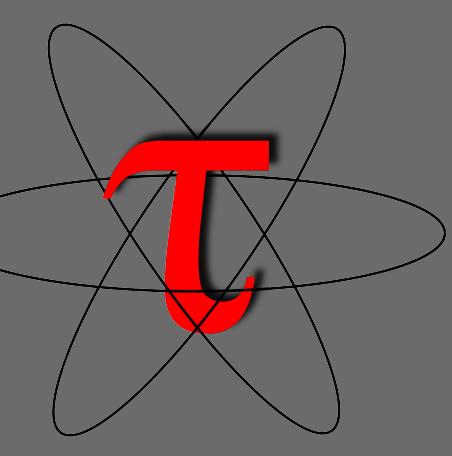


EFFICIENT PARALLEL RUNTIME BOUNDS CHECKING WITH THE TAU PERFORMANCE SYSTEM

John C. Linford, Sameer Shende, Allen D. Malony
ParaTools, Inc., Eugene, OR

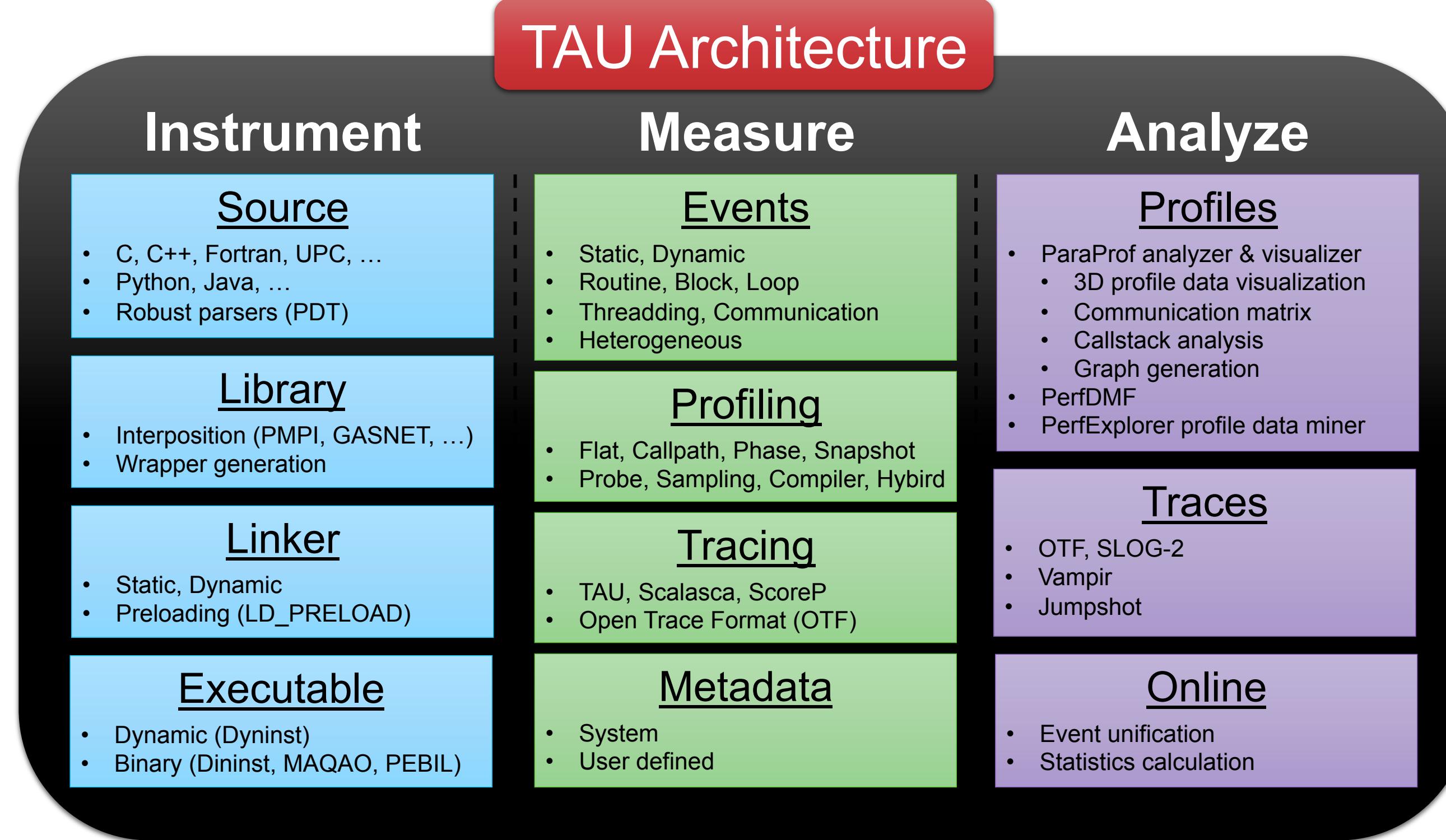


Andrew Wissink
U.S. Army Aviation Development Directorate - AFDD
NASA Ames Research Center, Moffett Field, CA

Problems Addressed

- Debug memory errors in parallel codes written in multiple languages.**
 - When and where do memory errors occur?
 - What is the heap memory usage?
 - Was the error cause by a read or a write?
 - Which processes or threads experienced the error?
 - Were there any memory leaks in the application?
 - What were the performance characteristics prior to fault?
- Protect sensitive data.**
 - Generate reports free of application data.
 - Open and portable debugging data file format.

