Building Effective Embedded Systems:

Architectural Best Practices

GILI KAMMA

Cppcon
The C++ Conference

2023
October 01 - 06

I am Gili Kamma

I love technology & people

20 years of experience

R&D manager @Blitz motors

Things I've done:

- Board design
- Drivers and boot loaders - C
- Multi threaded applications - C++
- Backend – C# and Python

HELLO!

I am Gili Kamma

I love technology & people

20 years of experience

R&D manager @Blitz motors

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

Disclaimer #1

Every rule presented here comes with an exception

Software isn't black and white

Disclaimer #2

The code snippets in this presentation are for illustration

Agenda

- Operating Systems
- Threads
- Layer Separation
- Network Problems
- External Interfaces
- Simulators
- Logs
- Monitoring

Agenda

- Operating Systems
- Threads
- Layer Separation
- Network Problems
- External Interfaces
- Simulators
- Logs
- Monitoring

Operating Systems
To Be or Not To Be

Hard/Soft Real Time Requirements

Hard Real Time

- Timing constraints are extremely strict

Soft Real Time

- Flexible Timing (5-10 milliseconds)

Hard/Soft Real Time Requirements

Hard Real Time

- Timing constraints are extremely strict
- A guaranteed response time

Soft Real Time

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

Hard/Soft Real Time Requirements

Hard Real Time

- Timing constraints are extremely strict
- A guaranteed response time
- Microseconds

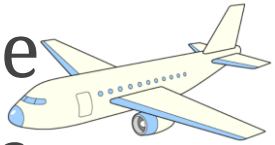
Soft Real Time

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

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

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?

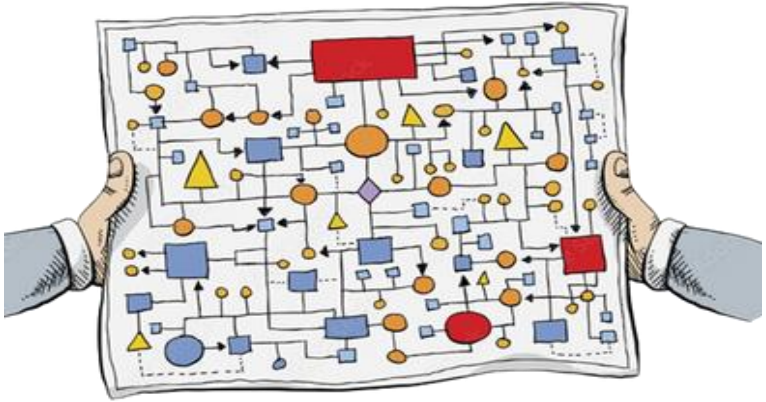
Less then 5 milliseconds – Don’t use an operating system*

*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	

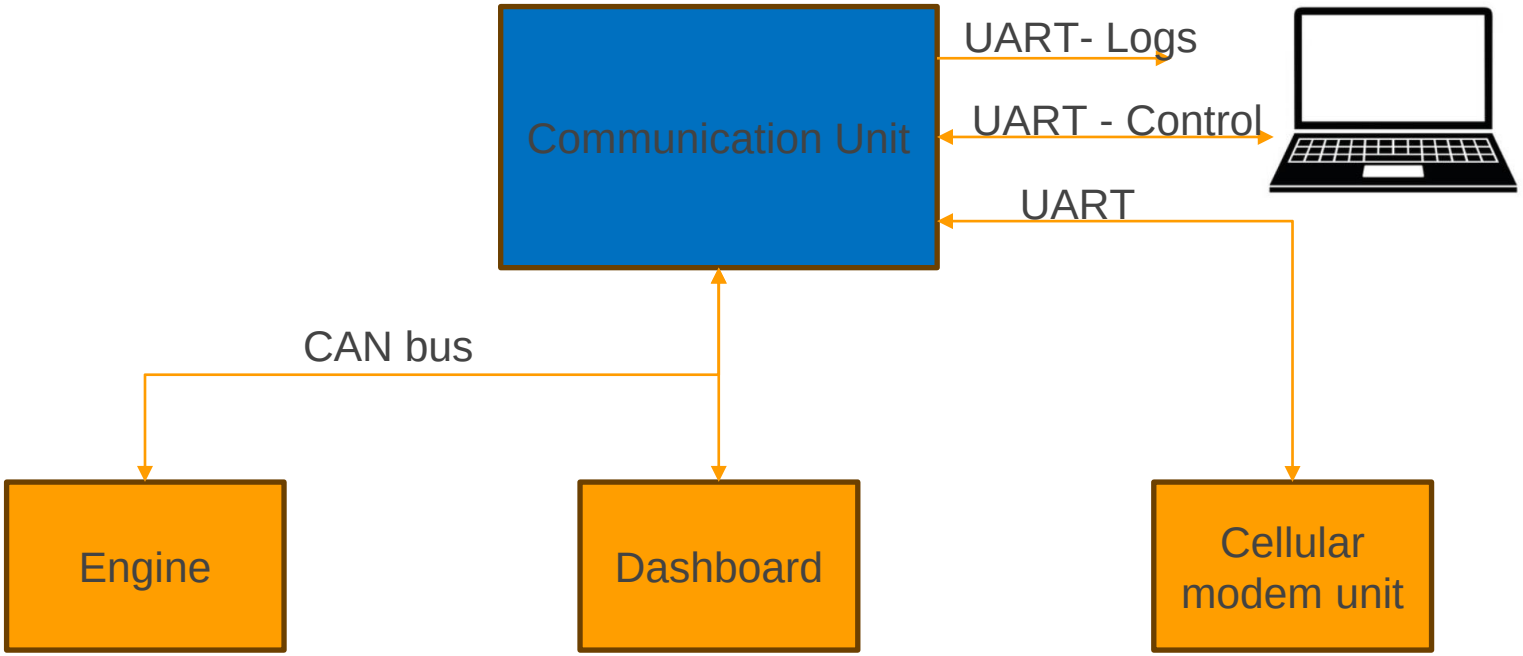
	Soft Real Time	**Hard Real Time**
Simple System	Don’t care	None
Complicated System	Operating system	

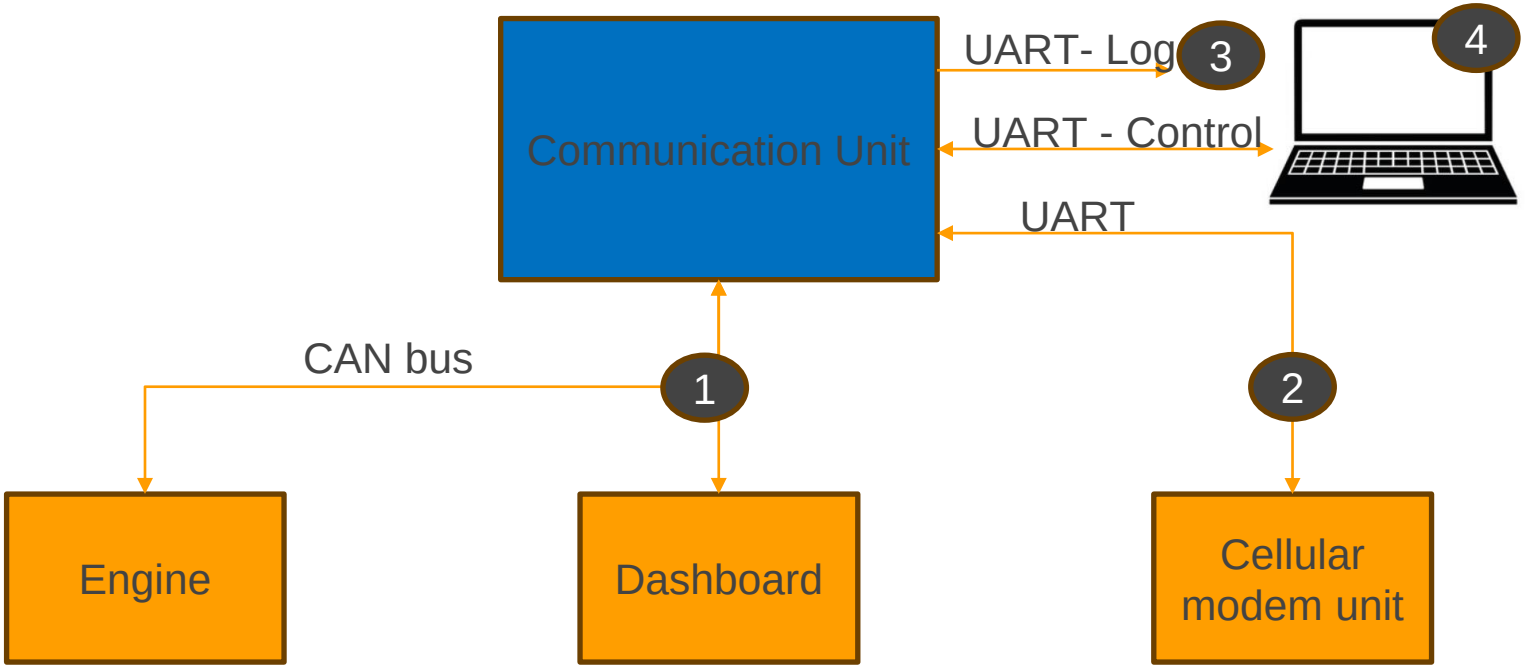
	Soft Real Time	**Hard Real Time**
Simple System	Don’t care	None
Complicated System	Operating system	?

	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

Threads


Communication Unit
UART- Logs
UART - Control
UART
CAN bus
Engine
Dashboard
Cellular modem unit

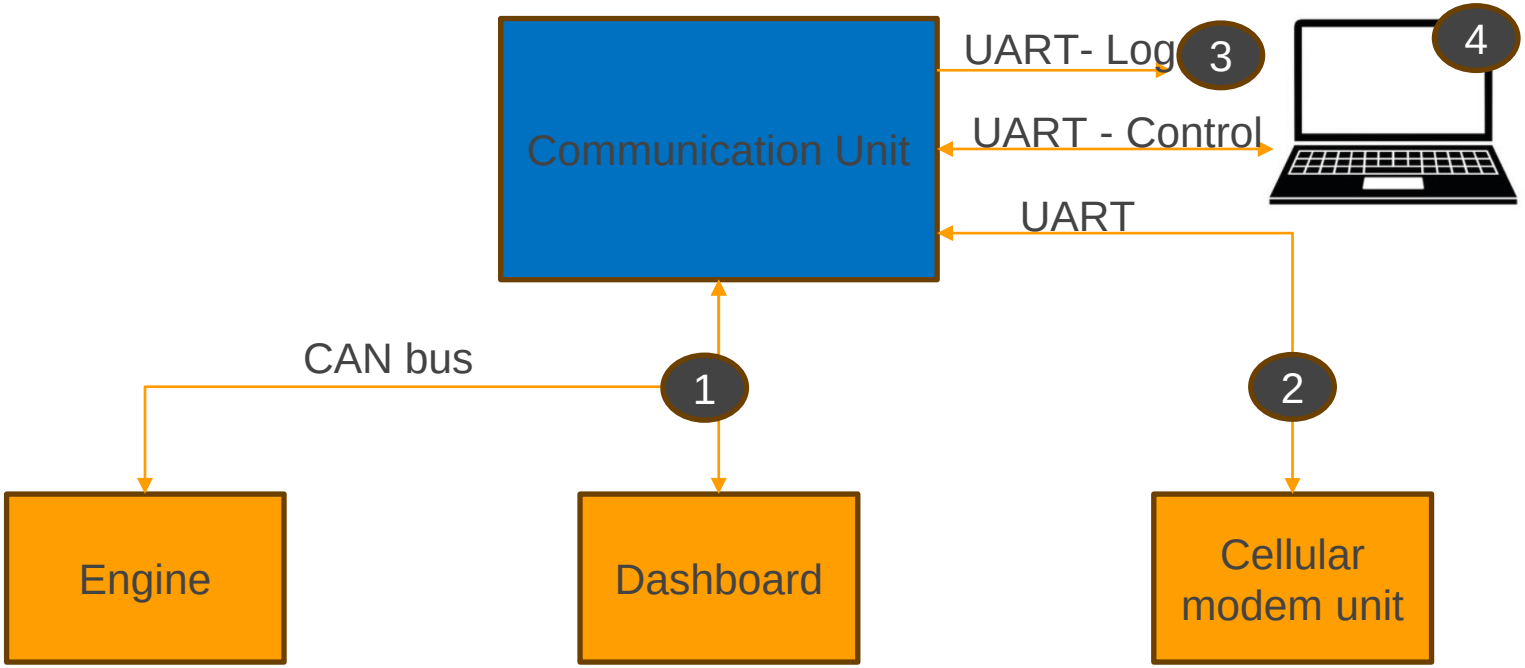


4 interfaces → complicated

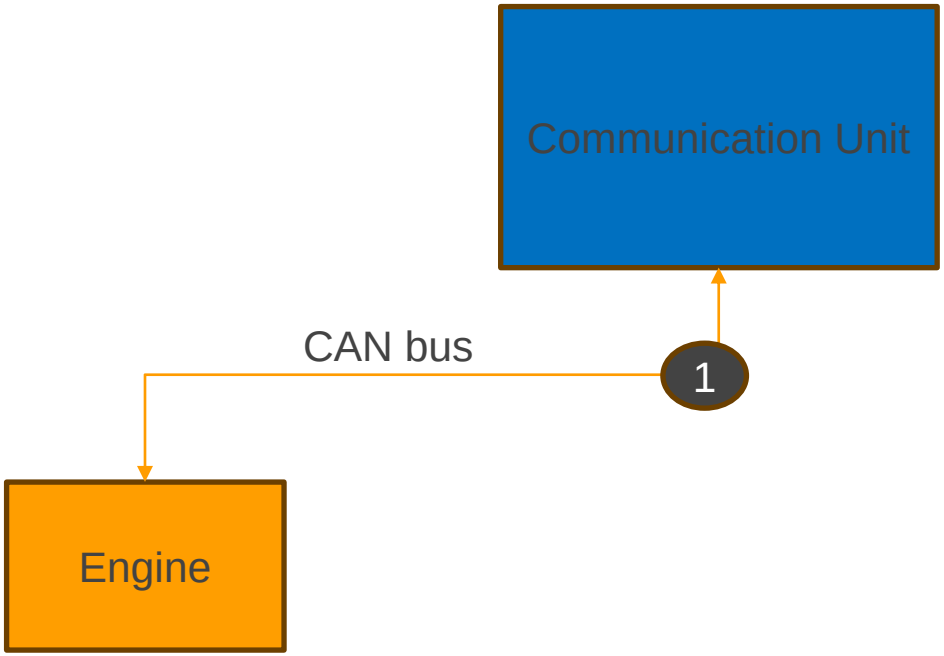
What level of time precision does the system require?

What level of time precision does the system require?

