Lessons learned from Implementing OMPD

A Debugging Interface for OpenMP

Joachim Protze
Matthias S. Müller
RWTH Aachen University

Ignacio Laguna
Dong H. Ahn
Martin Schulz
LLNL

John DelSignore
Ariel Burton
Rogue Wave Software

Part of this work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344. Lawrence Livermore National Security, LLC

Debugging OpenMP: gdb backtrace

```
(gdb) thread 3
[Switching to thread 3 (Thread 0x7ffff74ee780 (LWP 18484))]#0
tdm::L__ZN3tdm8residuumERPd_67__par_loop0_2_10 () at tdm.C:70
70                      sum=fabs(x[i-1]*e[i]+x[i]*d[i]+x[i+1]*f[i]-b[i]);
(gdb) bt
#0  tdm::L__ZN3tdm8residuumERPd_67__par_loop0_2_10 () at tdm.C:70
#1  0x00007ffff7d2abc3 in __kmp_invoke_microtask () from
/opt/intel/Compiler/16.0/0.109/rwthlnk/compiler/lib/intel64_lin/libiomp5.so
#2  0x00007ffff7cfc8c7 in __kmp_invoke_task_func (gtid=5) at
../../src/kmp_runtime.c:7585
#3  0x00007ffff7cfbfd5 in __kmp_launch_thread (this_thr=0x5) at
../../src/kmp_runtime.c:6037
#4  0x00007ffff7d2aee3 in __kmp_launch_worker (thr=0x5) at ../../src/z_Linux_util.c:786
#5  0x0000003f890079d1 in start_thread (arg=0x7ffff74ee780) at pthread_create.c:301
#6  0x0000003f888e88fd in clone () at ../sysdeps/unix/sysv/linux/x86_64/clone.S:115
(gdb)
```

Debugging OpenMP: gdb backtrace (master)

