Boosting Software Efficiency:

A Case Study of 100% Performance Improvement in an Embedded C++ System

GILI KAMMA

INTRO

- The talk today is about software development.
- I worked on a product for two years, and during this time, I drastically improved its capabilities.
- I wanted to share with you the journey I had.
- I hope you will get some useful ideas and inspiration to improve your own product.

I am Gili Kamma

20 years in the industry

B.S.c in Electronics

I love to improve things and solve problems

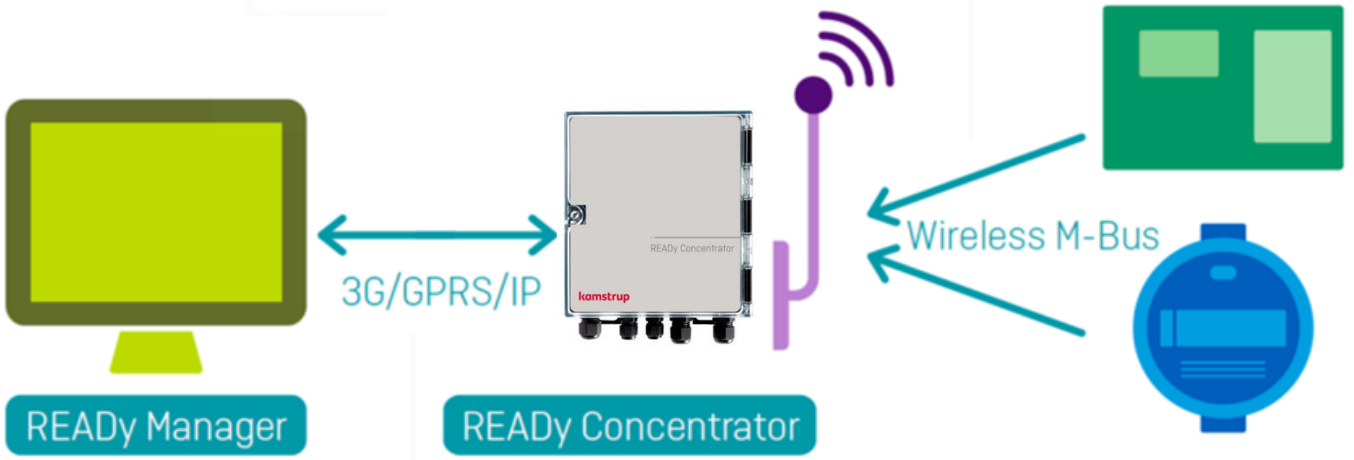
Team leader @ Priority-software

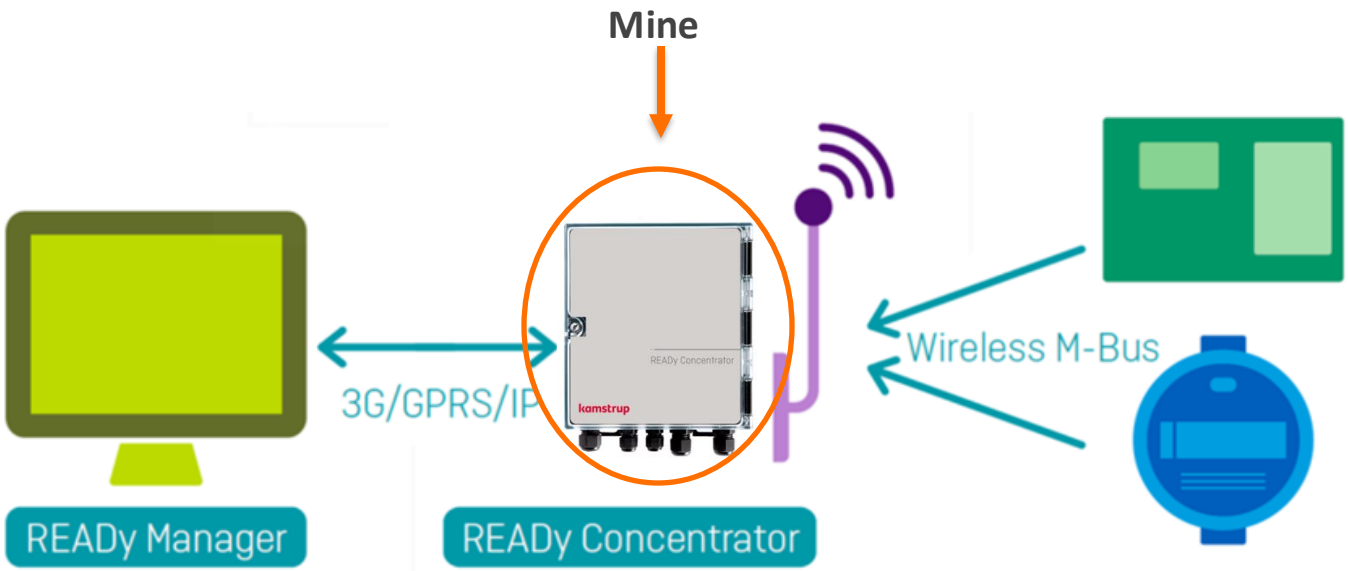
My story begins….

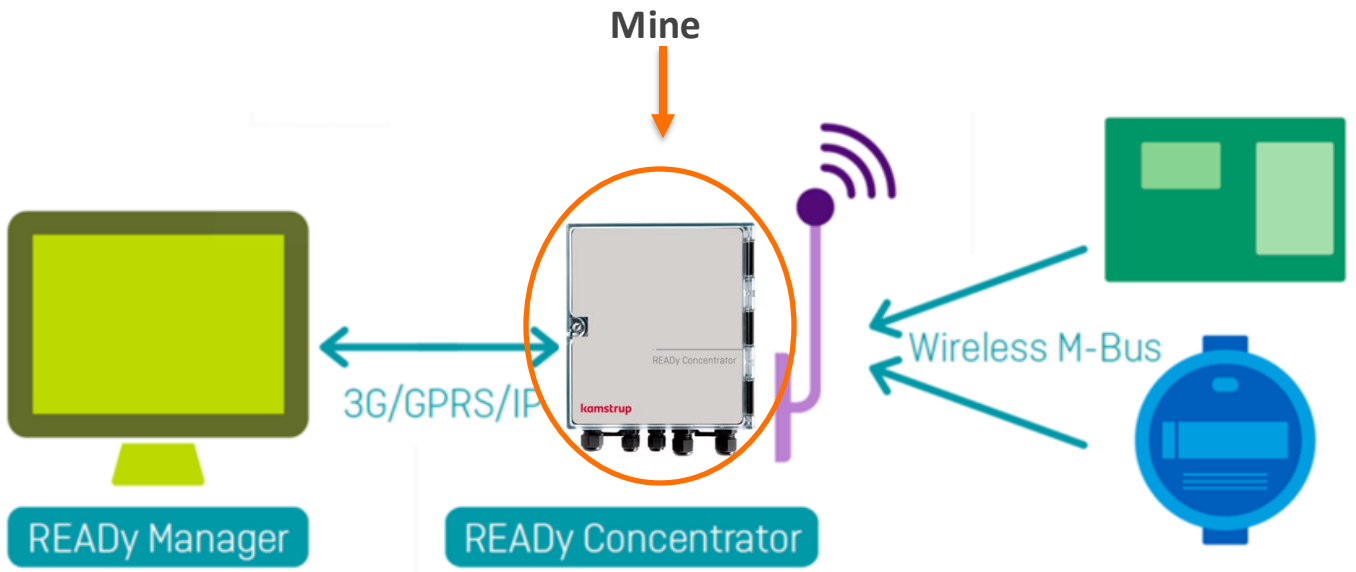
The business is meters (water, heat and electricity)

