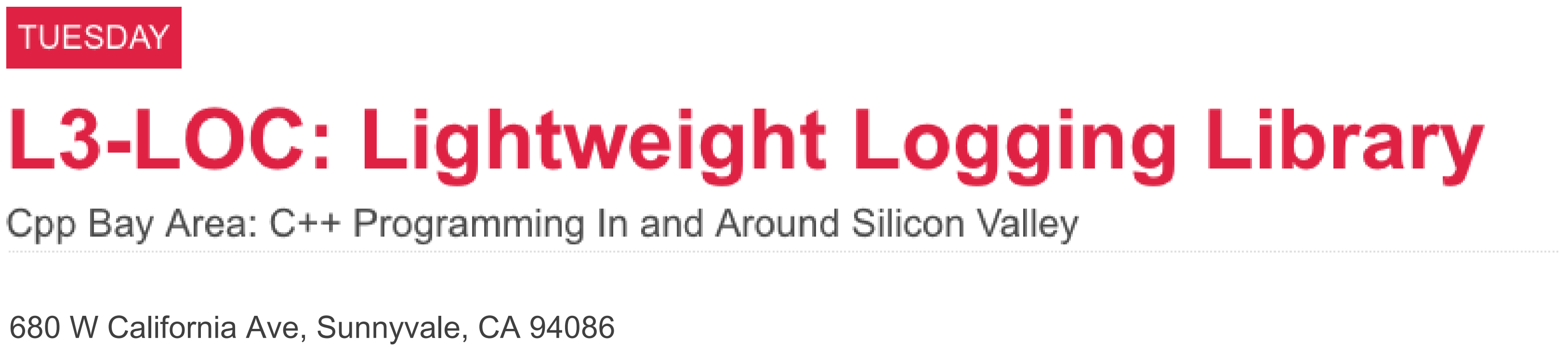


TUESDAY

L3-LOC: Lightweight Logging Library

Cpp Bay Area: C++ Programming In and Around Silicon Valley

680 W California Ave, Sunnyvale, CA 94086

This month's speaker is: Aditya, a Soft-Where engineer.

This talk presents L3, a small C/C++ library designed for high-speed, non-intrusive, logging of events in an mmap()'ed log file, integrated with C++20's source_location{} class.

We then present two alternate, extremely compact, Line-Of-Code [LOC] encoding techniques, both requiring just 4 bytes of footprint for each source-location reference tracked. And, both these schemes work with older C++ compilers and also with C.

We show how L3-LOC logging can be very effective to troubleshoot race-conditions in high-performance timing-sensitive applications.

Dr Greg Law
Founder / CEO

Aditya Gurajada, Soft Where Engineer, Bay Area, CA
Greg Law, Undo.io, Cambridge, UK