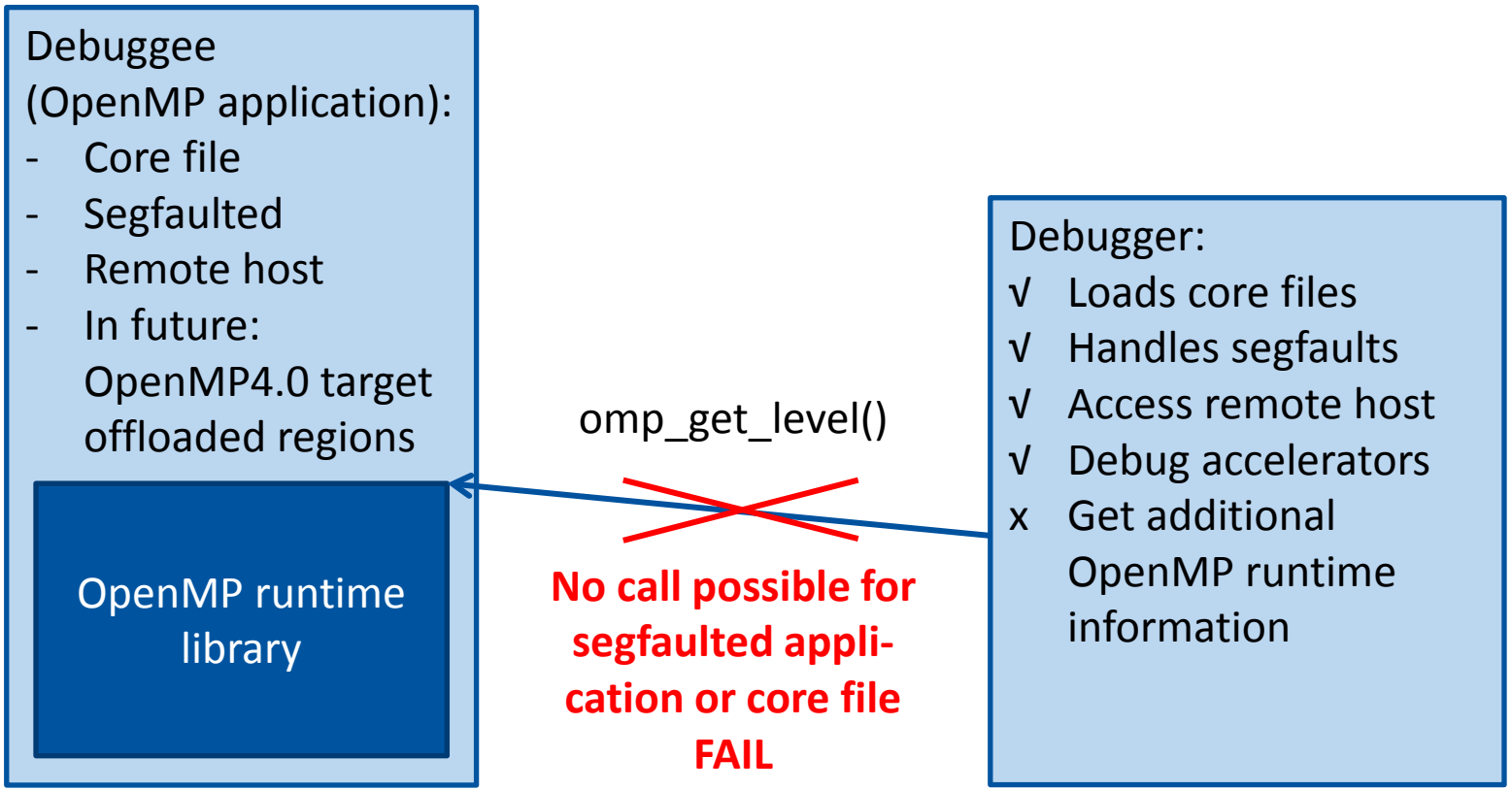
```
(gdb) thread 1
[Switching to thread 1 (Thread 0x7ffff76f1720 (LWP 18478))]#0
tdm::L__ZN3tdm8residuumERPd_67__par_loop0_2_10 () at tdm.C:70
70                      sum=fabs(x[i-1]*e[i]+x[i]*d[i]+x[i+1]*f[i]-b[i]);
(gdb) bt
#0  tdm::L__ZN3tdm8residuumERPd_67__par_loop0_2_10 () at tdm.C:70
#1  0x00007ffff7d2abc3 in __kmp_invoke_microtask () from
/opt/intel/Compiler/16.0/0.109/rwthlnk/compiler/lib/intel64_lin/libiomp5.so
#2  0x00007ffff7cfc8c7 in __kmp_invoke_task_func (gtid=1) at
../../src/kmp_runtime.c:7585
#3  0x00007ffff7cfdd0c in __kmp_fork_call(ident_t *, int, enum fork_context_e,
kmp_int32, microtask_t, launch_t, va_list *) (loc=0x1, gtid=0, call_context=4, argc=0,
    microtask=0x7ffff7618200, invoker=0x7fffffff7adc, ap=0x7fffffff7f80) at
../../src/kmp_runtime.c:2559
#4  0x00007ffff7cd26c8 in __kmpc_fork_call (loc=0x1, argc=0, microtask=0x4) at
../../src/kmp_csupport.c:325
#5  0x0000000000404094 in tdm::residuum (this=0x7fffffff8140, x=@0x7fffffff81a0) at
tdm.C:67
#6  0x0000000000402c48 in main (argc=1, argv=0x7fffffff8368) at solve.C:33
(gdb)
```

Overview

- **OMPD Interface**
- **Use cases:**
  - → OpenMP aware backtrace
  - → Stepping OpenMP application
- **Resolve target values**

