OpenMP Tools API (OMPT) and HPCToolkit

John Mellor-Crummey

Department of Computer Science
Rice University

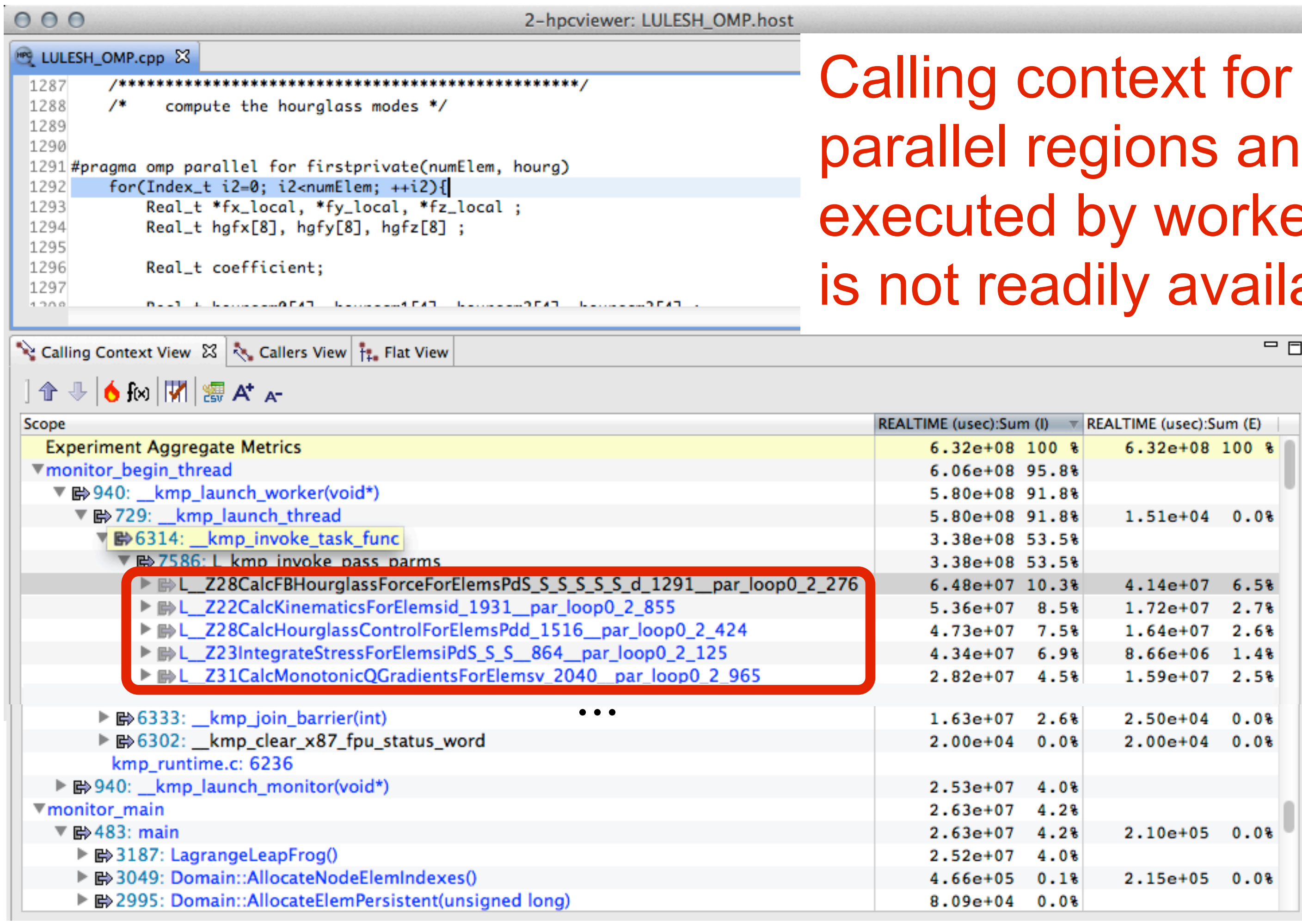
johnmc@rice.edu

OpenMP Tools Subcommittee

- **Executive lead**
  - Martin Schulz - LLNL
- **Technical leads**
  - Alexandre Eichenberger - IBM
  - John Mellor-Crummey - Rice
- **Active subcommittee members**
