Awards: 1450273, 1449918

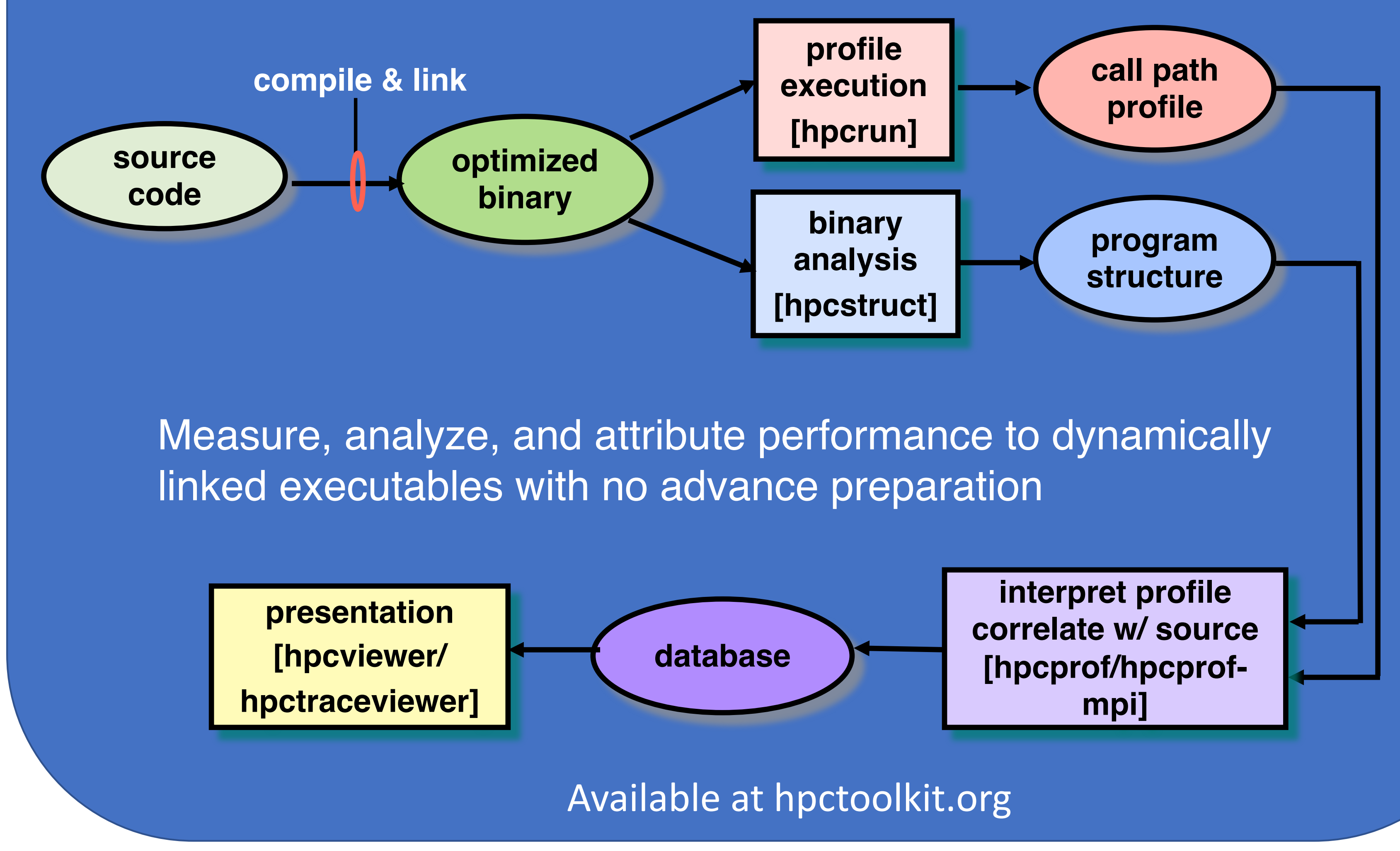
NSF CSSI PI Meeting, Seattle, WA, Feb. 13-14, 2020

SI2-SSI: Collaborative Research: A Sustainable Infrastructure for Performance, Security, and Correctness Tools