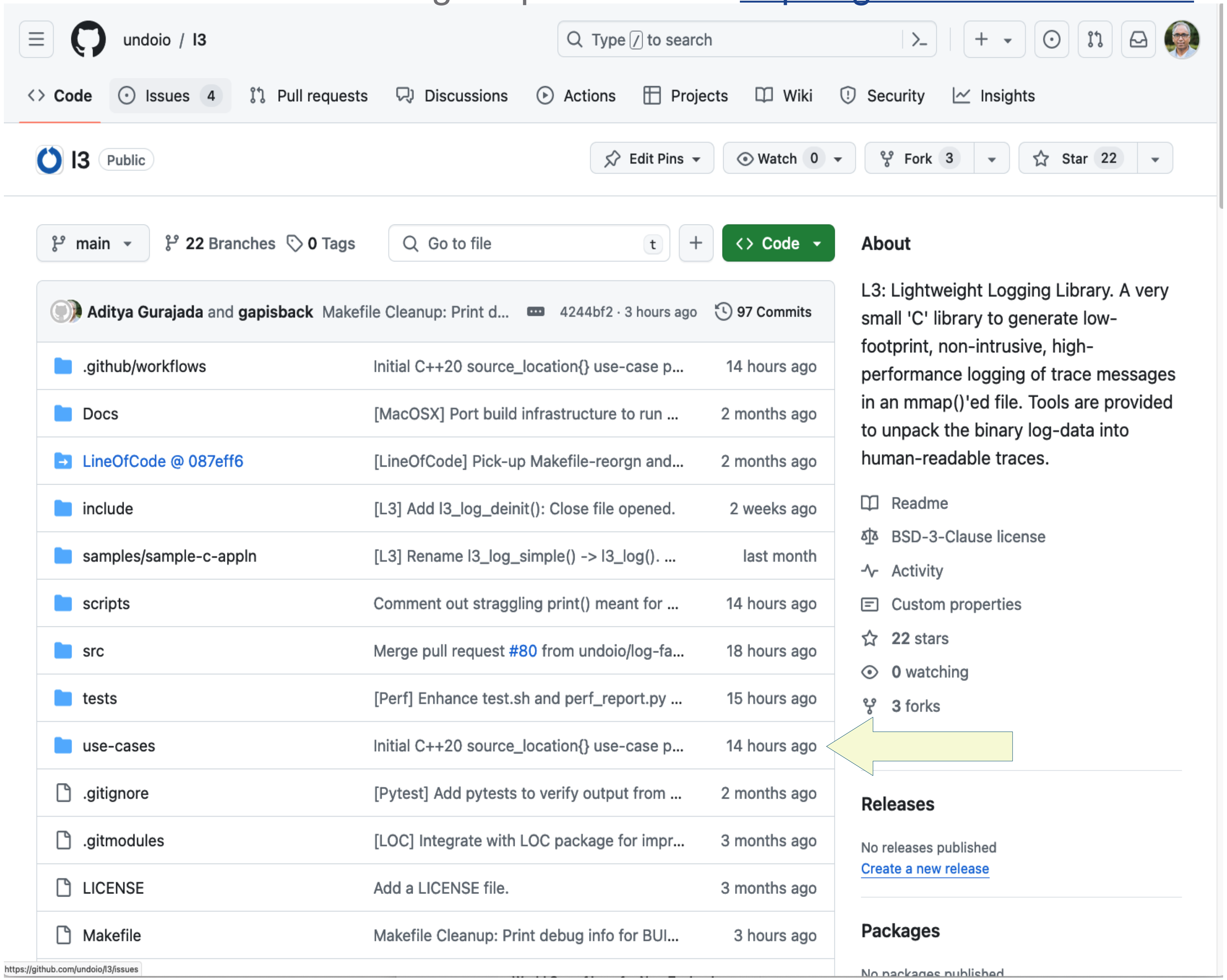
Tue, Jun 18 2024

undo

Undo is the time travel debugging company for Linux.

Win a FREE 6-month UDB license

Source code and benchmarking scripts are here: https://github.com/undoio/l3



Aditya Gurajada and gapisback Makefile Cleanup: Print d... 4244bf2 · 3 hours ago 97 Commits

.github/workflows	Initial C++20 source_location{} use-case p...	14 hours ago
Docs	[MacOSX] Port build infrastructure to run ...	2 months ago
LineOfCode @ 087eff6	[LineOfCode] Pick-up Makefile-reorgn and...	2 months ago
include	[L3] Add l3_log_deinit(): Close file opened.	2 weeks ago
samples/sample-c-appln	[L3] Rename l3_log_simple() -> l3_log(). ...	last month
scripts	Comment out straggling print() meant for ...	14 hours ago
src	Merge pull request #80 from undoio/log-fa...	18 hours ago
tests	[Perf] Enhance test.sh and perf_report.py ...	15 hours ago
use-cases	Initial C++20 source_location{} use-case p...	14 hours ago
.gitignore	[Pytest] Add pytests to verify output from ...	2 months ago
.gitmodules	[LOC] Integrate with LOC package for impr...	3 months ago
LICENSE	Add a LICENSE file.	3 months ago
Makefile	Makefile Cleanup: Print debug info for BUI...	3 hours ago

https://github.com/undoio/l3/issues

About

L3: Lightweight Logging Library. A very small 'C' library to generate low-footprint, non-intrusive, high-performance logging of trace messages in an mmap()'ed file. Tools are provided to unpack the binary log-data into human-readable traces.

- Readme
- BSD-3-Clause license
- Activity
- Custom properties
- 22 stars
- 0 watching
- 3 forks

Releases

No releases published

Create a new release

Packages

No packages published

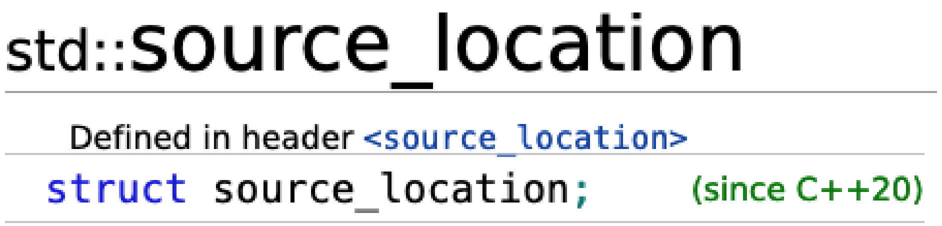
Agenda

1. Motivation: What is the problem you are solving?
2. Diagnostic interfaces
   - C++20 source_location{} - Overview and demo
   - L3: Lightweight Logging Library (github) – Overview and demo
   - LOC – Line-Of-Code encoding schemes – Overview and demo
3. Benchmarking Results: Comparing different logging schemes
4. Client/Server msg-exchange program: Perf u-benchmarking
5. Future directions

Motivation: Troubleshooting failure(s) in complex systems

- **Claim**: Debugging race conditions is difficult
- Instrumentation to track the “state” leading to the race could perturb timing – race gets hidden
- Traditional printf()-style logging is intrusive and could perturb timing
- **Requirement**: Minimally-intrusive logging scheme designed for highly concurrent programs
- Logging involves recording:
  - State of variables
  - And possibly code-location where the message was logged (or the state was gathered)
-

C++20 source_location{}



std::source_location

Defined in header <source_location>

```
struct source_location;
```

(since C++20)

- std::source_location class represents certain information about the source code:
  - File names, line numbers, and function names.
  - [Is] Better alternative to predefined macros like __FILE__, __LINE__, __FUNC__ which are expanded in the context of the caller (*i.e., at the call-site of the caller*)
- **Usage**: Functions that desire to obtain this information about the call site (for logging, testing, or debugging purposes)
- **Intended** that std::source_location has a small size and can be copied efficiently
- **Unspecified** whether the copy/move constructors and the copy / move assignment operators of std::source_location are trivial and/or constexpr
- std::source_location meets the *DefaultConstructible*, *CopyConstructible*, *CopyAssignable* and *Destructible* requirements.
  - lvalue of std::source_location meets the *Swappable* requirement.