~100 milliseconds



operating system



operating system

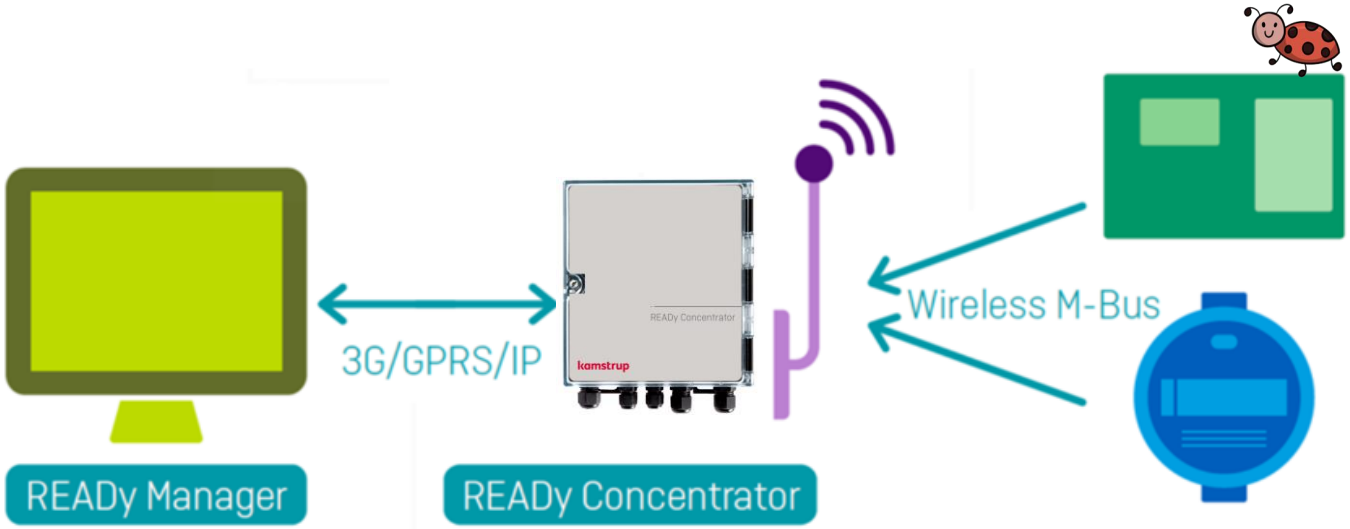
“

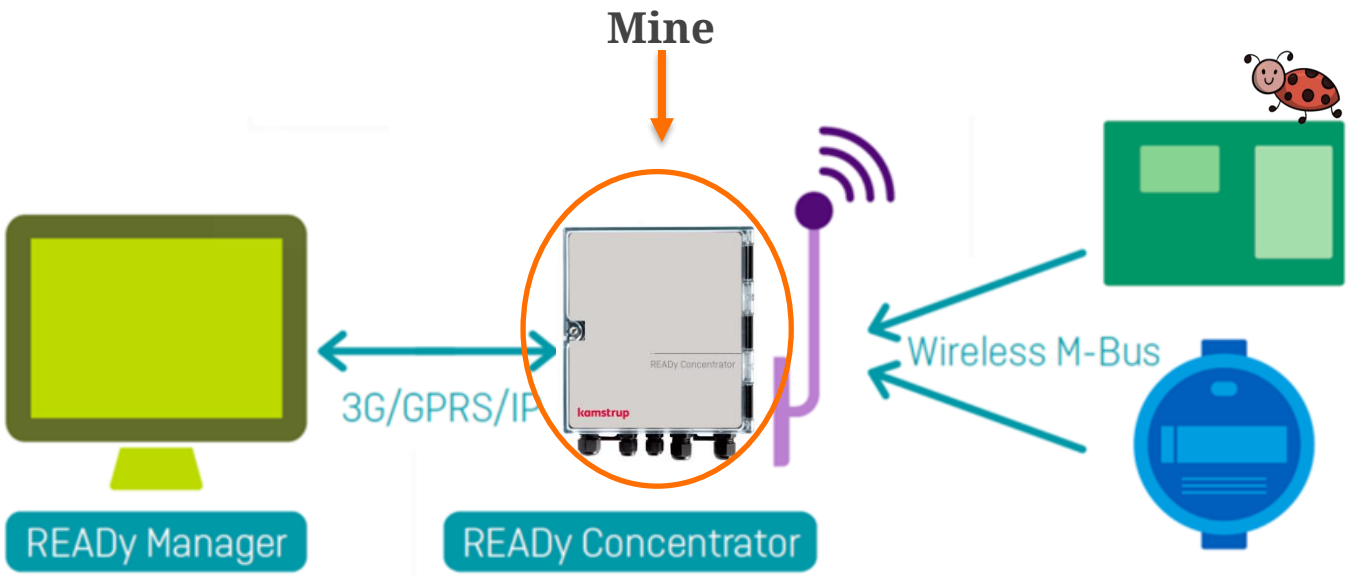
Use an operating system for complex systems with soft real-time requirements

Agenda

- Operating Systems
- Threads
- Layer Separation
- Network Problems
- External Interfaces
- Simulators
- Logs
- Monitoring

Threads


3G/GPRS/IP
kamstrup
READy Concentrator
Wireless M-Bus
READy Manager
READy Concentrator


Mine
3G/GPRS/IP
READy Concentrator
kamstrup
Wireless M-Bus
READy Manager
READy Concentrator

- Collects meter measurements from water meters (wireless) and sends them to the cloud by cellular modem or ethernet.
- Built to work with less than 1000 meters – in real life was expected to work with 7500 meters.
- Loss of data → loss of money.

- Collects meter measurements from water meters (wireless) and sends them to the cloud by cellular modem or ethernet.
- Built to work with less than 1000 meters – in real life was expected to work with 7500 meters.
- Loss of data → loss of money.

- Collects meter measurements from water meters (wireless) and sends them to the cloud by cellular modem or ethernet.
- Built to work with less than 1000 meters – in real life was expected to work with 7500 meters.
- Loss of data → loss of money.

- Loss of data in big sites (>5000 meters)
- Loss of data when there were network errors

→ None of the errors were seen in the lab.

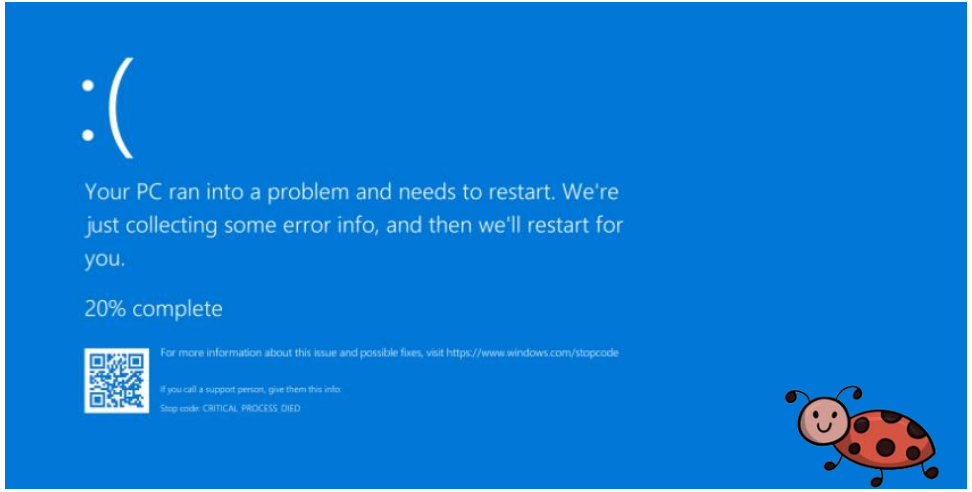
- Loss of data in big sites (>5000 meters)
- Loss of data when there were network errors
- None of the errors were seen in the lab.

I had unexplained resets
And I couldn’t understand why…

After some investigation, I found out that…

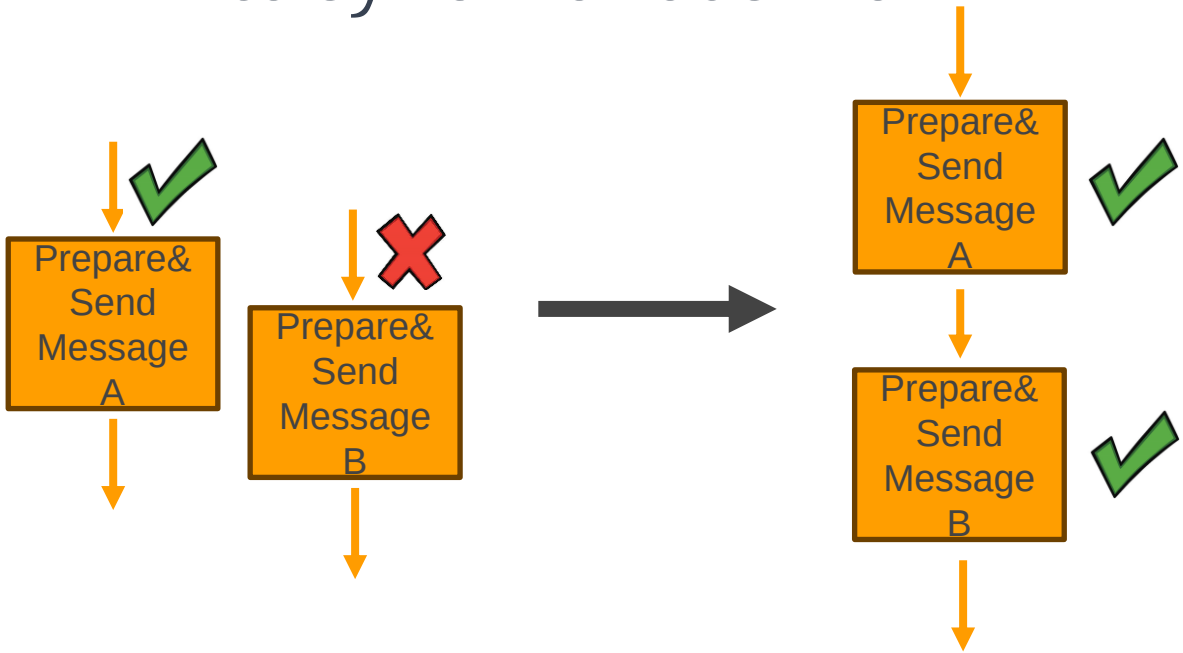
Two threads tried to create large messages about 5 MB each, at the same time.

The second one always failed



Solution

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

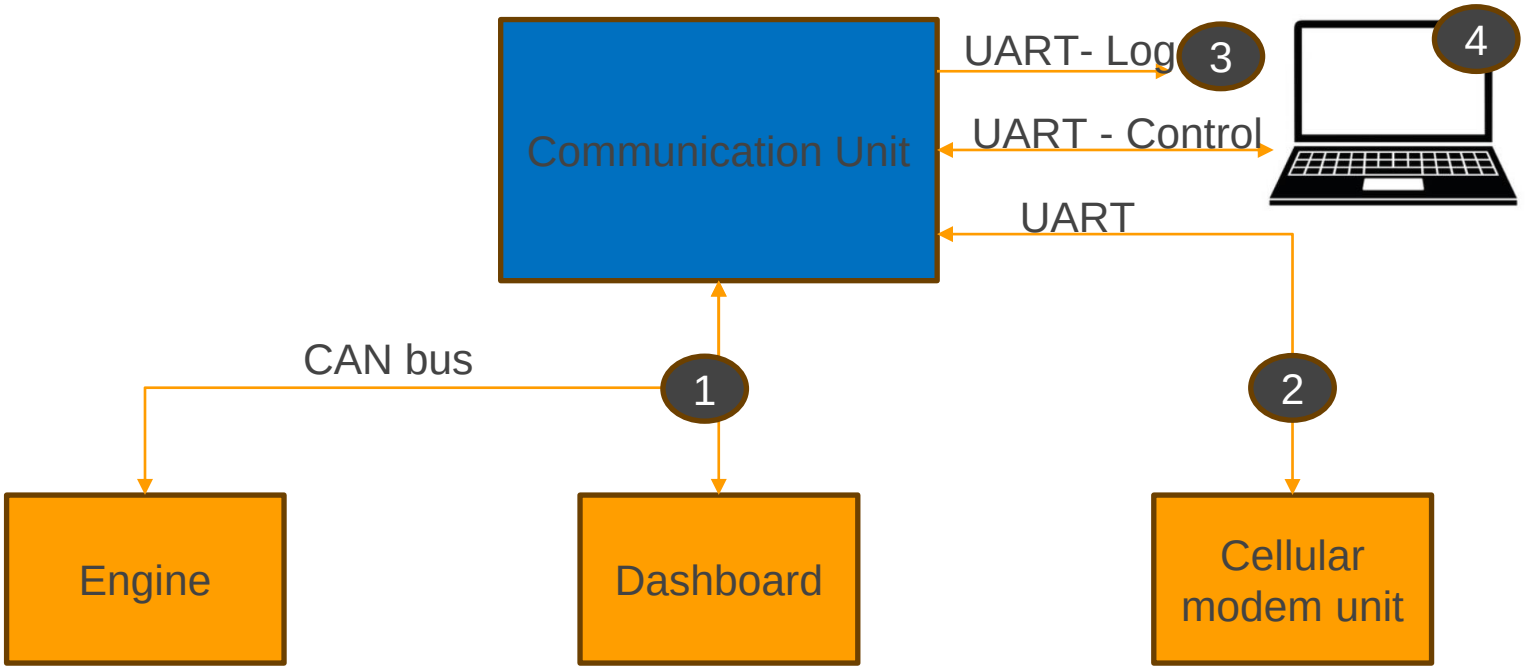
👍 Good practice:

Thread to each communication interface

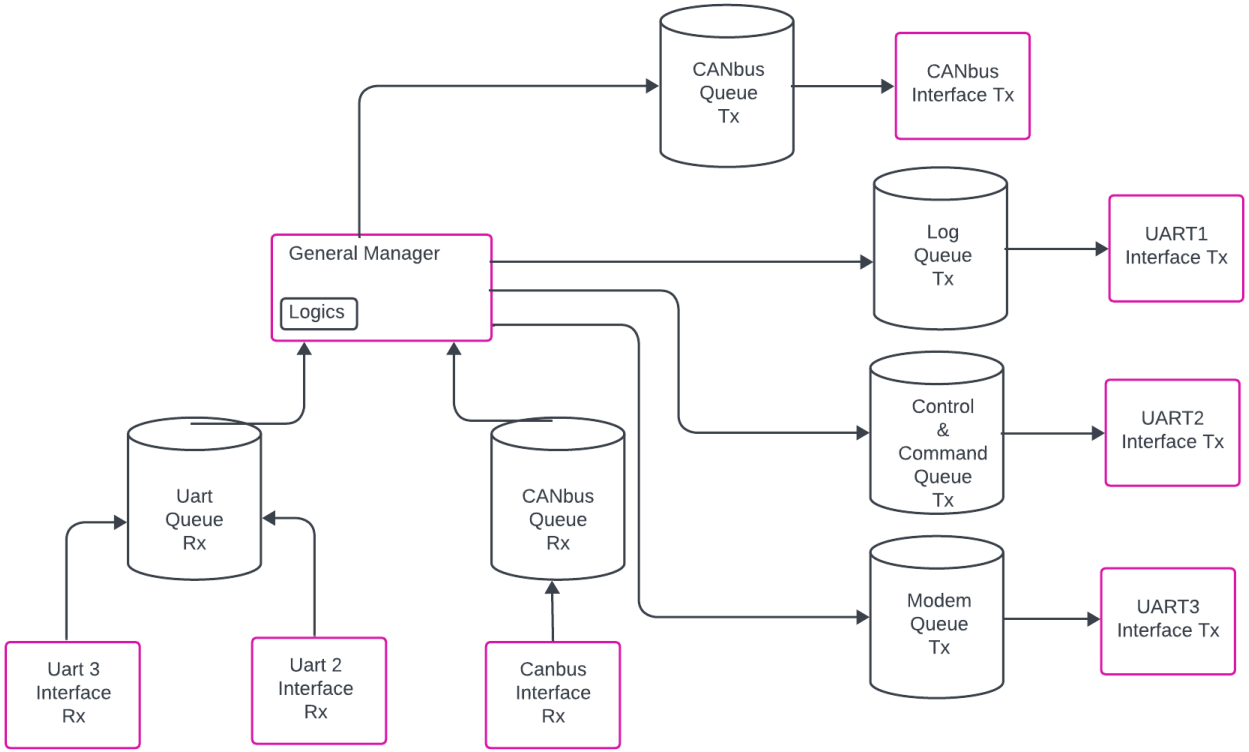
+

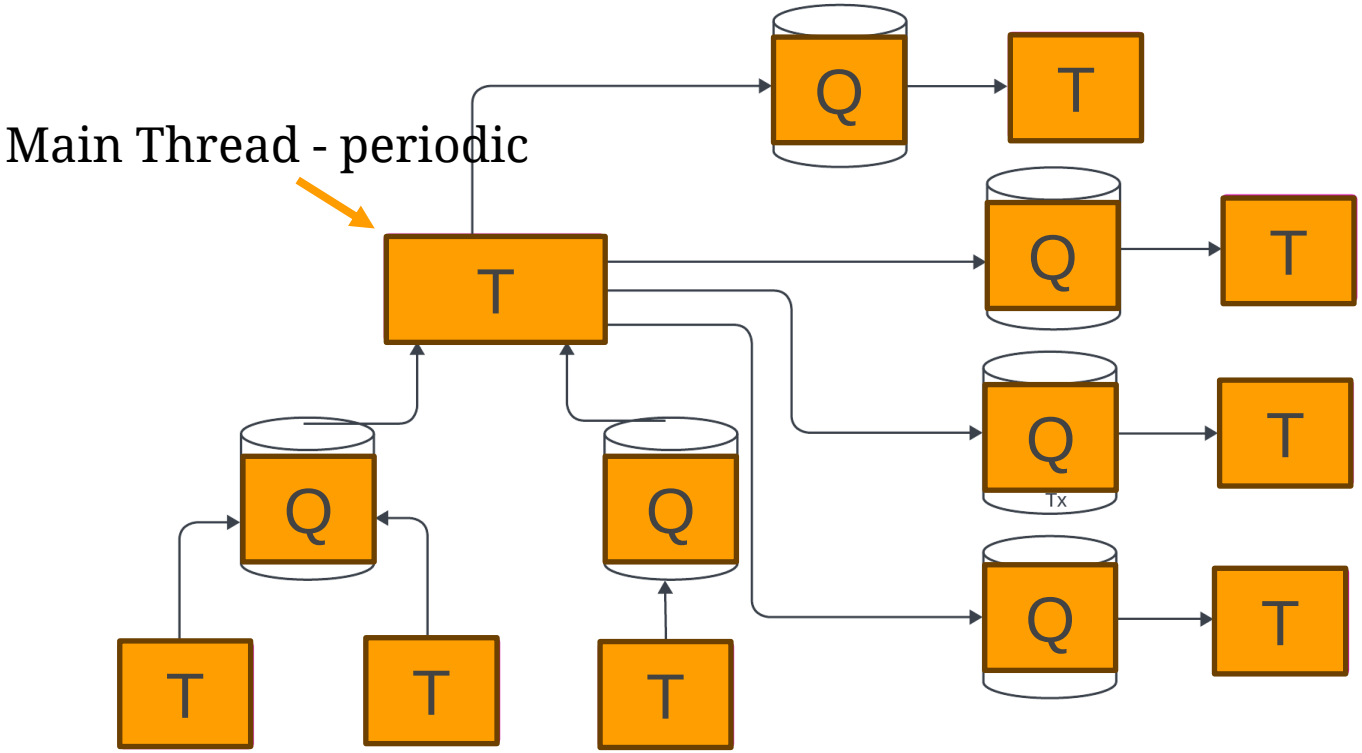
Periodic thread

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
General Manager
Logics
Log
Queue
Tx
UART1
Interface Tx
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


Main Thread - periodic
Q
T
Q
T
T
Q
T
Q
Tx
T
Q
Q
Q
T
T
T
T

```
1 void PerformPeriodicTask()
2 {
3     while(true)
4     {
5         GetInputSensors();
6         ReceiveAllMessages();
7         DoLogics()
8         SendAllMessages();
9         SetOutputs();
10        sleep(100);
11    }
12 }
```

```
1  void PerformPeriodicTask()
2  {
3      while(true)
4      {
5          GetInputSensors();
6          ReceiveAllMessages();
7          DoLogics()
8          SendAllMessages();
9          SetOutputs();
10         sleep(100);
11     }
12 }
```

```
1  void PerformPeriodicTask()
2  {
3      while(true)
4      {
5          GetInputSensors();
6          ReceiveAllMessages();
7          DoLogics()
8          SendAllMessages();
9          SetOutputs();
10         sleep(100);
11     }
12 }
```

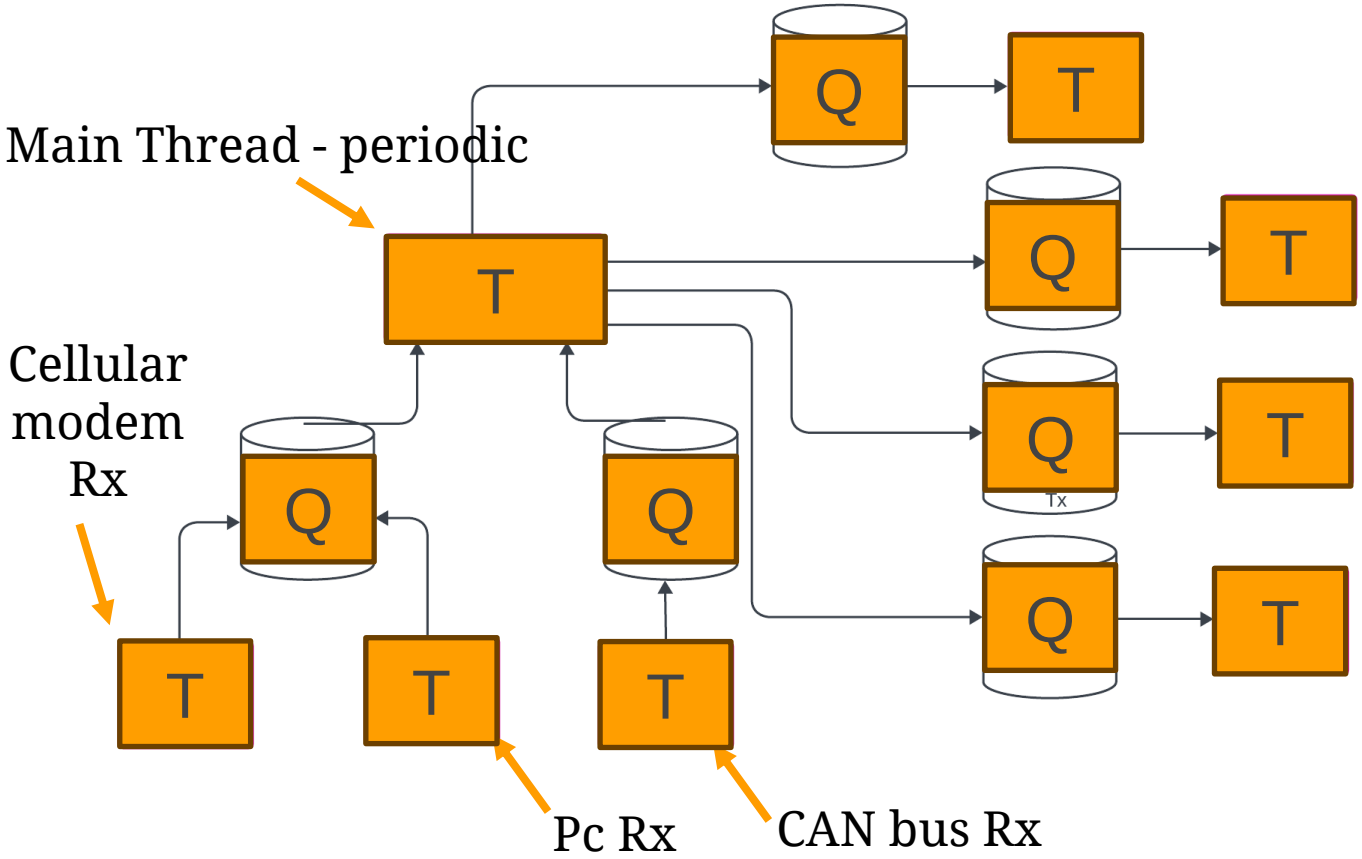
```
1  void PerformPeriodicTask()
2  {
3      while(true)
4      {
5          GetInputSensors();
6          ReceiveAllMessages();
7          DoLogics()
8          SendAllMessages();
9          SetOutputs();
10         sleep(100);
11     }
12 }
```

```
1  void PerformPeriodicTask()
2  {
3      while(true)
4      {
5          GetInputSensors();
6          ReceiveAllMessages();
7          DoLogics()
8          SendAllMessages();
9          SetOutputs();
10         sleep(100);
11     }
12 }
```

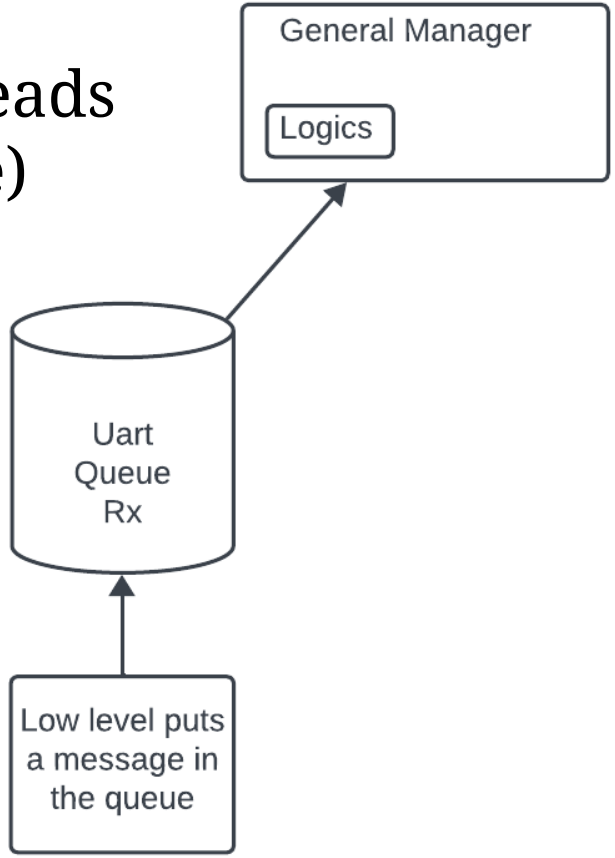
```
1  void PerformPeriodicTask()
2  {
3      while(true)
4      {
5          GetInputSensors();
6          ReceiveAllMessages();
7          DoLogics()
8          SendAllMessages();
9          SetOutputs();
10         sleep(100);
11     }
12 }
```

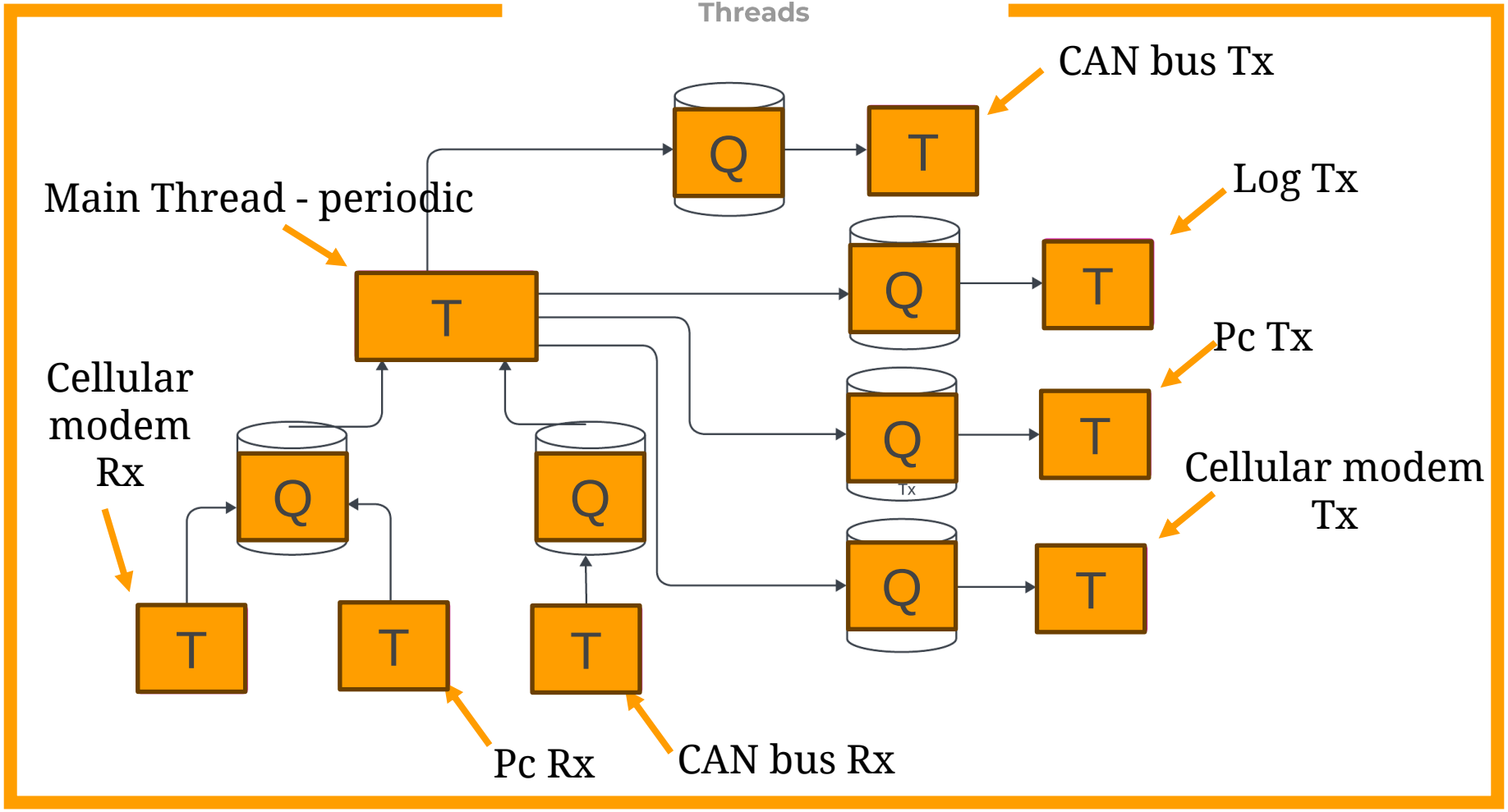
```
1  void PerformPeriodicTask()
2  {
3      while(true)
4      {
5          GetInputSensors();
6          ReceiveAllMessages();
7          DoLogics()
8          SendAllMessages();
9          SetOutputs();
10         sleep(100);
11     }
12 }
```

```
1  void PerformPeriodicTask()
2  {
3      while(true)
4      {
5          GetInputSensors();
6          ReceiveAllMessages();
7          DoLogics()
8          SendAllMessages();
9          SetOutputs();
10         sleep(100);
11     }
12 }
```

