Core C++ 2024

Building Effective Embedded Systems: Architectural Best Practices

Gili Kamma

Today’s spotlight:

Exploring best practices in embedded systems, with a focus on operating systems

Today’s spotlight:

Exploring best practices in embedded systems, with a focus on operating systems

Today’s take away:

Practical tips for building better software, applicable not only to embedded systems but also to software in general

HELLO!

I am Gili Kamma

20 years in the industry

I love to improve things and solve problems

Team leader @ Priority-software

Agenda

- Operating Systems
- Threads
- Layer Separation
- Network Problems
- Message Structure
- Simulators
- Logs
- Monitoring

Agenda

- ⊡ Operating Systems
- ⊡ Threads
- ⊡ Layer Separation
- ⊡ Network Problems
- ⊡ Message Structure
- ⊡ Simulators
- ⊡ Logs
- ⊡ Monitoring

Operating Systems
To Be or Not To Be

Hard/Soft Real Time Requirements

Hard Real Time

- Timing constraints are extremely strict
- A guaranteed response time
- Microseconds
- Flight control system

Hard/Soft Real Time Requirements

Hard Real Time

- Timing constraints are extremely strict
- A guaranteed response time
- Microseconds
- Flight control system

Soft Real Time

- Flexible Timing (5-10 milliseconds)
- ! A guaranteed response time
- Milliseconds
- Home automation

How to decide if we need
an operating system or not?

What level of time precision does our system require?

What level of time precision does our system require?

- 10 Microseconds?

What level of time precision does our system require?

- 10 Microseconds?
- 10 Milliseconds?

What level of time precision does our system require?

- ⊡ 10 Microseconds?
- ⊡ 10 Milliseconds?
- ⊡ 100 Milliseconds?

What level of time precision does our system require?

- ⊡ 10 Microseconds?
- ⊡ 10 Milliseconds?
- ⊡ 100 Milliseconds?

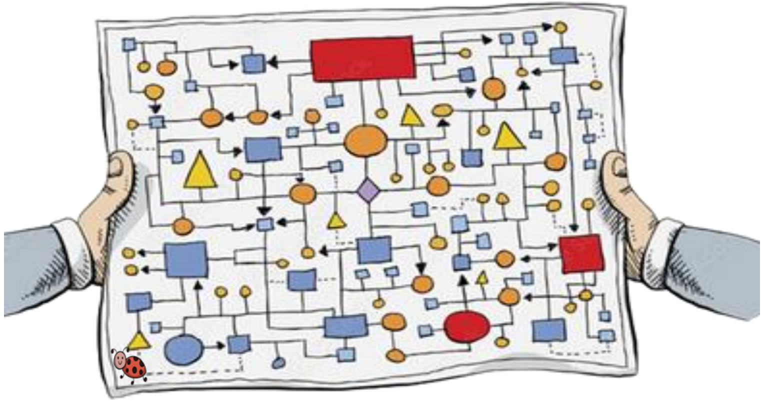
👍 Less then 5 milliseconds – Don’t use an operating system*

*not impossible but challenging

How complicated our software is going to be?

How complicated our software is going to be?

👍 The more interfaces and processes we have, we would like to have an operating system



	Soft Real Time	Hard Real Time
Simple System		
Complicated System		

	Soft Real Time	Hard Real Time
Simple System		None
Complicated System		

	Soft Real Time	Hard Real Time
Simple System		None
Complicated System	Operating system	

	Soft Real Time	Hard Real Time
Simple System	Don’t care	None
Complicated System	Operating system	

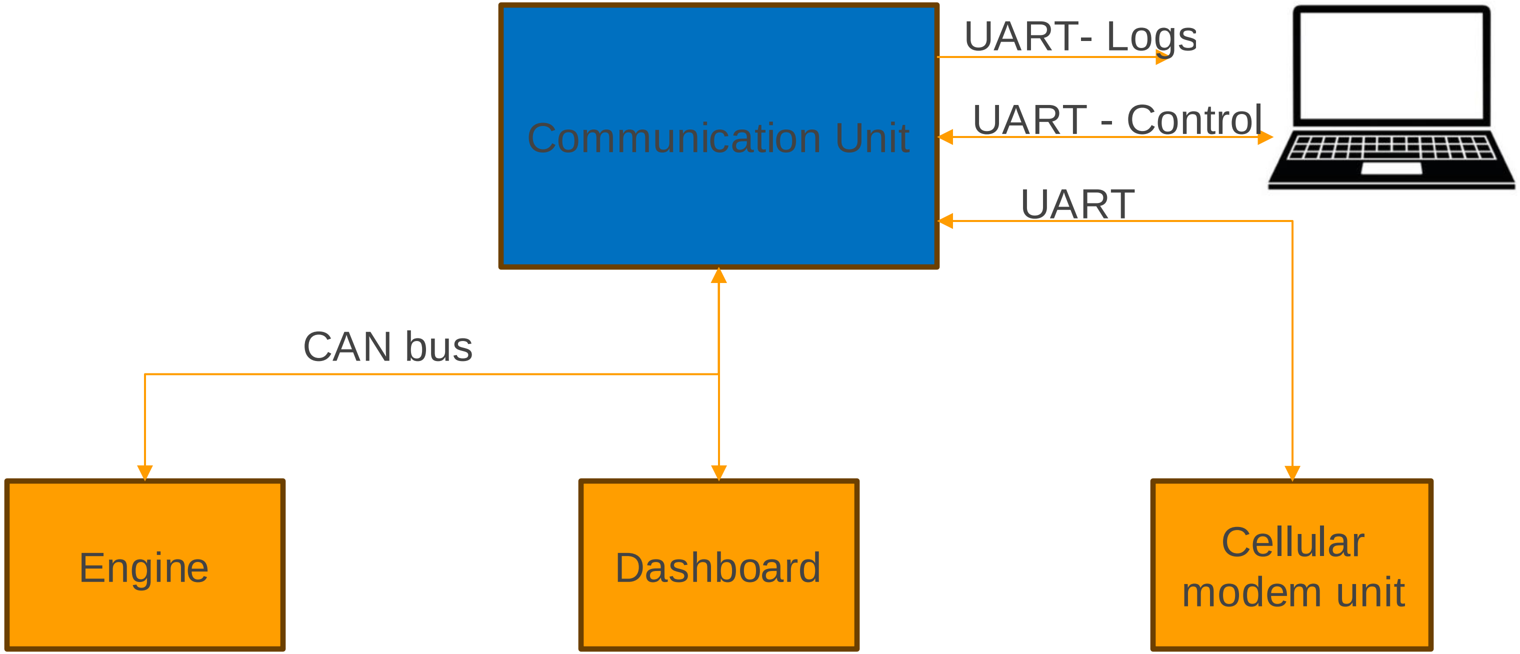
	Soft Real Time	Hard Real Time
Simple System	Don’t care	None
Complicated System	Operating system	?

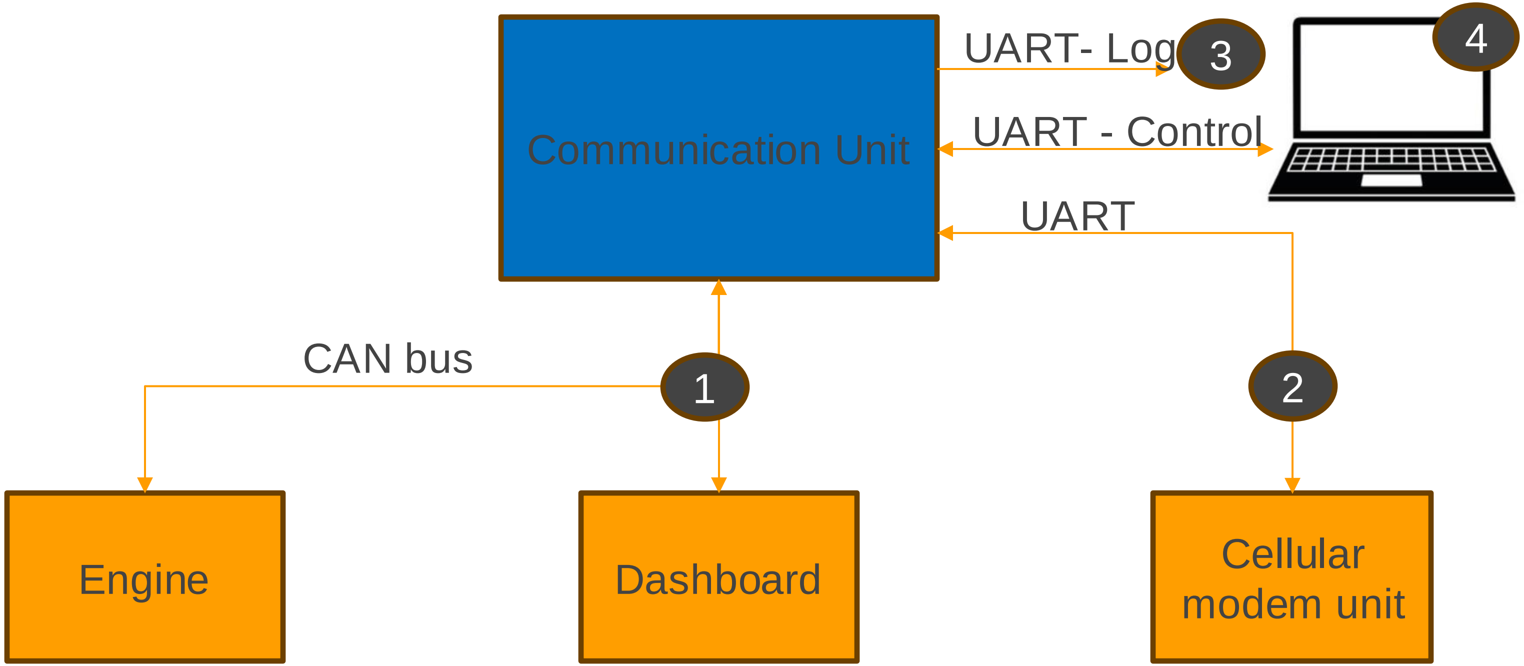
	Soft Real Time	**Hard Real Time**
Simple System	Don’t care	None
Complicated System	Operating system	FPGA/Chip + CPU with operating system

Let’s review a system
and decide if an operating system
is needed

Operating System

Operating System



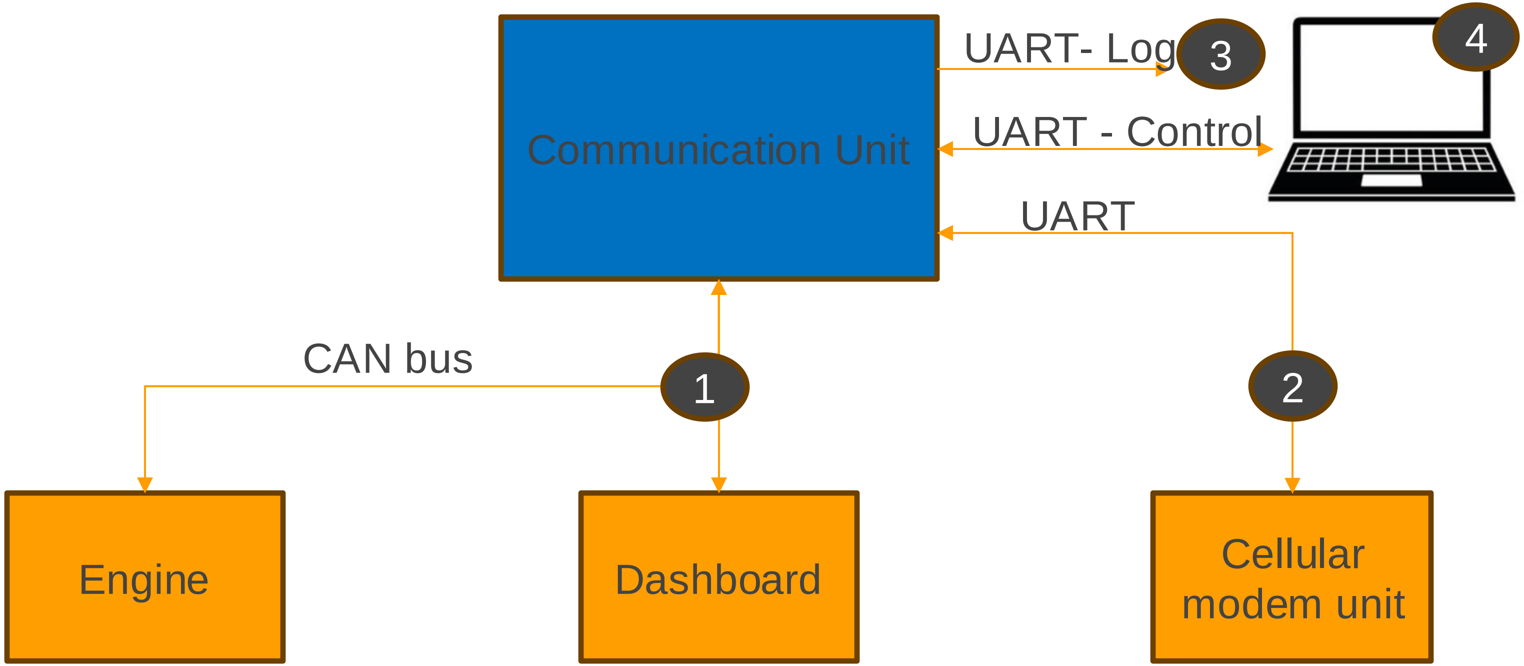


4 interfaces → complicated

What level of time precision does the system require?

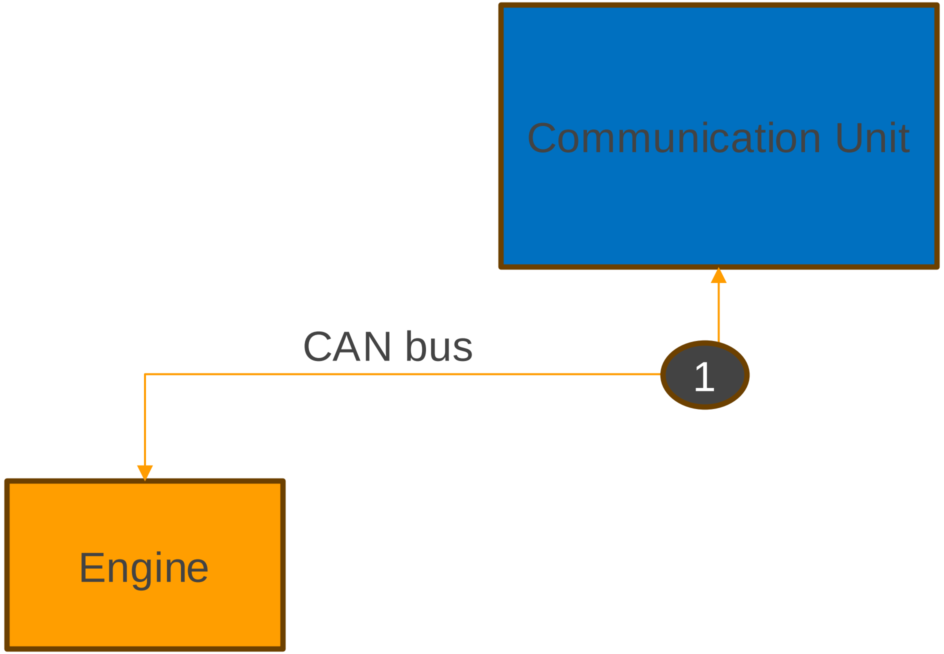
What level of time precision does the system require?

~100 milliseconds



operating system

Operating System



operating system

“

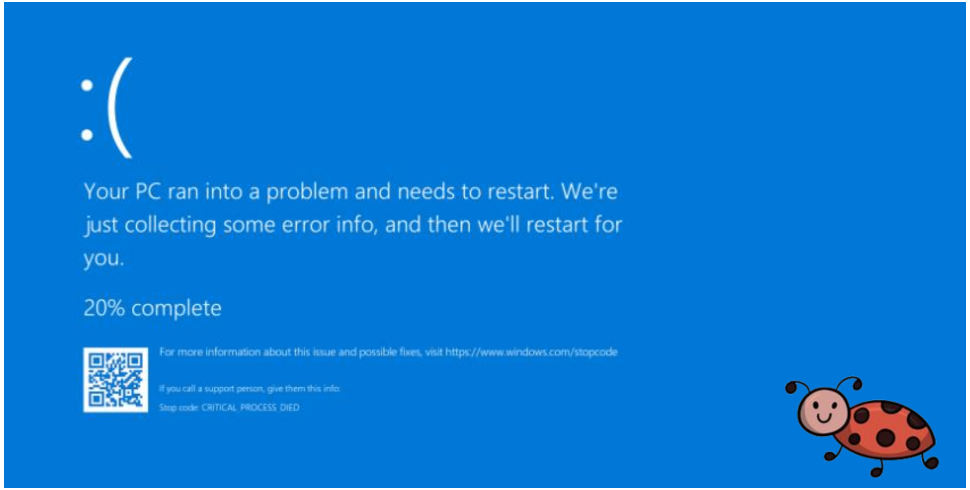
Use an operating system for complex systems with soft real-time requirements

Agenda

- Operating Systems
- Threads
- Layer Separation
- Network Problems
- Message Structure
- Simulators
- Logs
- Monitoring

Two threads tried to create large messages at the same time.

The second one always failed



:(

Your PC ran into a problem and needs to restart. We're just collecting some error info, and then we'll restart for you.

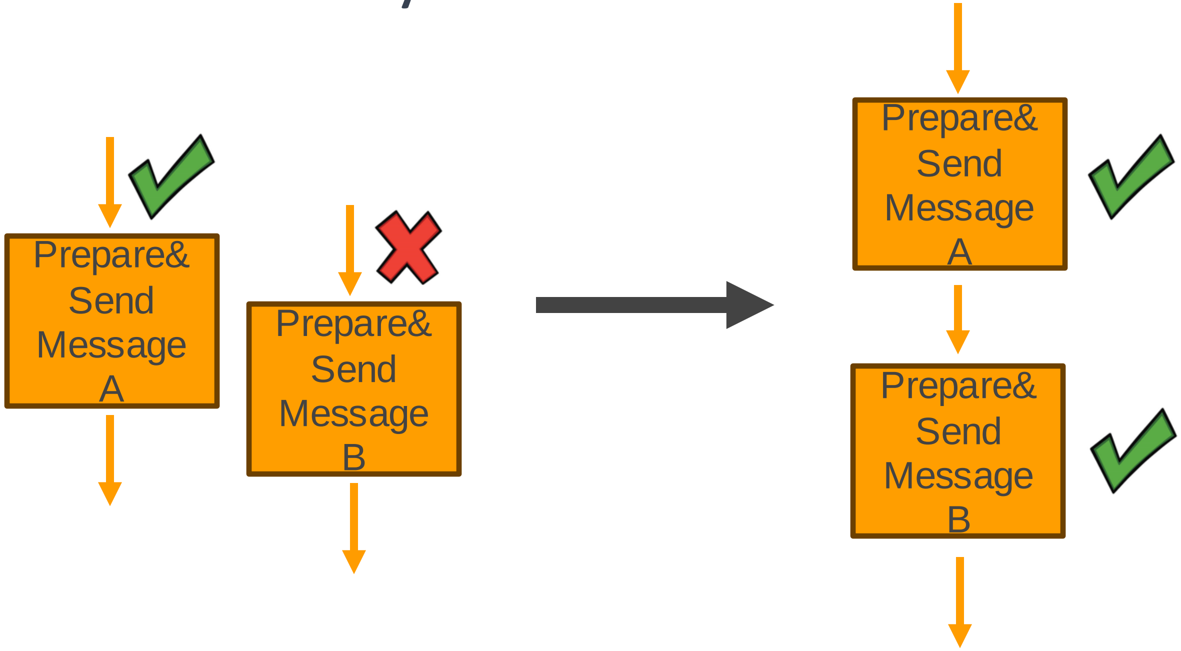
20% complete

For more information about this issue and possible fixes, visit https://www.windows.com/stopcode

If you call a support person, give them this info:

Stop code: CRITICAL_PROCESS_DIED

Change from asynchronous work to synchronous work



👍 Keep the number of threads to the bare minimum*

*The most difficult bugs in a system are related to multiple threads running simultaneously

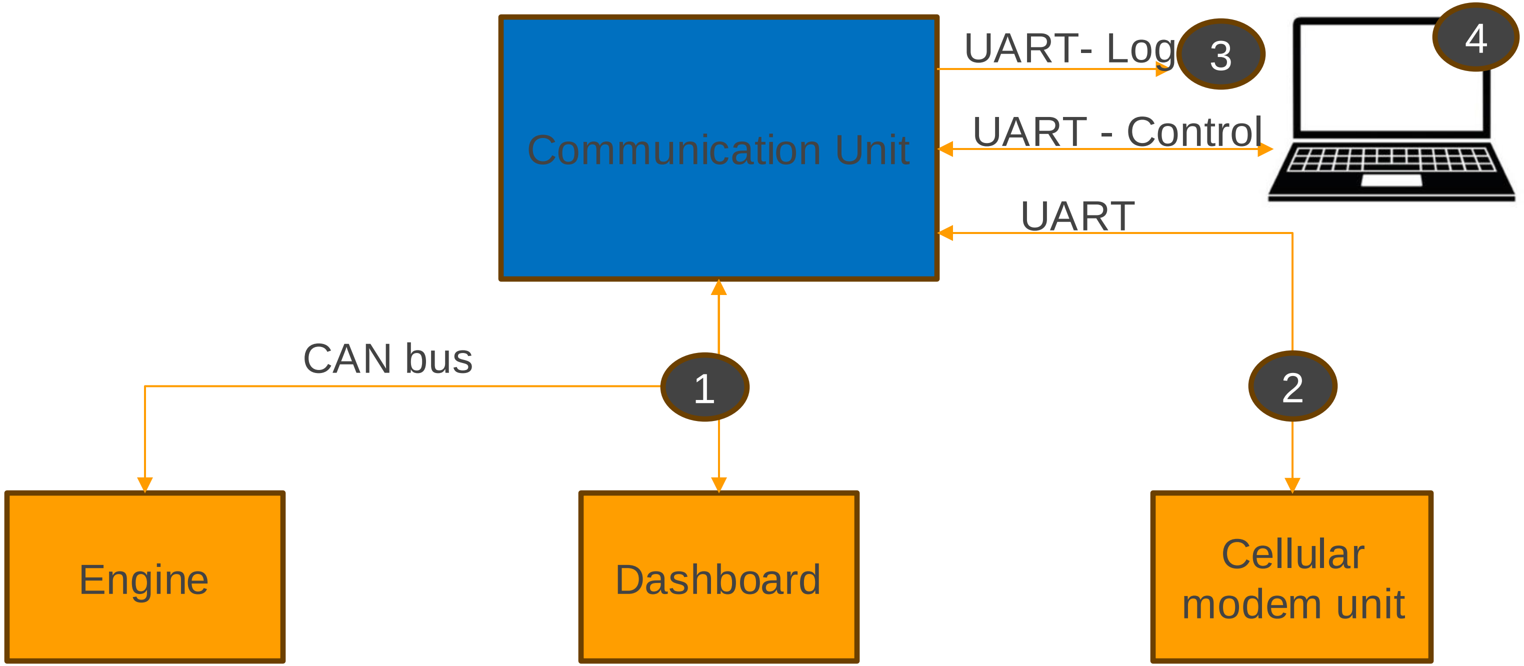
👍 Good practice:

Thread to each communication interface

+

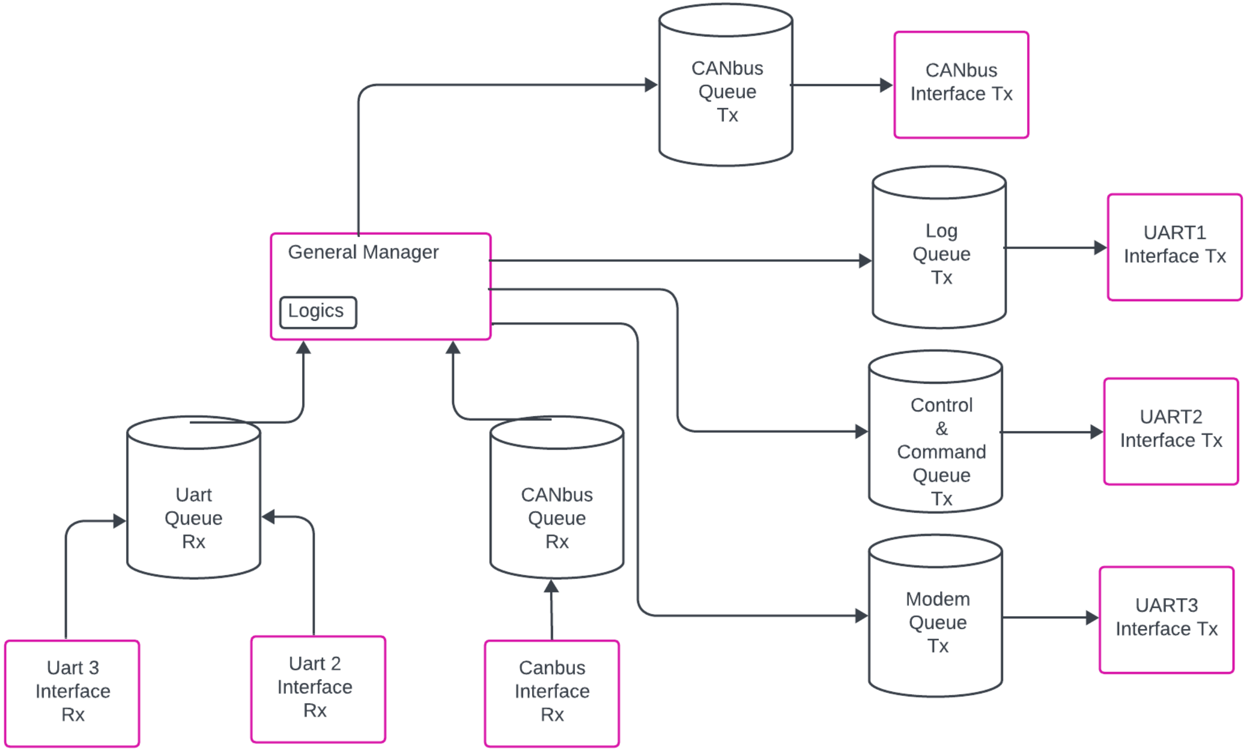
Periodic thread

Threads



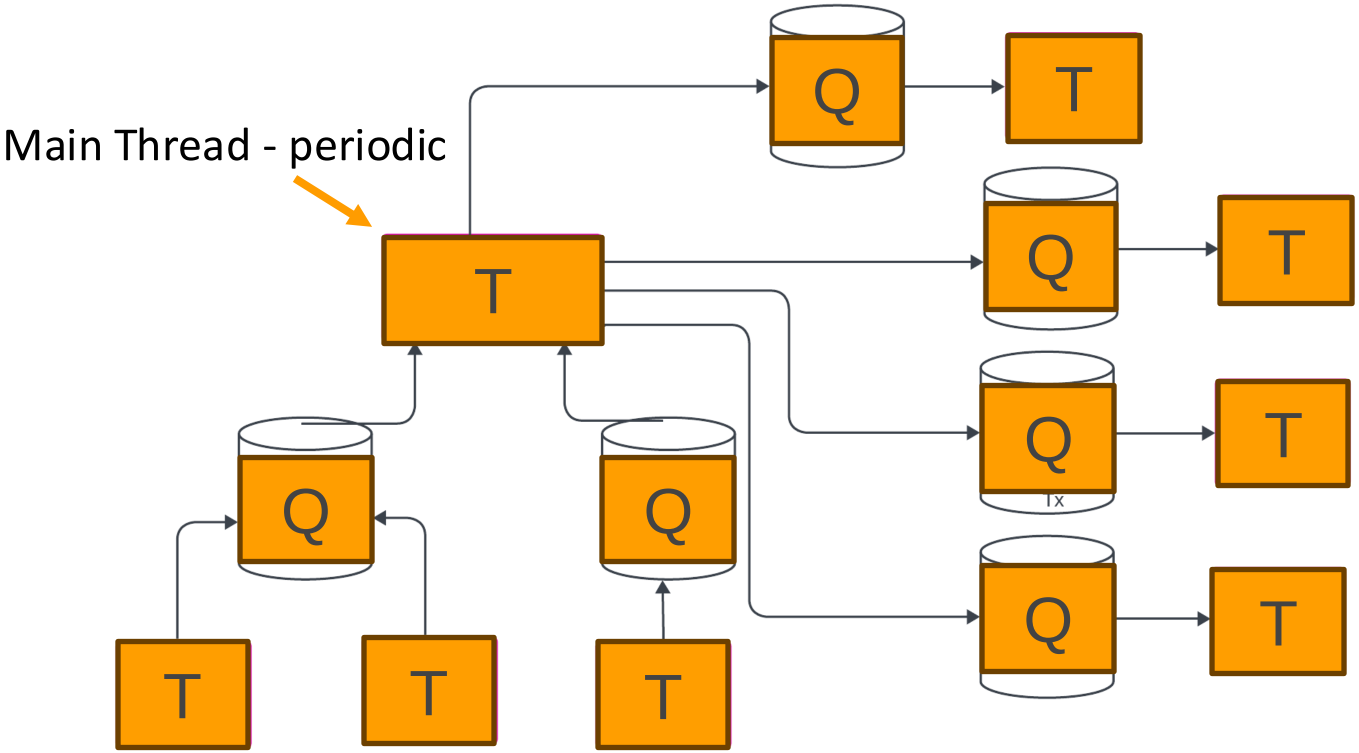
Communication Unit
UART- Log
3
4
UART - Control
UART
CAN bus
1
2
Engine
Dashboard
Cellular modem unit

Threads



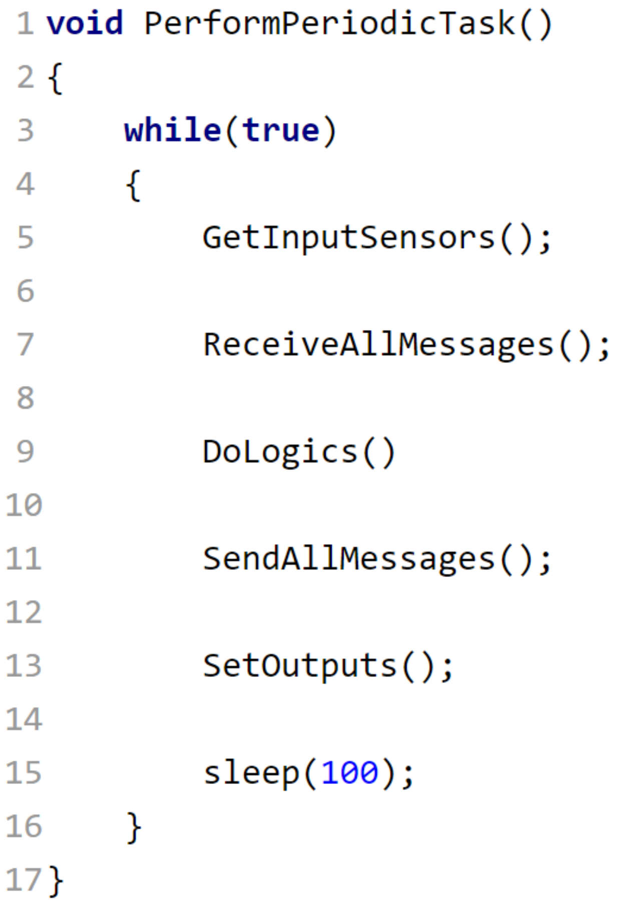
CANbus Queue Tx
CANbus Interface Tx
Log Queue Tx
UART1 Interface Tx
General Manager
Logics
Control & Command Queue Tx
UART2 Interface Tx
Uart Queue Rx
CANbus Queue Rx
Modem Queue Tx
UART3 Interface Tx
Uart 3 Interface Rx
Uart 2 Interface Rx
Canbus Interface Rx

Threads



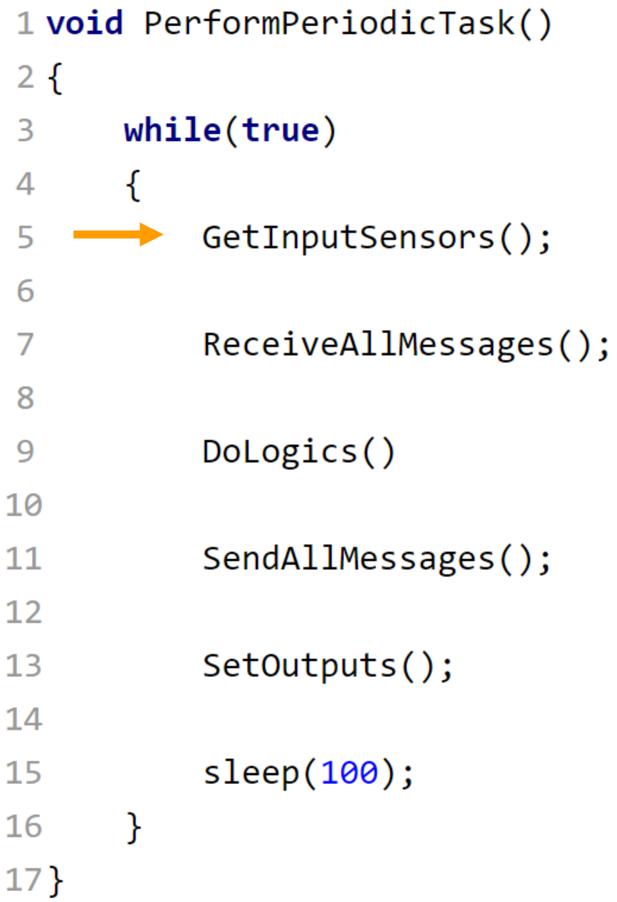
Main Thread - periodic

Threads



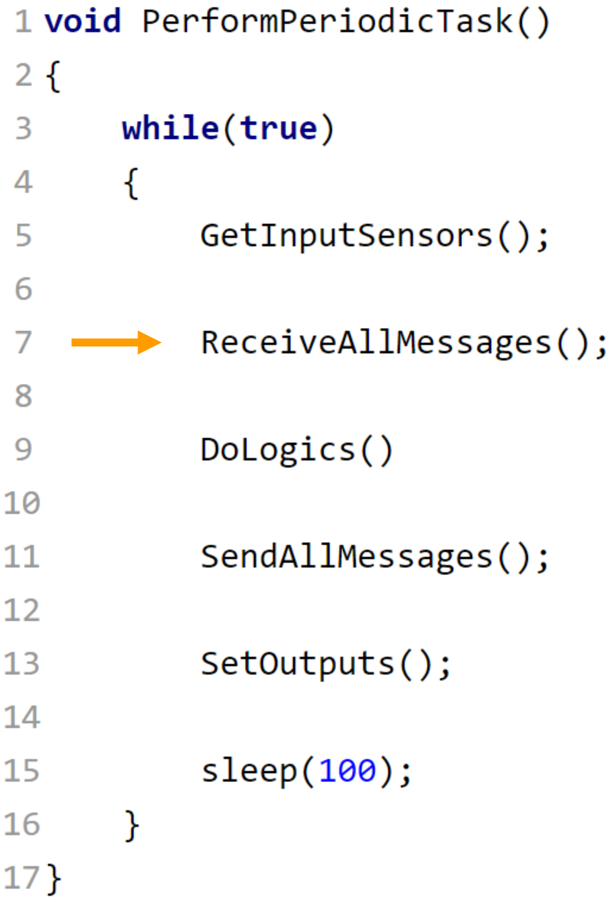
```
void PerformPeriodicTask()
{
    while(true)
    {
        GetInputSensors();

        ReceiveAllMessages();

        DoLogics()

        SendAllMessages();

        SetOutputs();

        sleep(100);
    }
}
```

Threads



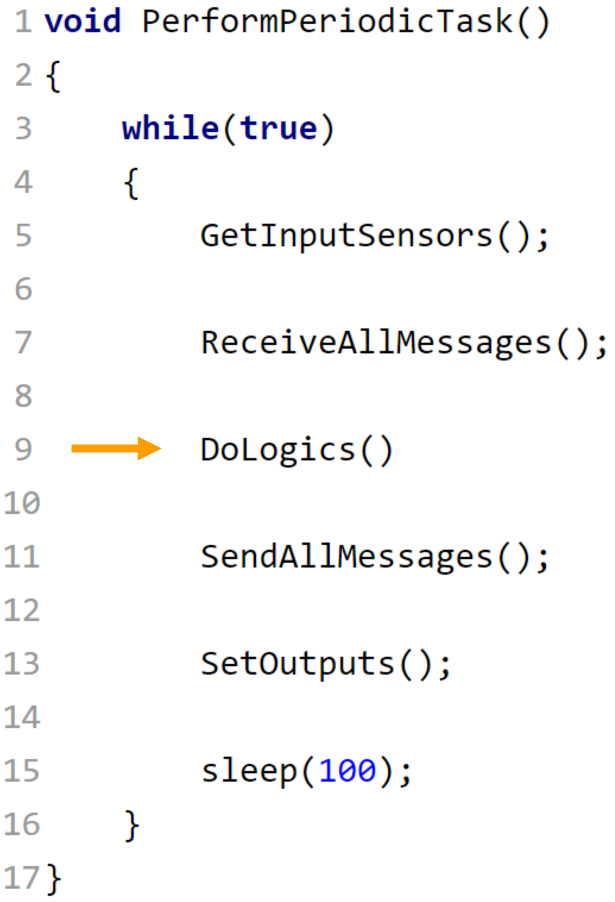
```
void PerformPeriodicTask()
{
    while(true)
    {
        GetInputSensors();

        ReceiveAllMessages();

        DoLogics()

        SendAllMessages();

        SetOutputs();

        sleep(100);
    }
}
```

Threads



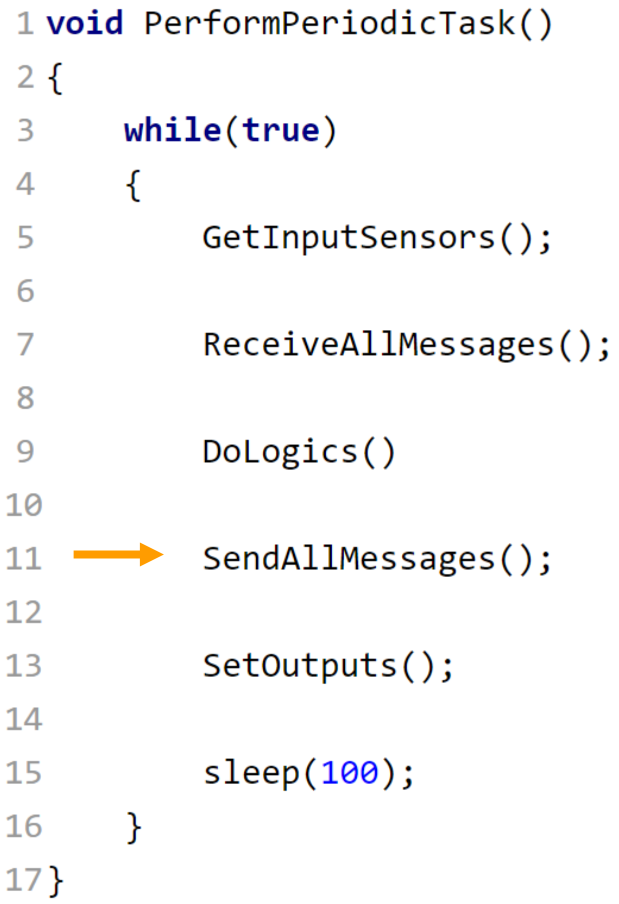
```
void PerformPeriodicTask()
{
    while(true)
    {
        GetInputSensors();

        ReceiveAllMessages();

        DoLogics()

        SendAllMessages();

        SetOutputs();

        sleep(100);
    }
}
```

Threads



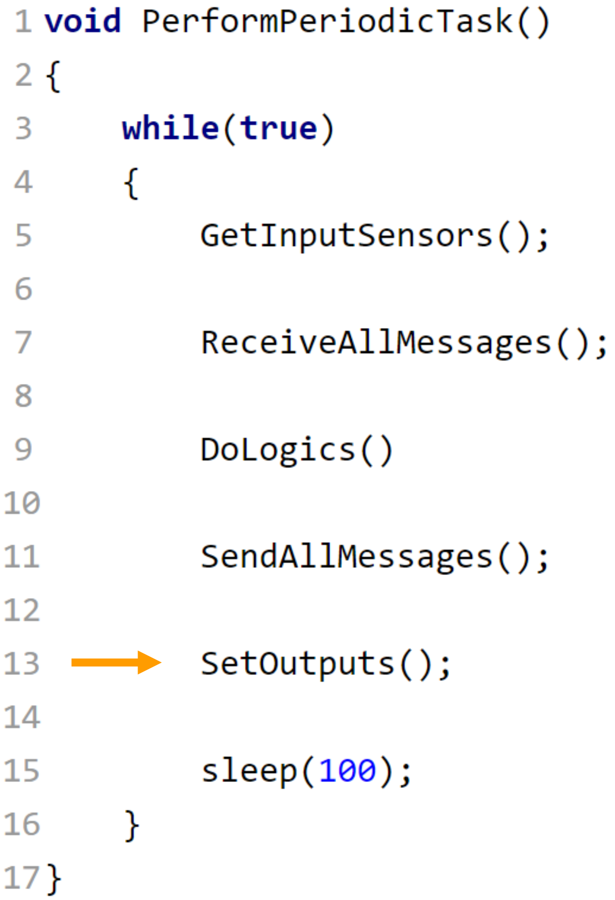
```
void PerformPeriodicTask()
{
    while(true)
    {
        GetInputSensors();

        ReceiveAllMessages();

        DoLogics()

        SendAllMessages();

        SetOutputs();

        sleep(100);
    }
}
```