PI: John Mellor-Crummey, Co-Pi: Barton P. Miller

Institutions: Rice University and University of Wisconsin - Madison

HPCToolkit: Performance Measurement, Attribution, and Analysis Tool Suite

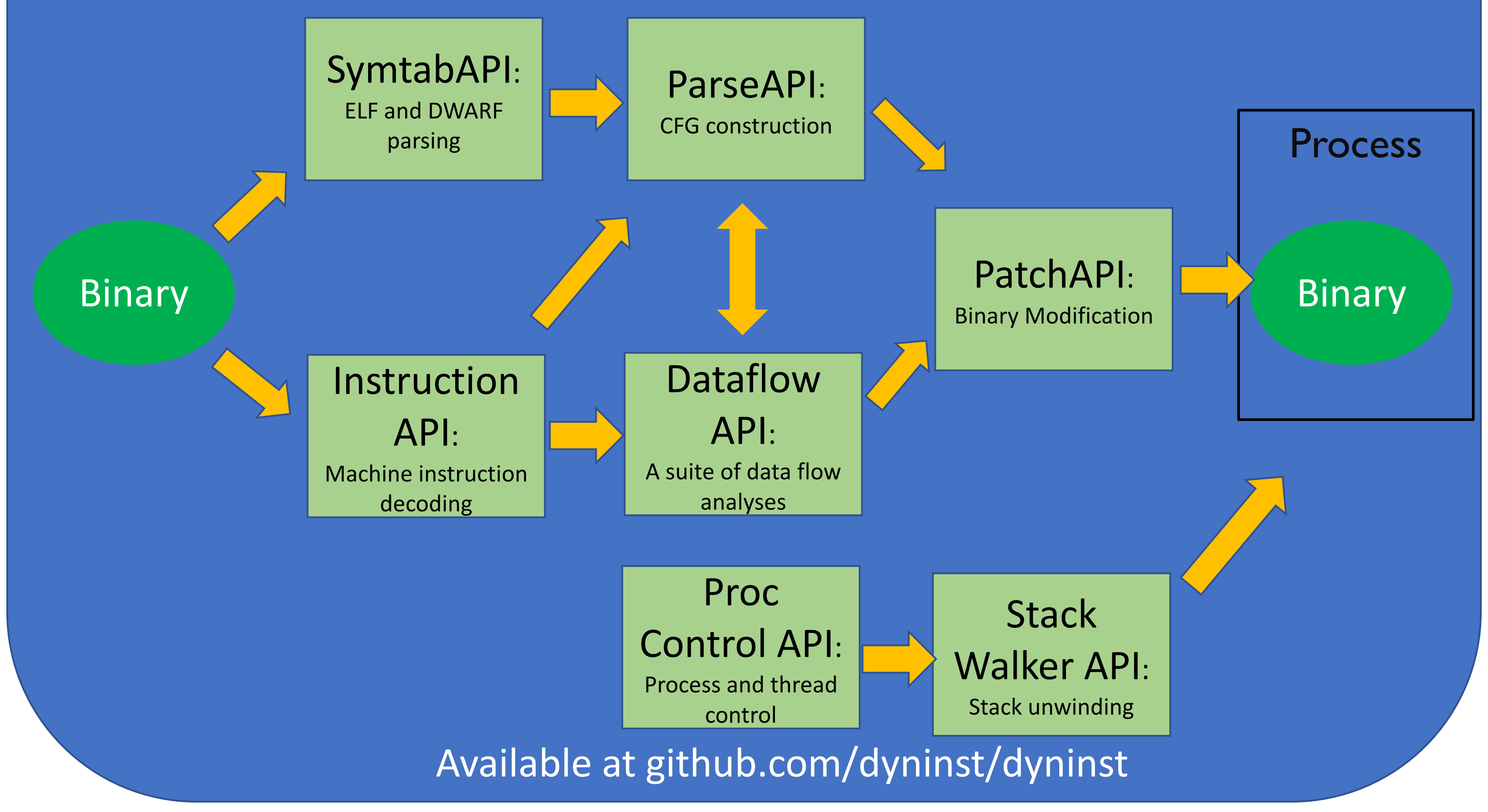

compile & link
source code
optimized binary
profile execution [hpcrun]
call path profile
binary analysis [hpcstruct]
program structure
interpret profile correlate w/ source [hpcprof/hpcprof-mpi]
database
presentation [hpcviewer/ hpctraceviewer]

Measure, analyze, and attribute performance to dynamically linked executables with no advance preparation

Available at hpctoolkit.org

Dyninst: Binary Analysis and Instrumentation Tool Suite

Component libraries for binary structure and code analysis, dynamic instrumentation, and static binary rewriting:

