Minimal Logging

Koen Poppe — Meeting C++ 2023 — November 12th

Meeting C++ 2023

Koen Poppe

‘20 Vandewiele Group
Textile Machine Manufacturer
Embedded Software

‘15 Nikon Metrology

‘10 KULeuven
Mathematical Engineering

‘05 First contact with C++

Logging

Logging

01 - Qt

- Filter @ start
- Formatting @ start

Logging

02 - std::iostream

```
#define LOG(message)                                    \
    std::cout << __FILE__ << "(" << __LINE__ << ") "    \
              << __FUNCTION__ << ": " << message << "\n"
...
LOG("Hello " << std::setprecision(3) << std::numbers::pi);
```

Preprocessor

Formatting

Streaming operator

```
.../02-iostreamMain.cpp (55) example: Hello 3.14
```

File

Line

Function

Logging

03 - std::source_location (C++20)

Compiler magic!

```
void log(const std::string_view message,
         const std::source_location location = std::source_location::current())
{
    std::cout << location.file_name() << "(" << location.line() << ") "
              << location.function_name() << ": " << message << "\n";
}
...
log("Hello");
```

```
03-SourceLocationMain.cpp(33) example: Hello
```

File name

Line

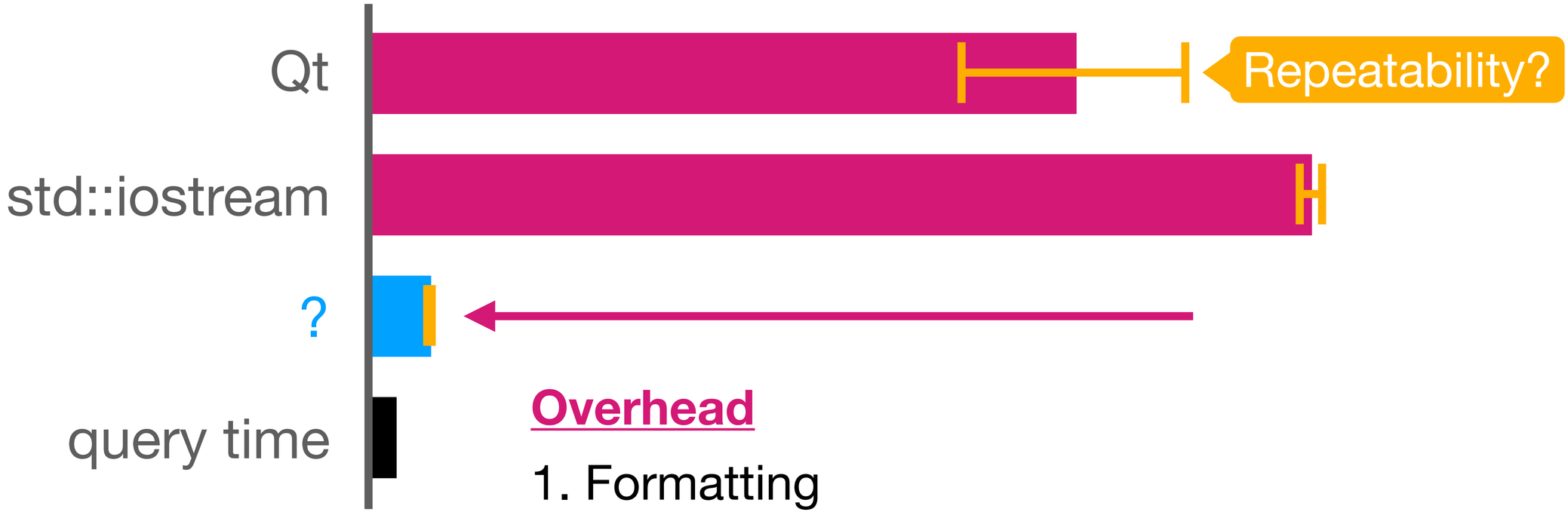
Function

Logging

A bool, an integer and a float

WWDC 2016 : Unified Logging and Activity Tracing:

“We all spend too much code converting information to strings to log”



Overhead

1. Formatting
2. Verbosity
3. Memory

Minimal Logging

Live coding

«Intermezzo»

- Ensure correctness?
  - Unit Tests
- Compile & run
  - With missing features

```
struct Logger {};
```

