

NanoLog: A Nanosecond Scale Logging System

Stephen Yang

John Ousterhout



PLATFORMLAB

Overview

- **Implemented a fast C++ Logging System**
 - **12.5ns** median latency at **60M** log msgs/sec
 - 10-100x faster than existing systems such as Log4j2 and spdlog
 - Maintains printf-like semantics
- **Shifts work out of the runtime hot-path**
 - Extraction of static information at compile-time
 - Compacted binary output at runtime
 - Defers formatting to an offline process
- **Benefits and Costs**
 - Allows detailed logs in low latency systems
 - Comes at the cost of 1MB of RAM per thread, one core, and disk bandwidth

Why *Fast Logging*?

- **Cornerstone of debugging**
 - Affords visibility application state
 - Helps in root cause analysis after execution
- **Problem: Logging is slow**
 - Application response times are getting faster (microseconds)
 - Logging is not (100-1000's of nanoseconds)
 - Example: RAMCloud response time= 5 μ s, but log time= 1 μ s

What makes logging slow?

```
1473057128.133777014 src/LogCleaner.cc:826 in TombstoneRatioBalancer  
NOTICE: Using tombstone ratio balancer with ratio = 0.400000
```

- **Compute: Complex Formatting**
 - Loggers need to provide context (i.e. file location, time, severity, etc)
 - The message above has **7 arguments** and takes **850ns** to compute
- **Output Bandwidth: Disk IO**
 - On a 250MB/s disk, the **129 byte** message above takes **500ns** to output!

Solutions

```
1473057128.133777014 src/LogCleaner.cc:826 in TombstoneRatioBalancer  
NOTICE: Using tombstone ratio balancer with ratio = 0.400000
```

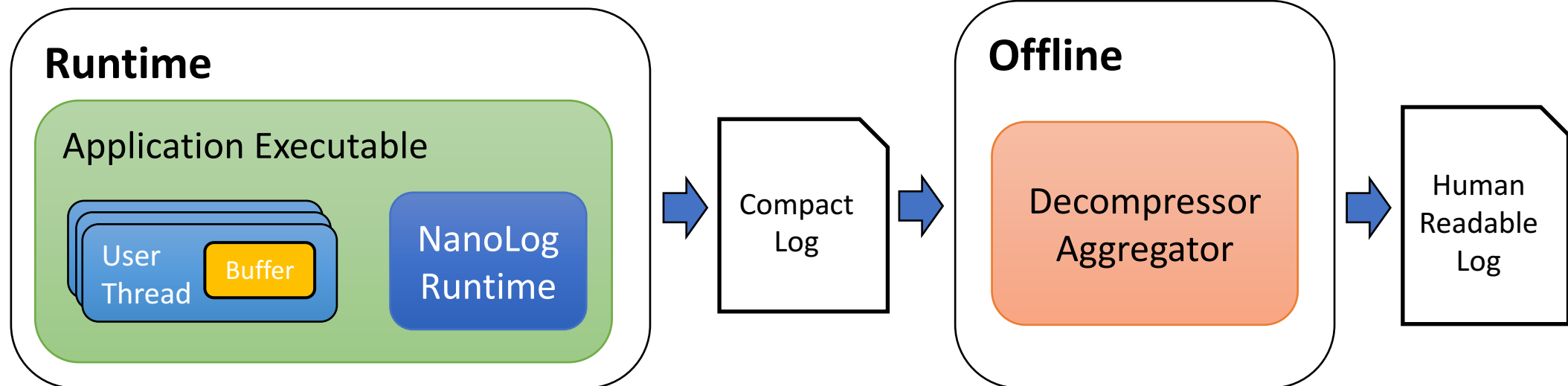
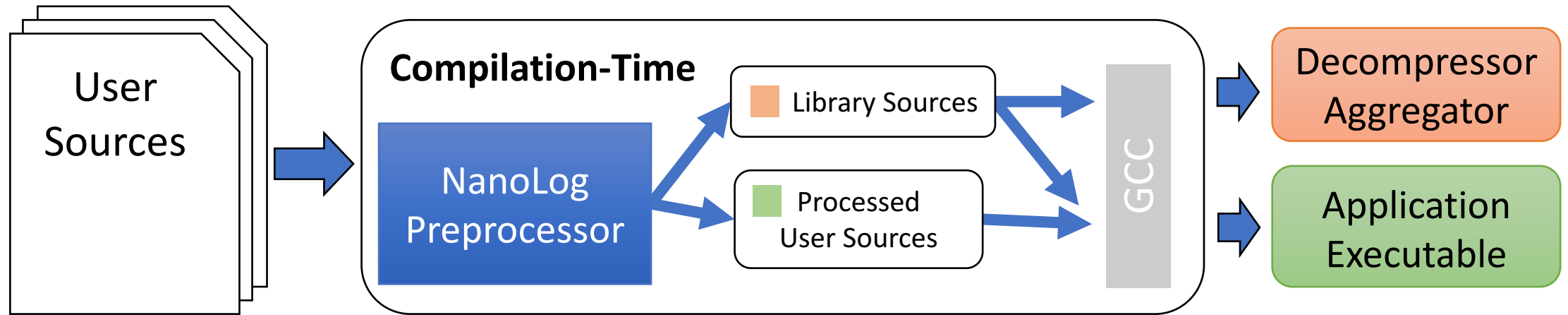
- **Compute: Raw Data Output**