C++20 source_location{} - Implementation

Documented in Cpp Reference

```
namespace std {
  struct source_location {
    // source location construction
    static consteval source_location current() noexcept;
    constexpr source_location() noexcept;
    // source location field access
    constexpr uint_least32_t line() const noexcept;
    constexpr uint_least32_t column() const noexcept;
    constexpr const char* file_name() const noexcept;
    constexpr const char* function_name() const noexcept;
  private:
    uint_least32_t line_;               // exposition only
    uint_least32_t column_;             // exposition only
    const char* file_name_;             // exposition only
    const char* function_name_;         // exposition only
  };
}
```

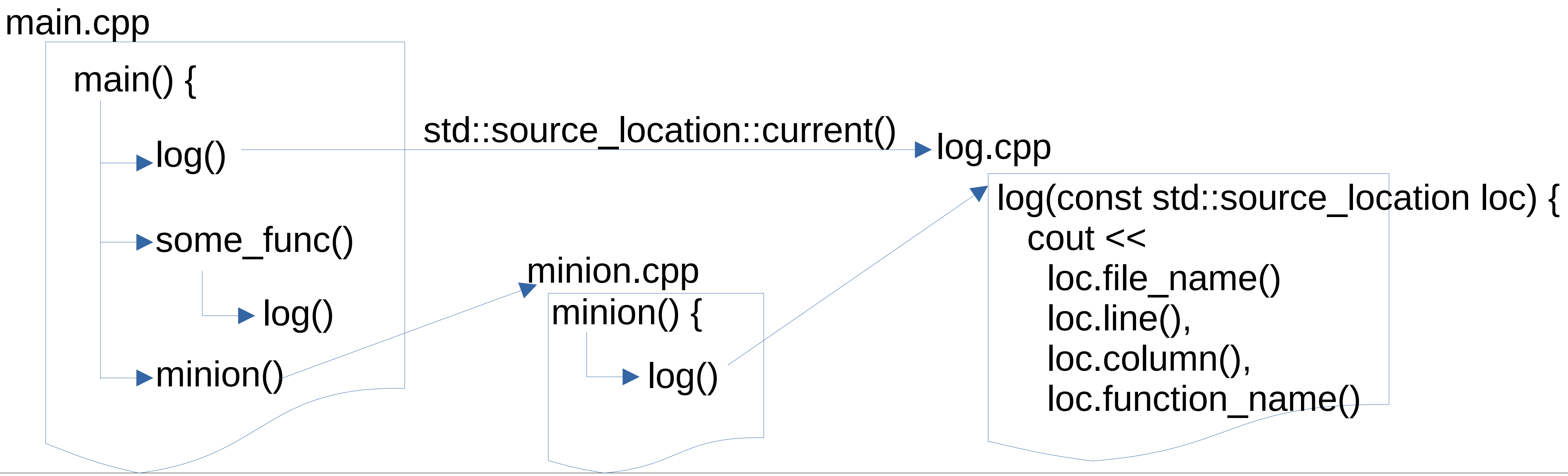
24-bytes[?]

On Ubuntu Linux v22.04.4: Under gdb

```
(gdb) ptype location
type = const struct std::source_location {
  private:
    const std::source_location::__impl *_M_impl;

  public:
    static std::source_location current(__builtin_ret_type);
    source_location(void);
    uint_least32_t line(void) const;
    uint_least32_t column(void) const;
    const char * file_name(void) const;
    const char * function_name(void) const;

  private:
    typedef const void *__builtin_ret_type;
    typedef unsigned int uint_least32_t;
}
```

8-bytes

C++20 source_location{} - Demo Sample program source layout



```
(Thread 2 hit Breakpoint 3, log (msg=..., loc=...) at use-cases/source-location-Cpp-program/source-location-log.cpp:13

(gdb)
use-cases/source-location-Cpp-program/source-location-main-C++20.cpp:20:35::void some_func(T) [with T = const char*]: 'Hello C++20: Lock Release!'

23              std::source_location curr_location = std::source_location::current();
(gdb)
25              return curr_location;

(gdb) p sizeof(curr_location)
$3 = 8

(gdb) p curr_location
$4 = <optimized out>
```

Line number

Column number

Function name with signature, showing that it's a template function

NOTE: [Seems like] It's just an 8-byte opaque handle to some region in the data section.