(Left out for clarity)

Substitution Failure Is Not An Error (SFINAE)

Template

```
void timeInfoImpl(auto logger)
{
    if constexpr (requires { logger.trace(); })
    {
        logger.trace();
        // … verify results
    }
    else
    {
        QFAIL(“trace not available”);
    }
}
void timeInfo()
{
    timeInfoImpl(Logger{})
}
```

… if this compiles (C++20)

Unit Tests

✗	timeInfo
✗	resolveFunctionSingle
✗	resolveFunctionsNested
✗	traceWithBool
✗	traceWithInt
✗	traceWithUnsignedInt
✗	traceWithFloat
✗	traceMulti

Components

What is logged?

1. When?
2. Where?
3. What?

Unit Tests

❌	timeInfo
❌	resolveFunctionSingle
❌	resolveFunctionsNested
❌	traceWithBool
❌	traceWithInt
❌	traceWithUnsignedInt
❌	traceWithFloat
❌	traceMulti

1 When?

Point in time

Unit Tests

❌	timeInfo
❌	resolveFunctionSingle
❌	resolveFunctionsNested
❌	traceWithBool
❌	traceWithInt
❌	traceWithUnsignedInt
❌	traceWithFloat
❌	traceMulti

Calendar & timezone

2023-11-12 13:20:27.123 …

24 byte

ms resolution

```
#include <chrono>

using Clock = std::chrono::high_resolution_clock;
using TimeUnit = std::chrono::nanoseconds;
TimeUnit::rep now()
{
    return duration_cast<TimeUnit>(
        Clock::now().time_since_epoch()).count();
}

struct Record
{
    TimeUnit::rep m_time;
};
```

8 byte

No formatting

Overhead: **Formatting** / **Verbosity** / Memory

1 When?

Logger implementation

Unit Tests	
❌	timeInfo
❌	resolveFunctionSingle
❌	resolveFunctionsNested
❌	traceWithBool
❌	traceWithInt
❌	traceWithUnsignedInt
❌	traceWithFloat
❌	traceMulti

```
template <std::size_t sizeLog2>
class Logger
{
public:
    void trace()
    {
        m_circularBuffer.append(Record{now()});
    }
…
private:
    CircularBuffer<sizeLog2> m_circularBuffer{};
};
```

```
template <std::size_t sizeLog2>
class CircularBuffer
{
public:
    template <typename T>
    void append(const T &t);
private:
    std::array<char, 1<<sizeLog2>
        m_buffer{};
};
```

std::memcpy
no formatting

Fixed capacity
(no allocations)

Power of 2
(mod → and)

Overhead: Formatting / Verbosity / **Memory**

1 When?

Unit test

Unit Tests	
✅	timeInfo
❌	resolveFunctionSingle
❌	resolveFunctionsNested
❌	traceWithBool
❌	traceWithInt
❌	traceWithUnsignedInt
❌	traceWithFloat
❌	traceMulti