- Most logs in production are not consumed by humans
- Save computation by deferring formatting to an *offline* process
- Side benefit: more efficient for analysis engines

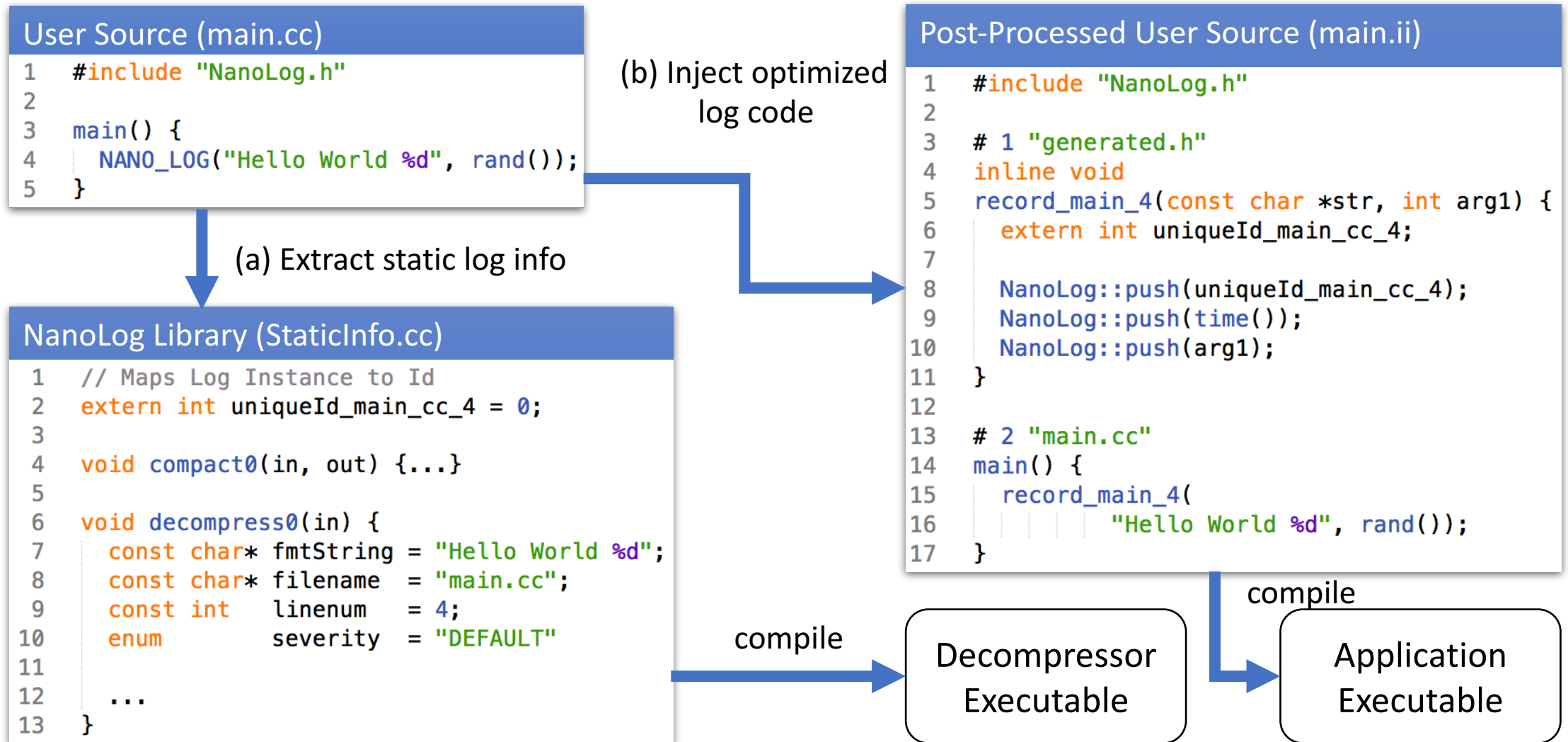
- **IO: Extracting Static Information**

- Static Info in message: file location, line #, function, severity, format string.
