARM-compatible microarchitectures.	New commercial tools to deliver actionable performance improvement advice to software developers.	Identification of issues in ARM builds of open-source packages and the upstreaming of fixes.	Parallel debugger, profiler and performance analysis tools for HPC

www.developer.arm.com/hpc

Debugging and Profiling MPI Applications

Print statement debugging

The first debugger: print statements

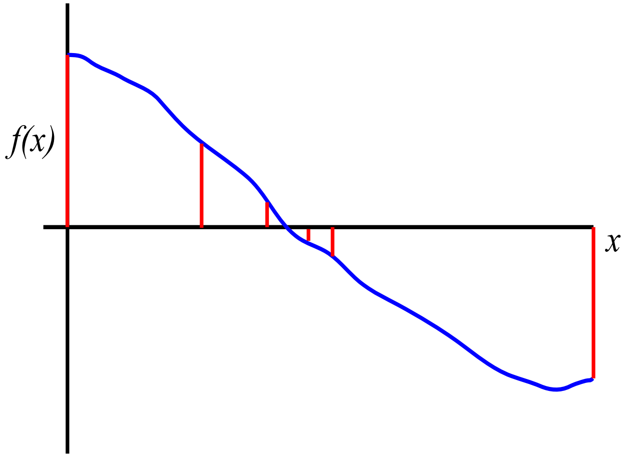
- Each process prints a message or value at defined locations
- Diagnose the problem from evidence and intuition

A long slow process

- Analogous to bisection root finding

Broken at modest scale

- Too much output – too many log files



Typical types of bugs

- Steady and dependable, I'll be there for you.

BOHR BUG

- Oh, you are debugging? Let me hide for a sec!

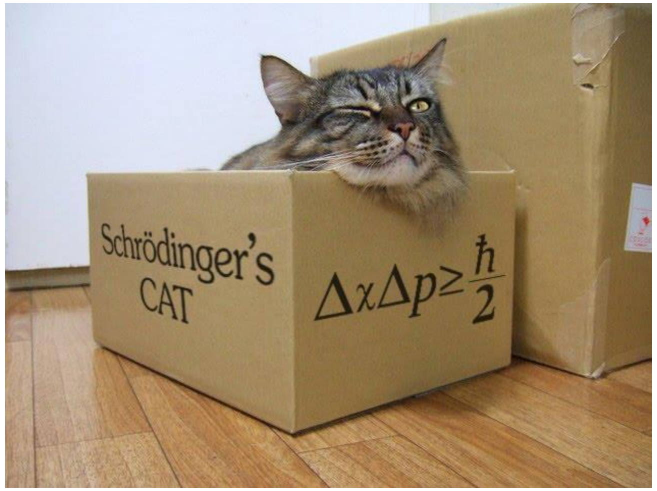
HEISEN BUG

- Chaos is my name and you shall fear me.

MANDEL BUG

- I am buggy ***AND*** not buggy. How about that?

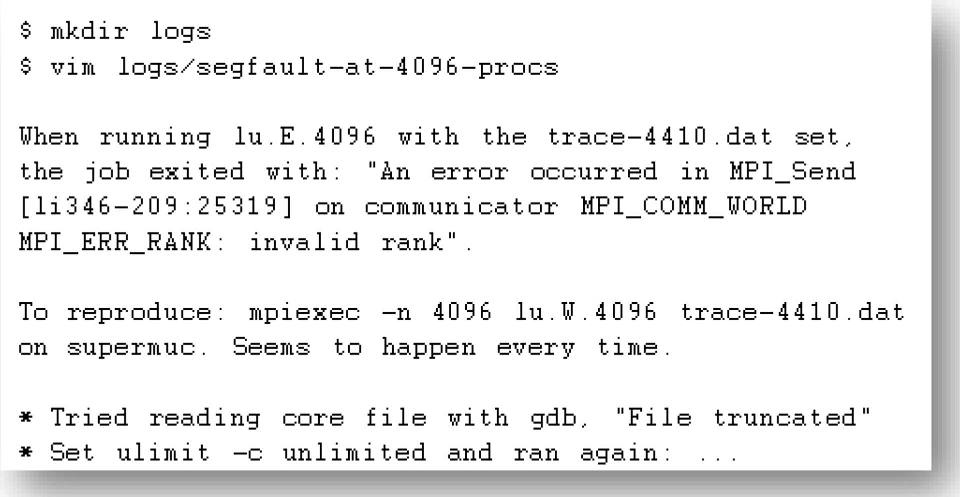
SCHRODIN BUG



Debugging by discipline

Debugging a problem is much easier when you can :

- Make and undo changes fearlessly
  - Use a source control (CVS, …)
- Track what you've tried so far
  - Write logbooks
- Reproduce bugs with a single command
  - Create and use test script