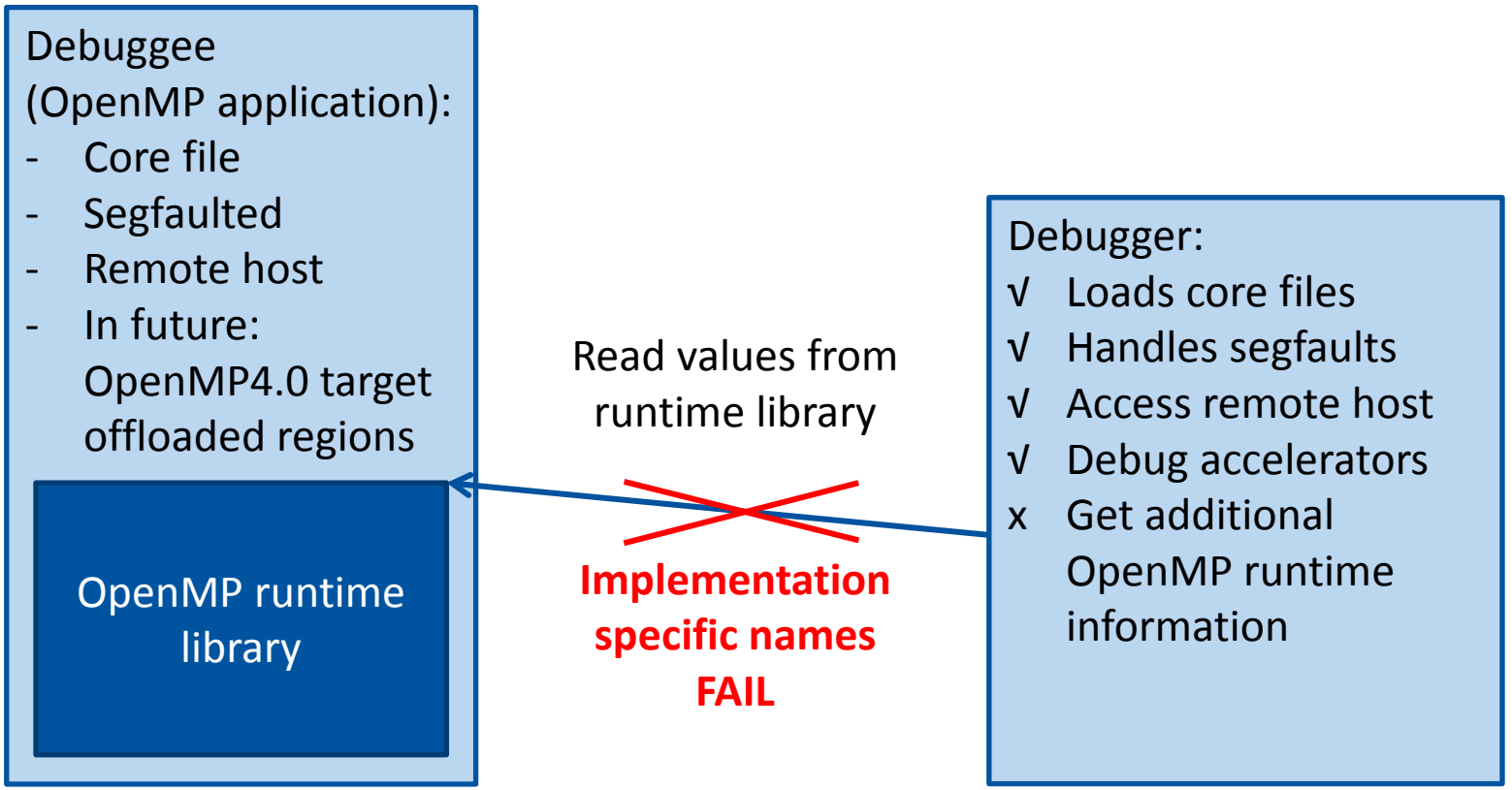
What is OMPD? Why do we need OMPD?