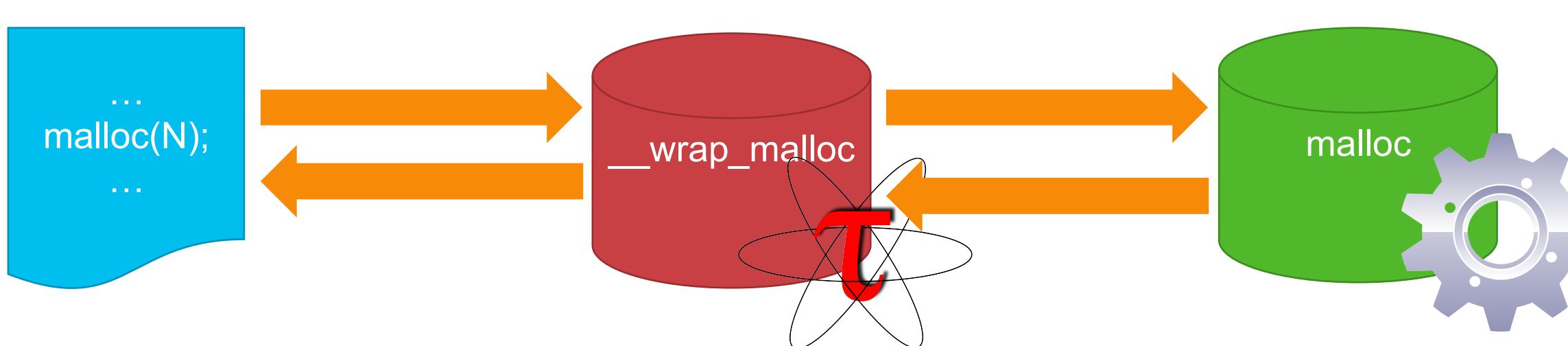
The TAU Performance System



Approach

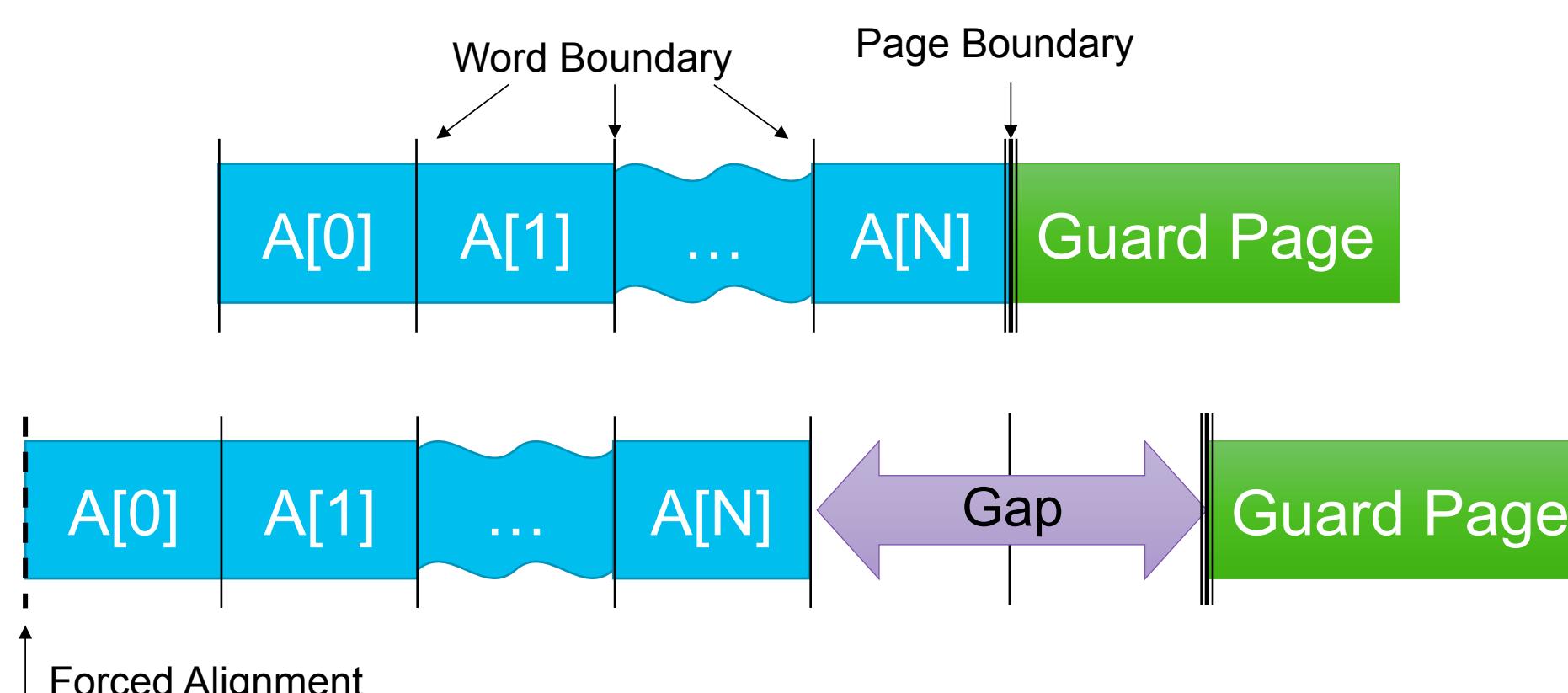
1. Instrument heap memory allocation and deallocation calls.

- Record heap memory use, allocation and deallocation rate.
- Associate every allocation with its use in the source code.



2. Allocate guard pages before and/or after memory allocations.

- Use the MMU hardware to protect regions of memory from access.
- Fast invalid memory access detection but high memory overhead.

