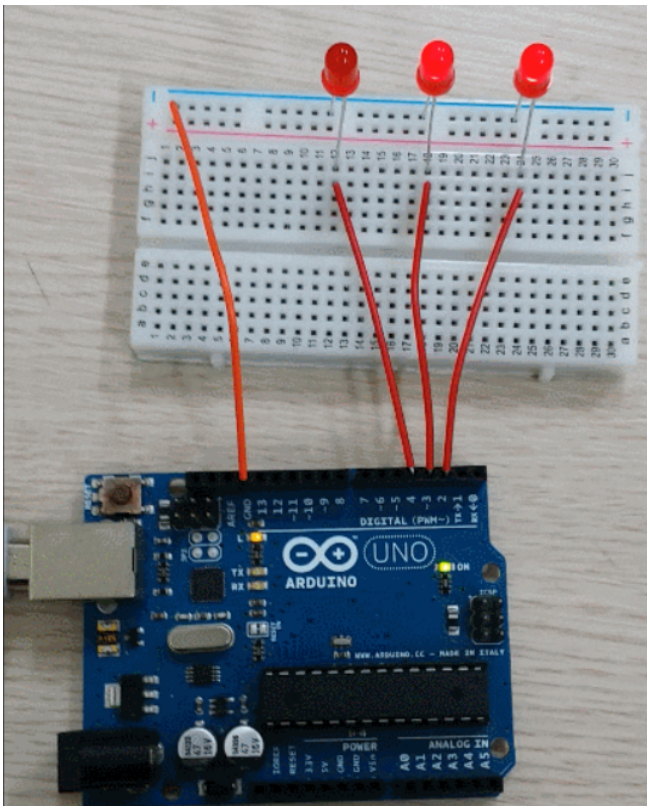
AkkaWithAduino

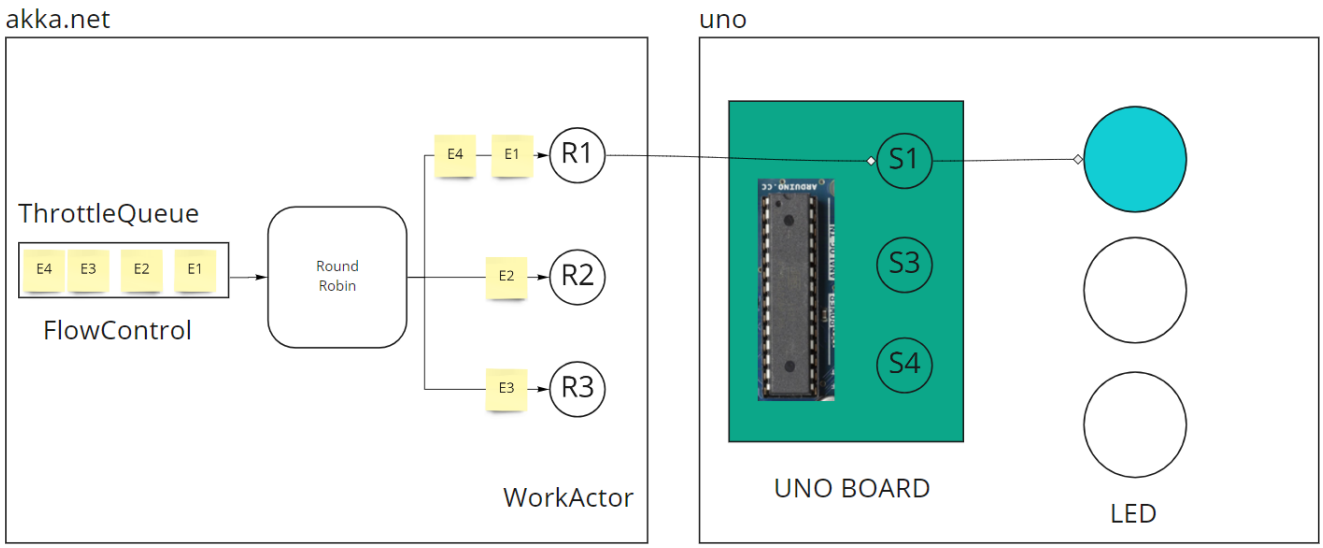
> Akka.net
>
> .
>
> : https://github.com/psmon/AkkaUno

,

(Microcontroller) .



, LED .



- ThrottleQueue : AkkaStream .
- RoundRobin() : , .
- WorkActor : , UnoActor LED . (n):LED(n) 1:1. ( )
- UnoActor : . .

, LED / .

C++ , Akka.net .