- **OMPD: OpenMP Debugging Interface**



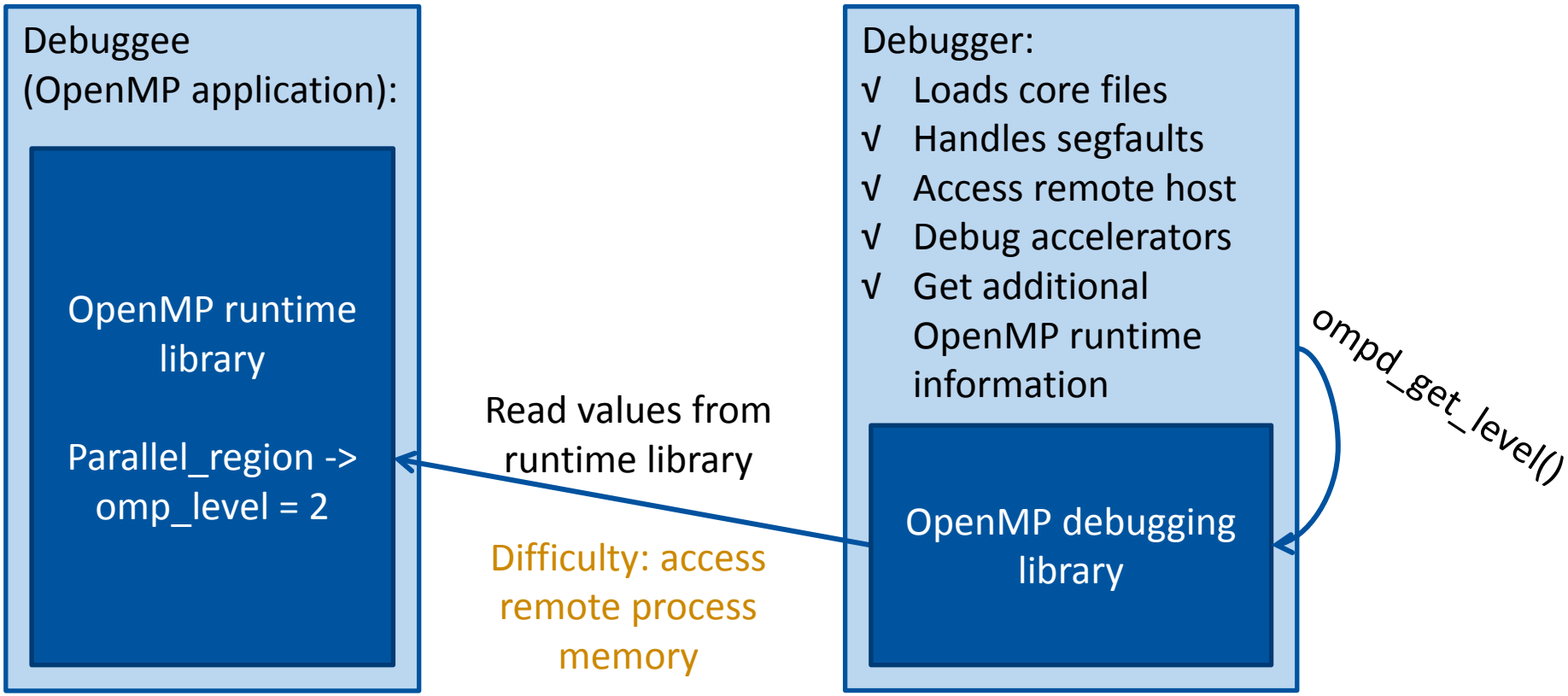
What is OMPD? Why do we need OMPD?