  - Nawal Copty - Oracle
  - Jim Cownie - Intel
  - Robert Dietrich - TU Dresden
  - Christian Iwainsky - TU Darmstadt
  - Xu Liu - Rice
  - Eugene Loh - Oracle
  - Daniel Lorenz - Juelich
  - Harald Servat - Barcelona Supercomputer Center

Motivation

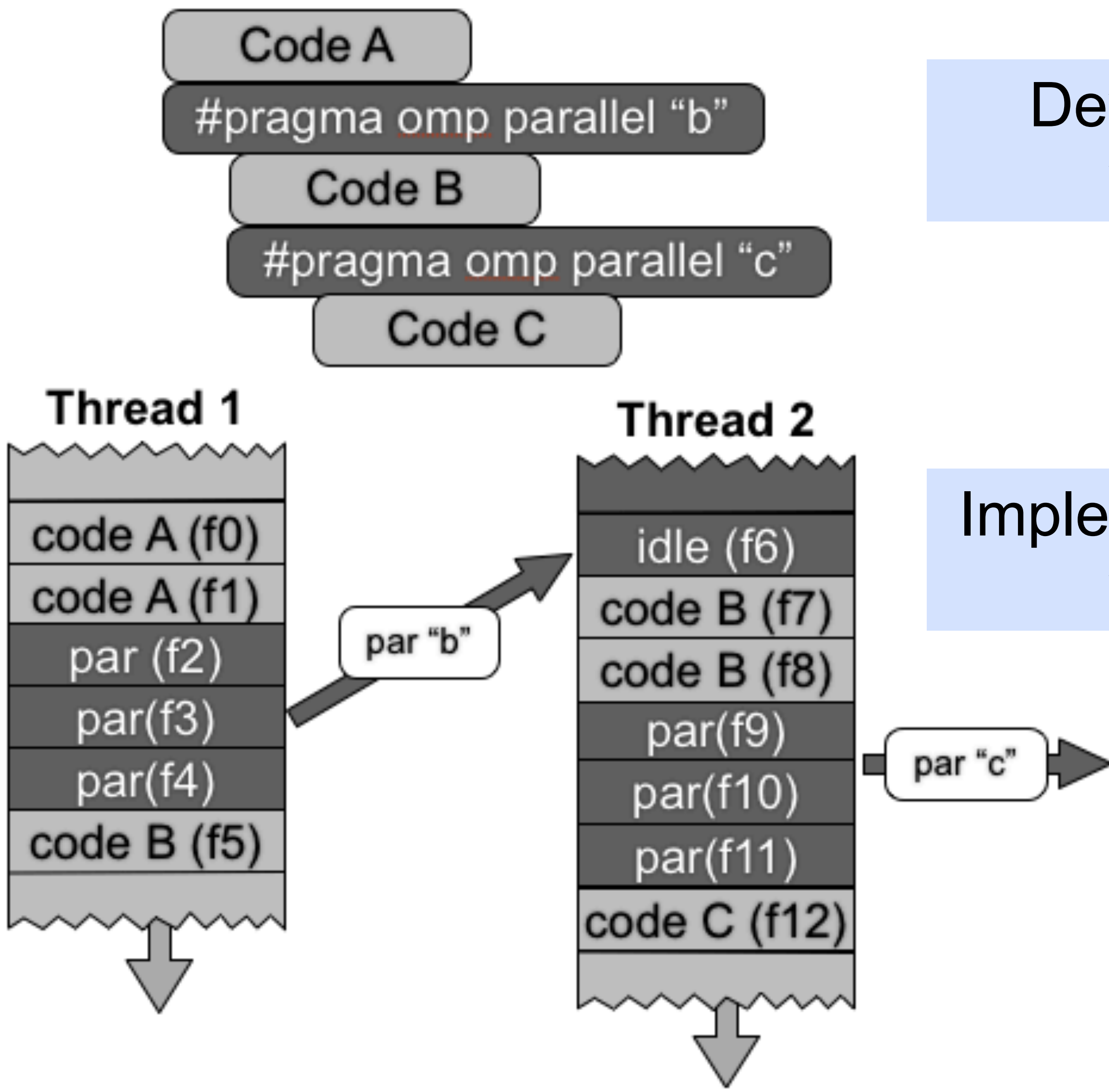
- **Large gap between OpenMP source and implementation**



