MTRN3500
Computing Applications in Mechatronics Systems

Serial Communication – Binary Data Sensors

T3-2020

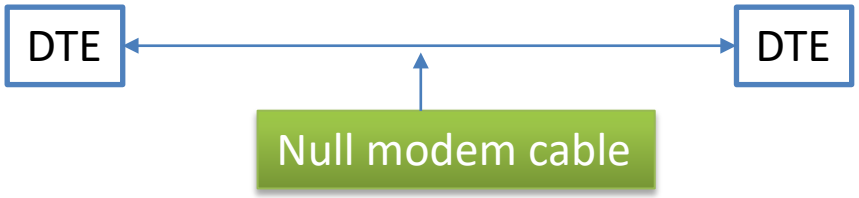
Use

- Some of the sensors we use have serial Interfaces. For example, the Novatel SMART V GPS sensor mounted on the UGV in the lab, in fact has a serial interface. Although we dispatch artificially created GPS data to you through Ethernet, the real sensor has a serial interface. Therefore it is very useful to be able to write software that interfaces control computers to sensors and actuators via serial ports. The serial standard we will be using is RS232C, which is the most commonly used interface.
- There are a number of elements involved with serial communication,
  - DTE – Data terminal equipment such as computer.
  - DCE – Data communication equipment – a modem.

Serial Communication Interface

- The full communication system is,

- When we connect a local sensor to a control computer, we do not need telephone lines and therefore modems.



  - The null modem cable has some of the wires twisted and looped to mimic the modems that physically does not exist.
  - For reliable connections, the cable length is best kept below 5m and in our case is about 1 m.
- To make this arrangement work, we must have physical serial ports in our computers. Generally, physical serial ports have names and on the PCs they range from COM1 to COM4.

USB-Serial Communication Interface

- Apart from industrial grade computers, nowadays most computers do not have serial ports, however, they may have USB ports or some facility to have USB devices connected to computers.
- The new approach is to have hardware that allows computers to communicate with serial devices through USB.
- From the computers point of view this is USB – Serial
- This requires making the USB devices appear as serial ports. These are called virtual serial ports in contrast to physical serial ports.
- This also means a driver is needed that comes with the USB-Serial convertor which makes the USB port appear as a serial port.
- From programming point of view, there is hardly any difference. We somehow need to figure out the virtual port assigned to the USB-Serial device.
- Device manager helps you to figure this out.

Elements Needed for Serial Communication

- We need a Serial Port object
- We need a Port name, if virtual port device manager will help.
- We need a place to store binary data to be transmitted.
- We need a place to store binary data received.
- We need a mechanism to send data.
- We need a mechanism to receive data.

Programming Serial Port

- Programming serial port is almost similar to programming the TCP client.
- However, there is no client/server distinction. The connections are symmetrical and always one to one.
- No Master/slave arrangement and they cannot be daisy chained.
- First include the following:

```
#using <System.dll>
//Either include metadata from System.dll by including it here or
//by going to project properties and then C++->Advanced->Forced #using file
using namespace System;
using namespace System::IO::Ports;
```

Declarations

```
SerialPort^ Port = nullptr;
String^ PortName = nullptr;
array<unsigned char>^ SendData = nullptr;
array<unsigned char>^ RecvData = nullptr;

unsigned int Checksum;
double Northing;
double Easting;
double Height;
```

Instantiations

```
Port = gcnew SerialPort;
PortName = gcnew String("COM1");
SendData = gcnew array<unsigned char>(16);
RecvData = gcnew array<unsigned char>(??);
// We will discuss the RecvData buffer size
```