Suspect it's optimized away as compiler generates a constexpr

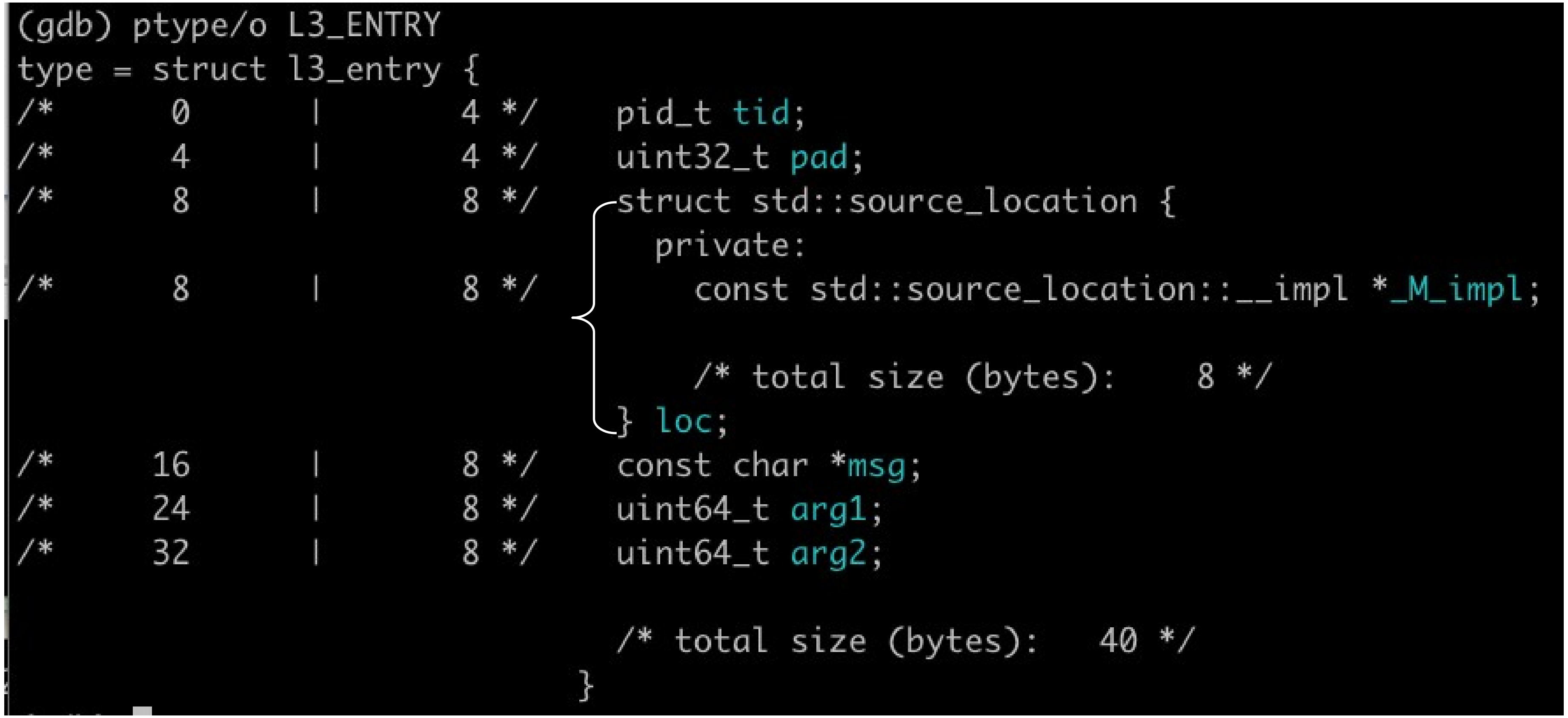
C++20 source_location{} - Sample program Demo

$ cd ~/Projects/l3

$ ./test.sh test-build-and-run-source-location-cpp20-sample

gdb info for L3 structure integrated with C++20 source_location{}:

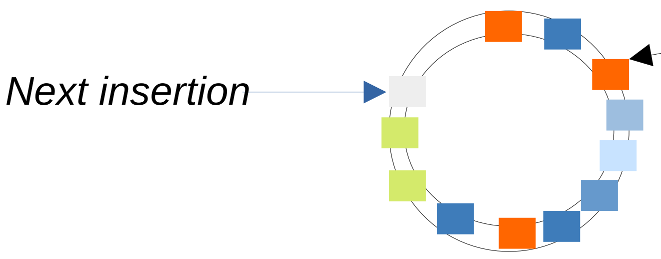
loc handle is just a pointer to some data location.



```
(gdb) ptype/o L3_ENTRY
type = struct l3_entry {
/*      0      |       4 */    pid_t tid;
/*      4      |       4 */    uint32_t pad;
/*      8      |       8 */    struct std::source_location {
                                 private:
/*      8      |       8 */        const std::source_location::__impl *_M_impl;

                                   /* total size (bytes):    8 */
                               } loc;
/*     16      |       8 */    const char *msg;
/*     24      |       8 */    uint64_t arg1;
/*     32      |       8 */    uint64_t arg2;

                               /* total size (bytes):   40 */
                             }
```

L3 Logging Interfaces, with LOC-support

- **Simple** interfaces to log a message, with 2 arguments, to a memory-mapped file
- Optionally, record Line-of-Code (LOC) where log was generated. ~ C++20 source_location
- Log-file is a ring-buffer of C-structs
- Indexed by an atomic global counter
- Designed for high-performance concurrent multi-threaded logging
- Python script to post-process log-file to generate human-readable output



Initialize Logging

```
logfile = "/tmp/l3.cpp-small-test.dat";

int e = l3_init(logfile);
```

Standard Logging: 2 parameters (no varargs, yet)

```
l3_log("Simple-log-msg-Args(arg1=%d, arg2=%d)", 1, 2);

void *bp = (void *) 0xdeadbabe;
l3_log("Potential memory overwrite (addr=%p, size=%d)",
           bp, 1024);
```