- **OMPD: OpenMP Debugging interface**



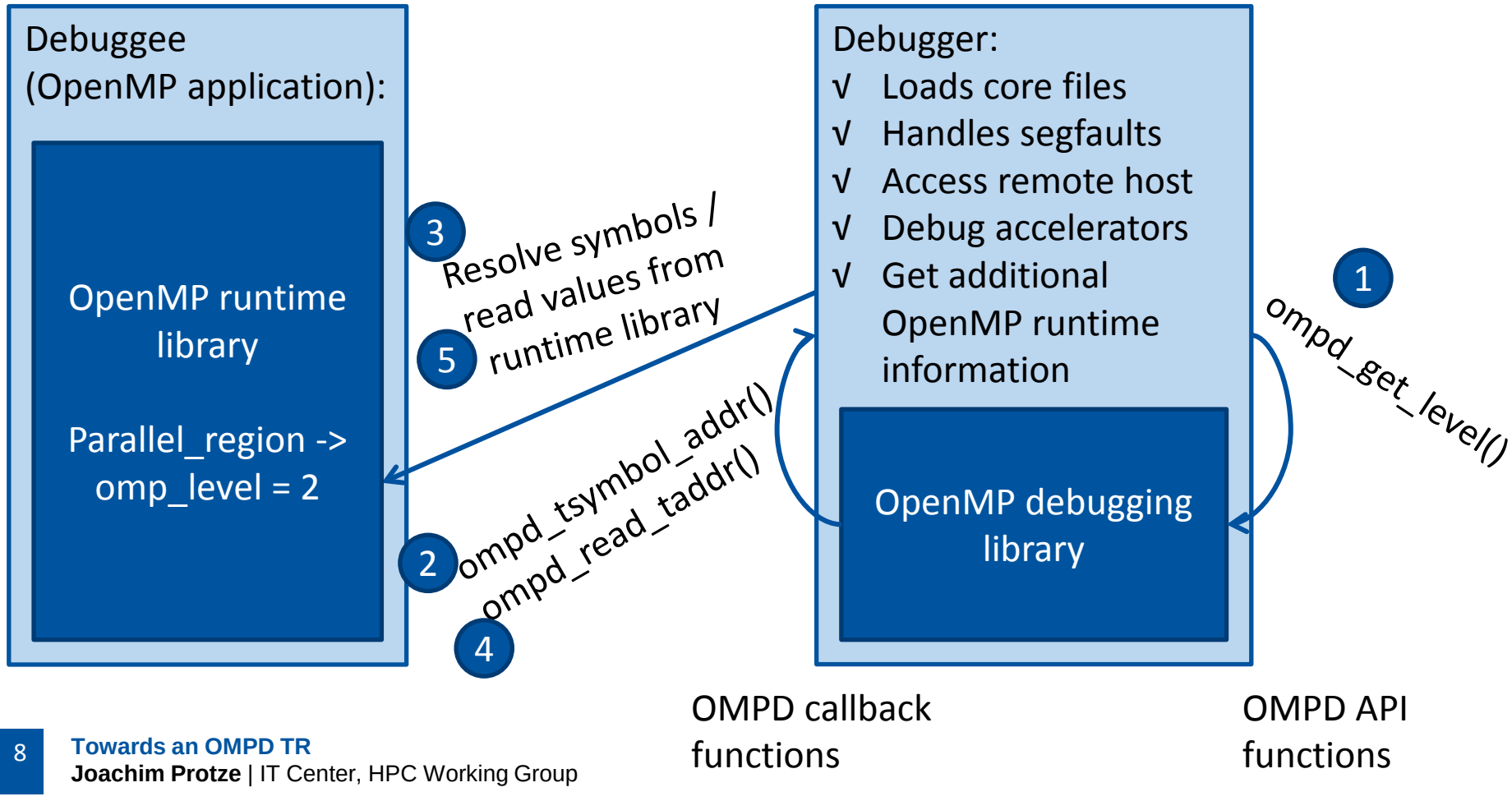
What is OMPD? Why do we need OMPD?

- **OMPD: OpenMP Debugging interface**



What is OMPD? Why do we need OMPD?

- **OMPD: OpenMP Debugging interface**



The two interfaces of OMPD

- **OMPD callback interface**

  Functions implemented by the debugger / debugging tool:

  → Resolve target symbols

  → Read/ write target memory

  → Alloc/free local memory

  → Output

  → Endianess conversion

  → ~~*Type size/offset lookup*~~

- **OMPD API**

  Functions called by the debugger / debugging tool

  → Function analogues for OpenMP / OMPT API functions.