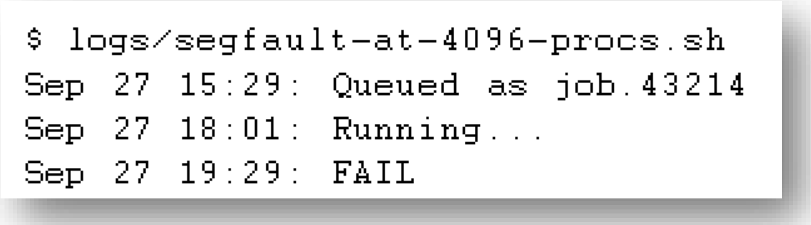


```
$ mkdir logs
$ vim logs/segfault-at-4096-procs

When running lu.E.4096 with the trace-4410.dat set,
the job exited with: "An error occurred in MPI_Send
[li346-209:25319] on communicator MPI_COMM_WORLD
MPI_ERR_RANK: invalid rank".

To reproduce: mpiexec -n 4096 lu.W.4096 trace-4410.dat
on supermuc. Seems to happen every time.

* Tried reading core file with gdb, "File truncated"
* Set ulimit -c unlimited and ran again: ...
```



```
$ logs/segfault-at-4096-procs.sh
Sep 27 15:29: Queued as job.43214
Sep 27 18:01: Running...
Sep 27 19:29: FAIL
```

Allinea DDT helps to understand

Who had a rogue behaviour ?

- Merges stacks from processes and threads

Where did it happen?

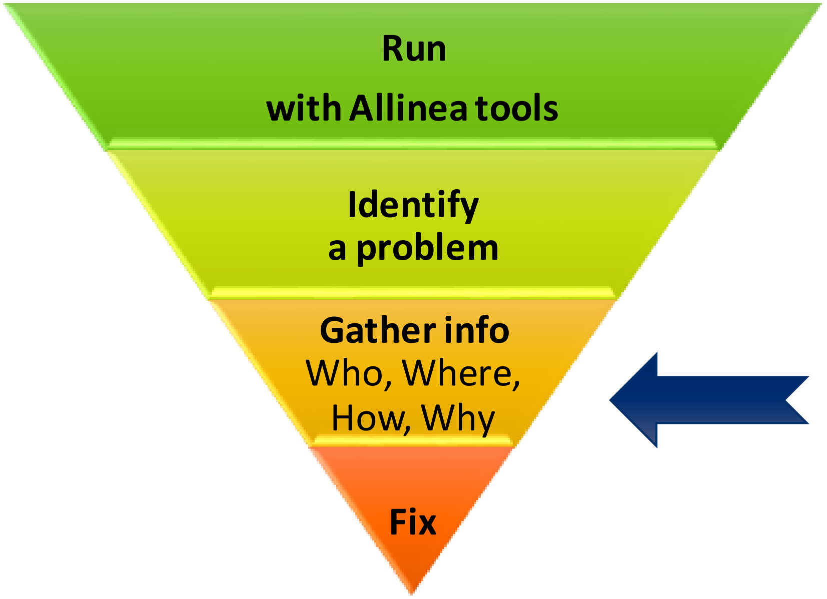
- Allinea DDT leaps to source automatically

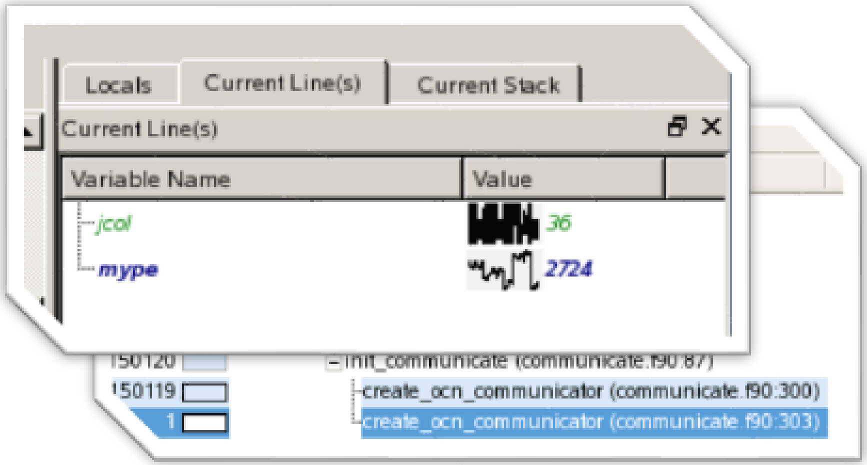
How did it happen?

- Detailed error message given to the user
- Some faults evident instantly from source

Why did it happen?