Smart metering

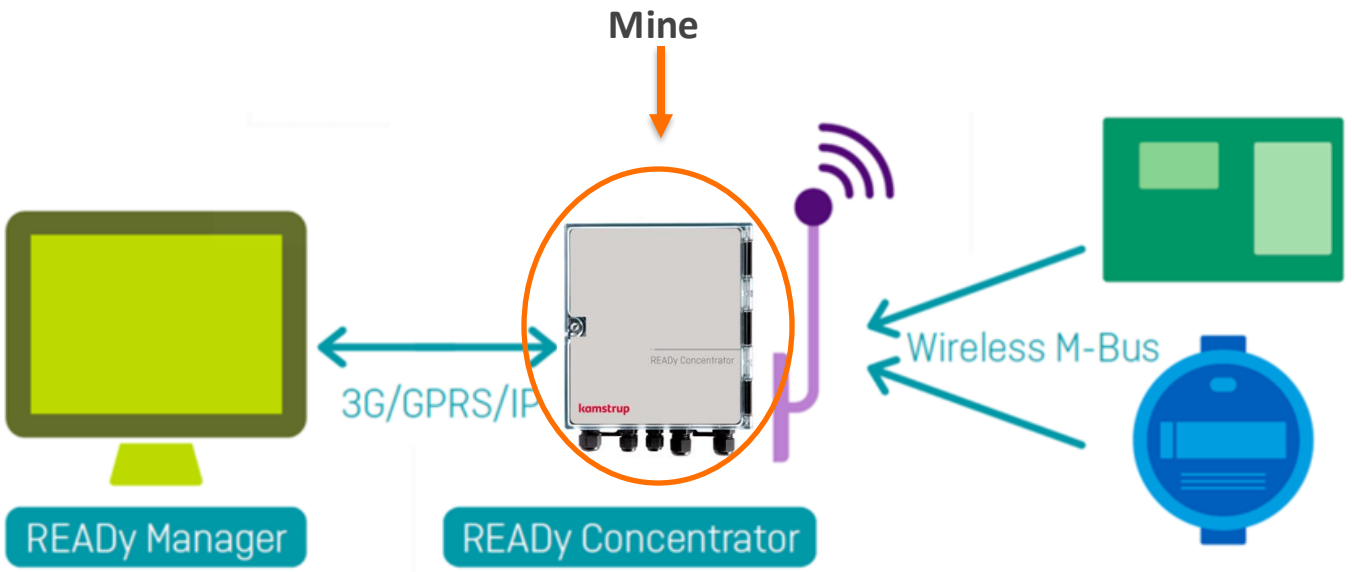
A system that uses digital devices to automatically send your utility data to the provider.


3G/GPRS/IP
READy Concentrator
kamstrup
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.



Loss of data → loss of money.

The Problems

Built to work with less than 1000 water meters – in real life was expected to work with 7500 meters.

Built to work with less than 1000 water meters – in real life was expected to work with 7500 meters.

- Unstable product.

Built to work with less than 1000 water meters – in real life was expected to work with 7500 meters.

- Unstable product.
- Unsatisfied customers.

Built to work with less than 1000 water meters – in real life was expected to work with 7500 meters.

- Unstable product.
- Unsatisfied customers.
- Unhappy developers.

- Memory issues – we didn't have enough memory.
- Unexplained crashes.
- Occasional data loss during network issues.

→ None of the errors were seen in the lab.

- Memory issues – we didn't have enough memory.
- Unexplained crashes.
- Occasional data loss during network issues.

→ None of the errors were seen in the lab.

- Memory issues – we didn’t have enough memory.
- Unexplained crashes.
- Occasional data loss during network issues.

→ None of the errors were seen in the lab.

- Memory issues – we didn’t have enough memory.
- Unexplained crashes.
- Occasional data loss during network issues.

→ None of the errors were seen in the lab.

Technology Stack

- Linux kernel, Yocto distribution – old image – no sources.
- 32 MB RAM.
- C++17 (in style of C++11 and earlier).
- Qt – an application framework – a wrapper to low-level programming.

- Linux kernel, Yocto distribution – old image – no sources.
- 32 MB RAM.
- C++17 (in style of C++11 and earlier).
- Qt – an application framework – a wrapper to low-level programming.

- Linux kernel, Yocto distribution – old image – no sources.
- 32 MB RAM.
- C++17 (in style of C++11 and earlier).
- Qt – an application framework – a wrapper to low-level programming.

- Linux kernel, Yocto distribution – old image – no sources.
- 32 MB RAM.
- C++17 (in style of C++11 and earlier).
- Qt – an application framework – a wrapper to low-level programming.

The Goal

Reduce (to zero) loss of data

and

create a stable product.

Reduce (to zero) loss of data

and

create a stable product.

How Am I Going to Do That?