Online

Offline

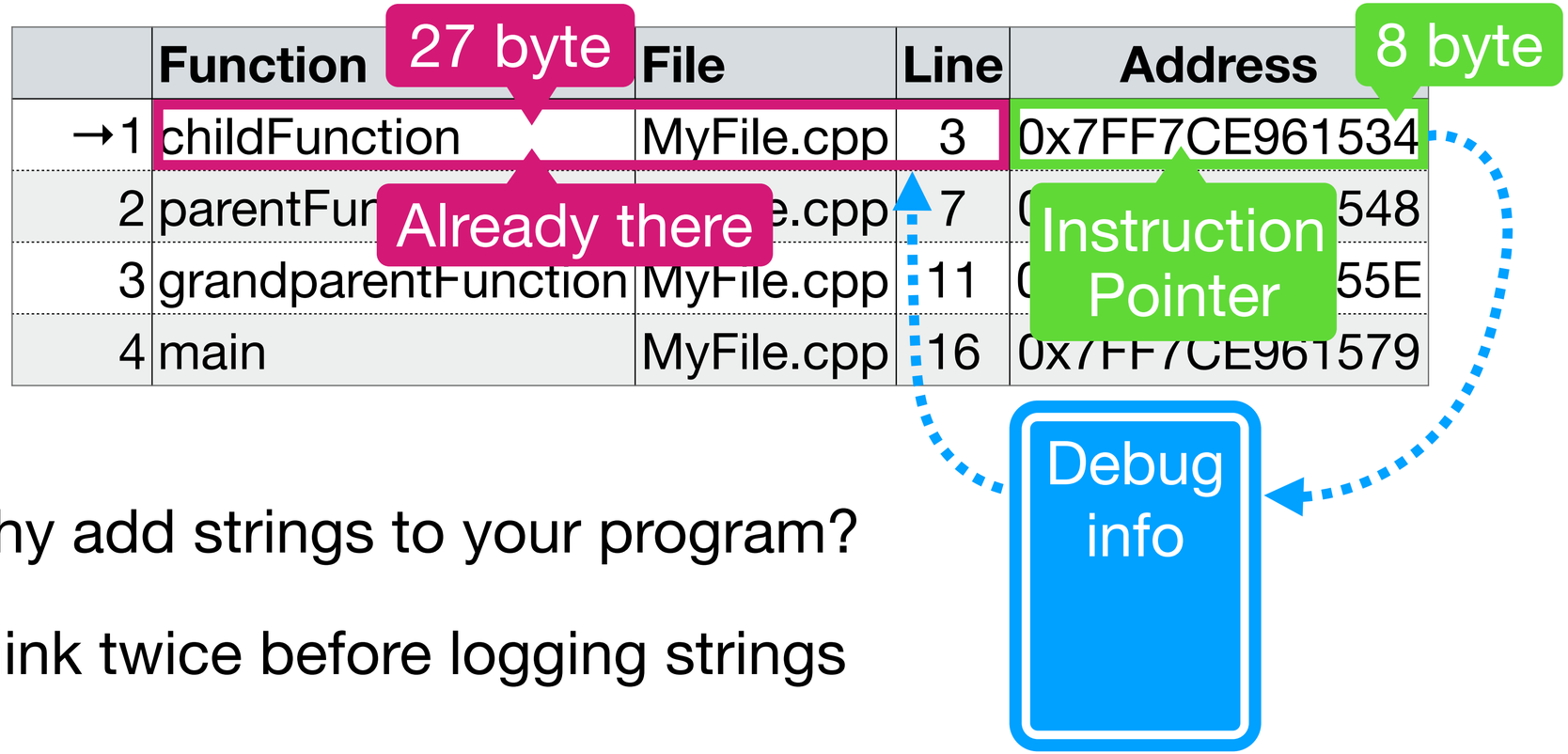
```
void timeInfo()
{
    // Test program
    Logger<8> logger;
    logger.trace();
    sleep(1000);
    logger.trace();

    // Verification
    const LogModel model(logger.data());
    const auto &records = model.records();
    QCOMPARE(records.size(), 2u);
    auto duration =
        records.at(1).time - records.at(0).time;
    …
```

2 Where?

The debugger knows…

	Function	File	Line	Address
→1	childFunction	MyFile.cpp	3	0x7FF7CE961534
2	parentFun	e.cpp	7	548
3	grandparentFunction	MyFile.cpp	11	55E
4	main	MyFile.cpp	16	0x7FF7CE961579



- Why add strings to your program?
- Think twice before logging strings

Overhead: Formatting / **Verbosity** / Memory

Crash dump workflow

«Intermezzo»