- Replace with identifier and compact remaining dynamic information

NanoLog System Architecture



Compile-time Optimizations



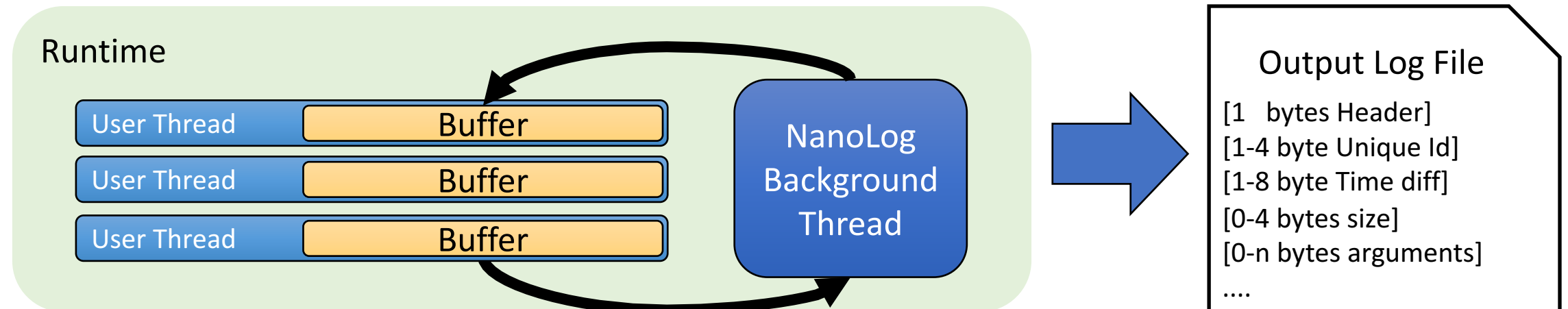
Fast Runtime Architecture

- **Isolate the Threads**

- Use per-thread buffers to lower synchronization
- Don't notify the background thread; let it poll for data