First Impression

A lot of news & deletes

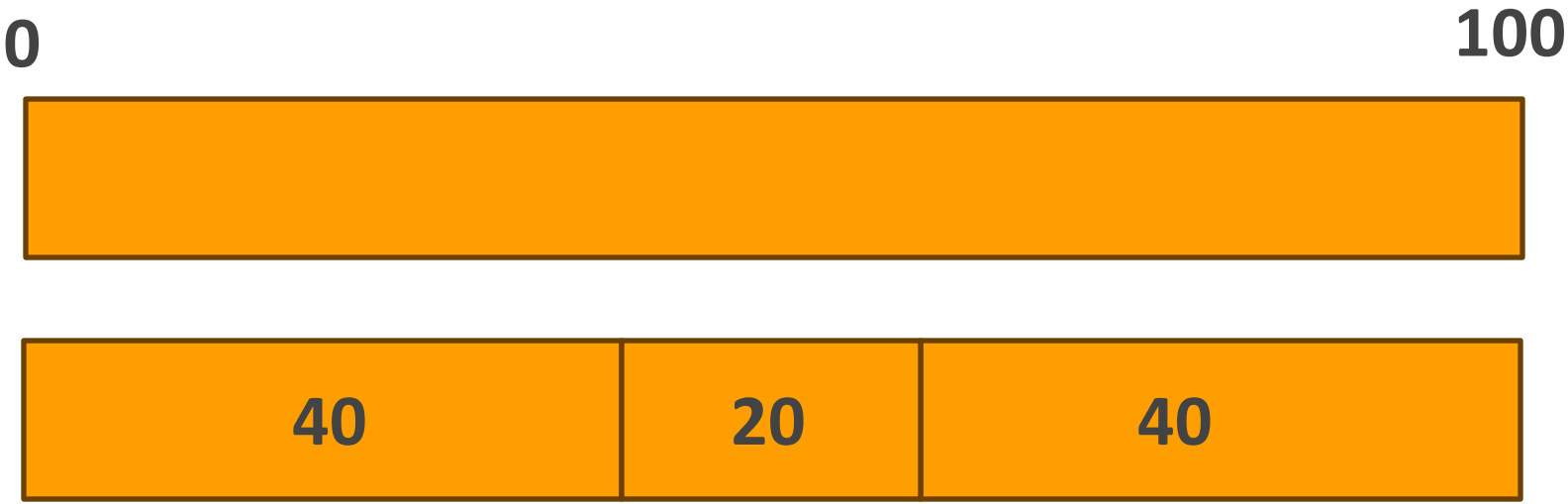
A lot of news & deletes

→ memory leaks & memory fragmentation

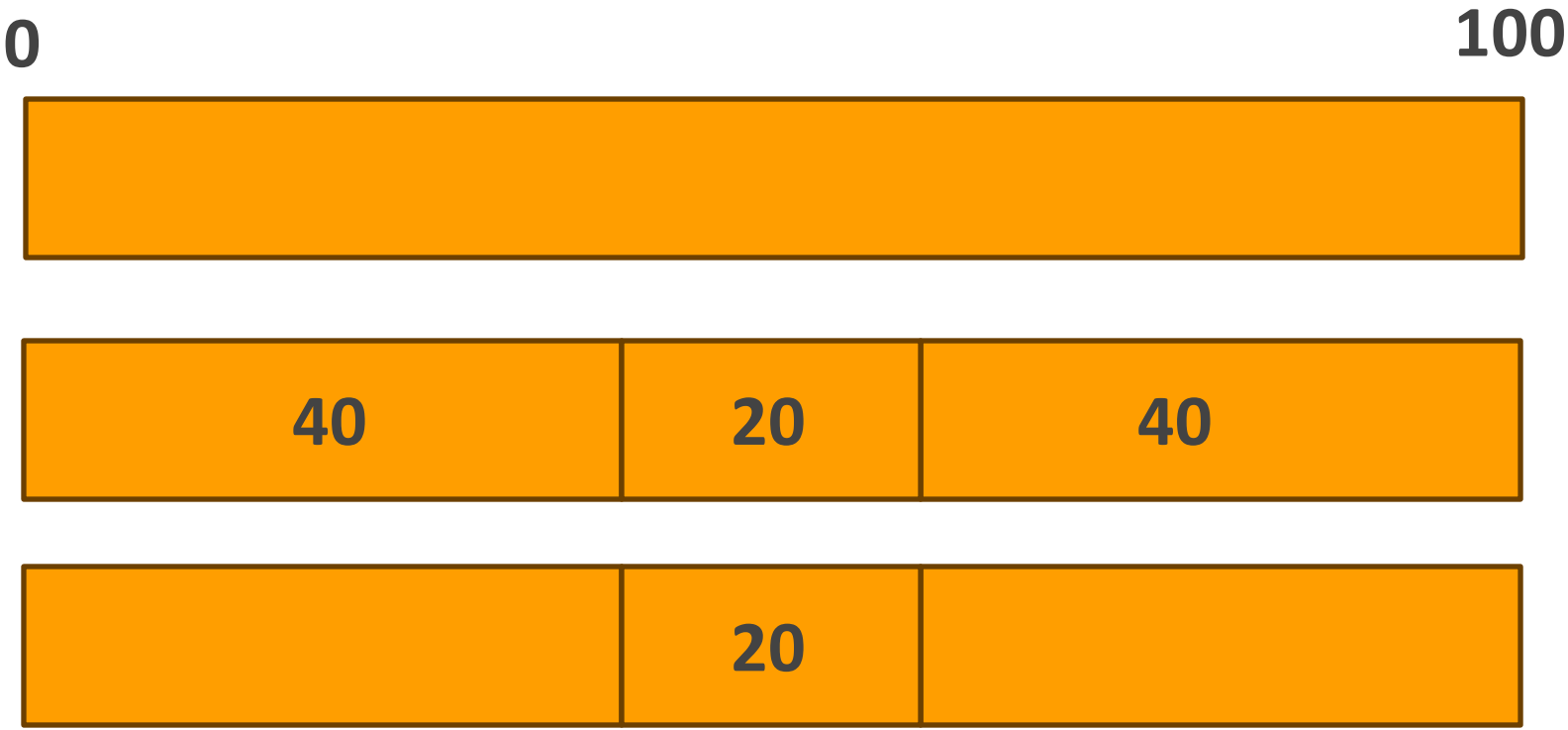
Memory fragmentation:

0 100

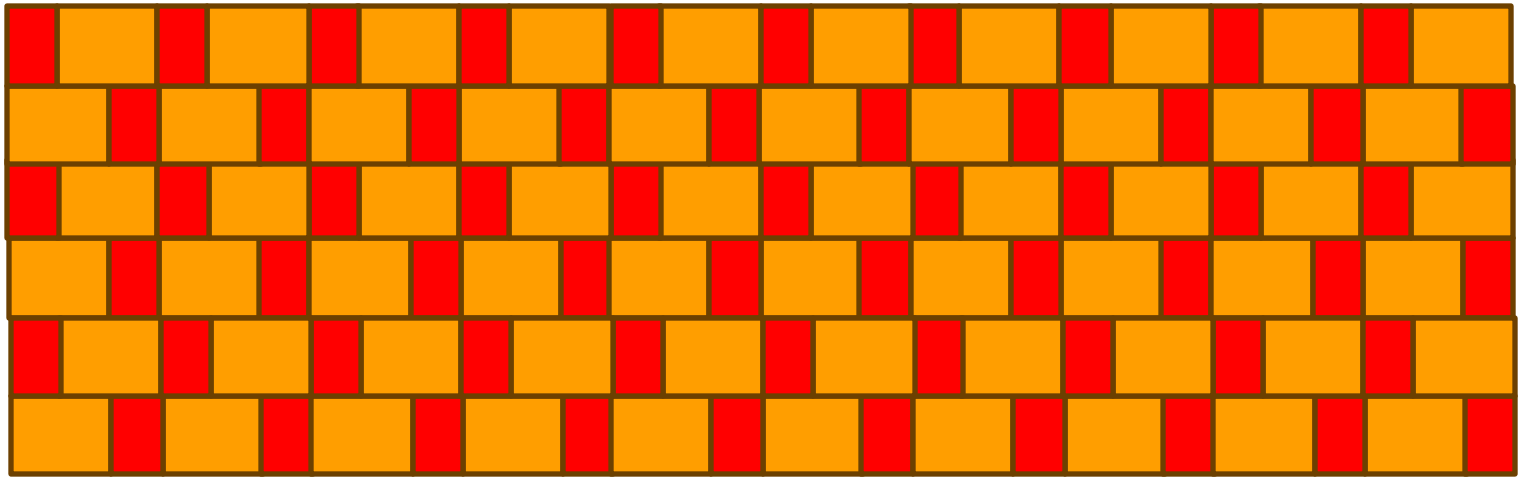
Memory fragmentation:



Memory fragmentation:



Memory fragmentation:



A lot of news & deletes

memory leaks & memory fragmentation

→ explains not enough memory

A lot of news & deletes

memory leaks & memory fragmentation

explains not enough memory

→ allocation fails

A lot of news & deletes

memory leaks & memory fragmentation

explains not enough memory

allocation fails

→ accessing null

A lot of news & deletes

memory leaks & memory fragmentation

explains not enough memory

allocation fails

accessing null

→ uncontrolled reset

A lot of news & deletes

memory leaks & memory fragmentation

explains not enough memory

allocation fails

accessing null

uncontrolled reset

→ Q.E.D 

Right?

Right?
Wrong!

Right?
Wrong!

It is not “wrong wrong” but it is wrong in this case

Wrong!