Calling context for code in parallel regions and tasks executed by worker threads is not readily available

- **Tools must bridge this gap to explain program performance**

Calling Context is Distributed across Threads



Developer view

Implementation view

Obstacles for Runtime-independent Tools

- No standard API for OpenMP tools
- Principal prior efforts
  - POMP - Mohr, Malony, Shende, Wolf
  - Collector API - Itzkowitz, Mazurov, Copty, Lin
- Differences in OpenMP implementations
  - support for static linking
  - Intel: extra “monitor” thread
  - PGI: cactus stack
  - IBM: neither

Outline

- OMPT - emerging performance tool API for OpenMP
  - —overview and goals
  - —state tracking
  - —event notification
  - —API
- Status and next steps

OMPT Design Objectives

- **Enable tools to gather information and associate costs with application source and runtime system**
  - —construct low-overhead tools based on asynchronous sampling
  - —identify application stack frames vs. runtime frames
  - —associate a thread's activity at any point with a descriptive state
    - – parallel work, idle, lock wait, ...
- **Negligible overhead if OMPT interface is not in use**
- **Define support for trace-based performance tools**
- **Don't impose an unreasonable development burden**
  - —runtime implementers
  - —tool developers

OMPT Performance Tools API

Overview and Goals

- **Focus on minimal set of functionality**
  - provide essential support for sampling-based tools
  - only support tools attached at link-time or program launch
- **Minimize runtime cost**
  - reduce cost in runtime and tool where possible
    - encourage integration into optimized runtimes
  - make support for higher-overhead features optional
    - callbacks for blame shifting
    - callbacks for full-featured tracing tools

Major OMPT Functionality

- **State tracking**
  - —have runtime track keep track of thread states (e.g., idle, parallel)
  - —allow tools to query this state at any time (async signal safe)
  - —provide (limited) persistent storage for tool data in runtime system
- **Call stack interpretation**
  - —provide hooks that enable tools to reconstruct application-level call stacks from implementation-level information
- **Event notification**
  - —provide callbacks for predefined events
    - few mandatory notifications
    - many optional ones

Runtime State Tracking

- OpenMP runtime keeps track of its own state
  - —predefined states on next slide
- Query routine
  - —ompt_state_t ompt_get_state(ompt_wait_id_t *wait_id)
  - —async signal safe
- Wait IDs
  - —only returned for states that signify waiting
  - —identifies the cause for waiting (lock, critical region, ...)

OMPT Runtime States

- Work
  - —serial, parallel
  - —reduction
- Idle
- Barrier
  - —implicit
  - —explicit
  - —either
- Task wait
  - —task wait
  - —task group wait
- Mutual Exclusion
  - —wait lock
  - —wait nest lock
  - —wait critical
  - —wait atomic
  - —wait ordered
- Overhead
- Miscellaneous
  - —first
  - —undefined

OMPT Event Notifications

- **Mandatory events**
- **Blame-shifting events (optional)**
- **Trace events (optional)**

Mandatory Events

Essential support for any performance tool

- **Threads**
- **Parallel regions**
- **Tasks**

create/exit event pairs

- **Runtime shutdown**
- **User-level control API**
  - **—e.g., support tool start/stop**

singleton events

Blame-shifting Events (Optional)

