OpenTracing

ATS Summit Fall 2021
Masakazu Kitajo <maskit@apache.org>

action.continuation->handleEvent(EVENT_HOST_DB_LOOKUP, nullptr);

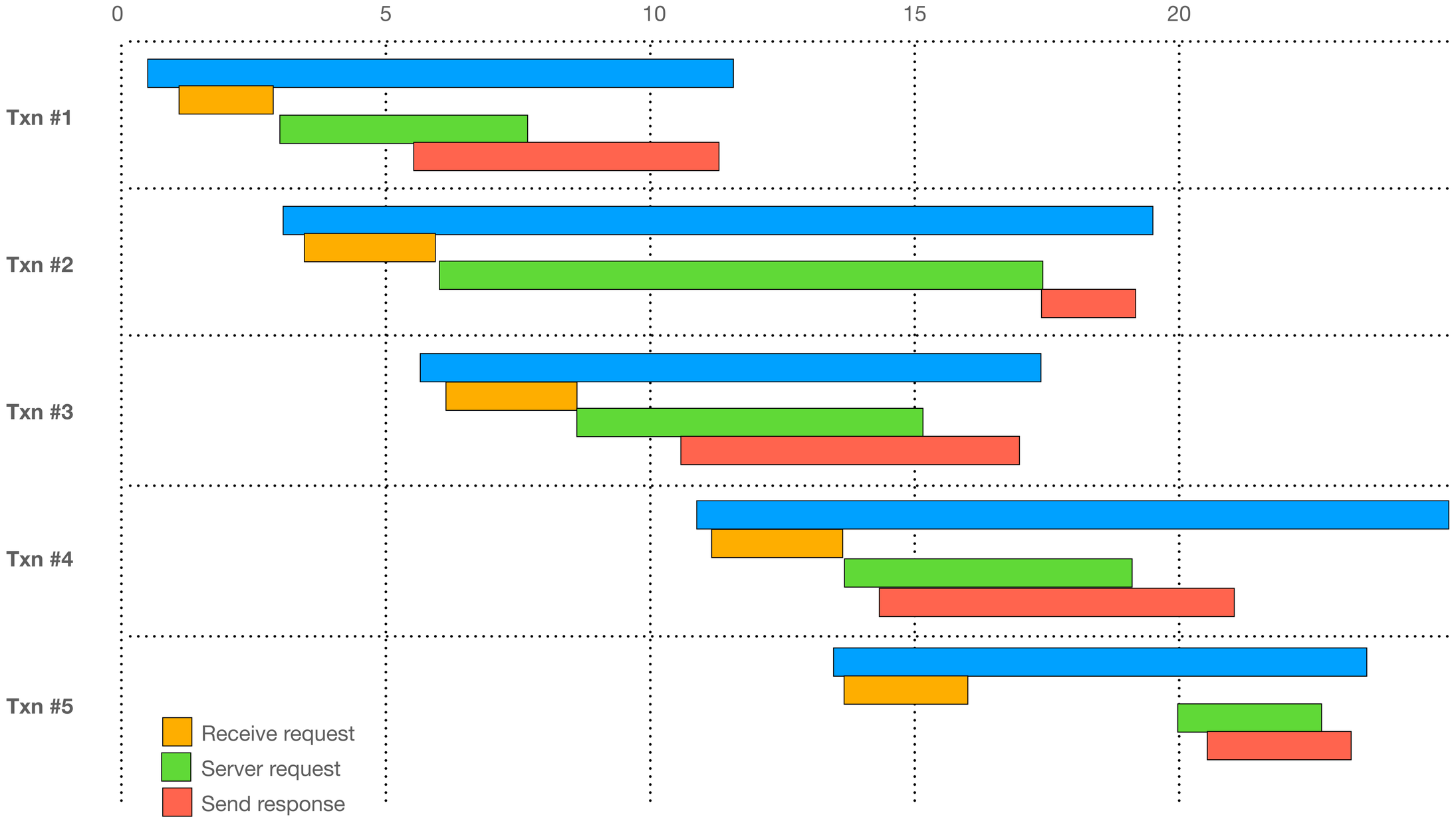
ua_txn->transaction_done();