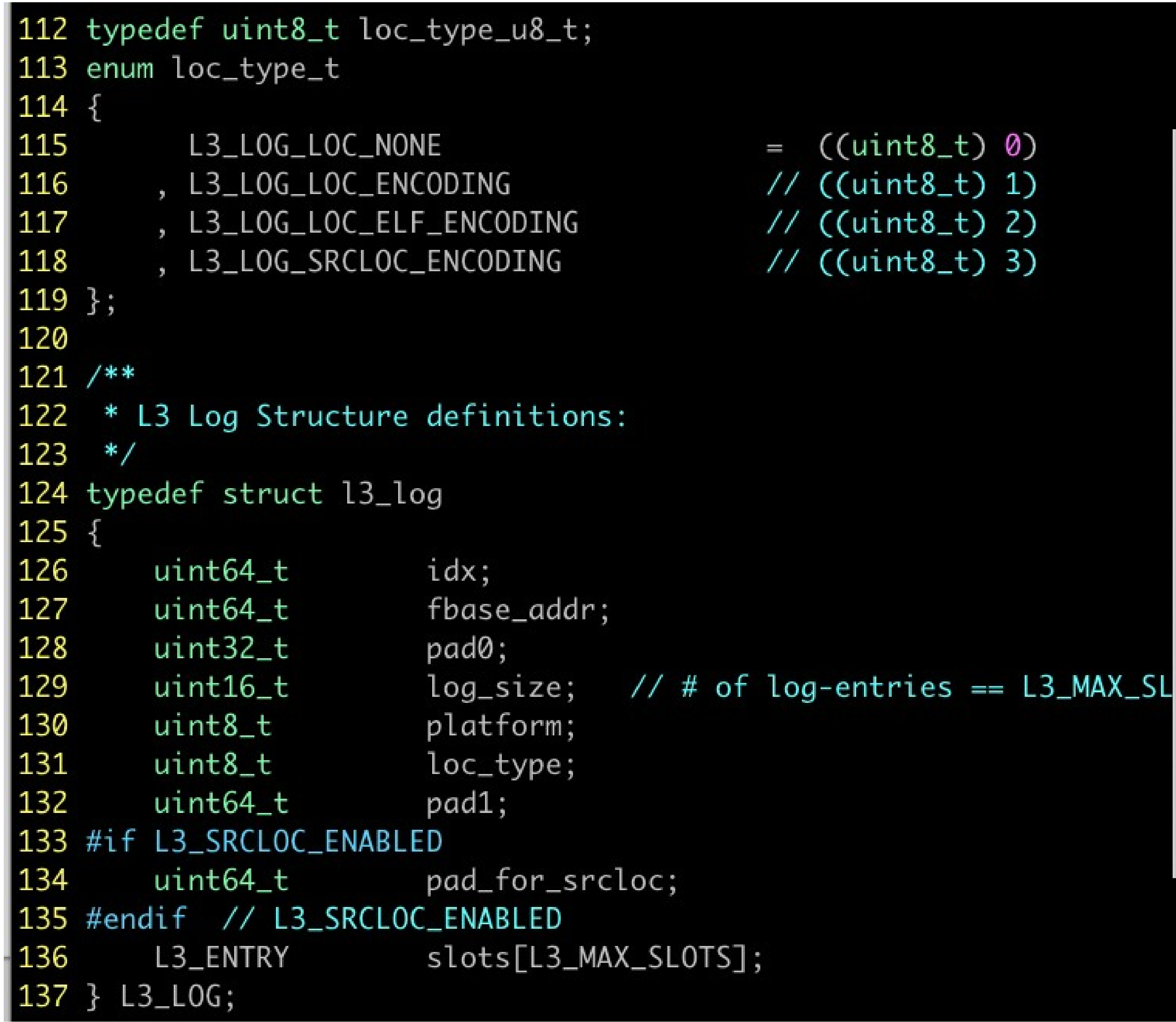
Fast Logging (x86-64 Assembly support)

```
bp = (int *) 0xdeadbeef;
l3_log_fast("Fast-logging ctr=%d, addr=%p", 10, bp);
```