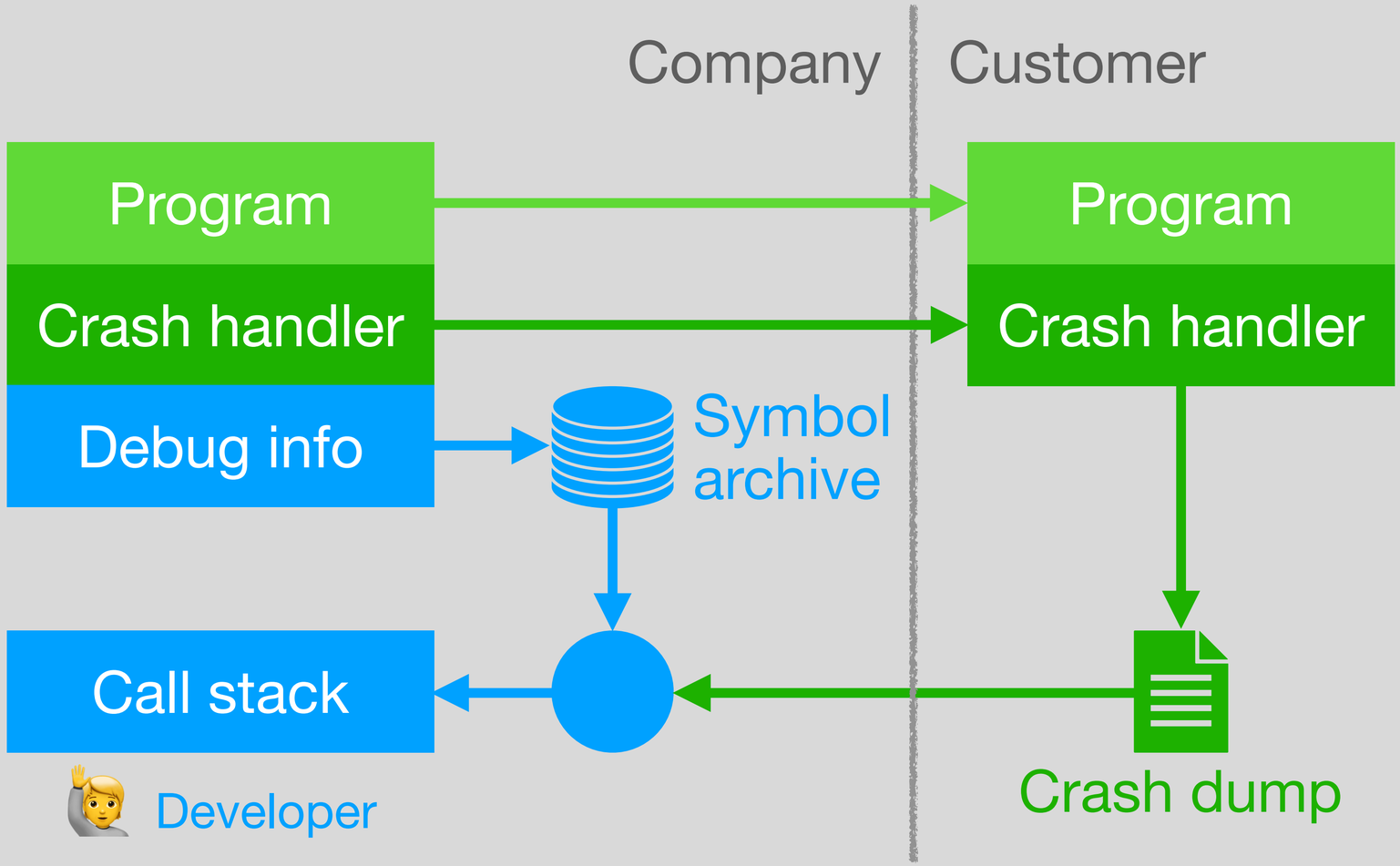
Debug info…

- takes up space
- intellectual risk

```
void writeProprietaryFileFormat(...)
```