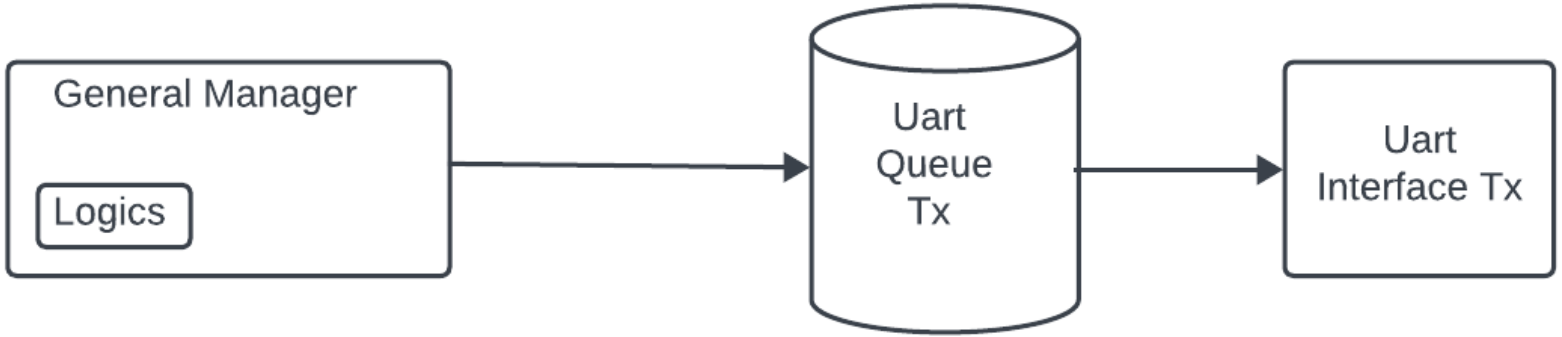

Main Thread - periodic
Cellular modem Rx
Pc Rx
CAN bus Rx
Q
T
Tx

Sync between threads (Input to queue)




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

Sync between threads (Queue to output)



“

Keep the number of threads to the bare minimum

Agenda

- Operating Systems
- Threads
- Layer Separation
- Network Problems
- External Interfaces
- Simulators
- Logs
- Monitoring

Separate the logic layer from the hardware layer

Embedded Software

Separate the logic layer from the hardware layer

Application Layer

Drivers handling Layer

Separate the logic layer from the hardware layer

Processes & Logics

Application Layer

Drivers handling Layer

Separate the logic layer from the hardware layer

Processes & Logics → Application Layer

Hardware Handling → Drivers handling Layer

```
1 void SetTrafficLight(int32_t waitingPeople, int32_t secFromGreen)
2 {
3     if(waitingPeople > 0 && secFromGreen > 50)
4     {
5         hwInterface->LedOn(TrafficColors::Green);
6     }
7     else
8     {
9         hwInterface->LedOn(TrafficColors::Red);
10    }
11 }
```

How are we going to test it?

```
1 void SetTrafficLight(int32_t waitingPeople, int32_t secFromGreen)
2 {
3     if(waitingPeople > 0 && secFromGreen > 50)
4     {
5         hwInterface->LedOn(TrafficColors::Green);
6     }
7     else
8     {
9         hwInterface->LedOn(TrafficColors::Red);
10    }
11 }
```

Probably, we're not...

```
1 void SetTrafficLight(int32_t waitingPeople, int32_t secFromGreen)
2 {
3     if(waitingPeople > 0 && secFromGreen > 50)
4     {
5         hwInterface->LedOn(TrafficColors::Green);
6     }
7     else
8     {
9         hwInterface->LedOn(TrafficColors::Red);
10    }
11 }
```

GetNextTrafficLight

```
1  TrafficColors GetNextTrafficLight(int32_t waitingPeople, int32_t secFromGreen)
2  {
3      if(waitingPeople > 0 && secFromGreen > 50)
4      {
5          return(TrafficColors::Green);
6      }
7      else
8      {
9          return(TrafficColors::Red);
10     }
11 }
```

Logic

SetTrafficLight

Logic

```
1 void SetTrafficLight(int32_t waitingPeople, int32_t secFromGreen)
2 {
3     TrafficColors color = GetNextTrafficLight(waitingPeople, secFromGreen);
4     hwInterface->LedOn(color);
5 }
6
```

SetTrafficLight

Logic

```
1 void SetTrafficLight(int32_t waitingPeople, int32_t secFromGreen)
2 {
3     TrafficColors color = GetNextTrafficLight(waitingPeople, secFromGreen);
4     hwInterface->LedOn(color);
5 }
6
```

Hardware

A unit test

```
1 #include "gtest/gtest.h"
2
3 TEST(TrafficLightLogic,GetNextTrafficLightTest)
4 {
5     Logic manager;
6     int32_t waitingPeople = 10;
7     int32_t secFromGreen = 50;
8     TrafficColors color = manager.GetNextTrafficLight(waitingPeople, secFromGreen);
9     EXPECT_EQ(color, TrafficColors::Red);
10 }
```

```
1 #include "gtest/gtest.h"
2
3 TEST(TrafficLightLogic,GetNextTrafficLightTest)
4 {
5     Logic manager;
6     int32_t waitingPeople = 10;
7     int32_t secFromGreen = 50;
8     TrafficColors color = manager.GetNextTrafficLight(waitingPeople, secFromGreen);
9     EXPECT_EQ(color, TrafficColors::Red);
10 }
```

Unit tests encourage us to keep the code simpler

 (Try to mock as little as possible)

```
1 class UartInterface
2 {
3     public:
4         bool Init(int32 baudrate);
5         bool Write(char* buffer, size_t size);
6         size_t Read(char* buffer, size_t maxSize);
7 };
```

```
1 class UartInterface
2 {
3     public:
4         bool Init(int32 baudrate);
5         bool Write(char* buffer, size_t size);
6         size_t Read(char* buffer, size_t maxSize);
7 };
```

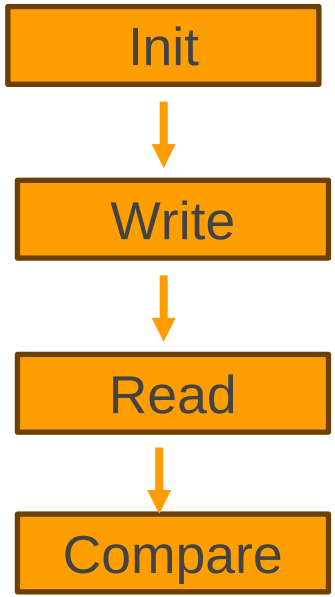
```
1 class UartInterface
2 {
3     public:
4         bool Init(int32 baudrate);
5         bool Write(char* buffer, size_t size);
6         size_t Read(char* buffer, size_t maxSize);
7 };
```

Layer Separation

```
1 class UartInterface
2 {
3     public:
4         bool Init(int32 baudrate);
5         bool Write(char* buffer, size_t size);
6         size_t Read(char* buffer, size_t maxSize);
7 };
```

```
1 class SharedMemoryInterface
2 {
3     public:
4         bool Init();
5         bool Write(char* buffer, size_t size);
6         size_t Read(char* buffer, size_t maxSize);
7 };
```

How Hardware tests should look like



Layer Separation

```
1  bool SharedMemoryTest()
2  {
3      SharedMemoryInterface sharedMem;
4      char writeBuffer[100] = {};
5      char readBuffer[100] = {};
6      size_t length = 0;
7      //prepare data to send
8      for(int i=0; i<100;i++)
9      {
10         writeBuffer[i] = i;
11     }
12     sharedMem.Init();
13     sharedMem.Write(writeBuffer,100);
14     length = sharedMem.Read(readBuffer,100);
15     //check
16     if(length!=100)
17     {
18         return false;
19     }
20     //compare
21     if( memcmp(writeBuffer,readBuffer,length)!= 0)
22     {
23         return false;
24     }
25     else
26     {
27         return true;
28     }
29 }
```

Prepare data

```
8   //prepare data to send
9   for(int i=0; i<100;i++)
10  {
11      writeBuffer[i] = i;
12  }
13  //Init
14  sharedMem.Init();
15
16  //write
17  sharedMem.Write(writeBuffer,100);
18
19  //read
20  length = sharedMem.Read(readBuffer,100);
```

Init

```
//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);
```

```
8   //prepare data to send
9   for(int i=0; i<100;i++)
10  {
11      writeBuffer[i] = i;
12  }
13  //Init
14  sharedMem.Init();
15
16  //write
17  sharedMem.Write(writeBuffer,100);
18
19  //read
20  length = sharedMem.Read(readBuffer,100);
```