: https://github.com/psmon/AkkaUno/blob/master/UnoAkkaApp/Module/AduinoApp.ino

.

```
using Akka.Actor;
using Akka.Event;
using System.IO.Ports;

namespace UnoAkkaApp.Actors
{
    public class UnoActor : ReceiveActor
    {
        private readonly ILoggingAdapter logger = Context.GetLogger();

        //  :   SerialPortStream (  )
        private readonly SerialPort arduSerialPort;

        public UnoActor()
        {
            arduSerialPort = new SerialPort();
            arduSerialPort.PortName = "COM3";   //
            arduSerialPort.BaudRate = 9600;     //
            arduSerialPort.Open();              //

            ReceiveAsync<string>(async command =>
            {
                logger.Info($"Receive : {command}");
                arduSerialPort.Write(command);
            });
        }
    }
}
```

nodeNo , UnoActor LED .

```
using Akka.Actor;
using Akka.Event;
using AkkaDotModule.Models;

namespace UnoAkkaApp.Actors
{
    public class WorkActor : ReceiveActor
    {
        private readonly ILoggingAdapter logger = Context.GetLogger();

        private readonly IActorRef _unoActor;

        public WorkActor(IActorRef unoActor,int nodeNum)
        {
            _unoActor = unoActor;

            int _nodeNo = nodeNum;

            ReceiveAsync<BatchData>(async command =>
            {
                logger.Info($"Receive Data..");
                unoActor.Tell(nodeNum.ToString());
            });
        }
    }
}
```

-

```
            // start ActorSystem
            AkkaSystem = ActorSystem.Create("AkkaSystem");

            var unoActor = AkkaSystem.ActorOf(Props.Create(() => new UnoActor()), "unoActor");

            List<string> workActors = new List<string>();

            for (int i = 0; i < 9; i++)
            {
                string actorName = $"workActor{i + 1}";
                var curWorkActor = AkkaSystem.ActorOf(Props.Create(() => new WorkActor(unoActor, i + 1)),
actorName);
                curWorkActor.Tell(new BatchData()); // Led  Test(  LedOn)
                workActors.Add($"/user/{actorName}");
            }

            //  : https://getakka.net/articles/actors/routers.html
            var router = AkkaSystem.ActorOf(Props.Empty.WithRouter(new RoundRobinGroup(workActors)),
"roundRobinGroup");

            //  Work :
            int timeSec = 1;
            int elemntPerSec = 2;
            var throttleWork = AkkaSystem.ActorOf(Props.Create(() => new ThrottleWork(elemntPerSec, timeSec)),
"throttleWork");
            //
            throttleWork.Tell(new SetTarget(router));

            // 100
            List<object> batchDatas = new List<object>();
            for (int i = 0; i < 100; i++)
            {
                batchDatas.Add(new BatchData() { Data="SomeData" });
            }
            BatchList batchList = new BatchList(batchDatas.ToImmutableList());

            //100 ()
            throttleWork.Tell(batchList);
```

- UnoActor : .
- ThrottleWork : . .
- WorkActor : N . Led . .
- RoundRobinGroup : N WorkActor .

-

.

, .

- : ()
- :
- :

AKKA , AKKA

.

```
var router = AkkaSystem.ActorOf(Props.Empty.WithRouter(new RandomGroup(workActors)), "roundRobinGroup");
==>
var router = AkkaSystem.ActorOf(Props.Empty.WithRouter(new RandomGroup(workActors)), "randomGroup");
```

AKKA ,

- / sleep . .
- / , . .

-

AKKA .

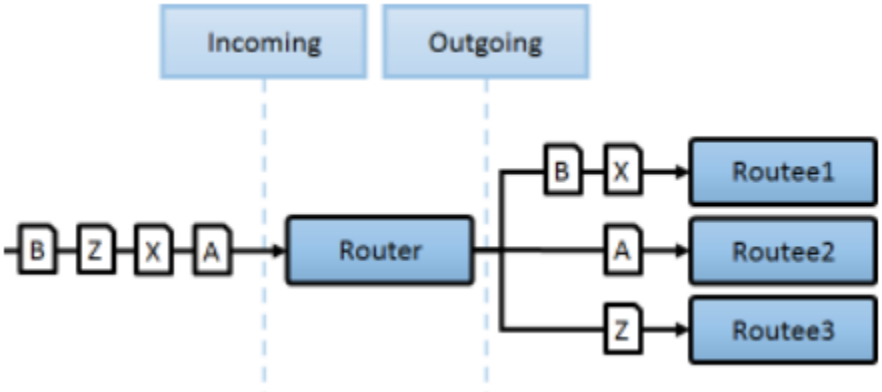
, .