Threads



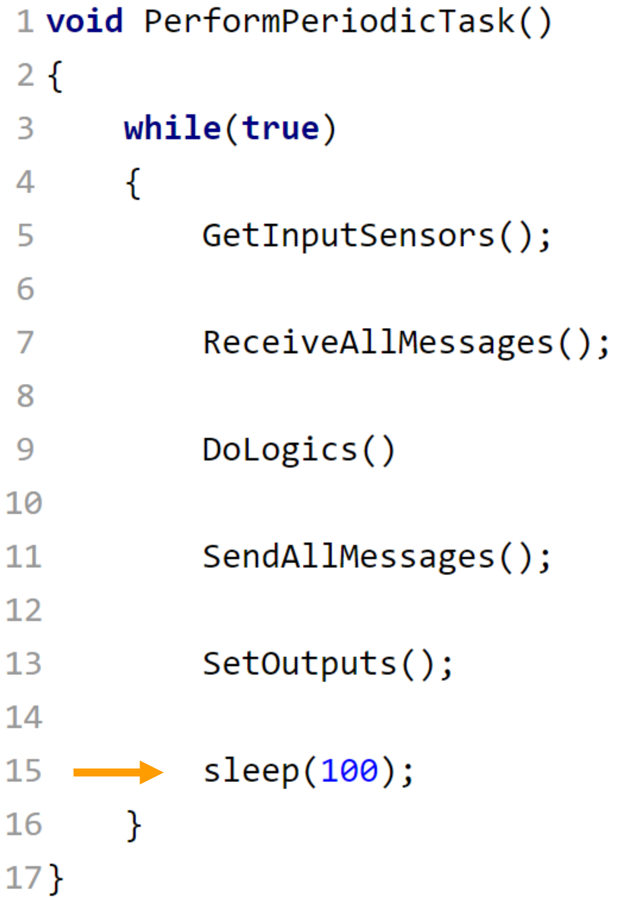
```
void PerformPeriodicTask()
{
    while(true)
    {
        GetInputSensors();

        ReceiveAllMessages();

        DoLogics()

        SendAllMessages();

        SetOutputs();

        sleep(100);
    }
}
```

Threads



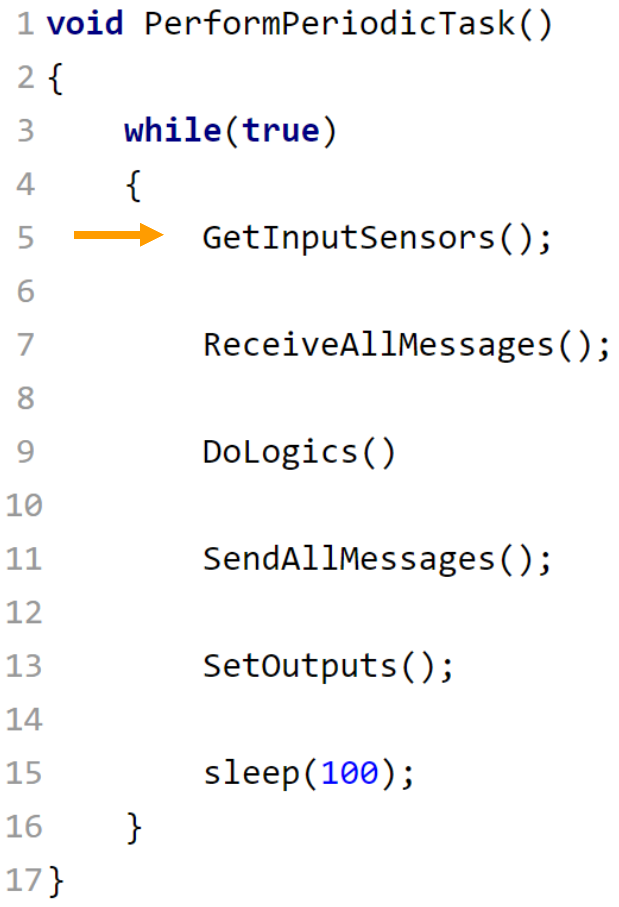
```
void PerformPeriodicTask()
{
    while(true)
    {
        GetInputSensors();

        ReceiveAllMessages();

        DoLogics()

        SendAllMessages();

        SetOutputs();

        sleep(100);
    }
}
```

Threads



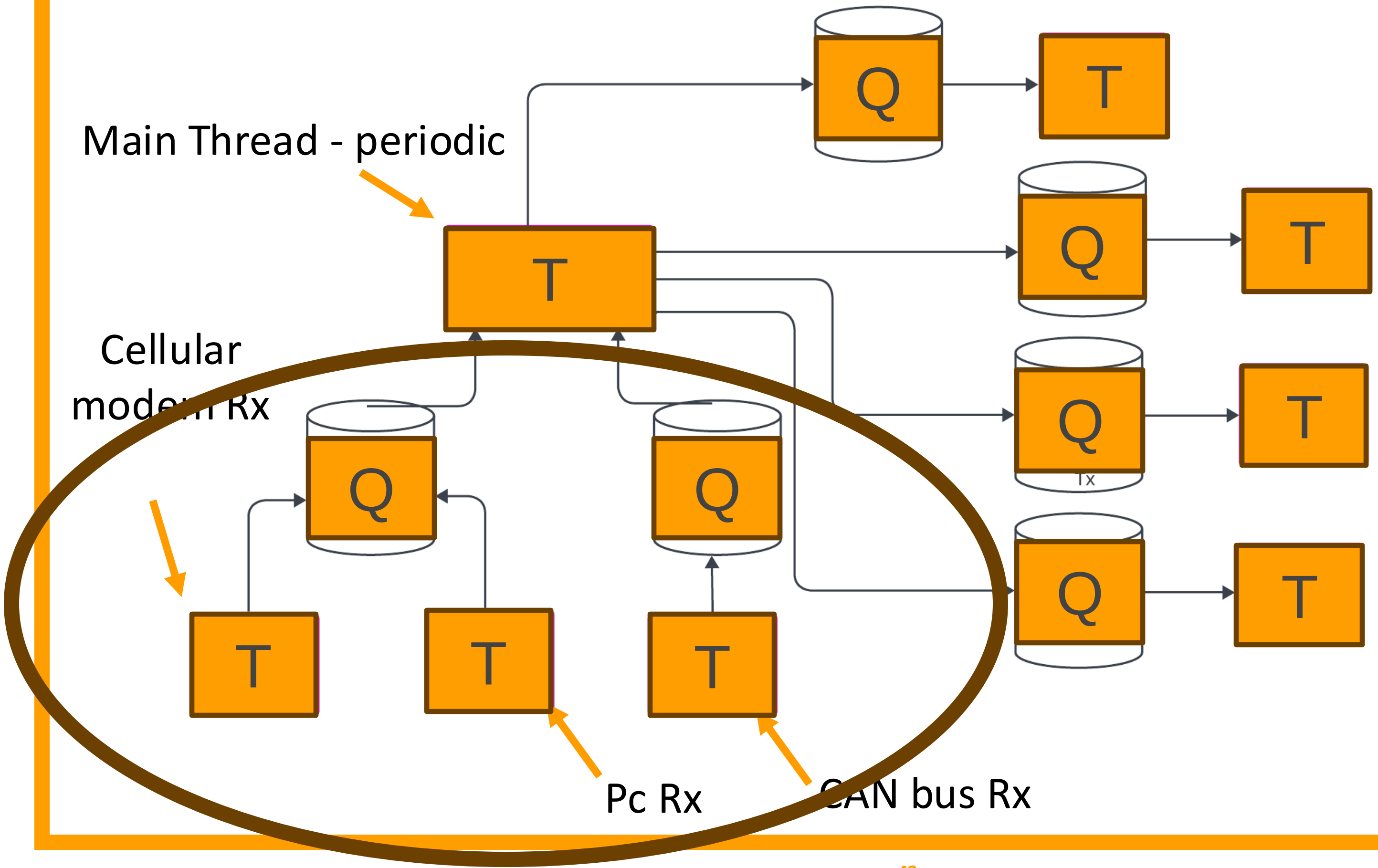
```
void PerformPeriodicTask()
{
    while(true)
    {
        GetInputSensors();

        ReceiveAllMessages();

        DoLogics()

        SendAllMessages();

        SetOutputs();

        sleep(100);
    }
}
```

Threads

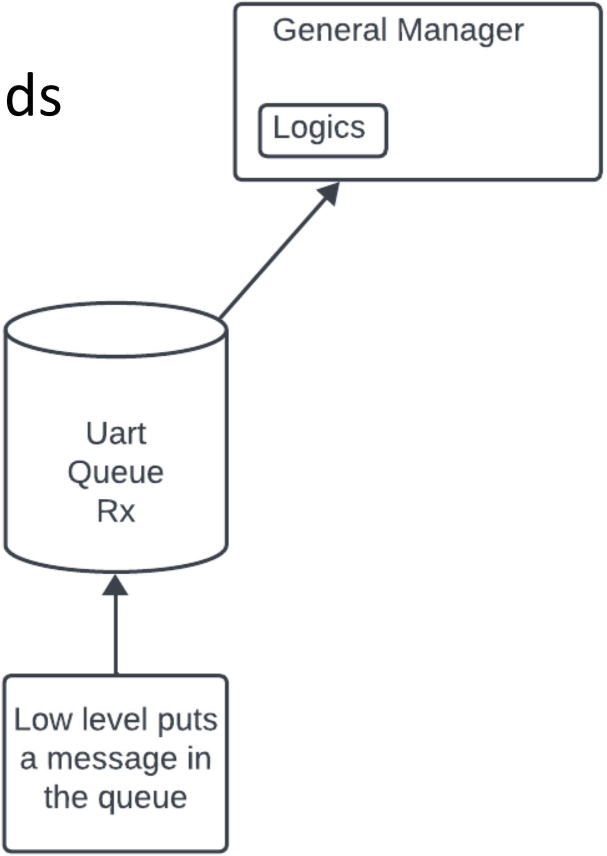


```
void PerformPeriodicTask()
{
    while(true)
    {
        GetInputSensors();

        ReceiveAllMessages();

        DoLogics()

        SendAllMessages();

        SetOutputs();

        sleep(100);
    }
}
```

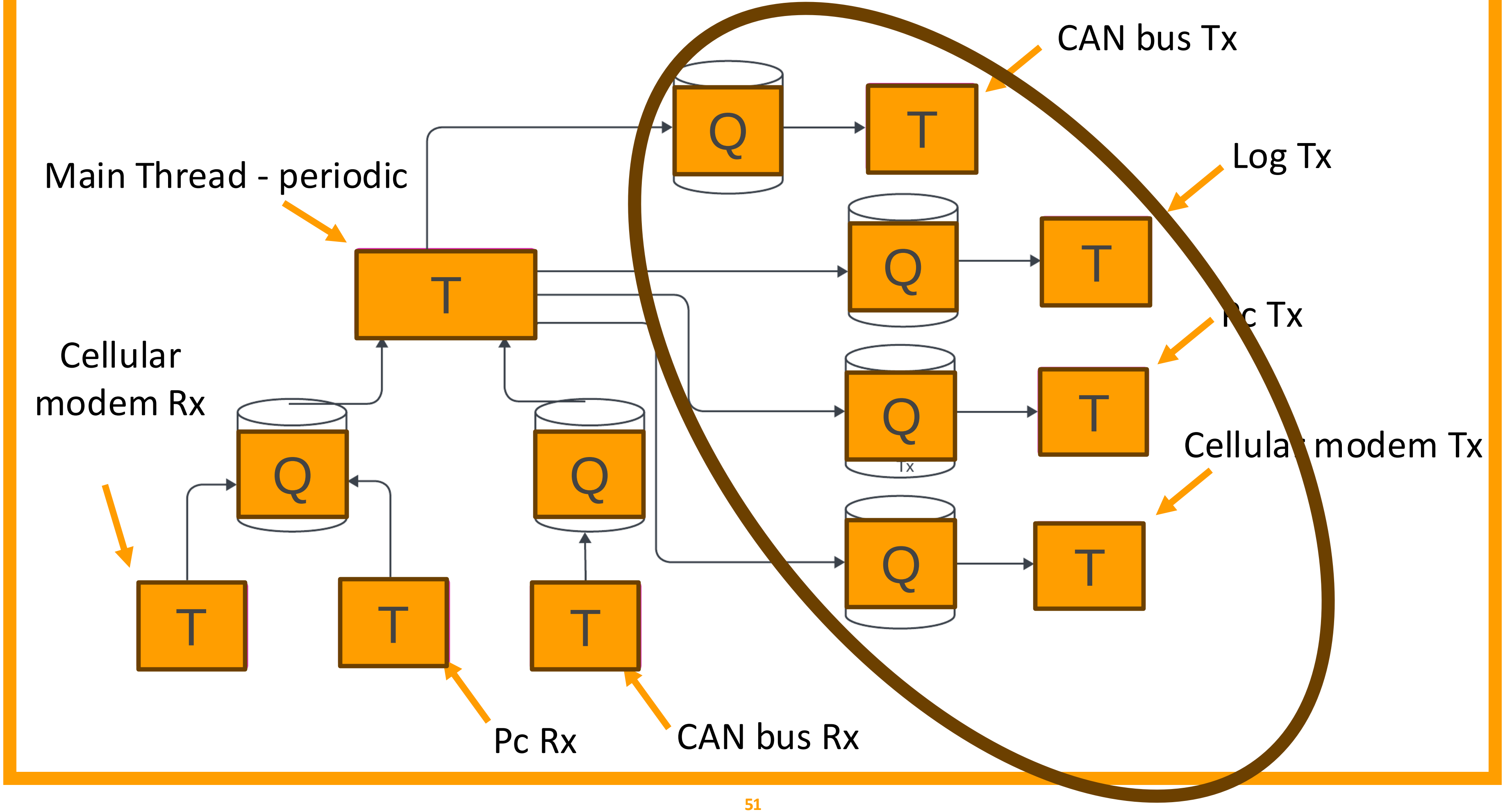
Threads


Main Thread - periodic
Q
T
Q
T
Q
Tx
T
Q
T
T
Cellular
modem Rx
Q
Q
T
T
T
Pc Rx
CAN bus Rx

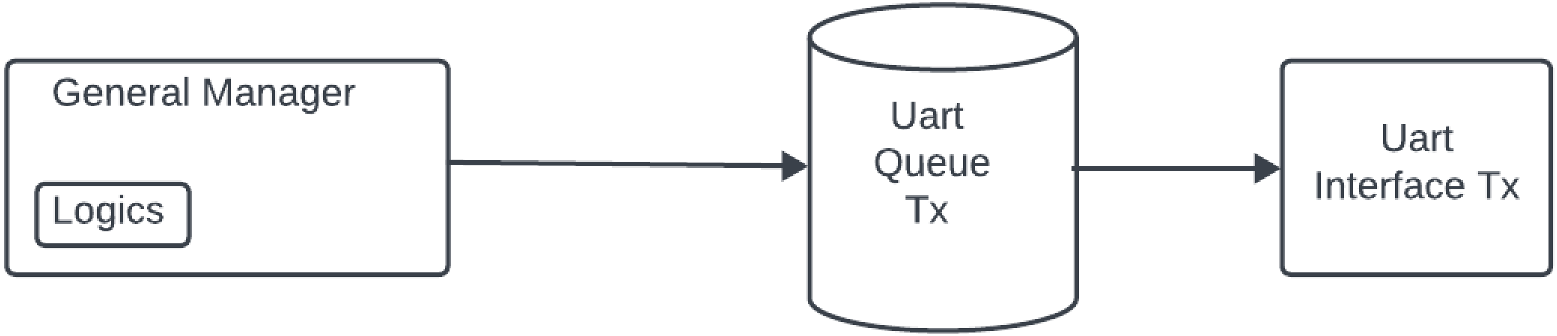
Sync between threads (Input to queue)



Threads


CAN bus Tx
Log Tx
Pc Tx
Cellular modem Tx
Main Thread - periodic
Cellular
modem Rx
Pc Rx
CAN bus Rx
Q
T
Q
T
T
Q
T
Q
T
Tx
Q
T
Q
Q
T
T
T

Sync between threads (Queue to output)



> *Keep the number of threads to the bare minimum*

Agenda

- □ Operating Systems
- □ Threads
- □ Layer Separation
- □ Network Problems
- □ Message Structure
- □ Simulators
- □ Logs
- □ Monitoring

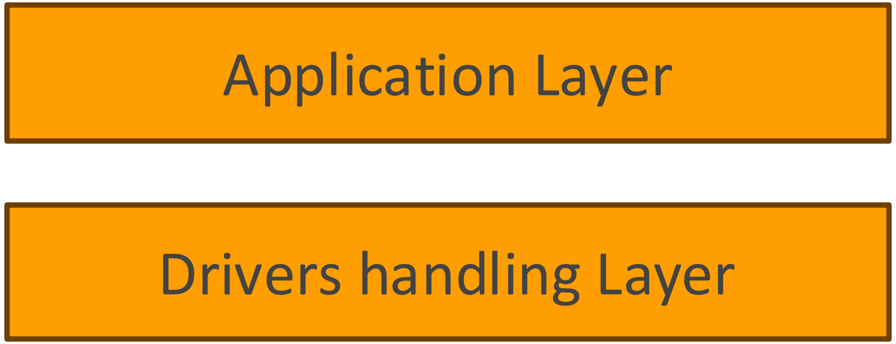
Layer Separation

👍 Separate the logic layer from the hardware layer

Embedded Software

Layer Separation

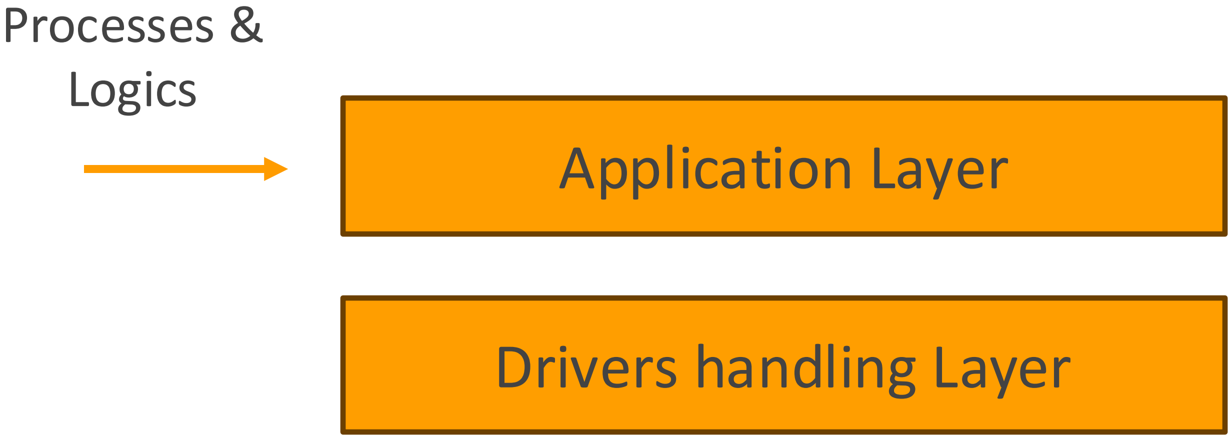
👍 Separate the logic layer from the hardware layer


Application Layer

Drivers handling Layer

Layer Separation

Separate the logic layer from the hardware layer



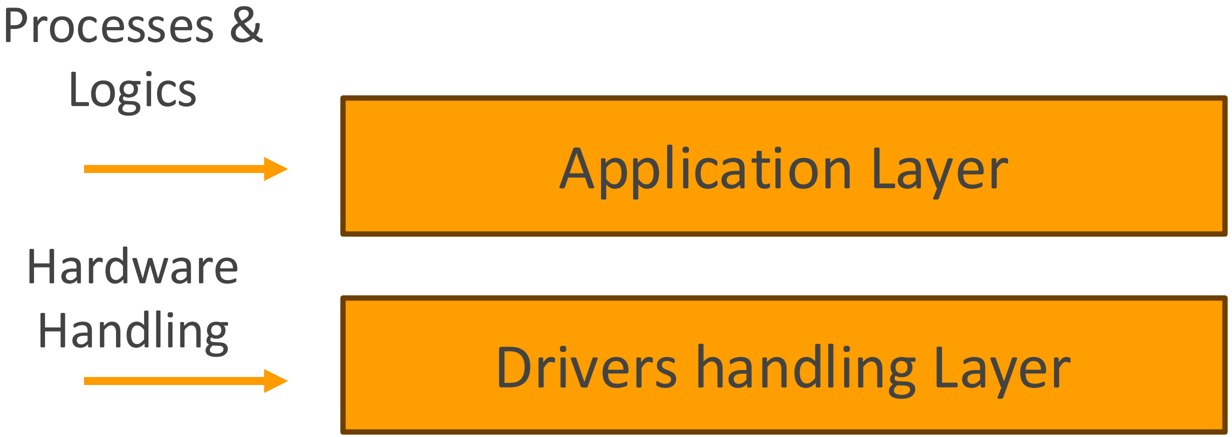
Processes & Logics

Application Layer

Drivers handling Layer

Layer Separation

Separate the logic layer from the hardware layer



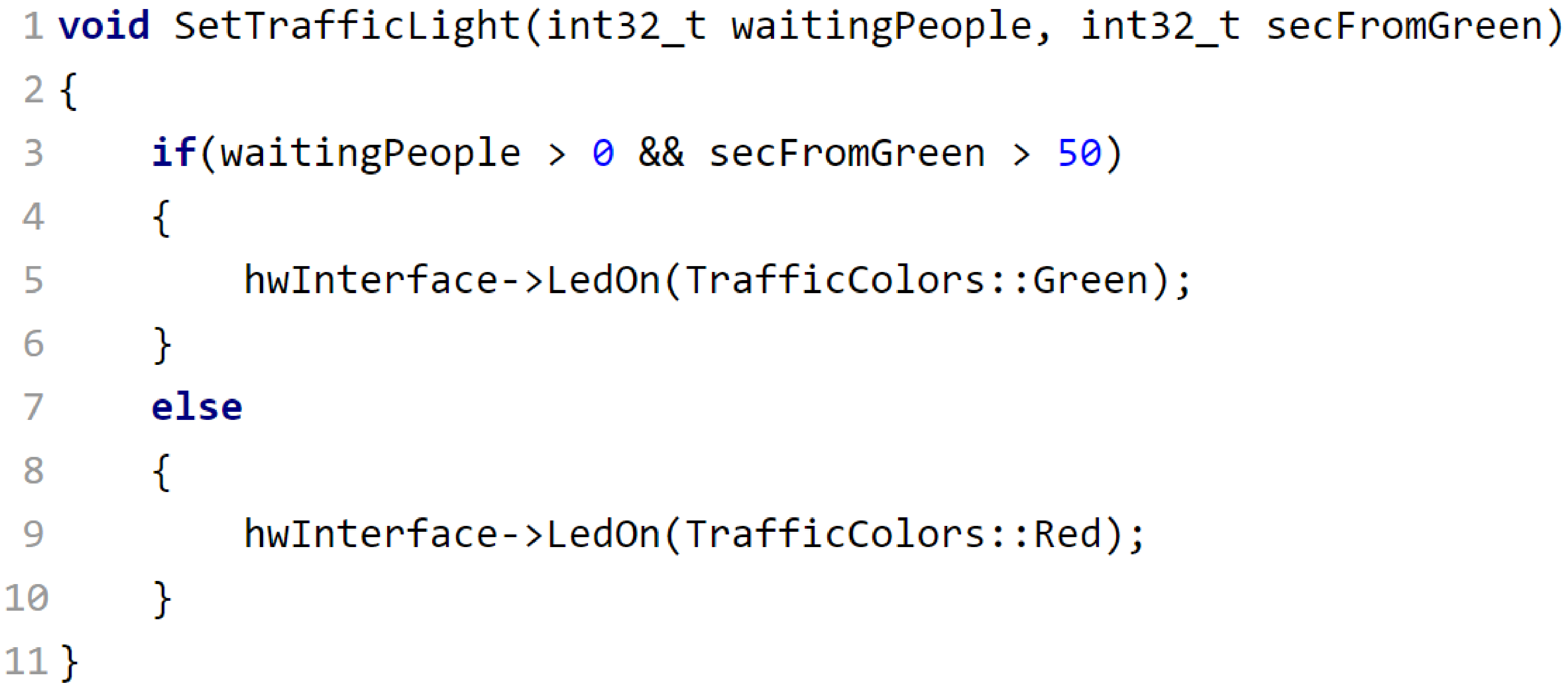
Processes & Logics

Application Layer

Hardware Handling

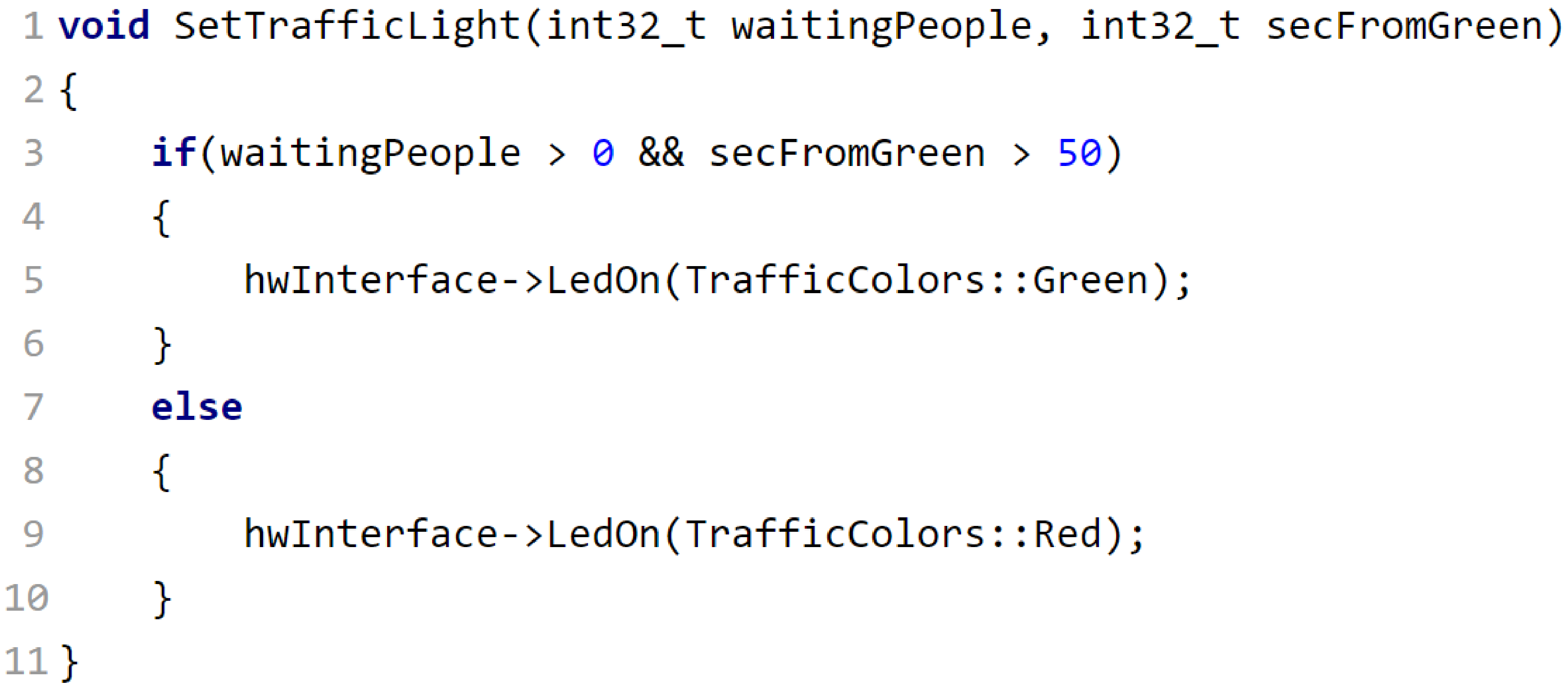
Drivers handling Layer

Layer Separation



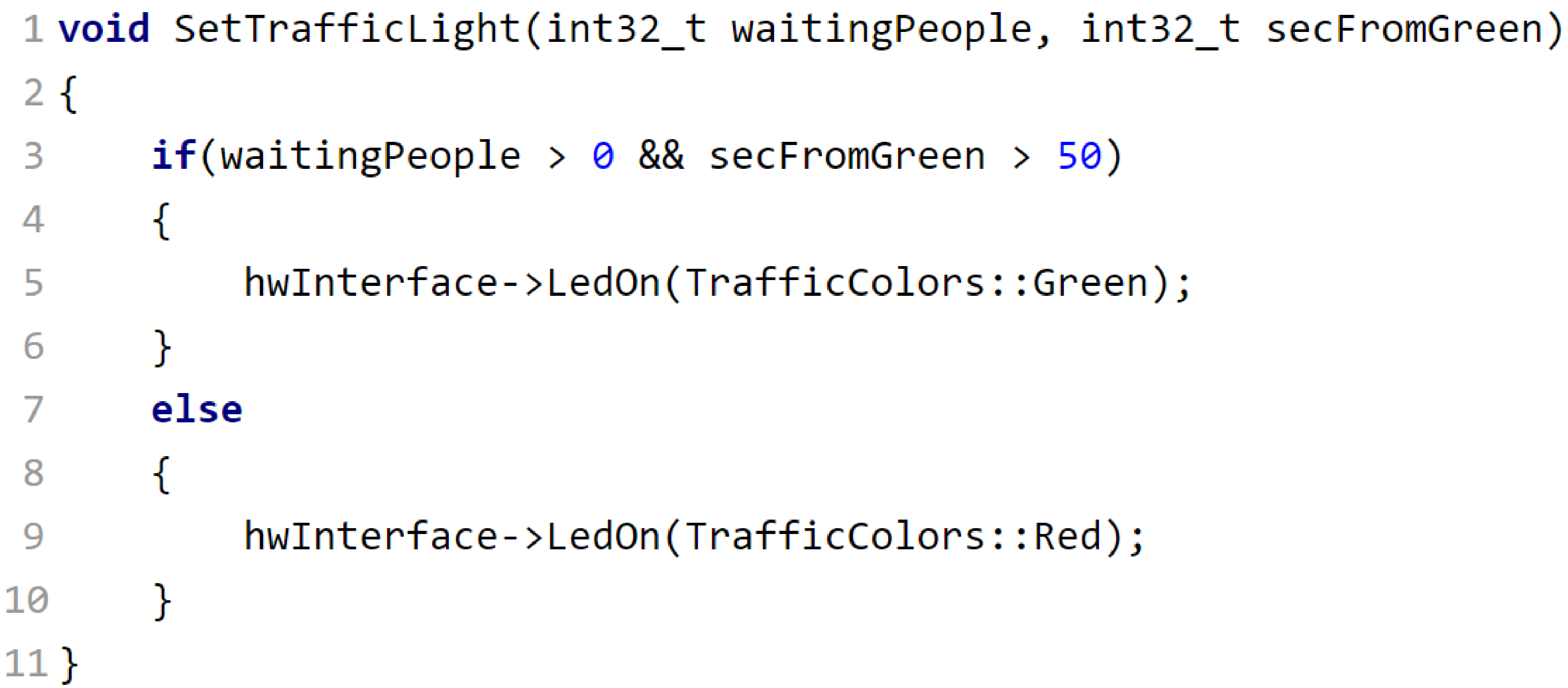
```
void SetTrafficLight(int32_t waitingPeople, int32_t secFromGreen)
{
    if(waitingPeople > 0 && secFromGreen > 50)
    {
        hwInterface->LedOn(TrafficColors::Green);
    }
    else
    {
        hwInterface->LedOn(TrafficColors::Red);
    }
}
```

How are we going to test it?



```
void SetTrafficLight(int32_t waitingPeople, int32_t secFromGreen)
{
    if(waitingPeople > 0 && secFromGreen > 50)
    {
        hwInterface->LedOn(TrafficColors::Green);
    }
    else
    {
        hwInterface->LedOn(TrafficColors::Red);
    }
}
```

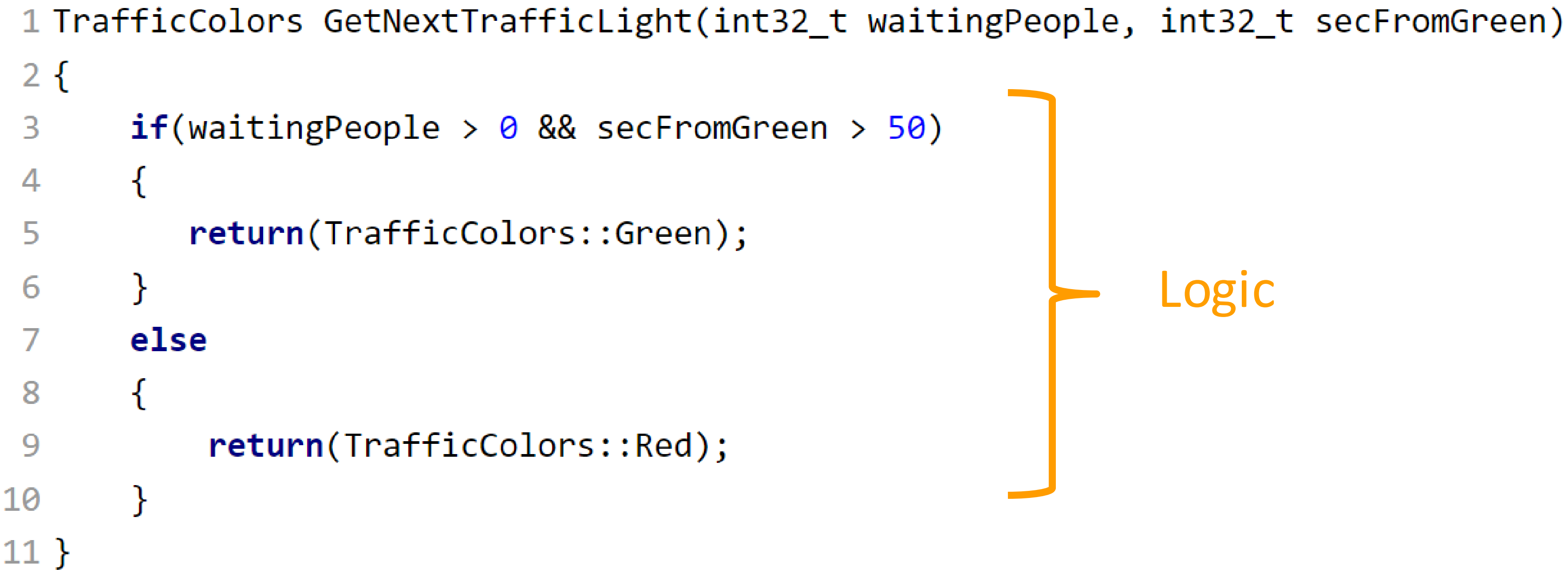
Probably, we’re not…



```
void SetTrafficLight(int32_t waitingPeople, int32_t secFromGreen)
{
    if(waitingPeople > 0 && secFromGreen > 50)
    {
        hwInterface->LedOn(TrafficColors::Green);
    }
    else
    {
        hwInterface->LedOn(TrafficColors::Red);
    }
}
```

Layer Separation

GetNextTrafficLight



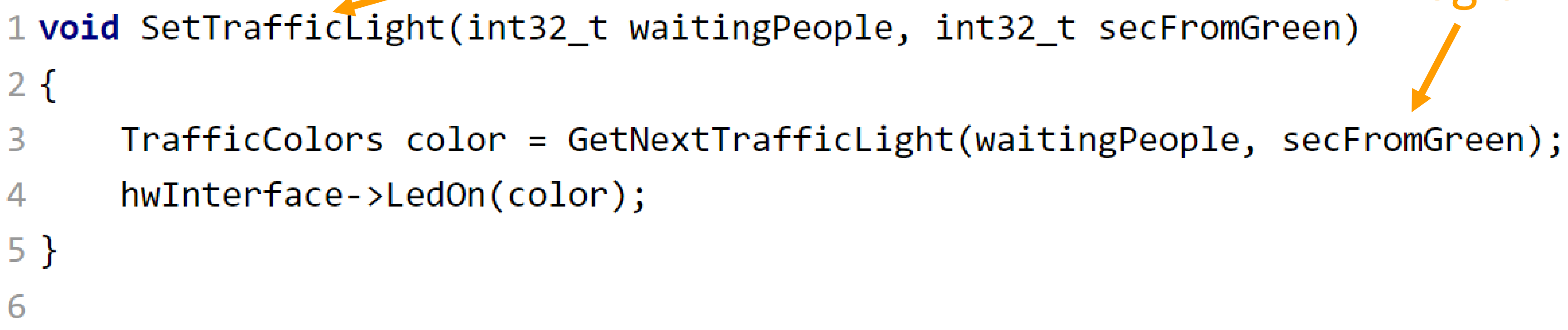
```
TrafficColors GetNextTrafficLight(int32_t waitingPeople, int32_t secFromGreen)
{
    if(waitingPeople > 0 && secFromGreen > 50)
    {
       return(TrafficColors::Green);
    }
    else
    {
        return(TrafficColors::Red);
    }
}
```

Logic

Layer Separation

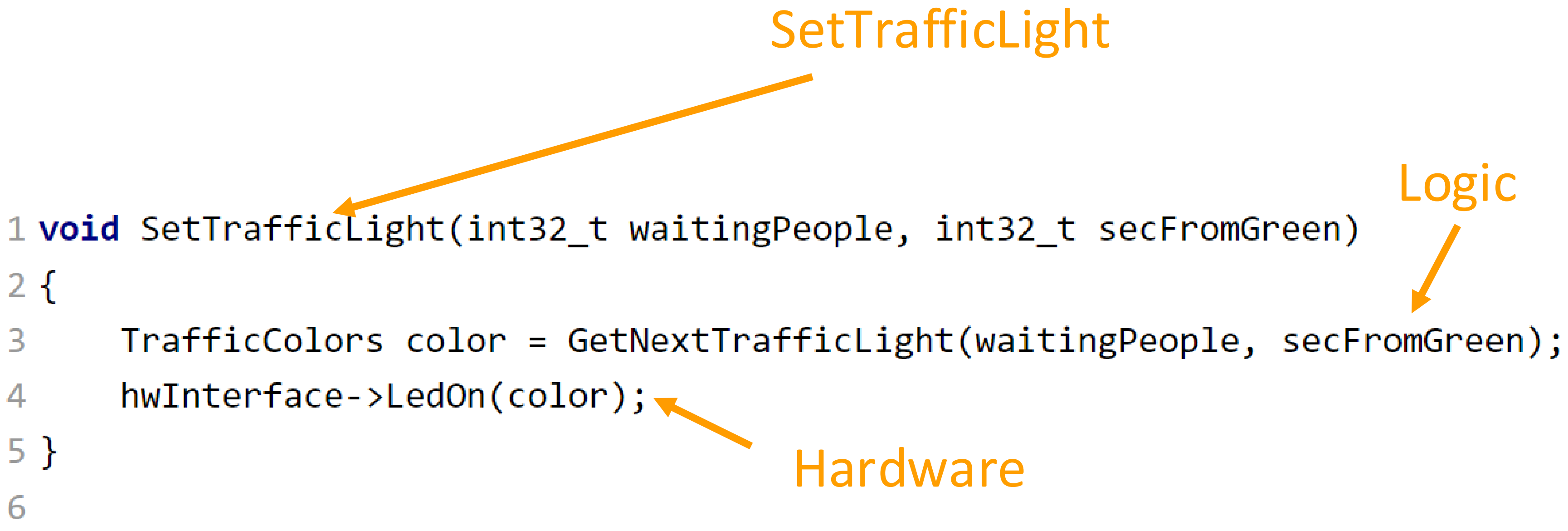
SetTrafficLight

Logic



```
void SetTrafficLight(int32_t waitingPeople, int32_t secFromGreen)
{
    TrafficColors color = GetNextTrafficLight(waitingPeople, secFromGreen);
    hwInterface->LedOn(color);
}