Write → 17

```
//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

Compare →

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

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

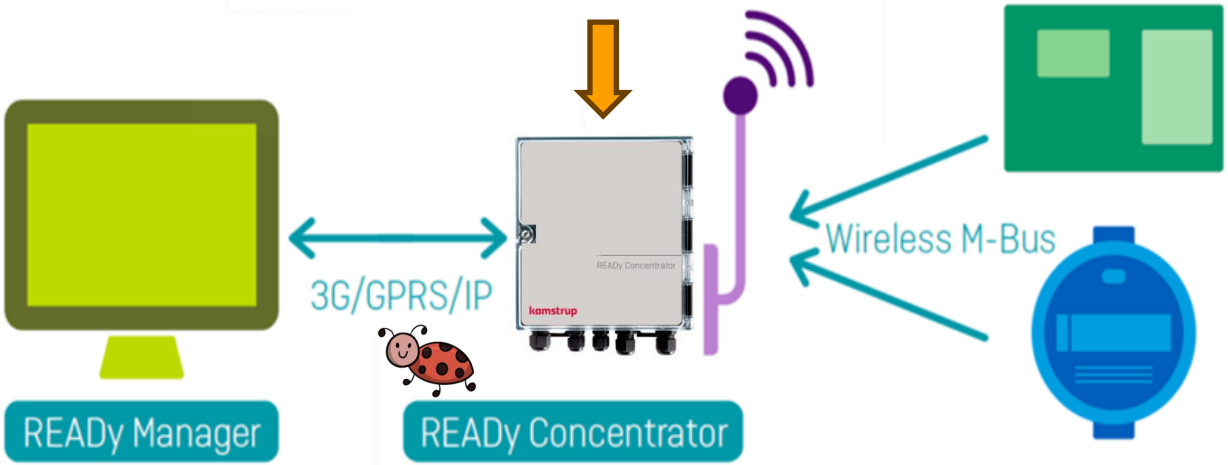
“

Separate the logic layer from the hardware layer

Agenda

- Operating Systems
- Threads
- Layer Separation
- Network Problems
- External Interfaces
- Simulators
- Logs
- Monitoring

Network Problems

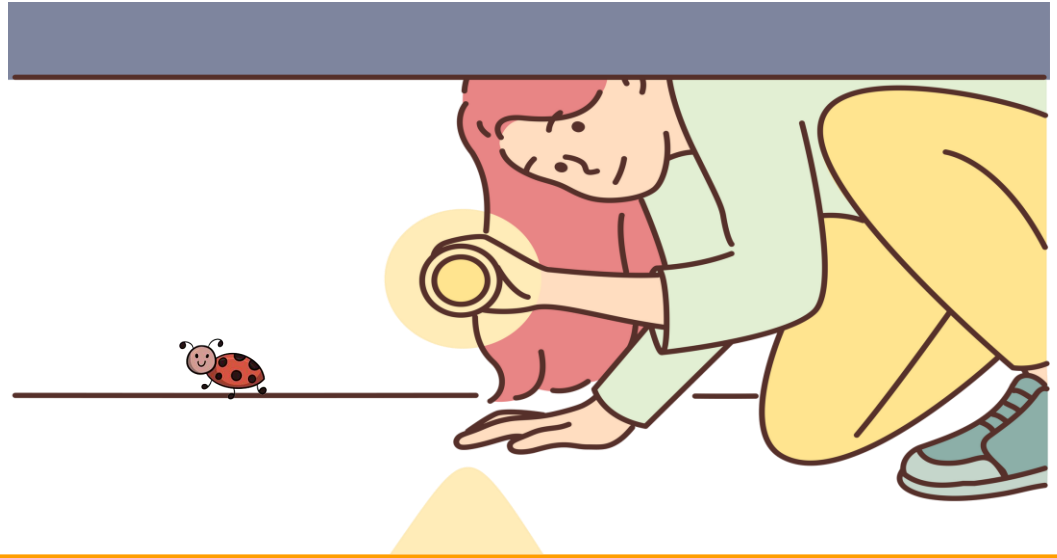


Network Problems

- Loss of data in big sites (>5000 meters)
- Loss of data when there were network errors
- None of the errors were seen in the lab.

Worked fine most of the time

Sometimes data was lost



The data prepared for transmission remained in the RAM, awaiting to be sent

The data prepared for transmission remained in the RAM, awaiting to be sent

So what is the problem with that?

The data prepared for transmission remained in the RAM, awaiting to be sent

In case of no communication:

- Start to aggregate - takes a lot of space
- Loss of data in case of reset

The solution?

Disconnect the Logic from the Network

Disconnect the Logic from the Network

1

Thread #1 → Logic

Thread #2 → Sending

Disconnect the Logic from the Network

1

Thread #1 → Logic

Thread #2 → Sending

2

Using SQLite (light DB) in the SD card

What was achieved by this implementation:

What was achieved by this implementation:

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

What was achieved by this implementation:

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

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

“

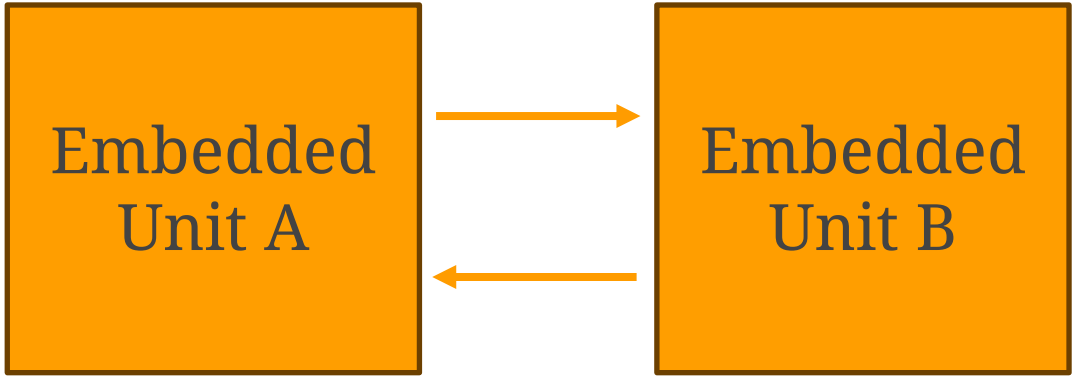
Disconnect the logic from the network

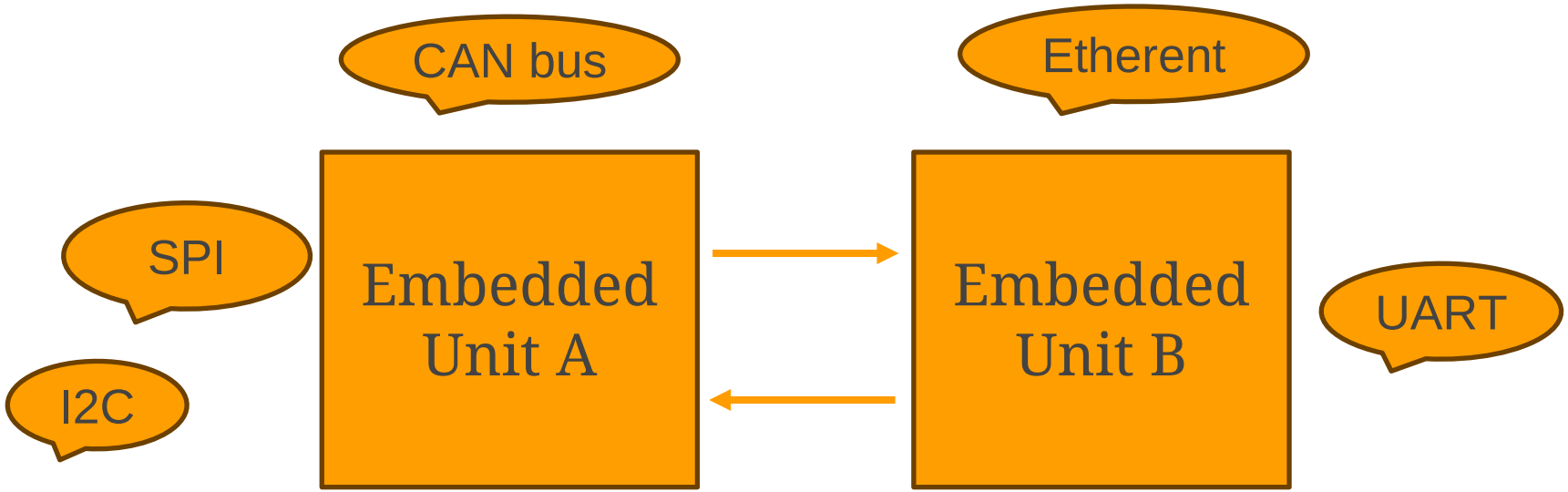
Agenda

- Operating Systems
- Threads
- Layer Separation
- Network Problems
- External Interfaces
- Simulators
- Logs
- Monitoring

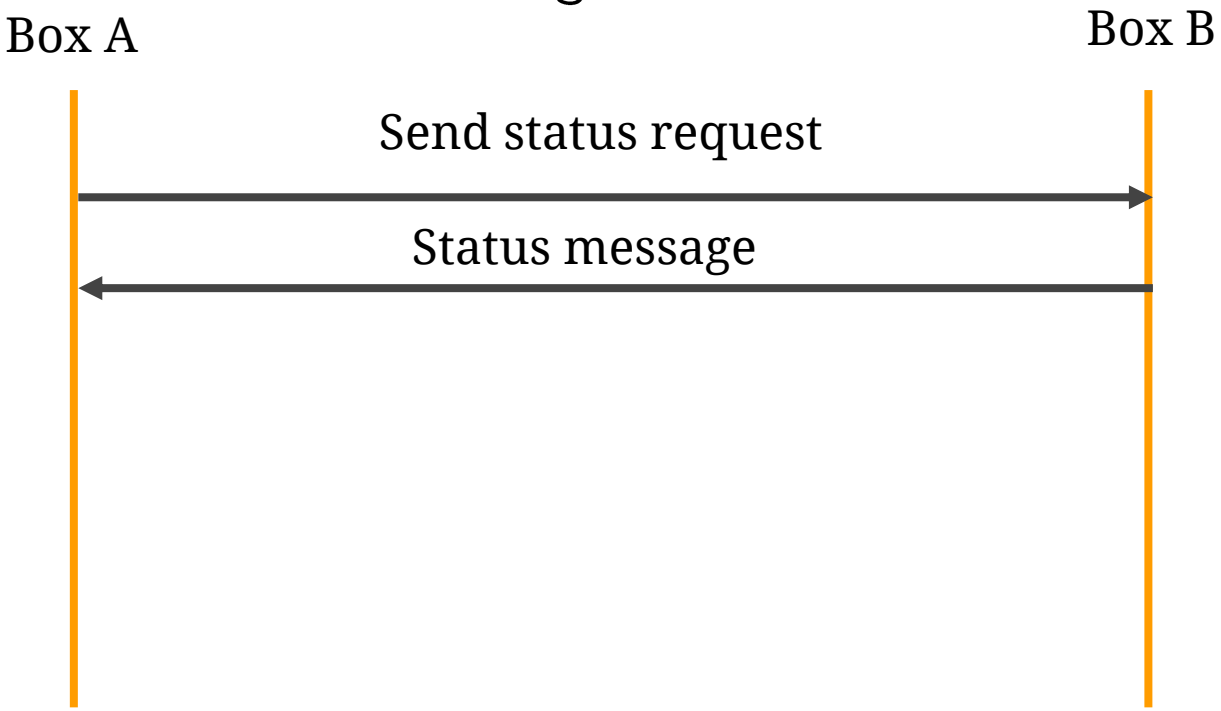
Design your protocol and messages
in a way you could always
bounce back
from a “bad” message

Embedded Unit A

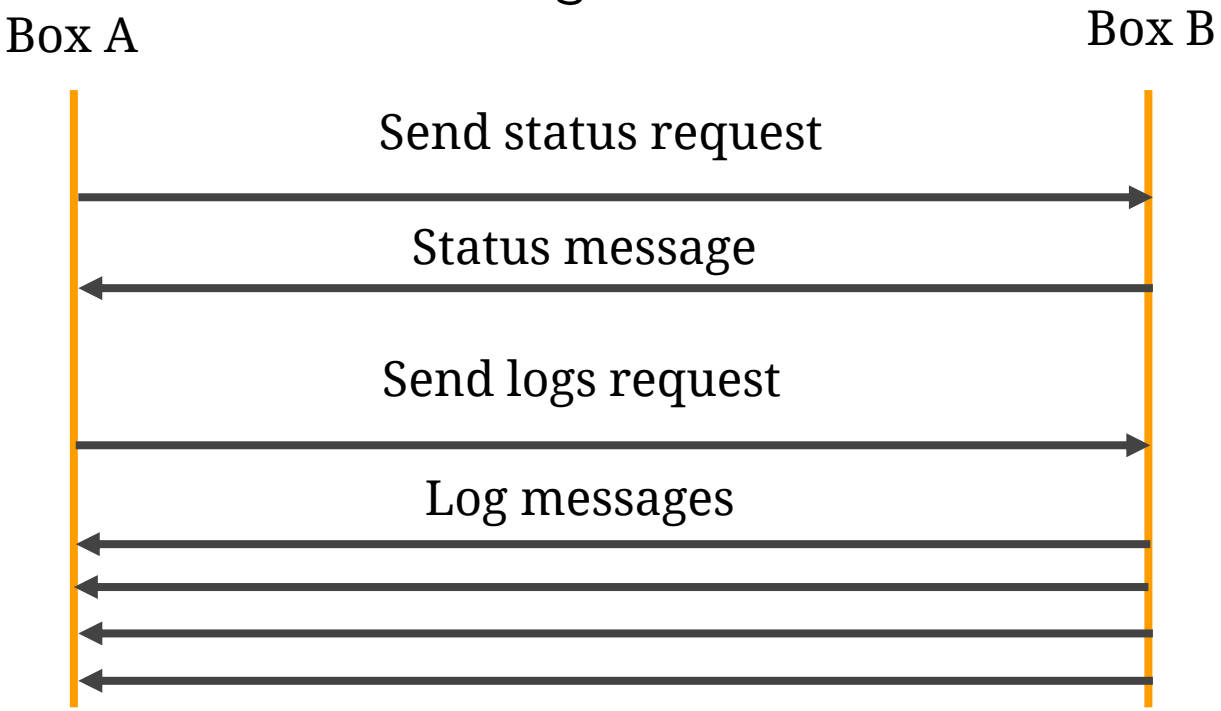

Embedded Unit A
Embedded Unit B


CAN bus
Etherent
SPI
Embedded Unit A
Embedded Unit B
UART
I2C

Messages Flow



Messages Flow

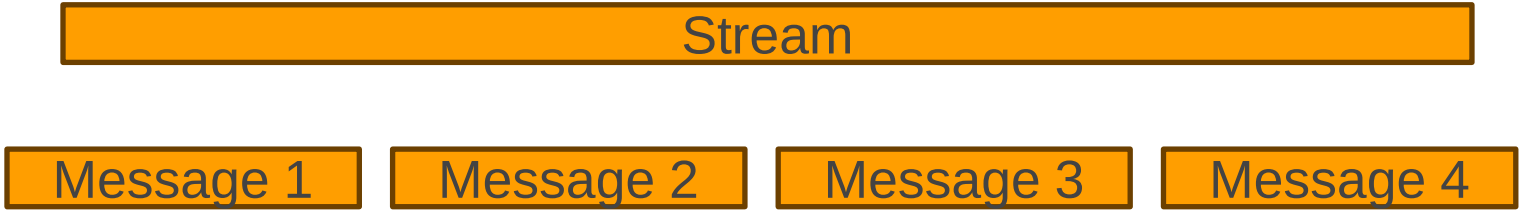


Depending on your low level either a stream or packets

Depending on your low level either a stream or packets

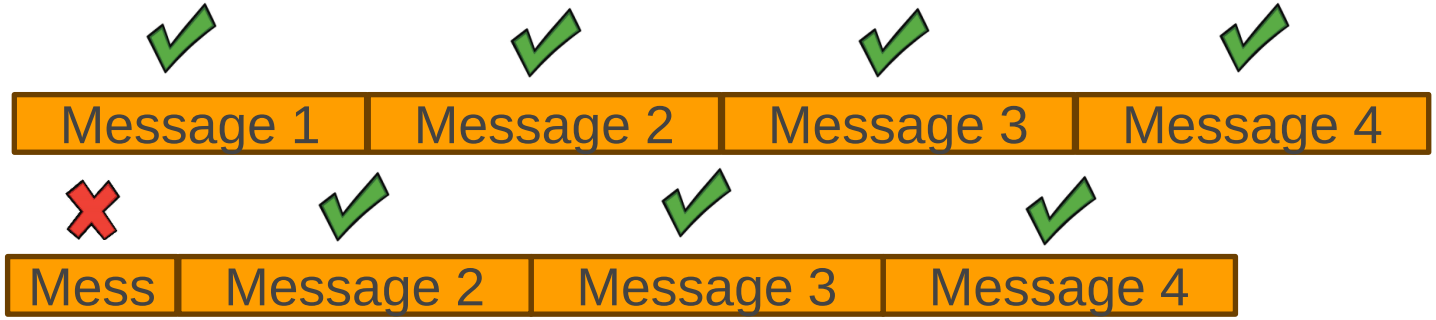
Stream

Depending on your low level either a stream or packets

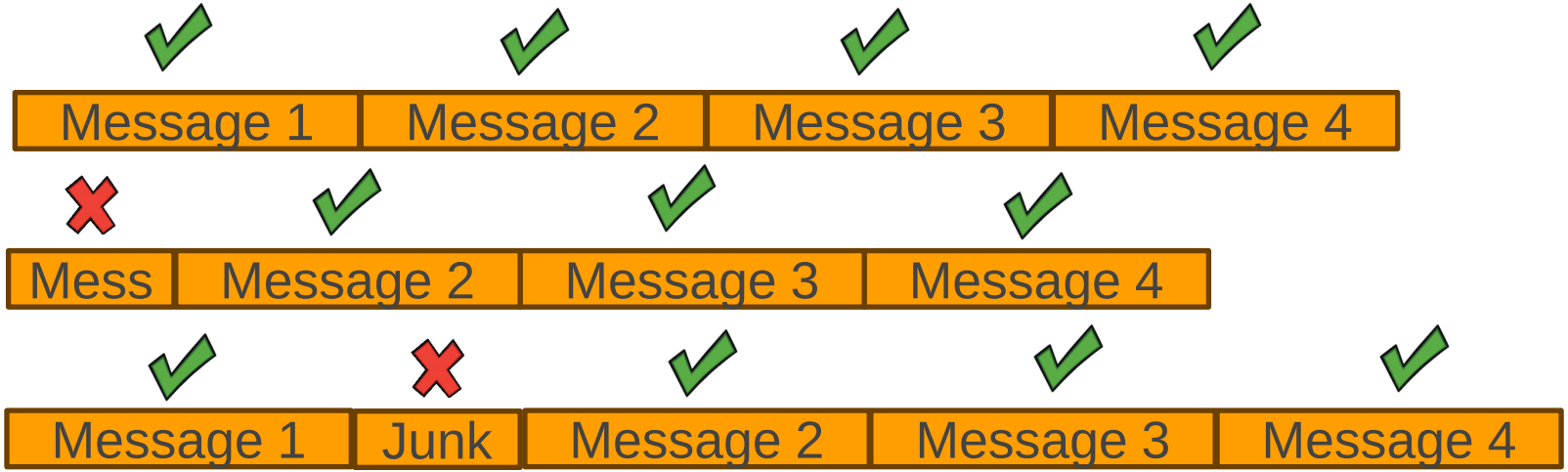


Potential problems with streaming

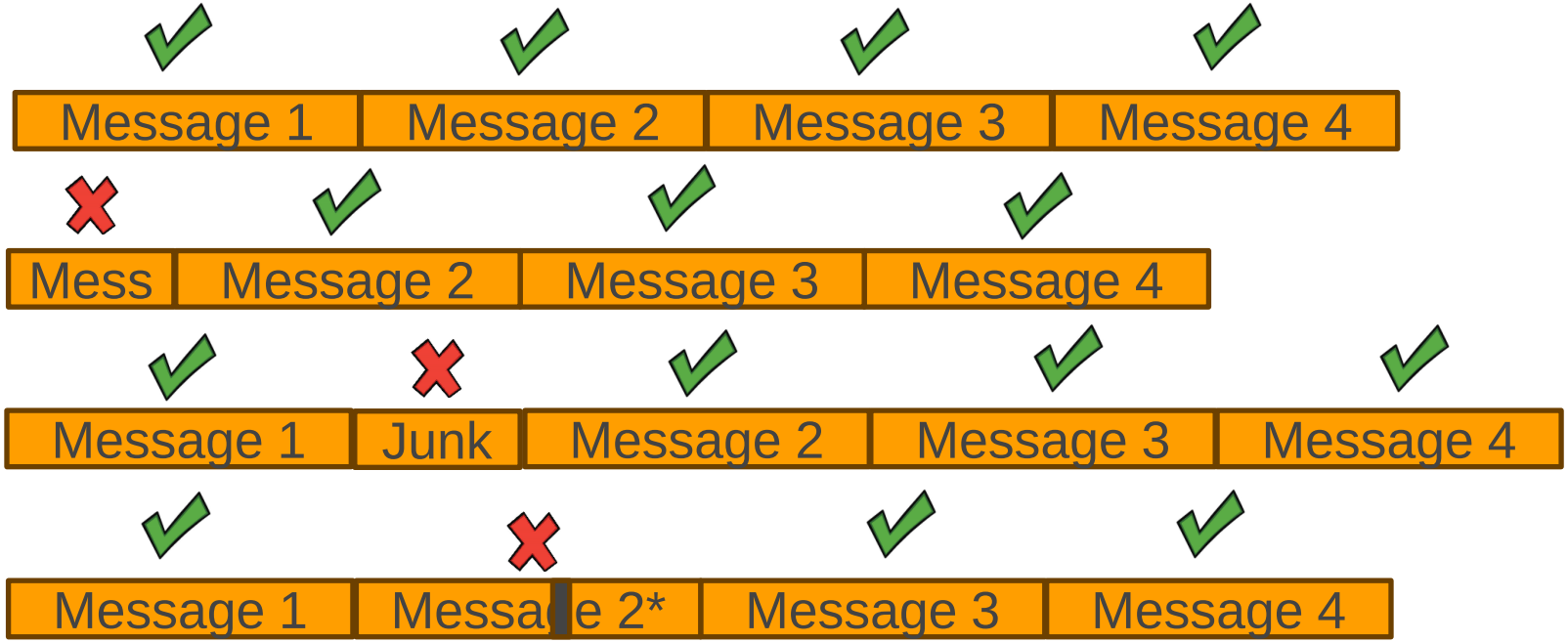
Potential problems with streaming



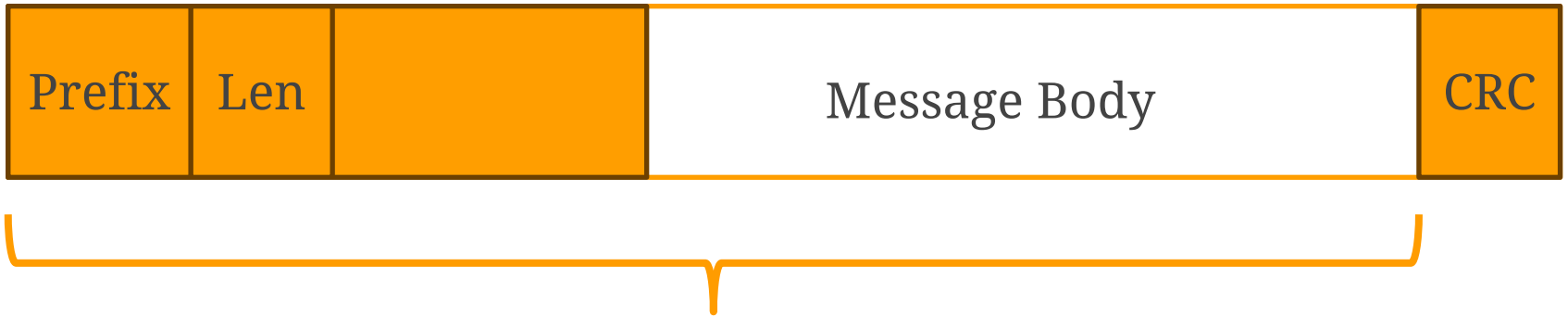
Potential problems with streaming



Potential problems with streaming



Recommended Message Structure (Streaming)



Verify this content is correct