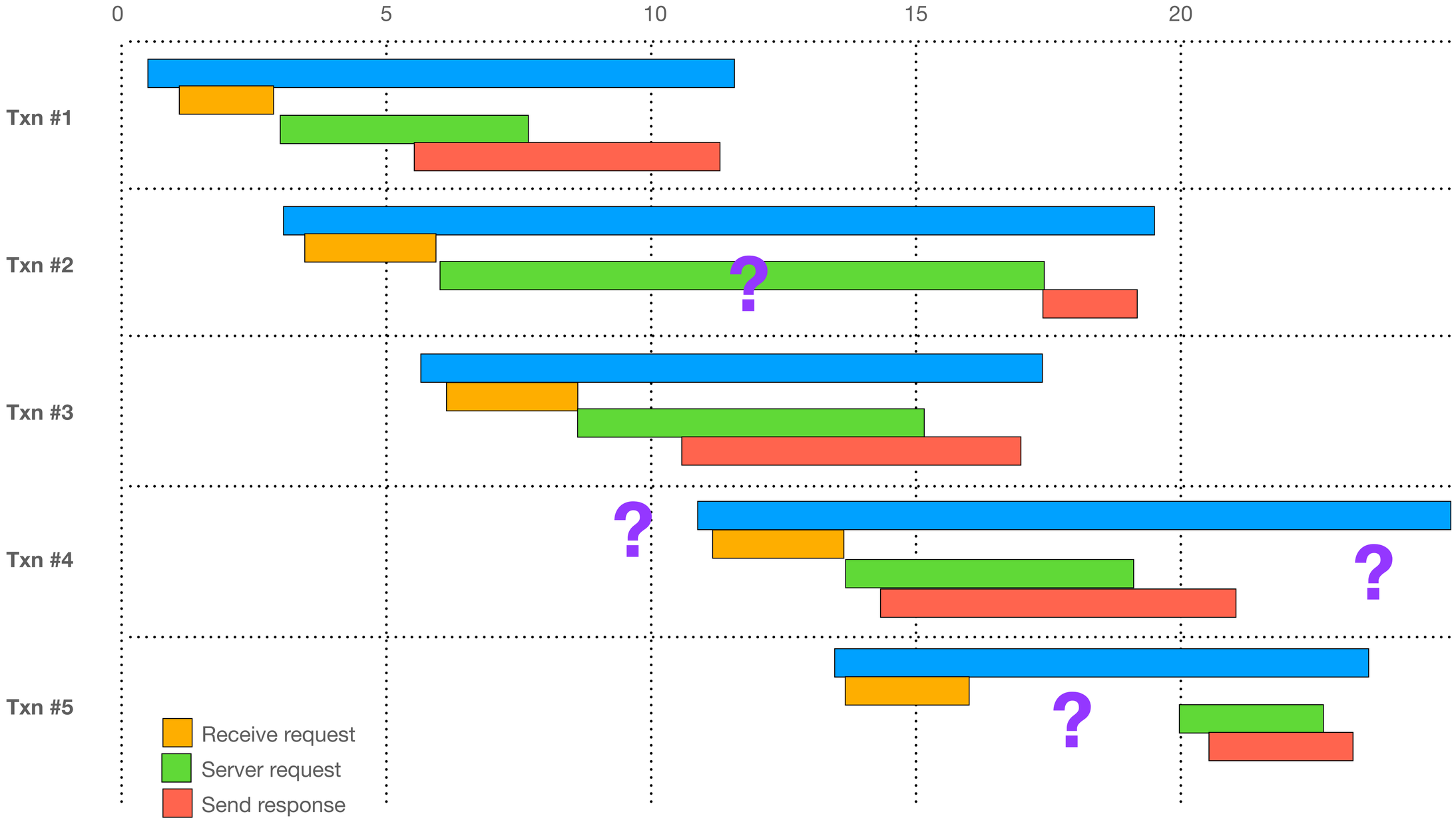
eventProcessor.schedule_imm(this, ET_NET);

Do you really know what happens inside ATS?

MUTEX_TRY_LOCK(trylock1, m_write_vio.mutex, this_ethread());

Event::schedule_at(ink_hrtime atimeout_at, int acallback_event)