```

Layer Separation



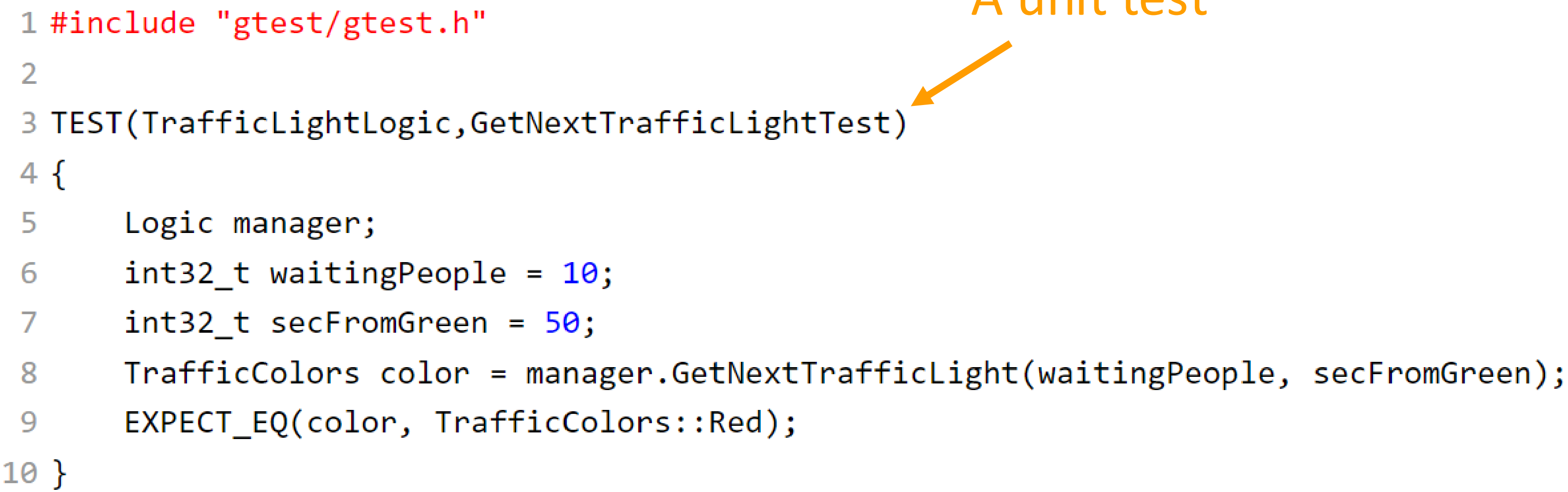
```
void SetTrafficLight(int32_t waitingPeople, int32_t secFromGreen)
{
    TrafficColors color = GetNextTrafficLight(waitingPeople, secFromGreen);
    hwInterface->LedOn(color);
}

```

SetTrafficLight

Logic

Hardware

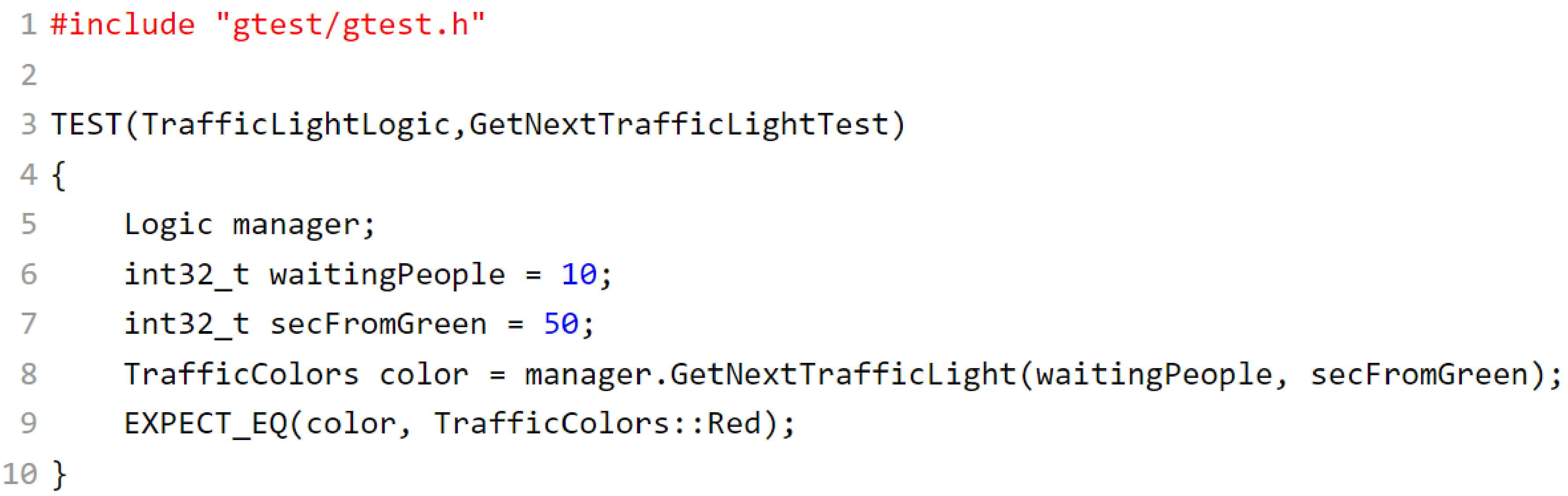
Layer Separation



```cpp
#include "gtest/gtest.h"

TEST(TrafficLightLogic,GetNextTrafficLightTest)
{
    Logic manager;
    int32_t waitingPeople = 10;
    int32_t secFromGreen = 50;
    TrafficColors color = manager.GetNextTrafficLight(waitingPeople, secFromGreen);
    EXPECT_EQ(color, TrafficColors::Red);
}
```

A unit test

Layer Separation



```cpp
#include "gtest/gtest.h"

TEST(TrafficLightLogic,GetNextTrafficLightTest)
{
    Logic manager;
    int32_t waitingPeople = 10;
    int32_t secFromGreen = 50;
    TrafficColors color = manager.GetNextTrafficLight(waitingPeople, secFromGreen);
    EXPECT_EQ(color, TrafficColors::Red);
}
```

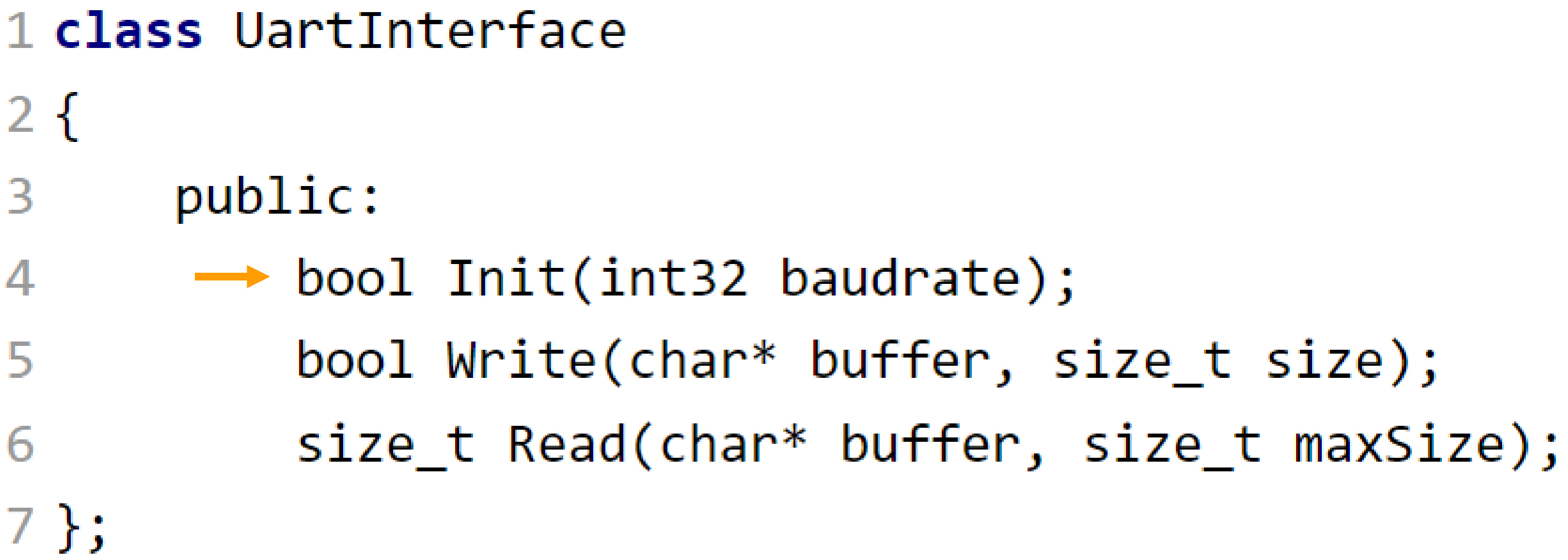
Unit tests encourage us to keep the code simpler

(Try to mock as little as possible)

Okay, we’ve discussed the application layer.

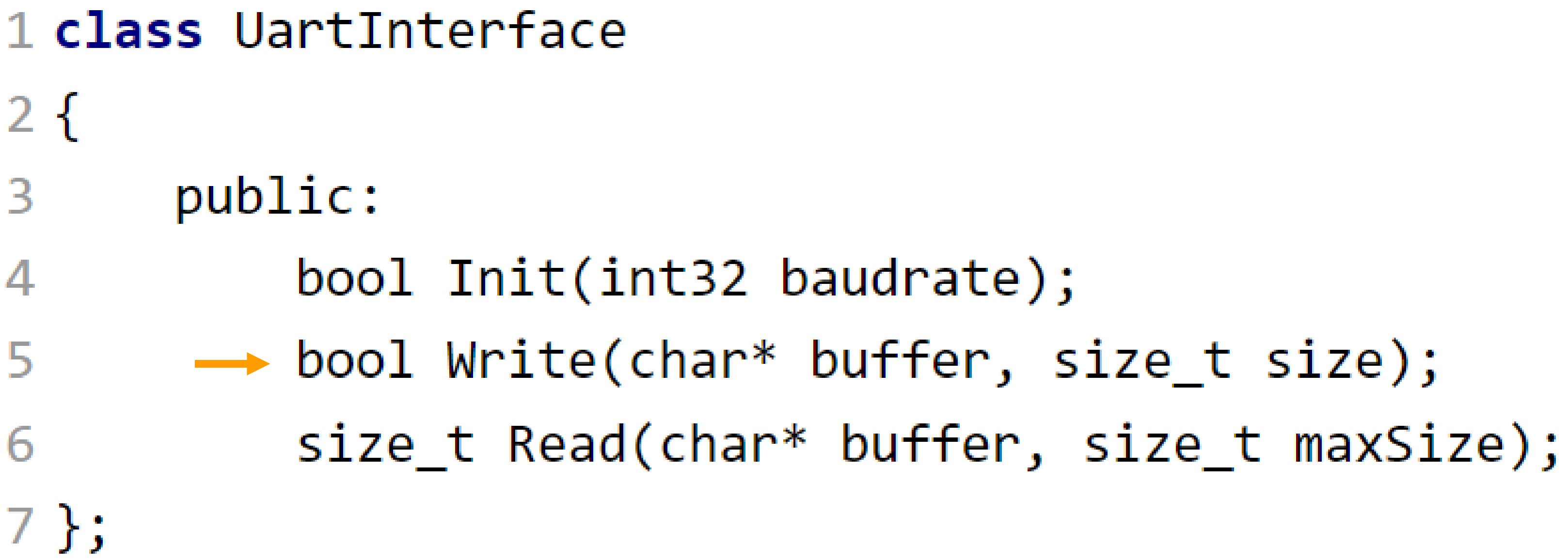
Now, let’s move on to the lower layer

Layer Separation



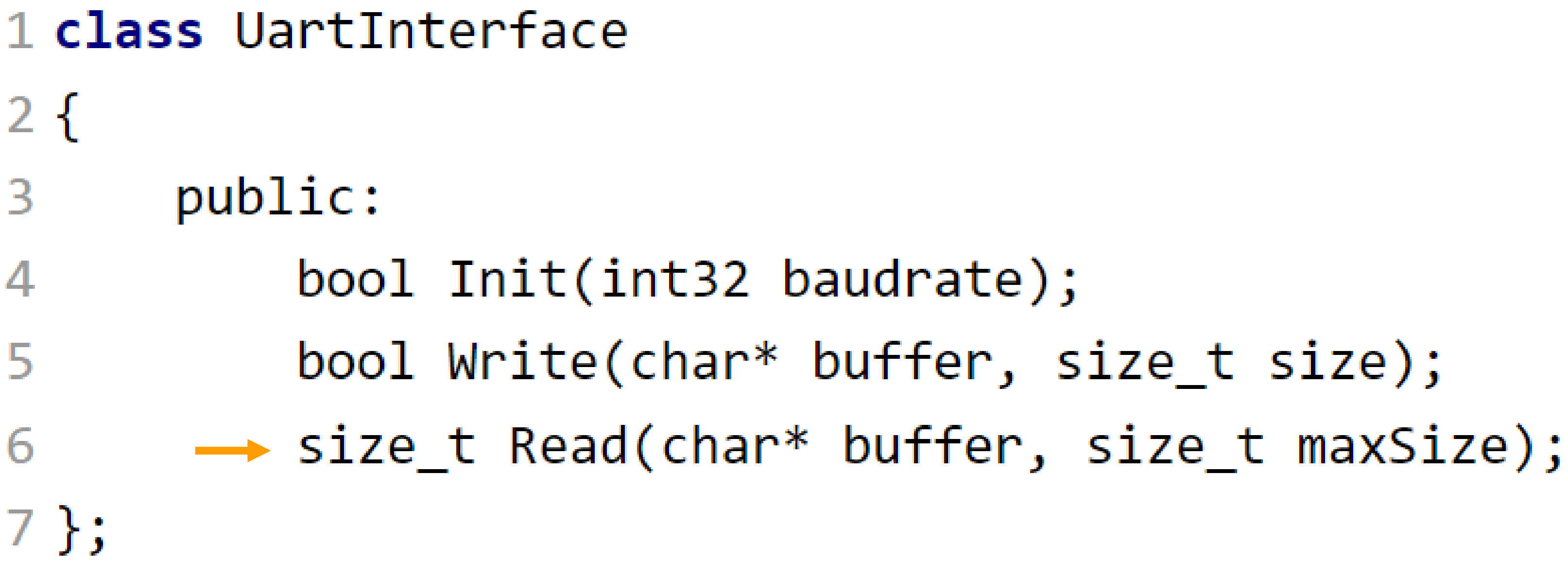
```
class UartInterface
{
    public:
        bool Init(int32 baudrate);
        bool Write(char* buffer, size_t size);
        size_t Read(char* buffer, size_t maxSize);
};
```

Layer Separation



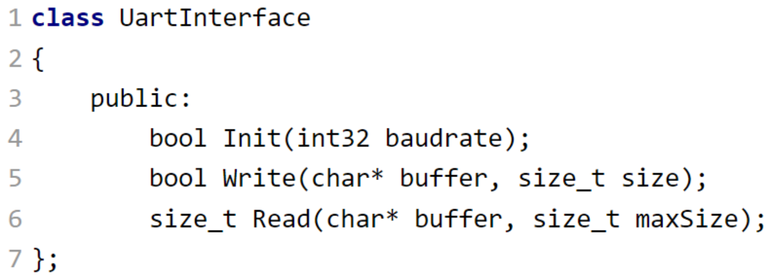
```
class UartInterface
{
    public:
        bool Init(int32 baudrate);
        bool Write(char* buffer, size_t size);
        size_t Read(char* buffer, size_t maxSize);
};
```

Layer Separation

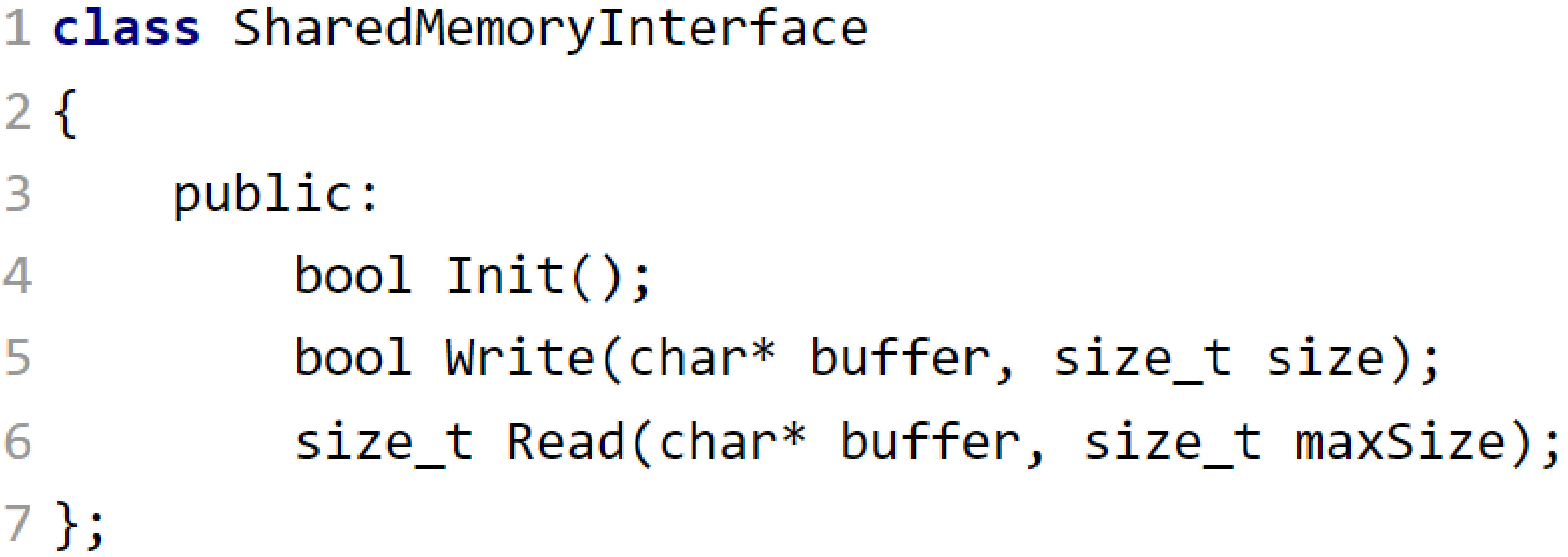


```
class UartInterface
{
    public:
        bool Init(int32 baudrate);
        bool Write(char* buffer, size_t size);
        size_t Read(char* buffer, size_t maxSize);
};
```

Layer Separation

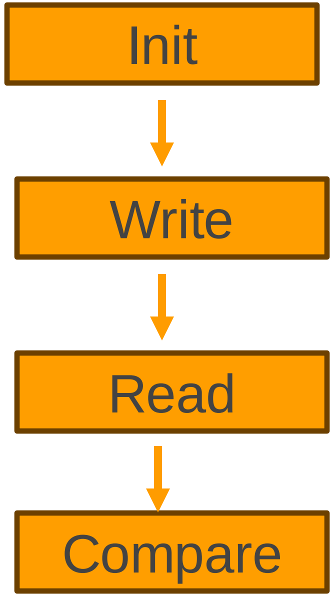


```
class UartInterface
{
    public:
        bool Init(int32 baudrate);
        bool Write(char* buffer, size_t size);
        size_t Read(char* buffer, size_t maxSize);
};
```



```
class SharedMemoryInterface
{
    public:
        bool Init();
        bool Write(char* buffer, size_t size);
        size_t Read(char* buffer, size_t maxSize);
};
```

How Hardware tests should look like



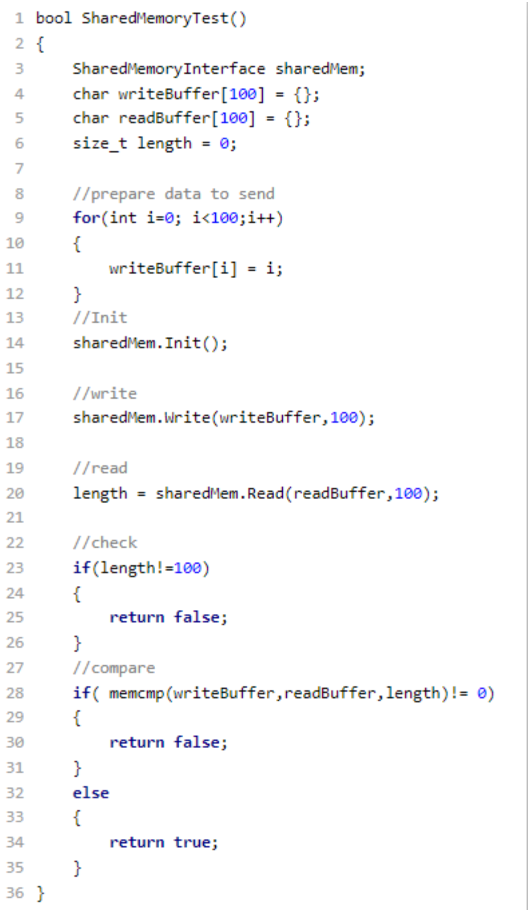
Init

Write

Read

Compare

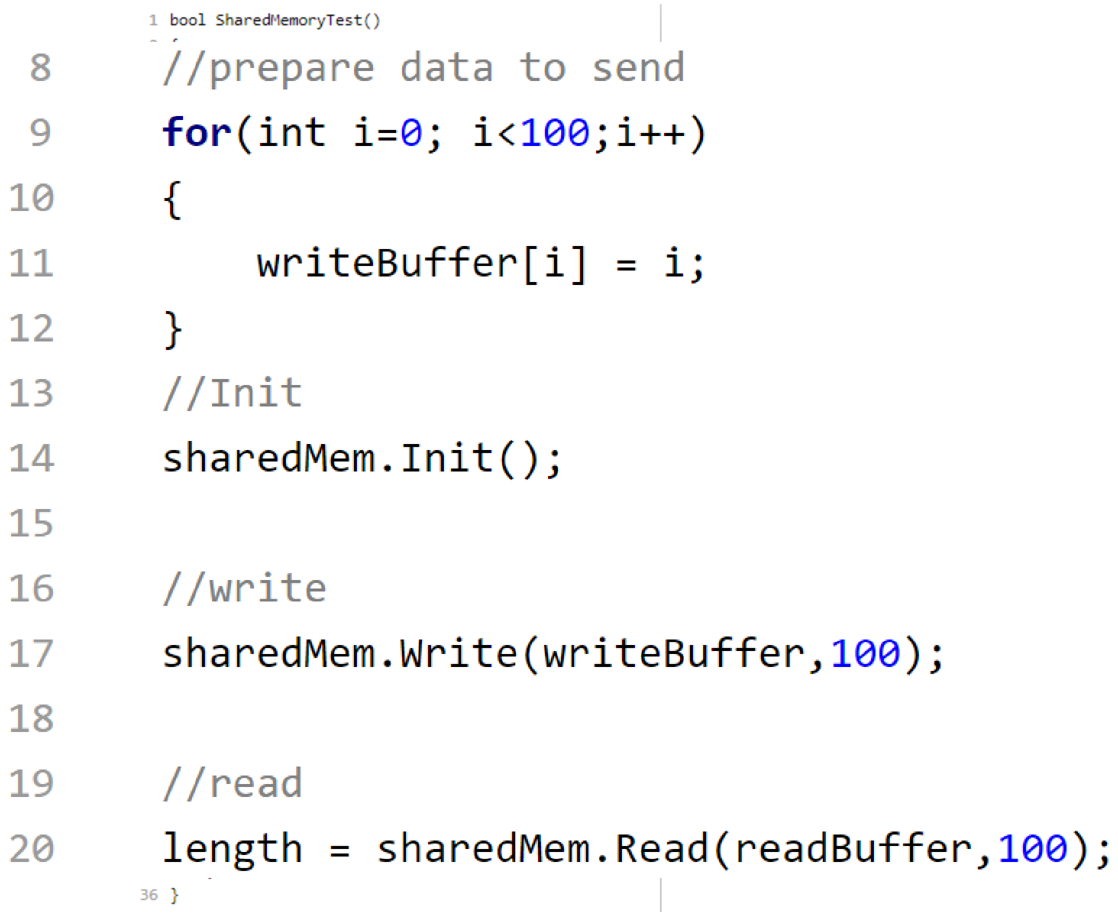
Layer Separation



```
bool SharedMemoryTest()
{
    SharedMemoryInterface sharedMem;
    char writeBuffer[100] = {};
    char readBuffer[100] = {};
    size_t length = 0;

    //prepare data to send
    for(int i=0; i<100;i++)
    {
        writeBuffer[i] = i;
    }
    //Init
    sharedMem.Init();

    //write
    sharedMem.Write(writeBuffer,100);

    //read
    length = sharedMem.Read(readBuffer,100);

    //check
    if(length!=100)
    {
        return false;
    }
    //compare
    if( memcmp(writeBuffer,readBuffer,length)!= 0)
    {
        return false;
    }
    else
    {
        return true;
    }
}
```

Layer Separation

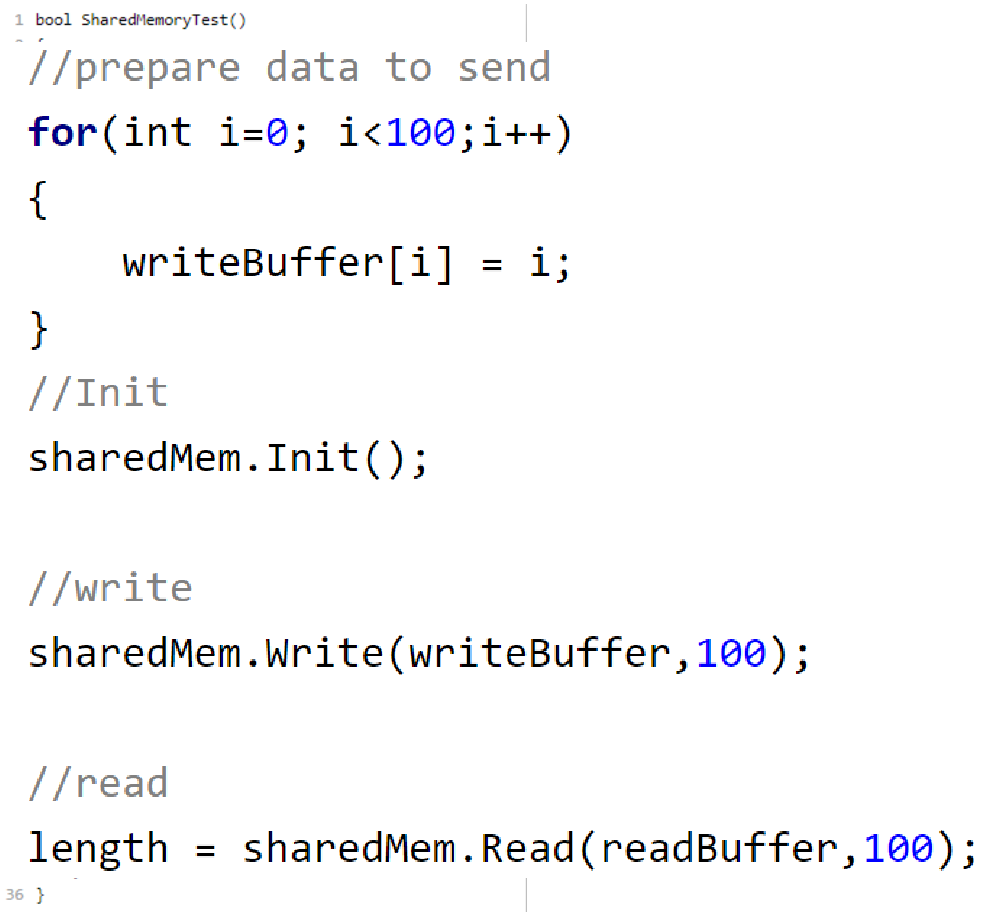
Prepare data



```
bool SharedMemoryTest()
//prepare data to send
for(int i=0; i<100;i++)
{
    writeBuffer[i] = i;
}
//Init
sharedMem.Init();

//write
sharedMem.Write(writeBuffer,100);

//read
length = sharedMem.Read(readBuffer,100);
}
```

Layer Separation

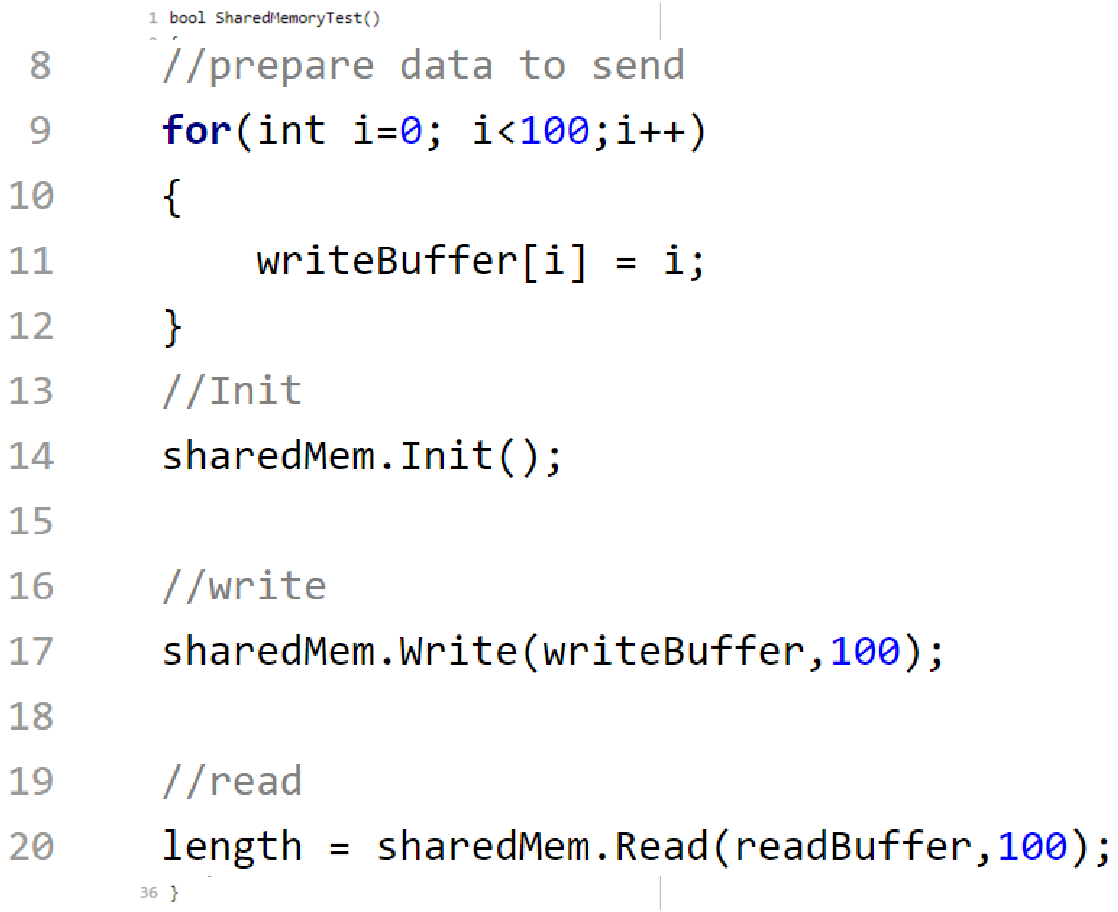
Init



```
bool SharedMemoryTest()
//prepare data to send
for(int i=0; i<100;i++)
{
    writeBuffer[i] = i;
}
//Init
sharedMem.Init();

//write
sharedMem.Write(writeBuffer,100);

//read
length = sharedMem.Read(readBuffer,100);
}
```

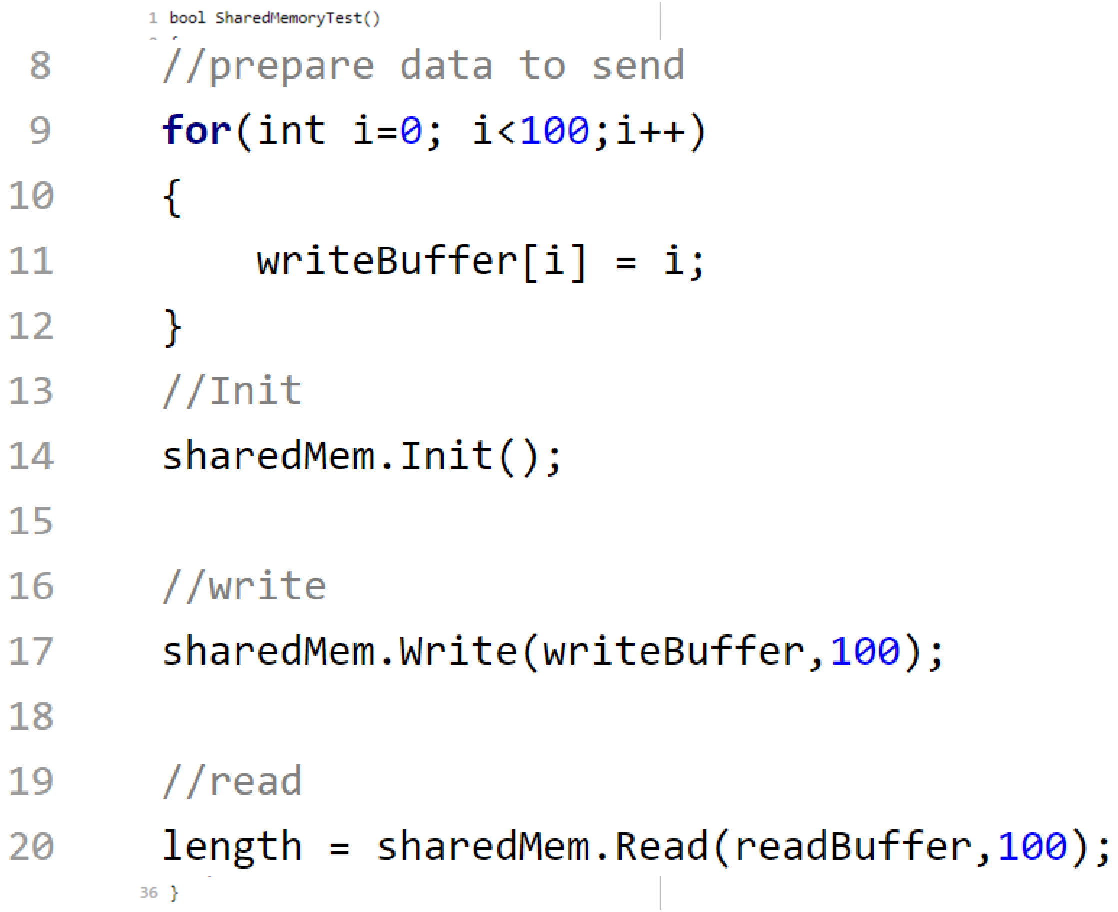
Layer Separation

Write



```
bool SharedMemoryTest()
//prepare data to send
for(int i=0; i<100;i++)
{
    writeBuffer[i] = i;
}
//Init
sharedMem.Init();

//write
sharedMem.Write(writeBuffer,100);

//read
length = sharedMem.Read(readBuffer,100);
}
```

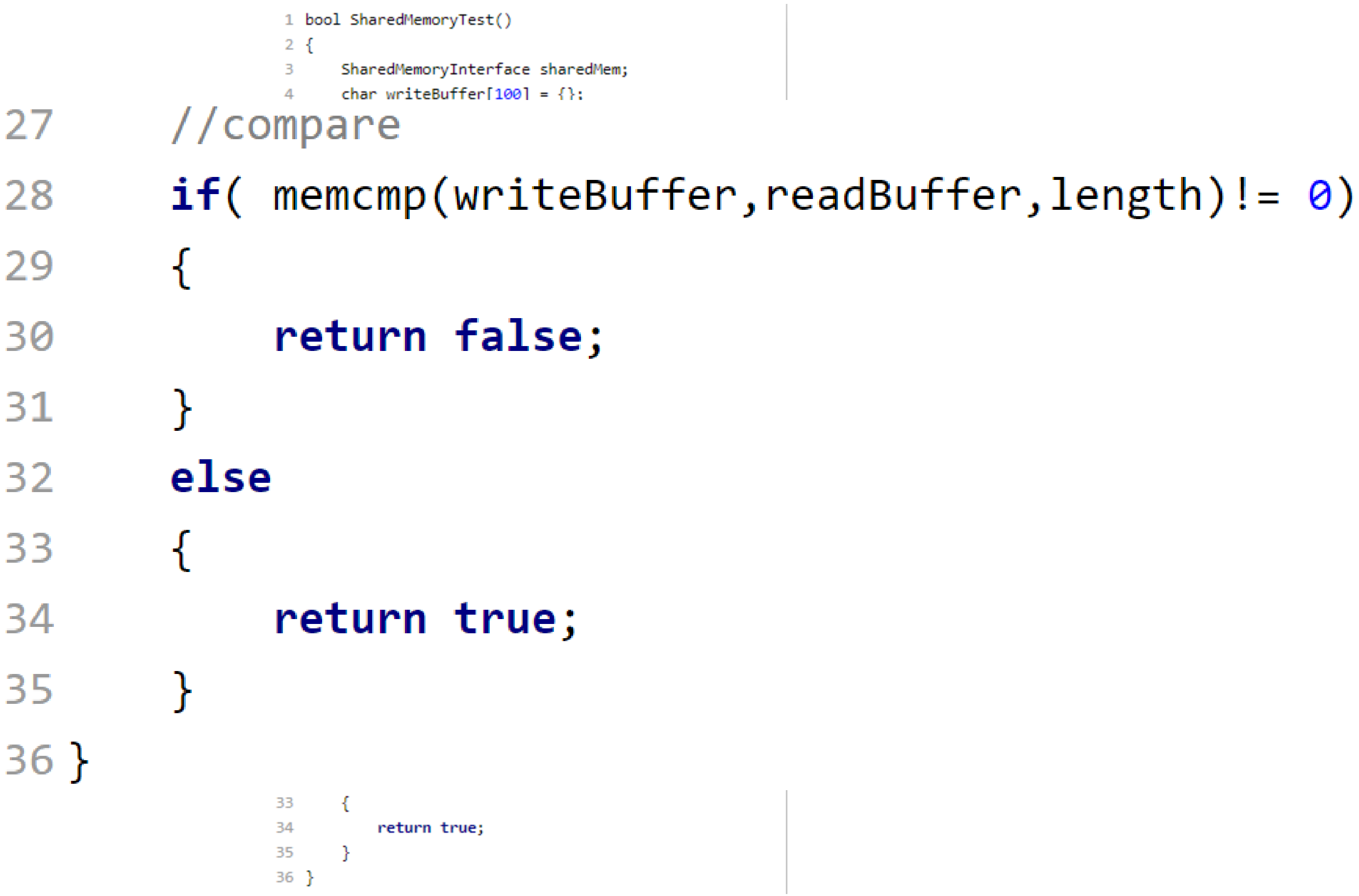
Layer Separation



```
bool SharedMemoryTest()
//prepare data to send
for(int i=0; i<100;i++)
{
    writeBuffer[i] = i;
}
//Init
sharedMem.Init();

//write
sharedMem.Write(writeBuffer,100);

//read
length = sharedMem.Read(readBuffer,100);
}
```

Read

Layer Separation



```
bool SharedMemoryTest()
{
    SharedMemoryInterface sharedMem;
    char writeBuffer[100] = {};
```

Compare →

```
    //compare
    if( memcmp(writeBuffer,readBuffer,length)!= 0)
    {
        return false;
    }
    else
    {
        return true;
    }
}
```

```
    {
        return true;
    }
}
```

Highly Effective:

- ⊡ Testing customer interfaces
- ⊡ Exemplary API usage

Highly Effective:

- ⊡ Testing customer interfaces
- ⊡ Exemplary API usage

Highly Effective:

- Testing customer interfaces
- Exemplary API usage

Motivation for layer separation:

⊡ Simplifies testing

⊡ Promotes cleaner code

⊡ Allows hardware/driver replacement without application changes

Motivation for layer separation:

- Simplifies testing
- Promotes cleaner code
- Allows hardware/driver replacement without application changes

Layer Separation

Motivation for layer separation:

- ⊡ Simplifies testing
- ⊡ Promotes cleaner code
- ⊡ Allows hardware/driver replacement without application changes

Layer Separation

Motivation for layer separation:

- Simplifies testing
- Promotes cleaner code
- Allows hardware/driver replacement without application changes

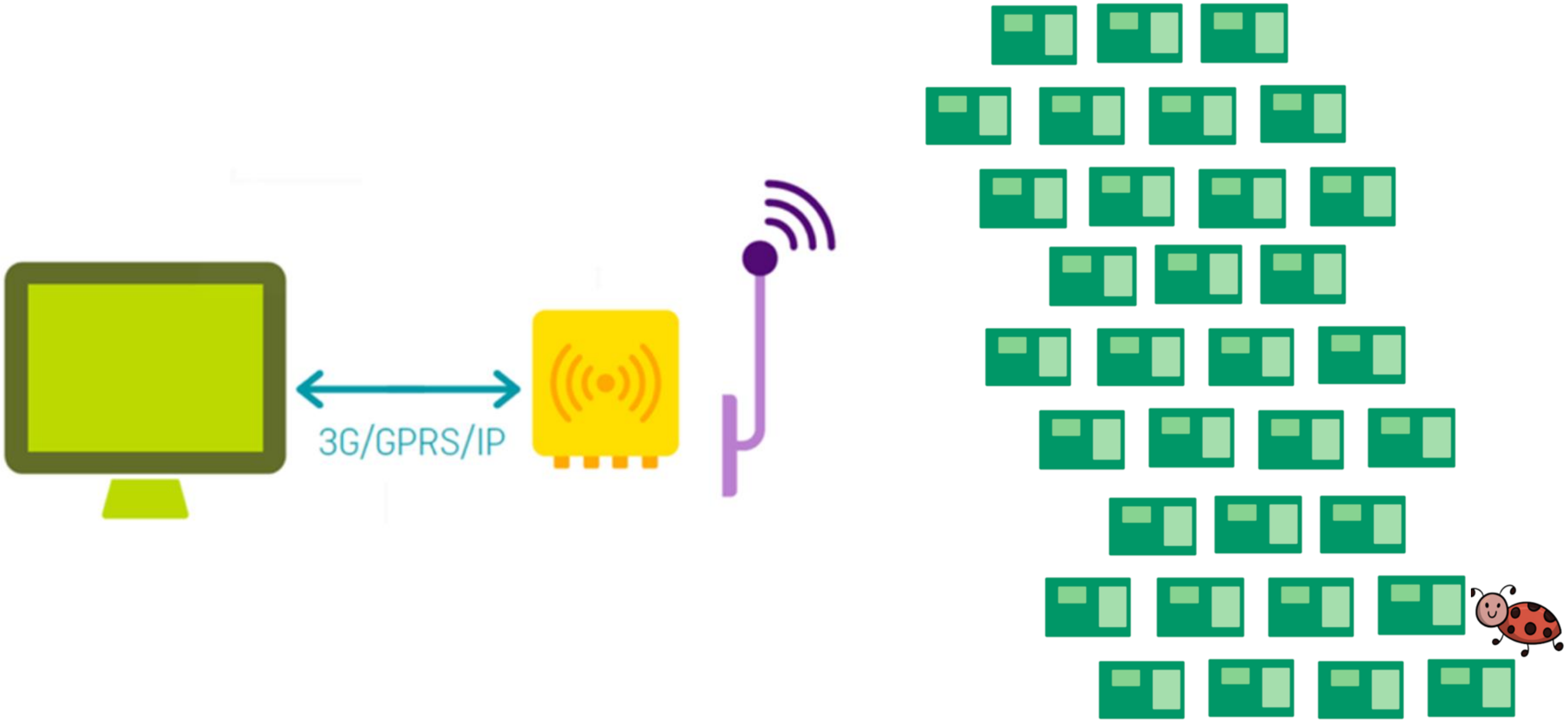
“

Separate the logic layer from the hardware layer

Agenda

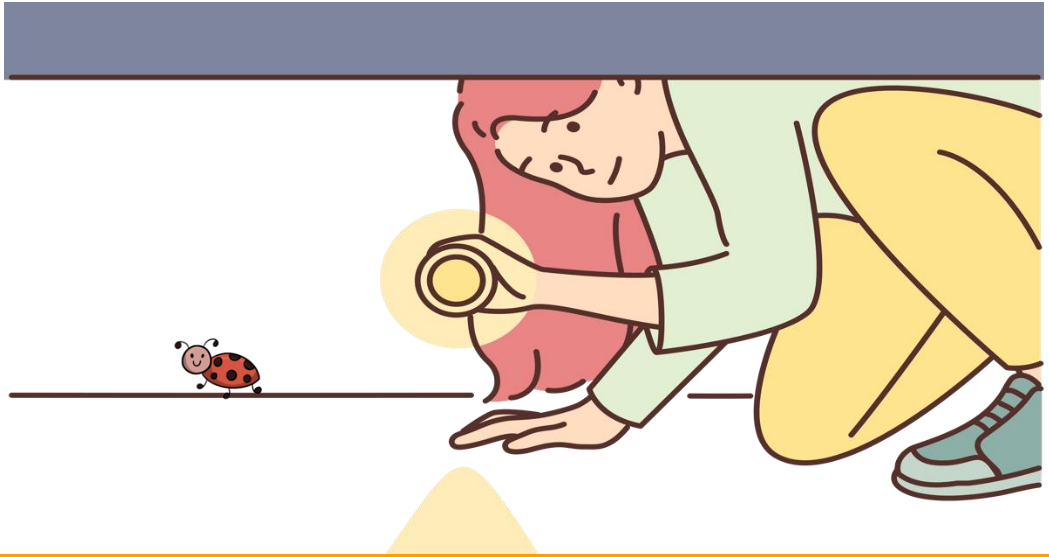
- □ Operating Systems
- □ Threads
- □ Layer Separation
- □ Network Problems
- □ Message Structure
- □ Simulators
- □ Logs
- □ Monitoring

Network Problems



Worked fine most of the time

Sometimes data was lost



Data for transmission remains in RAM, awaiting further processing.