A lot of news & deletes

memory leaks & memory fragmentation

explains not enough memory

allocation fails

accessing null

uncontrolled reset

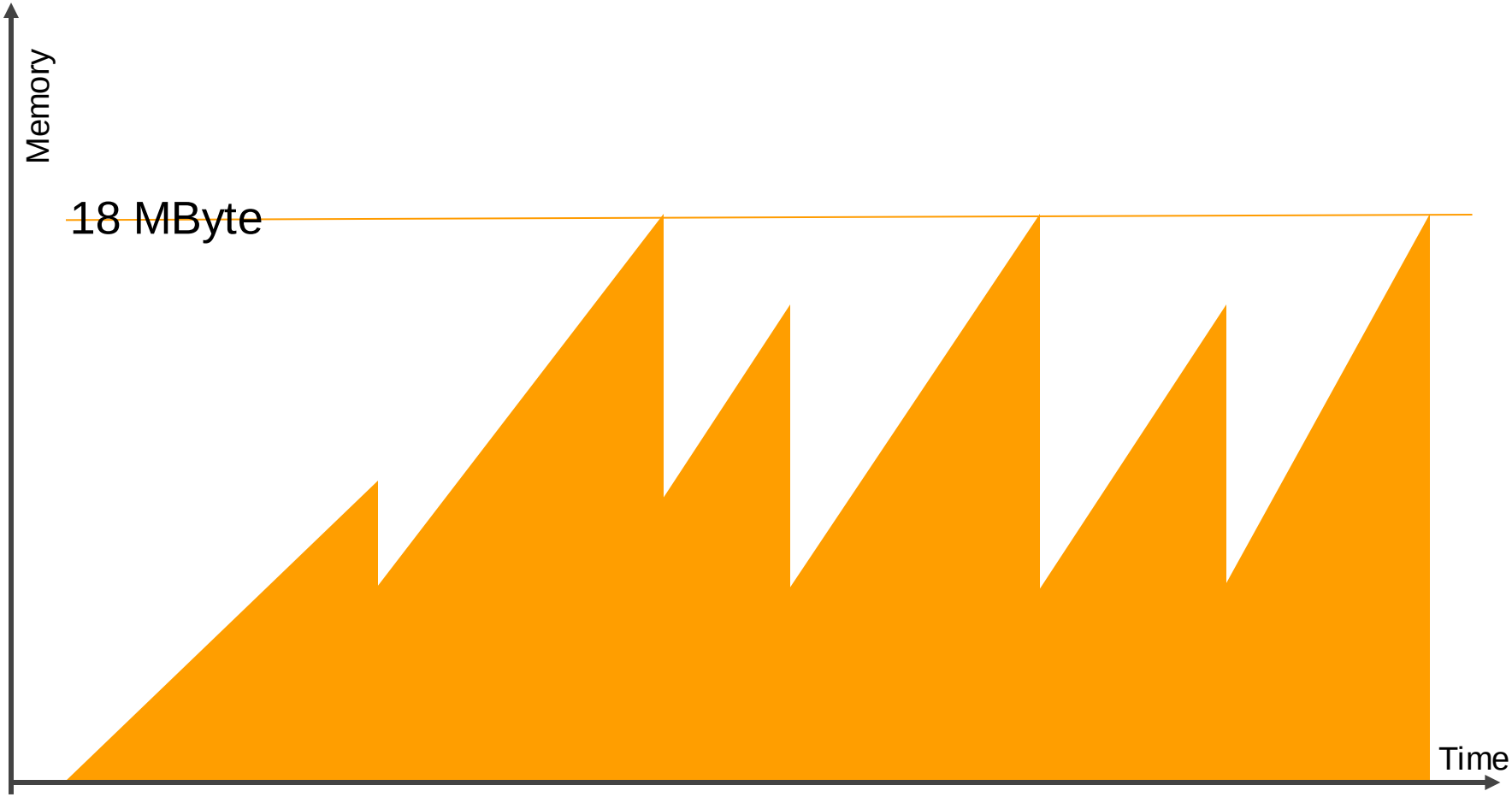
→ Q.E.D

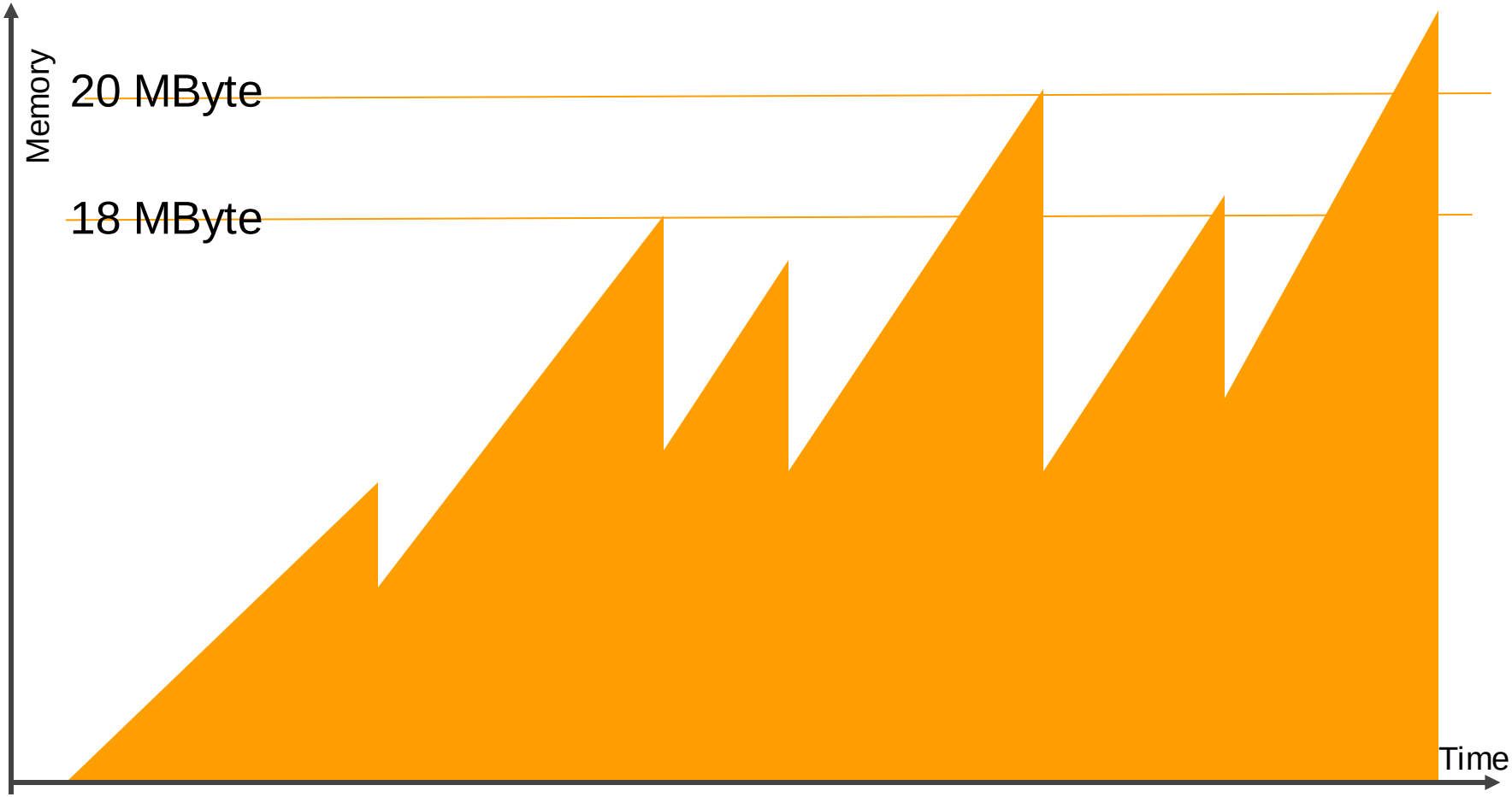
I needed to test my assumptions…

Memory Profiler

Gave me a good understanding of the sizes and the quantities of allocations.

I couldn’t use standard memory profilers as the Linux kernel was old.


Memory
18 MByte
Time


Memory
20 MByte
18 MByte
Time

Interesting measurements:

- Number of allocations per second.
- Current and maximum number of allocations.
- Current and maximum bytes allocated.
- Current and maximum allocations per size value.

Interesting measurements:

- Number of allocations per second.
- Current and maximum number of allocations.
- Current and maximum bytes allocated.
- Current and maximum allocations per size value.

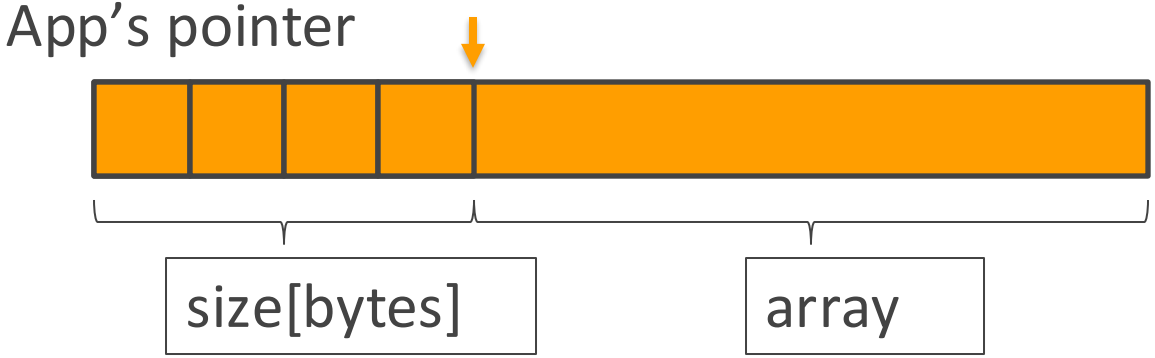
Interesting measurements:

- Number of allocations per second.
- Current and maximum number of allocations.
- Current and maximum bytes allocated.
- Current and maximum allocations per size value.