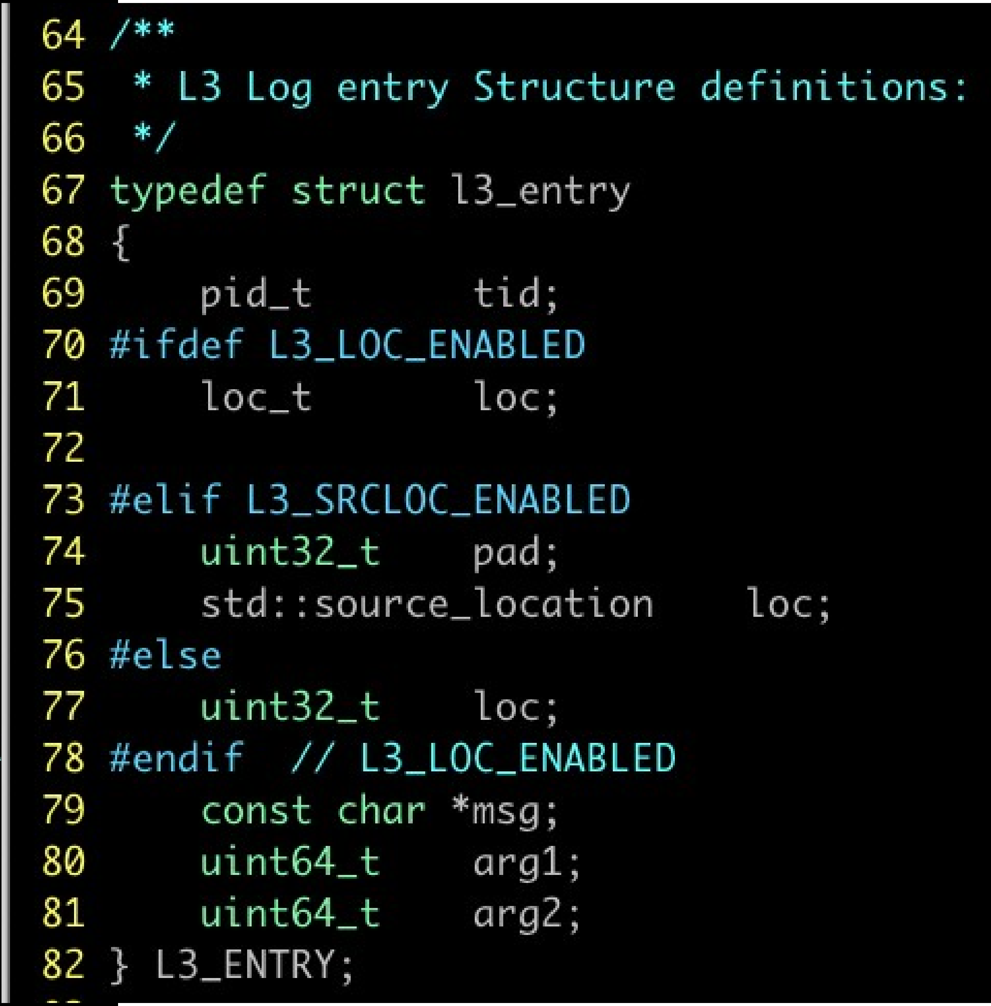
Log entry Structure (32 bytes)

```
struct {
  pid_t          tid;       // User thread-ID
  [u]int32_t     loc;       // (Optional) Line-of-Code ID
  const char    *msg;       // Diagnostic message literal
  uint64_t       arg1;      // Argument value-1
  uint64_t       arg2;      // Argument value-2
};
```

_ENABLED={0,1,2,3}



```
typedef uint8_t loc_type_u8_t;
enum loc_type_t
{
      L3_LOG_LOC_NONE                   =  ((uint8_t) 0)
    , L3_LOG_LOC_ENCODING               // ((uint8_t) 1)
    , L3_LOG_LOC_ELF_ENCODING           // ((uint8_t) 2)
    , L3_LOG_SRCLOC_ENCODING            // ((uint8_t) 3)
};

/**
 * L3 Log Structure definitions:
 */
typedef struct l3_log
{
    uint64_t        idx;
    uint64_t        fbase_addr;
    uint32_t        pad0;
    uint16_t        log_size;   // # of log-entries == L3_MAX_SL
    uint8_t         platform;
    uint8_t         loc_type;
    uint64_t        pad1;
#if L3_SRCLOC_ENABLED
    uint64_t        pad_for_srcloc;
#endif  // L3_SRCLOC_ENABLED
    L3_ENTRY        slots[L3_MAX_SLOTS];
} L3_LOG;
```



```
/**
 * L3 Log entry Structure definitions:
 */
typedef struct l3_entry
{
    pid_t       tid;
#ifdef L3_LOC_ENABLED
    loc_t       loc;

#elif L3_SRCLOC_ENABLED
    uint32_t    pad;
    std::source_location    loc;
#else
    uint32_t    loc;
#endif  // L3_LOC_ENABLED
    const char *msg;
    uint64_t    arg1;
    uint64_t    arg2;
} L3_ENTRY;
```

L3 Logging with LOC-encoding schemes: LOC_ENABLED={0,1,2,3}

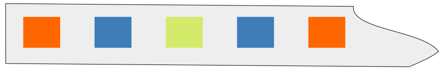
LOC_ENABLED	Description (Scrum stand-up update)	Compiler support	L3-dump support to decode LOC-ID into constituent file / function-name, line number	Platform support	
				Mac	Linux
0	(No call-site line-of-code info logged)	(n/a)	(n/a)		
1	4-byte LOC-ID encoding in .h/.c files generated by Python script at build-time	gcc / g++ (C++11)	Productized	✓	✓
2	4-byte LOC-ID encoding, generated using ELF-magic in .rodata section of the binary. Statically defined magic	gcc / g++ (C++11)	Prototyped		✓
3	8-byte source_location{} handle pointing to somewhere in the "data"-section of the program binary	g++ using C++20	To be developed		

- **Docker / VM-Linux**: Ubuntu 22.04.4 LTS: gcc v11.4.0; g++ v11.4.0
- CI-jobs run clean on Linux and Mac/OS as well

- **Mac/OSX** (Monterey v12.1): gcc (Homebrew GCC 14.1.0_1), g++ (Homebrew GCC 14.1.0_1)
  - Most of this works on [my] Mac/OS, where /usr/bin/gcc (/usr/bin/g++) is really clang (version 13.1.6). Needed gcc v14.1.0 to use C++20
- Clang: Haven't fully stabilized on Linux (Ubuntu clang version 14.0.0) or Mac/OS (Apple clang version 13.1.6)

