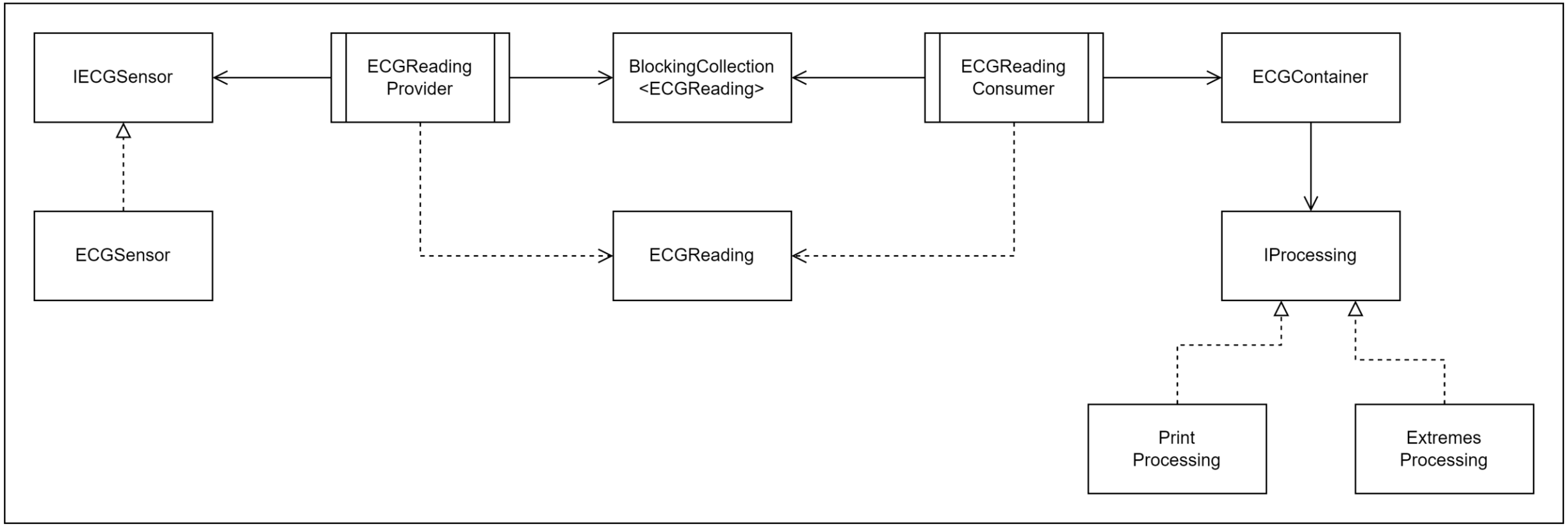
Client-Server communication in C#
With blocking collection