Data for transmission remains in RAM, awaiting further processing.

↑

So, what is the problem with that?

Data for transmission remains in RAM, awaiting further processing.

In case of unstable communication:

→ Start to aggregate - takes a lot of space.

→ Loss of data in case of reset.

.

Disconnect the Logic from the Network

Disconnect the Logic from the Network

Thread #1 → Logic

Thread #2 → Sending

Disconnect the Logic from the Network

Thread #1 → Logic

Thread #2 → Sending

Always execute the same logic and store the results in nonvolatile memory (disk) - regardless of the current network status.

Implementation Achievements:

Implementation Achievements:

- Maximum data loss is now limited.
- Not being sensitive any more to network errors.

Implementation Achievements:

- Maximum data loss is now limited.
- Not being sensitive any more to network errors.

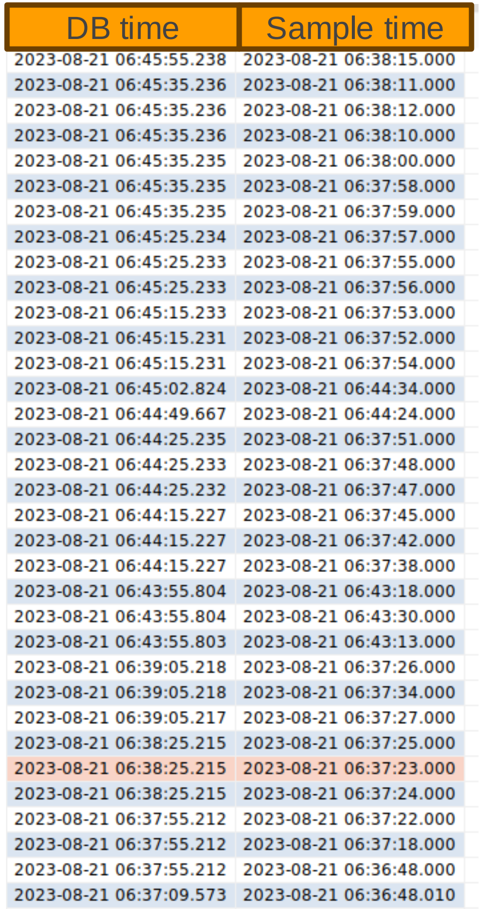
Implementation Achievements:

- Maximum data loss is now limited.
- Not being sensitive any more to network errors.

⇩

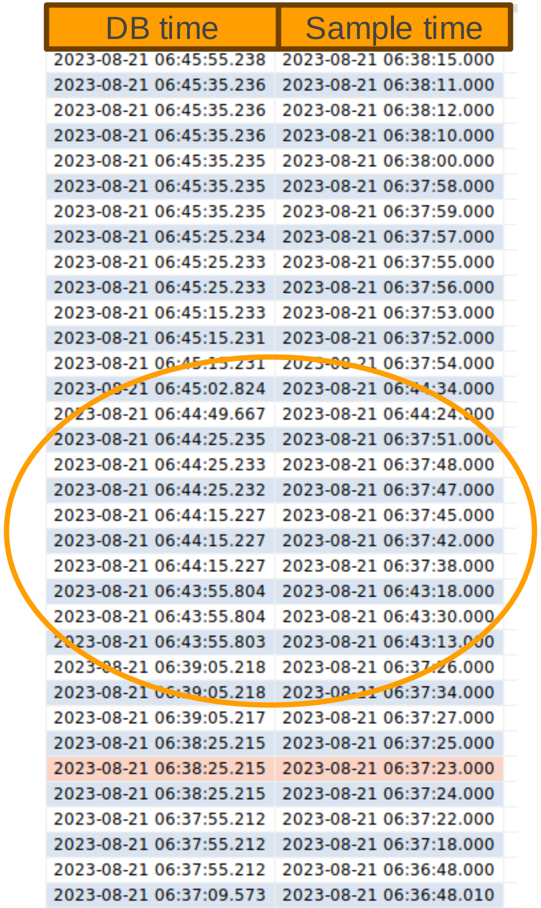
After implementing this change, we no longer experienced any data loss.

Second example

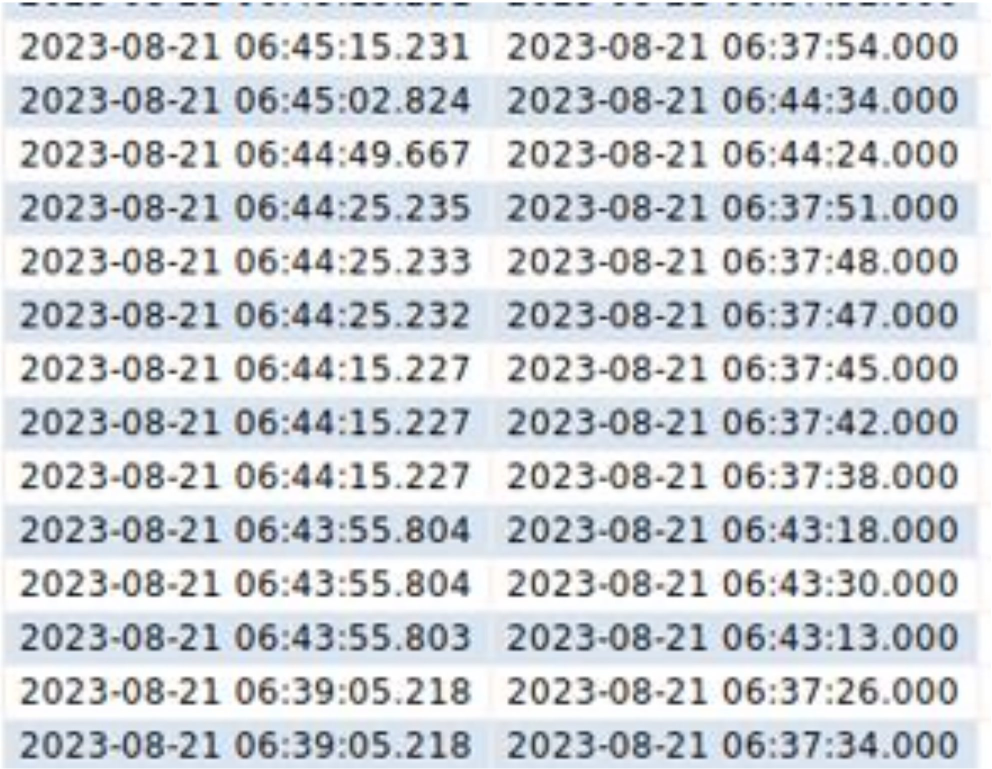


DB time	Sample time
2023-08-21 06:45:55.238	2023-08-21 06:38:15.000
2023-08-21 06:45:35.236	2023-08-21 06:38:11.000
2023-08-21 06:45:35.236	2023-08-21 06:38:12.000
2023-08-21 06:45:35.236	2023-08-21 06:38:10.000
2023-08-21 06:45:35.235	2023-08-21 06:38:00.000
2023-08-21 06:45:35.235	2023-08-21 06:37:58.000
2023-08-21 06:45:35.235	2023-08-21 06:37:59.000
2023-08-21 06:45:25.234	2023-08-21 06:37:57.000
2023-08-21 06:45:25.233	2023-08-21 06:37:55.000
2023-08-21 06:45:25.233	2023-08-21 06:37:56.000
2023-08-21 06:45:15.233	2023-08-21 06:37:53.000
2023-08-21 06:45:15.231	2023-08-21 06:37:52.000
2023-08-21 06:45:15.231	2023-08-21 06:37:54.000
2023-08-21 06:45:02.824	2023-08-21 06:44:34.000
2023-08-21 06:44:49.667	2023-08-21 06:44:24.000
2023-08-21 06:44:25.235	2023-08-21 06:37:51.000
2023-08-21 06:44:25.233	2023-08-21 06:37:48.000
2023-08-21 06:44:25.232	2023-08-21 06:37:47.000
2023-08-21 06:44:15.227	2023-08-21 06:37:45.000
2023-08-21 06:44:15.227	2023-08-21 06:37:42.000
2023-08-21 06:44:15.227	2023-08-21 06:37:38.000
2023-08-21 06:43:55.804	2023-08-21 06:43:18.000
2023-08-21 06:43:55.804	2023-08-21 06:43:30.000
2023-08-21 06:43:55.803	2023-08-21 06:43:13.000
2023-08-21 06:39:05.218	2023-08-21 06:37:26.000
2023-08-21 06:39:05.218	2023-08-21 06:37:34.000
2023-08-21 06:39:05.217	2023-08-21 06:37:27.000
2023-08-21 06:38:25.215	2023-08-21 06:37:25.000
2023-08-21 06:38:25.215	2023-08-21 06:37:23.000
2023-08-21 06:38:25.215	2023-08-21 06:37:24.000
2023-08-21 06:37:55.212	2023-08-21 06:37:22.000
2023-08-21 06:37:55.212	2023-08-21 06:37:18.000
2023-08-21 06:37:55.212	2023-08-21 06:36:48.000
2023-08-21 06:37:09.573	2023-08-21 06:36:48.010

Second example

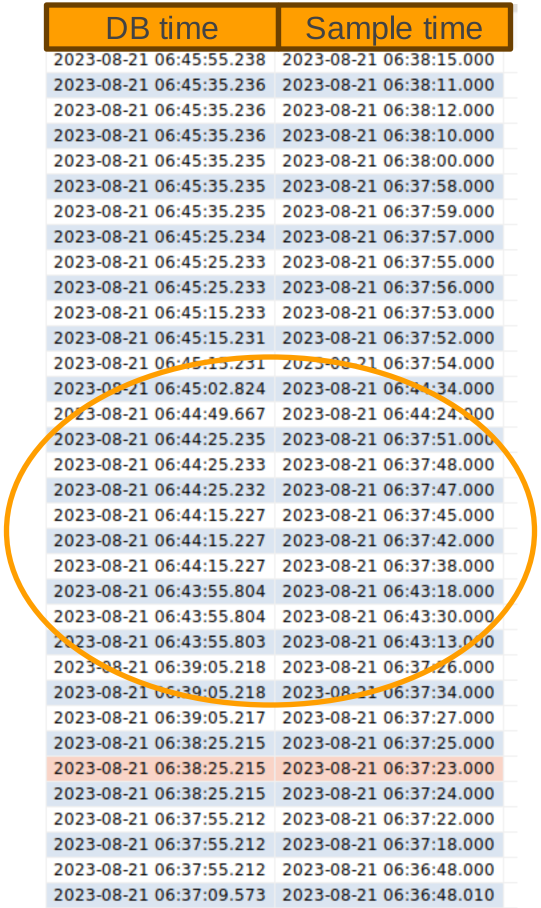


DB time	Sample time
2023-08-21 06:45:55.238	2023-08-21 06:38:15.000
2023-08-21 06:45:35.236	2023-08-21 06:38:11.000
2023-08-21 06:45:35.236	2023-08-21 06:38:12.000
2023-08-21 06:45:35.236	2023-08-21 06:38:10.000
2023-08-21 06:45:35.235	2023-08-21 06:38:00.000
2023-08-21 06:45:35.235	2023-08-21 06:37:58.000
2023-08-21 06:45:35.235	2023-08-21 06:37:59.000
2023-08-21 06:45:25.234	2023-08-21 06:37:57.000
2023-08-21 06:45:25.233	2023-08-21 06:37:55.000
2023-08-21 06:45:25.233	2023-08-21 06:37:56.000
2023-08-21 06:45:15.233	2023-08-21 06:37:53.000
2023-08-21 06:45:15.231	2023-08-21 06:37:52.000
2023-08-21 06:45:15.231	2023-08-21 06:37:54.000
2023-08-21 06:45:02.824	2023-08-21 06:44:34.000
2023-08-21 06:44:49.667	2023-08-21 06:44:24.000
2023-08-21 06:44:25.235	2023-08-21 06:37:51.000
2023-08-21 06:44:25.233	2023-08-21 06:37:48.000
2023-08-21 06:44:25.232	2023-08-21 06:37:47.000
2023-08-21 06:44:15.227	2023-08-21 06:37:45.000
2023-08-21 06:44:15.227	2023-08-21 06:37:42.000
2023-08-21 06:44:15.227	2023-08-21 06:37:38.000
2023-08-21 06:43:55.804	2023-08-21 06:43:18.000
2023-08-21 06:43:55.804	2023-08-21 06:43:30.000
2023-08-21 06:43:55.803	2023-08-21 06:43:13.000
2023-08-21 06:39:05.218	2023-08-21 06:37:26.000
2023-08-21 06:39:05.218	2023-08-21 06:37:34.000
2023-08-21 06:39:05.217	2023-08-21 06:37:27.000
2023-08-21 06:38:25.215	2023-08-21 06:37:25.000
2023-08-21 06:38:25.215	2023-08-21 06:37:23.000
2023-08-21 06:38:25.215	2023-08-21 06:37:24.000
2023-08-21 06:37:55.212	2023-08-21 06:37:22.000
2023-08-21 06:37:55.212	2023-08-21 06:37:18.000
2023-08-21 06:37:55.212	2023-08-21 06:36:48.000
2023-08-21 06:37:09.573	2023-08-21 06:36:48.010

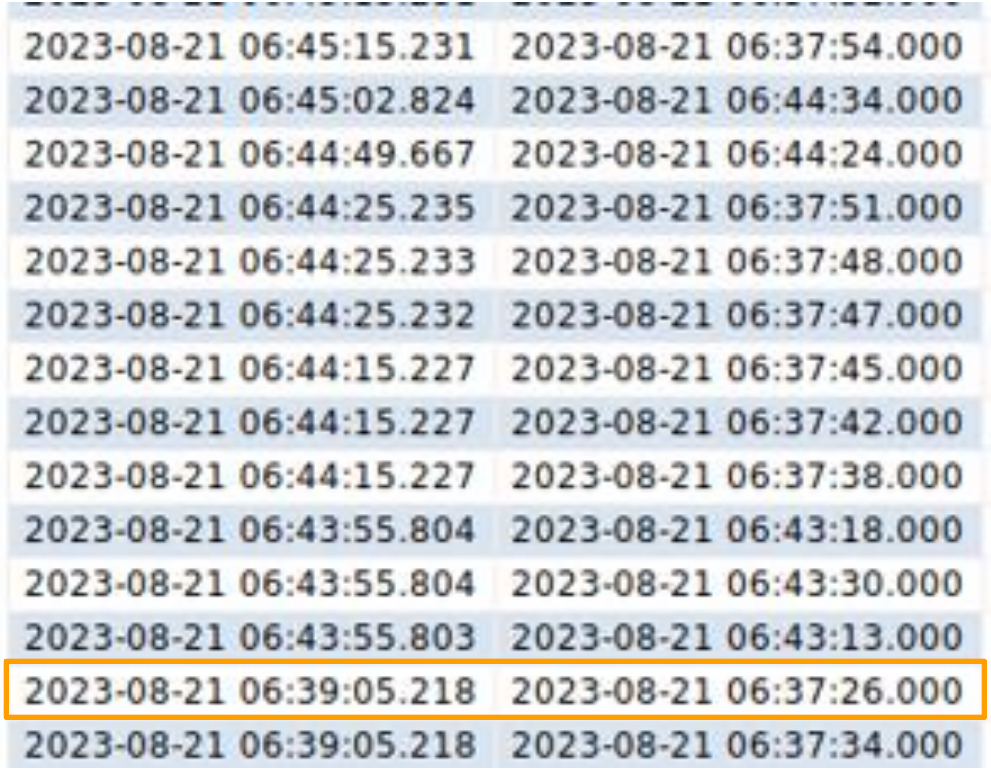


2023-08-21 06:45:15.231	2023-08-21 06:37:54.000
2023-08-21 06:45:02.824	2023-08-21 06:44:34.000
2023-08-21 06:44:49.667	2023-08-21 06:44:24.000
2023-08-21 06:44:25.235	2023-08-21 06:37:51.000
2023-08-21 06:44:25.233	2023-08-21 06:37:48.000
2023-08-21 06:44:25.232	2023-08-21 06:37:47.000
2023-08-21 06:44:15.227	2023-08-21 06:37:45.000
2023-08-21 06:44:15.227	2023-08-21 06:37:42.000
2023-08-21 06:44:15.227	2023-08-21 06:37:38.000
2023-08-21 06:43:55.804	2023-08-21 06:43:18.000
2023-08-21 06:43:55.804	2023-08-21 06:43:30.000
2023-08-21 06:43:55.803	2023-08-21 06:43:13.000
2023-08-21 06:39:05.218	2023-08-21 06:37:26.000
2023-08-21 06:39:05.218	2023-08-21 06:37:34.000

Second example

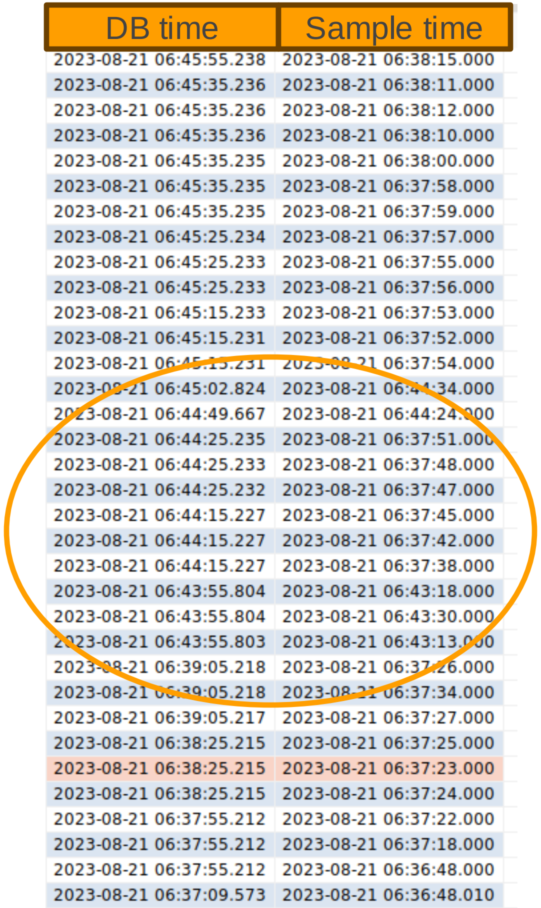


DB time	Sample time
2023-08-21 06:45:55.238	2023-08-21 06:38:15.000
2023-08-21 06:45:35.236	2023-08-21 06:38:11.000
2023-08-21 06:45:35.236	2023-08-21 06:38:12.000
2023-08-21 06:45:35.236	2023-08-21 06:38:10.000
2023-08-21 06:45:35.235	2023-08-21 06:38:00.000
2023-08-21 06:45:35.235	2023-08-21 06:37:58.000
2023-08-21 06:45:35.235	2023-08-21 06:37:59.000
2023-08-21 06:45:25.234	2023-08-21 06:37:57.000
2023-08-21 06:45:25.233	2023-08-21 06:37:55.000
2023-08-21 06:45:25.233	2023-08-21 06:37:56.000
2023-08-21 06:45:15.233	2023-08-21 06:37:53.000
2023-08-21 06:45:15.231	2023-08-21 06:37:52.000
2023-08-21 06:45:15.231	2023-08-21 06:37:54.000
2023-08-21 06:45:02.824	2023-08-21 06:44:34.000
2023-08-21 06:44:49.667	2023-08-21 06:44:24.000
2023-08-21 06:44:25.235	2023-08-21 06:37:51.000
2023-08-21 06:44:25.233	2023-08-21 06:37:48.000
2023-08-21 06:44:25.232	2023-08-21 06:37:47.000
2023-08-21 06:44:15.227	2023-08-21 06:37:45.000
2023-08-21 06:44:15.227	2023-08-21 06:37:42.000
2023-08-21 06:44:15.227	2023-08-21 06:37:38.000
2023-08-21 06:43:55.804	2023-08-21 06:43:18.000
2023-08-21 06:43:55.804	2023-08-21 06:43:30.000
2023-08-21 06:43:55.803	2023-08-21 06:43:13.000
2023-08-21 06:39:05.218	2023-08-21 06:37:26.000
2023-08-21 06:39:05.218	2023-08-21 06:37:34.000
2023-08-21 06:39:05.217	2023-08-21 06:37:27.000
2023-08-21 06:38:25.215	2023-08-21 06:37:25.000
2023-08-21 06:38:25.215	2023-08-21 06:37:23.000
2023-08-21 06:38:25.215	2023-08-21 06:37:24.000
2023-08-21 06:37:55.212	2023-08-21 06:37:22.000
2023-08-21 06:37:55.212	2023-08-21 06:37:18.000
2023-08-21 06:37:55.212	2023-08-21 06:36:48.000
2023-08-21 06:37:09.573	2023-08-21 06:36:48.010

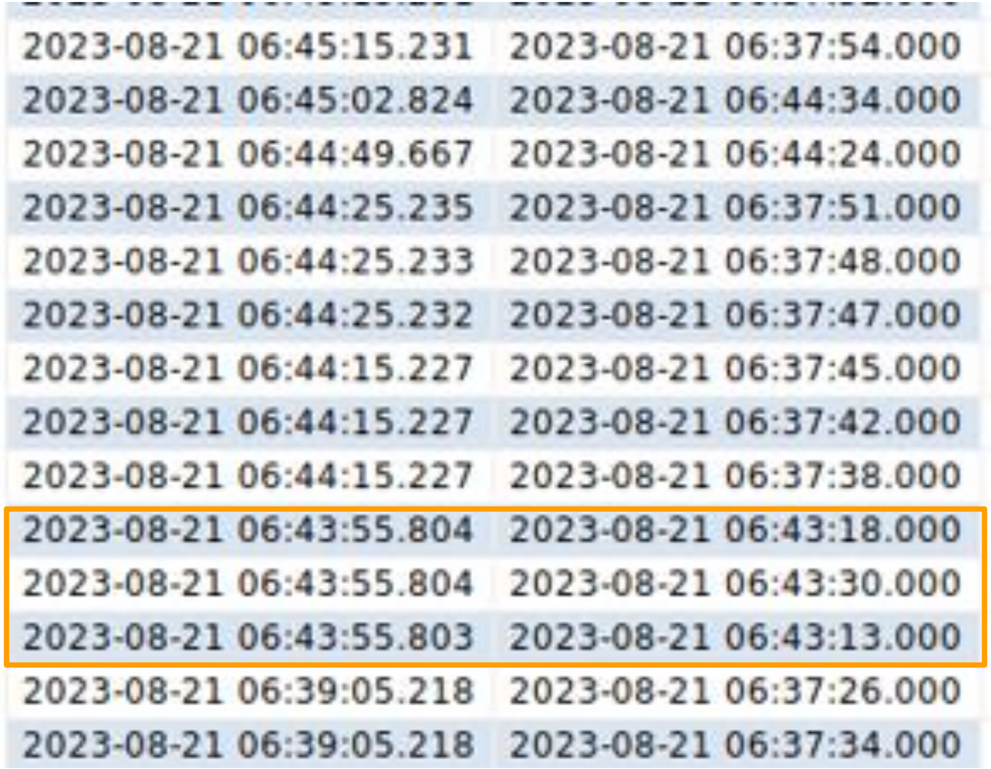


2023-08-21 06:45:15.231	2023-08-21 06:37:54.000
2023-08-21 06:45:02.824	2023-08-21 06:44:34.000
2023-08-21 06:44:49.667	2023-08-21 06:44:24.000
2023-08-21 06:44:25.235	2023-08-21 06:37:51.000
2023-08-21 06:44:25.233	2023-08-21 06:37:48.000
2023-08-21 06:44:25.232	2023-08-21 06:37:47.000
2023-08-21 06:44:15.227	2023-08-21 06:37:45.000
2023-08-21 06:44:15.227	2023-08-21 06:37:42.000
2023-08-21 06:44:15.227	2023-08-21 06:37:38.000
2023-08-21 06:43:55.804	2023-08-21 06:43:18.000
2023-08-21 06:43:55.804	2023-08-21 06:43:30.000
2023-08-21 06:43:55.803	2023-08-21 06:43:13.000
2023-08-21 06:39:05.218	2023-08-21 06:37:26.000
2023-08-21 06:39:05.218	2023-08-21 06:37:34.000

Second example

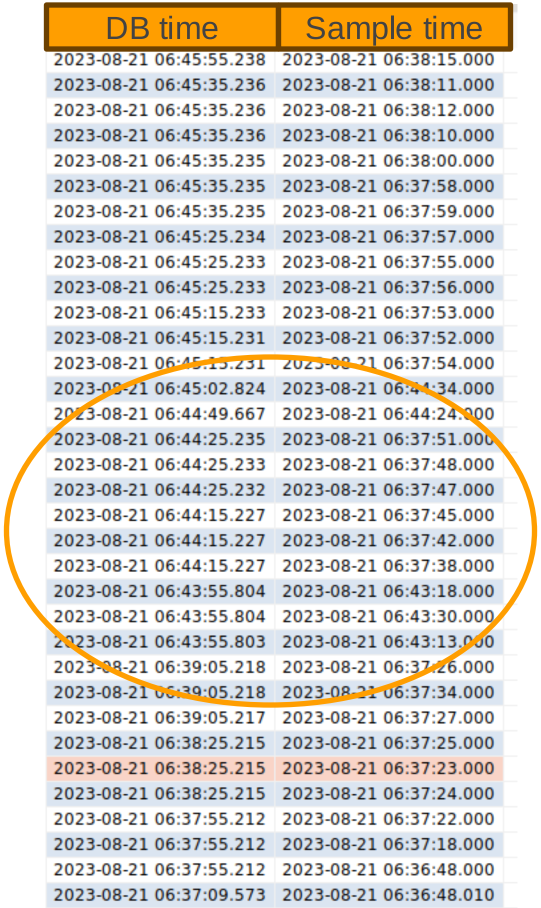


DB time	Sample time
2023-08-21 06:45:55.238	2023-08-21 06:38:15.000
2023-08-21 06:45:35.236	2023-08-21 06:38:11.000
2023-08-21 06:45:35.236	2023-08-21 06:38:12.000
2023-08-21 06:45:35.236	2023-08-21 06:38:10.000
2023-08-21 06:45:35.235	2023-08-21 06:38:00.000
2023-08-21 06:45:35.235	2023-08-21 06:37:58.000
2023-08-21 06:45:35.235	2023-08-21 06:37:59.000
2023-08-21 06:45:25.234	2023-08-21 06:37:57.000
2023-08-21 06:45:25.233	2023-08-21 06:37:55.000
2023-08-21 06:45:25.233	2023-08-21 06:37:56.000
2023-08-21 06:45:15.233	2023-08-21 06:37:53.000
2023-08-21 06:45:15.231	2023-08-21 06:37:52.000
2023-08-21 06:45:15.231	2023-08-21 06:37:54.000
2023-08-21 06:45:02.824	2023-08-21 06:44:34.000
2023-08-21 06:44:49.667	2023-08-21 06:44:24.000
2023-08-21 06:44:25.235	2023-08-21 06:37:51.000
2023-08-21 06:44:25.233	2023-08-21 06:37:48.000
2023-08-21 06:44:25.232	2023-08-21 06:37:47.000
2023-08-21 06:44:15.227	2023-08-21 06:37:45.000
2023-08-21 06:44:15.227	2023-08-21 06:37:42.000
2023-08-21 06:44:15.227	2023-08-21 06:37:38.000
2023-08-21 06:43:55.804	2023-08-21 06:43:18.000
2023-08-21 06:43:55.804	2023-08-21 06:43:30.000
2023-08-21 06:43:55.803	2023-08-21 06:43:13.000
2023-08-21 06:39:05.218	2023-08-21 06:37:26.000
2023-08-21 06:39:05.218	2023-08-21 06:37:34.000
2023-08-21 06:39:05.217	2023-08-21 06:37:27.000
2023-08-21 06:38:25.215	2023-08-21 06:37:25.000
2023-08-21 06:38:25.215	2023-08-21 06:37:23.000
2023-08-21 06:38:25.215	2023-08-21 06:37:24.000
2023-08-21 06:37:55.212	2023-08-21 06:37:22.000
2023-08-21 06:37:55.212	2023-08-21 06:37:18.000
2023-08-21 06:37:55.212	2023-08-21 06:36:48.000
2023-08-21 06:37:09.573	2023-08-21 06:36:48.010

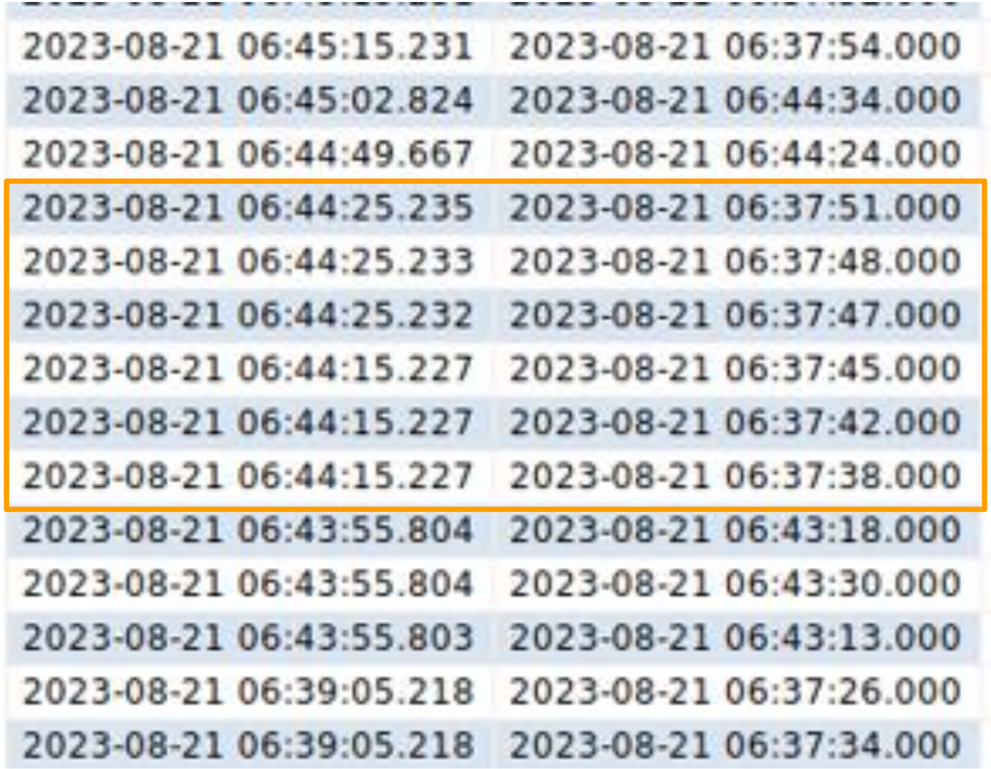


2023-08-21 06:45:15.231	2023-08-21 06:37:54.000
2023-08-21 06:45:02.824	2023-08-21 06:44:34.000
2023-08-21 06:44:49.667	2023-08-21 06:44:24.000
2023-08-21 06:44:25.235	2023-08-21 06:37:51.000
2023-08-21 06:44:25.233	2023-08-21 06:37:48.000
2023-08-21 06:44:25.232	2023-08-21 06:37:47.000
2023-08-21 06:44:15.227	2023-08-21 06:37:45.000
2023-08-21 06:44:15.227	2023-08-21 06:37:42.000
2023-08-21 06:44:15.227	2023-08-21 06:37:38.000
2023-08-21 06:43:55.804	2023-08-21 06:43:18.000
2023-08-21 06:43:55.804	2023-08-21 06:43:30.000
2023-08-21 06:43:55.803	2023-08-21 06:43:13.000
2023-08-21 06:39:05.218	2023-08-21 06:37:26.000
2023-08-21 06:39:05.218	2023-08-21 06:37:34.000

Second example

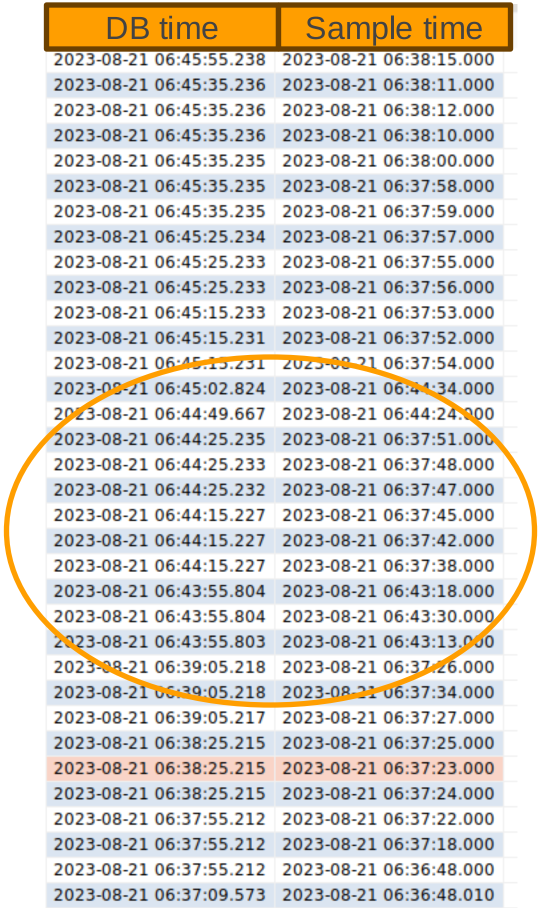


DB time	Sample time
2023-08-21 06:45:55.238	2023-08-21 06:38:15.000
2023-08-21 06:45:35.236	2023-08-21 06:38:11.000
2023-08-21 06:45:35.236	2023-08-21 06:38:12.000
2023-08-21 06:45:35.236	2023-08-21 06:38:10.000
2023-08-21 06:45:35.235	2023-08-21 06:38:00.000
2023-08-21 06:45:35.235	2023-08-21 06:37:58.000
2023-08-21 06:45:35.235	2023-08-21 06:37:59.000
2023-08-21 06:45:25.234	2023-08-21 06:37:57.000
2023-08-21 06:45:25.233	2023-08-21 06:37:55.000
2023-08-21 06:45:25.233	2023-08-21 06:37:56.000
2023-08-21 06:45:15.233	2023-08-21 06:37:53.000
2023-08-21 06:45:15.231	2023-08-21 06:37:52.000
2023-08-21 06:45:15.231	2023-08-21 06:37:54.000
2023-08-21 06:45:02.824	2023-08-21 06:44:34.000
2023-08-21 06:44:49.667	2023-08-21 06:44:24.000
2023-08-21 06:44:25.235	2023-08-21 06:37:51.000
2023-08-21 06:44:25.233	2023-08-21 06:37:48.000
2023-08-21 06:44:25.232	2023-08-21 06:37:47.000
2023-08-21 06:44:15.227	2023-08-21 06:37:45.000
2023-08-21 06:44:15.227	2023-08-21 06:37:42.000
2023-08-21 06:44:15.227	2023-08-21 06:37:38.000
2023-08-21 06:43:55.804	2023-08-21 06:43:18.000
2023-08-21 06:43:55.804	2023-08-21 06:43:30.000
2023-08-21 06:43:55.803	2023-08-21 06:43:13.000
2023-08-21 06:39:05.218	2023-08-21 06:37:26.000
2023-08-21 06:39:05.218	2023-08-21 06:37:34.000
2023-08-21 06:39:05.217	2023-08-21 06:37:27.000
2023-08-21 06:38:25.215	2023-08-21 06:37:25.000
2023-08-21 06:38:25.215	2023-08-21 06:37:23.000
2023-08-21 06:38:25.215	2023-08-21 06:37:24.000
2023-08-21 06:37:55.212	2023-08-21 06:37:22.000
2023-08-21 06:37:55.212	2023-08-21 06:37:18.000
2023-08-21 06:37:55.212	2023-08-21 06:36:48.000
2023-08-21 06:37:09.573	2023-08-21 06:36:48.010

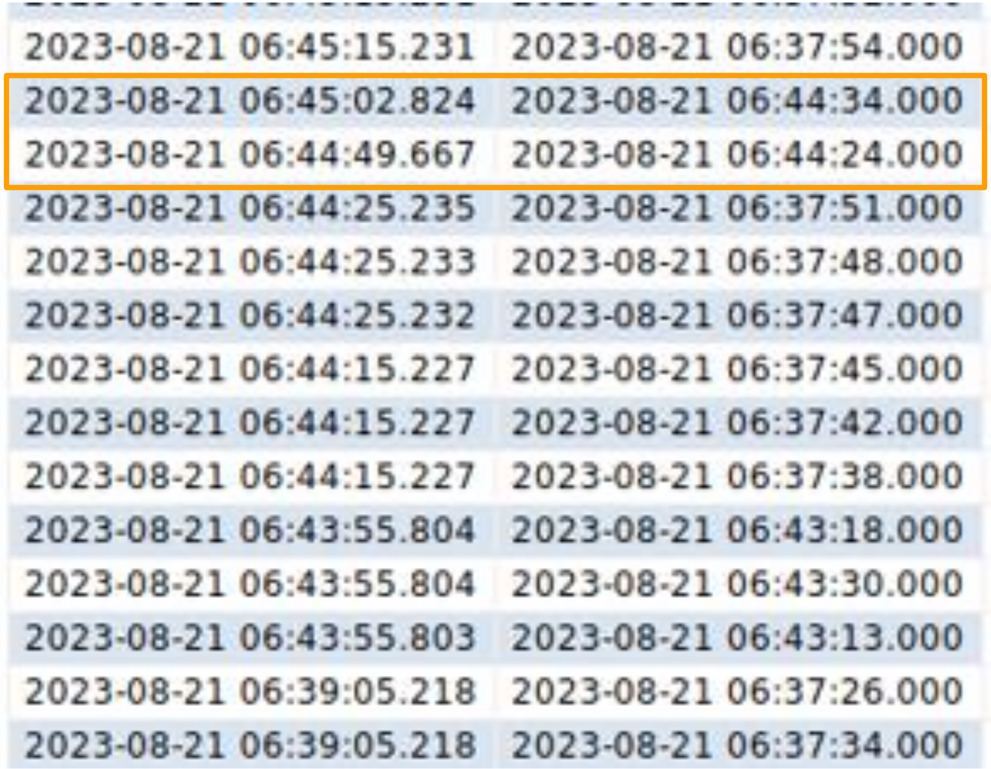


2023-08-21 06:45:15.231	2023-08-21 06:37:54.000
2023-08-21 06:45:02.824	2023-08-21 06:44:34.000
2023-08-21 06:44:49.667	2023-08-21 06:44:24.000
2023-08-21 06:44:25.235	2023-08-21 06:37:51.000
2023-08-21 06:44:25.233	2023-08-21 06:37:48.000
2023-08-21 06:44:25.232	2023-08-21 06:37:47.000
2023-08-21 06:44:15.227	2023-08-21 06:37:45.000
2023-08-21 06:44:15.227	2023-08-21 06:37:42.000
2023-08-21 06:44:15.227	2023-08-21 06:37:38.000
2023-08-21 06:43:55.804	2023-08-21 06:43:18.000
2023-08-21 06:43:55.804	2023-08-21 06:43:30.000
2023-08-21 06:43:55.803	2023-08-21 06:43:13.000
2023-08-21 06:39:05.218	2023-08-21 06:37:26.000
2023-08-21 06:39:05.218	2023-08-21 06:37:34.000

Second example



DB time	Sample time
2023-08-21 06:45:55.238	2023-08-21 06:38:15.000
2023-08-21 06:45:35.236	2023-08-21 06:38:11.000
2023-08-21 06:45:35.236	2023-08-21 06:38:12.000
2023-08-21 06:45:35.236	2023-08-21 06:38:10.000
2023-08-21 06:45:35.235	2023-08-21 06:38:00.000
2023-08-21 06:45:35.235	2023-08-21 06:37:58.000
2023-08-21 06:45:35.235	2023-08-21 06:37:59.000
2023-08-21 06:45:25.234	2023-08-21 06:37:57.000
2023-08-21 06:45:25.233	2023-08-21 06:37:55.000
2023-08-21 06:45:25.233	2023-08-21 06:37:56.000
2023-08-21 06:45:15.233	2023-08-21 06:37:53.000
2023-08-21 06:45:15.231	2023-08-21 06:37:52.000
2023-08-21 06:45:15.231	2023-08-21 06:37:54.000
2023-08-21 06:45:02.824	2023-08-21 06:44:34.000
2023-08-21 06:44:49.667	2023-08-21 06:44:24.000
2023-08-21 06:44:25.235	2023-08-21 06:37:51.000
2023-08-21 06:44:25.233	2023-08-21 06:37:48.000
2023-08-21 06:44:25.232	2023-08-21 06:37:47.000
2023-08-21 06:44:15.227	2023-08-21 06:37:45.000
2023-08-21 06:44:15.227	2023-08-21 06:37:42.000
2023-08-21 06:44:15.227	2023-08-21 06:37:38.000
2023-08-21 06:43:55.804	2023-08-21 06:43:18.000
2023-08-21 06:43:55.804	2023-08-21 06:43:30.000
2023-08-21 06:43:55.803	2023-08-21 06:43:13.000
2023-08-21 06:39:05.218	2023-08-21 06:37:26.000
2023-08-21 06:39:05.218	2023-08-21 06:37:34.000
2023-08-21 06:39:05.217	2023-08-21 06:37:27.000
2023-08-21 06:38:25.215	2023-08-21 06:37:25.000
2023-08-21 06:38:25.215	2023-08-21 06:37:23.000
2023-08-21 06:38:25.215	2023-08-21 06:37:24.000
2023-08-21 06:37:55.212	2023-08-21 06:37:22.000
2023-08-21 06:37:55.212	2023-08-21 06:37:18.000
2023-08-21 06:37:55.212	2023-08-21 06:36:48.000
2023-08-21 06:37:09.573	2023-08-21 06:36:48.010



2023-08-21 06:45:15.231	2023-08-21 06:37:54.000
2023-08-21 06:45:02.824	2023-08-21 06:44:34.000
2023-08-21 06:44:49.667	2023-08-21 06:44:24.000
2023-08-21 06:44:25.235	2023-08-21 06:37:51.000
2023-08-21 06:44:25.233	2023-08-21 06:37:48.000
2023-08-21 06:44:25.232	2023-08-21 06:37:47.000
2023-08-21 06:44:15.227	2023-08-21 06:37:45.000
2023-08-21 06:44:15.227	2023-08-21 06:37:42.000
2023-08-21 06:44:15.227	2023-08-21 06:37:38.000
2023-08-21 06:43:55.804	2023-08-21 06:43:18.000
2023-08-21 06:43:55.804	2023-08-21 06:43:30.000
2023-08-21 06:43:55.803	2023-08-21 06:43:13.000
2023-08-21 06:39:05.218	2023-08-21 06:37:26.000
2023-08-21 06:39:05.218	2023-08-21 06:37:34.000

What is the problem with that (Messages are not in order)?