.

ConsistentHashing

: , .

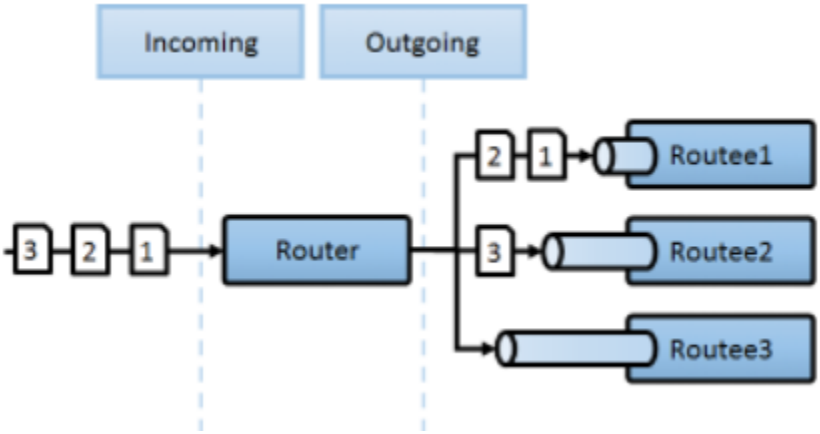
, , X .



SmallestMailbox

Task , Task .

.



TailChopping

.

, GC GC .

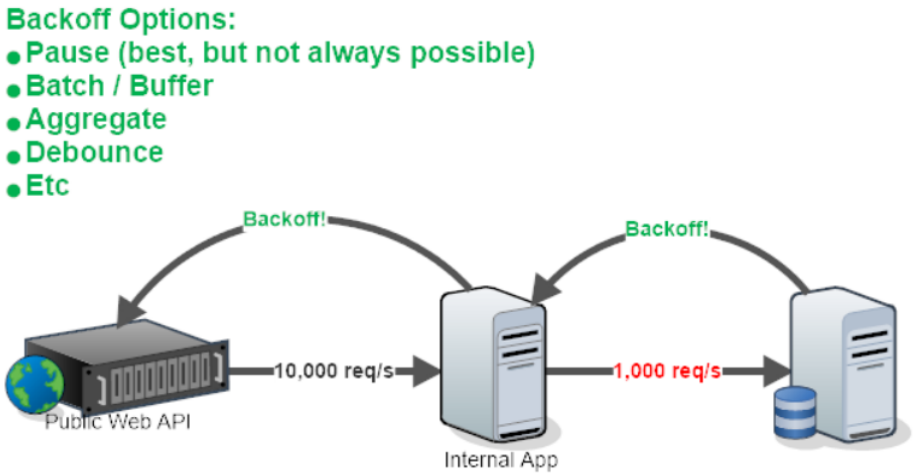
,

.

: https://getakka.net/articles/actors/routers.html

- AKKA STREAM

Akka , Akka Stream .



: https://petabridge.com/blog/why-akkadotnet-streams/

, ThrottleWork .

```
using System;
using Akka;
using Akka.Actor;
using Akka.Event;
using Akka.Streams;
using Akka.Streams.Dsl;
using AkkaDotModule.Models;

namespace UnoAkkaApp.Actors
{
    public class ThrottleWork : ReceiveActor
    {
        private readonly ILoggingAdapter logger = Context.GetLogger();

        private IActorRef consumer;

        private int countPerSec;

        public ThrottleWork(int element, int maxBust)
        {
            countPerSec = element;

            ReceiveAsync<SetTarget>(async target =>
            {
                consumer = target.Ref;
            });

            ReceiveAsync<int>(async count =>
            {
                countPerSec = count;

                logger.Info($"ThrottleWork Spped:{countPerSec}");
            });

            ReceiveAsync<BatchList>(async batchMessage =>
            {
                int Count = batchMessage.Obj.Count;
                Source<object, NotUsed> source = Source.From(batchMessage.Obj);

                using (var materializer = Context.Materializer())
                {
                    var factorials = source;
                    factorials
                         .Throttle(countPerSec, TimeSpan.FromSeconds(1), maxBust, ThrottleMode.Shaping)
                         .RunForeach(obj => {
                             var nowstr = DateTime.Now.ToString("mm:ss");
                             if (obj is BatchData batchData)
                             {
                                 if (consumer != null) consumer.Tell(batchData);
                             }
                         }, materializer)
                         .Wait();
                }
            });
        }
    }
}
```

- **Stream**

akkauni-stream.mp4

, (USB)

/ OS -

( ) .