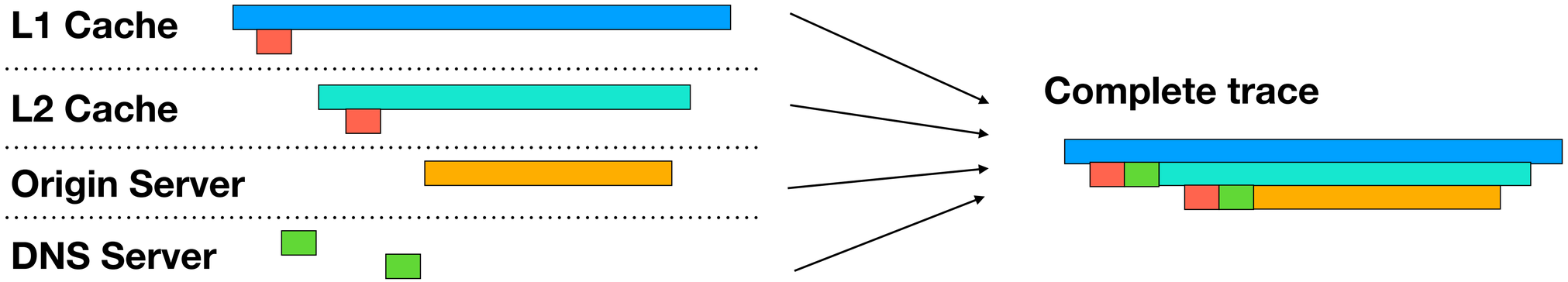

0
5
10
15
20
Txn #1
Txn #2
Txn #3
Txn #4
Txn #5
Receive request
Server request
Send response


0
5
10
15
20
Txn #1
Txn #2
Txn #3
Txn #4
Txn #5
?
?
?
?
Receive request
Server request
Send response

OpenTracing

“Vendor-neutral APIs and instrumentation for distributed tracing”

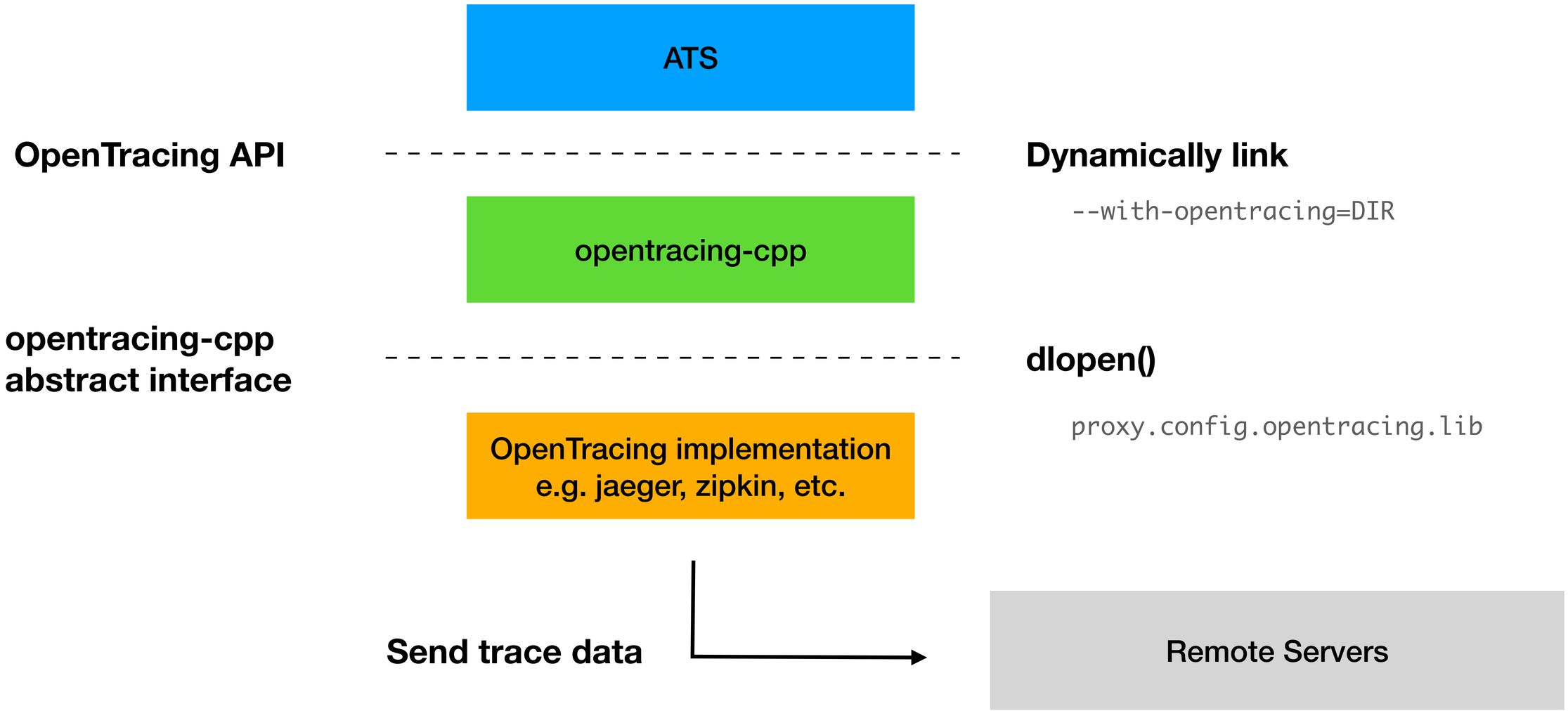
- Common use cases
  - Tracing functions - For application code and library code
  - Tracing server endpoints - For distributed systems