Support designed for sampling-based performance tools

- **Idle**
- **Wait**
  - barrier
  - taskwait
  - taskgroup wait

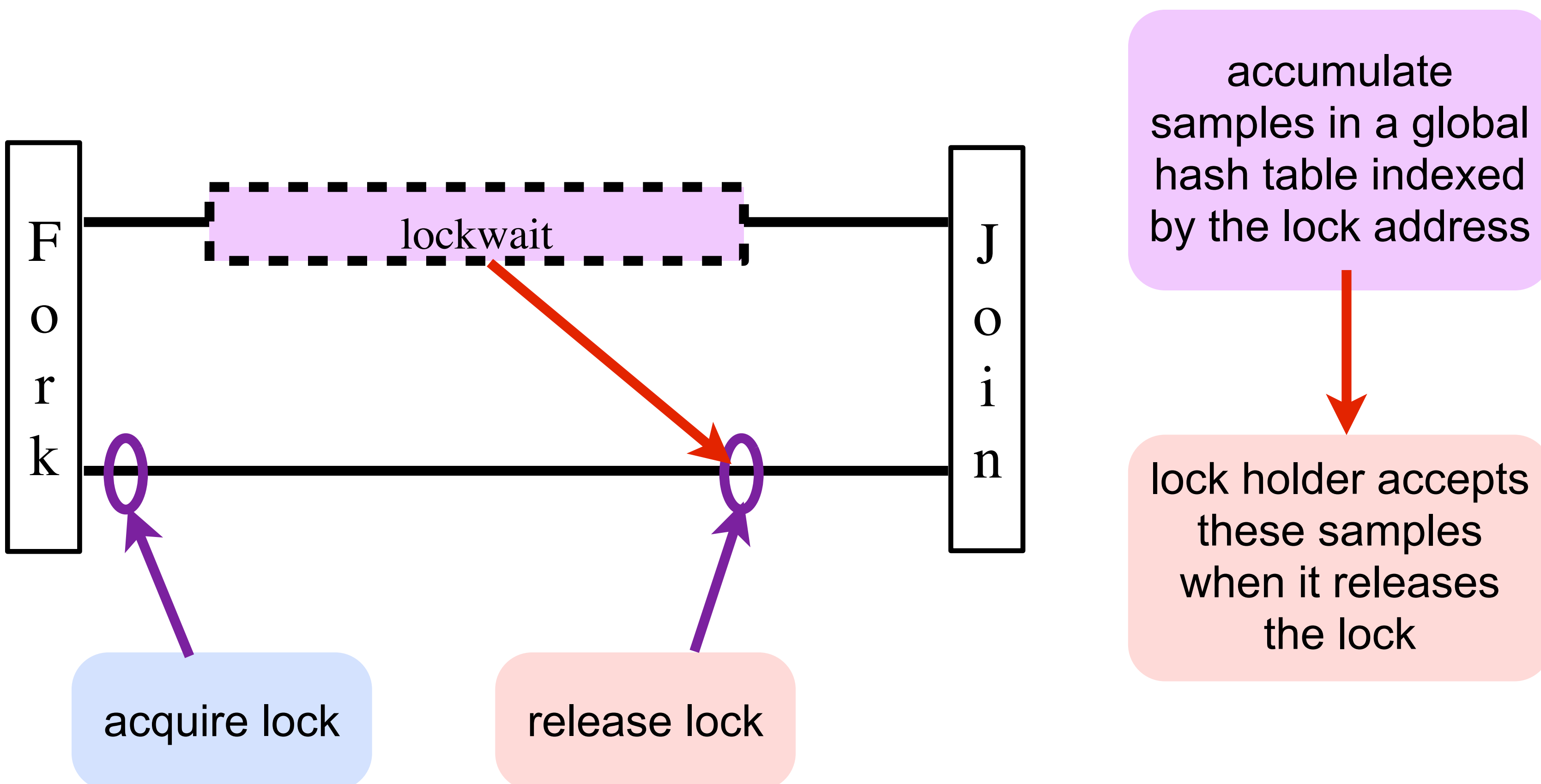
begin/end event pairs

- **Release**
  - lock
  - nest lock
  - critical
  - atomic
  - ordered section

singleton events

Directed Blame Shifting

- **Example:**
  - —threads waiting at a lock are the symptom
  - —the cause is the lock holder
- **Approach: blame lock waiting on lock holder**