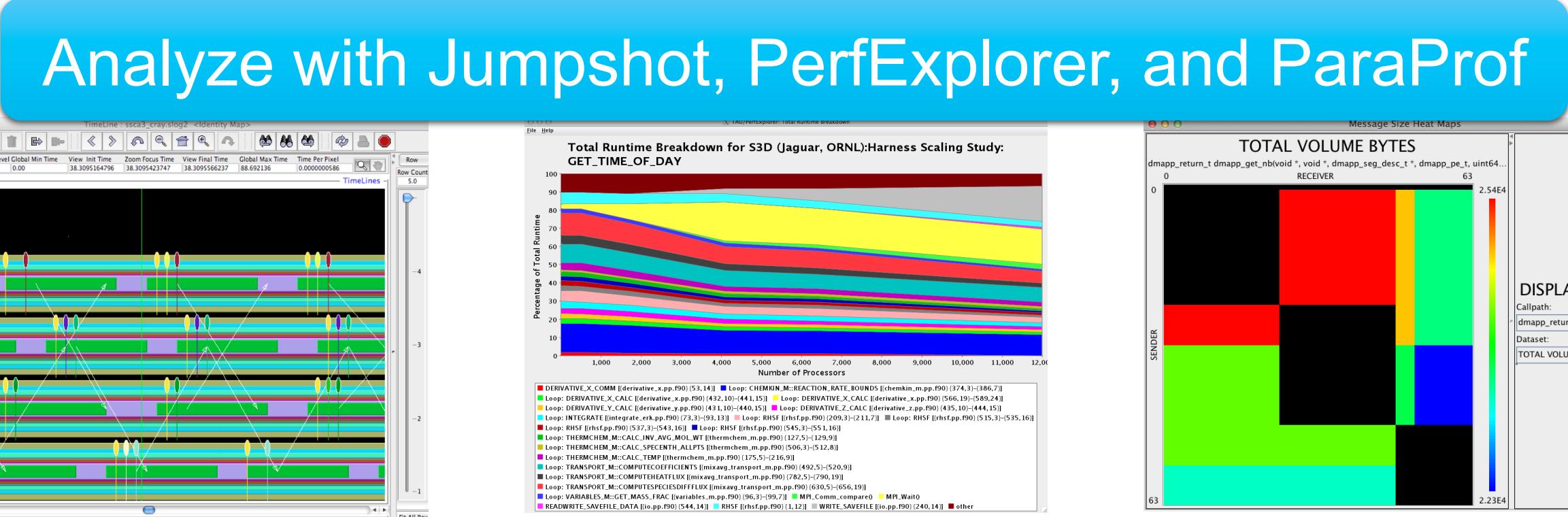