- Hard to put logic on the received side
- Confusing, easy to miss without noticing

What is the problem with that (Messages are not in order)?

- Hard to put logic on the received side
- Confusing, easy to miss without noticing

What is the problem with that (Messages are not in order)?

- Hard to put logic on the received side
- Confusing, easy to miss without noticing

How to avoid it?

Use one queue to send data out from a specific interface

> *Disconnect the logic from the network*

Agenda

⊡ Operating Systems

⊡ Threads

⊡ Layer Separation

⊡ Network Problems

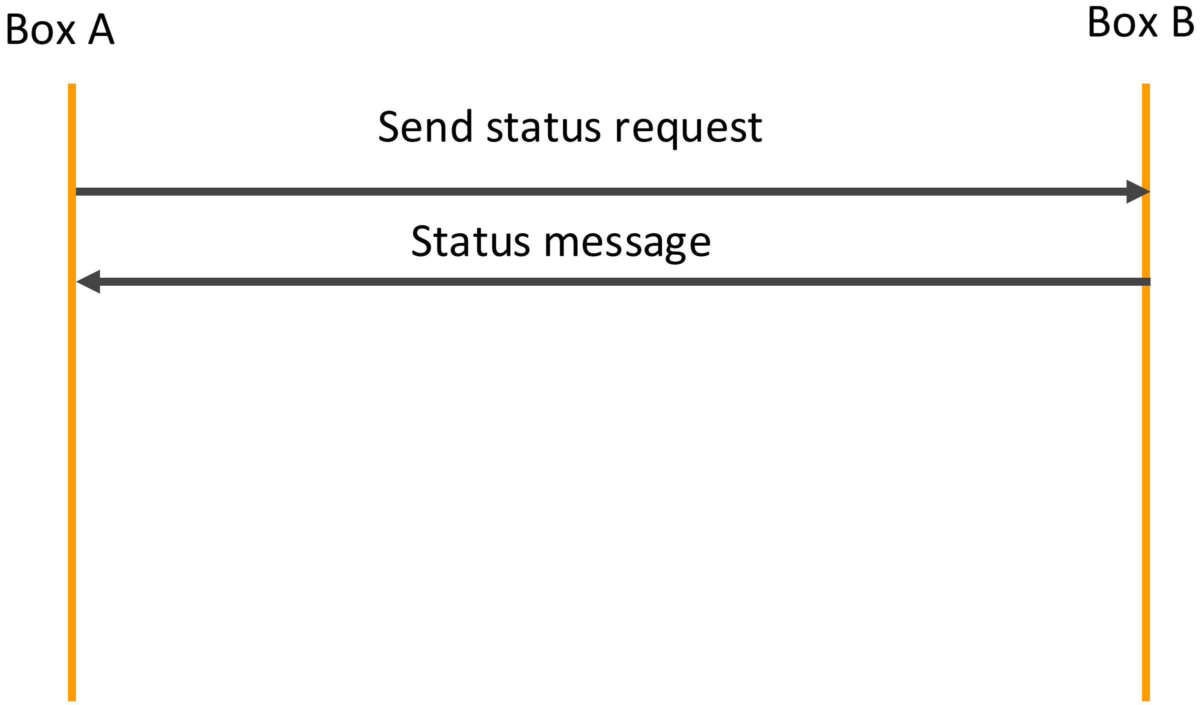
⊡ Message Structure

⊡ Simulators

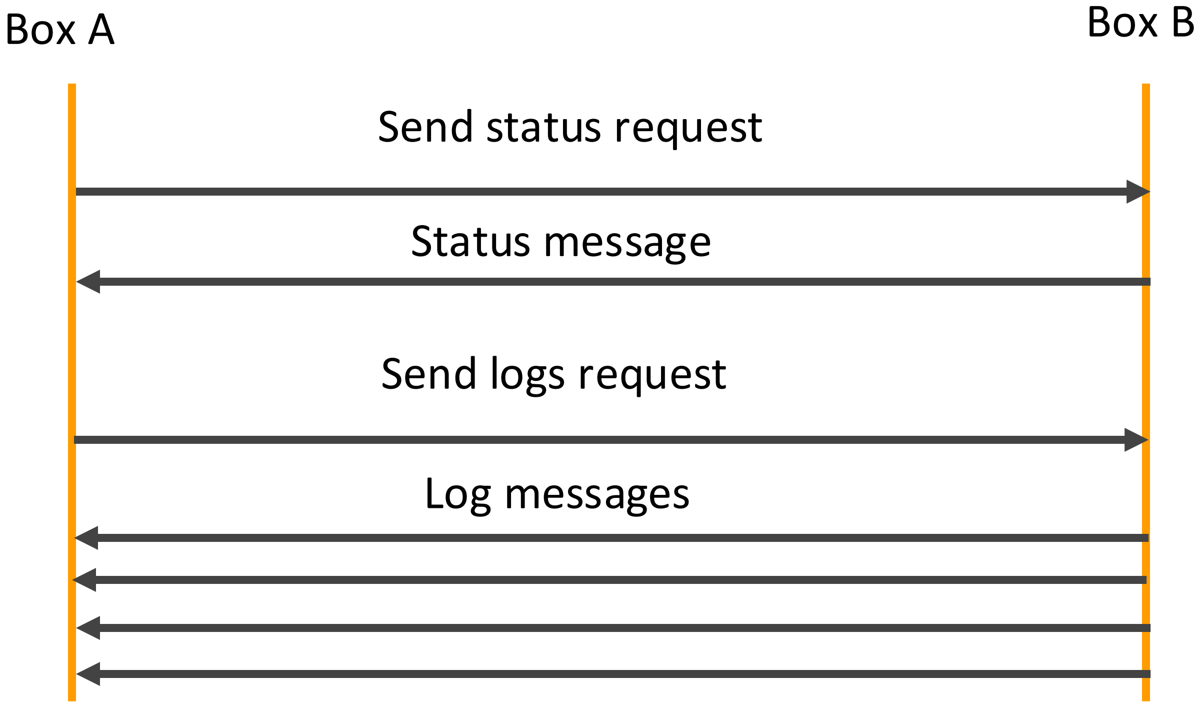
⊡ Logs

⊡ Monitoring

Messages Flow



Messages Flow



Box A

Box B

Send status request

Status message

Send logs request

Log messages

Depending on your low level
either a stream or packets

Message Structure

Depending on your low level

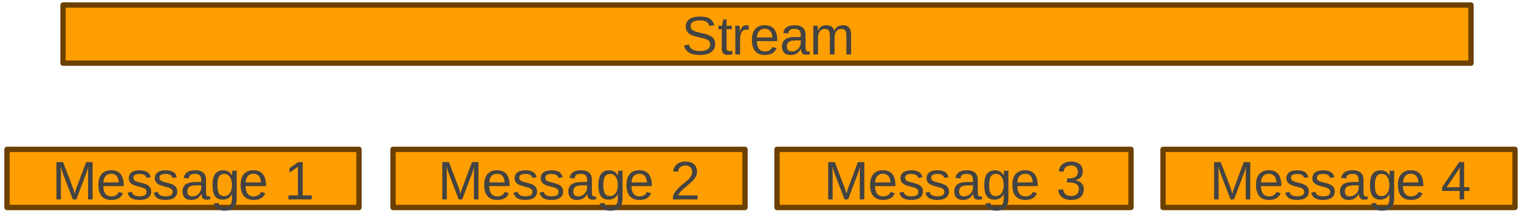
either a stream or packets

Stream

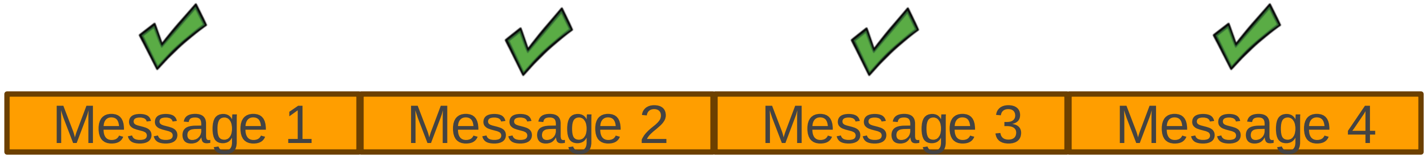
Message Structure

Depending on your low level

either a stream or packets



Potential problems with streaming



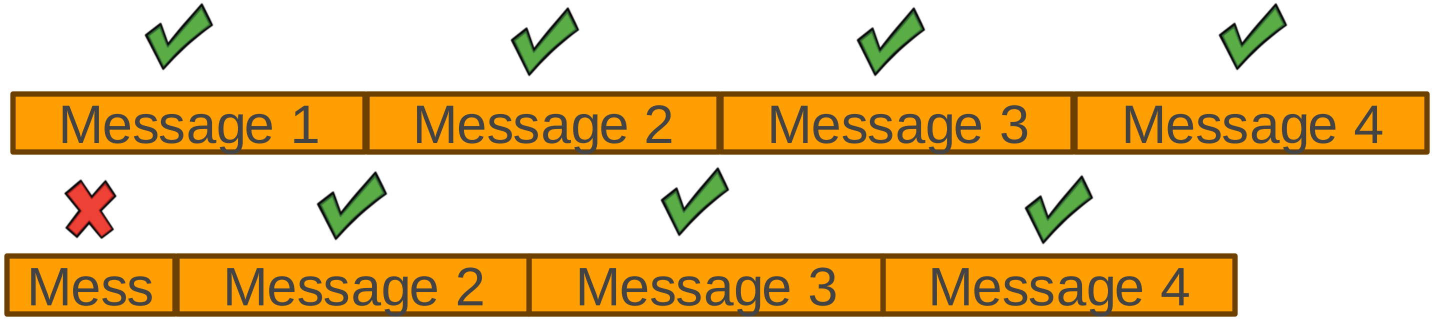
✓ Message 1

✓ Message 2

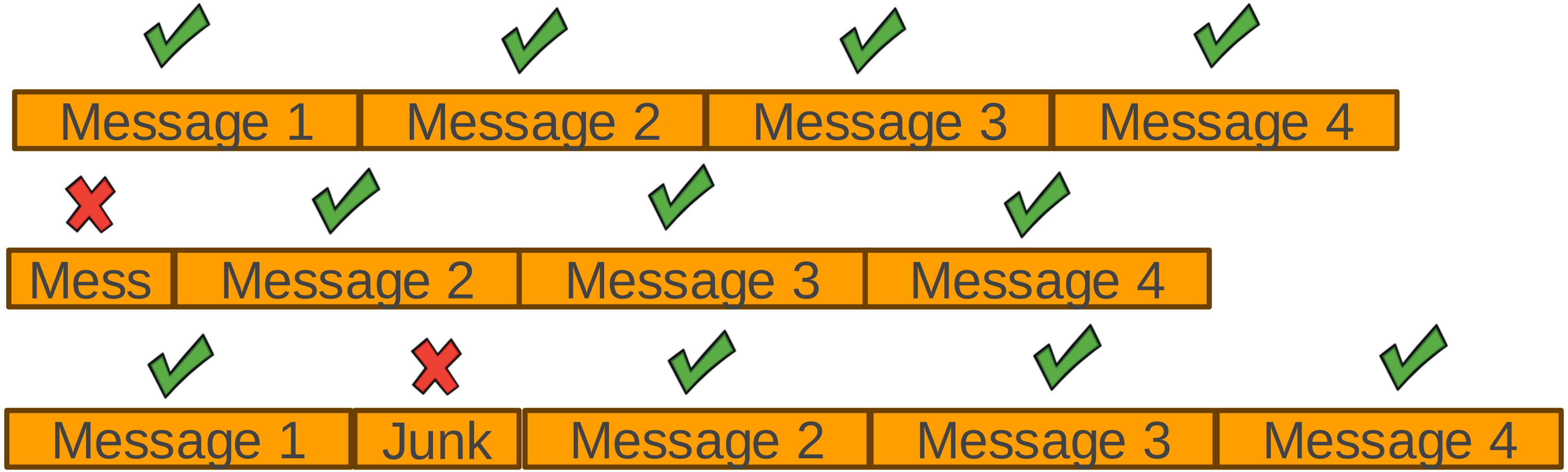
✓ Message 3

✓ Message 4

Potential problems with streaming



Potential problems with streaming

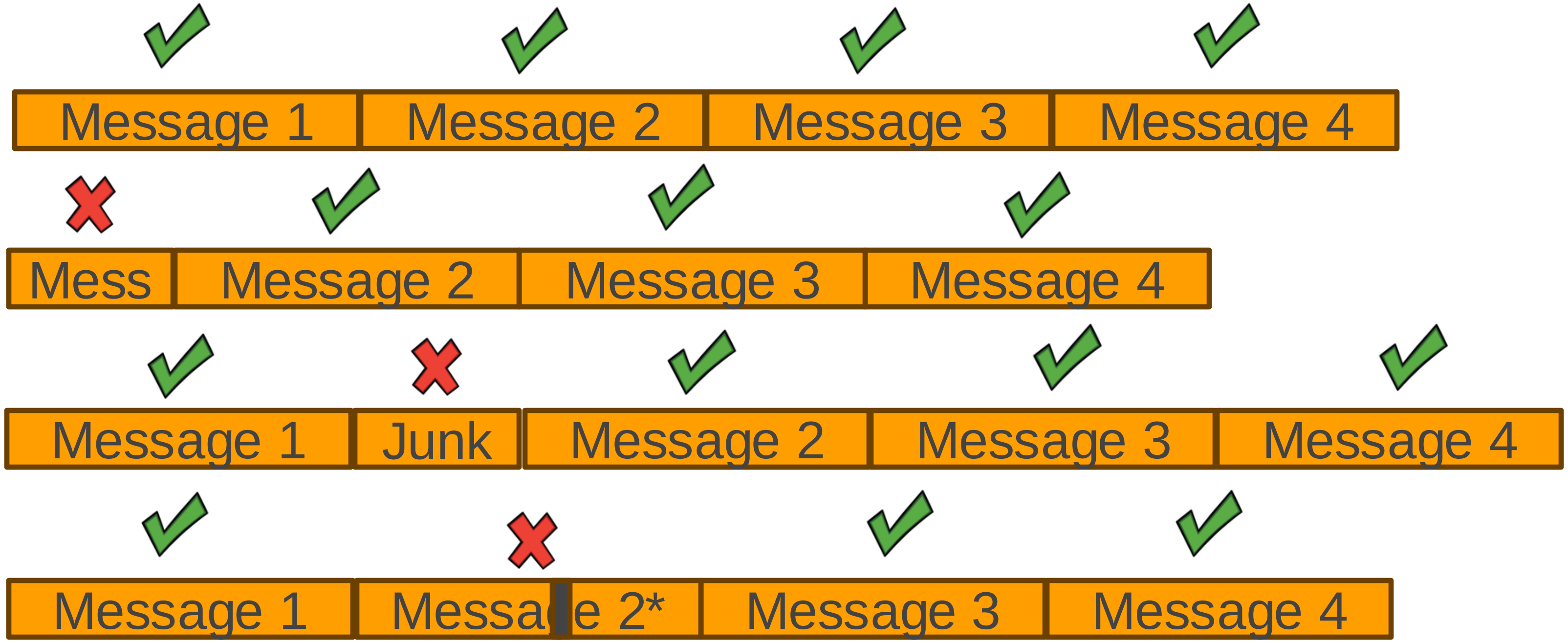


✓ Message 1 | ✓ Message 2 | ✓ Message 3 | ✓ Message 4

✗ Mess | ✓ Message 2 | ✓ Message 3 | ✓ Message 4

✓ Message 1 | ✗ Junk | ✓ Message 2 | ✓ Message 3 | ✓ Message 4

Potential problems with streaming



✓ ✓ ✓ ✓

Message 1 Message 2 Message 3 Message 4

✗ ✓ ✓ ✓

Mess Message 2 Message 3 Message 4

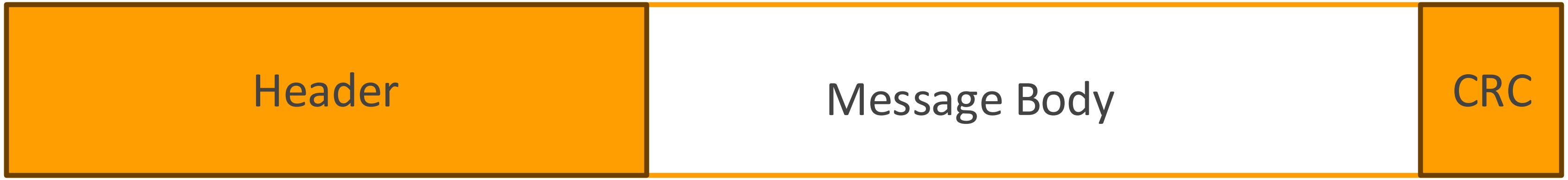
✓ ✗ ✓ ✓ ✓

Message 1 Junk Message 2 Message 3 Message 4

✓ ✗ ✓ ✓

Message 1 Message 2* Message 3 Message 4

Recommended Message Structure (Streaming)

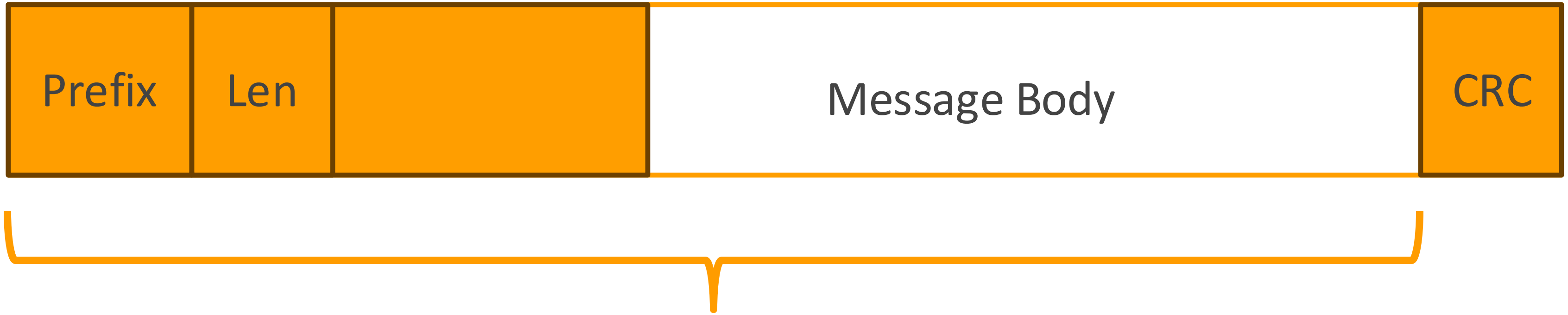


Header

Message Body

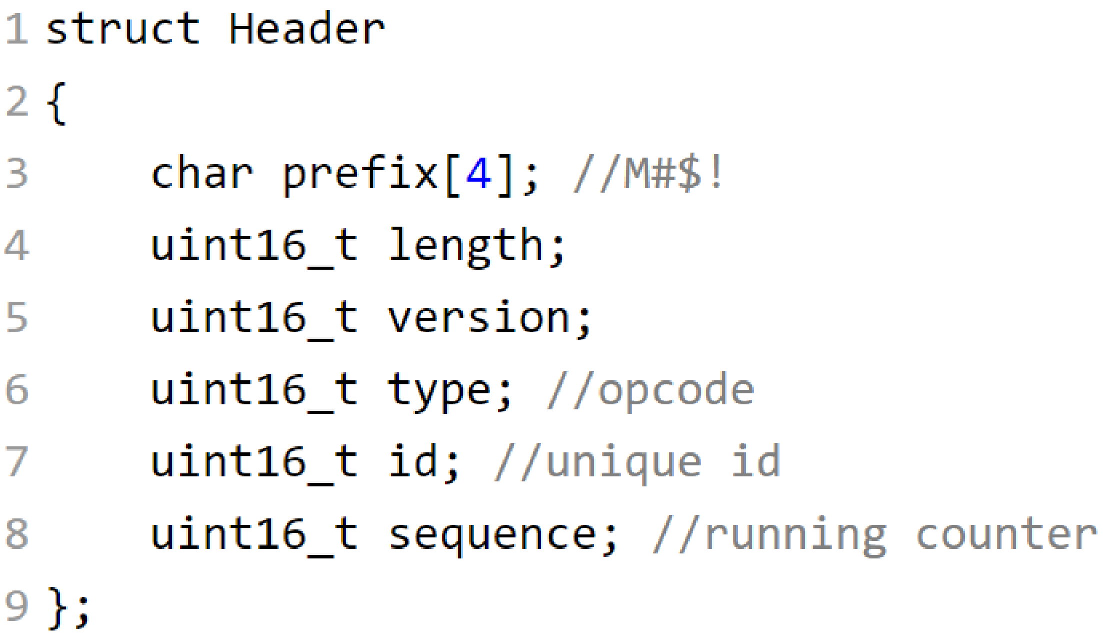
CRC

Message Structure



Verify this content is correct

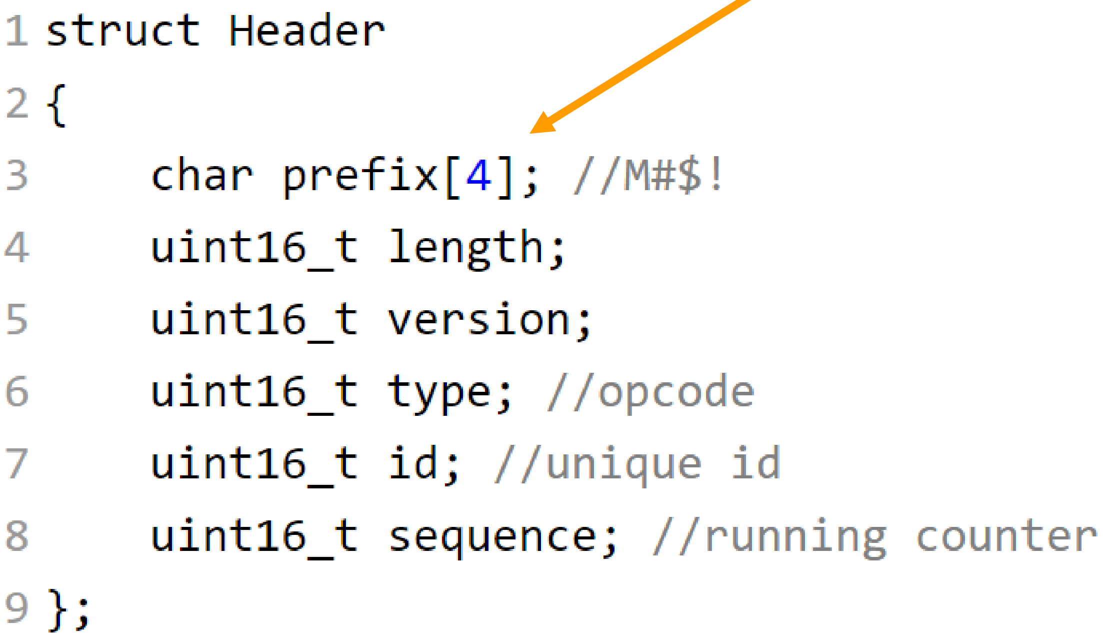
Message Structure



```
struct Header
{
    char prefix[4]; //M#$!
    uint16_t length;
    uint16_t version;
    uint16_t type; //opcode
    uint16_t id; //unique id
    uint16_t sequence; //running counter
};
```

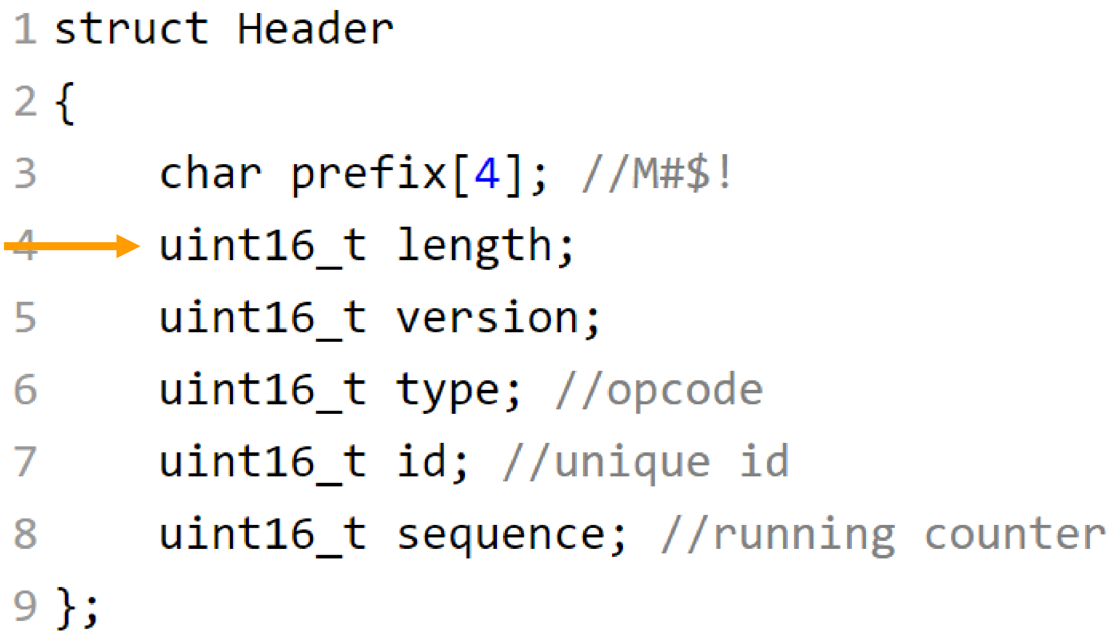
Message Structure

Easy to identify in memory



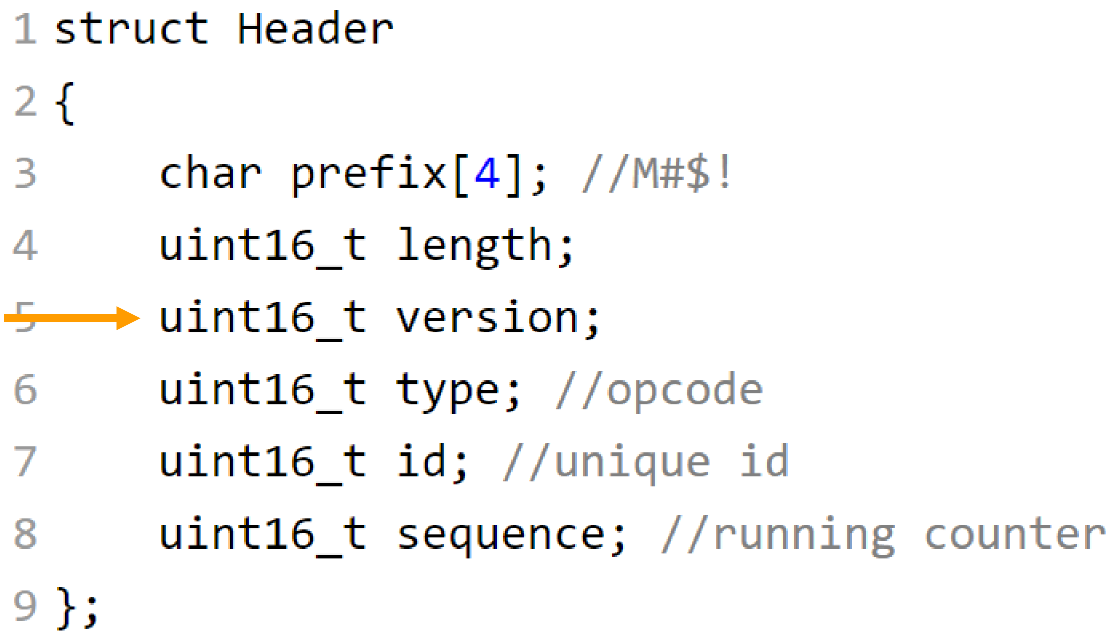
```
struct Header
{
    char prefix[4]; //M#$!
    uint16_t length;
    uint16_t version;
    uint16_t type; //opcode
    uint16_t id; //unique id
    uint16_t sequence; //running counter
};
```

Message Structure



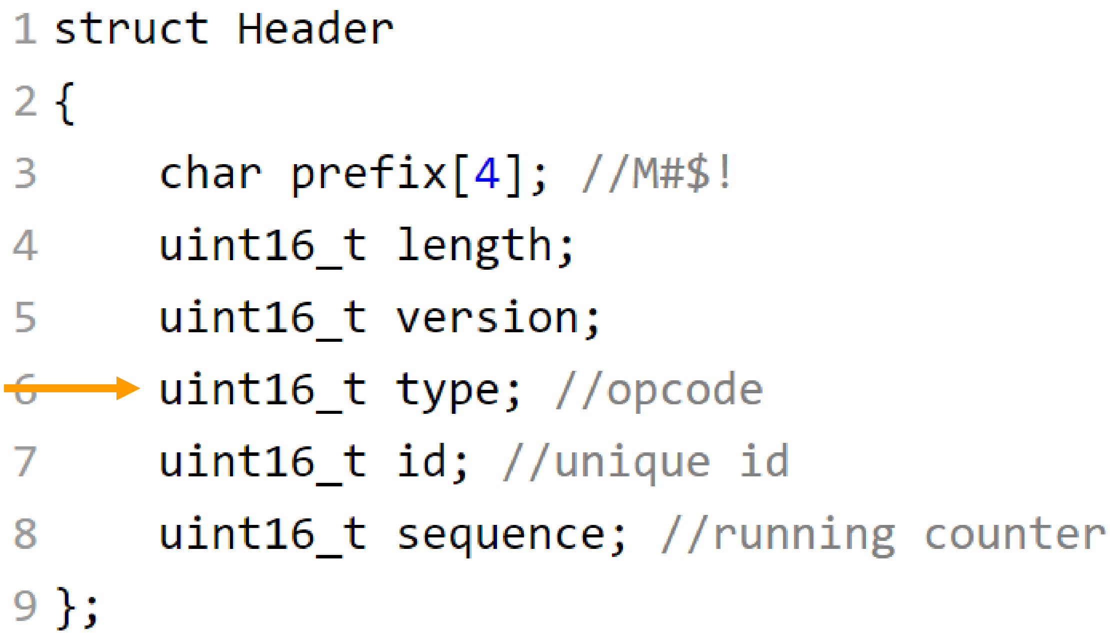
```
struct Header
{
    char prefix[4]; //M#$!
    uint16_t length;
    uint16_t version;
    uint16_t type; //opcode
    uint16_t id; //unique id
    uint16_t sequence; //running counter
};
```

Message Structure



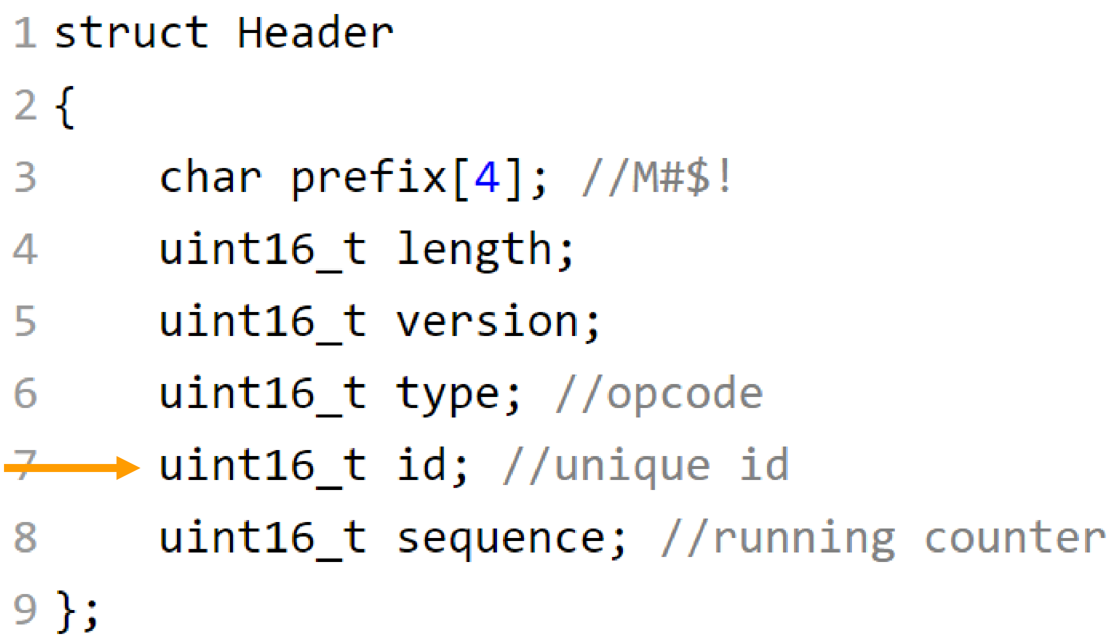
```
struct Header
{
    char prefix[4]; //M#$!
    uint16_t length;
    uint16_t version;
    uint16_t type; //opcode
    uint16_t id; //unique id
    uint16_t sequence; //running counter
};
```

Message Structure



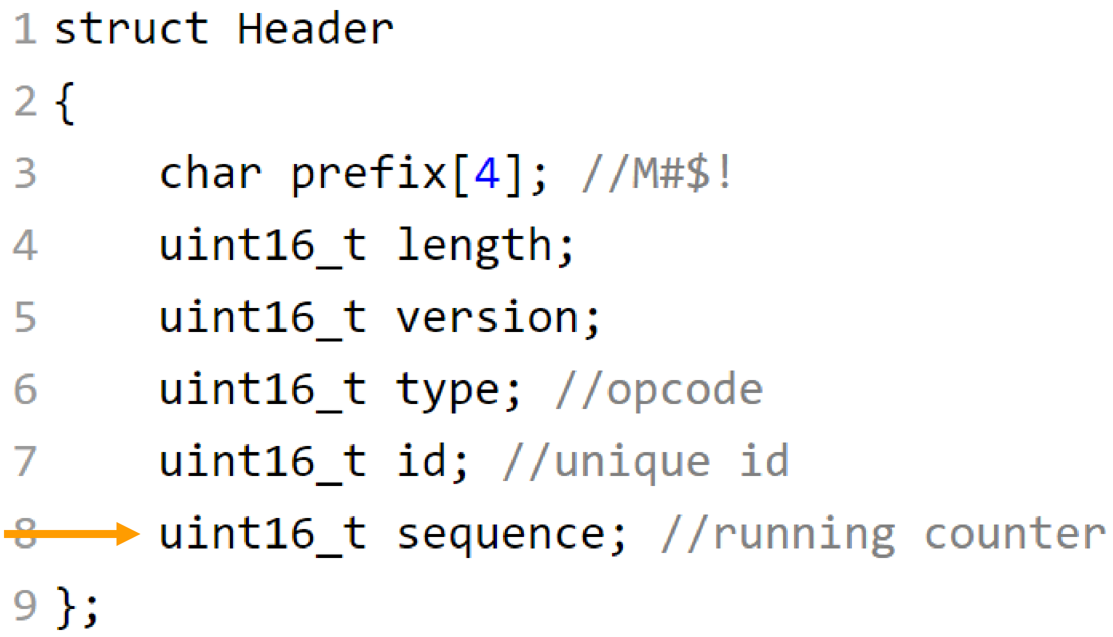
```
struct Header
{
    char prefix[4]; //M#$!
    uint16_t length;
    uint16_t version;
    uint16_t type; //opcode
    uint16_t id; //unique id
    uint16_t sequence; //running counter
};
```

Message Structure



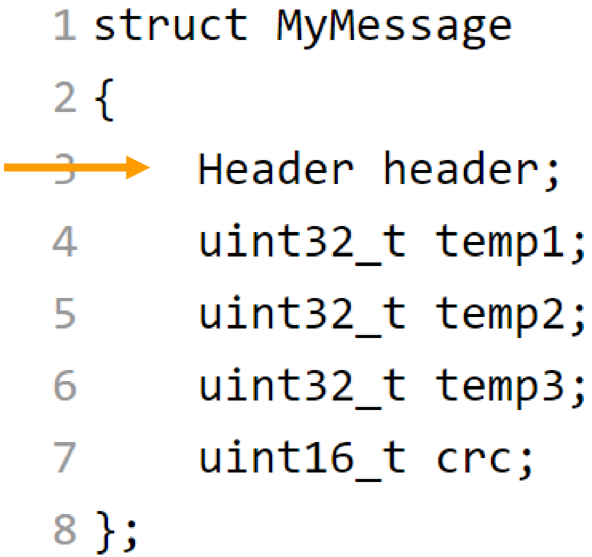
```
struct Header
{
    char prefix[4]; //M#$!
    uint16_t length;
    uint16_t version;
    uint16_t type; //opcode
    uint16_t id; //unique id
    uint16_t sequence; //running counter
};
```