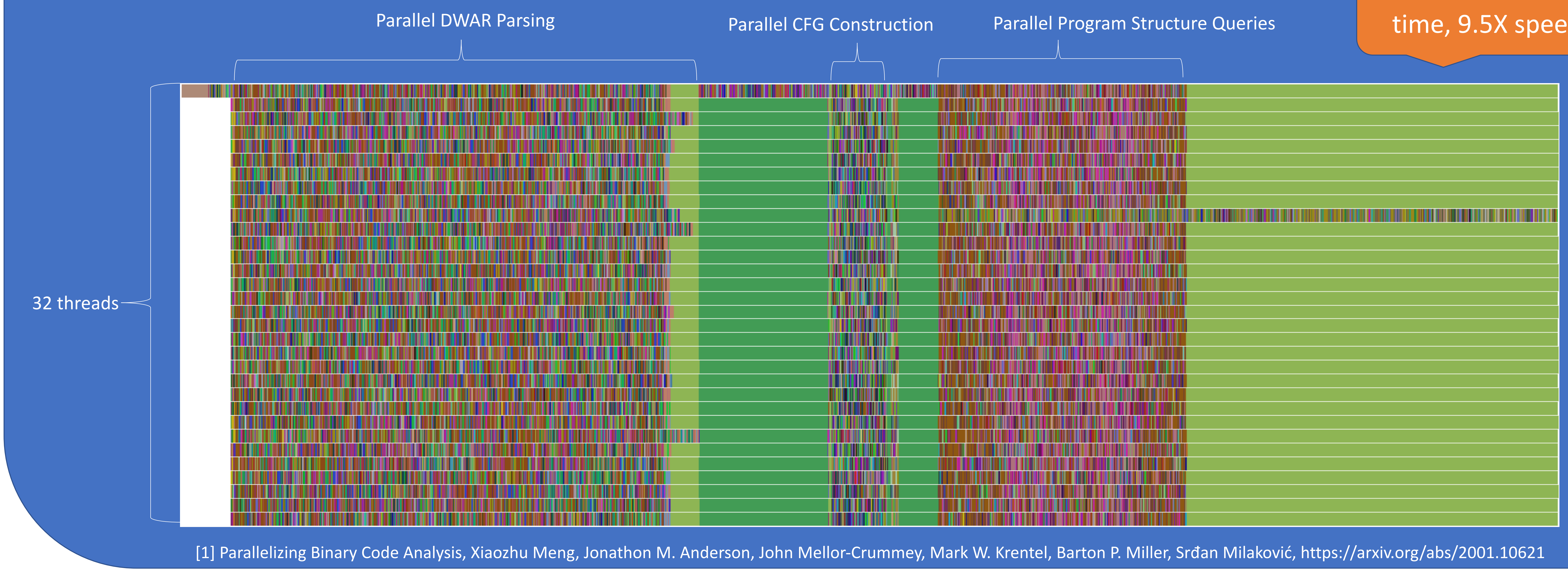

SymtabAPI: ELF and DWARF parsing
ParseAPI: CFG construction
Binary
Instruction API: Machine instruction decoding
Dataflow API: A suite of data flow analyses
PatchAPI: Binary Modification
Process
Binary
Proc Control API: Process and thread control
Stack Walker API: Stack unwinding

Available at github.com/dyninst/dyninst

Analyze Large Scale Binaries

- Added multi-threading to Dyninst's ParseAPI and SymtabAPI [1]
- HPCToolkit's hpcstruct uses Dyninst's ParseAPI to analyze loop nesting, source line mapping, and function inlining
- A trace view of hpcstruct analyzing a 8.2GB shared library from TensorFlow