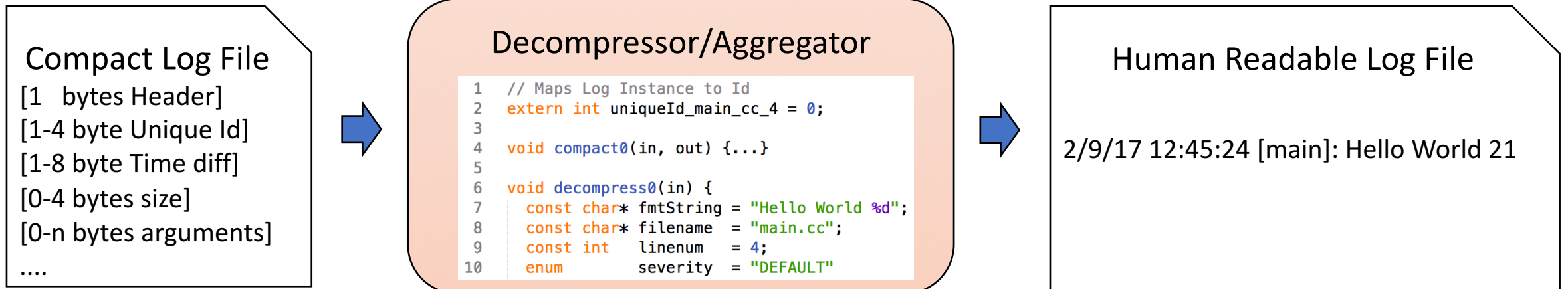
- **Minimize Output Cost**

- Caller pushes data *uncompressed* to save on compute
- IO Thread needs to save on both IO and compute times.
 - Use only rudimentary compaction (deltas + smallest byte representations)



Decompressor/Aggregator

- **Offline process to decompress log**
 - Recombines the static + dynamic data to produce a human-readable file
- **Future Work**
 - Query/Aggregate in compacted format



Benchmarks

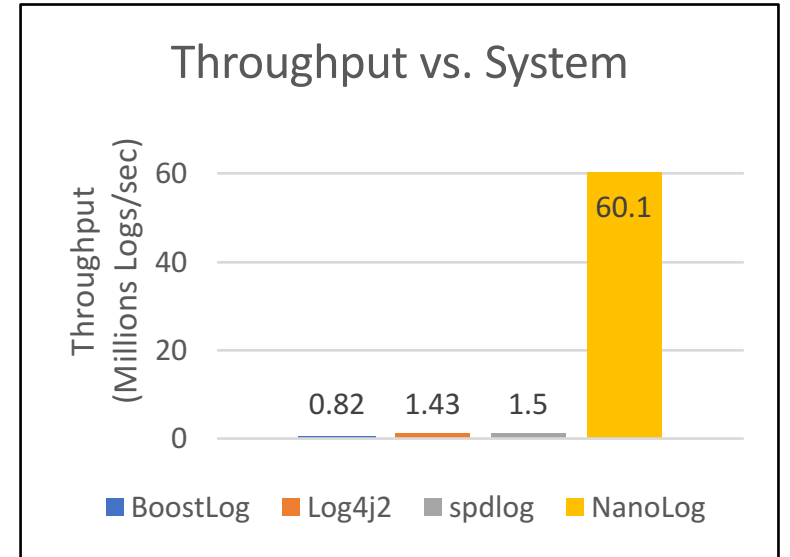
- **System Setup**

- Processor: Quad-Core Intel Xeon X3470 @ 2.93GHz
- Memory: 24GB DDR3 @ 1333Mhz
- Disk: 120GB Crucial M4 over SATAII (~250MB/s)