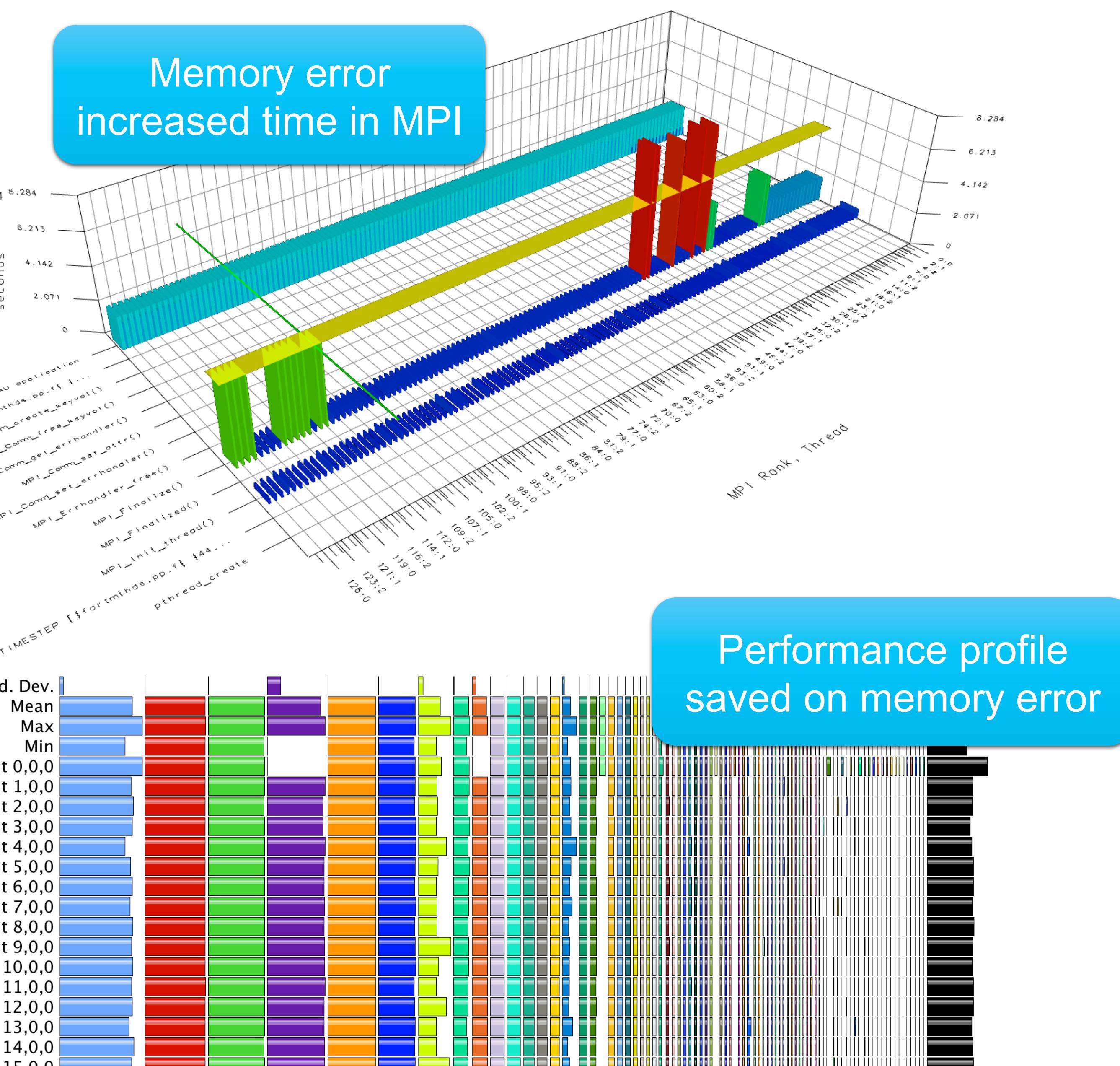