2.5 mins total analysis time, 9.5X speed up

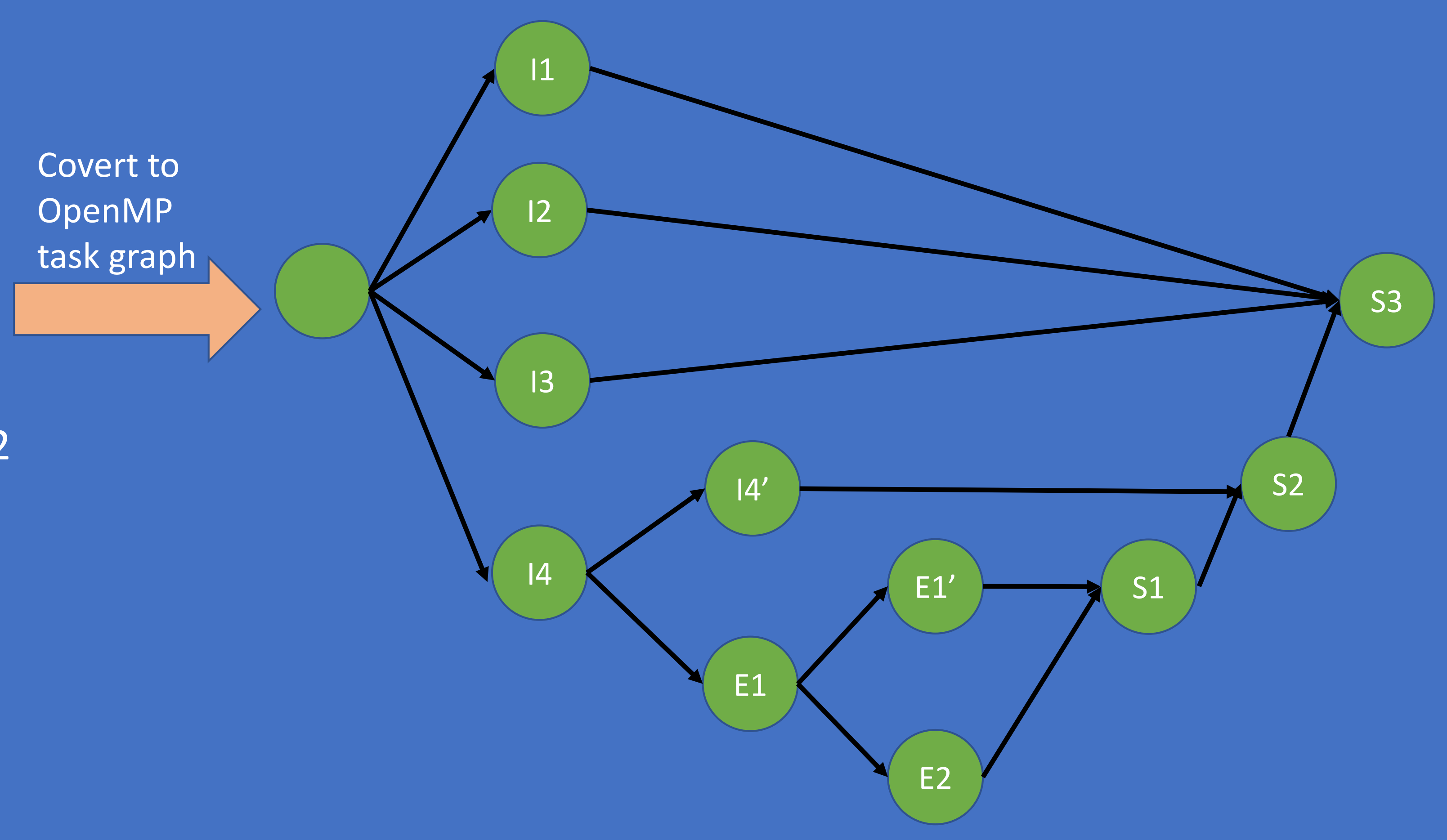


[1] Parallelizing Binary Code Analysis, Xiaozhu Meng, Jonathon M. Anderson, John Mellor-Crummey, Mark W. Krentel, Barton P. Miller, Srđan Milaković, https://arxiv.org/abs/2001.10621

Data Race Detection for OpenMP Programs

- Instrument memory reads and writes using Dyninst to track happens-before relation
- Reduce state information maintained by leveraging OpenMP semantics

```
#pragma omp parallel for
for (int i = 0; i < 4; ++i) {  // I1 – I4
    if (i < 3) {
        int local = shared;
        // Processing local
    } else {
        #pragma omp task // E1
        {
            int local = shared;
            #pragma omp task // E2
            {
                shared *= 2;
            }
            #pragma omp taskwait
        }
    }
}
```



Basic algorithm execution trace
1. I1, I2, and I3 perform concurrent read
-> Record access history: I1, I2, and I3 read shared
2. E1 and E2 perform concurrent read
-> Record access history: E1 and E2 read shared
3. E2 performs a concurrent write
-> Data race
Five task IDs are recorded for shared

New algorithm execution trace
1. I1, I2, and I3 perform concurrent read
-> Record access history: I1 and I2 read shared; no need to record I3
2. E1 and E2 perform concurrent read
-> no need to record E1 or E2
3. E2 performs concurrent write
-> Data race
Two task IDs are recorded for shared

Use static binary analysis to reduce instrumentation:

- Avoid instrumentation for accesses to the same variable by the same task in the same synchronization interval
- Avoid instrumentation for accesses to read-only memory locations