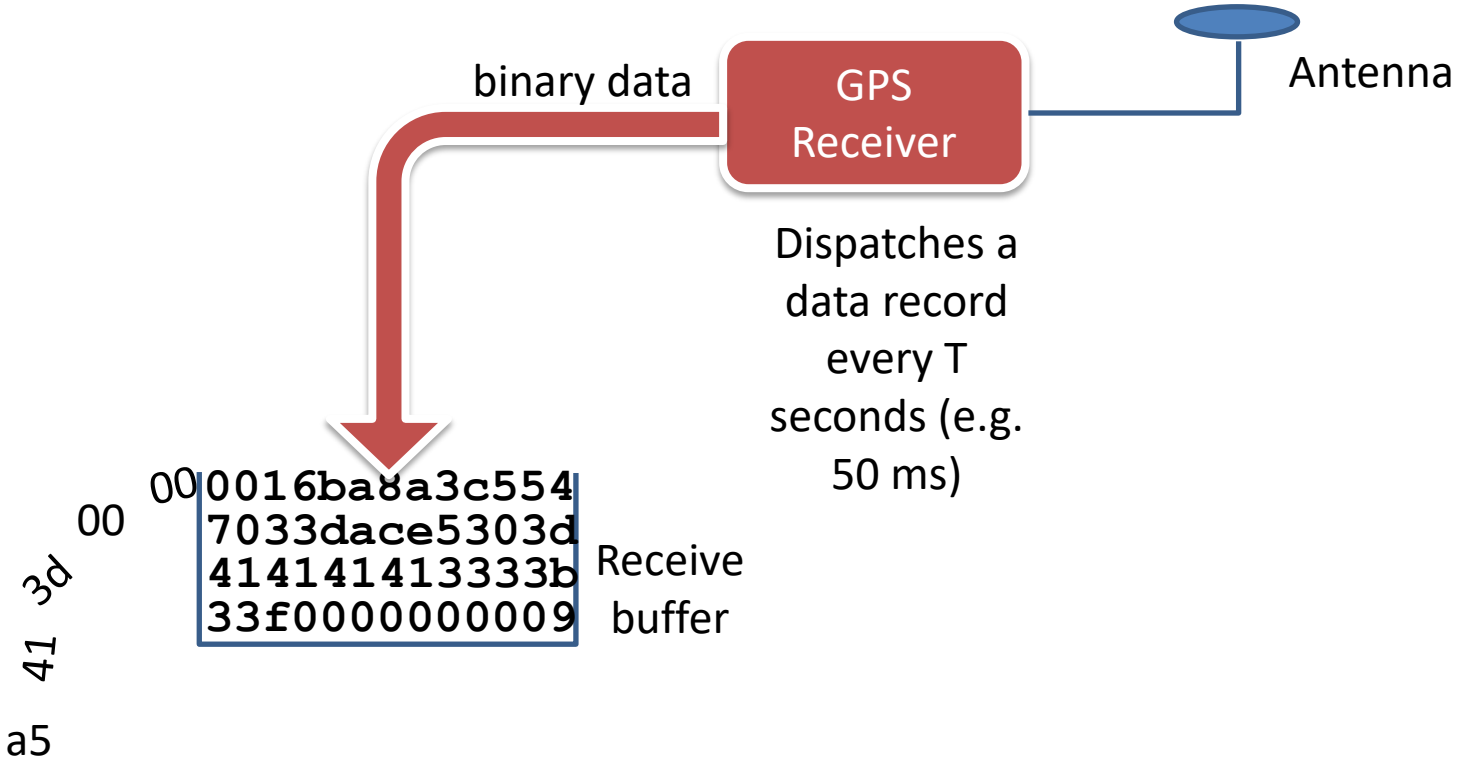
Configurations

```
Port->PortName  = PortName;
Port->BaudRate  = 115200;
Port->StopBits  = StopBits::One;
Port->DataBits  = 8;
Port->Parity    = Parity::None;
Port->Handshake = Handshake::None;

// Set the read/write timeouts & buffer size
Port->ReadTimeout      = 500;
Port->WriteTimeout     = 500;
Port->ReadBufferSize   = ??;
Port->WriteBufferSize = 1024;
```