- security risk

```
bool isSoftwareLicensed(...)
```



Based on Getting started with Breakpad

2 Where?

Crash dump workflow

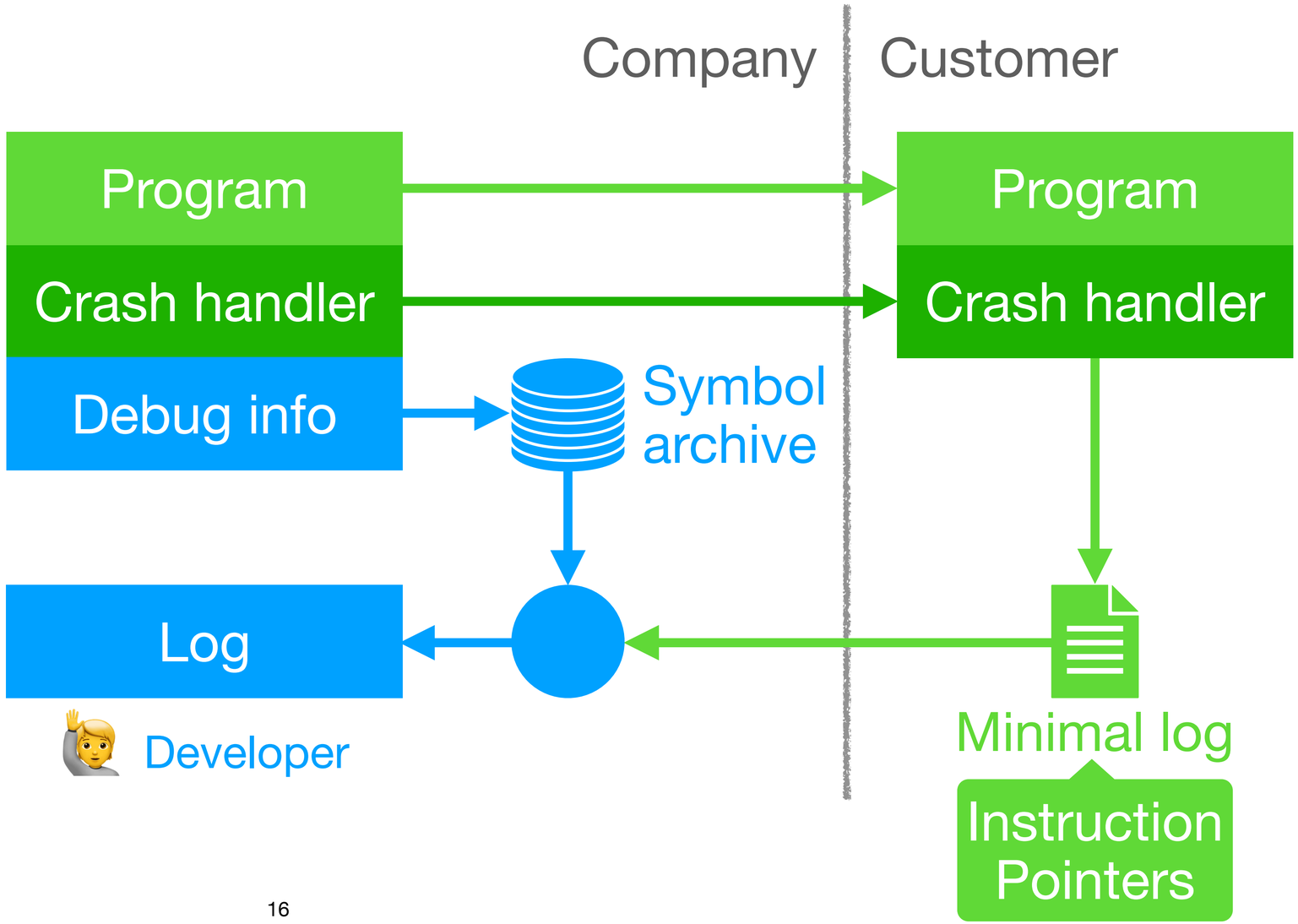
Minimal Log

- ~~takes up space~~
- ~~intellectual risk~~

> “Writing secret file to …”

- ~~security risk~~

> Checksum should be 42



Unit Tests	
✅	timeInfo
❌	resolveFunctionSingle
❌	resolveFunctionsNested
❌	traceWithBool
❌	traceWithInt
❌	traceWithUnsignedInt
❌	traceWithFloat
❌	traceMulti

2 Where?

Retrieve the Instruction Pointer

```
uintptr_t instructionpointer() __attribute__((always_inline))
{
    uintptr_t ip;
#if “ARM”
    asm volatile("ADR %0, ." : "=r"(ip));
#elif “x86”
    asm volatile("lea (%%rip),%0" : "=r"(ip));
#else
    // ...
#endif
    return ip;
}
```

Evaluate at caller

Matches architecture

Readily available

- ~ compiler support std::source_location::current
- __builtin_return_address compiler intrinsic is wrong
  - where from vs to

Unit Tests	
✅	timeInfo
✅	resolveFunctionSingle
❌	resolveFunctionsNested
❌	traceWithBool
❌	traceWithInt
❌	traceWithUnsignedInt
❌	traceWithFloat
❌	traceMulti