3. Intercept error signal if error occurs.

- Save backtrace in the application profile.
- Write application profile to disk and gracefully shut down.

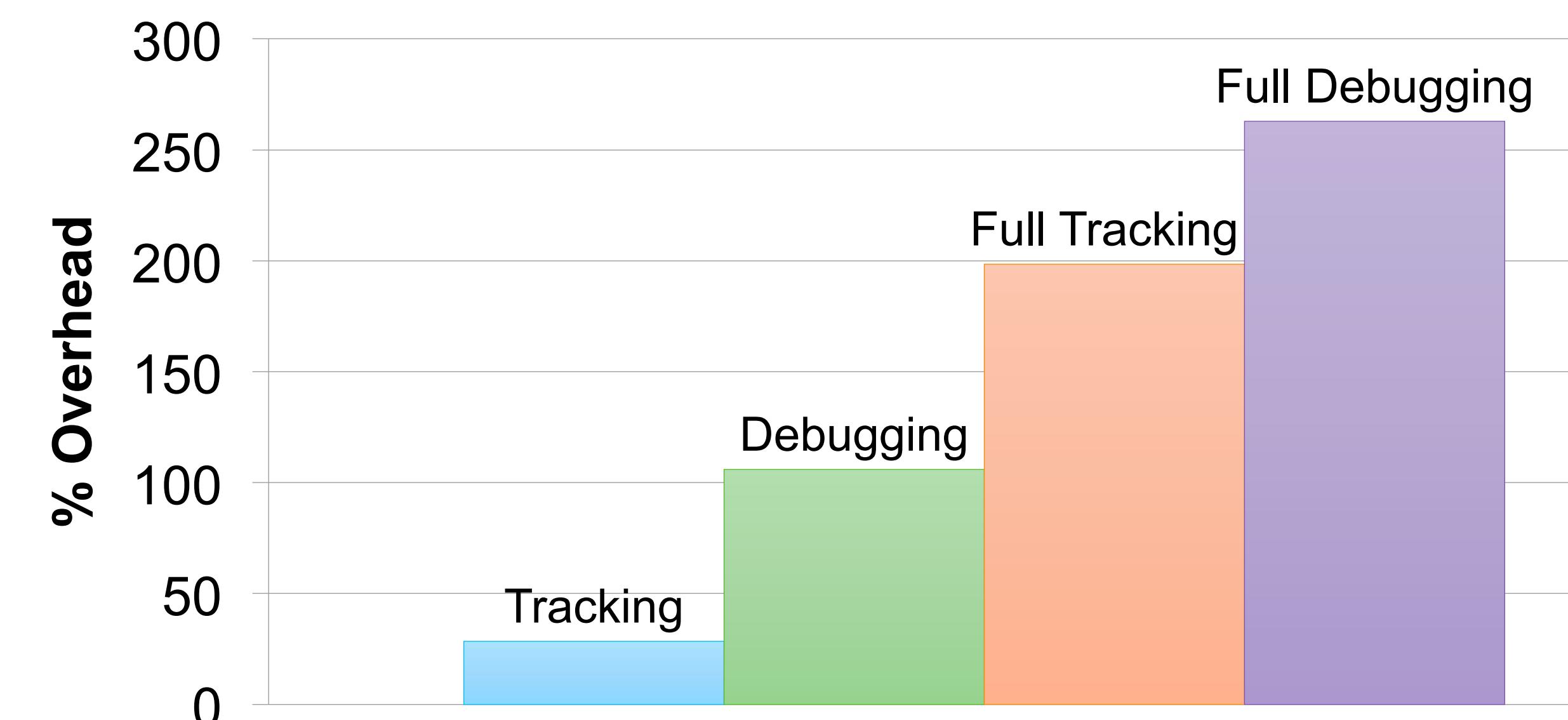
4. Send application profile to developers for analysis.