- Unique “Smart Highlighting”
- Sparklines comparing data across processes





Variable Name	Value
jcol	36
mype	2724

```
150120      − init_communicate (communicate.f90:87)
150119        create_ocn_communicator (communicate.f90:300)
     1        create_ocn_communicator (communicate.f90:303)
```

Allinea DDT cheat sheet

Prepare the code

-
  ```
  $ mpicc -O0 -g myapp.c –o myapp.exe
  ```

Load the environment module

-
  ```
  $ module load allinea-forge
  ```

Start Allinea DDT in interactive mode (in an interactive job session)

-
  ```
  $ ddt srun ./myapp.exe arg1 arg2
  ```

Or use the reverse connect mechanism (by submitting a batch job)

- On the login node:

  -
    ```
    $ ddt &
    ```
- (or use the remote client http://www.allinea.com/products/downloads/)
- Then, edit the job script to run the following command and submit:

  -
    ```
    ddt --connect mpirun -n 8 ./myapp.exe arg1 arg2
    ```

Example 1

Copy the archive in your working directory

- $ cp /scratch/k/k203064/flebeau/allinea_workshop.tar.gz .
- $ tar xzvf allinea_workshop.tar.gz
- $ cd allinea_workshop

Load the environment

- $. env

And go to the first exercise

- $ cd 1_interactive_debugging/

Compile with:

- $ make

And submit the job

- $ sbatch job.sub

The initial application crashes

Recompile for debugging with:

- $ make DEBUG=1

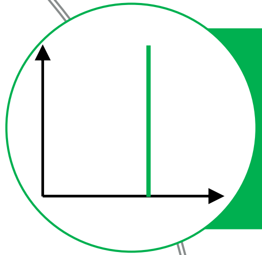
Launch Allinea DDT on the login node, edit the job script to prefix the execution command with “ddt --connect” and debug the application

How to profile?

Different methods

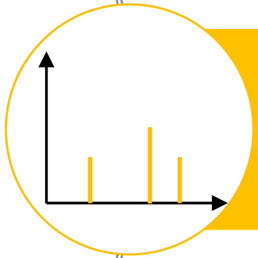
- Tracing
  - Records and timestamps all operations
  - Intrusive
- Instrumenting
  - Add instructions in the source code to collect data
  - Intrusive
- Sampling
  - Automatically collect data
  - Not intrusive

Some types of profiles



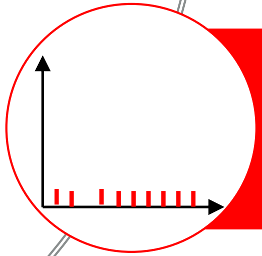
Hotspot

- One function corresponds to more 80% of the runtime
- Large speed-up potential
- Best optimisation scenario



Spike

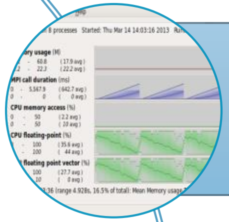
- The application spends most of the time in a few functions
- Speed-up potential depends on the aggregated time
- Variable optimisation time



Flat

- Runtime split evenly between numerous functions, each one with a very small runtime
- Little speed-up potential without algorithmic changes
- Worst optimisation scenario

Allinea MAP: Performance made easy

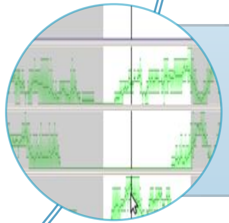


Low overhead measurement

- Accurate, non-intrusive application performance profiling
- Seamless – no recompilation or relinking required

Easy to use

- Source code viewer pinpoints bottleneck locations
- Zoom in to explore iterations, functions and loops



Deep

- Measures CPU, communication, I/O and memory to identify problem causes
- Identifies vectorization and cache performance

Allinea MAP cheat sheet

Prepare the code

- $ mpicc –O3 **-g** myapp.c –o myapp.exe

Load the environment module

- $ module load **allinea-forge**

Edit the job script to run Allinea MAP in “profile” mode

- $ **map --profile** srun ./myapp.exe arg1 arg2

Open the results

- On the login node:
  - $ map myapp_Xp_Yn_YYYY-MM-DD_HH-MM.map
- (or load the corresponding file using the remote client http://www.allinea.com/products/downloads/)

Example 2

Go to

- $ cd 2_profiling/

Compile with:

- $ make

Edit the job script to prefix the execution command with “map --profile” and submit the job