```
1 struct Header
2 {
3     char prefix[4]; //M#$!
4     uint16_t length;
5     uint16_t version;
6     uint16_t type; //opcode
7     uint16_t id; //unique id
8     uint16_t sequence; //running counter
9 };
```

Easy to identify in memory

```
1 struct Header
2 {
3     char prefix[4]; //M#$!
4     uint16_t length;
5     uint16_t version;
6     uint16_t type; //opcode
7     uint16_t id; //unique id
8     uint16_t sequence; //running counter
9 };
```

```
1 struct Header
2 {
3     char prefix[4]; //M#$!
4     uint16_t length;
5     uint16_t version;
6     uint16_t type; //opcode
7     uint16_t id; //unique id
8     uint16_t sequence; //running counter
9 };
```

```
1 struct Header
2 {
3     char prefix[4]; //M#$!
4     uint16_t length;
5     uint16_t version;
6     uint16_t type; //opcode
7     uint16_t id; //unique id
8     uint16_t sequence; //running counter
9 };
```

```
1 struct Header
2 {
3     char prefix[4]; //M#$!
4     uint16_t length;
5     uint16_t version;
6     uint16_t type; //opcode
7     uint16_t id; //unique id
8     uint16_t sequence; //running counter
9 };
```

```
1 struct Header
2 {
3     char prefix[4]; //M#$!
4     uint16_t length;
5     uint16_t version;
6     uint16_t type; //opcode
7     uint16_t id; //unique id
8     uint16_t sequence; //running counter
9 };
```

```
1 struct Header
2 {
3     char prefix[4]; //M#$!
4     uint16_t length;
5     uint16_t version;
6     uint16_t type; //opcode
7     uint16_t id; //unique id
8     uint16_t sequence; //running counter
9 };
```

```
1 struct MyMessage
2 {
3     Header header;
4     uint32_t temp1;
5     uint32_t temp2;
6     uint32_t temp3;
7     uint16_t crc;
8 };
```

```
1 struct MyMessage
2 {
3     Header header;
4     uint32_t temp1;
5     uint32_t temp2;
6     uint32_t temp3;
7     uint16_t crc;
8 };
```

Body structure

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