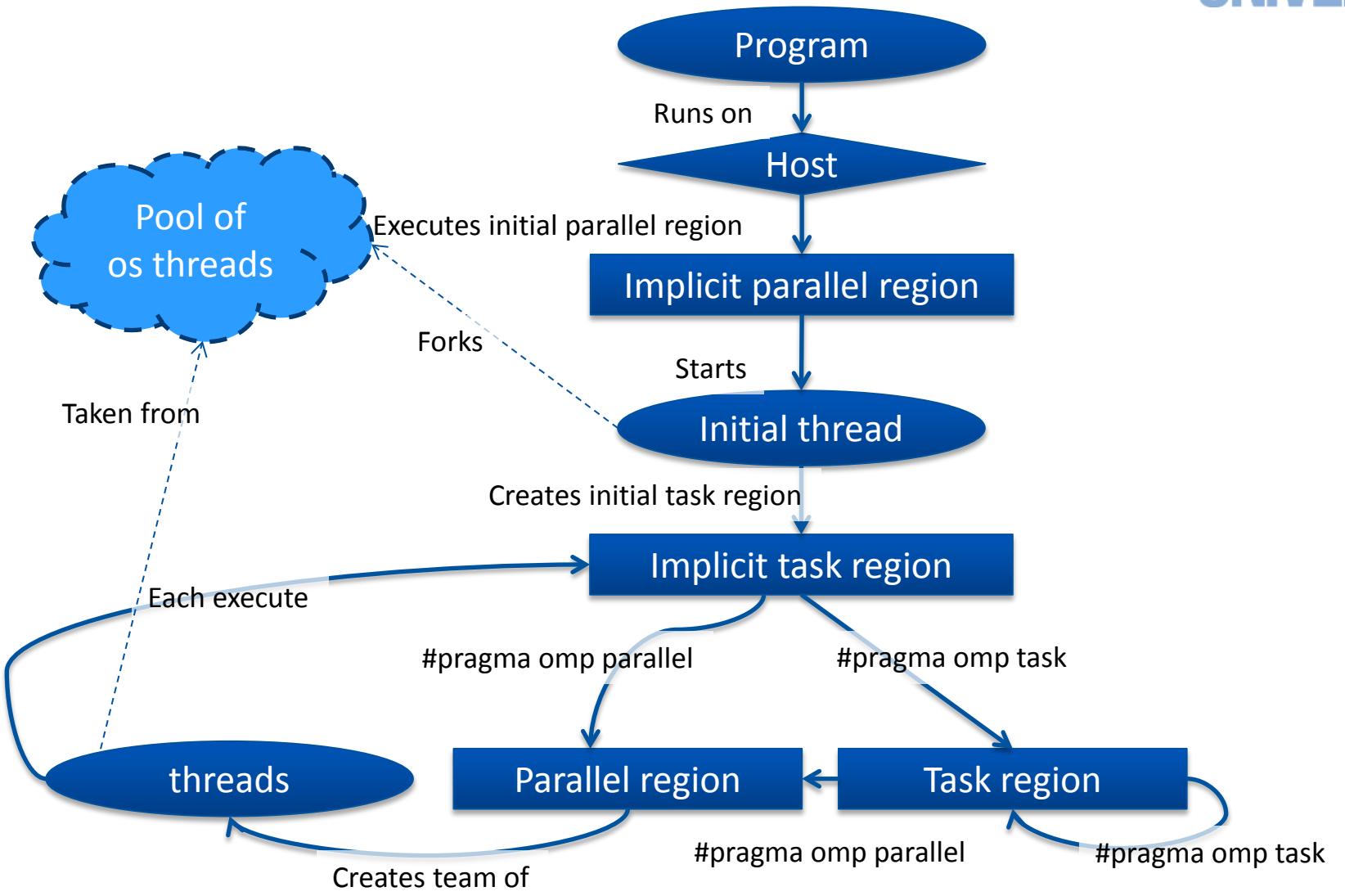
  → Functions to navigate to the scope of interest (unwinding the stack of scopes)