Message Structure



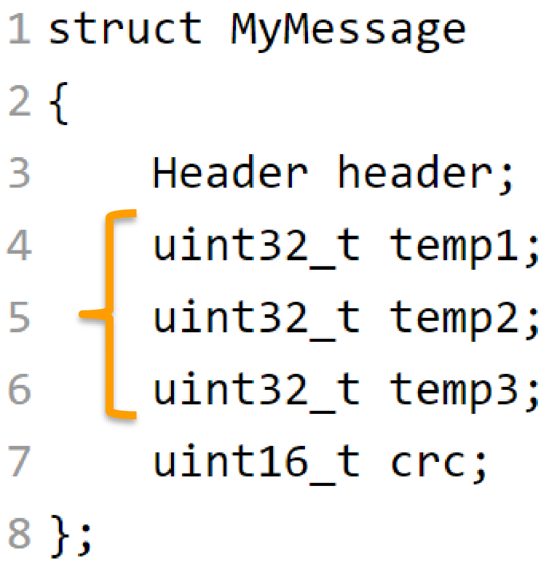
```
struct Header
{
    char prefix[4]; //M#$!
    uint16_t length;
    uint16_t version;
    uint16_t type; //opcode
    uint16_t id; //unique id
    uint16_t sequence; //running counter
};
```

Message Structure



```
struct MyMessage
{
    Header header;
    uint32_t temp1;
    uint32_t temp2;
    uint32_t temp3;
    uint16_t crc;
};
```

Message Structure

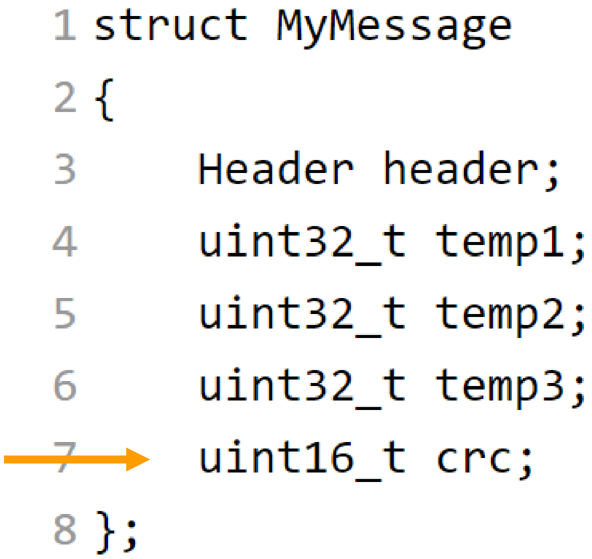


```
struct MyMessage
{
    Header header;
    uint32_t temp1;
    uint32_t temp2;
    uint32_t temp3;
    uint16_t crc;
};
```

Body structure

- Propriety protocol
- Protobuf : (https://protobuf.dev)
- CBOR : (https://cbor.io)
- JSON
- YAML

Message Structure



```
struct MyMessage
{
      Header header;
      uint32_t temp1;
      uint32_t temp2;
      uint32_t temp3;
      uint16_t crc;
};
```

> Design your protocol
> *in a way you could always*
> *bounce back*
> *from a “bad” message*

Agenda

- Operating Systems
- Threads
- Layer Separation
- Network Problems
- Message Structure
- Simulators
- Logs
- Monitoring

Simulators

Simulators

The simulator allows us to replicate scenarios that would be difficult to test in real life

The simulator allows us to replicate scenarios that would be difficult to test in real life

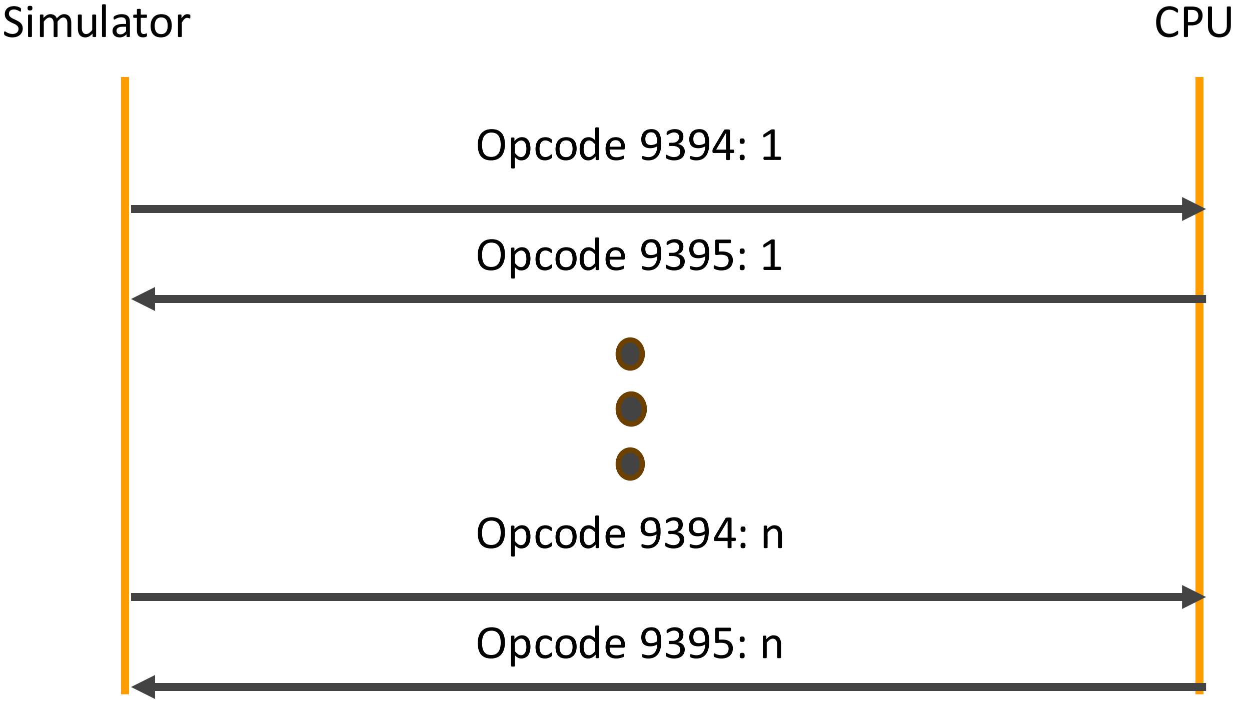
→ Interfaces simulator

→ Load simulator

Simulator #1

Q:How to verify our CPU supports 200 messages per second?

Simulators



Simulator

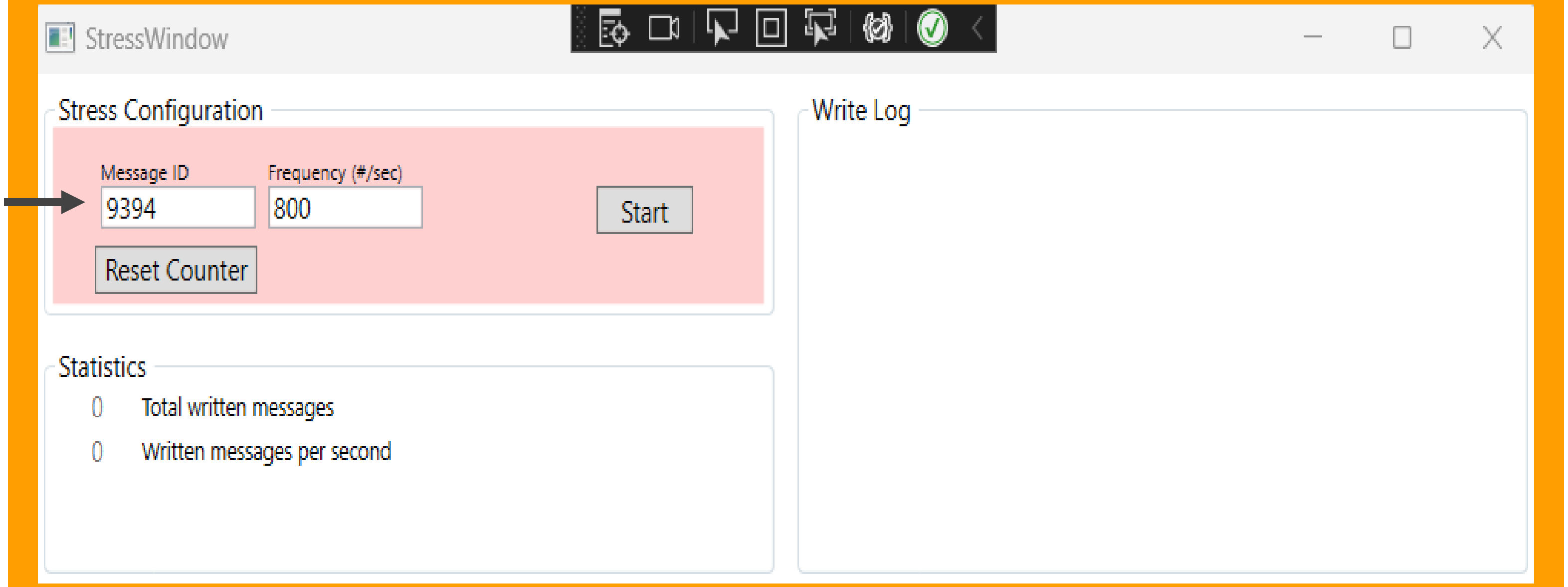
CPU

Opcode 9394: 1

Opcode 9395: 1

Opcode 9394: n

Opcode 9395: n



Stress Configuration

Message ID: 9394

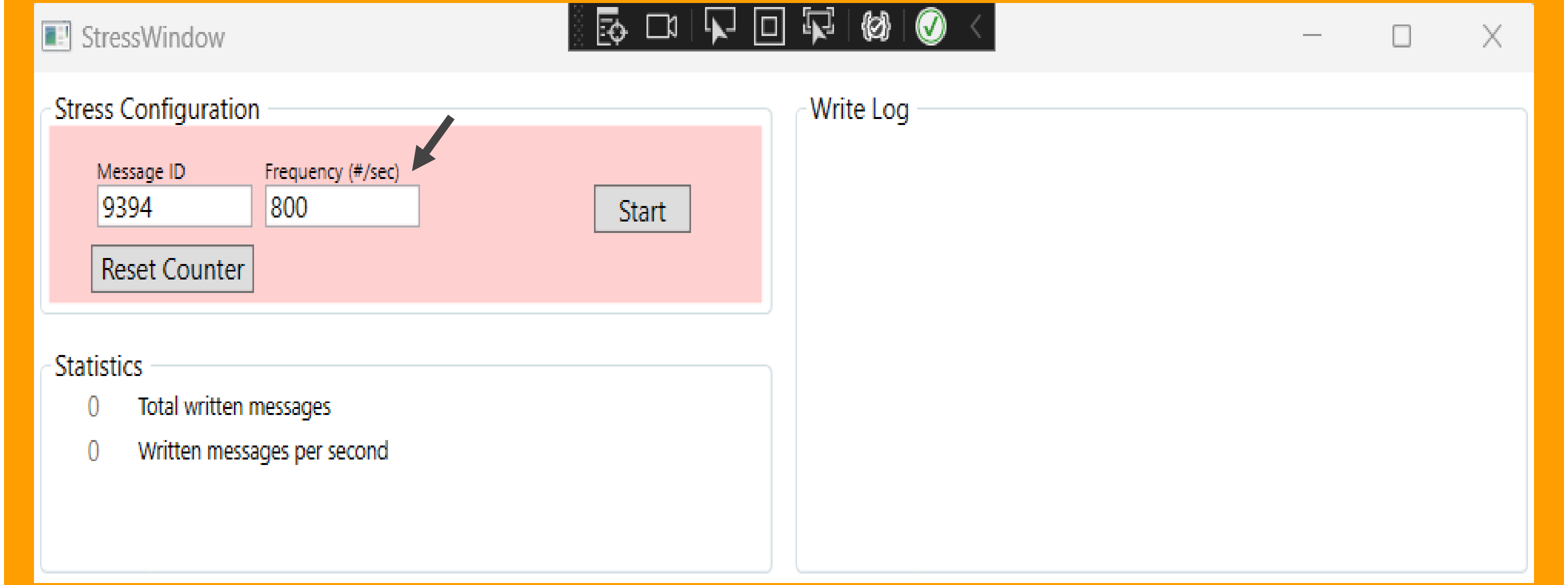
Frequency (#/sec): 800

Statistics

0 Total written messages

0 Written messages per second

Write Log

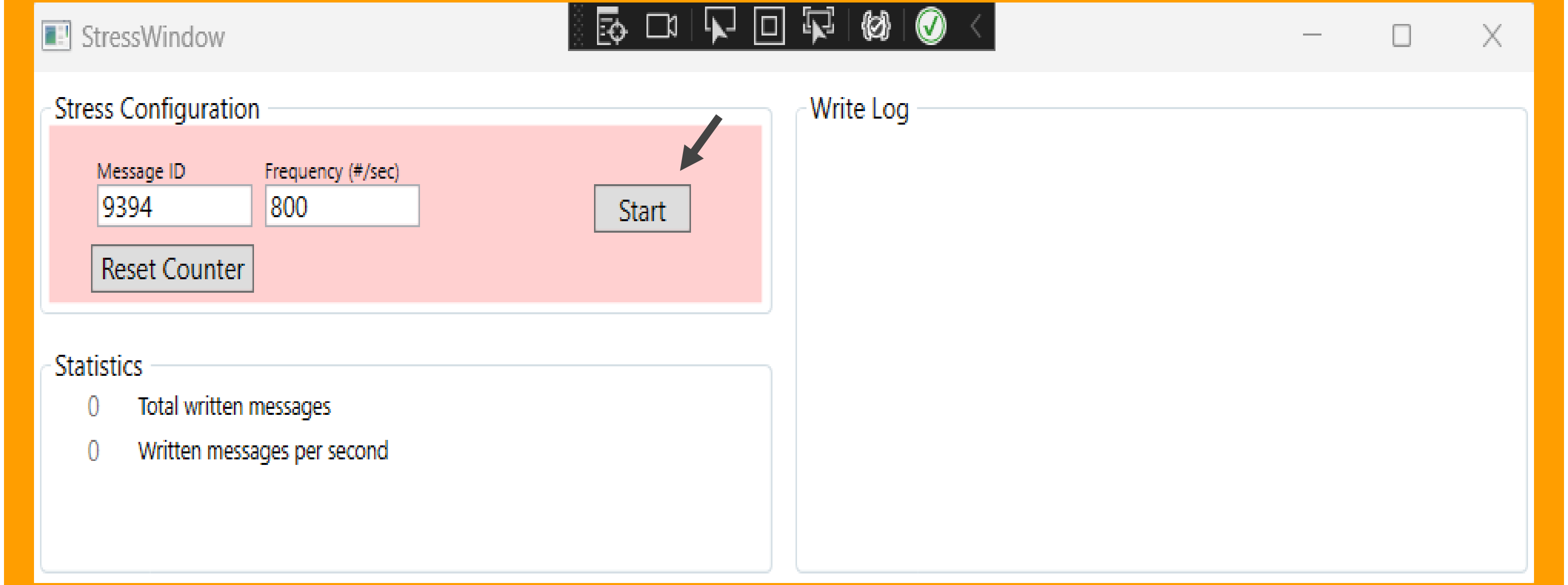


Message ID: 9394

Frequency (#/sec): 800

0 Total written messages

0 Written messages per second

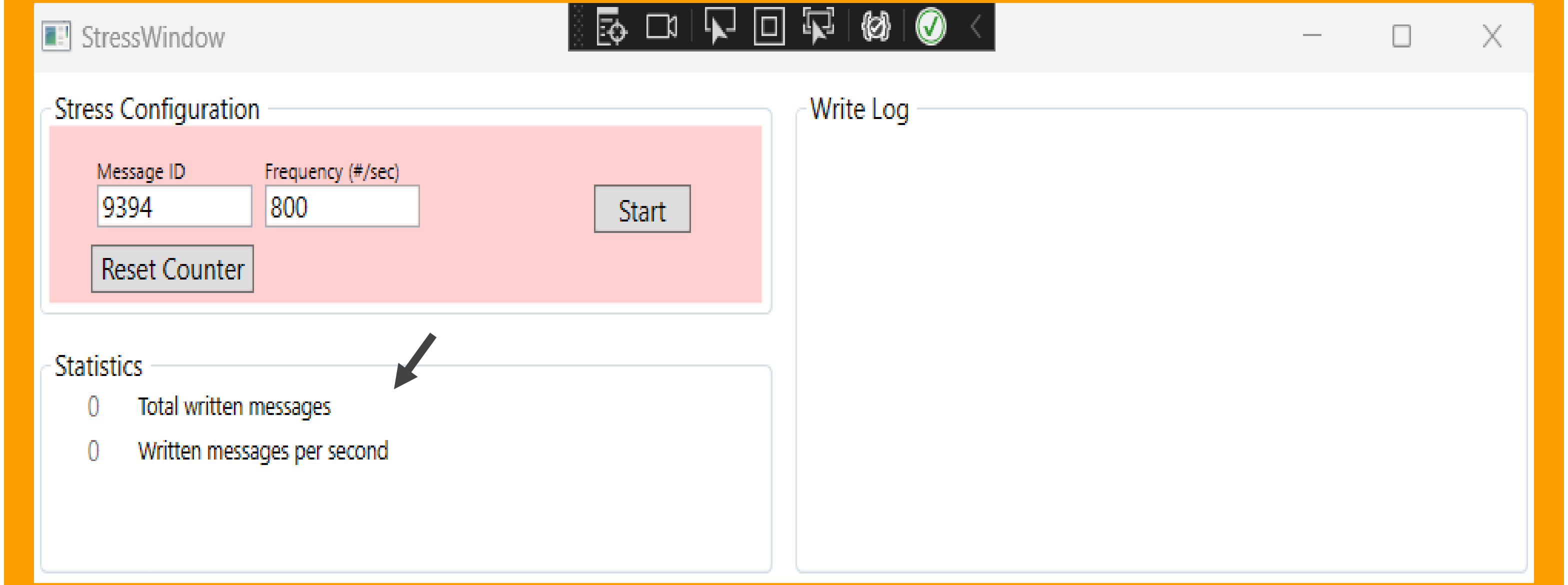


Message ID: 9394

Frequency (#/sec): 800

0 Total written messages

0 Written messages per second

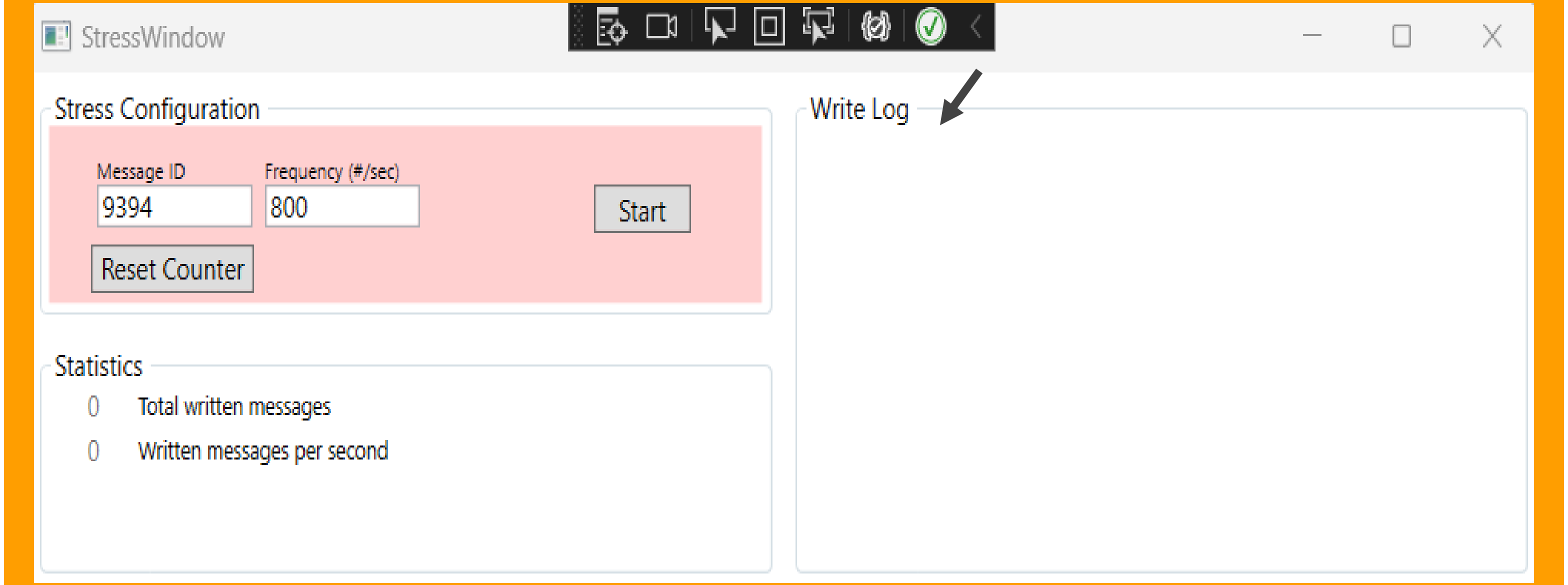


Message ID: 9394

Frequency (#/sec): 800

0 Total written messages

0 Written messages per second



Stress Configuration

Message ID: 9394

Frequency (#/sec): 800

Statistics

0 Total written messages

0 Written messages per second

Write Log

Simulators

- The simulator sends X messages per second and verifies that it receives all of them back
- At 850 messages per second, it stopped working
- The PC didn’t have enough resources
- The embedded CPU passed the test

- The simulator sends X messages per second and verifies that it receives all of them back
- At 850 messages per second, it stopped working
- The PC didn’t have enough resources
- The embedded CPU passed the test

Simulators

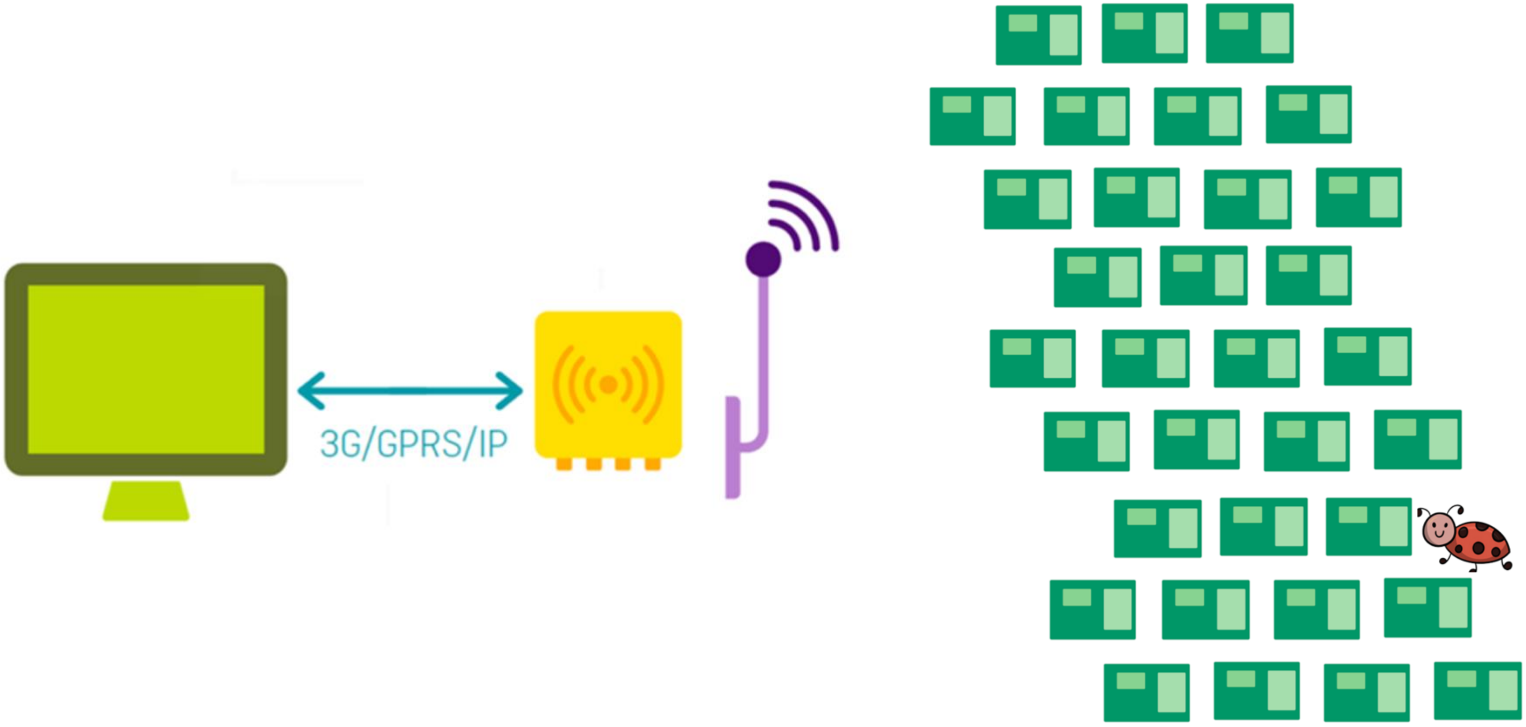
- The simulator sends X messages per second and verifies that it receives all of them back
- At 850 messages per second, it stopped working
- The PC didn’t have enough resources
- The embedded CPU passed the test

Simulators

- The simulator sends X messages per second and verifies that it receives all of them back
- At 850 messages per second, it stopped working
- The PC didn’t have enough resources
- The embedded CPU passed the test

Simulator #2

Threads



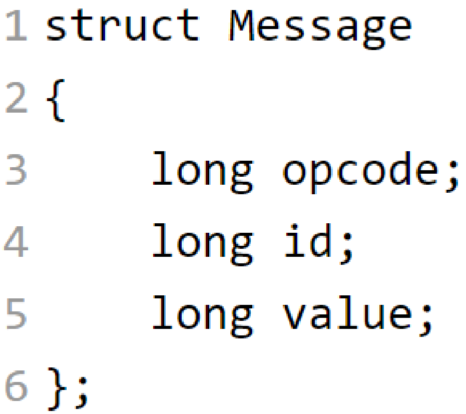
Q: How to simulate 5K+ endpoints?

- 1 data frame to 100 fake data frames
- The unit crashed on my table when I simulated 6k endpoints and put “aggressive configuration”

Simulators

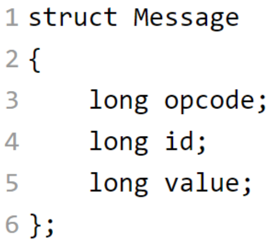
- 1 data frame to 100 fake data frames
- The unit crashed on my table when I simulated 6k endpoints and put “aggressive configuration”

Simulators

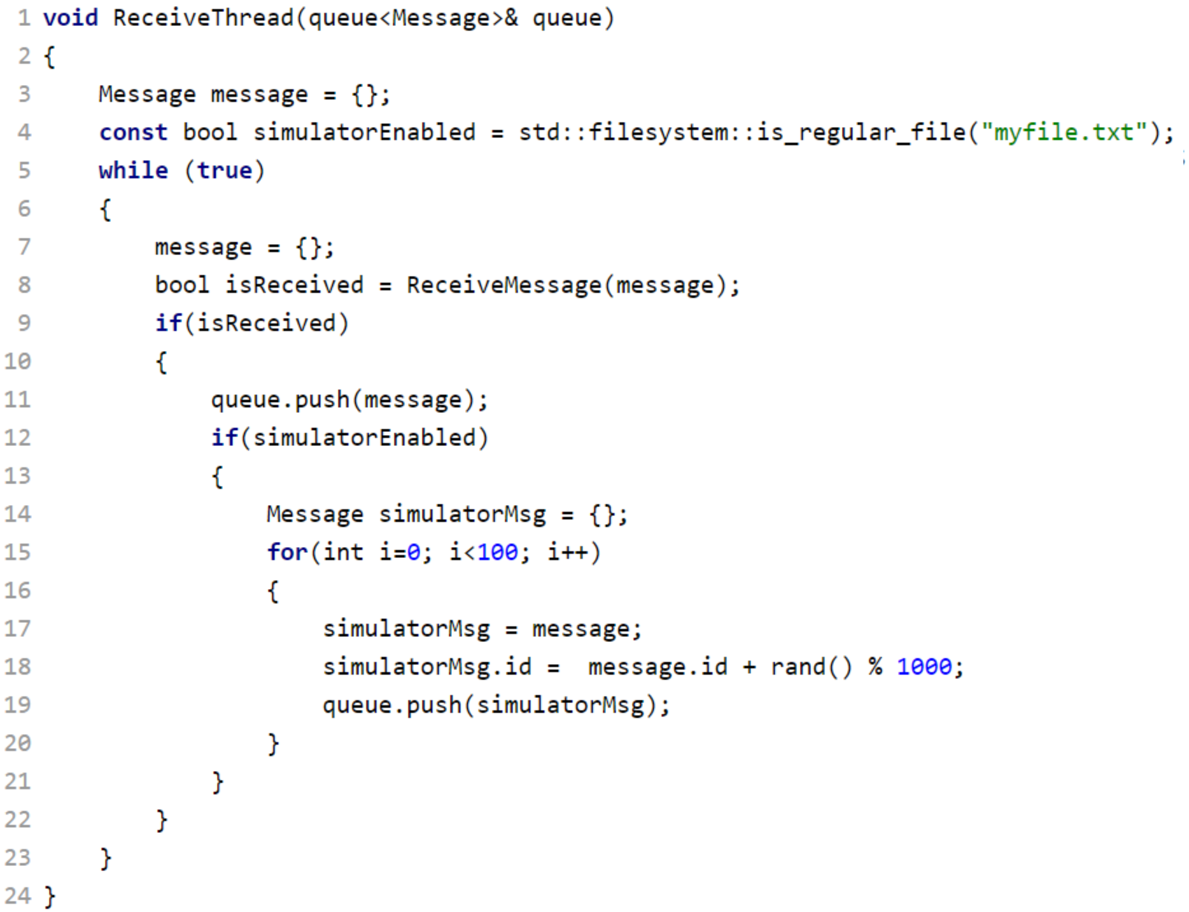


```
struct Message
{
      long opcode;
      long id;
      long value;
};
```

Simulators

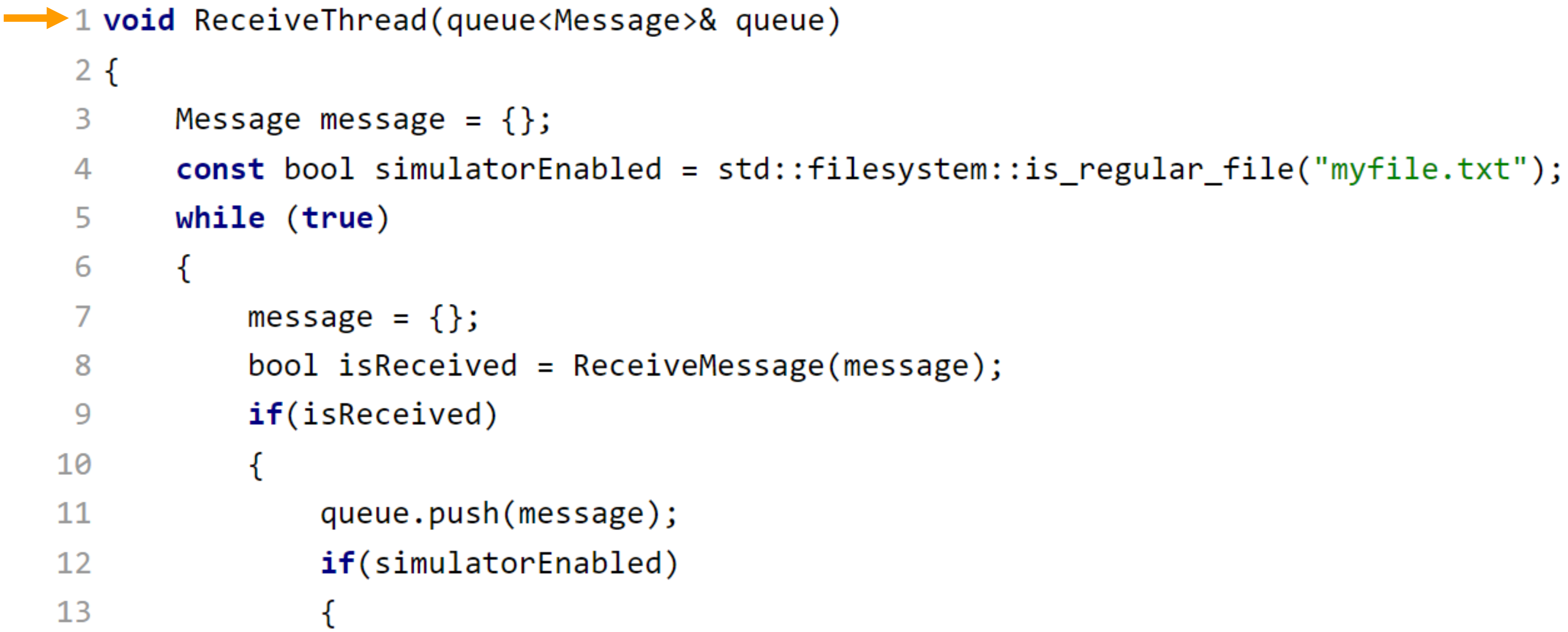


```cpp
struct Message
{
      long opcode;
      long id;
      long value;
};
```



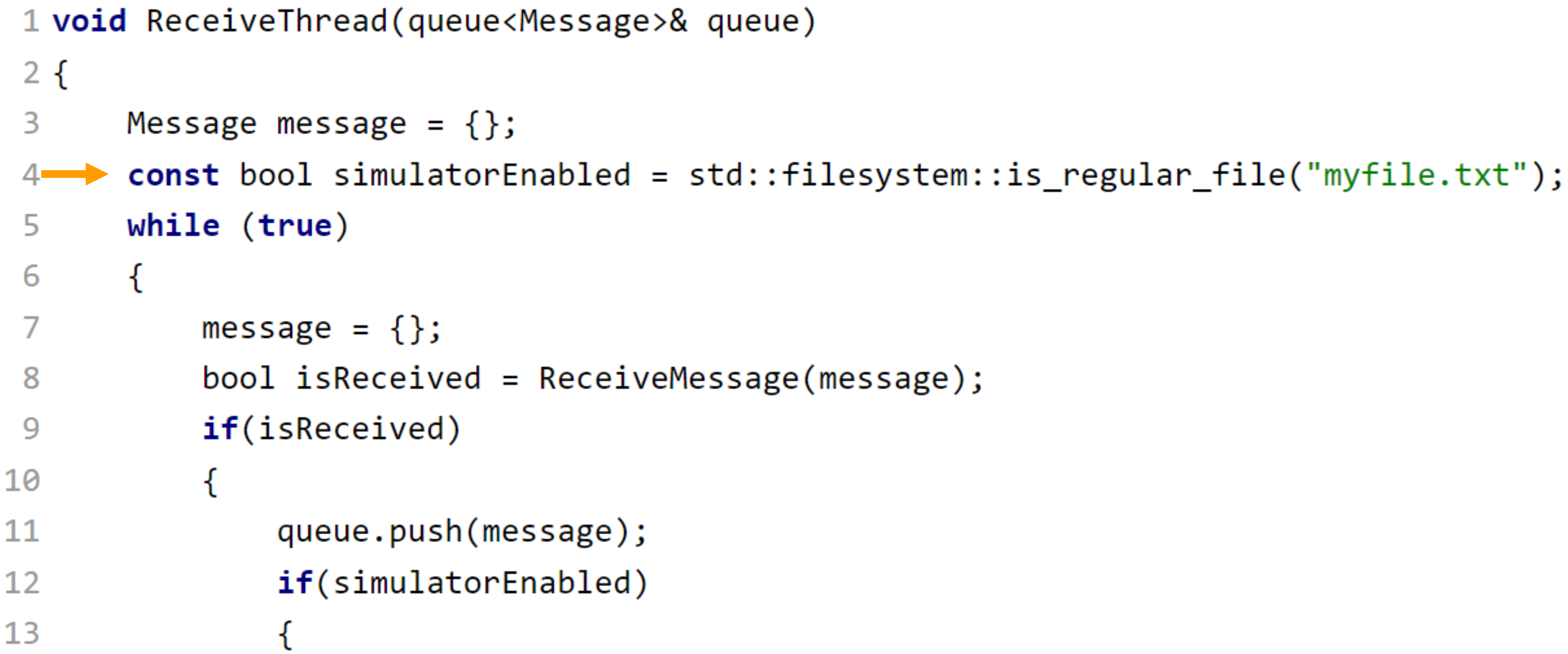
```cpp
void ReceiveThread(queue<Message>& queue)
{
    Message message = {};
    const bool simulatorEnabled = std::filesystem::is_regular_file("myfile.txt");
    while (true)
    {
        message = {};
        bool isReceived = ReceiveMessage(message);
        if(isReceived)
        {
            queue.push(message);
            if(simulatorEnabled)
            {
                Message simulatorMsg = {};
                for(int i=0; i<100; i++)
                {
                    simulatorMsg = message;
                    simulatorMsg.id =  message.id + rand() % 1000;
                    queue.push(simulatorMsg);
                }
            }
        }
    }
}
```

Simulators



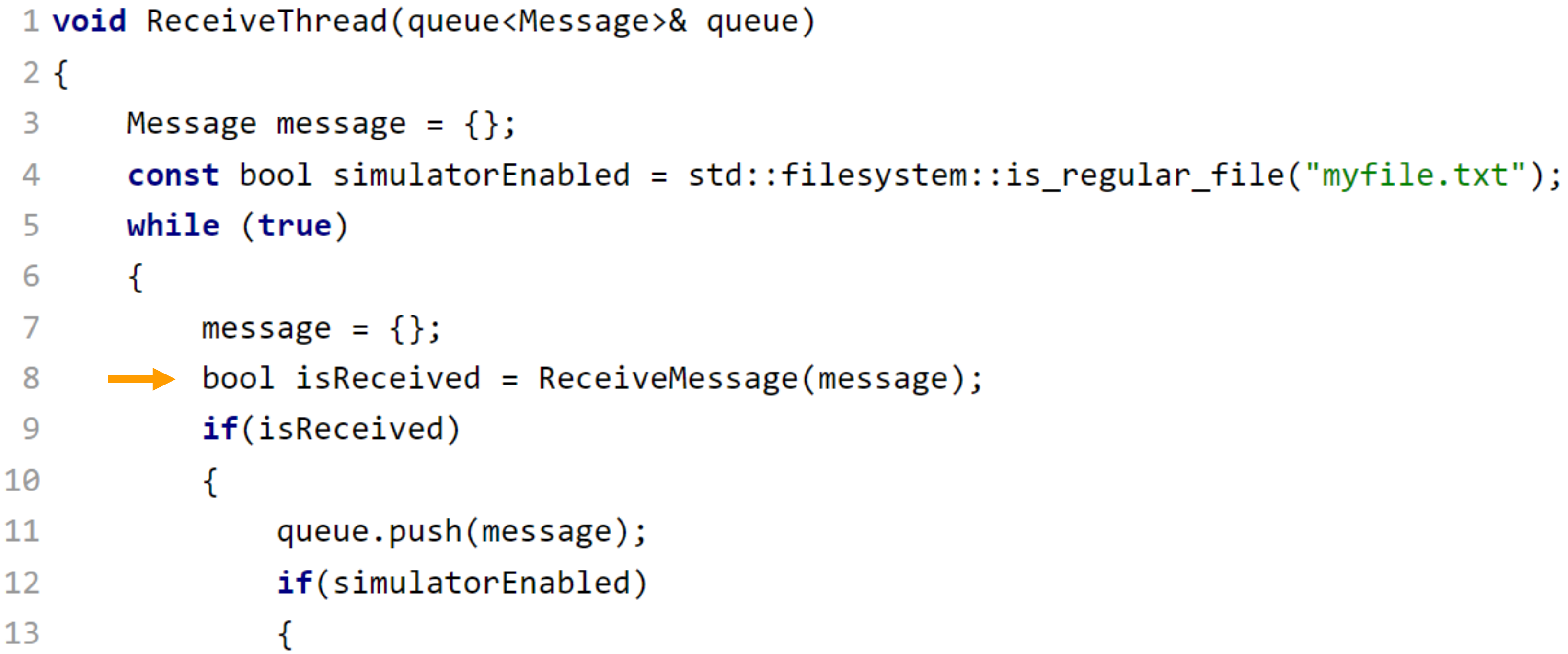
```cpp
void ReceiveThread(queue<Message>& queue)
{
    Message message = {};
    const bool simulatorEnabled = std::filesystem::is_regular_file("myfile.txt");
    while (true)
    {
        message = {};
        bool isReceived = ReceiveMessage(message);
        if(isReceived)
        {
            queue.push(message);
            if(simulatorEnabled)
            {
```