AARHUS UNIVERSITY
AARHUS UNIVERSITY SCHOOL OF ENGINEERING

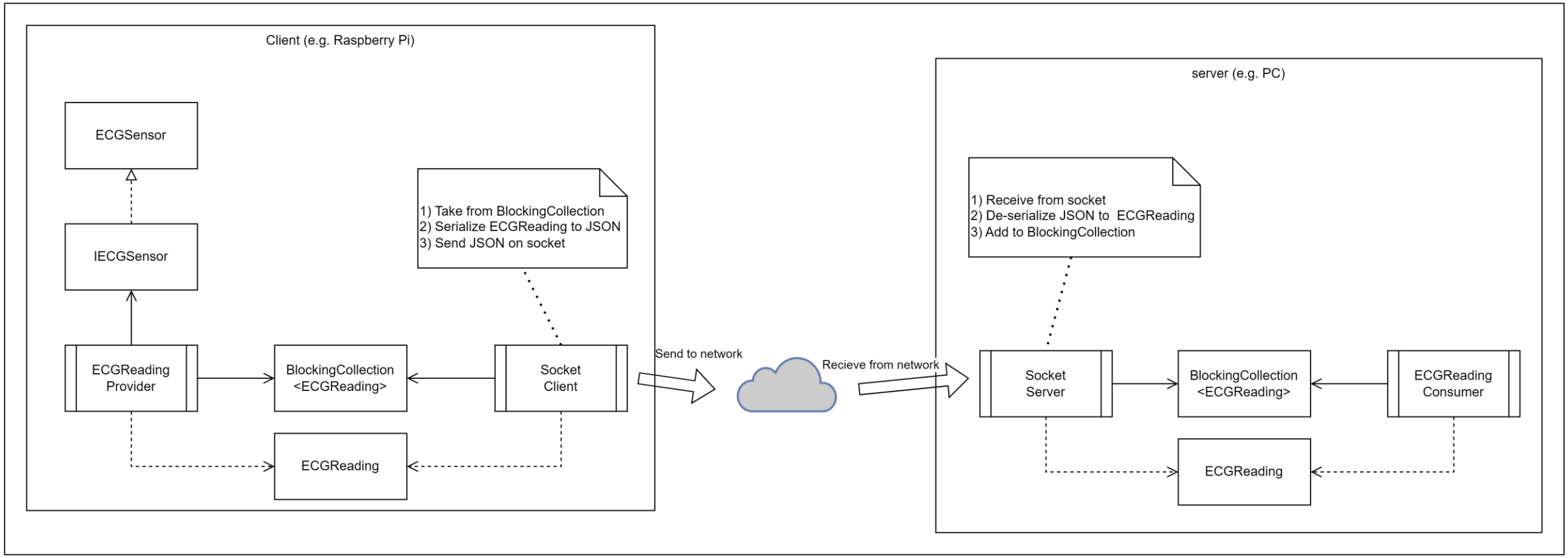
MICHAEL LOFT
ML@ASE.AU.DK

Where we want to end up

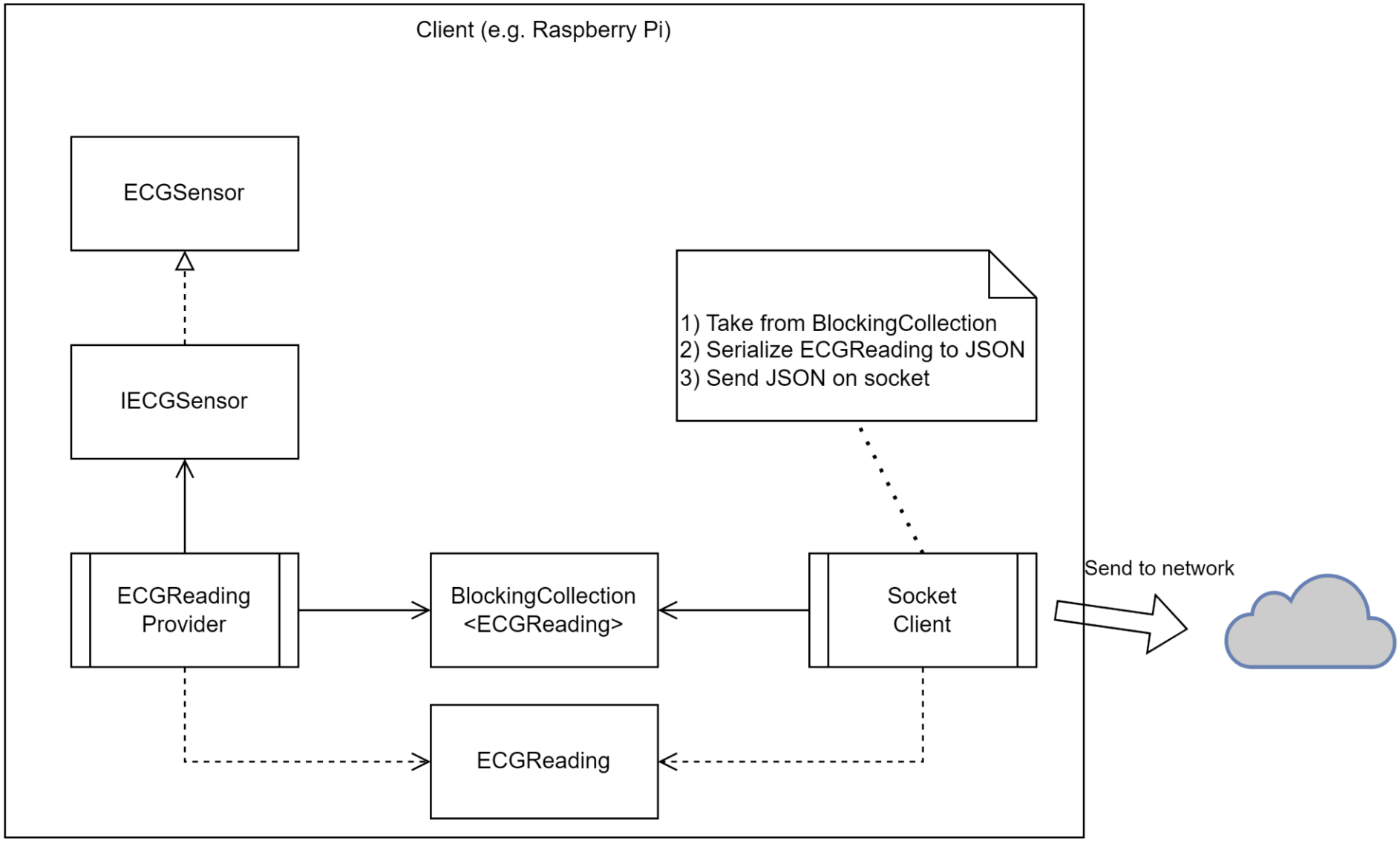
We want to go from this



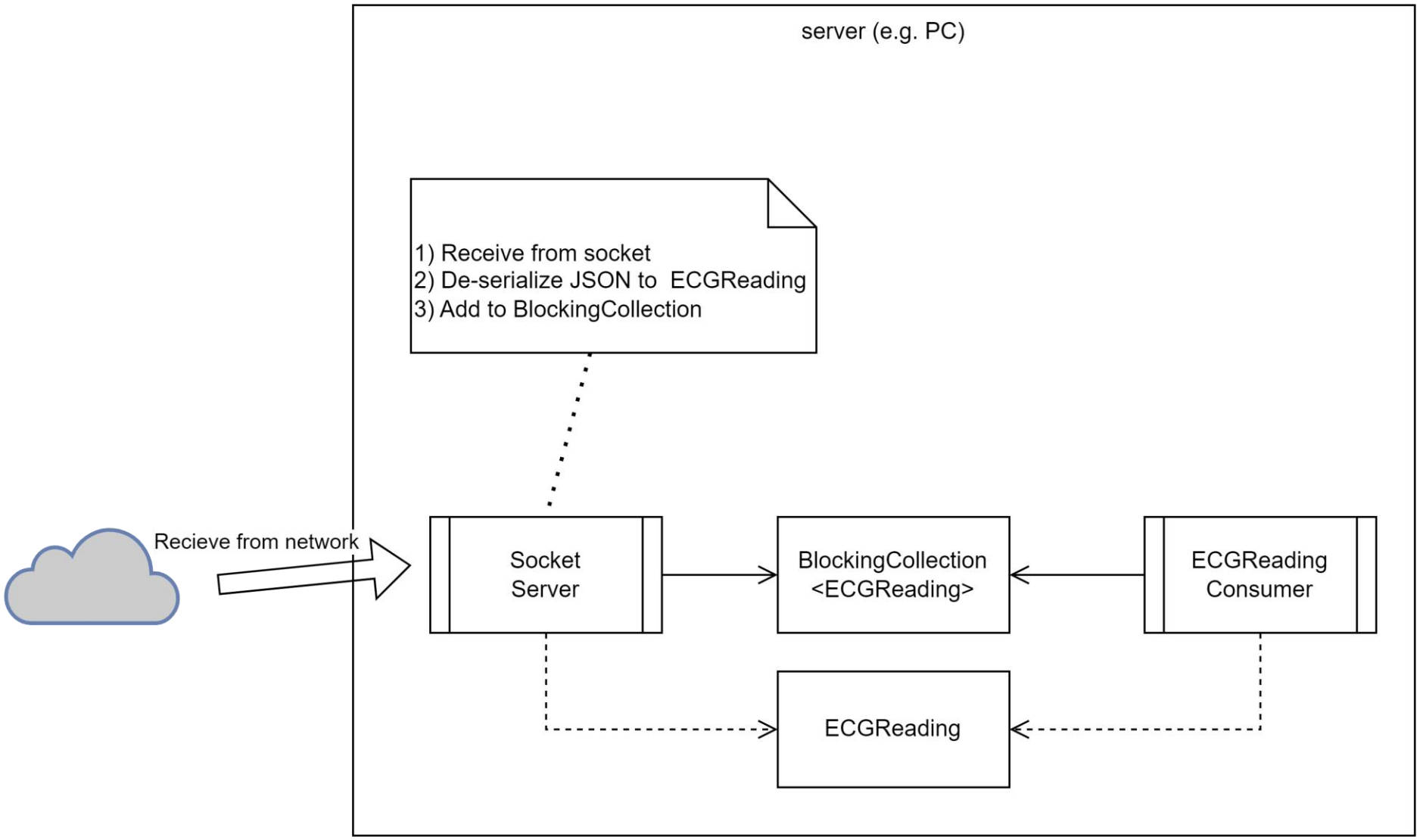
To this


Client (e.g. Raspberry Pi)
ECGSensor
IECGSensor
ECGReading Provider
BlockingCollection <ECGReading>
Socket Client
ECGReading
1) Take from BlockingCollection