- $ sbatch job.sub

Analyse the profiling results

- $ map *.map

Analysing Memory Issues

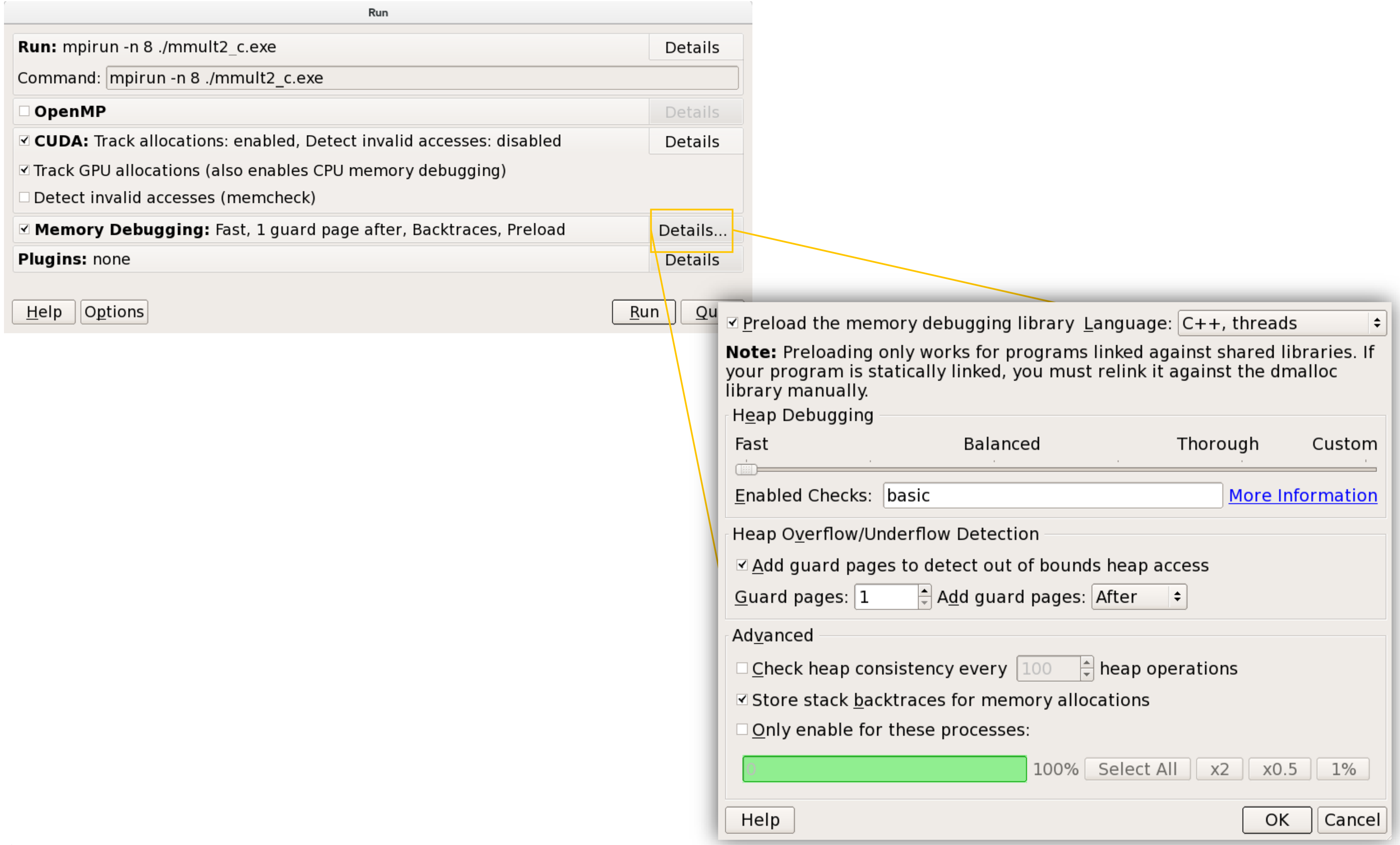
It works… Well, most of the time

SCHRODIN BUG !

A strange behaviour where the application "sometimes" crashes is a typical sign of a memory bug

Allinea DDT is able to force the crash to happen

Memory debugging menu in Allinea DDT



Run

Run: mpirun -n 8 ./mmult2_c.exe

Command: mpirun -n 8 ./mmult2_c.exe

□ **OpenMP**

☑ **CUDA:** Track allocations: enabled, Detect invalid accesses: disabled

☑ Track GPU allocations (also enables CPU memory debugging)

□ Detect invalid accesses (memcheck)

☑ **Memory Debugging:** Fast, 1 guard page after, Backtraces, Preload

Plugins: none

☑ Preload the memory debugging library Language: C++, threads

Note: Preloading only works for programs linked against shared libraries. If your program is statically linked, you must relink it against the dmalloc library manually.