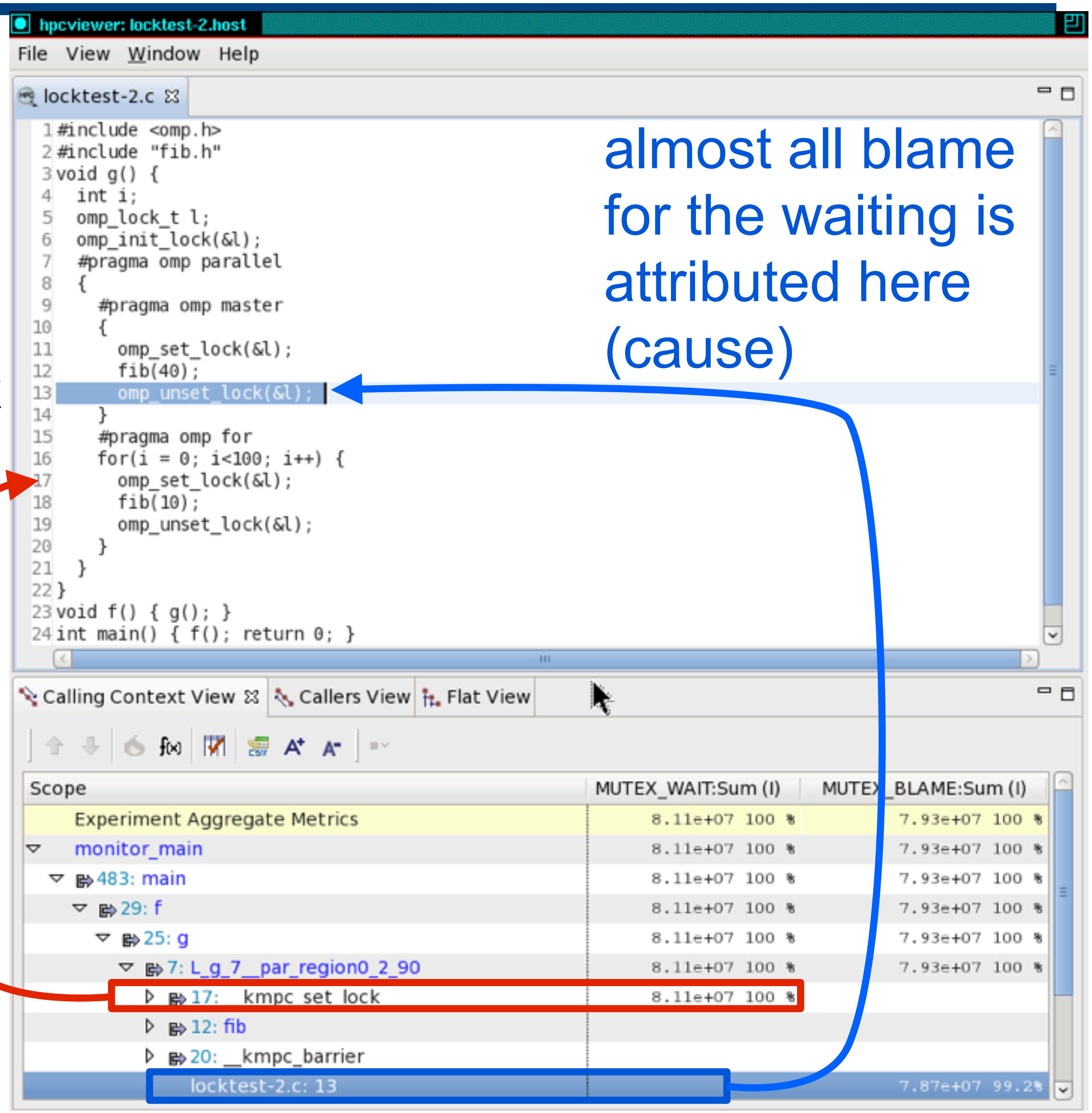


Example: Directed Blame Shifting for

Blame a lock holder for delaying waiting threads

- **Charge all samples that threads receive while awaiting a lock to the lock itself**
- **When releasing a lock, accept blame at the lock**

all of the waiting occurs here (symptom)



Trace Events (Optional)

ompt_event_implicit_task_create	ompt_event_taskgroup_end
ompt_event_implicit_task_exit	ompt_event_release_nest_lock_prev
ompt_event_task_switch	ompt_event_wait_lock
ompt_event_loop_begin	ompt_event_wait_nest_lock
ompt_event_loop_end	ompt_event_wait_critical
ompt_event_section_begin	ompt_event_wait_atomic
ompt_event_section_end	ompt_event_wait_ordered
ompt_event_single_in_block_begin	ompt_event_acquired_lock
ompt_event_single_in_block_end	ompt_event_acquired_nest_lock_first
ompt_event_single_others_begin	ompt_event_acquired_nest_lock_next
ompt_event_single_others_end	ompt_event_acquired_critical
ompt_event_master_begin	ompt_event_acquired_atomic
ompt_event_master_end	ompt_event_acquired_ordered
ompt_event_barrier_begin	ompt_event_init_lock
ompt_event_barrier_end	ompt_event_init_nest_lock
ompt_event_taskwait_begin	ompt_event_destroy_lock
ompt_event_taskwait_end	ompt_event_destroy_nest_lock
ompt_event_taskgroup_begin	ompt_event_flush