Interesting measurements:

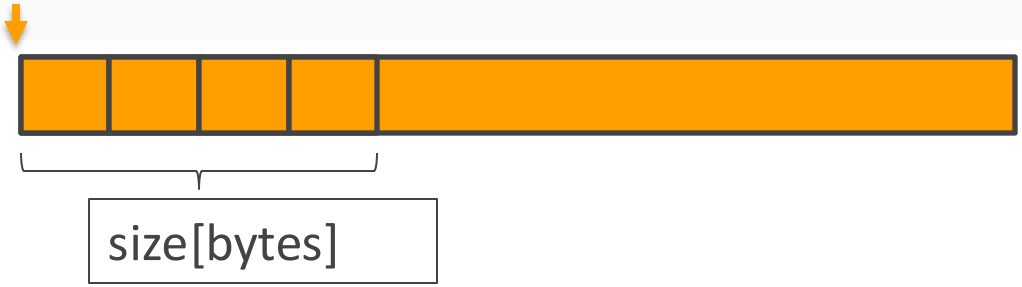
- Number of allocations per second.
- Current and maximum number of allocations.
- Current and maximum bytes allocated.
- Current and maximum allocations per size value.

Implementation



- Overloaded new and delete operators.
- Added 4 bytes to each allocation.

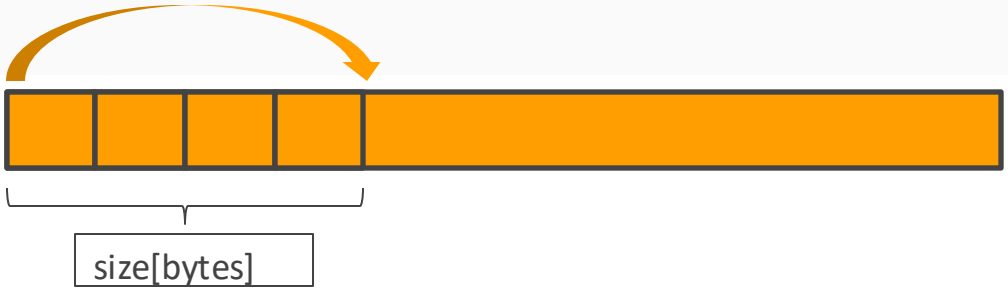
```
void* operator new(size_t size)
{
    void* ptr = malloc(size+4);
    *reinterpret_cast<size_t*>(ptr) = size;
    profiler.addAllocation(size);
    return (reinterpret_cast<char*>(ptr)+4);
}
```



```
void* operator new(size_t size)
{
    void* ptr = malloc(size+4);
    *reinterpret_cast<size_t*>(ptr) = size;
    profiler.addAllocation(size);
    return (reinterpret_cast<char*>(ptr)+4);
}
```

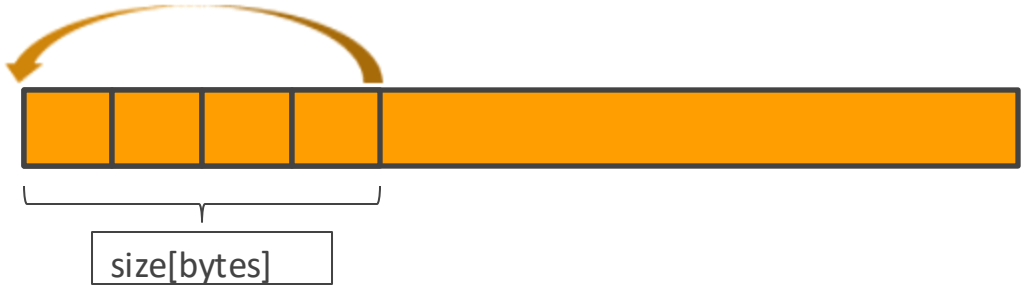
```
void* operator new(size_t size)
{
    void* ptr = malloc(size+4);
    *reinterpret_cast<size_t*>(ptr) = size;
    profiler.addAllocation(size);
    return (reinterpret_cast<char*>(ptr)+4);
}
```

```
void* operator new(size_t size)
{
    void* ptr = malloc(size+4);
    *reinterpret_cast<size_t*>(ptr) = size;
    profiler.addAllocation(size);
    return (reinterpret_cast<char*>(ptr)+4);
}
```

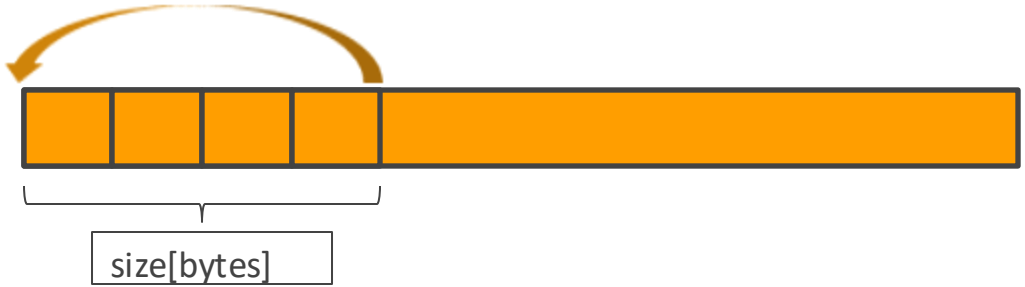


```
void operator delete(void* ptr, std::size_t)
{
    size_t size = *reinterpret_cast<size_t*>(reinterpret_cast<char*>(ptr)-4);
    profiler.removeAllocation(size);
    free(reinterpret_cast<char*>(ptr)-4);
}
```

```
void operator delete(void* ptr, std::size_t)
{
    size_t size = *reinterpret_cast<size_t*>(reinterpret_cast<char*>(ptr)-4);
    profiler.removeAllocation(size);
    free(reinterpret_cast<char*>(ptr)-4);
}
```



```
void operator delete(void* ptr, std::size_t)
{
    size_t size = *reinterpret_cast<size_t*>(reinterpret_cast<char*>(ptr)-4);
    profiler.removeAllocation(size);
    free(reinterpret_cast<char*>(ptr)-4);
}
```



```
void operator delete(void* ptr, std::size_t)
{
    size_t size = *reinterpret_cast<size_t*>(reinterpret_cast<char*>(ptr)-4);
    profiler.removeAllocation(size);
    free(reinterpret_cast<char*>(ptr)-4);
}
```

It helped me:

- Find the memory leaks and clean them.
- Figure out this is not my main problem.

It helped me:

- Find the memory leaks and clean them.
- Figure out this is not my main problem.

I've come across many problems, some big and some small.

Let’s talk more about them

Problems:

- Unexplained crashes.
- Not enough information after crash.
- Occasional data loss during network issues.
- Suddenly, no meters are being detected anymore.

Unexplained crash after crash continuously
(with gaps of several minutes)
in big sites (>5000 water meters).

How to simulate 5000+ water meters?

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

REPRODUCE THE ERROR

```
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  }
```

REPRODUCE THE ERROR

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

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.