Heap Debugging

Fast Balanced Thorough Custom

Enabled Checks: basic

Heap Overflow/Underflow Detection

☑ Add guard pages to detect out of bounds heap access

Guard pages: 1 Add guard pages: After

Advanced

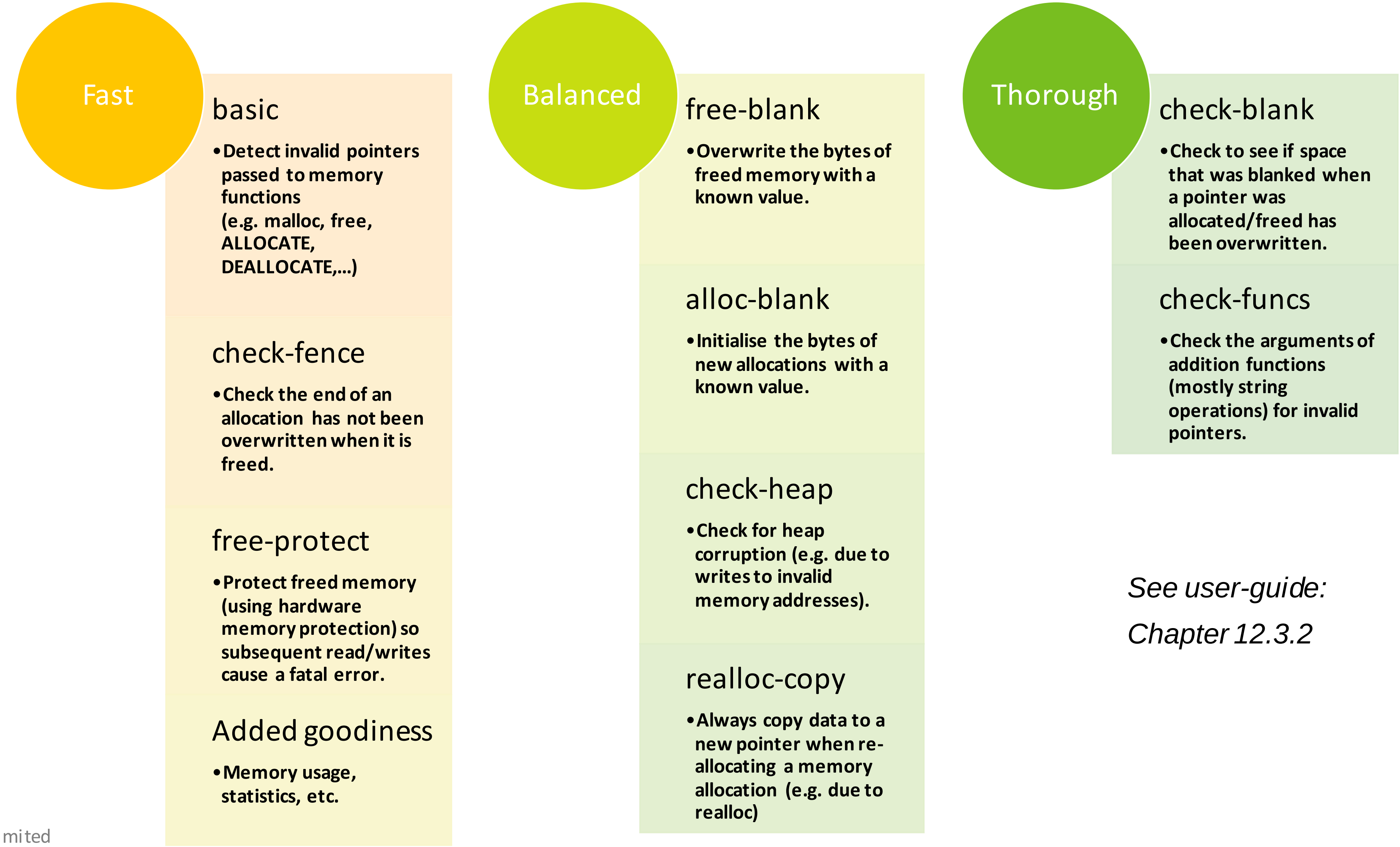
□ Check heap consistency every 100 heap operations

☑ Store stack backtraces for memory allocations

□ Only enable for these processes:

100%

Heap debugging options available



Fast

basic

- Detect invalid pointers passed to memory functions (e.g. malloc, free, ALLOCATE, DEALLOCATE,...)

check-fence

- Check the end of an allocation has not been overwritten when it is freed.

free-protect

- Protect freed memory (using hardware memory protection) so subsequent read/writes cause a fatal error.

Added goodiness

- Memory usage, statistics, etc.

Balanced

free-blank

- Overwrite the bytes of freed memory with a known value.

alloc-blank

- Initialise the bytes of new allocations with a known value.

check-heap

- Check for heap corruption (e.g. due to writes to invalid memory addresses).

realloc-copy

- Always copy data to a new pointer when re-allocating a memory allocation (e.g. due to realloc)

Thorough

check-blank

- Check to see if space that was blanked when a pointer was allocated/freed has been overwritten.

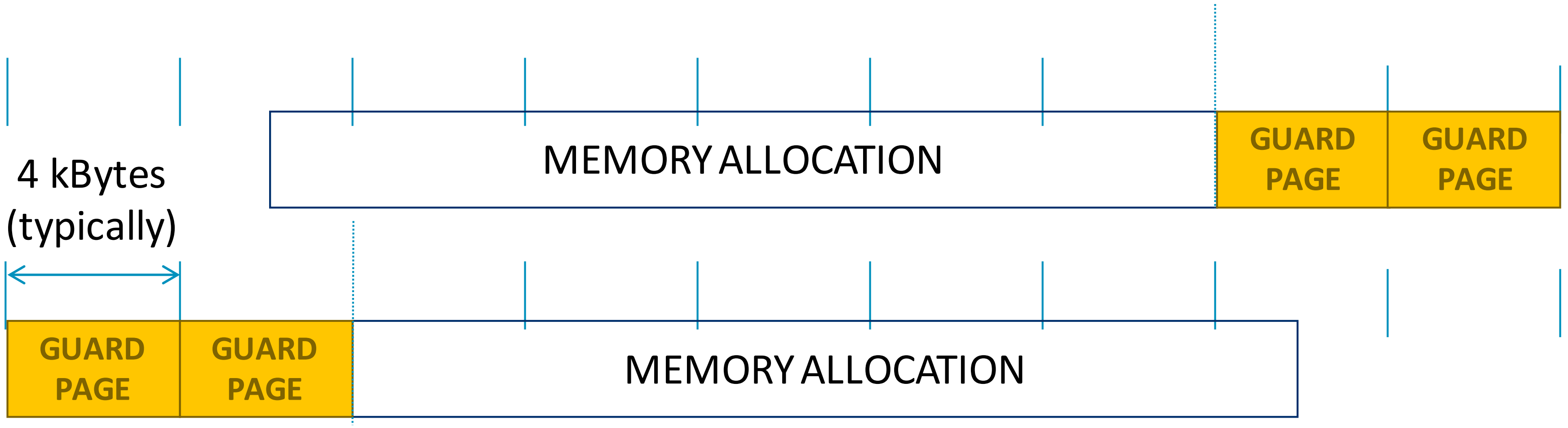
check-funcs

- Check the arguments of addition functions (mostly string operations) for invalid pointers.