- **Test Setup**

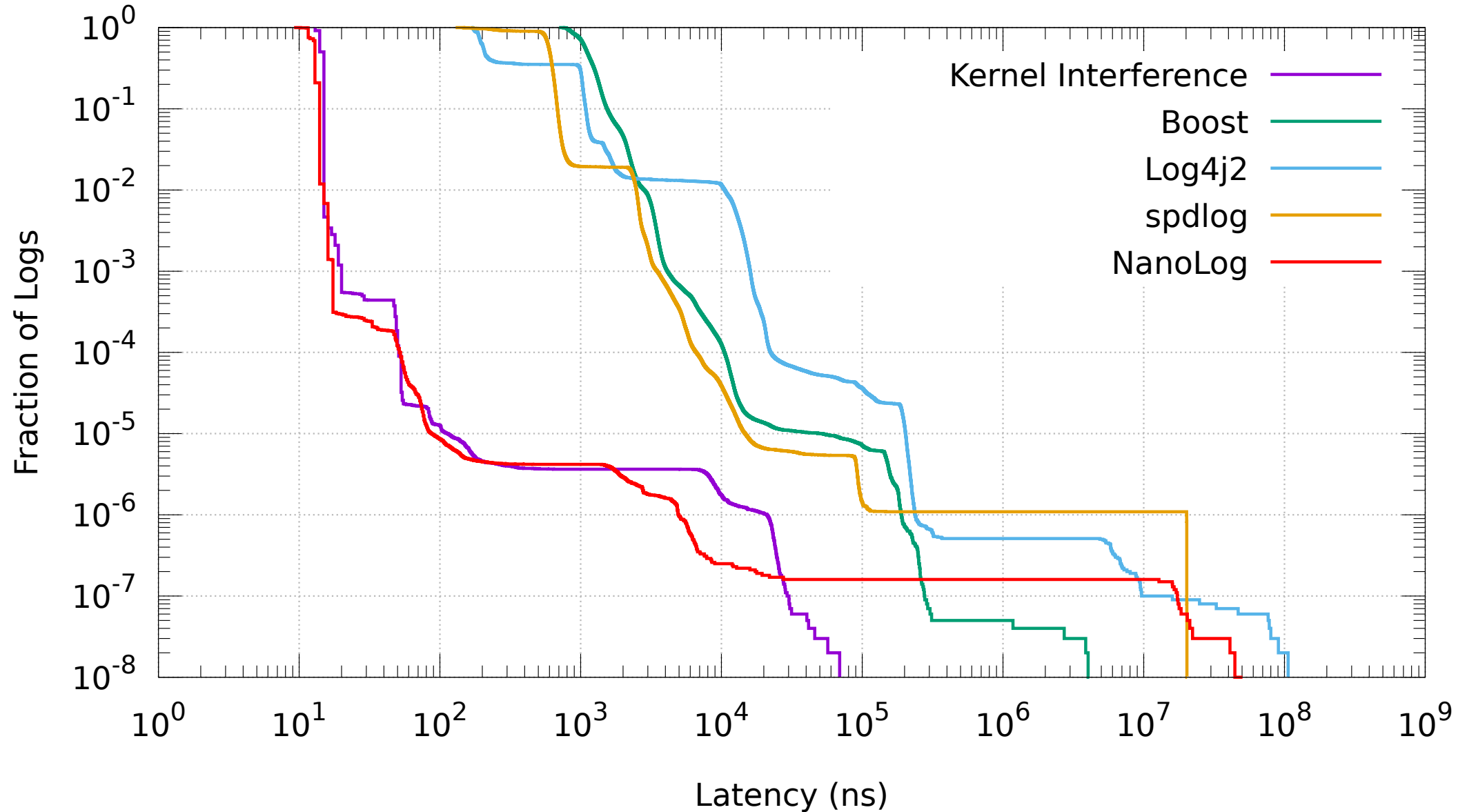
- 100M iterations of log messages, back to back
- Log Message: “{time} {severity}: {56-byte message}”