2 Where?

Extending the Record

```
struct Record
{
    TimeUnit::rep m_time;
    uintptr_t m_ip;
};
```

```
template <std::size_t sizeLog2>
class Logger
{
public:
    void trace() __attribute__((always_inline))
    {
        m_circularBuffer.append(
            Record{now(), instructionPointer()});
    }
...
```

At caller site

Overhead: Formatting / **Verbosity** / Memory

2 Where?

Unit test (1)

Unit Tests	
✅	timeInfo
✅	resolveFunctionSingle
❌	resolveFunctionsNested
❌	traceWithBool
❌	traceWithInt
❌	traceWithUnsignedInt
❌	traceWithFloat
❌	traceMulti

Online

Offline

```
void resolveFunctionSingle()
{
    // Test program
    Logger<8> logger;
    logger.trace();

    // Verification
    const std::string data = logger.data();
    QCOMPARE(data.size(),
        sizeof(Logger::TimeUnit::rep) + sizeof(uintptr_t));
    const LogModel model(data, s_pathToSymbols);
    const auto &records = model.records();
    QCOMPARE(records.size(), 1u);
    QCONTAINS(records.at(0).functionName(),
                        “resolveFunctionSingle”);
```

Instruction Pointer

Symbol archive

2 Where?

Unit test (2)

Unit Tests	
✅	timeInfo
✅	resolveFunctionSingle
✅	resolveFunctionsNested
❌	traceWithBool
❌	traceWithInt
❌	traceWithUnsignedInt
❌	traceWithFloat
❌	traceMulti

```
struct Nested
{
    Logger<8> m_logger;
    void first() __attribute__((noinline))
    {
        m_logger.trace();
        second();
        m_logger.trace();
    }
    void second() __attribute__((noinline))
    {
        m_logger.trace();
    }
};
```

3x Clock + Instruction Pointer

```
Nested::first
Nested::second
Nested::first
```

3 What?

Extending the Record

Unit Tests

✅	timeInfo
✅	resolveFunctionSingle
✅	resolveFunctionsNested
❌	traceWithBool
❌	traceWithInt
❌	traceWithUnsignedInt
❌	traceWithFloat
❌	traceMulti

```
struct Record
{
   TimeUnit::rep m_time;
   uintptr_t m_ip;
};
template <typename T>
struct RecordT : Record
{
    T t;
};
```

Primitive data types

```
void trace() __attribute__((always_inline))
{
    m_circularBuffer.append(
        Record{now(), instructionPointer()});
}
template <typename T>
void trace(T t) __attribute__((always_inline))
{
    m_circularBuffer.append(
        RecordT{now(), instructionPointer(), t});
}
```

Binary data
No precision loss

Issues: **Formatting** / **Verbosity** / Memory

3 What?

Identifying type

RecordT
…+1 bytes
…+8 bytes
…+8 bytes
…+4 bytes

```
traceInner<bool>
traceInner<int>
traceInner<unsigned>
traceInner<float>
```

Unit Tests

✅	timeInfo
✅	resolveFunctionSingle
✅	resolveFunctionsNested
✅	traceWithBool
✅	traceWithInt
✅	traceWithUnsignedInt
✅	traceWithFloat
❌	traceMulti

```
template <typename T>
void __attribute__((always_inline)) trace(T t)
{
    m_circularBuffer.append(Record<T>{
        now(), instructionPointer(), t});
}
```

```
template <typename T>
void trace(T t) __attribute__((always_inline))
{
    traceInner(instructionPointer(), t);
}
template <typename T>
void traceInner(uintptr_t traceCallSite, T t)
    __attribute__((noinline, used))
{
    m_circularBuffer.append(RecordT<T>{
        now(), traceCallSite, instructionPointer(), t});
}
```

call site trace(…)