See user-guide:

Chapter 12.3.2

Guard pages (aka "Electric Fences")



A powerful feature…:

- Forbids read/write on guard pages throughout the whole execution
(because it overrides C Standard Memory Management library)

… to be used carefully:

- Kernel limitation: up to 32k guard pages max ("mprotect fails" error)
- Beware the additional memory usage cost

Example 3

Go to

- $ cd 3_mem_dbg/

Compile with:

- $ make
- /!\ Don’t forget to compile with “-O0 -g”

Edit the job script to prefix the execution command with “ddt --connect”, launch ddt on the login node and submit the job

- $ ddt &
- $ sbatch job.sub

In the “Run” window, select “Fast” memory debugging first

Submit the job again and enable “Guard pages”

Detecting Deadlocks

My application has been running for a while now…

A strange behaviour where the application runs for “longer than expected” is a typical sign of a deadlock.

The application is hanging in the queue: alive and dead…

Allinea DDT is able to attach to the running processes and observe what is happening.

Example 4

Go to

- $ cd 4_deadlock/

Compile with:

- $ make
- Start Allinea DDT

Run the job with 10 processes: it works.

- $ srun --account=kg0166 --partition=compute -N 1 -n 10 ./cpi.exe

Run it again with 8 processes: it hangs!

- $ srun --account=kg0166 --partition=compute -N 1 -n 8 ./cpi.exe

In Allinea DDT’s GUI, select “Attach” from the main menu.

Allinea DDT should be able to detect the application automatically. Select it and debug it!

MPMD Applications

Example 5: How to run Forge on MPMD applications

Same logic, just prefix the execution command with the command:

- $ cd 5_mpmd/
- $ **ddt --connect** mpirun -n 8 myapp1.exe : -n 16 myapp2.exe
- $ **map --profile** srun –multi-prog cmd.srun
  - With cmd.srun:

    0-7   ./myapp1.exe

    8-23 ./myapp2.exe

Since 7.1, the ranks to profile can be specified:

- $ map **--select-ranks=0-7** --profile srun cmd.srun

Allinea DDT in manual launch

For complex launch mechanisms, for example if SLURM actually launches wrapper scripts, it is possible to launch the debugger in manual launch.

To do so:

- Launch the GUI on the login node and select “Manual Launch” from the Allinea DDT GUI
- Specify the number of processes and click on “Listen”
- The debugger now awaits for the processes to connect
  - Click on “Help” on the window to know how to connect the processes
    - By prefixing the processes to debug in the wrapper script with the following for example:
      - ddt-client --ddtsessionfile /home/flebeau/.allinea/session/toutatis-1 PROGRAM
  - Submit the job and see the processes attaching in the debugger

Increase Productivity with Automation

ESiWACE Project partner

Centre of Excellence in Simulation of Weather and Climate in Europe

A main goal of ESiWACE is to substantially improve efficiency and productivity of numerical weather and climate simulation on high-performance computing platforms by supporting the end-to-end workflow of global Earth system modelling in HPC environment.

esiwace
CENTRE OF EXCELLENCE IN SIMULATION OF WEATHER AND CLIMATE IN EUROPE

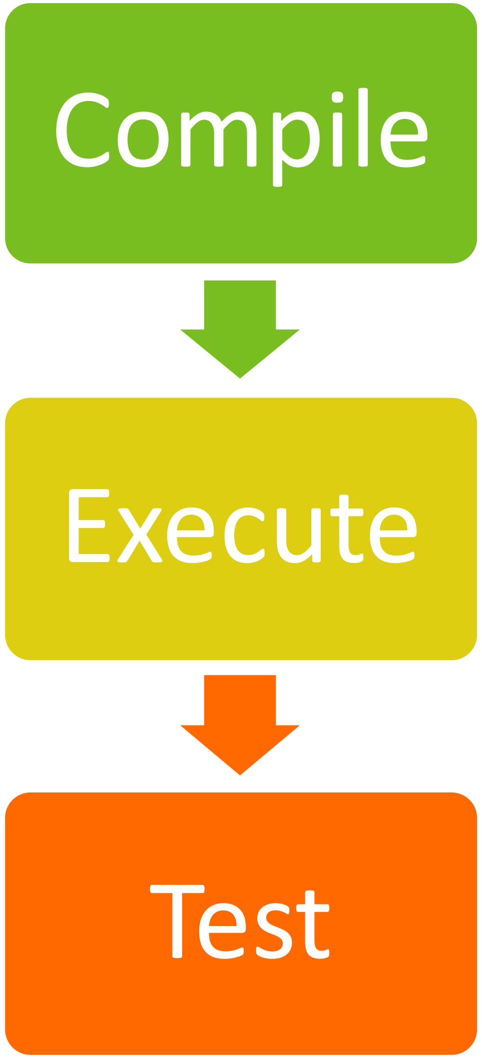
Automation script example

```
#!/bin/bash –l
# Job submission file name
jobfile=test_jacobi_mpi_omp_gnu.sub
# Load environment
module load compiler/gnu mpi/openmpi_gnu
module load allinea/perf-report
# Compile
make clean && make

# Job submission file configuration
cat << EOF > $jobfile
#!/bin/bash –l
#SBATCH --job-name=‘test_jacobi_mpi_omp_gnu‘
#SBATCH --time=00:05:00
#SBATCH --ntasks=128
#SBATCH –ntasks-per-node=2
export OMP_NUM_THREADS=16
srun ./jacobi_omp_mpi_gnu.exe
EOF

# Submit
sbatch $jobfile

# Check results
[…]
```