L3-Logging Demo - Unit-tests program

```
$ cd ~/Projects/l3
$ make clean && CC=g++ LD=g++ make run-unit-tests
```



```
$ ./build/release/bin/unit/l3_dump.py-test
Generated 4 slow log-entries to log-file: /tmp/l3.c-small-unit-test.dat
Generated 5 fast log-entries to log-file: /tmp/l3.c-fast-unit-test.dat
```

```
python3 l3_dump.py --log-file /tmp/l3.c-small-unit-test.dat --binary ./build/release/bin/unit/l3_dump.py-test
tid=1809 'Simple-log-msg-Args(arg1=1, arg2=2)'
tid=1809 'Simple-log-msg-Args(arg3=3, arg4=4)'
tid=1809 'Potential memory overwrite (addr=0xdeadbabe, size=1024)'
tid=1809 'Invalid buffer handle (addr=0xbeefabcd), lockrec=0x0'
Unpacked nentries=4 log-entries.
```

```
python3 l3_dump.py --log-file /tmp/l3.c-fast-unit-test.dat --binary ./build/release/bin/unit/l3_dump.py-test
tid=1809 'Fast-log-msg: Args(arg1=1, arg2=2)'
tid=1809 'Fast-log-msg: Args(arg3=3, arg4=4)'
tid=1809 'Fast-log-msg: Args(arg1=10, arg2=20)'
tid=1809 'Fast-log-msg: Potential memory overwrite (addr=0xdeadbabe, size=1024)'
tid=1809 'Fast-log-msg: Invalid buffer handle (addr=0xbeefabcd), unused=0'
Unpacked nentries=5 log-entries.
```

L3-LOC Logging Demo - use-cases/single-file-Cpp-program

```
$ cd ~/Projects/l3
$ make clean && CC=g++ CXX=g++ LD=g++ L3_LOC_ENABLED=1 make all-cpp-tests
```

```
$ build/release/bin/use-cases/single-file-Cpp-program
Exercise in-memory logging performance benchmarking: 300 Mil simple/fast log msgs. L3-log file: /tmp/l3.cpp-test.dat
300 Mil simple log msgs: 2ns/msg (avg)
300 Mil fast log msgs: 3ns/msg (avg)
L3-logging 5 entries to unit-tests log file: /tmp/l3.cpp-small-test.dat
```

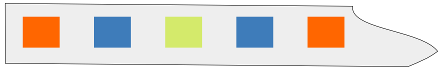
Dump script “recognizes” LOC-encoded dump and decodes loc-ID to filename / line number.

```
$ python3 l3_dump.py --log-file /tmp/l3.cpp-small-test.dat \
                     --binary build/release/bin/use-cases/single-file-Cpp-program
tid=2093 single-file-Cpp-program/test-main.cpp:68  'Simple-log-msg-Args(arg1=1, arg2=2)'
tid=2093 single-file-Cpp-program/test-main.cpp:71  'Potential memory overwrite (addr=0xdeadbabe, size=1024)'
tid=2093 single-file-Cpp-program/test-main.cpp:74  'Invalid buffer handle (addr=0xbeefabcd, refcount=0)'
tid=2093 single-file-Cpp-program/test-main.cpp:77  'Fast-logging msg1=10, addr=0xdeadbeef'
tid=2093 single-file-Cpp-program/test-main.cpp:79  'Fast-logging msg2=20, addr=0xbeefbabe'
Unpacked nentries=5 log-entries.
```

L3-LOC-ELF Demo - use-cases/single-file-Cpp-program

```
$ cd ~/Projects/l3
$ make clean && CC=g++ CXX=g++ LD=g++ L3_LOC_ENABLED=2 make all-cpp-tests
$ build/release/bin/use-cases/single-file-Cpp-program
```

```
$ python3 l3_dump.py --log-file /tmp/l3.cpp-small-test.dat --binary build/release/bin/use-cases/single-file-Cpp-program

tid=3158 loc=-32 'Simple-log-msg-Args(arg1=1, arg2=2)'
tid=3158 loc=-64 'Potential memory overwrite (addr=0xdeadbabe, size=1024)'
tid=3158 loc=-96 'Invalid buffer handle (addr=0xbeefabcd, refcount=0)'
tid=3158 loc=-128 'Fast-logging msg1=10, addr=0xdeadbeef'
tid=3158 loc=-160 'Fast-logging msg2=20, addr=0xbeefbabe'
Unpacked nentries=5 log-entries.
```



Provide LOC-decoder binary to dump script which “recognizes” LOC-encoded dump and decodes loc-ID to filename / line number.

```
$ python3 l3_dump.py --log-file /tmp/l3.cpp-small-test.dat --binary build/release/bin/use-cases/single-file-Cpp-program \
                                     --loc-binary ~/tmp/loc-elf-id-decoder

tid=3158 use-cases/single-file-Cpp-program/test-main.cpp:73::main() 'Simple-log-msg-Args(arg1=1, arg2=2)'
tid=3158 use-cases/single-file-Cpp-program/test-main.cpp:76::main() 'Potential memory overwrite (addr=0xdeadbabe, size=1024)'
tid=3158 use-cases/single-file-Cpp-program/test-main.cpp:79::main() 'Invalid buffer handle (addr=0xbeefabcd, refcount=0)'
tid=3158 use-cases/single-file-Cpp-program/test-main.cpp:84::main() 'Fast-logging msg1=10, addr=0xdeadbeef'
tid=3158 use-cases/single-file-Cpp-program/test-main.cpp:86::main() 'Fast-logging msg2=20, addr=0xbeefbabe'
Unpacked nentries=5 log-entries.
```