Scoping of API functions

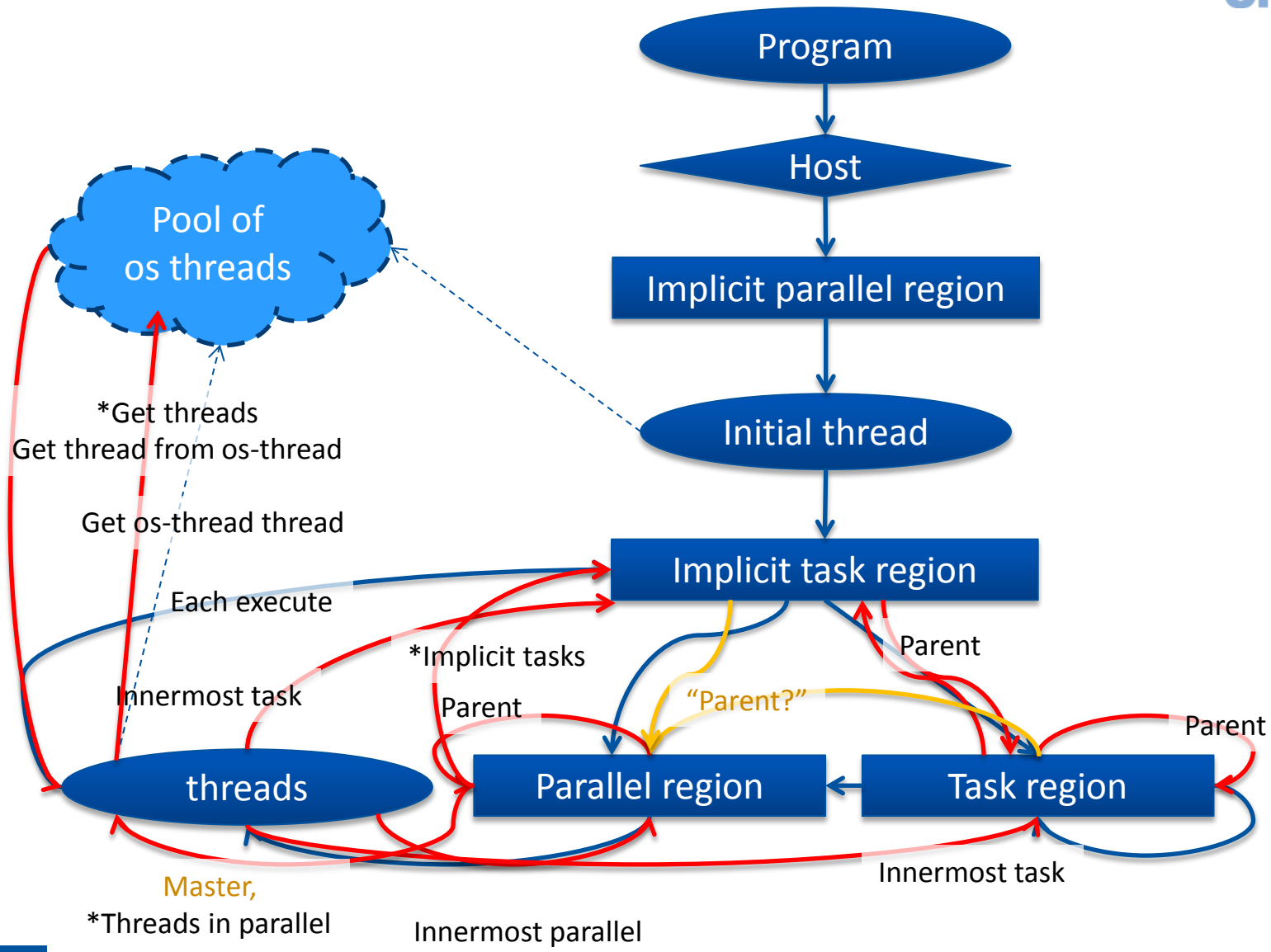
- **For OpenMP API calls and OMPT functions, the scope of interest is given by the scope of execution.**
  - → Current device, thread, parallel region, task region
  - → *`level`* argument to address ancestor scopes.

- **For OMPD there is no natural scope**
  - → We use scope handles to address a specific scope.
  - → Need for functions to navigate to the scope of interest
  - → Reduce overhead for successive queries on a single scope

Execution model of OpenMP (3.1)



Navigating between scopes in OMPD



Labels represent API function calls to navigate to scopes, labels with * return a list

Use Cases

OpenMP aware Backtrace

Stepping OpenMP Application

Use case: OpenMP Backtrace

```
1   {
2     // code before parallel region
3     #pragma omp parallel
4     {
5       // parallel region code
6     }
7     // code after parallel region
8   }
```