- Profile contains no application input or memory content information.



Results

CREATE-AV Test Code



TrialField	Name	Value
	callpath.ppk	
	Application ID	0
	Experiment ID	0
	Trial ID	0
BACKTRACE(1)	<code>_restore_rt() [sigaction.c:0]</code>	[lib64/libpthread-2.5.so]
BACKTRACE(2)	<code>__wrap_munmap() [/usr/lib64/libc-2.11.3.so]</code>	[libc-2.11.3.so]
BACKTRACE(3)	<code>__wrap_munmap(double) [/usr/lib64/libc-2.11.3.so]</code>	[libc-2.11.3.so]
BACKTRACE(4)	<code>__sanitizer_double_double() [/usr/lib64/libasan-4.3.1.so]</code>	[libasan-4.3.1.so]
BACKTRACE(5)	<code>__wrap_samarcStep() [/mnt/cfs/scratch/jlinfo/mpi4py-c+190-create/samint_c3881]</code>	[mfc-scratch/jlinfo/mpi4py-c+190-create/samint_c3881]
BACKTRACE(6)	<code>__wrap_samarcStep() [/mnt/cfs/scratch/jlinfo/mpi4py-c+190-create/samint_c3881]</code>	[mfc-scratch/jlinfo/mpi4py-c+190-create/samint_c3881]
BACKTRACE(7)	<code>fast_function() [/usr/HOME2/jlinfo/prodstrc-0.55/build/python-2.7.2/python/cEval-4013]</code>	[mfc-pkgs/PTOOLS/pkgs/prodstrc-0.55/linux-redhat5.6-gnu-x86_64/python-2.7.2]
BACKTRACE(8)	<code>PyEval_EvalFrameEx() [/usr/HOME2/jlinfo/prodstrc-0.55/build/python-2.7.2/python/cEval-4013]</code>	[mfc-pkgs/PTOOLS/pkgs/prodstrc-0.55/linux-redhat5.6-gnu-x86_64/python-2.7.2]
BACKTRACE(9)	<code>PyEval_EvalCodeEx() [/usr/HOME2/jlinfo/prodstrc-0.55/build/python-2.7.2/python/cEval-673]</code>	[mfc-pkgs/PTOOLS/pkgs/prodstrc-0.55/linux-redhat5.6-gnu-x86_64/python-2.7.2]
BACKTRACE(10)	<code>PyImport_ExecCodeModuleEx() [/usr/HOME2/jlinfo/prodstrc-0.55/build/python-2.7.2/python/import-682]</code>	[mfc-pkgs/PTOOLS/pkgs/prodstrc-0.55/linux-redhat5.6-gnu-x86_64/python-2.7.2]
BACKTRACE(11)	<code>PyImport_ImportModuleEx() [/usr/HOME2/jlinfo/prodstrc-0.55/build/python-2.7.2/python/import-1021]</code>	[mfc-pkgs/PTOOLS/pkgs/prodstrc-0.55/linux-redhat5.6-gnu-x86_64/python-2.7.2]
BACKTRACE(12)	<code>PyImport_ImportModuleEx() [/usr/HOME2/jlinfo/prodstrc-0.55/build/python-2.7.2/python/import-1021]</code>	[mfc-pkgs/PTOOLS/pkgs/prodstrc-0.55/linux-redhat5.6-gnu-x86_64/python-2.7.2]
BACKTRACE(13)	<code>load_pext() [/usr/HOME2/jlinfo/prodstrc-0.55/build/python-2.7.2/python/import-2319]</code>	[mfc-pkgs/PTOOLS/pkgs/prodstrc-0.55/linux-redhat5.6-gnu-x86_64/python-2.7.2]
BACKTRACE(14)	<code>import_mod() [/usr/HOME2/jlinfo/prodstrc-0.55/build/python-2.7.2/python/import-2319]</code>	[mfc-pkgs/PTOOLS/pkgs/prodstrc-0.55/linux-redhat5.6-gnu-x86_64/python-2.7.2]
BACKTRACE(15)	<code>builtin_import() [/usr/HOME2/jlinfo/prodstrc-0.55/build/python-2.7.2/python/builtinmodule-c49]</code>	[mfc-pkgs/PTOOLS/pkgs/prodstrc-0.55/linux-redhat5.6-gnu-x86_64/python-2.7.2]
BACKTRACE(16)	<code>PyObject_Call() [/usr/HOME2/jlinfo/prodstrc-0.55/build/python-2.7.2/python/cEval-3882]</code>	[mfc-pkgs/PTOOLS/pkgs/prodstrc-0.55/linux-redhat5.6-gnu-x86_64/python-2.7.2]
BACKTRACE(17)	<code>PyEval_CallObjectWithKeywords() [/usr/HOME2/jlinfo/prodstrc-0.55/build/python-2.7.2/python/cEval-3882]</code>	[mfc-pkgs/PTOOLS/pkgs/prodstrc-0.55/linux-redhat5.6-gnu-x86_64/python-2.7.2]