Simulators



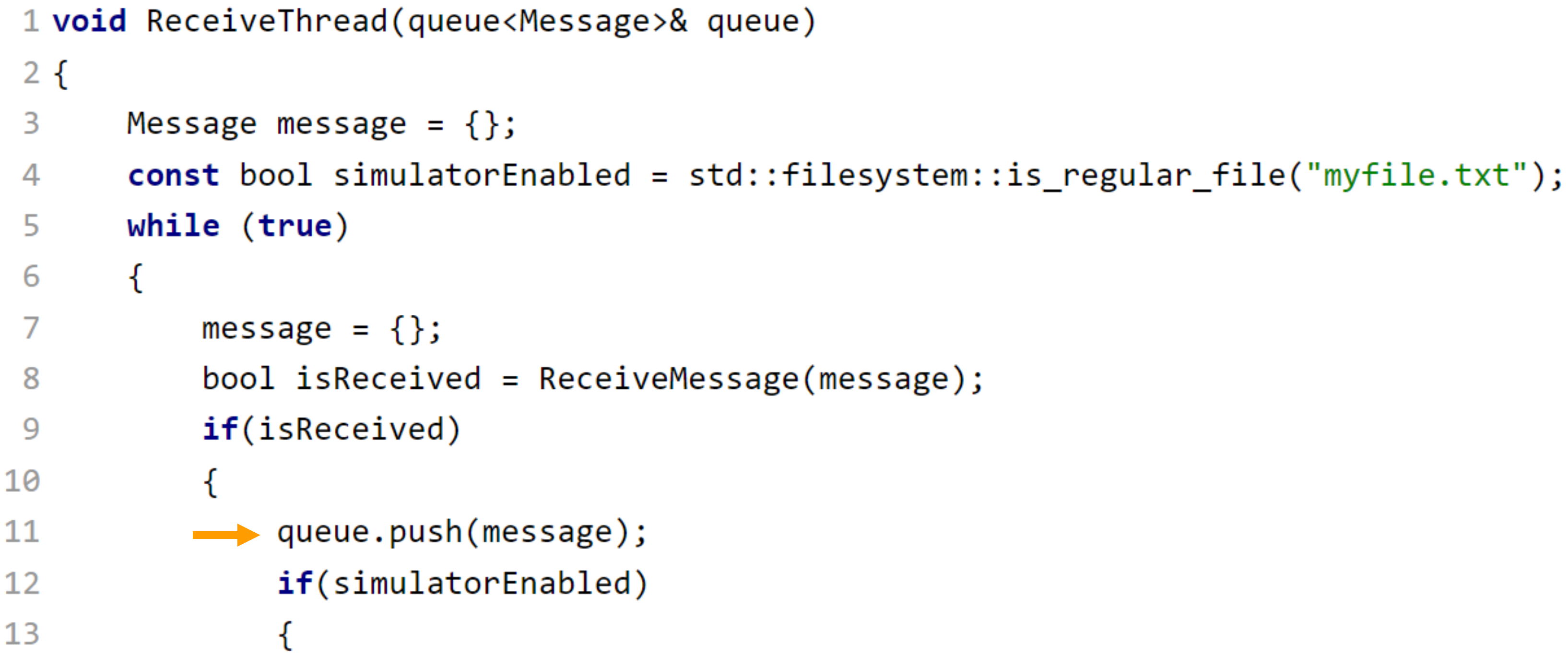
```
void ReceiveThread(queue<Message>& queue)
{
    Message message = {};
    const bool simulatorEnabled = std::filesystem::is_regular_file("myfile.txt");
    while (true)
    {
        message = {};
        bool isReceived = ReceiveMessage(message);
        if(isReceived)
        {
            queue.push(message);
            if(simulatorEnabled)
            {
```

Simulators



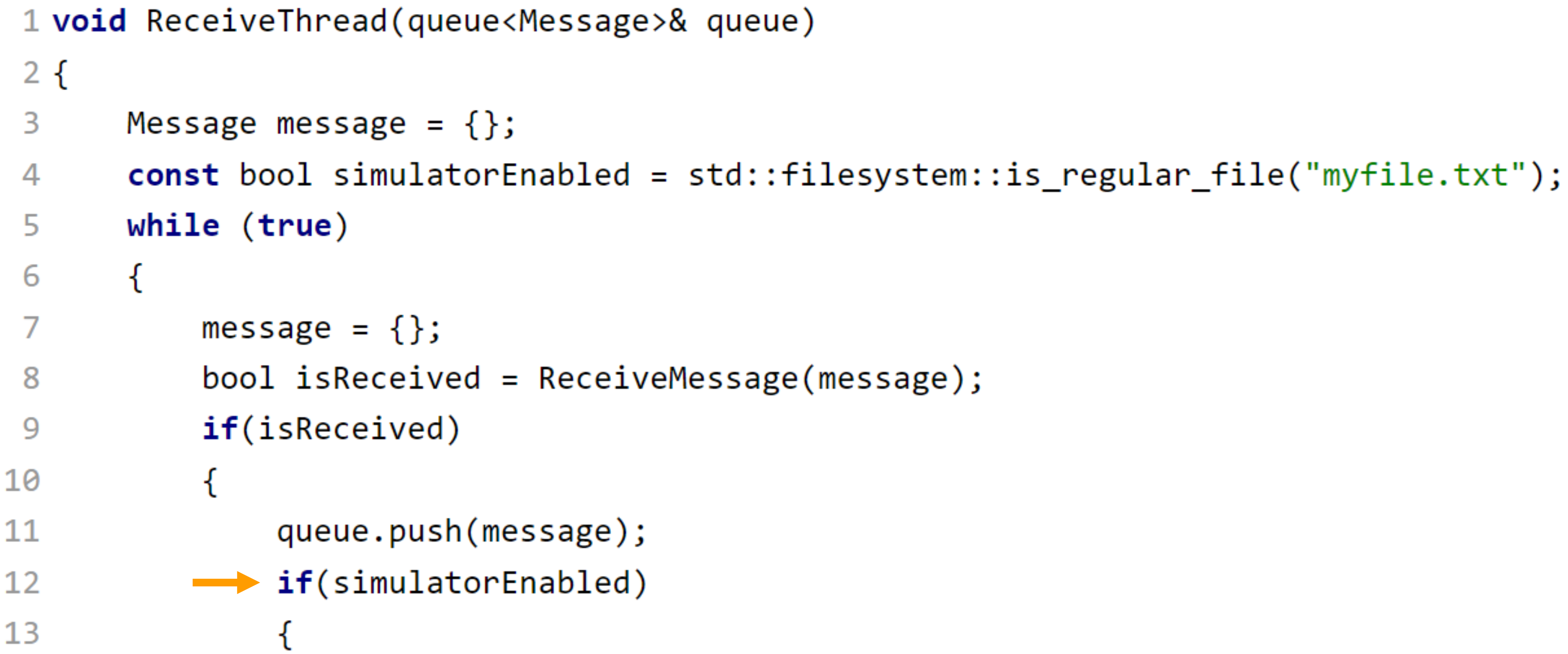
```
void ReceiveThread(queue<Message>& queue)
{
    Message message = {};
    const bool simulatorEnabled = std::filesystem::is_regular_file("myfile.txt");
    while (true)
    {
        message = {};
        bool isReceived = ReceiveMessage(message);
        if(isReceived)
        {
            queue.push(message);
            if(simulatorEnabled)
            {
```

Simulators



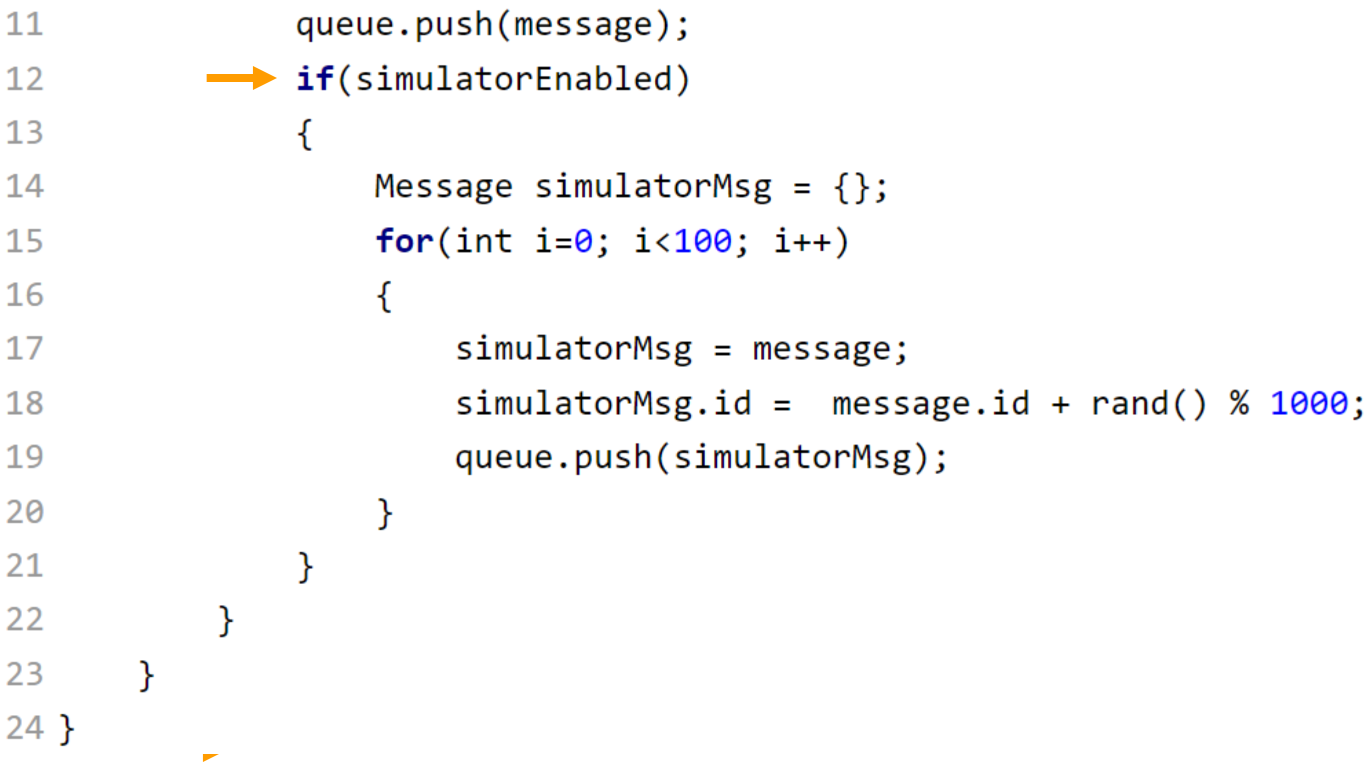
```cpp
void ReceiveThread(queue<Message>& queue)
{
    Message message = {};
    const bool simulatorEnabled = std::filesystem::is_regular_file("myfile.txt");
    while (true)
    {
        message = {};
        bool isReceived = ReceiveMessage(message);
        if(isReceived)
        {
            queue.push(message);
            if(simulatorEnabled)
            {
```

Simulators



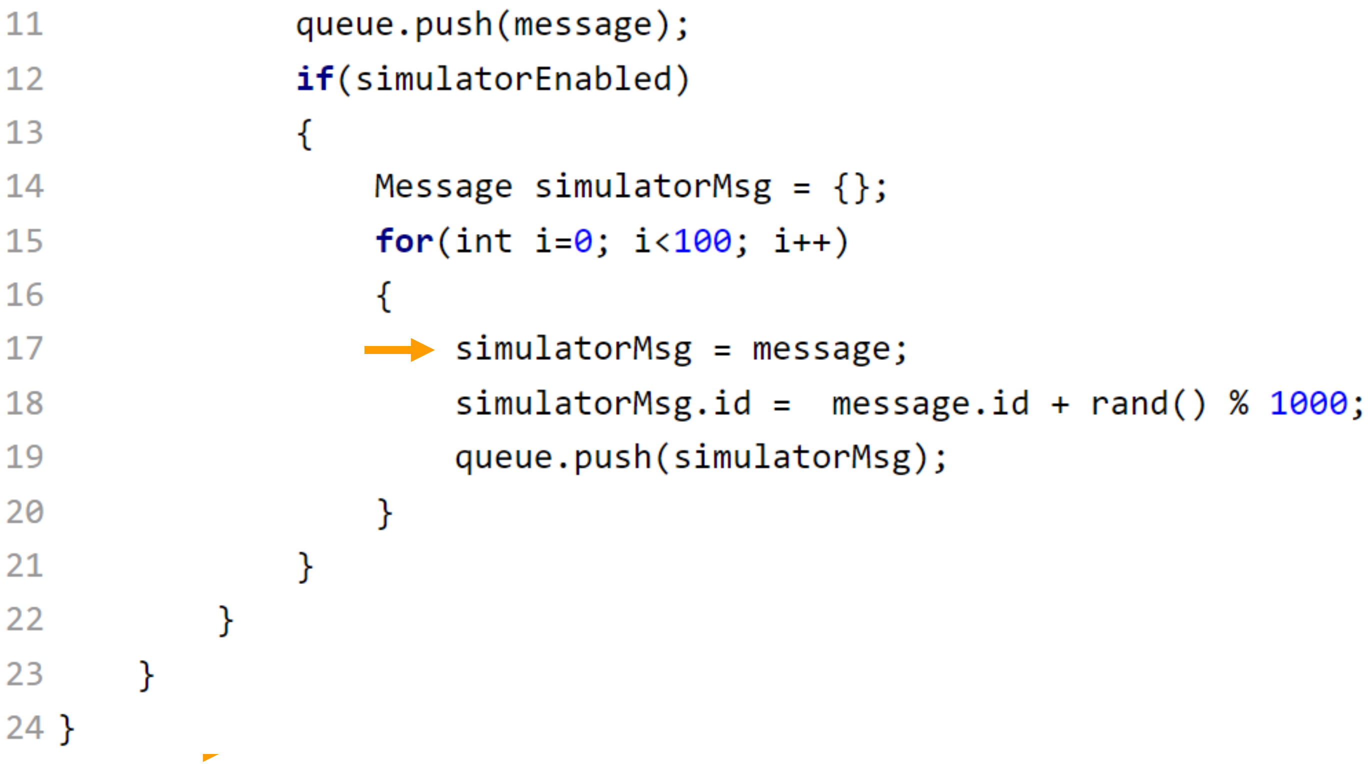
```
void ReceiveThread(queue<Message>& queue)
{
    Message message = {};
    const bool simulatorEnabled = std::filesystem::is_regular_file("myfile.txt");
    while (true)
    {
        message = {};
        bool isReceived = ReceiveMessage(message);
        if(isReceived)
        {
            queue.push(message);
            if(simulatorEnabled)
            {
```

Simulators



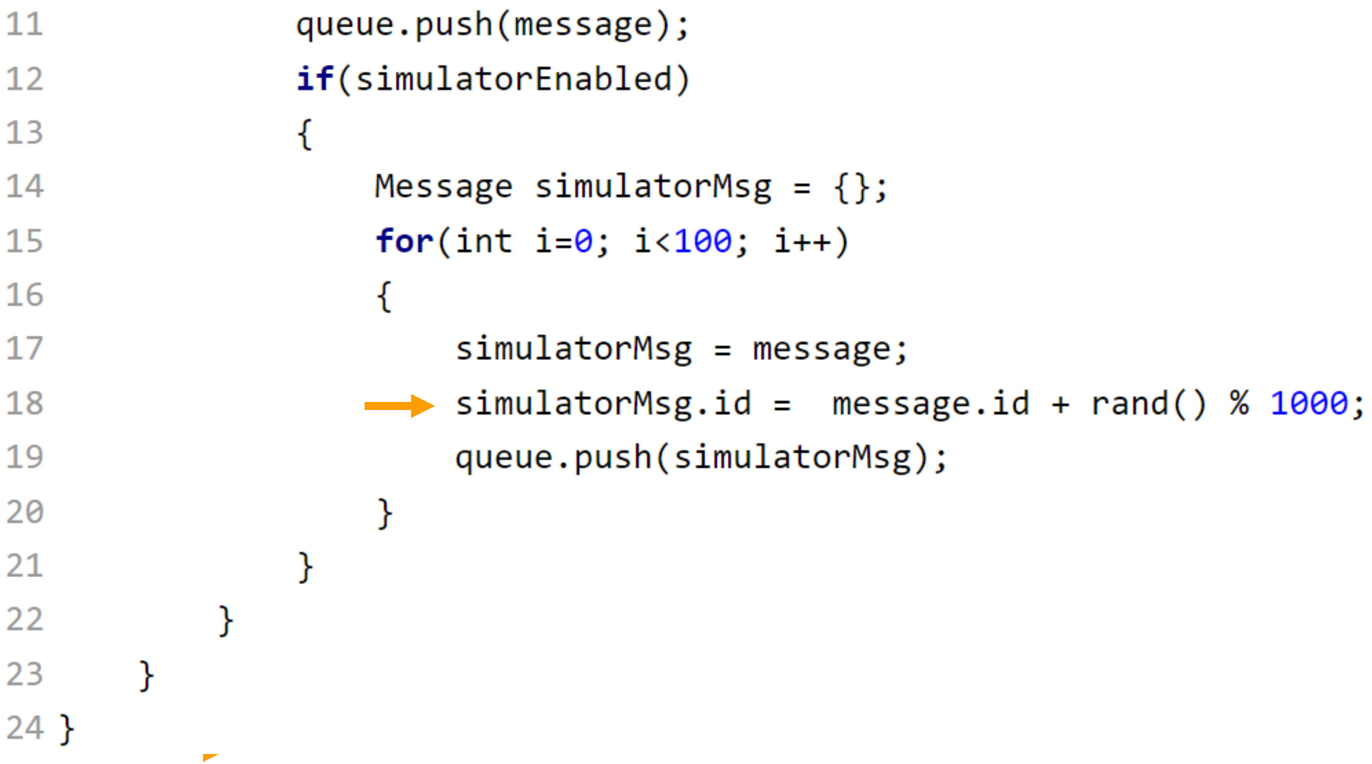
```
            queue.push(message);
            if(simulatorEnabled)
            {
                Message simulatorMsg = {};
                for(int i=0; i<100; i++)
                {
                    simulatorMsg = message;
                    simulatorMsg.id =  message.id + rand() % 1000;
                    queue.push(simulatorMsg);
                }
            }
        }
    }
}
```

Simulators



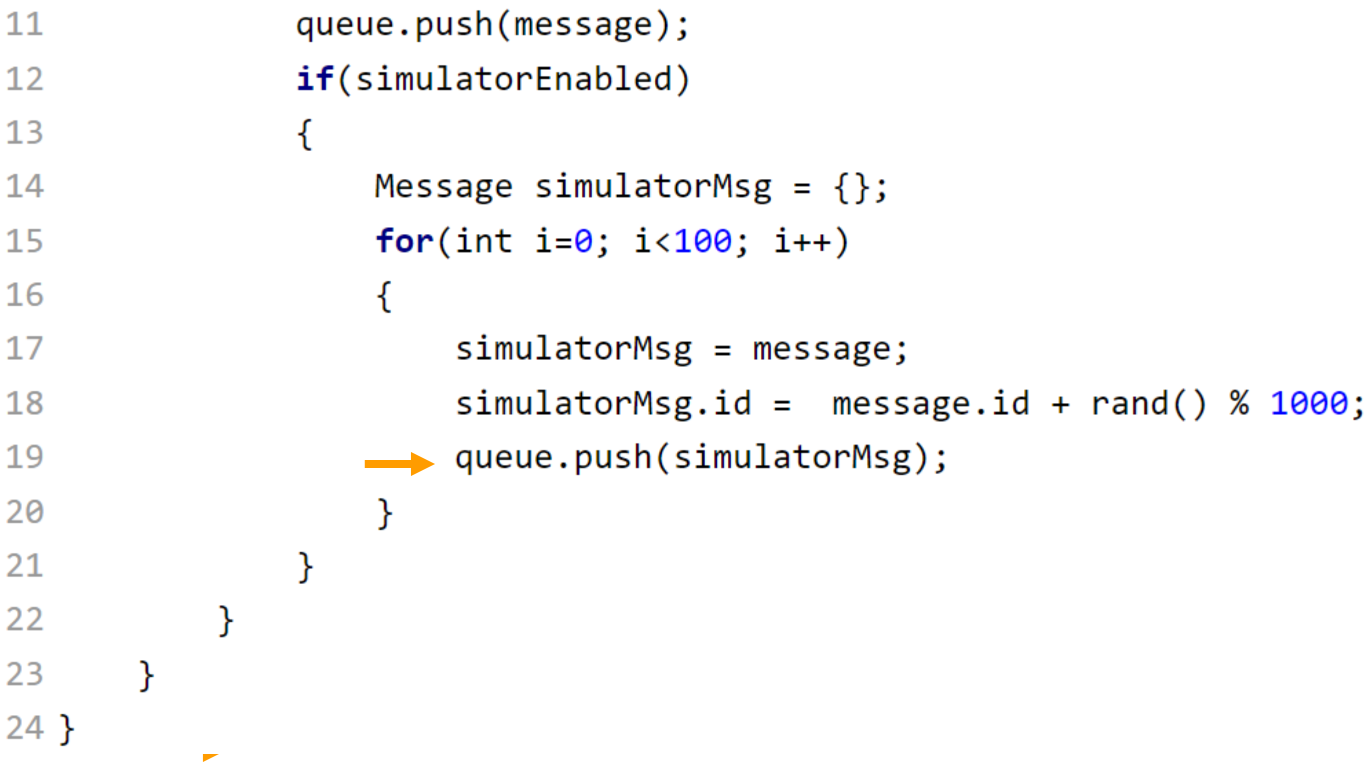
```
            queue.push(message);
            if(simulatorEnabled)
            {
                Message simulatorMsg = {};
                for(int i=0; i<100; i++)
                {
                    simulatorMsg = message;
                    simulatorMsg.id =  message.id + rand() % 1000;
                    queue.push(simulatorMsg);
                }
            }
        }
    }
}
```

Simulators



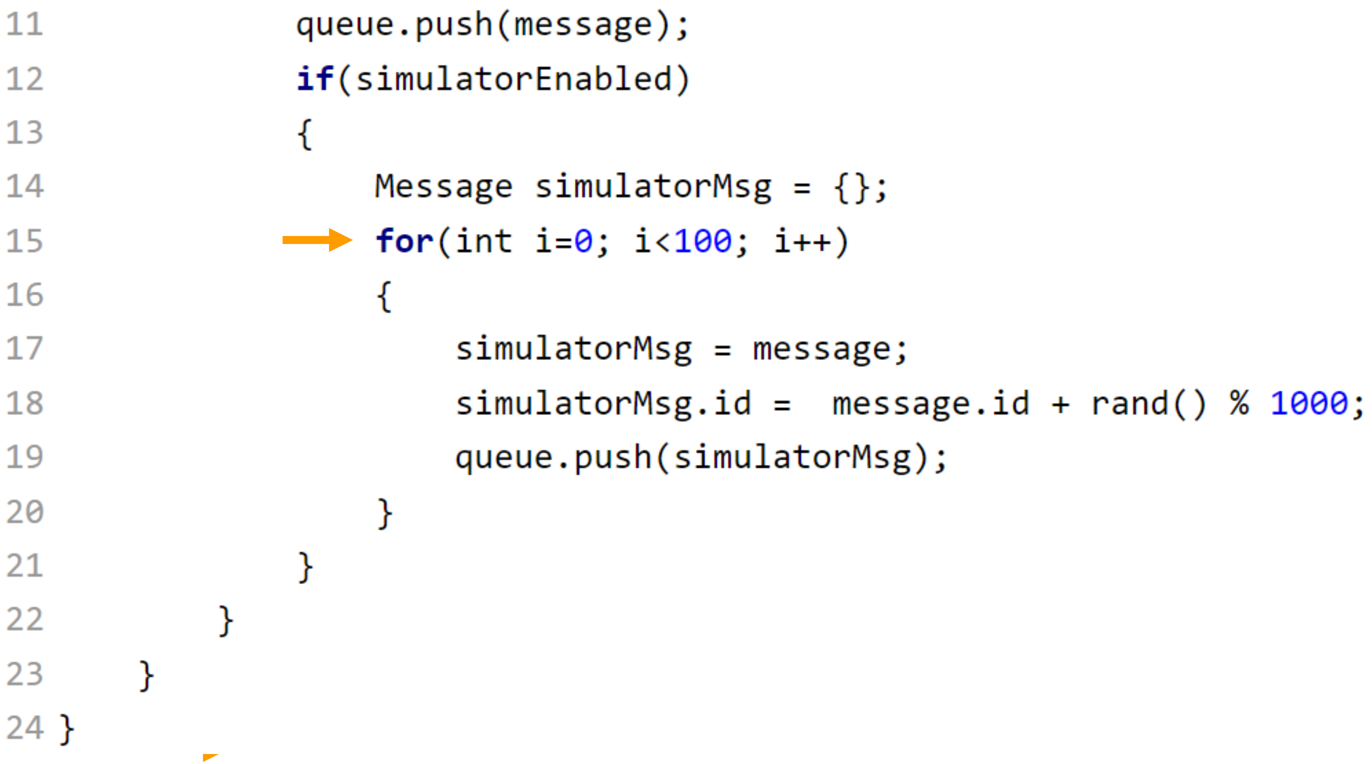
```
            queue.push(message);
            if(simulatorEnabled)
            {
                Message simulatorMsg = {};
                for(int i=0; i<100; i++)
                {
                    simulatorMsg = message;
                    simulatorMsg.id =  message.id + rand() % 1000;
                    queue.push(simulatorMsg);
                }
            }
        }
    }
}
```

Simulators



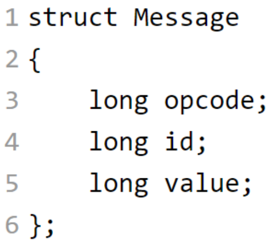
```
            queue.push(message);
            if(simulatorEnabled)
            {
                Message simulatorMsg = {};
                for(int i=0; i<100; i++)
                {
                    simulatorMsg = message;
                    simulatorMsg.id =  message.id + rand() % 1000;
                    queue.push(simulatorMsg);
                }
            }
        }
    }
}
```

Simulators

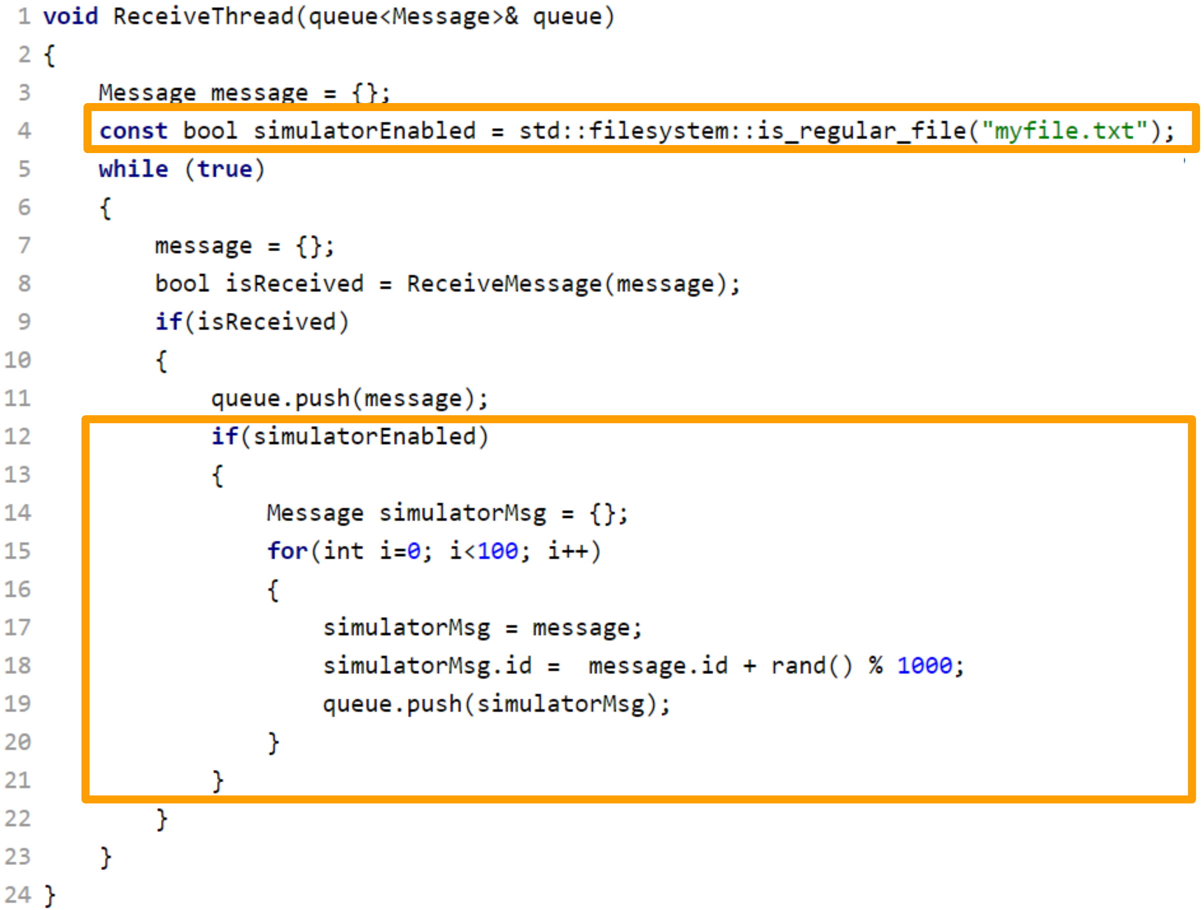


```
            queue.push(message);
            if(simulatorEnabled)
            {
                Message simulatorMsg = {};
                for(int i=0; i<100; i++)
                {
                    simulatorMsg = message;
                    simulatorMsg.id =  message.id + rand() % 1000;
                    queue.push(simulatorMsg);
                }
            }
        }
    }
}
```

Simulators



```cpp
struct Message
{
    long opcode;
    long id;
    long value;
};
```



```cpp
void ReceiveThread(queue<Message>& queue)
{
    Message message = {};
    const bool simulatorEnabled = std::filesystem::is_regular_file("myfile.txt");
    while (true)
    {
        message = {};
        bool isReceived = ReceiveMessage(message);
        if(isReceived)
        {
            queue.push(message);
            if(simulatorEnabled)
            {
                Message simulatorMsg = {};
                for(int i=0; i<100; i++)
                {
                    simulatorMsg = message;
                    simulatorMsg.id =  message.id + rand() % 1000;
                    queue.push(simulatorMsg);
                }
            }
        }
    }
}
```

Simulators

What is special here:

- ⊡ No special hardware required.
- ⊡ Simple implementation.
- ⊡ Easy access to simulator mode without additional building.
- ⊡ Consistently crashed the system.

What is special here:

- ⊡ No special hardware required.
- ⊡ Simple implementation.
- ⊡ Easy access to simulator mode without additional building.
- ⊡ Consistently crashed the system.

What is special here:

- No special hardware required.
- Simple implementation.
- Easy access to simulator mode without additional building.
- Consistently crashed the system.

Simulators

What is special here:

- ⊡ No special hardware required.
- ⊡ Simple implementation.
- ⊡ Easy access to simulator mode without additional building.
- ⊡ Consistently crashed the system.

Simulators

What is special here:

⊡ No special hardware required.

⊡ Simple implementation.

⊡ Easy access to simulator mode without additional building.

⊡ Consistently crashed the system.

“

Use simulators

Agenda

⊡ Operating Systems

⊡ Threads

⊡ Layer Separation

⊡ Network Problems

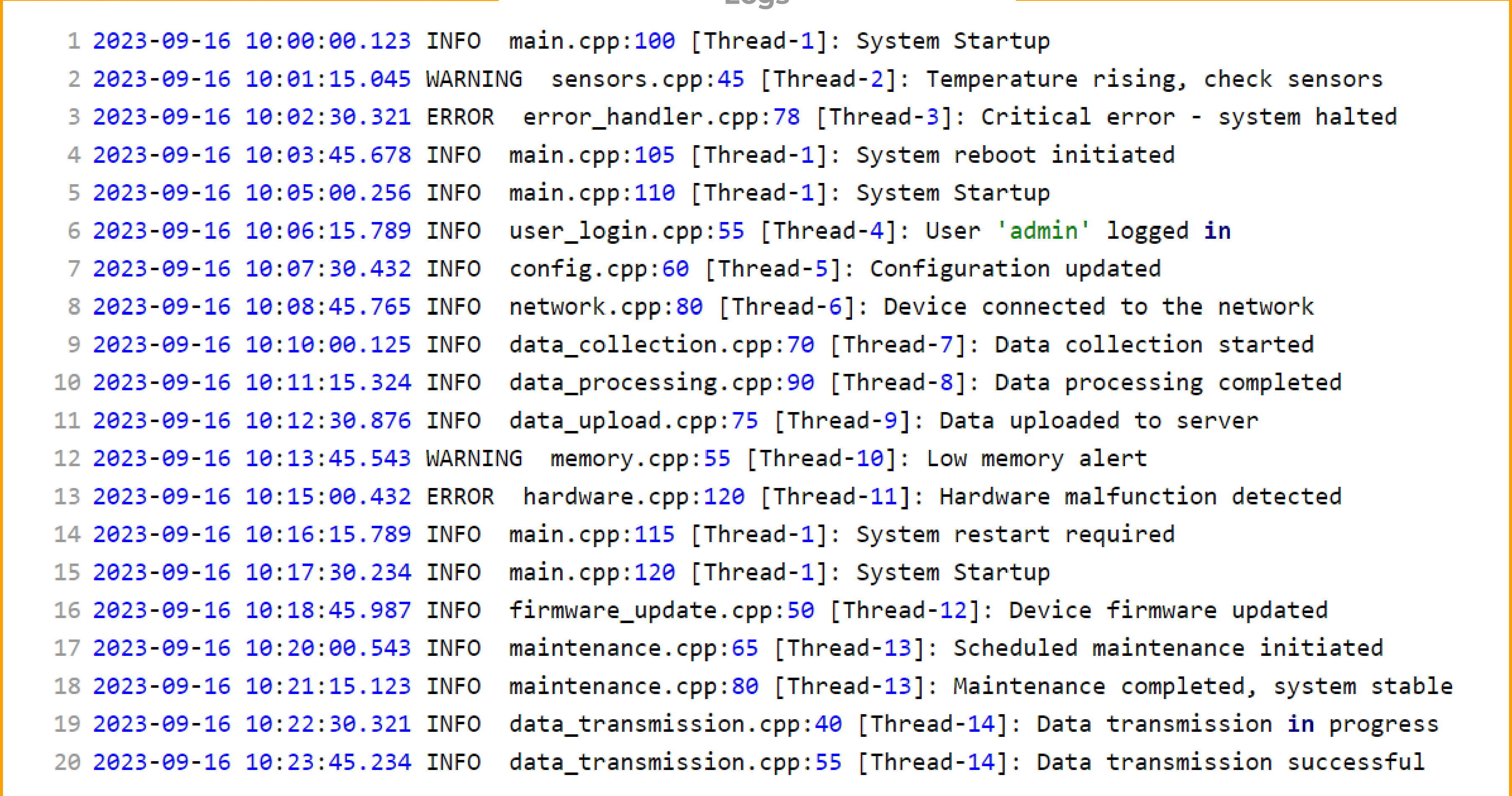
⊡ Message Structure

⊡ Simulators

⊡ Logs

⊡ Monitoring

Logs

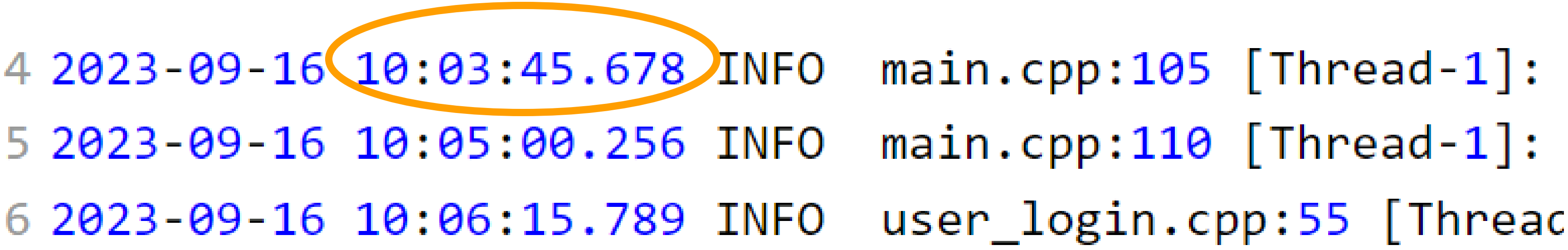


```
2023-09-16 10:00:00.123 INFO  main.cpp:100 [Thread-1]: System Startup
2023-09-16 10:01:15.045 WARNING  sensors.cpp:45 [Thread-2]: Temperature rising, check sensors
2023-09-16 10:02:30.321 ERROR  error_handler.cpp:78 [Thread-3]: Critical error - system halted
2023-09-16 10:03:45.678 INFO  main.cpp:105 [Thread-1]: System reboot initiated
2023-09-16 10:05:00.256 INFO  main.cpp:110 [Thread-1]: System Startup
2023-09-16 10:06:15.789 INFO  user_login.cpp:55 [Thread-4]: User 'admin' logged in
2023-09-16 10:07:30.432 INFO  config.cpp:60 [Thread-5]: Configuration updated
2023-09-16 10:08:45.765 INFO  network.cpp:80 [Thread-6]: Device connected to the network
2023-09-16 10:10:00.125 INFO  data_collection.cpp:70 [Thread-7]: Data collection started
2023-09-16 10:11:15.324 INFO  data_processing.cpp:90 [Thread-8]: Data processing completed
2023-09-16 10:12:30.876 INFO  data_upload.cpp:75 [Thread-9]: Data uploaded to server
2023-09-16 10:13:45.543 WARNING  memory.cpp:55 [Thread-10]: Low memory alert
2023-09-16 10:15:00.432 ERROR  hardware.cpp:120 [Thread-11]: Hardware malfunction detected
2023-09-16 10:16:15.789 INFO  main.cpp:115 [Thread-1]: System restart required
2023-09-16 10:17:30.234 INFO  main.cpp:120 [Thread-1]: System Startup
2023-09-16 10:18:45.987 INFO  firmware_update.cpp:50 [Thread-12]: Device firmware updated
2023-09-16 10:20:00.543 INFO  maintenance.cpp:65 [Thread-13]: Scheduled maintenance initiated
2023-09-16 10:21:15.123 INFO  maintenance.cpp:80 [Thread-13]: Maintenance completed, system stable
2023-09-16 10:22:30.321 INFO  data_transmission.cpp:40 [Thread-14]: Data transmission in progress
2023-09-16 10:23:45.234 INFO  data_transmission.cpp:55 [Thread-14]: Data transmission successful
```

Logs

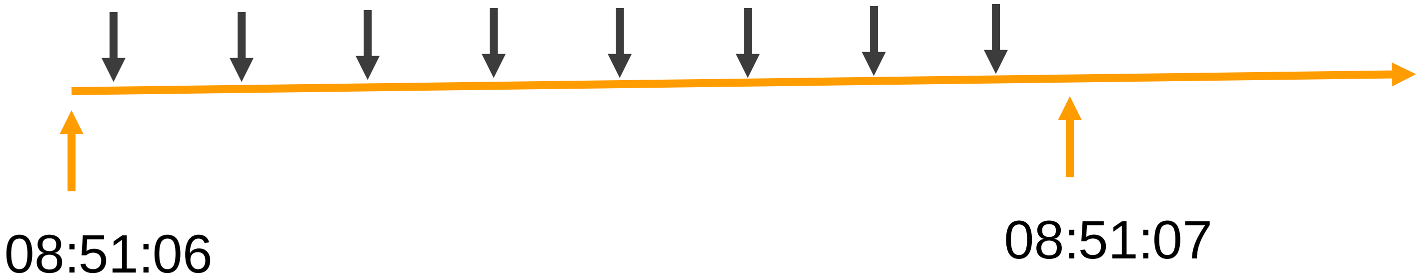
- 👍 ⊡ Add timestamps with milliseconds
- ⊡ Add metadata (log level, file, line and thread)
- ⊡ Use the same logs configuration in all sites
- ⊡ Keep number of logs to the bare minimum
- ⊡ Write logs with details
- ⊡ Prepare your logs to automatic monitoring

Logs

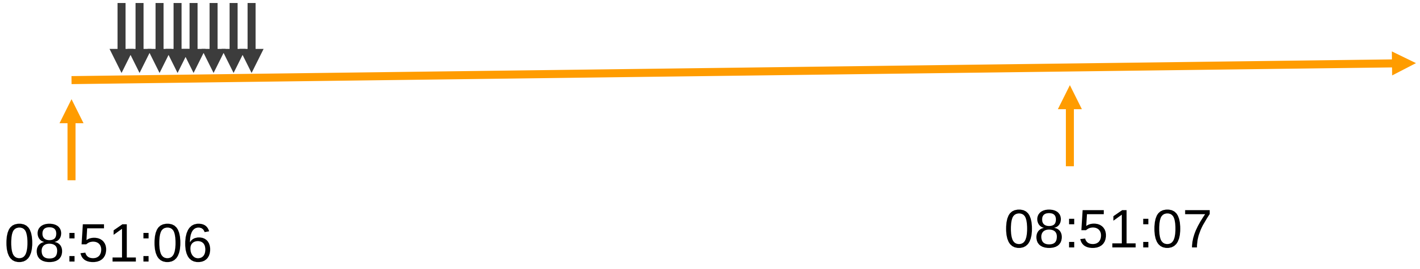


```
2023-09-16 10:03:45.678 INFO  main.cpp:105 [Thread-1]:
2023-09-16 10:05:00.256 INFO  main.cpp:110 [Thread-1]:
2023-09-16 10:06:15.789 INFO  user_login.cpp:55 [Threac
```