Actions

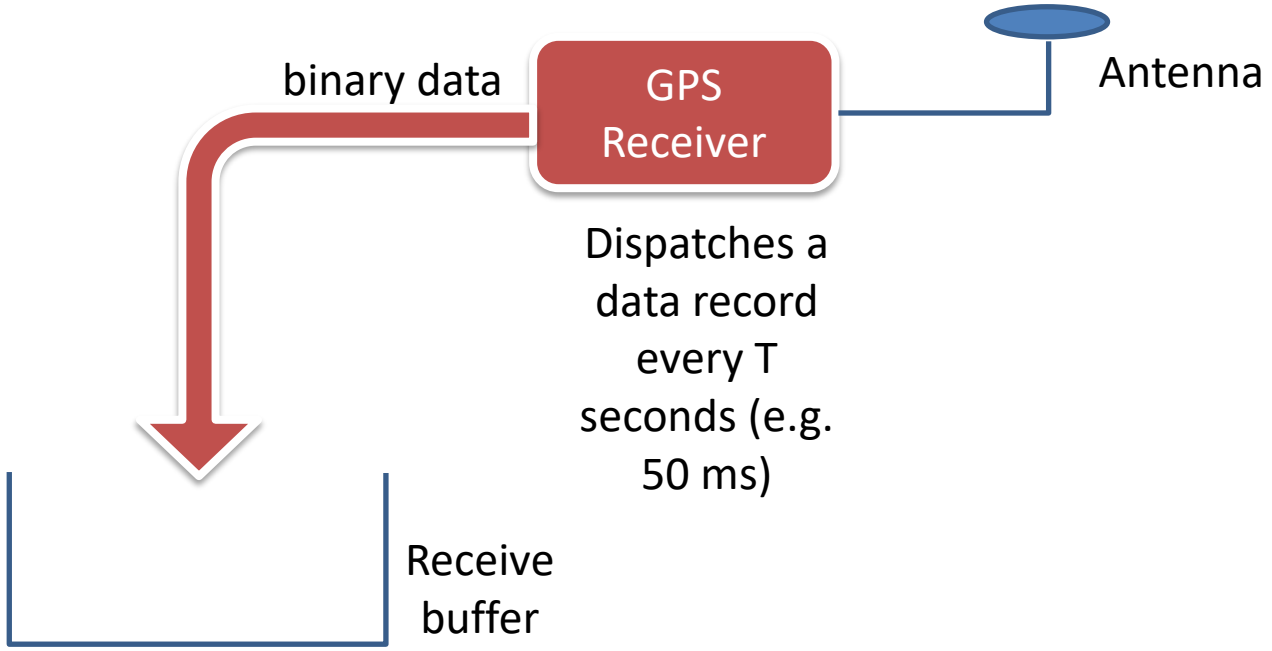
```
Port->Open();

Port->Read(RecvData, 0, sizeof(GPS));
```