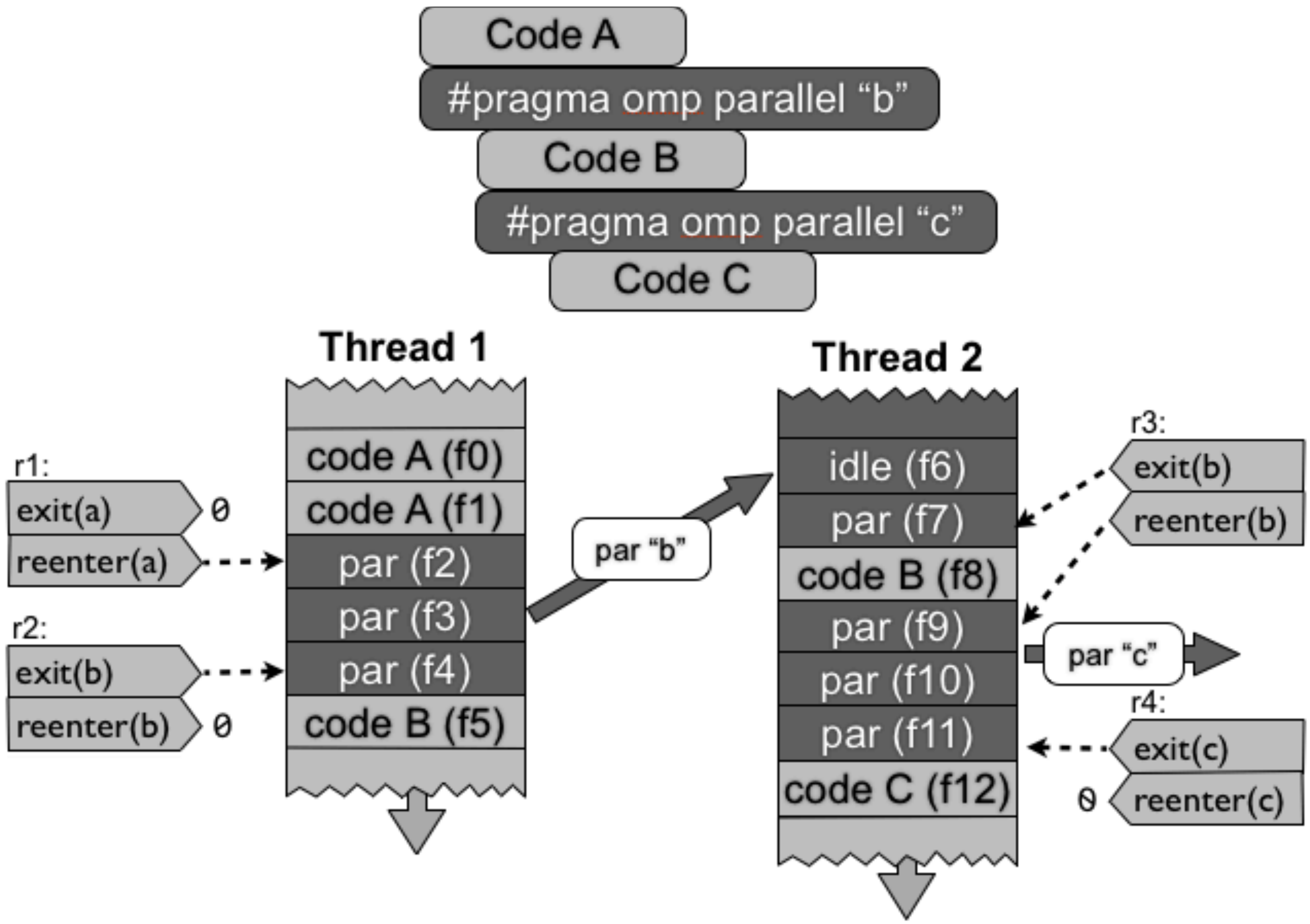
Parallel Region and Task IDs

- Unique id for each parallel region instance and task instance
- Ability to query parallel region and task IDs
  - —ompt_parallel_id_t ompt_get_parallel_id(int ancestor_level)
  - —ompt_task_id_t ompt_get_task_id(int ancestor_level)

  - —current task: ancestor_level = 0
  - —query IDs of ancestor task/region using higher ancestor levels

  - —async signal safe

Call Stack Interpretation



ompt_get_idle_frame(0) → 0

ompt_get_task_frame(0) → r2
ompt_get_task_frame(1) → r1
ompt_get_task_frame(2) → 0

ompt_get_idle_frame(0) → f6

ompt_get_task_frame(0) → r4
ompt_get_task_frame(1) → r3
ompt_get_task_frame(2) → r1
ompt_get_task_frame(3) → 0

Miscellaneous API Features

- Tool-facing API functions
  - initialization
    - int ompt_initialize(ompt_function_lookup_t ompt_fn_lookup,
      const char * version, int ompt_version)
    - int ompt_set_callback(ompt_event_t e, ompt_callback_t cb)
  - state enumeration
    - int ompt_enumerate_state(int current_state, int *next_state,
      const char **next_state_name)
- Tool control
  - void ompt_control(uint64_t command, uint64_t modifier)

Outline

- OMPT - emerging performance tool API for OpenMP
  - —overview and goals
  - —state tracking
  - —event notification
  - —API
- Status and next steps

OMPT Status and Next Steps

- Subcommittee approved the document a few weeks ago
  - a few recent text changes based on late comments
- Available on the WWW for comment: http://bit.ly/ompt-tr
- Nov 7 ARB meeting
  - tools group approved as subcommittee of language committee
- Submit it to OpenMP language committee for official comment
  - turn it into an official OpenMP TR
- Runtime implementations
  - IBM implementation of OMPT interface for BG/Q and Power
  - Rice and Oregon prototype implementation OMPT in Intel runtime
- Tools
  - Rice University's HPCToolkit (development branch)
  - University of Oregon's Tau