2) Serialize ECGReading to JSON
3) Send JSON on socket
Send to network
Recieve from network
server (e.g. PC)
1) Receive from socket
2) De-serialize JSON to ECGReading
3) Add to BlockingCollection
Socket Server
BlockingCollection <ECGReading>
ECGReading Consumer
ECGReading

Client

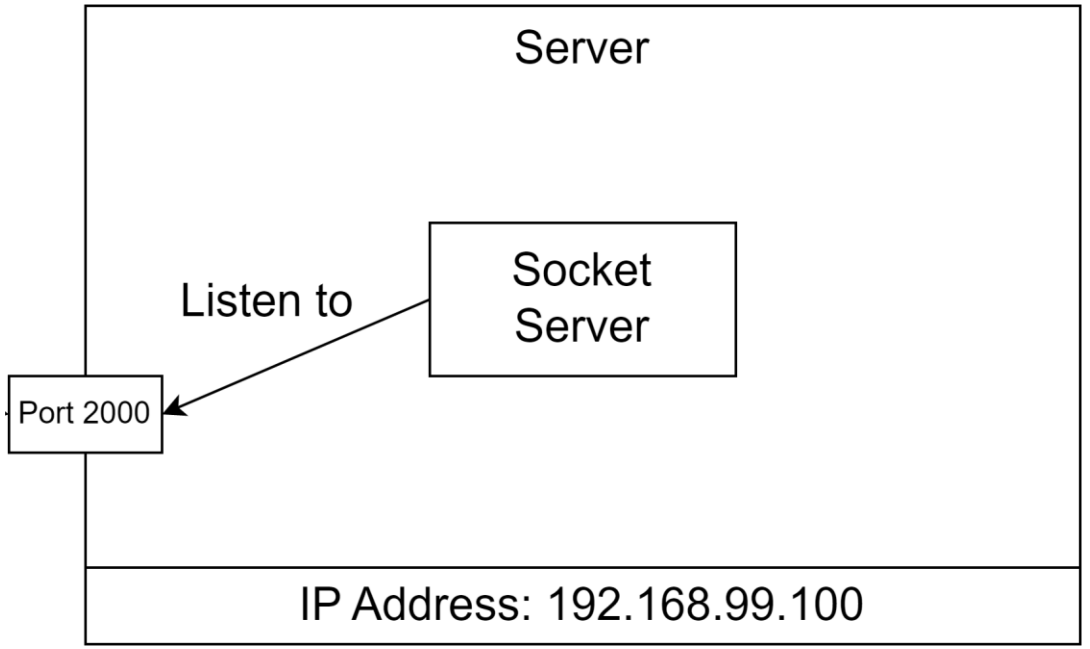

Client (e.g. Raspberry Pi)
ECGSensor
IECGSensor
1) Take from BlockingCollection
2) Serialize ECGReading to JSON
3) Send JSON on socket
ECGReading Provider
BlockingCollection <ECGReading>
Socket Client
Send to network
ECGReading

Server


server (e.g. PC)
1) Receive from socket
2) De-serialize JSON to ECGReading
3) Add to BlockingCollection
Recieve from network
Socket Server
BlockingCollection <ECGReading>
ECGReading Consumer
ECGReading

Socket communication

The server listens



The Server specifies where to listen:

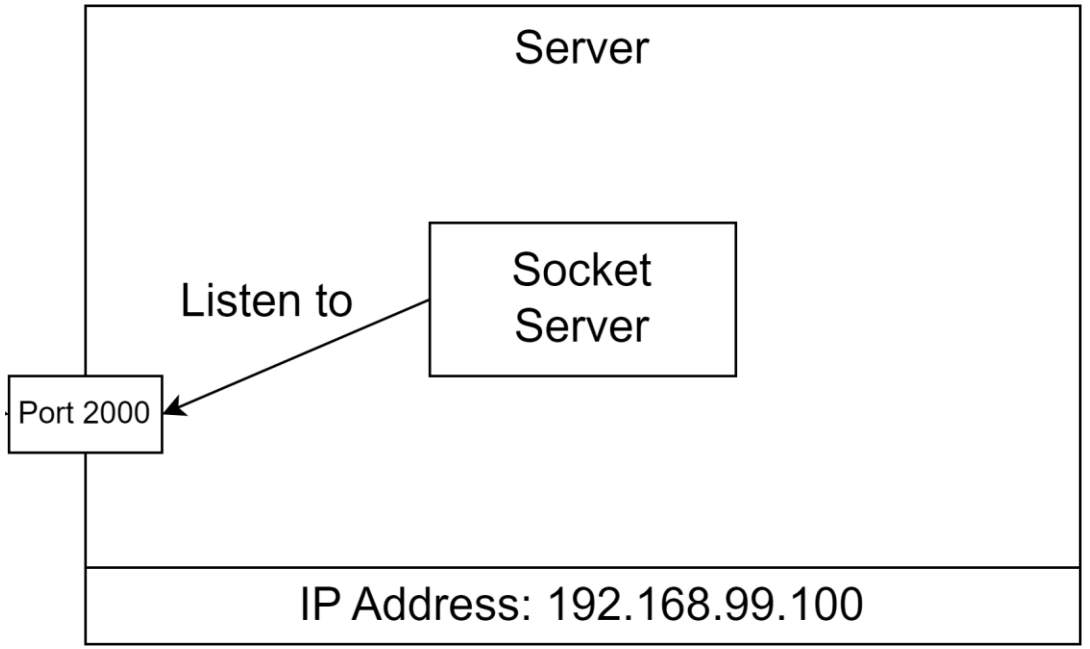
- IP Address
- Port number

The server listens

The server can have multiple network cards and more than one address.

IP Address to listen to can be:

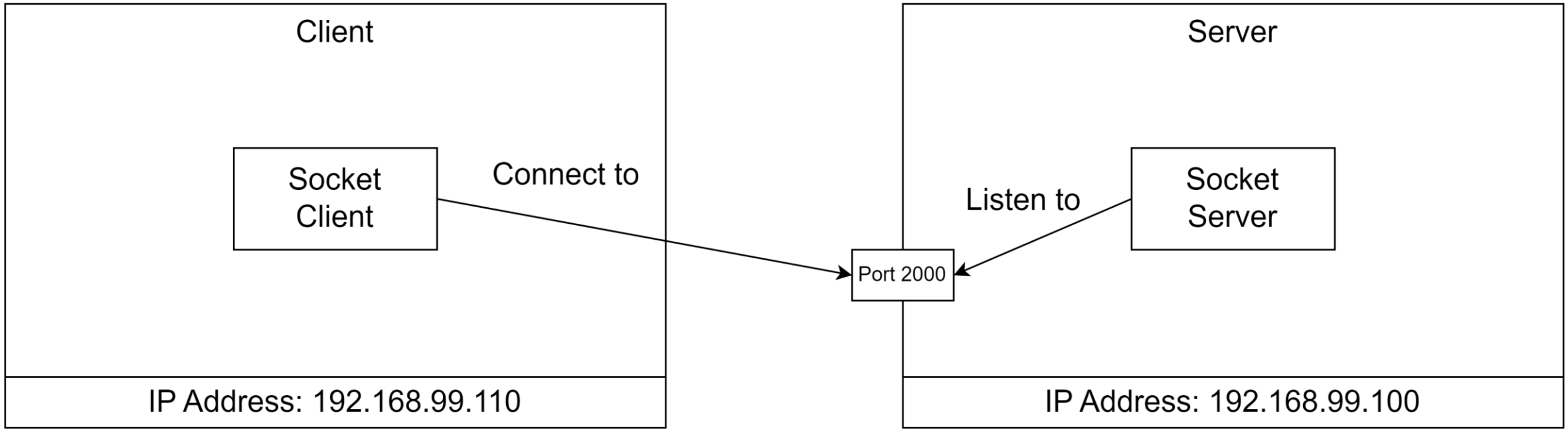
- **Any** <= all network interfaces
- **Loopback** <= 127.0.0.1
- A specific IP address



The Server specifies where to listen:

- IP Address
- Port number

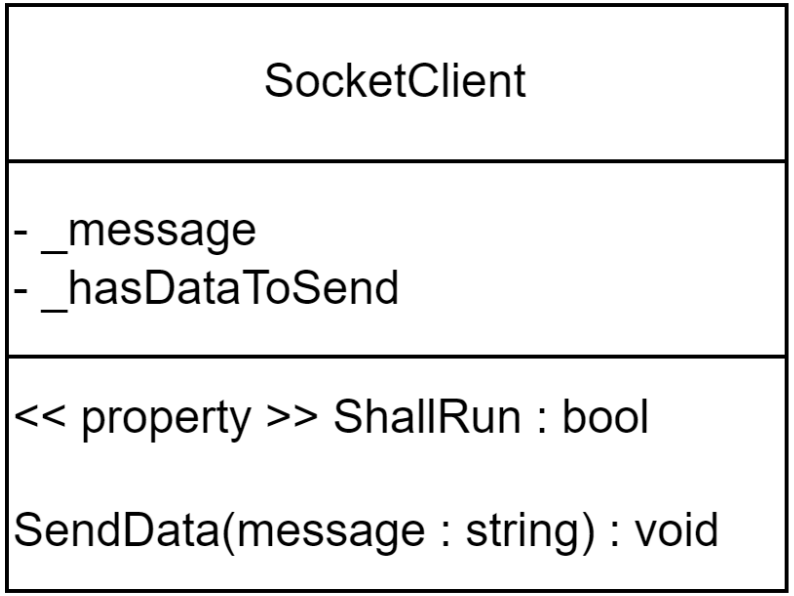
The client connects to the server

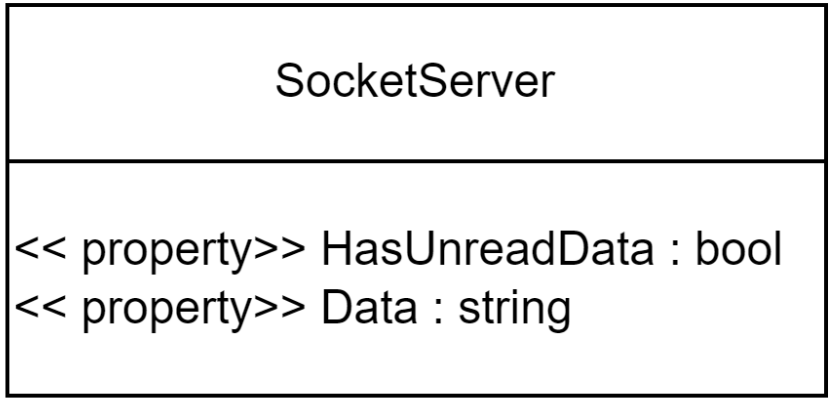


The client connects to a server socket:

- IP Address
- Port number

Using the socket communication from other threads






WARNING
WARNING
WARNING
WARNING
WARNING
WARNING
WARNING
WARNING
WARNING

Problems in this approach

There are some problems with the way we do this:

- What if SendData() is called more than once, before the data is sent on the socket?
- What if SendData() is called before _hasDataToSend is reset?
- What if a user of the SocketServer reads the Data property at the same time as data is received?

Problems in this approach

There are some problems with the way we do this:

- What if SendData() is called more than once, before the data is sent on the socket?
- What if SendData() is called before _hasDataToSend is reset?
- What if a user of the SocketServer reads the Data property at the same time as data is received?

Some data might be lost!

Problems in this approach

There are some problems with the way we do this:

- What if SendData() is called more than once, before the data is sent on the socket?
- What if SendData() is called before _hasDataToSend is reset?
- What if a user of the SocketServer reads the Data property at the same time as data is received?