Do not re-use

traceInner<T>(…) instance

3 What?

Arguments

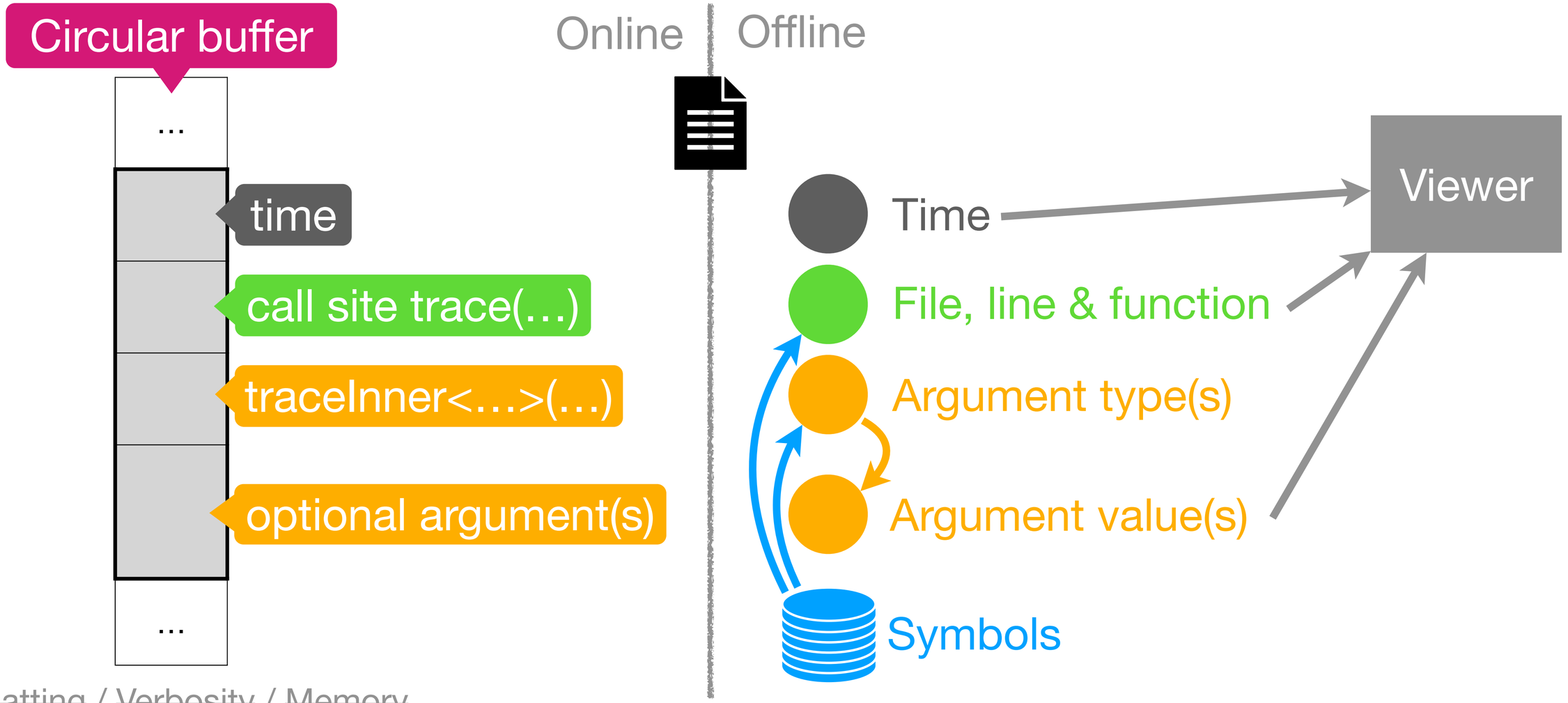
Unit Tests	
✅	timeInfo
✅	resolveFunctionSingle
✅	resolveFunctionsNested
✅	traceWithBool
✅	traceWithInt
✅	traceWithUnsignedInt
✅	traceWithFloat
✅	traceMulti

RecordT ~ packed std::tuple

```
template <typename... Ts>
void trace(Ts... ts) __attribute__((always_inline))
{
    traceInner(instructionPointer(), ts...);
}
template <typename... Ts>
void traceInner(uintptr_t traceCallSite, Ts... ts)
    __attribute__((noinline, used))
{
    m_circularBuffer.append(RecordT<Ts...>{
        now(), traceCallSite, instructionPointer(), ts...});
}
```