How does GPS systems work



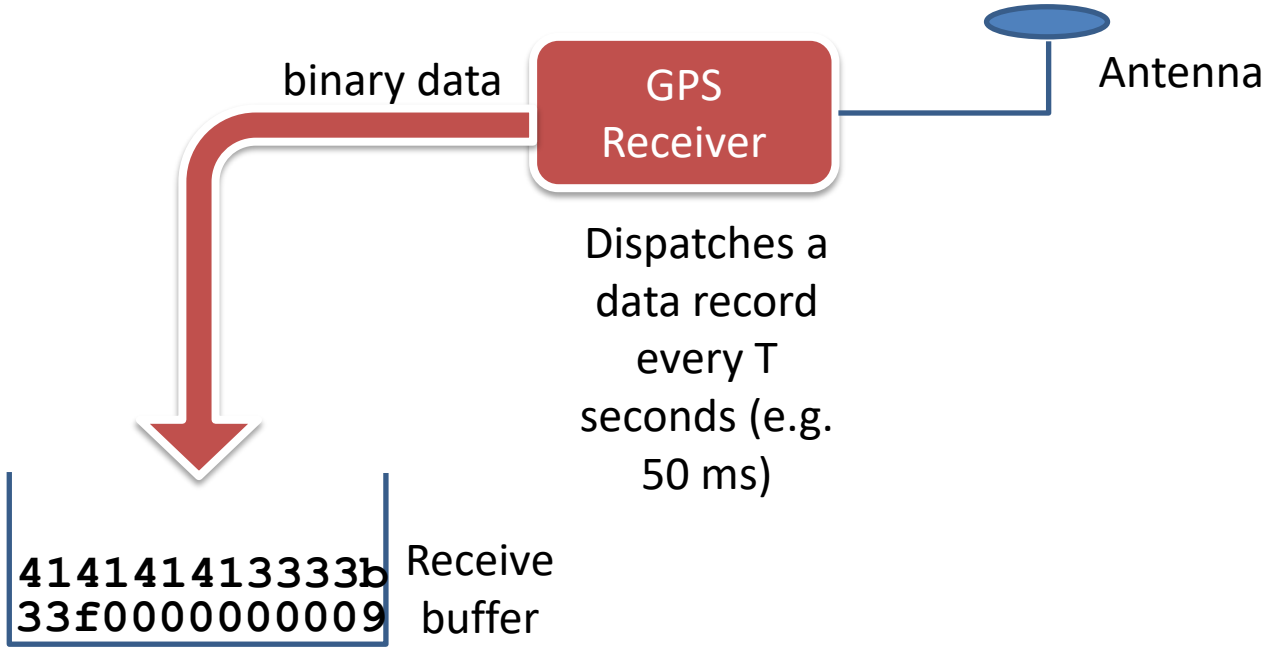
If we do not read the buffer quickly this is what happens. The old data remains in the buffer. New data disappears. The GPS we would get will be too old.

How does GPS systems work



The buffer is empty immediately after a read, provided