What is special here:

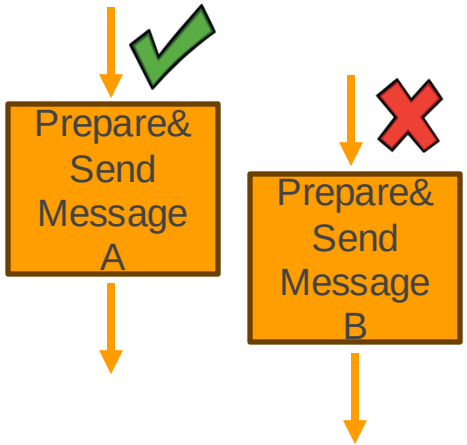
- No special hardware required.
- Simple implementation.
- Easy access to simulator mode without additional building.
- Consistently crashed the system.

```
void* operator new(size_t size)
{
    void* ptr = malloc(size+4);
    *reinterpret_cast<size_t*>(ptr) = size;
    profiler.addAllocation(size);
    return (reinterpret_cast<char*>(ptr)+4);
}
```

```
void* operator new(size_t size)
{
    void* ptr = malloc(size+4);
    *reinterpret_cast<size_t*>(ptr) = size;
    profiler.addAllocation(size);
    return (reinterpret_cast<char*>(ptr)+4);
}
```

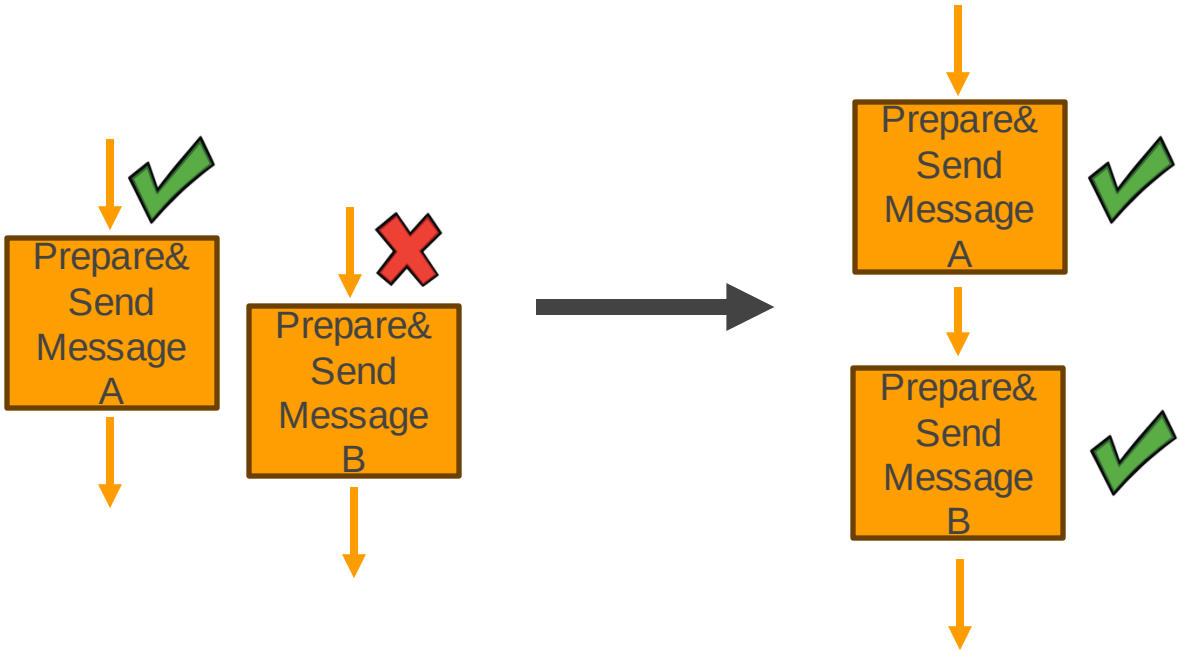
Two threads tried to create large messages about 5 MB each, at the same time.

The second one always failed.



Imagine this issue on your system. How would you solve it?

- Changed from asynchronous work to synchronous work.
- Split large messages into smaller ones.
- Rewrote the message preparation function using C++ instead of Qt.
- Used a static array instead of dynamic allocation.


Prepare&
Send
Message
A
Prepare&
Send
Message
B
Prepare&
Send
Message
A
Prepare&
Send
Message
B

- Asynchronous to synchronous

5MB+5MB ➡ 5MB

So, I am not crashing anymore. Can I stop here?

No!

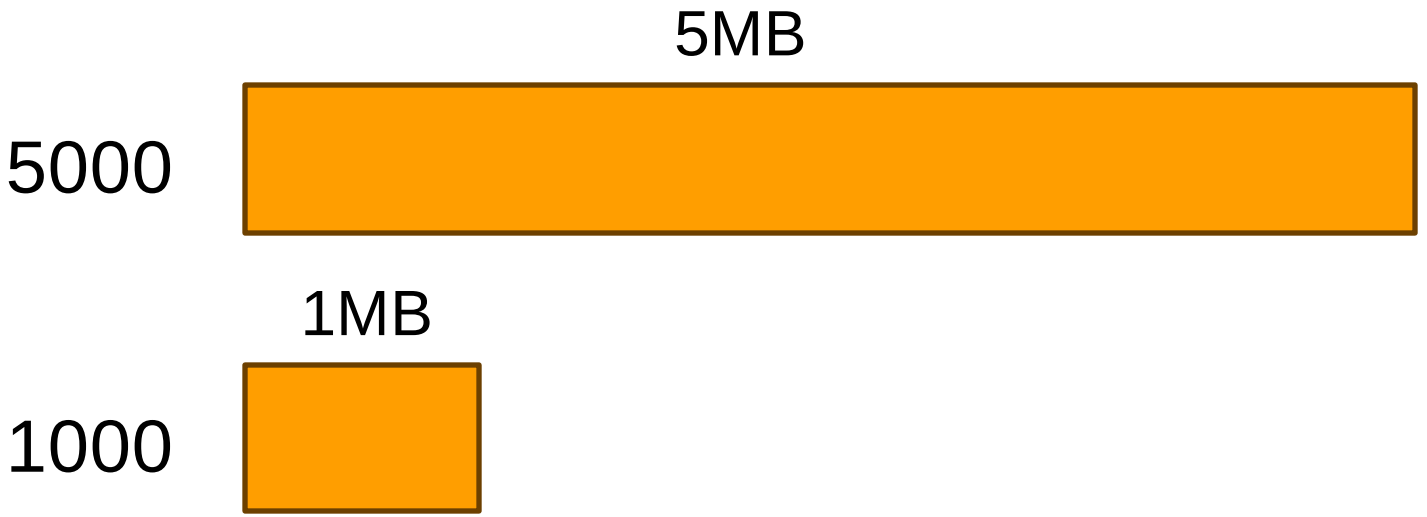
7500 meters and not just 5000

- Asynchronous to synchronous

5MB+5MB ➡ 5MB

Why doesn’t it happen in all sites?

The message size was proportional to the number of water meters the unit listened to.



- Changed from asynchronous work to synchronous work.
- Split large messages into smaller ones.
- Rewrote the message preparation function using C++ instead of Qt.
- Used a static array instead of dynamic allocation.

5MB

1X


5MB
1X
1MB
5X

- Asynchronous to synchronous

5MB+5MB ➡ 5MB

- Split large messages into smaller ones

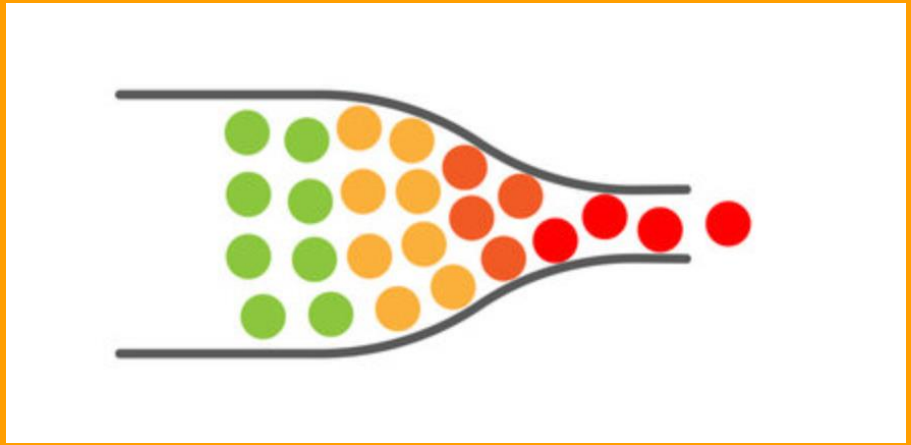
5MB ➡ 1MB

- Changed from asynchronous work to synchronous work.
- Split large messages into smaller ones.
- Rewrote the message preparation function using C++ instead of Qt.
- Used a static array instead of dynamic allocation.