Compile

Execute

Test

Automate profiling

```
map --profile --output jacobi_omp_mpi_gnu_perf.map \
                      --stop-after=300
                      srun ./jacobi_omp_mpi_gnu.exe
map --export=jacobi_omp_gnu_perf.json jacobi_omp_gnu_perf.map
```

--output specifies the name of the output

- *.map file

--stop-after=X enables to stop sampling after X seconds after the program starts

--start-after=Y enables to start sampling after Y seconds after the program starts

--export=FILE exports a specified *.map file in JSON file

Automate debugging

```
ddt --offline -o jacobi_omp_mpi_gnu_debug.txt \
                        --trace-at _jacobi.F90:83,residual \
                        srun ./jacobi_omp_mpi_gnu.exe
```

--offline enable non-interactive debugging

-o specifies the name and output of the non-interactive debugging session

- Html
- Txt

Automate debugging

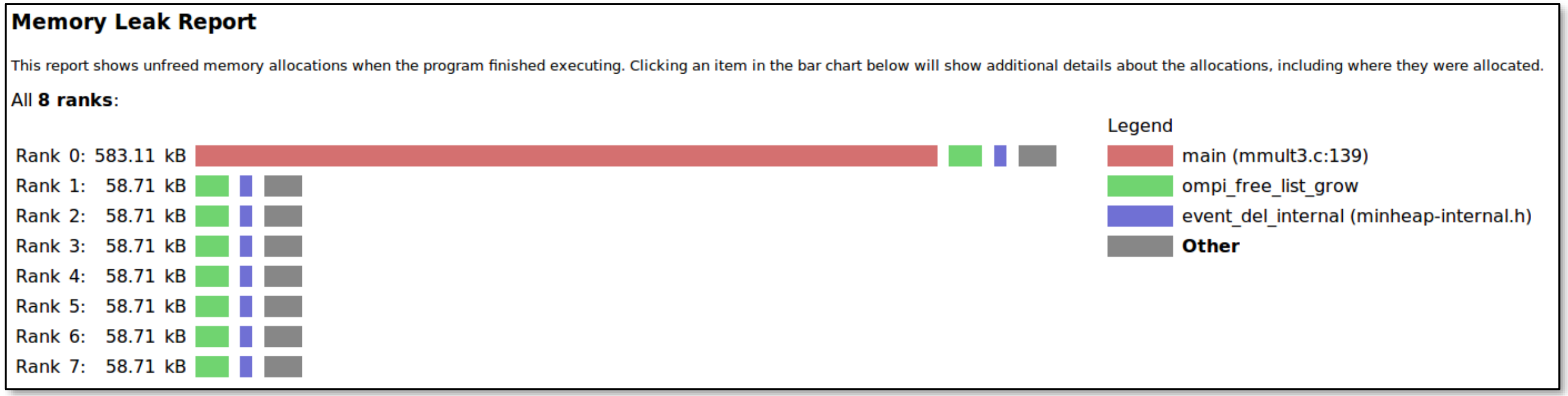
#	Time	Tracepoint	Processes	Values
1	21:18.172	jacobi_mpi_omp_gnu.exe (_jacobi.f90:83)	0-127	residual: 2.57

```
fail=0
# --- check DDT tracepoint (residual)
f=jacobi_omp_mpi_gnu_debug.txt
resid=`grep ^tracepoint $f | awk -Fresidual:  '{print $2}' |tail -1 |cut -c2-5`
if [ "$resid" != "2.57" ] ; then
         ((fail++))
        echo "Test has failed resid=$resid“
else
         echo “Test has succeeded”
```

Automate debugging

Other available options:

- --trace-changes: set a tracepoint on the variable introduced by the latest commit (git, svn, mercurial)
- --break-at: set a breakpoint
- --mem-debug: check for memory defects and leaks
- --check-bounds: check for out of bounds array accesses



Memory Leak Report

This report shows unfreed memory allocations when the program finished executing. Clicking an item in the bar chart below will show additional details about the allocations, including where they were allocated.