- **Overall Results**

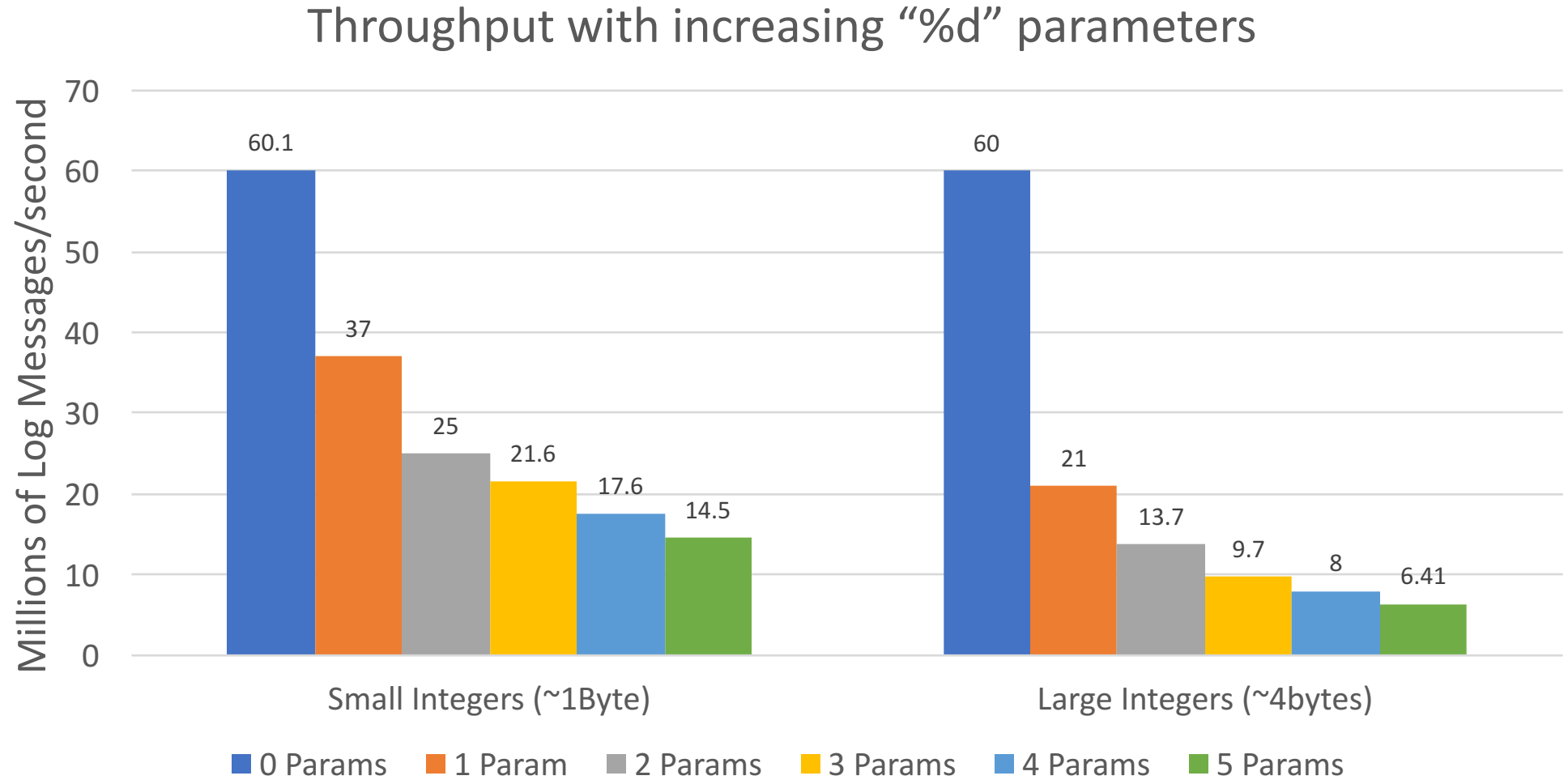


Zero Arguments	Boost v1.55	Log4j2	Spdlog	NanoLog
Throughput (Log/s)	0.82M	1.43M	1.50M	60.1M
Average Latency (ns)	1110 ns	697ns	668 ns	16.5ns

Tail Latencies



Increasing Parameters



Limitations/Future Work

- **Better Compression?**

- Is there a better way to compact the output, but in a performant way?

- **Fully featured decompressor/aggregator**

- Operating on the compact representation is more efficient.
- Iterating over a compact log message takes about 100ns vs. 1.3 μ s to output

- **Resource Utilization**

- Currently the system requires 1MB per user thread, a full core to compact, and the full bandwidth of a SATA SSD to maintain low latency. How does this change with new hardware?

NanoLog System Summary

- **Compile-Time Preprocessor**

- Extract static information from log messages at compile time
 - File name, line #, function name, etc
- Catalogs static info and assigns a unique ID to each log statement
- Code Injection to record only an identifier + parameter arguments

- **Runtime Library**

- Producer/Consumer Log output
- Simple compaction (taking deltas/compacting integers)

- **Offline Decompressor/Aggregator**

- Recombine static information for human consumption (if necessary)
- Offline Search/Grep/Aggregate in compressed format

Questions