- Asynchronous to synchronous — 5MB+5MB ➡ 5MB
- Split large messages into smaller ones — 5MB ➡ 1MB
- Qt to C++ — 1MB ➡ 0.5MB

- Changed from asynchronous work to synchronous work.
- Split large messages into smaller ones.
- Rewrote the message preparation function using C++ instead of Qt.
- Used a static array instead of dynamic allocation.

Bottle neck of the system



Problems:

- Unexplained resets.
- Not enough information after crash.
- Occasional data loss during network issues.
- Suddenly, no meters are being detected anymore.

- ❑ I had standard application logs.
- ❑ I didn’t have a remote access.
- ❑ I had an event mechanism (RAM).

- ❑ I had standard application logs.
- ❑ I didn’t have a remote access.
- ❑ I had an event mechanism (RAM).

- ❑ I had standard application logs.
- ❑ I didn’t have a remote access.
- ❑ I had an event mechanism (RAM).

When the unit wakes up, send the last 100 lines from the application log to the backend using the event service.

When the unit wakes up, send the last 100 lines from the application log to the backend using the event service.

- tail -n 100 /var/log/my_log.txt → shortLog
- SendEvent (”reset reason”, shortLog)

Good example to good enough solution

80% result / 20% effort

Problems:

- Unexplained resets.
- Not enough information after reset.
- Occasional data loss during network issues.
- Suddenly, no meters are being detected anymore.

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 (regardless of the current network status).

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:

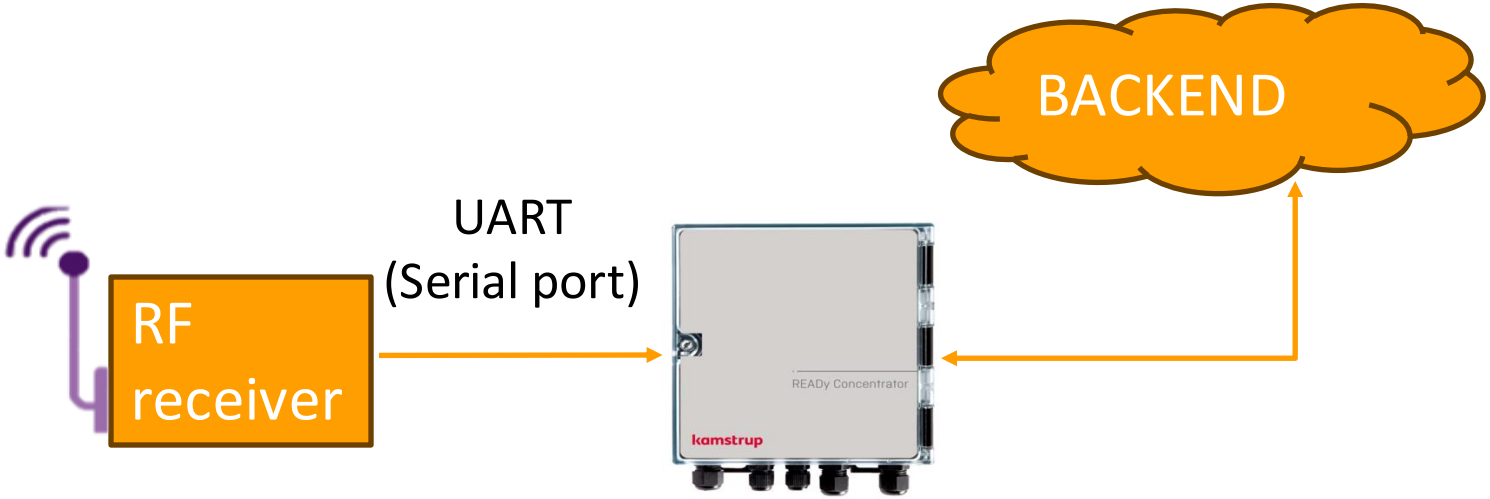
- Maximum data loss is now limited.
- Not being sensitive any more to network errors.

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

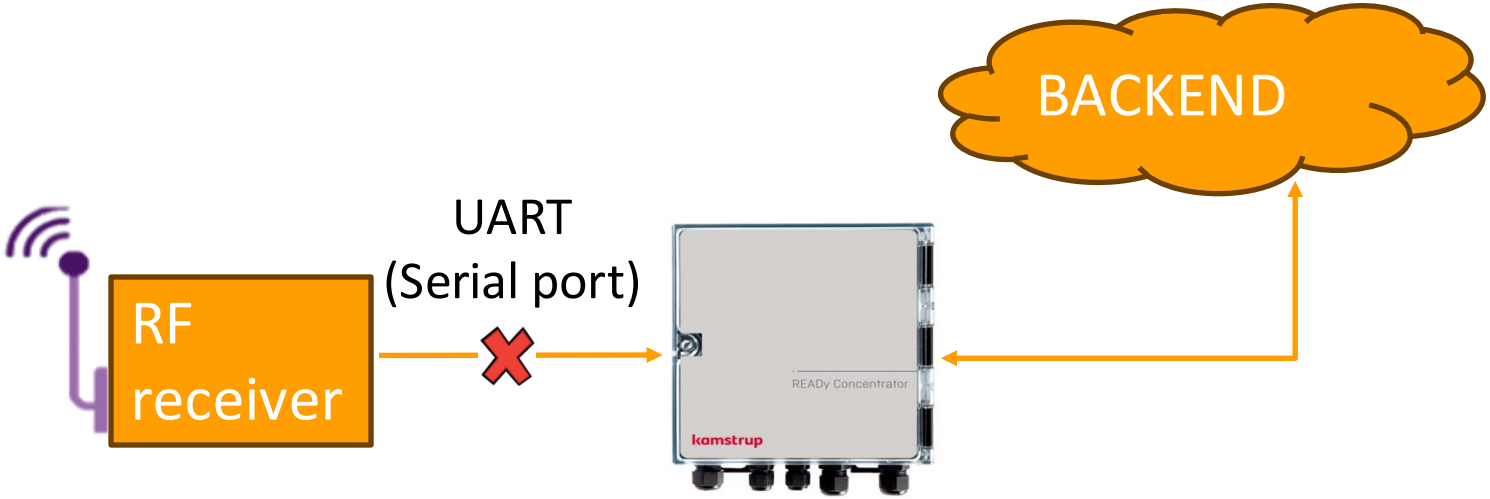
A good example of always using the same scenario to avoid any edge cases

Problems:

- Unexplained resets.
- Not enough information after reset.
- Occasional data loss during network issues.
- Suddenly, no meters are being detected anymore.



READy concentrator



READy concentrator

Once in a while we stopped receiving frames

(The system doesn’t recover on its own)

- Customers complained.
- Every time it was a different unit.
- We didn’t understand why it happens.
- Controlled Reset solved the problem.

- Customers complained.
- Every time it was a different unit.
- We didn’t understand why it happens.
- Controlled Reset solved the problem.

- Customers complained.
- Every time it was a different unit.
- We didn’t understand why it happens.
- Controlled Reset solved the problem.

- Customers complained.
- Every time it was a different unit.
- We didn’t understand why it happens.
- Controlled Reset solved the problem.

First solution:

- Reset the HW after 1 hour of silence.

First solution:

- Reset the HW after 1 hour of silence.

Second solution :

- Reset the HW after 4 minutes of silence (sweet spot).

First solution:

- Reset the HW after 1 hour of silence.

Second solution :

- Reset the HW after 4 minutes of silence (sweet spot).

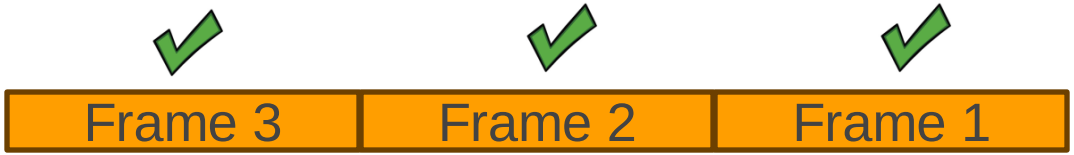
First solution:

- Reset the HW after 1 hour of silence.