Backtrace I want to get:

```
in #omp parallel from file:5 @ T3
in block () from file:3 @ T1
```

```
1   void parallel_region_block()
2   {
3     // parallel region code
4   }
5   [...]
6   {
7     // code before parallel region
8     omprt_run_parallel(parallel_region_block);
9     // code after parallel region
10  }
```

Backtrace you get:

```
in parallel_region_block () from file:5
in omprt_internal () from libopenmp
in omprt_run_parallel () from libopenmp
in block () from file:8
```

Use case: OpenMP Backtrace

```
1   {
2     // code before parallel region
3     #pragma omp parallel
4     {
5       // parallel region code
6     }
7     // code after parallel region
8   }
```

Backtrace I want to get:

```
in #omp parallel from file:5 @ T3
in block () from file:3 @ T1
```

Get information on the parallel region from the handle returned by get_top_parallel_region

Get information about the calling thread with get_master_thread

Parallel region handle provides:
- Frame information
- Other threads in this region

Possibility to continue backtrace on this thread with provided frame information

Use case: OpenMP Single-Stepping

- **Use case: single stepping into a parallel region**
- **Single step should go from line 2 to line 5**
- **Debugger might single step though OpenMP runtime**

```
1   {
2     // code before parallel region
3     #pragma omp parallel
4     {
5       // parallel region code
6     }
7     // code after parallel region
8   }
```

- **Better: Debugger gets information about line 5 from OMPD**
  - → Breakpoint at ompd_break_pre_parallel()
  - → Query parallel_function from OMPD
  - → Set breakpoint at parallel_function

Specification of Breakpoints

- **Provide an address to break at:**
  - → Address differs from process to process: not scalable for hybrid MPI+OpenMP
- **Provide a function name to break at:**
  - → Need to query for the function name
- **Our proposal:**
  - → OMPD requires to call an empty dummy function with specified name:
    - →void ompd_break_pre_parallel() {}
    - →void ompd_break_post_parallel() {}
    - →void ompd_break_pre_task() {}
    - →void ompd_break_post_task() {}

Resolve target symbols

Resolve Structures for Target Architecture

- **Example: omp_get_num_threads():**
  - → `return __kmp_entry_thread() -> th.th_team -> t.t_nproc;`
  - → Equivalent to `__kmp_threads[__kmp_gtid] -> th.th_team -> t.t_nproc;`

Value in thread local storage (TLS) of the local thread.

- **Read `__kmp_gtid` from TLS**
  - → get_thread_context_for_osthread(tcontext, …)
  - → tsymbol_addr(tcontext, “__kmp_gtid”, address …)
  - → read_tmemory(tcontext, address, …)

Resolve Structures for Target Architecture

- **Example: omp_get_num_threads():**
  - → `return __kmp_entry_thread() -> th.th_team -> t.t_nproc;`
  - → Equivalent to `__kmp_threads[__kmp_gtid] -> th.th_team -> t.t_nproc;`

Access array element

- **Access array element in `__kmp_threads`**
  - → Size of `__kmp_threads` needed
  - → Runtime provides size information:

    const uint64_t ompd_sizeof__##t = sizeof(t);

Resolve Structures for Target Architecture

RWTH AACHEN UNIVERSITY

- **Example: omp_get_num_threads():**
  - → `return __kmp_entry_thread() -> th.th_team -> t.t_nproc;`
  - → Equivalent to `__kmp_threads[__kmp_gtid] -> th.th_team -> t.t_nproc;`

- **Access structure element `th_team` in `th`**
  - → Need offset of `th_team` in `th`
  - → Runtime provides offset information:

```
const uint64_t ompd_access__##t##__##m = (uint64_t)&(((t*)0)->m);
```

Status of the work

- **OMPD prototype (for OpenMP 3.1) almost finished**
- **Based on and written for Intel OpenMP runtime**
  - → Runtime code: 5 files changed, 65 insertions(+), 7 deletions(-)
  - → Total: 12 files changed, 308 insertions(+), 9 deletions(-)
- **Currently in process of releasing it from LLNL internal**

Next steps

- **OpenMP for accelerators**
- **The current model of OMPD does not care for devices or target construct**

- **We already changed the semantics of context to be prepared for devices**

Thanks for your attention!