we read a number of bytes equal to or greater than the buffer size.

How does GPS systems work



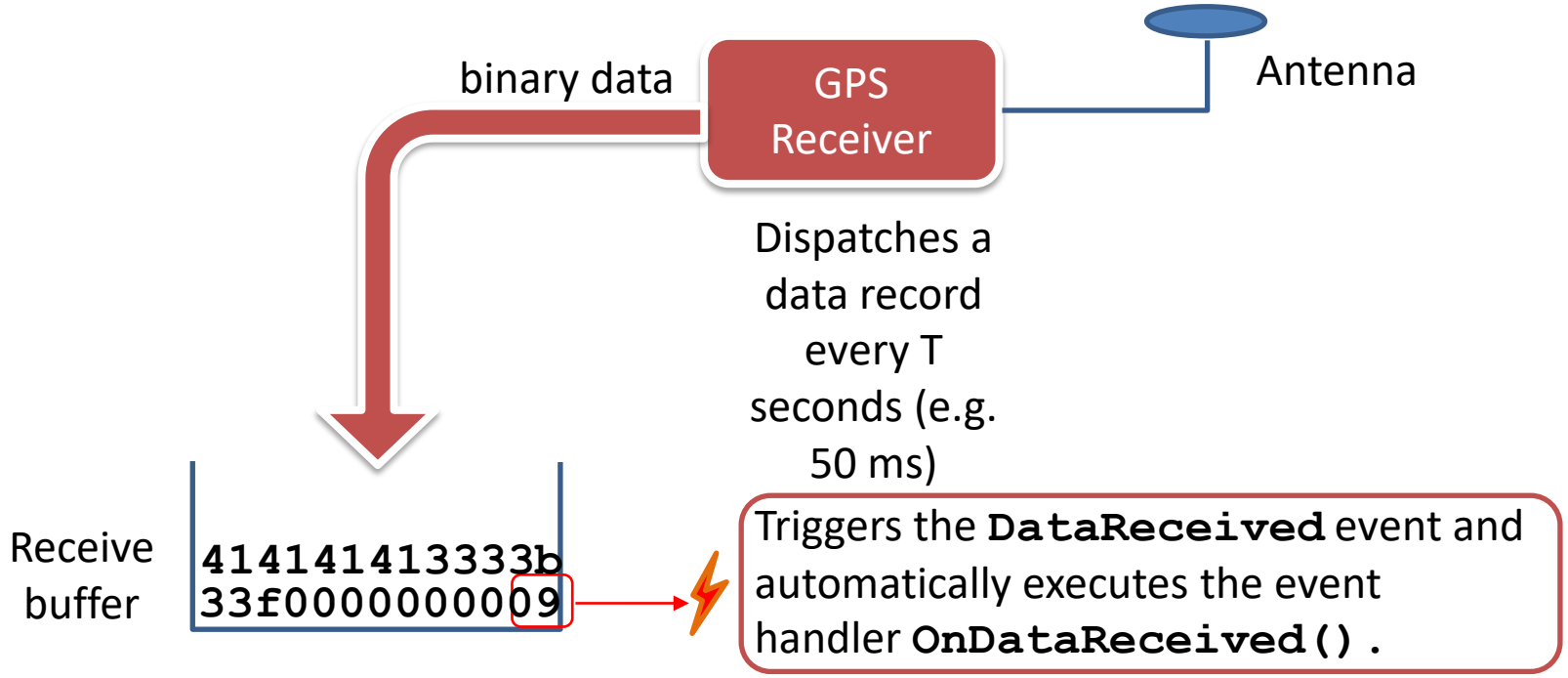
- Create the buffer size > data packet size
- Start by clearing the receive data buffer.
- Read data immediately after they arrive.
- How? We can use events.