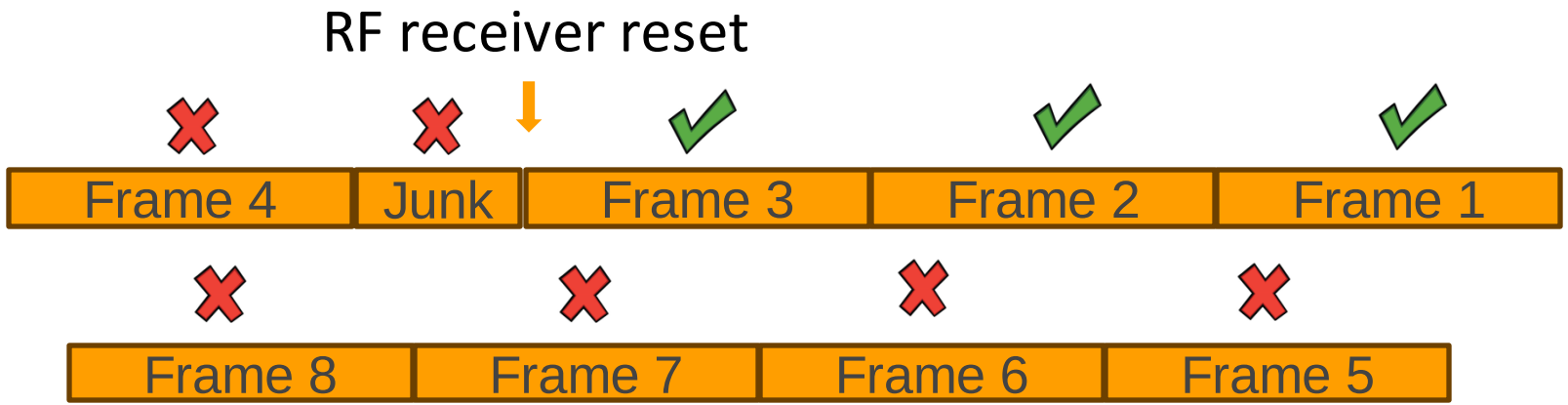
Second solution :

- Reset the HW after 4 minutes of silence (sweet spot).

Months later, I was looking in logs of a unit that had this silence issue – and I realized the RF receiver reset itself.



As a result, the READY concentrator couldn’t sync again on the next message.



A real solution:

- Initialized the UART when I detected silence instead of a controlled reset.

Sometimes it takes time to identify the real problem,
so we use temporary solutions in the meantime.

Monitoring

How it was when I started to work:

Complaints

Look at the event service

After several weeks:

Look at the event service

! Complaints

How I started:

- Started reading events for specific units, but it was uncomfortable.
- Two students proposed writing a script to retrieve the events and save them to a local text file.
- Wrote a Python script to analyze the text and export to an Excel file (sort, filter, plot).

How I started:

- Started reading events for specific units, but it was uncomfortable.
- Two students proposed writing a script to retrieve the events and save them to a local text file.
- Wrote a Python script to analyze the text and export to an Excel file (sort, filter, plot).

How I started:

- Started reading events for specific units, but it was uncomfortable.
- Two students proposed writing a script to retrieve the events and save them to a local text file.
- Wrote a Python script to analyze the text and export to an Excel file (sort, filter, plot).

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			

How I started:

- Ran the scripts every morning, analyzing for 10 minutes daily.
- Added more units and parameters.
- Automated the scripts to run at night and receive results by email in the morning.

How I started:

- Ran the scripts every morning, analyzing for 10 minutes daily.
- Added more units and parameters.
- Automated the scripts to run at night and receive results by email in the morning.

How I started:

- Ran the scripts every morning, analyzing for 10 minutes daily.
- Added more units and parameters.
- Automated the scripts to run at night and receive results by email in the morning.

Metrics (once a day) per unit

- ⊡ Max memory usage.
- ⊡ How many errors.
- ⊡ Last time to be seen.
- ⊡ How many resets.
- ⊡ How many meters were reported.
- ⊡ Current configuration.

Metrics (once a day) per unit

- Max memory usage.
- How many errors.
- Last time to be seen.
- How many resets.
- How many meters were reported.
- Current configuration.

Metrics (once a day) per unit

- Max memory usage.
- How many errors.
- Last time to be seen.
- How many resets.
- How many meters were reported.
- Current configuration.

Metrics (once a day) per unit

- Max memory usage.
- How many errors.
- Last time to be seen.
- How many resets.
- How many meters were reported.
- Current configuration.

Metrics (once a day) per unit

- Max memory usage.
- How many errors.
- Last time to be seen.
- How many resets.
- How many meters were reported.
- Current configuration.

Metrics (once a day) per unit

- Max memory usage.
- How many errors.
- Last time to be seen.
- How many resets.
- How many meters were reported.
- Current configuration.

Metrics (once a day) per unit

- Max memory usage.
- How many errors.
- Last time to be seen.
- How many resets.
- How many meters were reported.
- Current configuration.

Be proactive!

Use the data you already have

Testing

E2E tests:

- ☑ Rewrote the test paper from manual tests to automatic tests.
- ☑ Implemented the automated tests.
  - ☐ Two pipelines – long and short tests.
  - ☐ 24/7 - to stabilize the tests.

E2E tests:

- Rewrote the test paper from manual tests to automatic tests.
- Implemented the automated tests.
  - Two pipelines – long and short tests.
  - 24/7 - to stabilize the tests.

System tests:

- ~10 units are running with a load simulator and “aggressive configuration” 24/7.
- ~10 units are running regularly (with standard configuration) 24/7.

System tests:

- ~10 units are running with a load simulator and “aggressive configuration” 24/7.
- ~10 units are running regularly (with standard configuration) 24/7.

The question isn’t if there’s a bug - it’s who will find it first

How did it end?

8 Months Later

- Product without any resets.
- Supports 10,000 water meters instead of 5,000.

- Product without any resets.
- Supports 10,000 water meters instead of 5,000.

2 Years Later

- [ ] Test automation.
- [ ] (Half) automatic monitoring.
- [ ] Release every 2-3 months.
- [ ] Added features and improvements.

- [ ] Test automation.
- [ ] (Half) automatic monitoring.
- [ ] Release every 2-3 months.
- [ ] Added features and improvements.

- [ ] Test automation.
- [ ] (Half) automatic monitoring.
- [ ] Release every 2-3 months.
- [ ] Added features and improvements.

- Test automation.
- (Half) automatic monitoring.
- Release every 2-3 months.
- Added features and improvements.

Take Home Messages

Measure Everything – Avoid Assumptions

(Remember – my first impression)

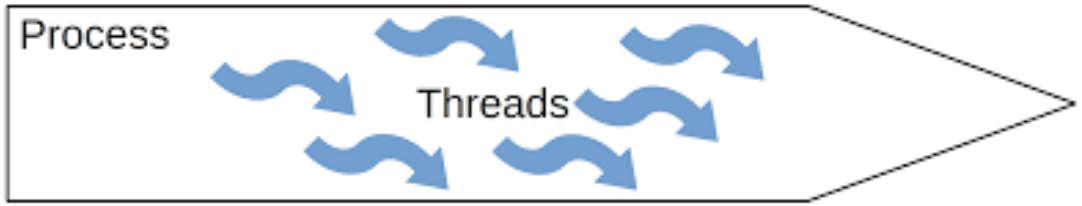
Write Robust Software

Unexpected inputs are unavoidable, but don’t let your system crash – have safeguards in place

- Split large messages into smaller, fixed sizes.
- Decouple logic and network operations.

- Split large messages into smaller, fixed sizes.
- Decouple logic and network operations.

Reduce the Number of Threads



(The most challenging bugs in the system arise from multiple threads running simultaneously)

Monitoring – Be Proactive.

Don't Wait for Customer Complaints

Create Simple Software

Creating simple software is more challenging than making it complex.

Thanks!

Any questions?