We will fix this in the coming weeks ☺

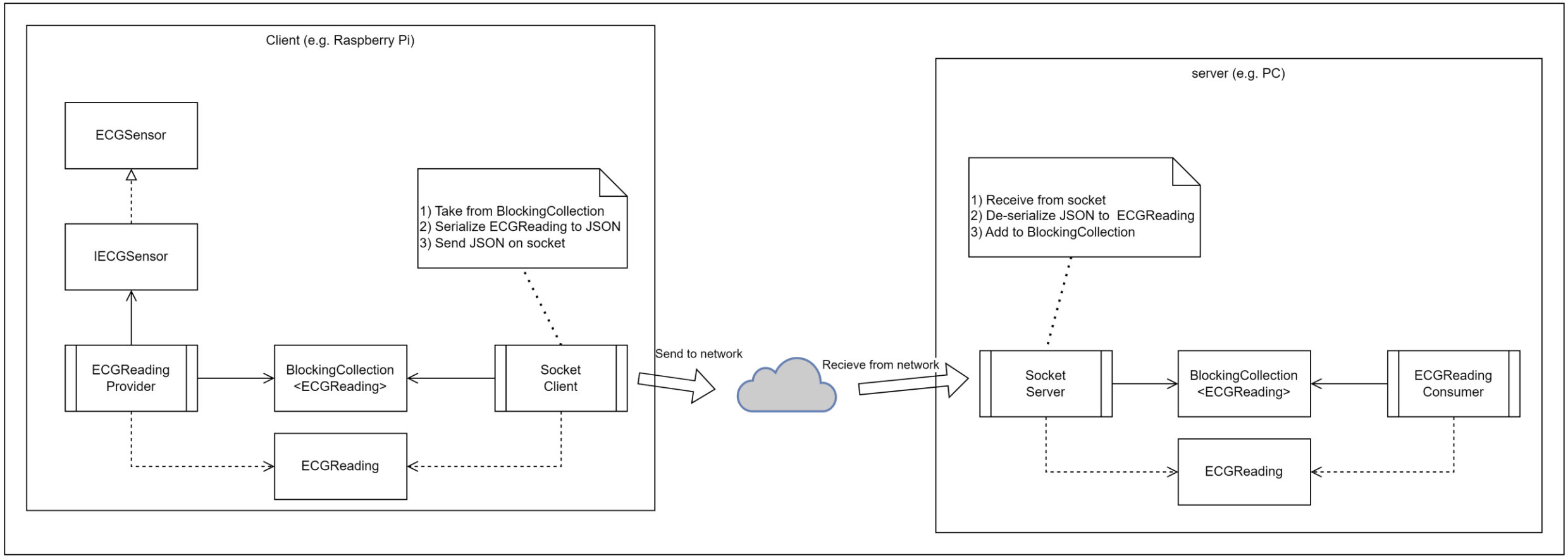
Problems in this approach

There are some problems with the way we do this:

- What if SendData() is called more than once, before the data is sent on the socket?
- What if SendData() is called before _hasDataToSend is reset?
- What if a user of the SocketServer reads the Data property at the same time as data is received?

We will fix this NOW!

Better approach



Sending objects from the client

```
internal class SocketClient
{
    private readonly BlockingCollection<ECGReading> _ecgReadings;

    public SocketClient(BlockingCollection<ECGReading> ecgReadings)
    {
        _ecgReadings = ecgReadings;
    }

    public void Run()
    {
        IPAddress ipAddress = IPAddress.Parse("127.0.0.1");
        IPEndPoint ipEndPoint = new IPEndPoint(ipAddress, 2000);

        using Socket client = new Socket(ipEndPoint.AddressFamily, SocketType.Stream, ProtocolType.Tcp);
        client.Connect(ipEndPoint);

        while (!_ecgReadings.IsCompleted)
        {
            try
            {
                ECGReading ecgReading = _ecgReadings.Take();

                string objectAsJson = JsonSerializer.Serialize(ecgReading);
                var messageBytes = Encoding.UTF8.GetBytes(objectAsJson);
                client.Send(messageBytes, SocketFlags.None);
                Console.WriteLine($"Socket client sent message: {objectAsJson}");
            }
            catch (InvalidOperationException)
            {
                // IOE means that Take() was called on a completed collection.
            }
        }
        client.Shutdown(SocketShutdown.Both);
    }
}
```

Sending objects from the client

```
internal class SocketClient
{
    private readonly BlockingCollection<ECGReading> _ecgReadings;

    public SocketClient(BlockingCollection<ECGReading> ecgReadings)
    {
        _ecgReadings = ecgReadings;
    }

    public void Run()
    {
        IPAddress ipAddress = IPAddress.Parse("127.0.0.1");
        IPEndPoint ipEndPoint = new IPEndPoint(ipAddress, 2000);

        using Socket client = new Socket(ipEndPoint.AddressFamily, SocketType.Stream,
ProtocolType.Tcp);
        client.Connect(ipEndPoint);
```

```
        client.Connect(ipEndPoint);

        while (!_ecgReadings.IsCompleted)
        {
            try
            {
                ECGReading ecgReading = _ecgReadings.Take();

                string objectAsJson = JsonSerializer.Serialize(ecgReading);
                var messageBytes = Encoding.UTF8.GetBytes(objectAsJson);
                client.Send(messageBytes, SocketFlags.None);
                Console.WriteLine($"Socket client sent message: {objectAsJson}");
            }
            catch (InvalidOperationException)
            {
                // IOE means that Take() was called on a completed collection.
            }
        }
        client.Shutdown(SocketShutdown.Both);
    }
}
```