Events of Serial Communication Interface

- Serial communication provides four events.
- One of them is the **`DataReceived`** event.
- We can then attach a **`OnDataReceived`** event handler to deal with reading data. This way data will be read immediately.
- However, we have to wait till we get the full data packet. For this we can test **`BytesToRead`** property of the **`SerialPort`** class.
- Instead of trapping the header, check if first four bytes are the header byte values.

- This approach is good for slow data rates (T > 50 ms)

Using `DataReceived` Event



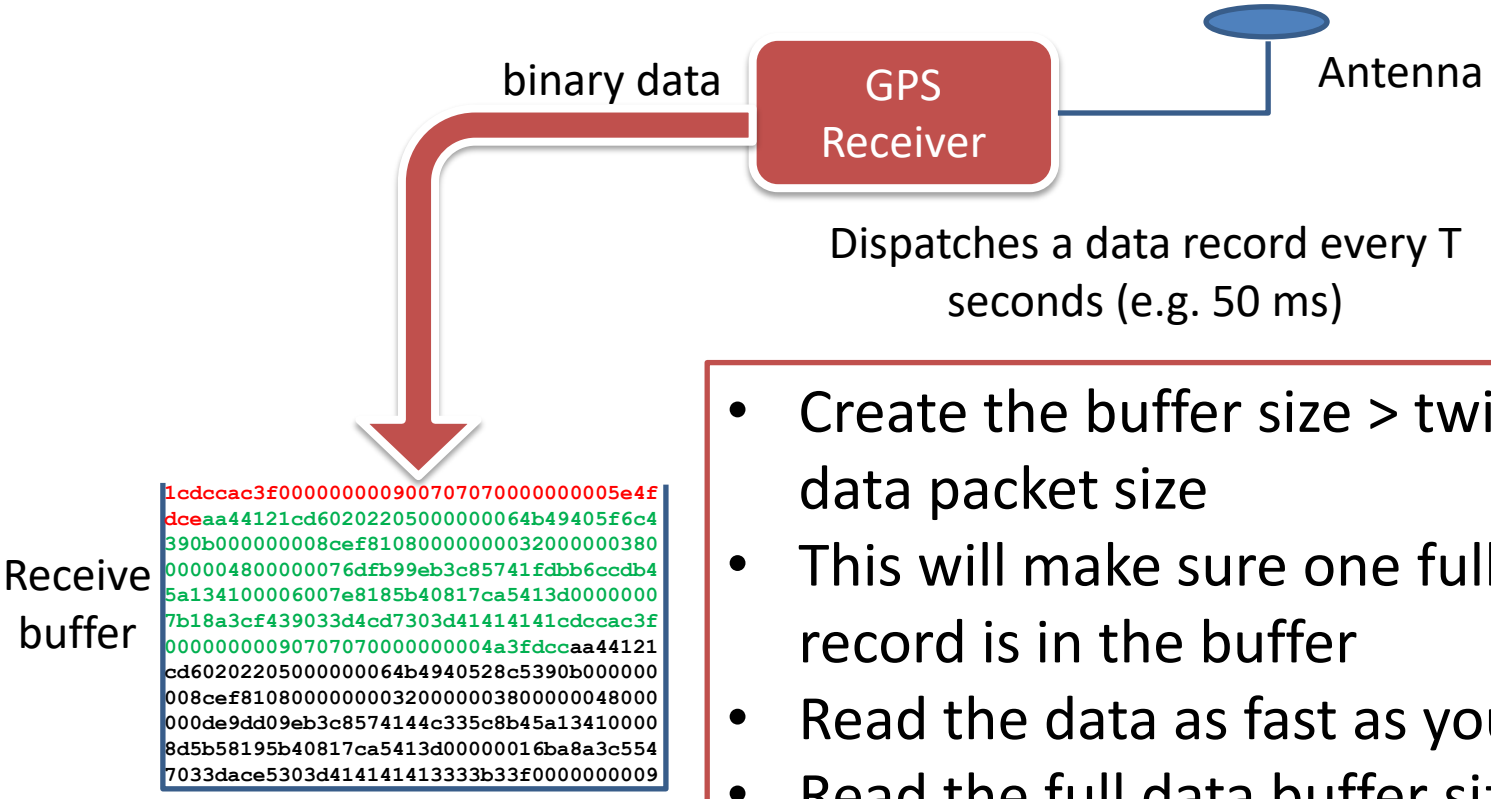
- Within the event handler, we must make sure a full data record has been received into the buffer.
- Then we read the data into **`RecvData`**.
- We then transfer **`RecvData`** to a structure.

An Alternative Approach

- This approach does not give you tightly timed GPS results.
- If you can afford to throw away one data record or a part thereof every time, you can use this approach.
- Create a buffer twice as large or more as the data packet size.
- Read the entire buffer and then process the data.
- This requires locating a good data packet in the buffer.
- In continuous data streams this may be a better approach.