PR #8495: Add support for OpenTracing

- Build option to use OpenTracing
- Settings to enable tracing
- Code to initialize libraries
- Generic interface for tracing
- Code to trace transactions

Overview


ATS
OpenTracing API
Dynamically link
--with-opentracing=DIR
opentracing-cpp
opentracing-cpp
abstract interface
dlopen()
proxy.config.opentracing.lib
OpenTracing implementation
e.g. jaeger, zipkin, etc.
Send trace data
Remote Servers

Generic interface

Tracing.h

```
class Tracing
{
public:
  void enable(int value = 1);
  void disable();

  bool is_enabled();

  TRACER *make_tracer(const char *name);
  void delete_tracer(TRACER *tracer);

private:
  std::atomic<int> _enabled;
};

#define TRACE_TAG(out, category, message)
#define TRACE_LOG(out, category, message)
```

Tracing_opentracing.h

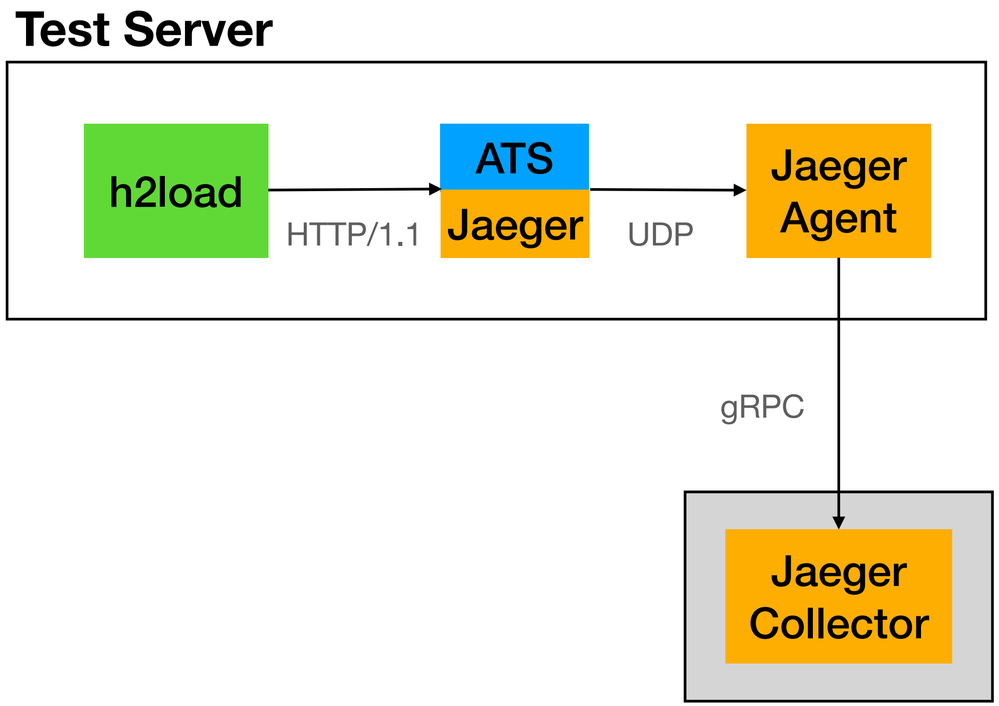
```
using TRACER = opentracing::Span;

inline TRACER *
tracing_new(const char *name)
{
  opentracing::Tracer *tracer =
    static_cast<opentracing::Tracer *>(
      ink_thread_getspecific(thread_specific_tracer_key)
    );
  if (tracer == nullptr) {
  ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
  }
  auto span = tracer->StartSpan(name);
  return span.release();
}
```

Where you use trace

```
this->_tracer = tracing->make_tracer("ProxyTransaction");
TRACE_TAG(this->tracer(), "txn_id", this->get_transaction_id());
TRACE_LOG(ua_txn->tracer(), TRACE_CAT_MILESTONE, “SERVER_READ_HEADER_DONE”);
```

Performance

- Non-TLS HTTP/1.1
- 8KB cached object



	Sampling	Req/sec	Percentage
Tracing OFF	N/A	73808.31	100%
Tracing ON	100%	62339.33	84%
	10%	69519.02	94%

h2load --h1 -n 3000000 -c 1000 -m 1 -t 10 http://127.0.0.1/8k

To-Do List

- Context propagation
- Selective tracing
- Trace level / tag