Receiving objects on the server

```
internal class SocketServer
{
    private readonly BlockingCollection<ECGReading> _ecgReadings;

    public SocketServer(BlockingCollection<ECGReading> ecgReadings)
    {
        _ecgReadings = ecgReadings;
    }
    public void Run()
    {
        RunServer();
    }

    public void RunServer()
    {
        // listen to 'Any' which means all network addresses for this machine
        IPAddress ipAddress = IPAddress.Any;
        IPEndPoint ipEndPoint = new IPEndPoint(ipAddress, 2000);

        using Socket listener = new Socket(ipEndPoint.AddressFamily, SocketType.Stream, ProtocolType.Tcp);

        listener.Bind(ipEndPoint);

        Console.WriteLine($"Listening on: {ipAddress}");
        listener.Listen();

        var handler = listener.Accept();

        while (!ShallStop)
        {
            byte[] buffer = new byte[1024];
            int numberOfBytesReceived = handler.Receive(buffer, SocketFlags.None);
            if (numberOfBytesReceived > 0)
            {
                string receivedData = Encoding.UTF8.GetString(buffer, 0, numberOfBytesReceived);
                Console.WriteLine($"Server received:{receivedData}");

                try
                {
                    ECGReading? ecgReading = JsonSerializer.Deserialize<ECGReading>(receivedData);
                    if (ecgReading != null) _ecgReadings.Add(ecgReading);
                }
                catch (System.Text.Json.JsonException e)
                {
                    // log any parsing exceptions
                    Console.WriteLine(e);
                }
            }
        }
        listener.Close();
        _ecgReadings.CompleteAdding();
    }

    public bool ShallStop { get; set; } = false;
}
```

Receiving objects on the server

```
internal class SocketServer
{
    private readonly BlockingCollection<ECGReading> _ecgReadings;

    public SocketServer(BlockingCollection<ECGReading> ecgReadings)
    {
        _ecgReadings = ecgReadings;
    }
    public void Run()
    {
        RunServer();
    }

    public void RunServer()
    {
        // listen to 'Any' which means all network addresses for this machine
        IPAddress ipAddress = IPAddress.Any;
        IPEndPoint ipEndPoint = new IPEndPoint(ipAddress, 2000);

        using Socket listener = new Socket(ipEndPoint.AddressFamily, SocketType.Stream,
```

```
        using Socket listener = new Socket(ipEndPoint.AddressFamily, SocketType.Stream,
ProtocolType.Tcp);

        listener.Bind(ipEndPoint);

        Console.WriteLine($"Listening on: {ipAddress}");
        listener.Listen();

        var handler = listener.Accept();

        while (!ShallStop)
        {
            byte[] buffer = new byte[1024];
            int numberOfBytesReceived = handler.Receive(buffer, SocketFlags.None);
            if (numberOfBytesReceived > 0)
            {
                string receivedData = Encoding.UTF8.GetString(buffer, 0,
numberOfBytesReceived);
                Console.WriteLine($"Server received:{receivedData}");

                try
                {
                    ECGReading? ecgReading =
JsonSerializer.Deserialize<ECGReading>(receivedData);
```

```
            {
                string receivedData = Encoding.UTF8.GetString(buffer, 0,
numberOfBytesReceived);
                Console.WriteLine($"Server received:{receivedData}");

                try
                {
                    ECGReading? ecgReading =
JsonSerializer.Deserialize<ECGReading>(receivedData);
                    if (ecgReading != null) _ecgReadings.Add(ecgReading);
                }
                catch (System.Text.Json.JsonException e)
                {
                    // log any parsing exceptions
                    Console.WriteLine(e);
                }
            }
        }
        listener.Close();
        _ecgReadings.CompleteAdding();
    }

    public bool ShallStop { get; set; } = false;
}
```

Your turn

Solve the
” Exercise - ECG Network”

References and image sources

Warning tape: https://www.pngall.com/warning-sign-png/download/69428

AARHUS UNIVERSITY