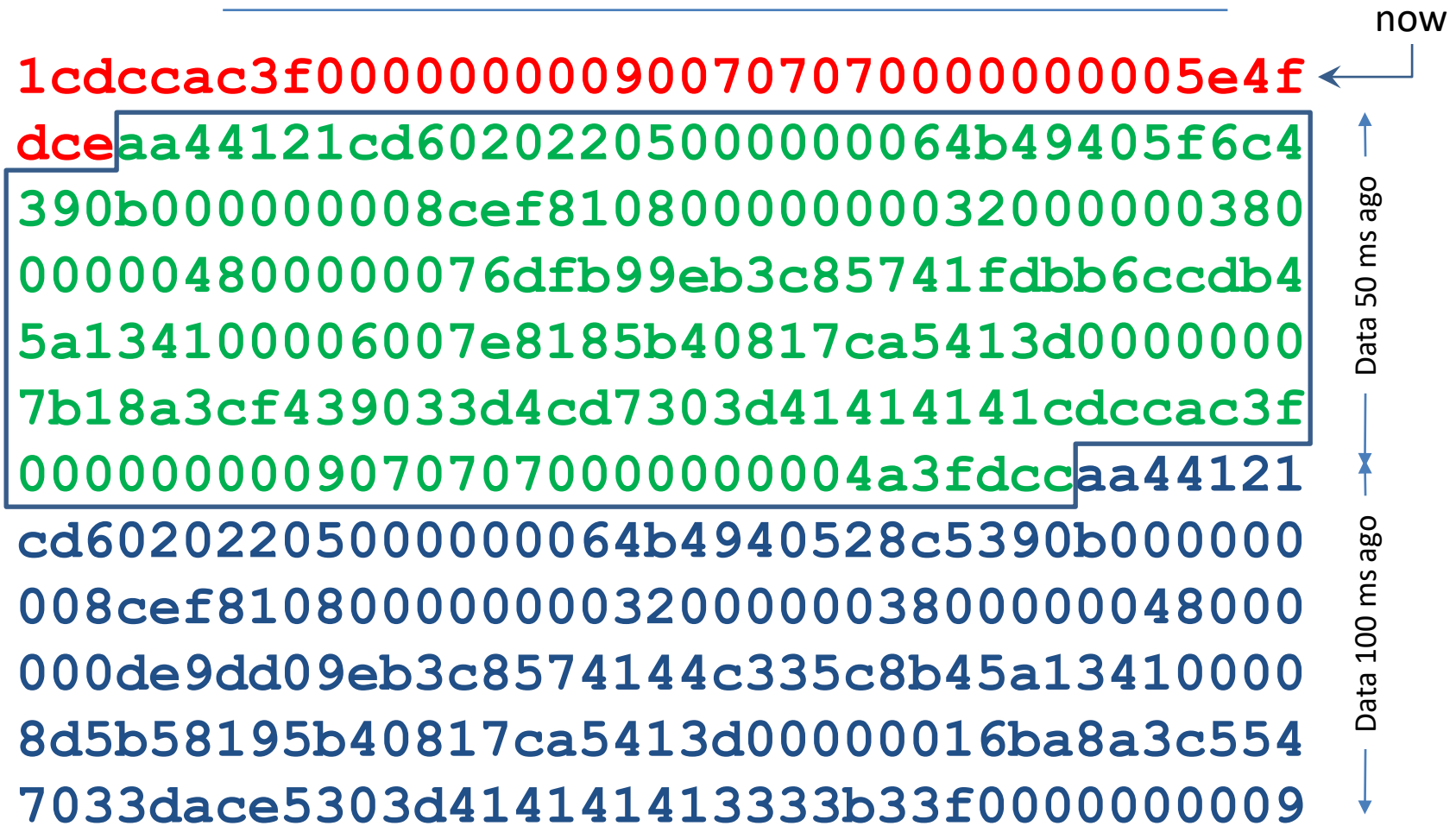
- This approach is good for high data rates (T < 20 ms).

An Alternative Approach



- Create the buffer size > twice the data packet size
- This will make sure one full data record is in the buffer
- Read the data as fast as you can
- Read the full data buffer size or empty the data buffer after reading the number of bytes required.

Sample GPS Data Stream



Trapping the Header

```
unsigned int Header = 0;
int i = 0;
int Start; //Start of data
do
{
  Data = RecvData[i++];
  Header = ((Header << 8) | Data);
}
while (Header != 0xaa44121c);
Start = i - 4;
```

- Note that from here on it has nothing to do with serial communication or TCP communication. This is a way of extracting useful data from an array of **`unsigned char`**.

Forming a Structure

```
struct GPS//112 bytes
{
  unsigned int Header;
  unsigned char Discards1[40];
  double Northing;
  double Easting;
  double Height;
  unsigned char Discards2[40];
  unsigned int Checksum;
};
```

Filling in the Structure

```
GPS  NovatelGPS;
unsigned char *BytePtr = nullptr;
BytePtr = (unsigned char*)&NovatelGPS;

for(int i = Start; i < Start+sizeof(GPS); i++)
{
  *(BytePtr++) = RecvData[i];
}

Console::WriteLine("{0:F3} ", NovatelGPS.Easting); // ok
```