All **8 ranks**:

Rank 0: 583.11 kB

Rank 1: 58.71 kB

Rank 2: 58.71 kB

Rank 3: 58.71 kB

Rank 4: 58.71 kB

Rank 5: 58.71 kB

Rank 6: 58.71 kB

Rank 7: 58.71 kB

Legend

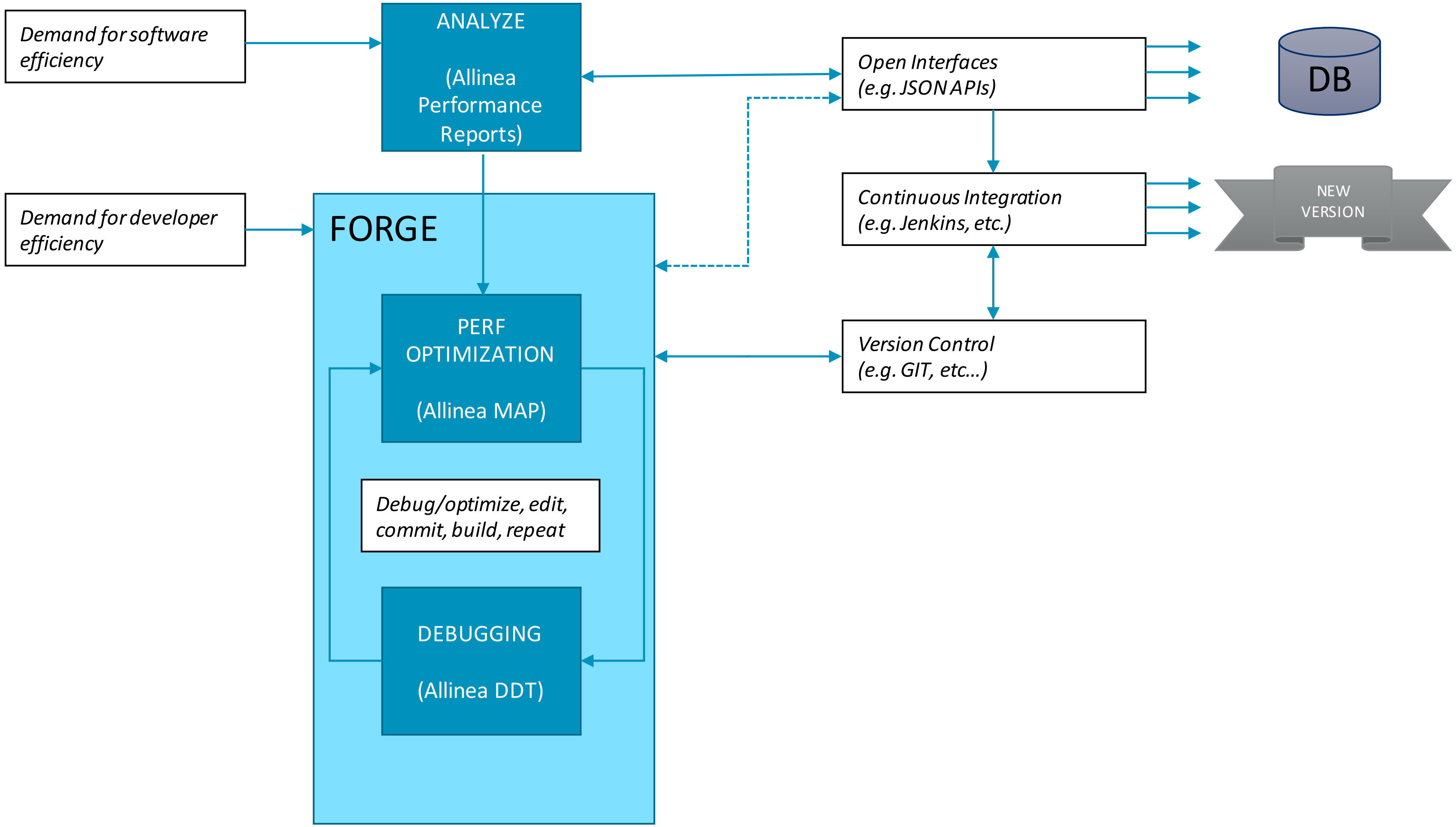
🟥 main (mmult3.c:139)

🟩 ompi_free_list_grow

🟦 event_del_internal (minheap-internal.h)

⬛ **Other**

Development process workflow


Demand for software efficiency
ANALYZE
(Allinea Performance Reports)
Open Interfaces (e.g. JSON APIs)
DB
Demand for developer efficiency
FORGE
Continuous Integration (e.g. Jenkins, etc.)
NEW VERSION
PERF OPTIMIZATION
(Allinea MAP)
Version Control (e.g. GIT, etc...)
Debug/optimize, edit, commit, build, repeat
DEBUGGING
(Allinea DDT)

Thank You!

Danke!

Merci!

谢谢!

ありがとう!

Gracias!

Kiitos!