Backtrace across several languages

runStep [[samarc.py:[104]]]	
Heap Allocate	88
Heap Memory Used (KB) at Entry	54,053.779
Heap Memory Used (KB) at Exit	54,062.459
▼ samarcStep	
Heap Used (KB) at Entry	54,053.865
Heap Memory Used (KB) at Exit	54,062.459
▼ __wrap_samarcStep() [/mnt/cfs/scratch/jlinfo/mpi4py-c]	
Heap Used (KB) at Entry	54,053.865
Heap Memory Used (KB) at Exit	54,062.459
▶ SWIG_AsVal_double, double() [/mnt/cfs/scr...	
int samarcStep(double, double) {	
Heap Used (KB) at Entry	54,053.865
Heap Memory Used (KB) at Exit	54,062.459
} void SAMINT::timestamp(double, double) {SAMINT.i	
Heap Allocate	8,800
Heap Memory Used (KB) at Entry	54,053.865
Heap Memory Used (KB) at Exit	54,062.459
MEMORY LEAK! Heap Allocate	17,256,763
Invalid memory access	9,327
}	

View memory events and statistics

236835	Heap Memory Used (KB)
131283	Heap Allocate
93961	Heap Free
53148	Heap Memory Used (KB) at Entry
53148	Heap Memory Used (KB) at Exit
37322	MEMORY LEAK! Heap Allocate : TAU application
19368	Heap Allocate : TAU application => OurMain()
17712	Heap Allocate : TAU application => OurMain()
15576	Heap Free : TAU application => OurMain() =>
6112	Heap Allocate : TAU application => OurMain()

Related Work

- H. Aygün, D.U.M.A – Detect Unintended Memory Access, <http://duma.sourceforge.net/>, May 2013.
- Apple Corp., “Guard Malloc Manual Page,” <https://developer.apple.com/>, March 2009.
- B. Perens, “efence – Electric Fence Malloc Debugger,” <http://linux.die.net/man/3/efence>, 1999.
- J. Seward and N. Nethercote, “Using Valgrind to detect unintended value errors with bit-precision,” in USENIX ATC, Anaheim, CA, USA, April 2005.
- K. Serebryany, D. Bruening, A. Potapenko, and D. Vyukov, “Address-sanitizer: A fast address sanity checker,” in USENIX ATC, 2012.
- Microsoft Corp., “GFlags and PageHeap,” <http://msdn.microsoft.com/>, July 2013.
- S. Shende, A. D. Malony, J. Linford, A. Wissink, and S. Adamec, “Isolating Runtime Faults with Callstack Debugging using TAU,” in Proceedings of the IEEE HPEC 2012 Conference. IEEE Computer Society, 2012.