L3-C++20 source_location Demo - use-cases/single-file-Cpp-program

```
$ cd ~/Projects/l3
$ make clean && CC=g++ CXX=g++ LD=g++ L3_LOC_ENABLED=3 make all-cpp-tests
$ build/release/bin/use-cases/single-file-Cpp-program
```

```
$ python3 l3_dump.py --log-file /tmp/l3.cpp-small-test.dat --binary build/release/bin/use-cases/single-file-Cpp-program

tid=3235 loc=94918291243120 'Simple-log-msg-Args(arg1=1, arg2=2)'
tid=3235 loc=94918291243088 'Potential memory overwrite (addr=0xdeadbabe, size=1024)'
tid=3235 loc=94918291243056 'Invalid buffer handle (addr=0xbeefabcd, refcount=0)'
Unpacked nentries=3 log-entries.
```

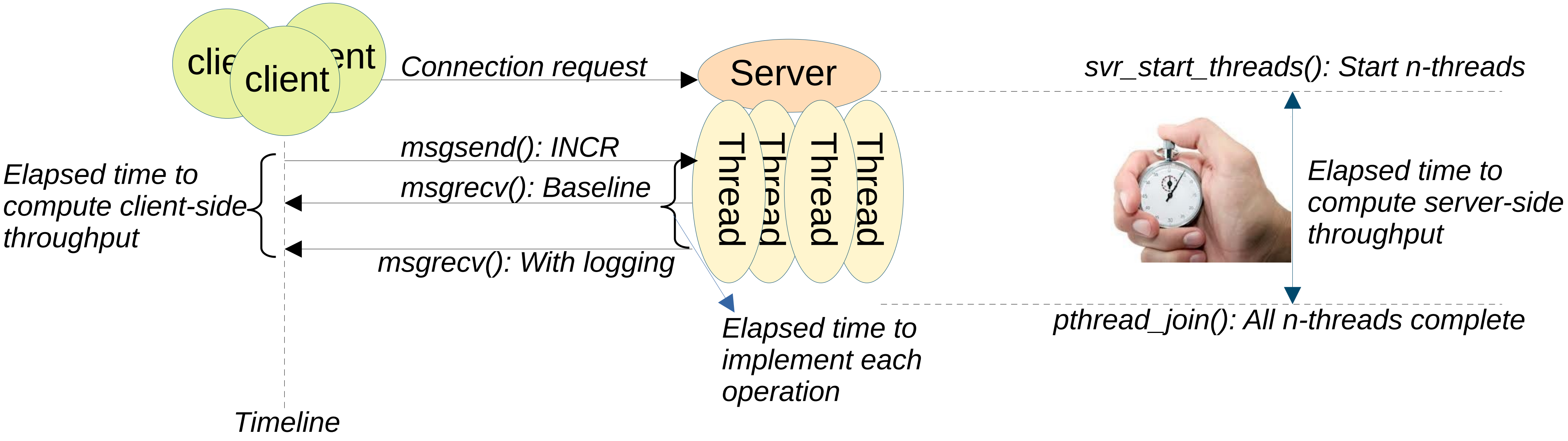
Performance Evaluation: Stand-alone u-benchmarking

```
$ cd ~/Projects/l3

$ make clean && CC=gcc LD=g++ make run-unit-tests

$ CC=g++ CXX=g++ LD=g++ L3_LOC_ENABLED=1 make all-cpp-tests

$ build/release/bin/use-cases/single-file-Cpp-program
```

Performance Evaluation: Client-Server program

Developed a very simple client/server message-passing application for performance micro-benchmarking

- Started from Michael Kerrisk's sample program from The Linux Programming Interface book. Enhanced ...
- Server and client communicate using System V message queues: msgsend(), msgrecv()
- Multi-threaded server; each thread implementing a simple "INCREMENT" op-msg
- Multiple clients send x-Million messages to the server
- Measure server-side throughput (#-msgs/sec) and client-side throughput of RPC
- Calibrate metrics with baseline (no logging) and different logging schemes
- Compare v/s: L3, L3-LOC, L3-ELF-LOC, L3-fprintf(), L3-write(), spdlog, spdlog-backtrace)
- Spdlog: Fast C++ library: https://github.com/gabime/spdlog (22.8K★stars, 4.3K forks)



Connection request

svr_start_threads(): Start n-threads

msgsend(): INCR

msgrecv(): Baseline

msgrecv(): With logging

Elapsed time to compute client-side throughput

Elapsed time to compute server-side throughput

Elapsed time to implement each operation

pthread_join(): All n-threads complete

Timeline

Future directions?

1. More Performance measurements: Understand results
2. Productize LOC-ELF-ID decoder tool for Linux & Mac/OSX
3. Develop source_location{}-ID decoder tool – Linux & Mac/OSX
   - Assembly support for fast-logging source_location{}-ID
4. Productize Clang support
5. Support older /other Linux distros Ubuntu 20.xx [?]

Win a FREE UDB license

Scan and fill in the form to enter

Source code and benchmarking scripts are here: https://github.com/undoio/l3

Thank You

Email: adityagurajada@yahoo.com
Social Media: X