```
1 struct MyMessage
2 {
3     Header header;
4     uint32_t temp1;
5     uint32_t temp2;
6     uint32_t temp3;
7     uint16_t crc;
8 };
```

“

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

Agenda

- Operating Systems
- Threads
- Layer Separation
- Network Problems
- External Interfaces
- Simulators
- Logs
- Monitoring

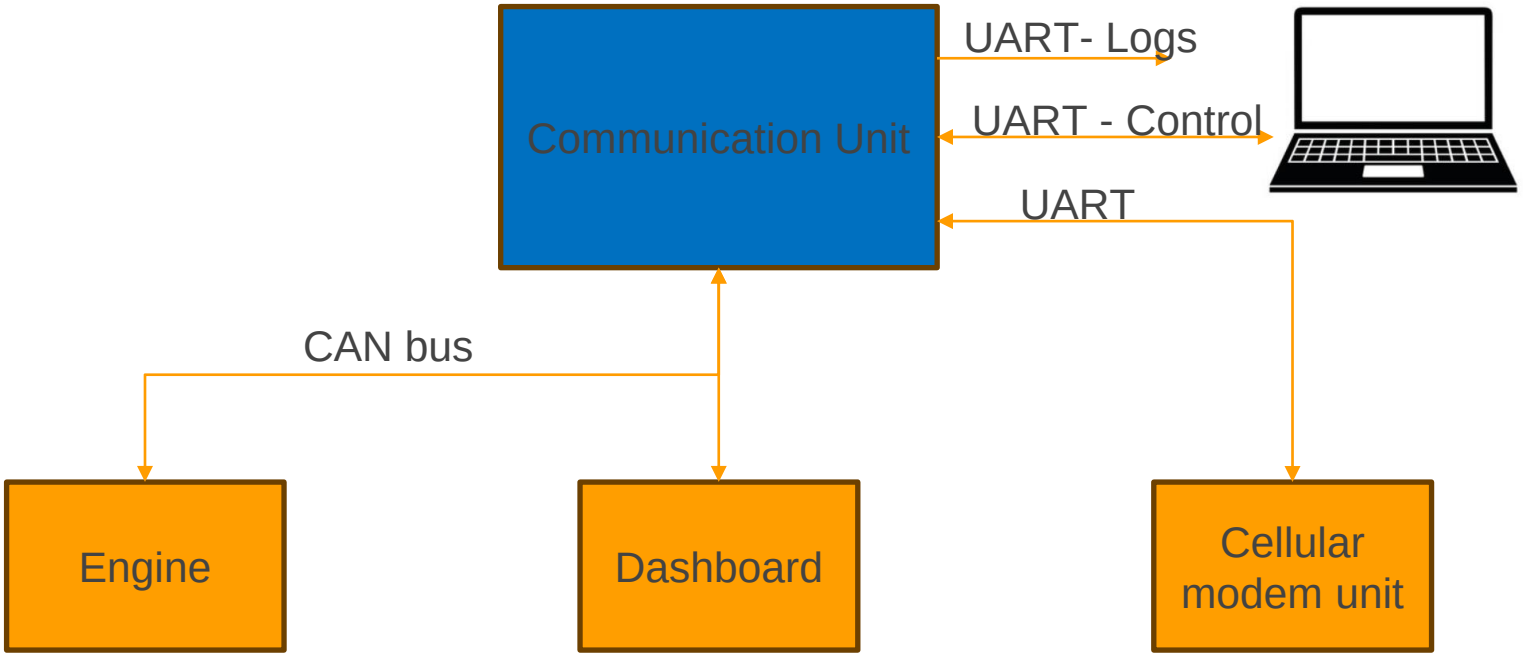

BCN LAND TAXI NAV STROBE
FUEL PUMP
PITOT HEAT
RELEASE
PARKING BRAKE

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

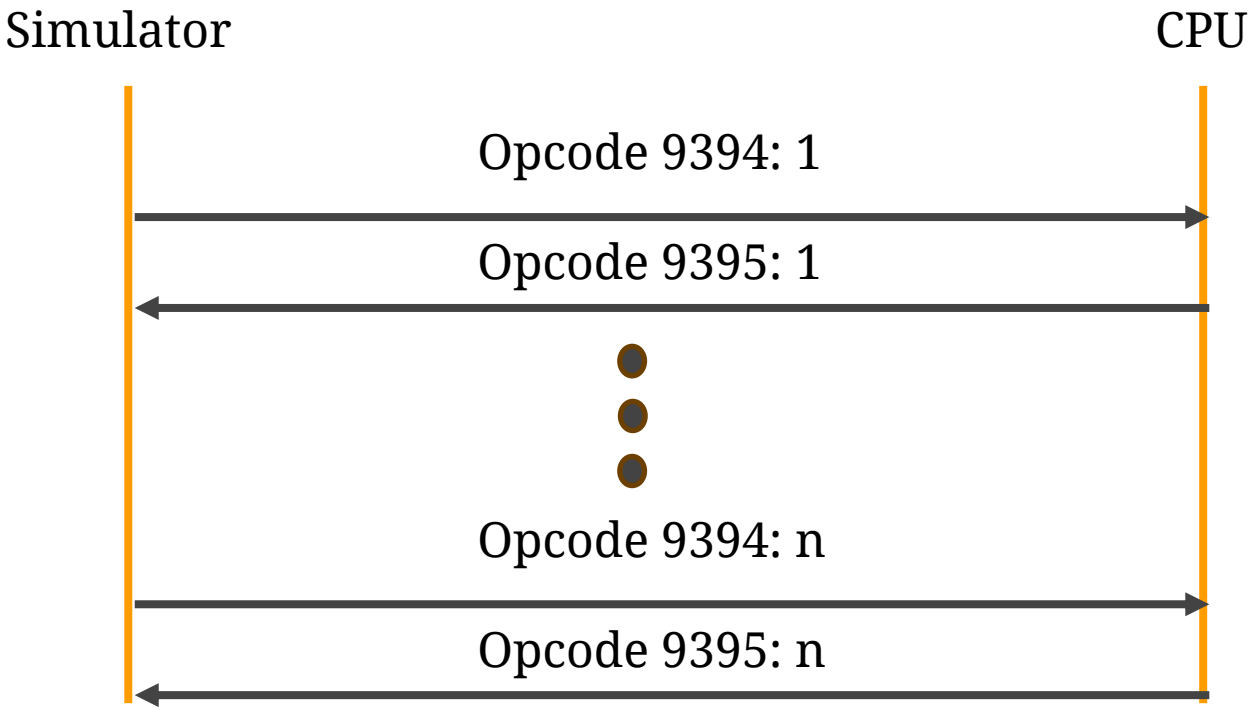
The simulator allows us to replicate scenarios that would be difficult to test in real life

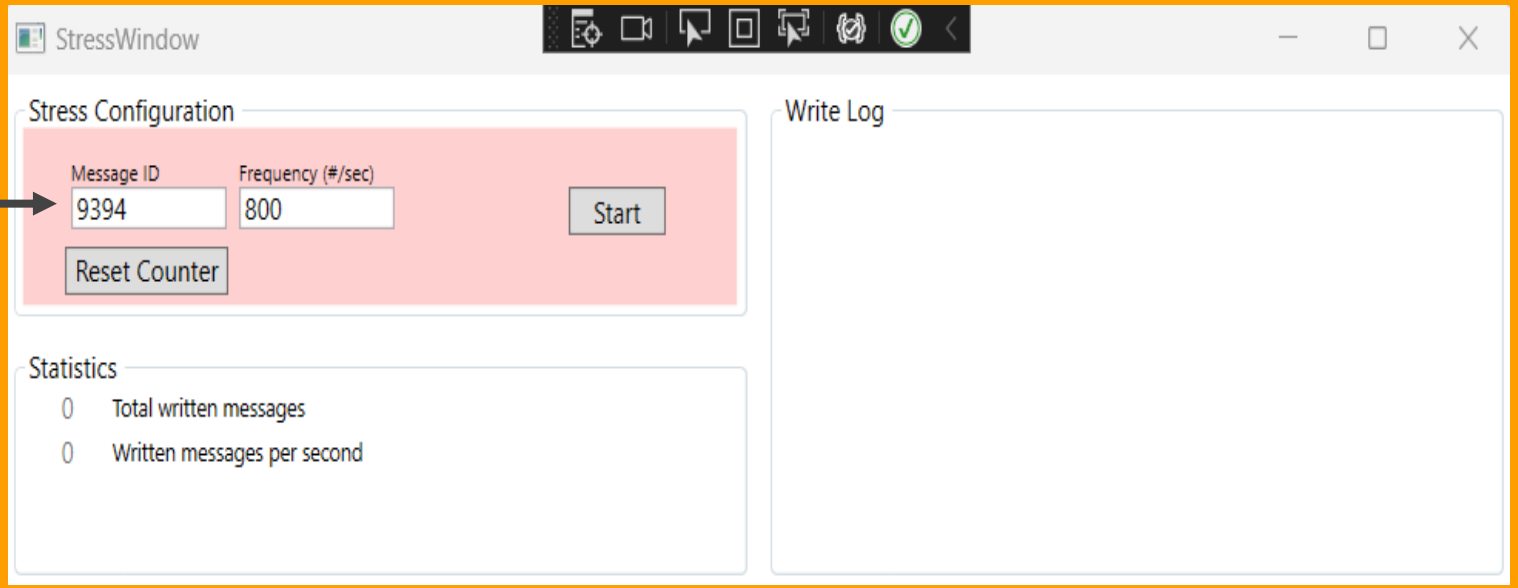
- Interfaces simulator
- Load simulator

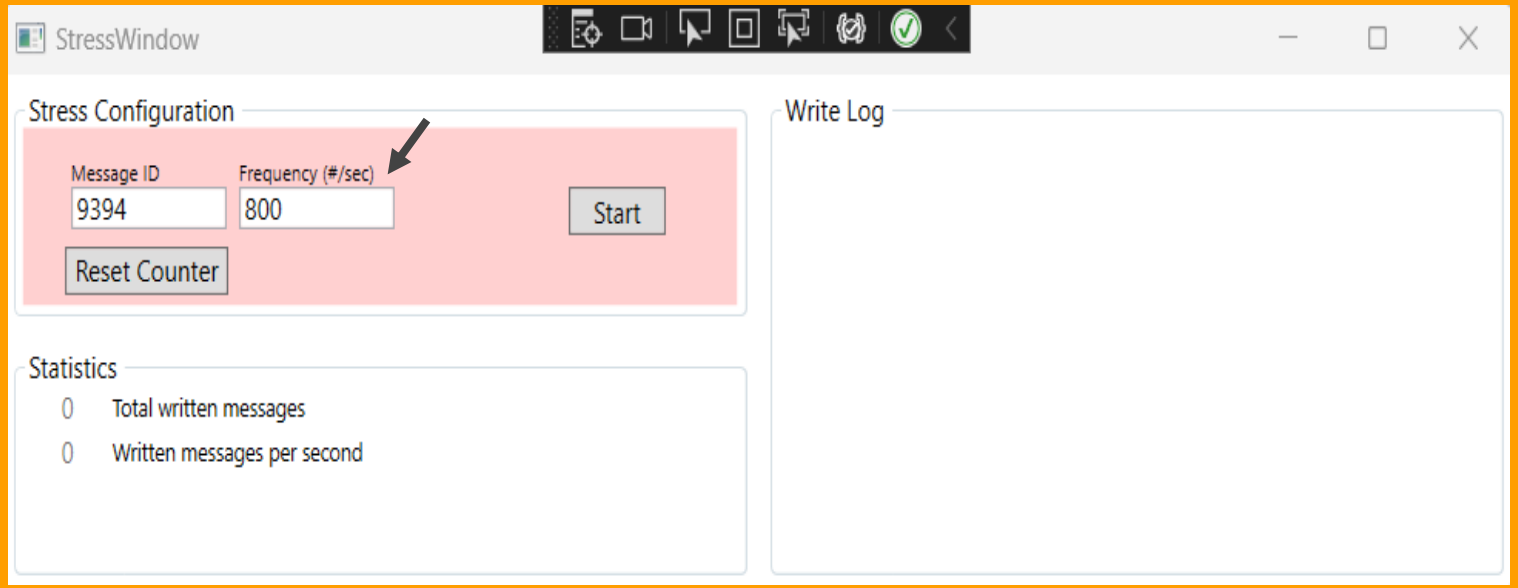
Simulator #1

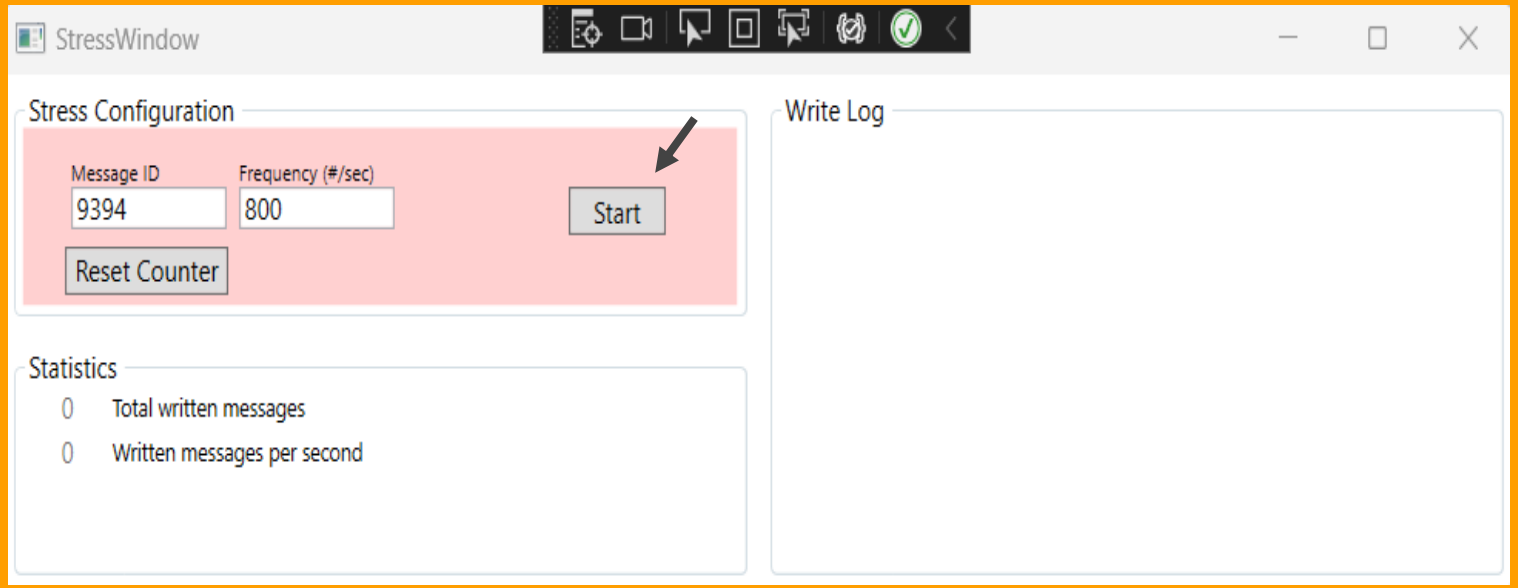

Communication Unit
UART- Logs
UART - Control
UART
CAN bus
Engine
Dashboard
Cellular modem unit

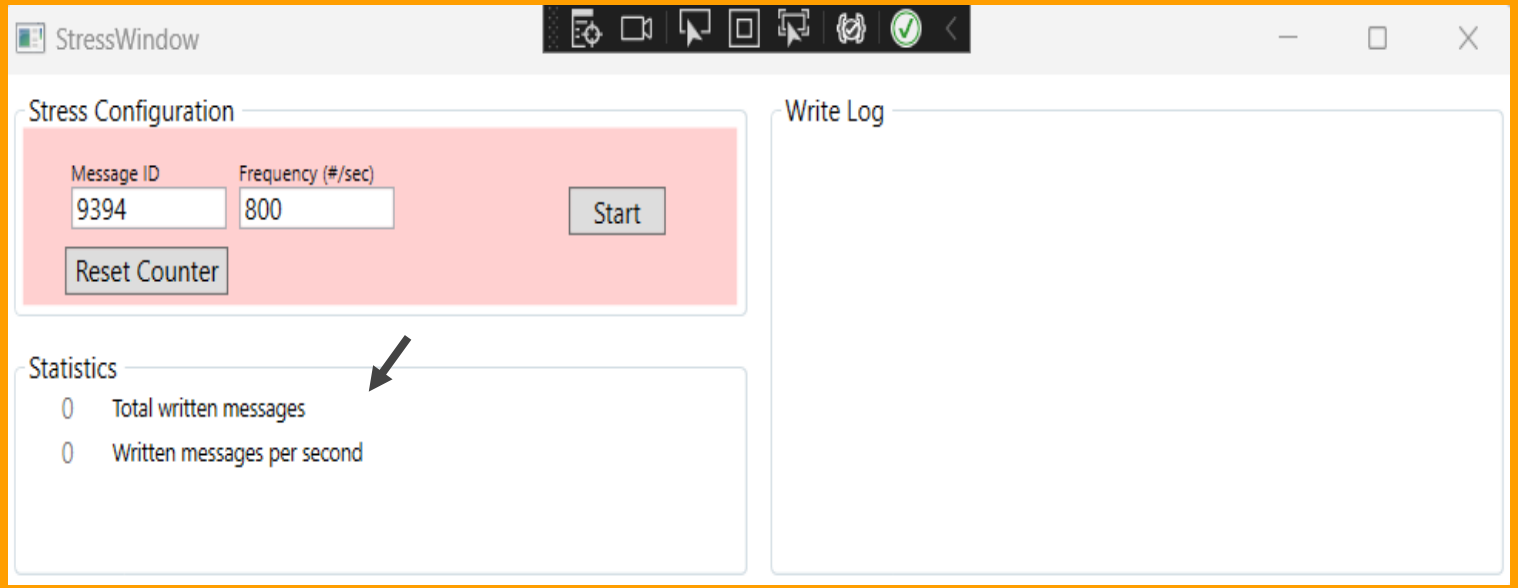
Q:How to verify our CPU supports 200 messages per second?

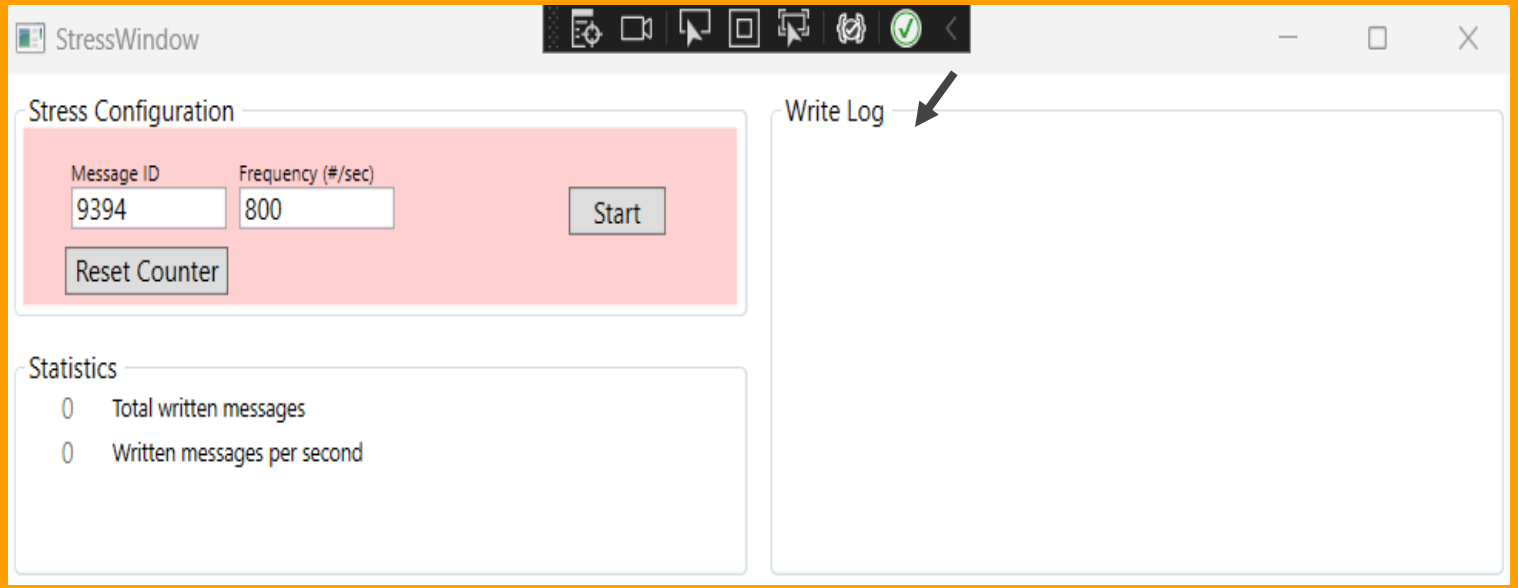

Simulator
CPU
Opcode 9394: 1
Opcode 9395: 1
Opcode 9394: n
Opcode 9395: n


StressWindow
Stress Configuration
Message ID
9394
Frequency (#/sec)
800
Start
Reset Counter
Statistics
0 Total written messages
0 Written messages per second
Write Log


StressWindow
Stress Configuration
Message ID
9394
Frequency (#/sec)
800
Start
Reset Counter
Statistics
0 Total written messages
0 Written messages per second
Write Log


StressWindow
Stress Configuration
Message ID
9394
Frequency (#/sec)
800
Start
Reset Counter
Statistics
0 Total written messages
0 Written messages per second
Write Log


StressWindow
Stress Configuration
Message ID
9394
Frequency (#/sec)
800
Start
Reset Counter
Statistics
0 Total written messages
0 Written messages per second
Write Log


StressWindow
Stress Configuration
Message ID
9394
Frequency (#/sec)
800
Start
Reset Counter
Statistics
0 Total written messages
0 Written messages per second
Write Log

- 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

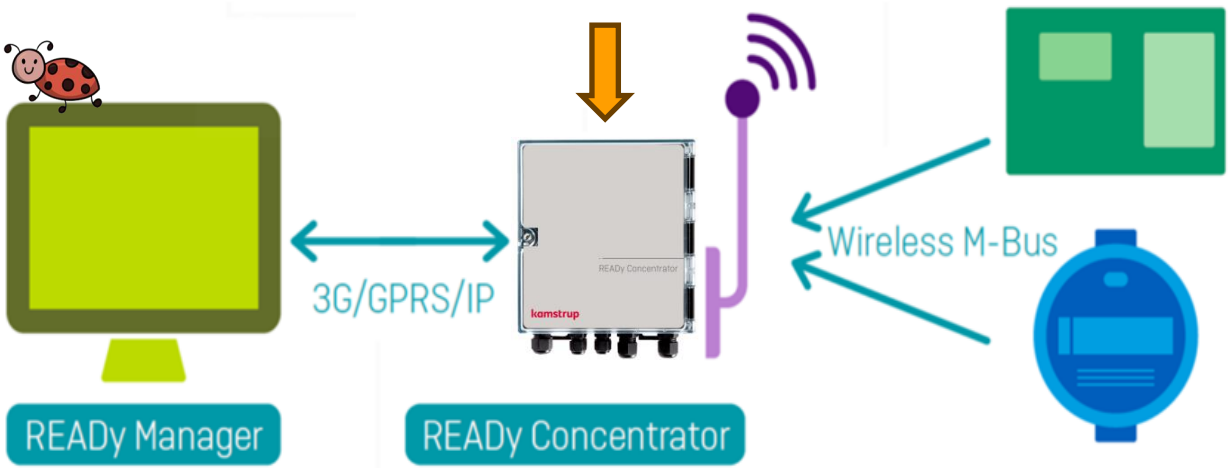
- The simulator sends X messages per second and verifies that it receives all of them back
- At 850 messages per second, it stopped working
- The PC didn’t have enough resources
- The embedded CPU passed the test

- 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

Simulator #2

Loss of data in big sites (>5000 meters)



Q: How to simulate 5K+ meters?

- 1 data frame meter to 10-50 fake data frames
- The unit crashed on my table when I simulated 6k meters and put “hard configuration”

- 1 data frame meter to 10-50 fake data frames
- The unit crashed on my table when I simulated 6k meters and put “hard configuration”