Issues: Formatting / **Verbosity** / Memory

Minimal logging summary



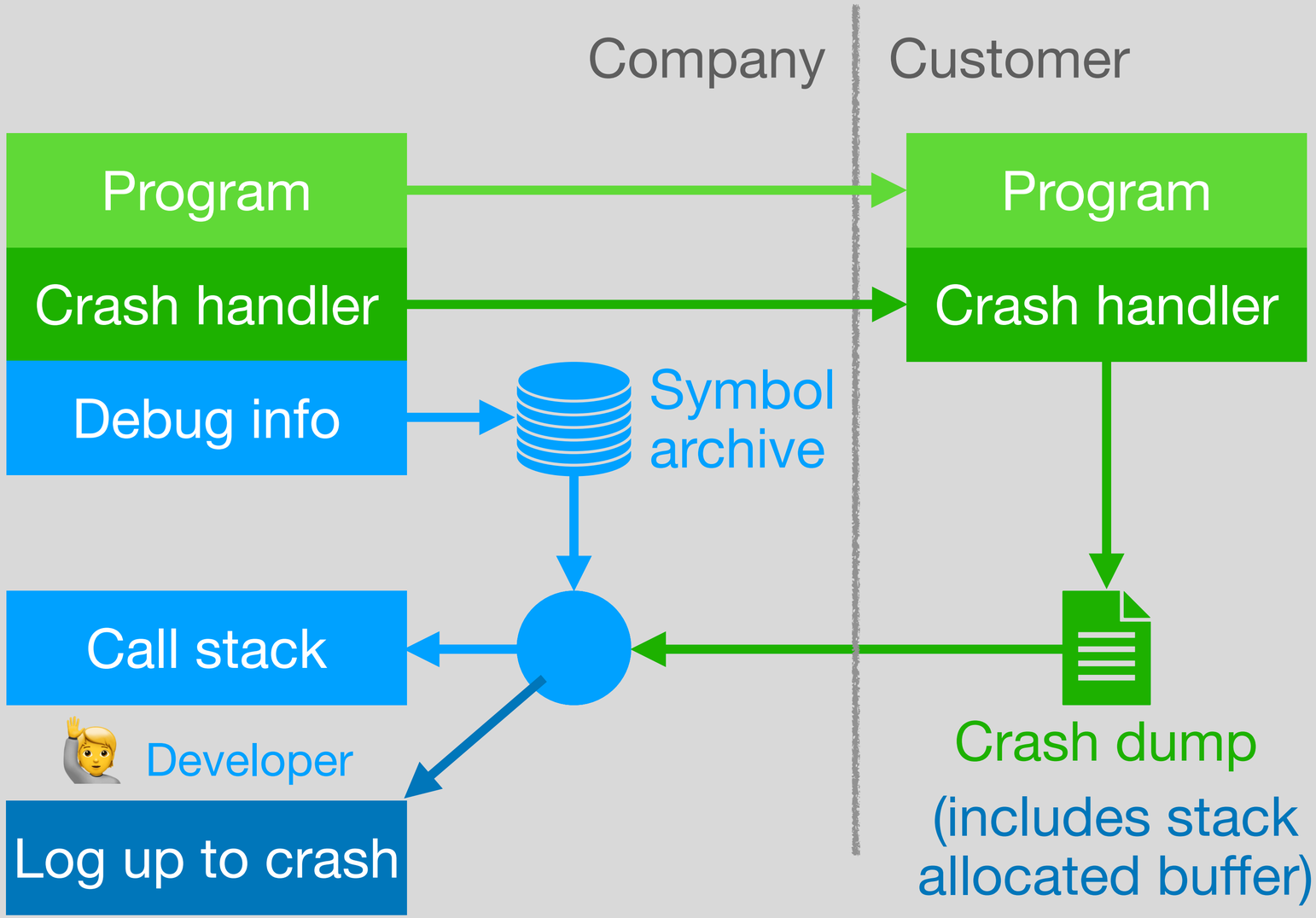
~~Formatting~~ / ~~Verbosity~~ / ~~Memory~~

Bonus slides

- Crash dumps
- Literals
- Shared Libraries
- Versioning

Crash dump bonus

What is in it?



Return

Reducing log size

Literals

```
logger.trace("String literal", 42, true, 3.14f);
```

Idea: overload RecordT for literals

- in the code, no reason to log
- tooling can do formatting and add literals

Return

Shared libraries

Extra bookkeeping

- symbols ~ relative to load address
- use same approach as crash dump
  - store for each shared library
- model takes care of this when resolving

Return

Software versioning

Identify log

- use same approach as crash dump
  - based on *build-id*
    - store for each shared library

Return