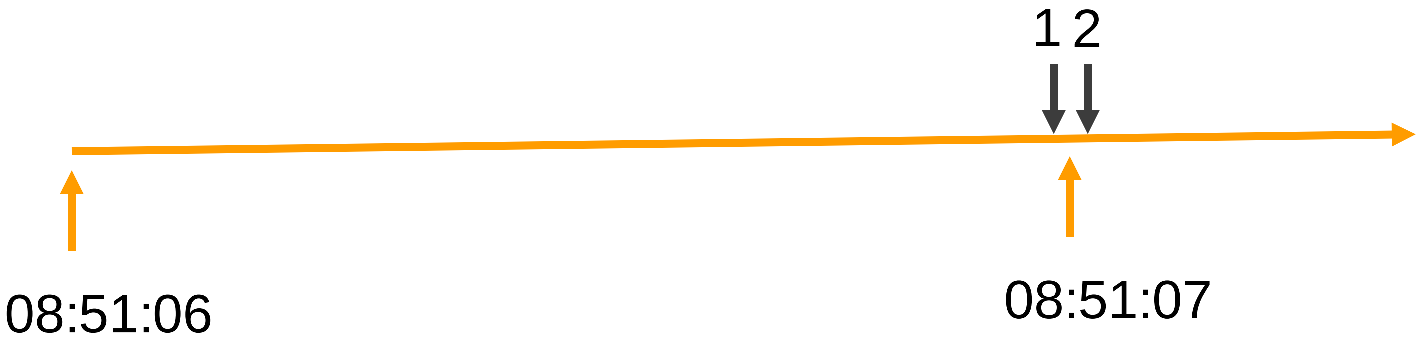
Logs



Logs



Logs



Logs

👍 □ Add timestamps with milliseconds

□ Add metadata (log level, file, line and thread)

□ Use the same logs configuration in all sites

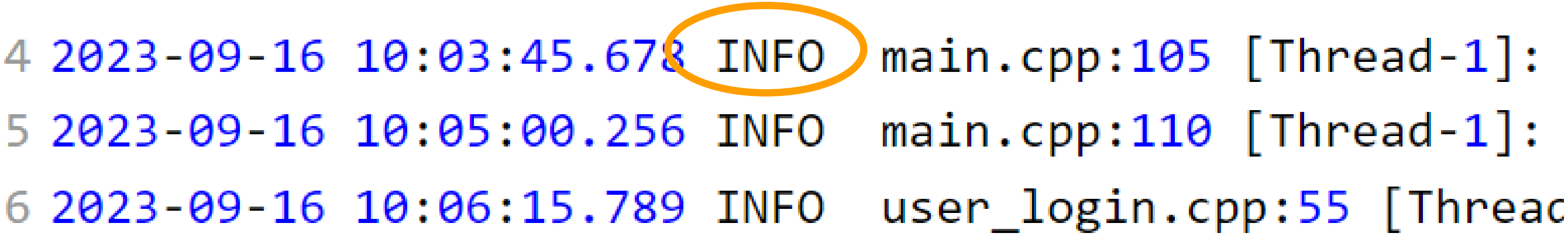
□ Keep number of logs to the bare minimum

□ Write logs with details

□ Prepare your logs to automatic monitoring

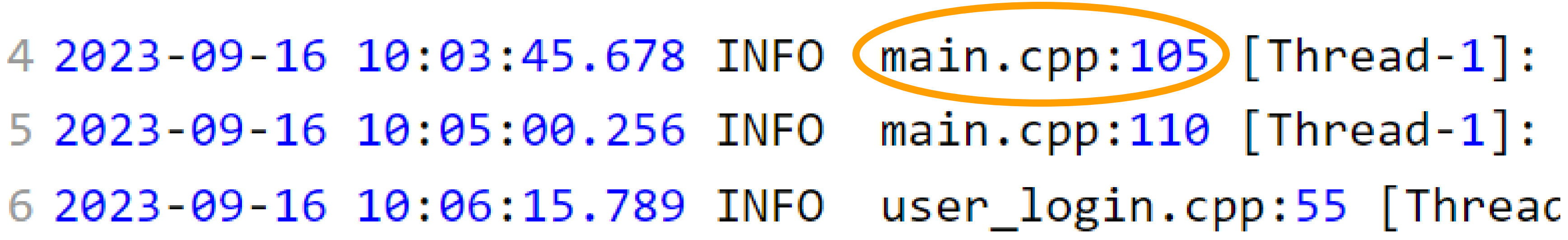
Logs

Log level



```
2023-09-16 10:03:45.678 INFO  main.cpp:105 [Thread-1]:
2023-09-16 10:05:00.256 INFO  main.cpp:110 [Thread-1]:
2023-09-16 10:06:15.789 INFO  user_login.cpp:55 [Threac
```

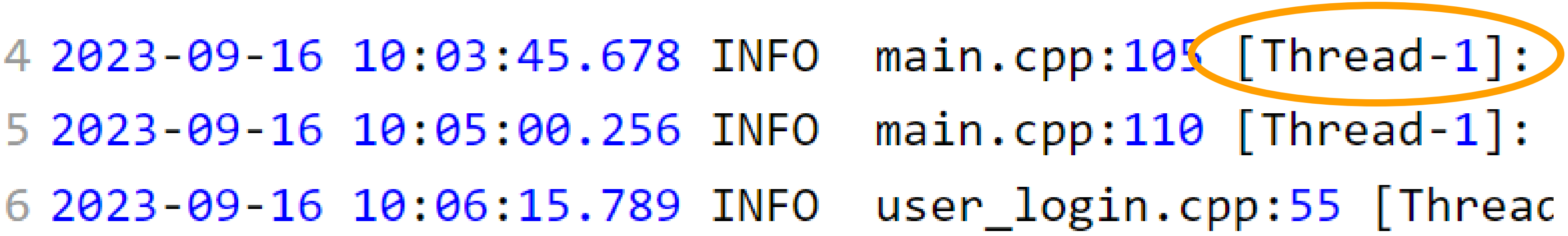
Logs



File:Line

```
2023-09-16 10:03:45.678 INFO  main.cpp:105 [Thread-1]:
2023-09-16 10:05:00.256 INFO  main.cpp:110 [Thread-1]:
2023-09-16 10:06:15.789 INFO  user_login.cpp:55 [Threac
```

Thread id



```
2023-09-16 10:03:45.678 INFO  main.cpp:105 [Thread-1]:
2023-09-16 10:05:00.256 INFO  main.cpp:110 [Thread-1]:
2023-09-16 10:06:15.789 INFO  user_login.cpp:55 [Threac
```

Logs

- 👍 ⊡ Add timestamps with milliseconds
- ⊡ Add metadata (log level, file, line and thread)
- ⊡ Use the same logs configuration in all sites
- ⊡ Keep number of logs to the bare minimum
- ⊡ Write logs with details
- ⊡ Prepare your logs to automatic monitoring

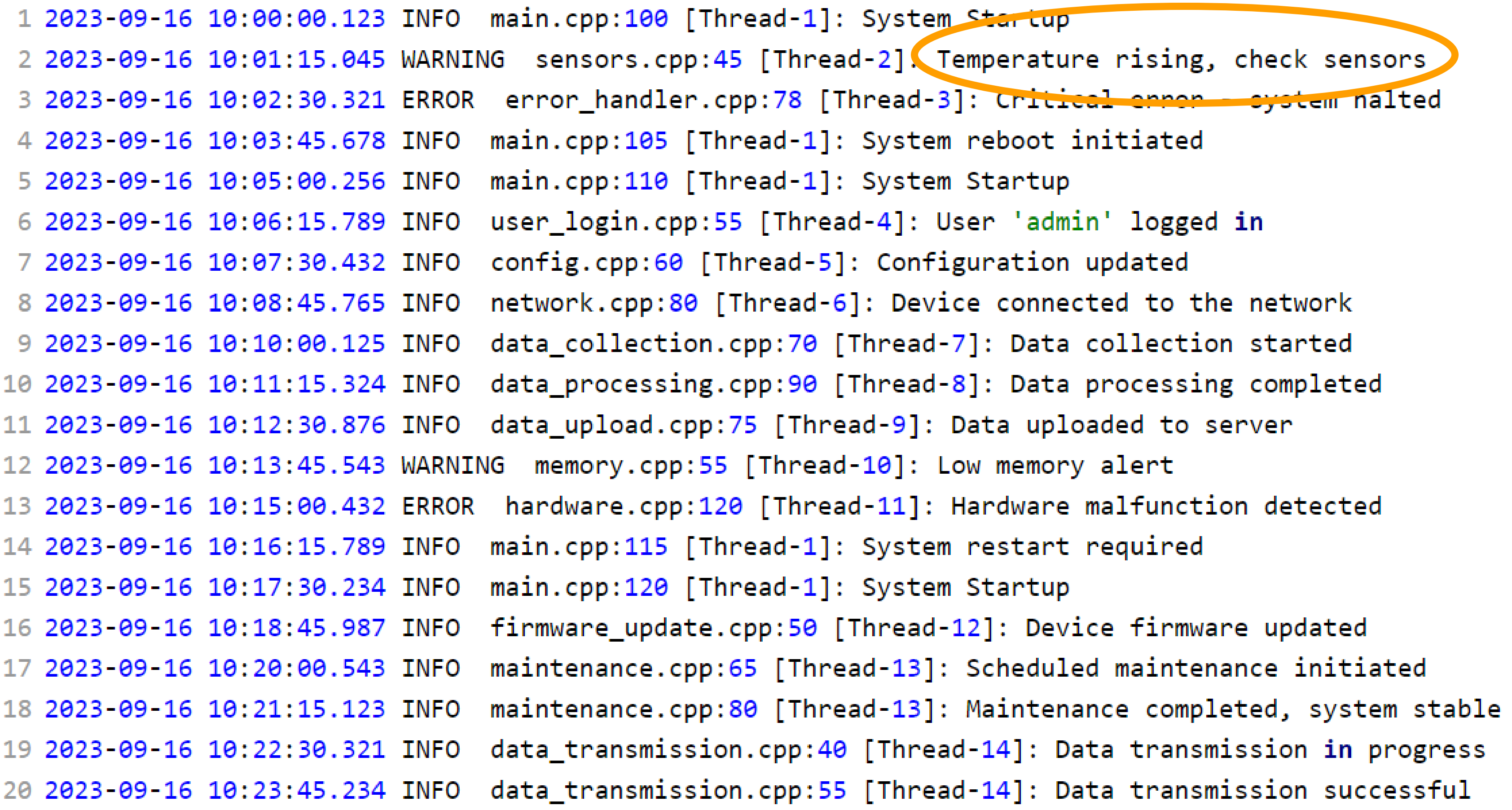
Logs

- 👍 ⊡ Add timestamps with milliseconds
- ⊡ Add metadata (log level, file, line and thread)
- ⊡ Use the same logs configuration in all sites
- ⊡ Keep number of logs to the bare minimum
- ⊡ Write logs with details
- ⊡ Prepare your logs to automatic monitoring

Logs

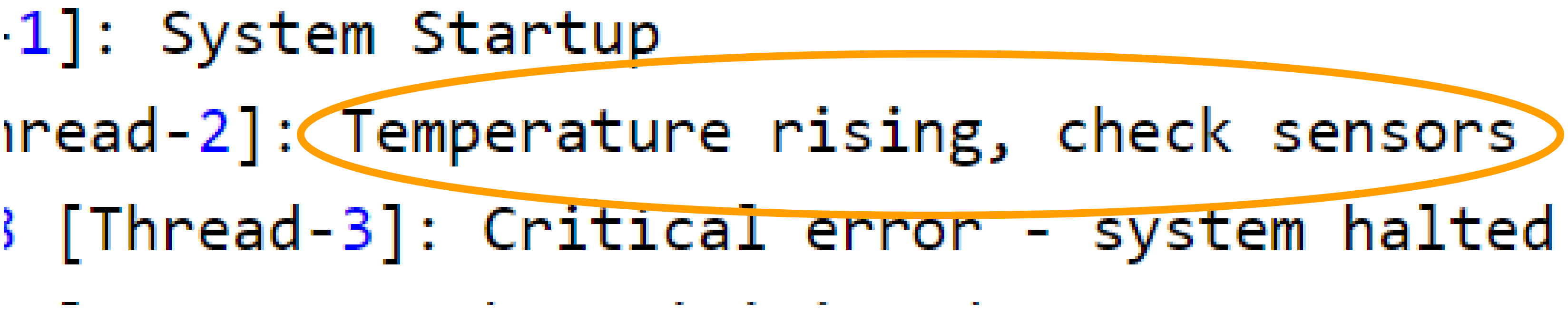
- □ Add timestamps with milliseconds
- □ Add metadata (log level, file, line and thread)
- □ Use the same logs configuration in all sites
- □ Keep number of logs to the bare minimum
- □ Write logs with details
- □ Prepare your logs to automatic monitoring

Logs



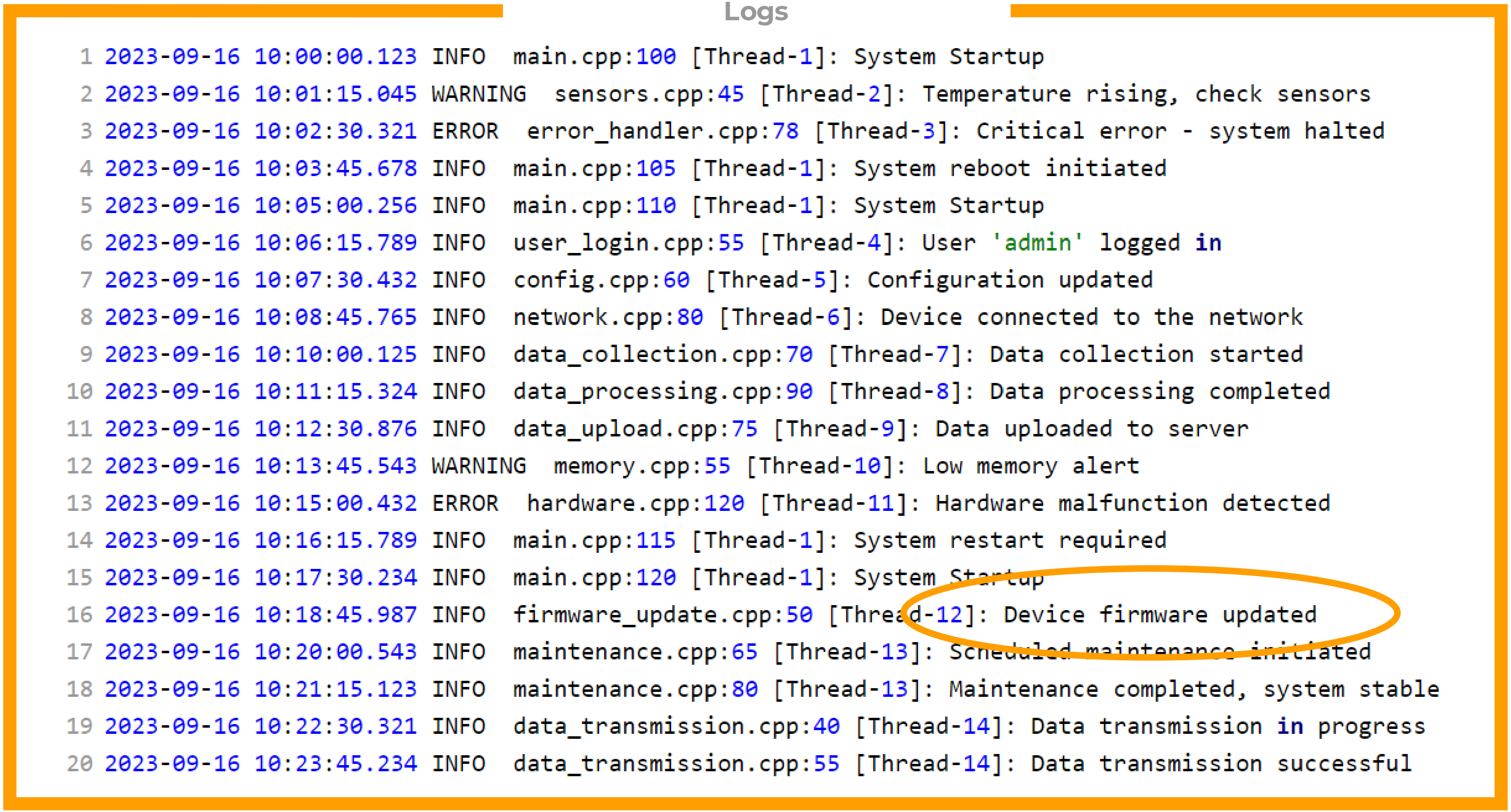
```
2023-09-16 10:00:00.123 INFO  main.cpp:100 [Thread-1]: System Startup
2023-09-16 10:01:15.045 WARNING  sensors.cpp:45 [Thread-2]: Temperature rising, check sensors
2023-09-16 10:02:30.321 ERROR  error_handler.cpp:78 [Thread-3]: Critical error - system halted
2023-09-16 10:03:45.678 INFO  main.cpp:105 [Thread-1]: System reboot initiated
2023-09-16 10:05:00.256 INFO  main.cpp:110 [Thread-1]: System Startup
2023-09-16 10:06:15.789 INFO  user_login.cpp:55 [Thread-4]: User 'admin' logged in
2023-09-16 10:07:30.432 INFO  config.cpp:60 [Thread-5]: Configuration updated
2023-09-16 10:08:45.765 INFO  network.cpp:80 [Thread-6]: Device connected to the network
2023-09-16 10:10:00.125 INFO  data_collection.cpp:70 [Thread-7]: Data collection started
2023-09-16 10:11:15.324 INFO  data_processing.cpp:90 [Thread-8]: Data processing completed
2023-09-16 10:12:30.876 INFO  data_upload.cpp:75 [Thread-9]: Data uploaded to server
2023-09-16 10:13:45.543 WARNING  memory.cpp:55 [Thread-10]: Low memory alert
2023-09-16 10:15:00.432 ERROR  hardware.cpp:120 [Thread-11]: Hardware malfunction detected
2023-09-16 10:16:15.789 INFO  main.cpp:115 [Thread-1]: System restart required
2023-09-16 10:17:30.234 INFO  main.cpp:120 [Thread-1]: System Startup
2023-09-16 10:18:45.987 INFO  firmware_update.cpp:50 [Thread-12]: Device firmware updated
2023-09-16 10:20:00.543 INFO  maintenance.cpp:65 [Thread-13]: Scheduled maintenance initiated
2023-09-16 10:21:15.123 INFO  maintenance.cpp:80 [Thread-13]: Maintenance completed, system stable
2023-09-16 10:22:30.321 INFO  data_transmission.cpp:40 [Thread-14]: Data transmission in progress
2023-09-16 10:23:45.234 INFO  data_transmission.cpp:55 [Thread-14]: Data transmission successful
```

To which temperature?



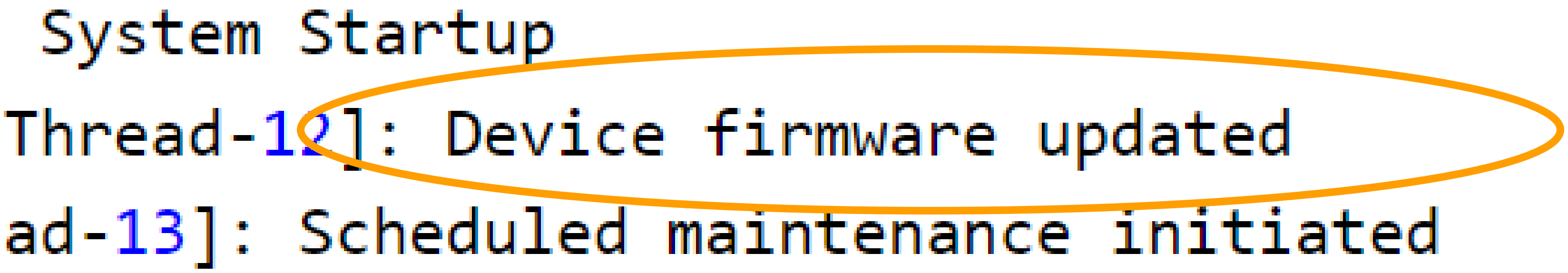
```
-1]: System Startup
read-2]: Temperature rising, check sensors
3 [Thread-3]: Critical error - system halted
```

Logs



```
2023-09-16 10:00:00.123 INFO  main.cpp:100 [Thread-1]: System Startup
2023-09-16 10:01:15.045 WARNING  sensors.cpp:45 [Thread-2]: Temperature rising, check sensors
2023-09-16 10:02:30.321 ERROR  error_handler.cpp:78 [Thread-3]: Critical error - system halted
2023-09-16 10:03:45.678 INFO  main.cpp:105 [Thread-1]: System reboot initiated
2023-09-16 10:05:00.256 INFO  main.cpp:110 [Thread-1]: System Startup
2023-09-16 10:06:15.789 INFO  user_login.cpp:55 [Thread-4]: User 'admin' logged in
2023-09-16 10:07:30.432 INFO  config.cpp:60 [Thread-5]: Configuration updated
2023-09-16 10:08:45.765 INFO  network.cpp:80 [Thread-6]: Device connected to the network
2023-09-16 10:10:00.125 INFO  data_collection.cpp:70 [Thread-7]: Data collection started
2023-09-16 10:11:15.324 INFO  data_processing.cpp:90 [Thread-8]: Data processing completed
2023-09-16 10:12:30.876 INFO  data_upload.cpp:75 [Thread-9]: Data uploaded to server
2023-09-16 10:13:45.543 WARNING  memory.cpp:55 [Thread-10]: Low memory alert
2023-09-16 10:15:00.432 ERROR  hardware.cpp:120 [Thread-11]: Hardware malfunction detected
2023-09-16 10:16:15.789 INFO  main.cpp:115 [Thread-1]: System restart required
2023-09-16 10:17:30.234 INFO  main.cpp:120 [Thread-1]: System Startup
2023-09-16 10:18:45.987 INFO  firmware_update.cpp:50 [Thread-12]: Device firmware updated
2023-09-16 10:20:00.543 INFO  maintenance.cpp:65 [Thread-13]: Scheduled maintenance initiated
2023-09-16 10:21:15.123 INFO  maintenance.cpp:80 [Thread-13]: Maintenance completed, system stable
2023-09-16 10:22:30.321 INFO  data_transmission.cpp:40 [Thread-14]: Data transmission in progress
2023-09-16 10:23:45.234 INFO  data_transmission.cpp:55 [Thread-14]: Data transmission successful
```

To which version? from which version?



```
 System Startup
Thread-12]: Device firmware updated
ad-13]: Scheduled maintenance initiated
```

Logs

👍

⊡ Add timestamps with milliseconds

⊡ Add metadata (log level, file, line and thread)

⊡ Use the same logs configuration in all sites

⊡ Keep number of logs to the bare minimum

⊡ Write logs with details

⊡ Prepare your logs to automatic monitoring

Logs

```
2024-09-21 10:10:00.125 INFO ride.cpp:70 [Thread-7]: gpioC7 is 0
2024-09-21 10:10:00.125 INFO ride.cpp:93 [Thread-7]: gpioA3 is 1
2024-09-21 10:10:00.126 INFO ride.cpp:102 [Thread-7]: gpioA8 is 1
```

Logs

~~2024-09-21 10:10:00.125 INFO ride.cpp:70 [Thread-7]: gpioC7 is 0~~

~~2024-09-21 10:10:00.125 INFO ride.cpp:93 [Thread-7]: gpioA3 is 1~~

~~2024-09-21 10:10:00.126 INFO ride.cpp:102 [Thread-7]: gpioA8 is 1~~

```
2024-09-21 10:10:00.126 INFO ride.cpp:102 [Thread-7]: gpioC7 is 0, gpioA3 is 1 and gpioA8 is 1
```

> *Milliseconds*
>
> *Metadata*
>
> *Same configuration*
>
> *Bare minimum*
>
> *With details*
>
> *Prepare for automation*

Agenda

- □ Operating Systems
- □ Threads
- □ Layer Separation
- □ Network Problems
- □ Message Structure
- □ Simulators
- □ Logs
- □ Monitoring

1

Identify Errors Proactively: Don't Wait for Customer Complaints

1

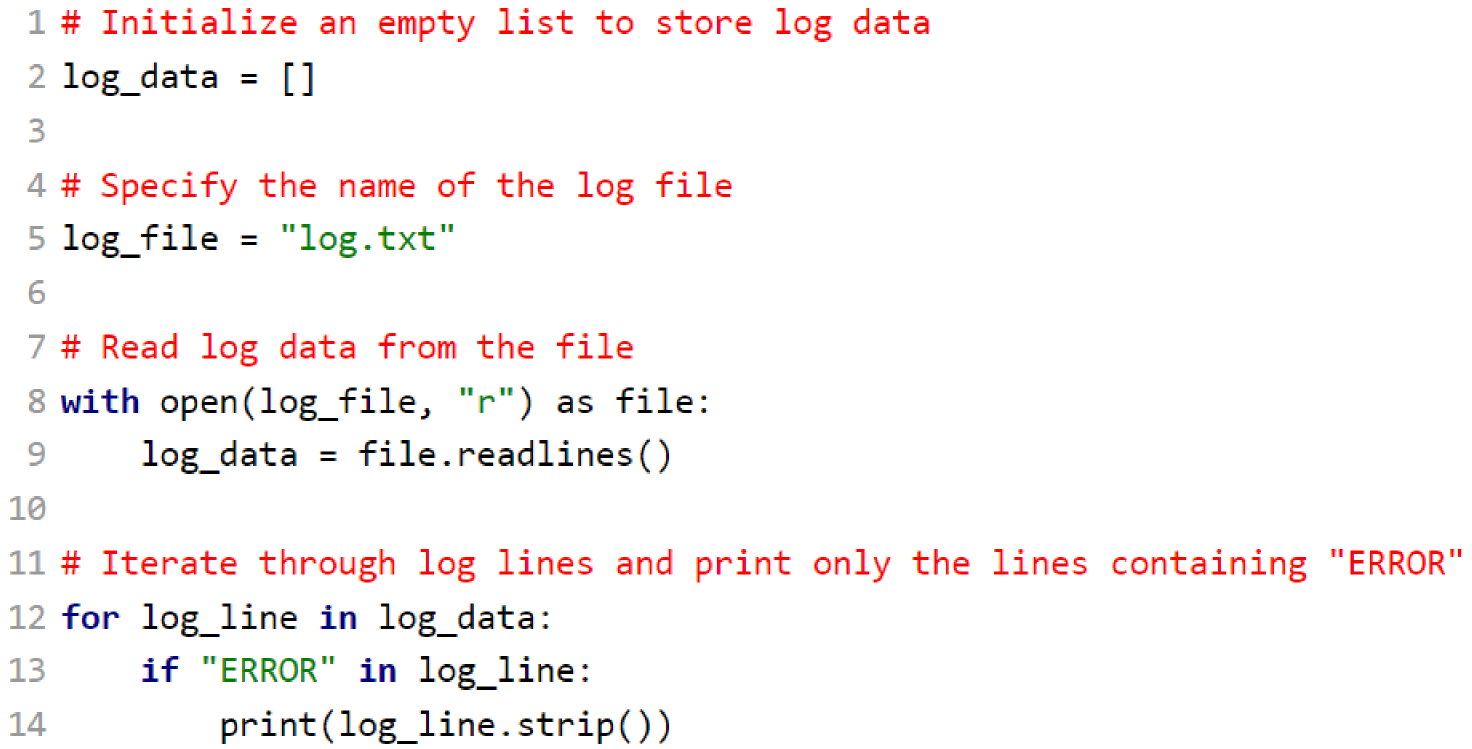
Identify Errors Proactively:
Don't Wait for Customer Complaints

2

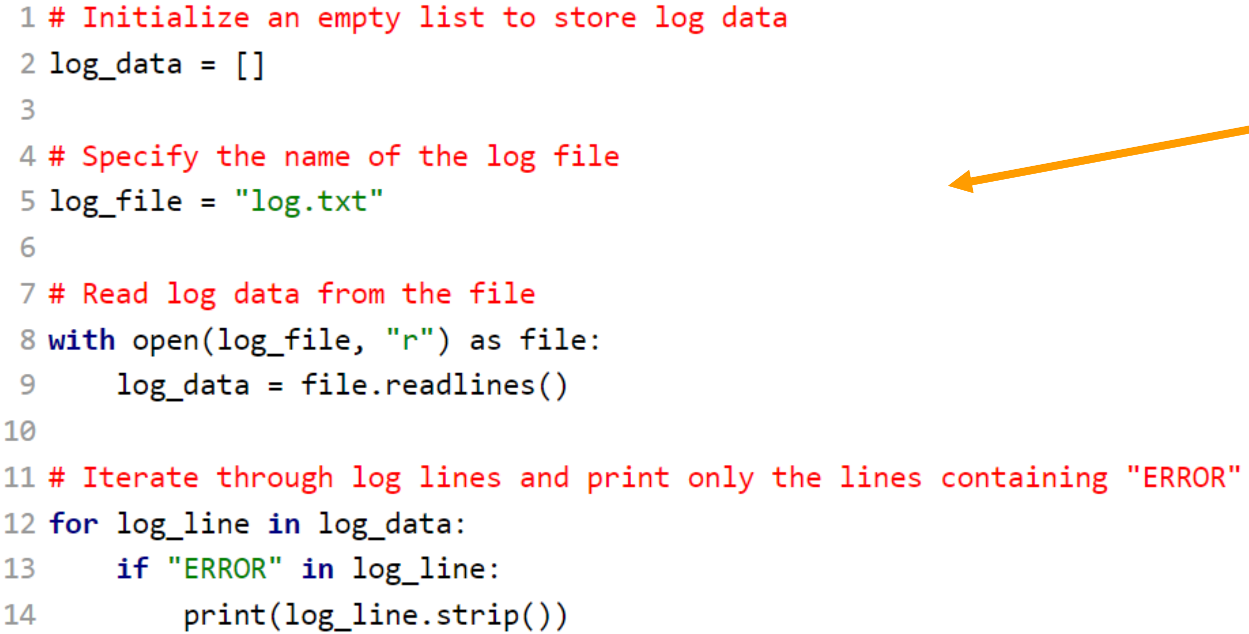
If it's not automated, it won't get done

Monitoring

- Write a Python script - start with the errors
- Monitor periodic activities
- Count interesting events and create summary for each unit
- Compare between units



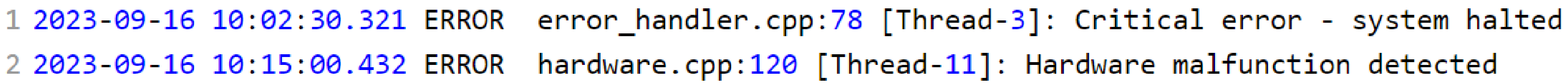
```python
# Initialize an empty list to store log data
log_data = []

# Specify the name of the log file
log_file = "log.txt"

# Read log data from the file
with open(log_file, "r") as file:
    log_data = file.readlines()

# Iterate through log lines and print only the lines containing "ERROR"
for log_line in log_data:
    if "ERROR" in log_line:
        print(log_line.strip())
```



```python
# Initialize an empty list to store log data
log_data = []

# Specify the name of the log file
log_file = "log.txt"

# Read log data from the file
with open(log_file, "r") as file:
    log_data = file.readlines()

# Iterate through log lines and print only the lines containing "ERROR"
for log_line in log_data:
    if "ERROR" in log_line:
        print(log_line.strip())
```

Python script

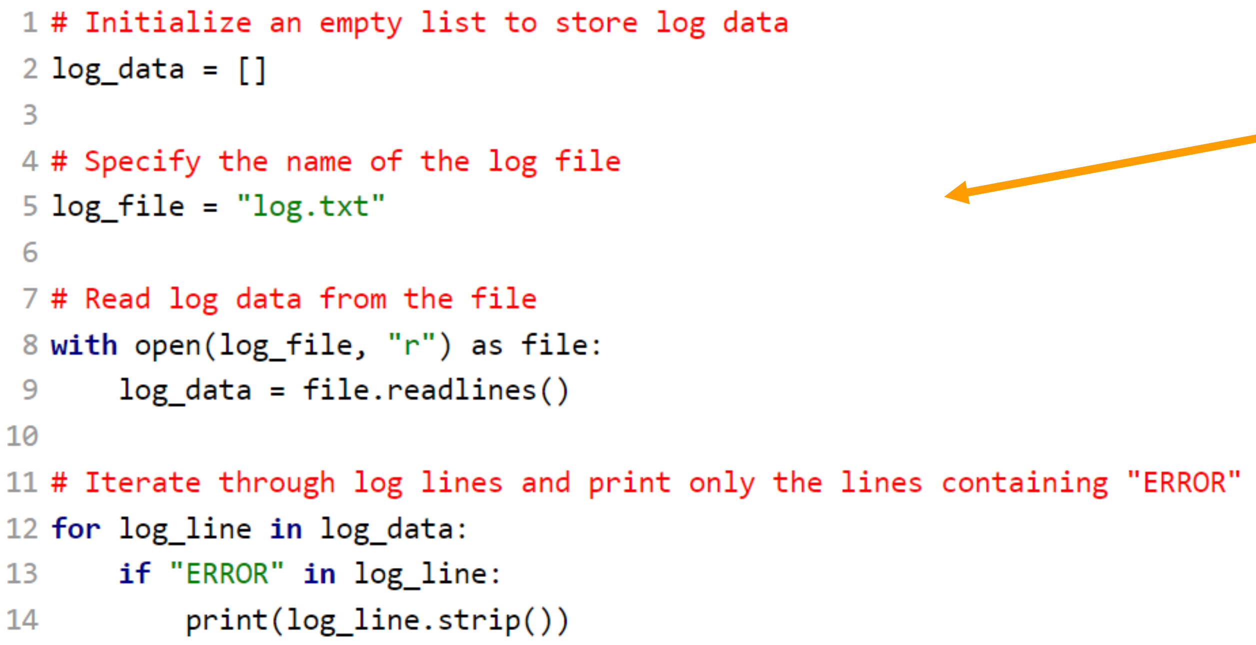
Output



```
2023-09-16 10:02:30.321 ERROR  error_handler.cpp:78 [Thread-3]: Critical error - system halted
2023-09-16 10:15:00.432 ERROR  hardware.cpp:120 [Thread-11]: Hardware malfunction detected
```

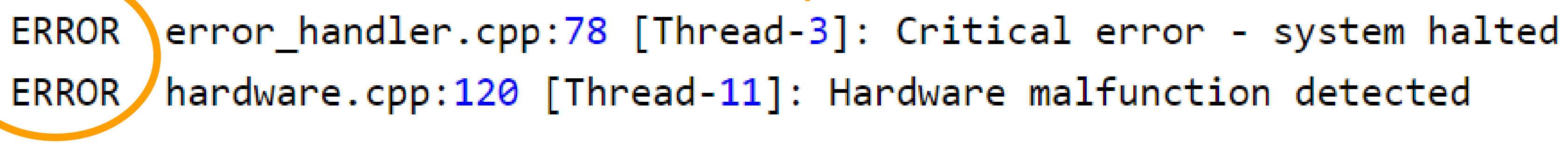
Monitoring

Python script



```python
# Initialize an empty list to store log data
log_data = []

# Specify the name of the log file
log_file = "log.txt"

# Read log data from the file
with open(log_file, "r") as file:
    log_data = file.readlines()

# Iterate through log lines and print only the lines containing "ERROR"
for log_line in log_data:
    if "ERROR" in log_line:
        print(log_line.strip())
```

Output

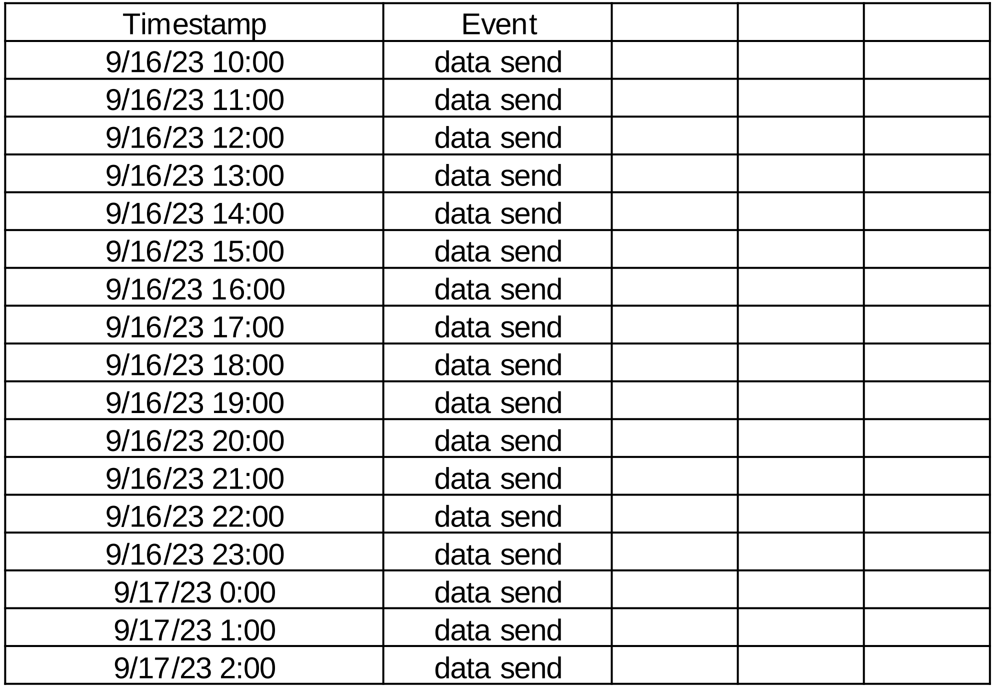


```
ERROR error_handler.cpp:78 [Thread-3]: Critical error - system halted
ERROR hardware.cpp:120 [Thread-11]: Hardware malfunction detected
```

Monitoring

- Write a Python script - start with the errors
- Monitor periodic activities
- Count interesting events and create summary for each unit
- Compare between units

Monitoring



Timestamp	Event			
9/16/23 10:00	data send			
9/16/23 11:00	data send			
9/16/23 12:00	data send			
9/16/23 13:00	data send			
9/16/23 14:00	data send			
9/16/23 15:00	data send			
9/16/23 16:00	data send			
9/16/23 17:00	data send			
9/16/23 18:00	data send			
9/16/23 19:00	data send			
9/16/23 20:00	data send			
9/16/23 21:00	data send			
9/16/23 22:00	data send			
9/16/23 23:00	data send			
9/17/23 0:00	data send			
9/17/23 1:00	data send			
9/17/23 2:00	data send			

- Write a Python script - start with the errors
- Monitor periodic activities
- Count interesting events and create summary for each unit
- Compare between units

Monitoring

Serial number	Firmware	Errors	Last time	Data sending event	Max temperature (c)
346523	F2	0	3/4/2024 10:15:32	24	65

Monitoring

- Write a Python script - start with the errors
- Monitor periodic activities
- Count interesting events and create summary for each unit
- Compare between units

Monitoring

Serial number	Firmware	Errors	Last time	Data sending event	Max temperature (c)
346523	F2	0	3/4/2024 10:15:32	24	65
345251	F1	0	3/4/2024 10:12:15	48	68
723642	F2	87	3/4/2024 10:16:52	12	75
548328	F2	0	3/4/2024 10:14:09	1	59

Monitoring

Serial number	Firmware	Errors	Last time	Data sending event	Max temperature (c)
346523	F2	0	3/4/2024 10:15:32	24	65
345251	F1	0	3/4/2024 10:12:15	48	68
723642	F2	87	3/4/2024 10:16:52	12	75
548328	F2	0	3/4/2024 10:14:09	1	59

Monitoring

Serial number	Firmware	Errors	Last time	Data sending event	Max temperature (c)
346523	F2	0	3/4/2024 10:15:32	24	65
345251	F1	0	3/4/2024 10:12:15	48	68
723642	F2	87	3/4/2024 10:16:52	12	75
548328	F2	0	3/4/2024 10:14:09	1	59

“

> *If it's not automated, it won't get done*

Summary

- Use an operating system for complex systems with soft real-time requirements

Summary

- Use an operating system for complex systems with soft real-time requirements
- Keep number of threads to the bare minimum

Summary

- Use an operating system for complex systems with soft real-time requirements
- Keep number of threads to the bare minimum
- Separate logic layer from hardware layer

Summary

- Use an operating system for complex systems with soft real-time requirements
- Keep number of threads to the bare minimum
- Separate logic layer from hardware layer
- Disconnect the logic from the network

Summary

- Use an operating system for complex systems with soft real-time requirements
- Keep number of threads to the bare minimum
- Separate logic layer from hardware layer
- Disconnect the logic from the network
- Design your protocol in a way you could always bounce back from a “bad” message

Summary

- Use an operating system for complex systems with soft real-time requirements
- Keep number of threads to the bare minimum
- Separate logic layer from hardware layer
- Disconnect the logic from the network
- Design your protocol in a way you could always bounce back from a “bad” message
- Work with simulators

Summary

- Use an operating system for complex systems with soft real-time requirements
- Keep number of threads to the bare minimum
- Separate logic layer from hardware layer
- Disconnect the logic from the network
- Design your protocol in a way you could always bounce back from a “bad” message
- Work with simulators
- Logs: put timestamp with milliseconds…

Summary

- Use an operating system for complex systems with soft real-time requirements
- Keep number of threads to the bare minimum
- Separate logic layer from hardware layer
- Disconnect the logic from the network
- Design your protocol in a way you could always bounce back from a “bad” message
- Work with simulators
- Logs: put timestamp with milliseconds…
- Monitoring: if it's not automated, it won't get done

Thank You!

Any questions?