```
1 struct Message
2 {
3     long opcode;
4     long id;
5     long value;
6 };
```

Simulators

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

```
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);
                }
            }
        }
    }
}
```

```
1  void ReceiveThread(queue<Message>& queue)
2  {
3      Message message = {};
4      const bool simulatorEnabled = std::filesystem::is_regular_file("myfile.txt");
5      while (true)
6      {
7          message = {};
8          bool isReceived = ReceiveMessage(message);
9          if(isReceived)
10         {
11             queue.push(message);
12             if(simulatorEnabled)
13             {
```

```
1  void ReceiveThread(queue<Message>& queue)
2  {
3      Message message = {};
4      const bool simulatorEnabled = std::filesystem::is_regular_file("myfile.txt");
5      while (true)
6      {
7          message = {};
8          bool isReceived = ReceiveMessage(message);
9          if(isReceived)
10         {
11             queue.push(message);
12             if(simulatorEnabled)
13             {
```

```
1  void ReceiveThread(queue<Message>& queue)
2  {
3      Message message = {};
4      const bool simulatorEnabled = std::filesystem::is_regular_file("myfile.txt");
5      while (true)
6      {
7          message = {};
8          bool isReceived = ReceiveMessage(message);
9          if(isReceived)
10         {
11             queue.push(message);
12             if(simulatorEnabled)
13             {
```

```
1  void ReceiveThread(queue<Message>& queue)
2  {
3      Message message = {};
4      const bool simulatorEnabled = std::filesystem::is_regular_file("myfile.txt");
5      while (true)
6      {
7          message = {};
8          bool isReceived = ReceiveMessage(message);
9          if(isReceived)
10         {
11             queue.push(message);
12             if(simulatorEnabled)
13             {
```

```
1  void ReceiveThread(queue<Message>& queue)
2  {
3      Message message = {};
4      const bool simulatorEnabled = std::filesystem::is_regular_file("myfile.txt");
5      while (true)
6      {
7          message = {};
8          bool isReceived = ReceiveMessage(message);
9          if(isReceived)
10         {
11             queue.push(message);
12             if(simulatorEnabled)
13             {
```

```
11              queue.push(message);
12              if(simulatorEnabled)
13              {
14                  Message simulatorMsg = {};
15                  for(int i=0; i<100; i++)
16                  {
17                      simulatorMsg = message;
18                      simulatorMsg.id =  message.id + rand() % 1000;
19                      queue.push(simulatorMsg);
20                  }
21              }
22          }
23      }
24  }
```

```
11                queue.push(message);
12                if(simulatorEnabled)
13                {
14                    Message simulatorMsg = {};
15                    for(int i=0; i<100; i++)
16                    {
17                        simulatorMsg = message;
18                        simulatorMsg.id =  message.id + rand() % 1000;
19                        queue.push(simulatorMsg);
20                    }
21                }
22            }
23        }
24    }
```

```
11                queue.push(message);
12                if(simulatorEnabled)
13                {
14                    Message simulatorMsg = {};
15                    for(int i=0; i<100; i++)
16                    {
17                        simulatorMsg = message;
18                        simulatorMsg.id =  message.id + rand() % 1000;
19                        queue.push(simulatorMsg);
20                    }
21                }
22            }
23        }
24    }
```

```
11              queue.push(message);
12              if(simulatorEnabled)
13              {
14                  Message simulatorMsg = {};
15                  for(int i=0; i<100; i++)
16                  {
17                      simulatorMsg = message;
18                      simulatorMsg.id =  message.id + rand() % 1000;
19                      queue.push(simulatorMsg);
20                  }
21              }
22          }
23      }
24  }
```

```
11              queue.push(message);
12              if(simulatorEnabled)
13              {
14                  Message simulatorMsg = {};
15                  for(int i=0; i<100; i++)
16                  {
17                      simulatorMsg = message;
18                      simulatorMsg.id =  message.id + rand() % 1000;
19                      queue.push(simulatorMsg);
20                  }
21              }
22          }
23      }
24  }
```

Simulators

```
1 struct Message
2 {
3     long opcode;
4     long id;
5     long value;
6 };
```

```
1 void ReceiveThread(queue<Message>& queue)
2 {
3     Message message = {};
4     const bool simulatorEnabled = std::filesystem::is_regular_file("myfile.txt");
5     while (true)
6     {
7         message = {};
8         bool isReceived = ReceiveMessage(message);
9         if(isReceived)
10        {
11            queue.push(message);
12            if(simulatorEnabled)
13            {
14                Message simulatorMsg = {};
15                for(int i=0; i<100; i++)
16                {
17                    simulatorMsg = message;
18                    simulatorMsg.id =  message.id + rand() % 1000;
19                    queue.push(simulatorMsg);
20                }
21            }
22        }
23    }
24 }
```

Significant benefits:

Significant benefits:

- No need for special hardware

Significant benefits:

- No need for special hardware
- Short development time

Significant benefits:

- No need for special hardware
- Short development time
- Easy access to simulator mode without additional building

Significant benefits:

- No need for special hardware
- Short development time
- Easy access to simulator mode without additional building
- Can be used as a release test before launching a new version

“

Use simulators

Agenda

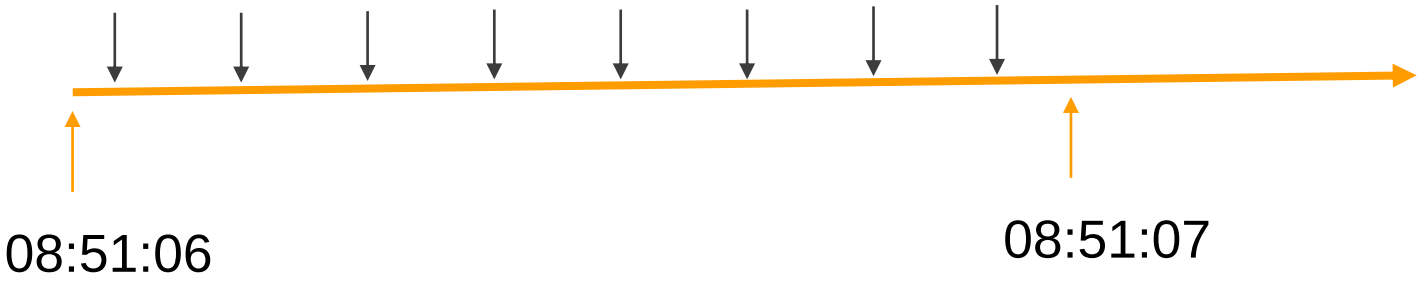
- Operating Systems
- Threads
- Layer Separation
- Network Problems
- External Interfaces
- Simulators
- Logs
- Monitoring

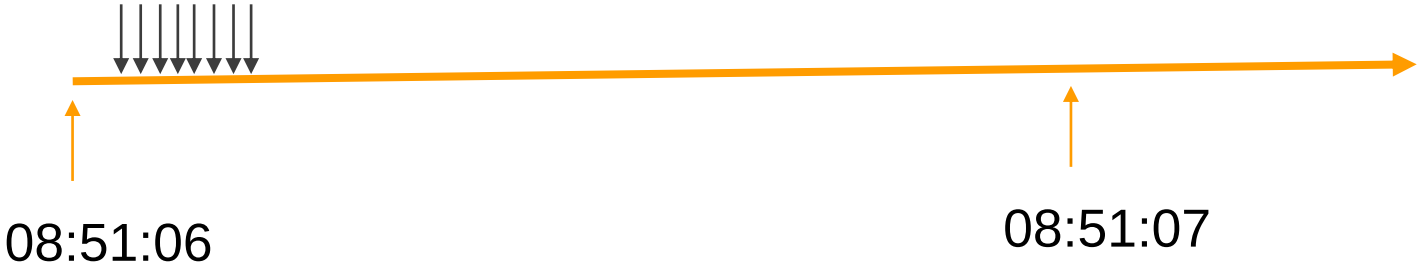
Logs

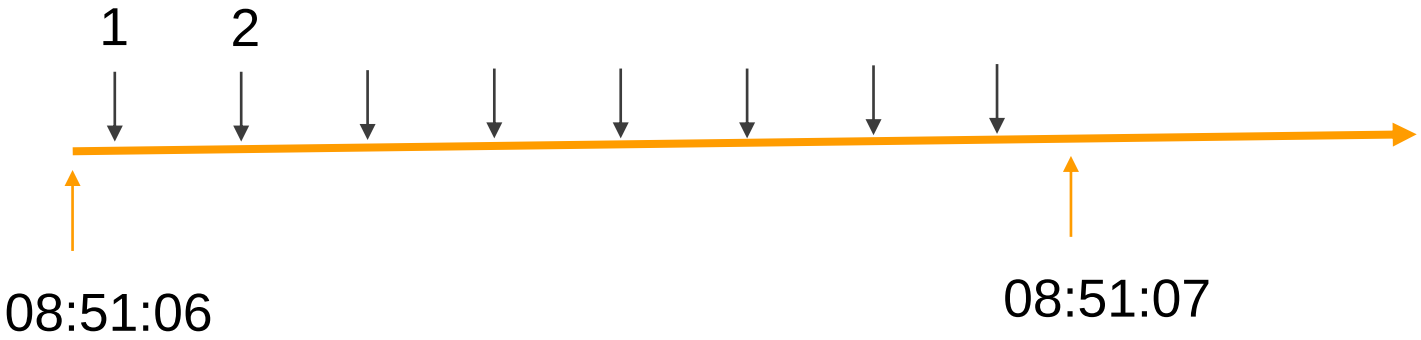
```
2023-09-16 10:00:00.123 INFO  main.cpp:100 [Thread-1]: System Startup
2023-09-16 10:01:15.045 WARNING  sensors.cpp:45 [Thread-2]: Temperature rising, check sensors
2023-09-16 10:02:30.321 ERROR  error_handler.cpp:78 [Thread-3]: Critical error - system halted
2023-09-16 10:03:45.678 INFO  main.cpp:105 [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
```

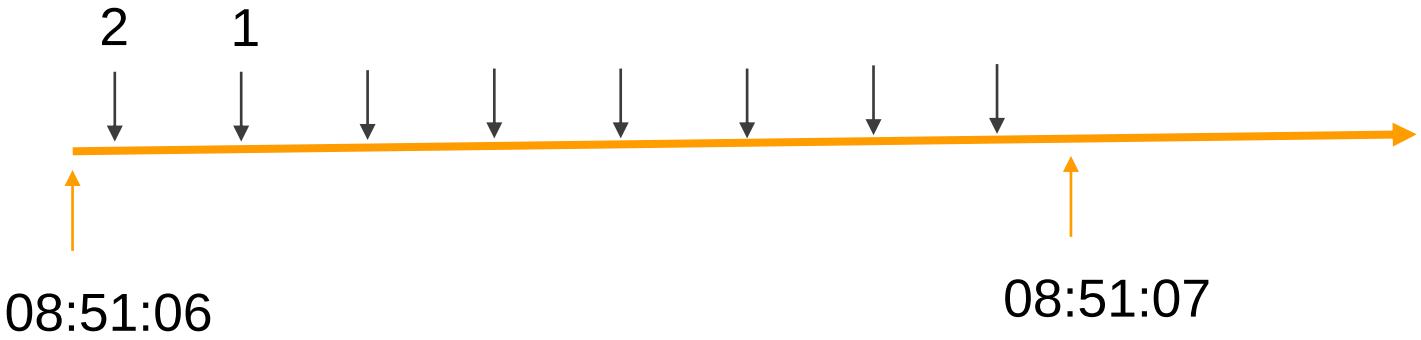
- ☐ Add timestamps with milliseconds
- ☐ Add metadata (log level, file, line and thread)
- ☐ Use the same logs configuration in all sites
- ☐ Keep number of logs to the bare minimum
- ☐ Write logs with details
- ☐ Prepare your logs to automatic monitoring

```
4 2023-09-16 10:03:45.678 INFO  main.cpp:105 [Thread-1]:
5 2023-09-16 10:05:00.256 INFO  main.cpp:110 [Thread-1]:
6 2023-09-16 10:06:15.789 INFO  user_login.cpp:55 [Threac
```


08:51:06
08:51:07


08:51:06
08:51:07


1
2
08:51:06
08:51:07


2
1
08:51:06
08:51:07


1 2
08:51:06
08:51:07

- ☐ 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

Log level

```
4 2023-09-16 10:03:45.678 INFO  main.cpp:105 [Thread-1]:
5 2023-09-16 10:05:00.256 INFO  main.cpp:110 [Thread-1]:
6 2023-09-16 10:06:15.789 INFO  user_login.cpp:55 [Threac
```

File:Line

```
4 2023-09-16 10:03:45.678 INFO  main.cpp:105 [Thread-1]:
5 2023-09-16 10:05:00.256 INFO  main.cpp:110 [Thread-1]:
6 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
```

- ☐ 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

- ☐ 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

- ☐ 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
ıread-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
```

- 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

“

Milliseconds

Metadata

Same configuration

Bare minimum

With details

Prepare for automation

Agenda

- Operating Systems
- Threads
- Layer Separation
- Network Problems
- External Interfaces
- 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

- 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 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

```
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
```

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
```

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

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

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

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

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

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

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

Summery

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

Summery

- 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

Summery

- 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

Summery

- 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

Summery

- 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
- Logs: Put timestamp with milliseconds…

Summery

- 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
- Logs: Put timestamp with milliseconds…
- Work with simulators

Summery

- 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
- Logs: Put timestamp with milliseconds…
- Work with simulators
- If it's not automated, it won't get done

Thank You!

Any questions?

Building Effective Embedded Systems: Architectural Best Practices

Gili.kamma@gmail.com

https://www.linkedin.com/in/**gili-kamma**/