Extensions to the Concurrent Communications Library

Kenneth Skovhede
Niels Bohr Institute
University of Copenhagen

CPA 2016 - Copenhagen

Extensions to the Concurrent Communications Library

Kenneth Skovhede
Niels Bohr Institute
University of Copenhagen

CPA 2016 - Copenhagen

Background

Overview

A CSP inspired library in/for C# (and other CIL languages)

- Utilizes await/async
- Builds FSMs from sequential code
- Low memory overhead pr. process
- Fewer context switches

```
while(true)
    out.Write(
      in.Read());
```

```
while(true)
    await out.WriteAsync(
      await in.ReadAsync());
```

Features

Only one channel type, generics for type-safety

Named and anonymous channels

Alternation with priority, random, and fair select

Sequence guarantee through request lists

Two-phase commit used to guarantee atomic selection external to the channel

Alternation with input or output and timeouts

Skip guards supported through immediate timeouts

Minimalistic channel core, features outside

Overview

A CSP inspired library in/for C# (and other CIL languages)

- Utilizes await/async
- Builds FSMs from sequential code
- Low memory overhead pr. process
- Fewer context switches

```
while(true)
    out.Write(
      in.Read());
```

```
while(true)
    await out.WriteAsync(
       await in.ReadAsync());
```

Features

Named and anonymous channels

Only one channel type,
generics for type-safety

Sequence guarantee through request lists

Alternation with priority,
random, and fair select

Two-phase commit used to guarantee
atomic selection external to the channel

Alternation with input or output and timeouts

Skip guards supported through immediate timeouts

Minimalistic channel core, features outside

Productivity enhancements

Mixing read and write requests

Overflowing a channel

```
var warn = ChannelManager.GetChannel<int>("warn");
var report = ChannelManager.GetChannel<int>("report");

while(true) {
  var current = ReadHardwareSensor ();
  if (current > 10)
    warn.Write(current);
  report.Write(current);
}

var warn = ChannelManager.GetChannel<int>("warn");
var report = ChannelManager.GetChannel<int>("report",
  new BoundedChannelOptions(1)
  {
    FullMode = BoundedChannelFullMode.DropOldest
  });

while(true) {
  var current = ReadHardwareSensor ();
  if (current > 10)
    warn.Write(current);
  report.WriteAsync(current);
}
```

- Enables Actor-like paradigms
- The WriteAsync() method returns a value that can be checked for failure

Channel scopes

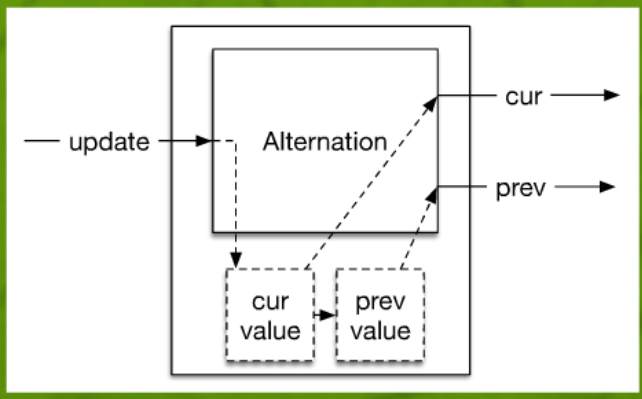
```
var external = ChannelManager.GetChannel<int>("out");
using(new IsolatedScope())
    await Task.WhenAll(
        Task.Run(async () => { // Process 1
            var internal =
                ChannelManager.GetChannel<int>("out");

            while(true)
                await internal.WriteAsync(42);
        }),

        Task.Run(async () => { // Process 2
            var internal =
                ChannelManager.GetChannel<int>("out");

            while(true)
                await external.WriteAsync(
                    await internal.ReadAsync()
                );
        })
    );
```

Wiring channels automatically

Leave and Join

Mixing read and write requests

```
var cur = 0, prev = 0;
var updatechan = ChannelManager.GetChannel <string >("update");
var curchan = ChannelManager.GetChannel<int>("cur");
var prevchan = ChannelManager.GetChannel<int>("prev");

while(true) {
  var res = await MultiChannelAccess.ReadOrWriteAnyAsync(
    MultisetRequest.Read(updatechan),
    MultisetRequest.Write(cur, curchan),
    MultisetRequest.Write(prev, prevchan),
  );

  if (res.IsRead) {
    prev = cur;
    cur = int.Parse((string)res.Value);
  }
}
```



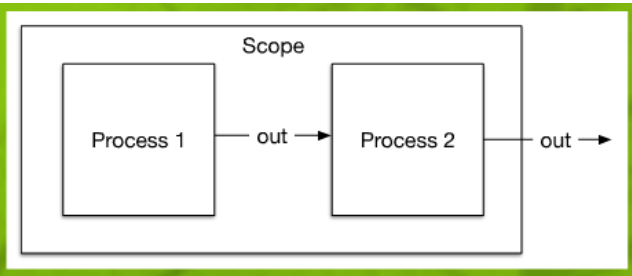
Overflowing a channel

```
var warn = ChannelManager.GetChannel<int>("warn");
var report = ChannelManager.GetChannel<int>("report");

while(true) {
  var current = ReadHardwareSensor ();
  if (current > 10)
    warn.Write(current);
  report.Write(current);
}
```

```
var warn = ChannelManager.GetChannel<int>("warn");
var report = ChannelManager.GetChannel<int>("report",
    maxPendingWriters: 1,
    pendingWritersOverflowStrategy:
        QueueOverflowStrategy.FIFO
);

while(true) {
  var current = ReadHardwareSensor ();
  if (current > 10)
    warn.Write(current);
  report.WriteNoWait(current);
}
```

- Enables Actor-like paradigme
- The WriteAsync() method returns a value that can be checked for failure

Channel scopes




```
var external = ChannelManager.GetChannel<int>("out");
using(new IsolatedScope())
    await Task.WhenAll(
        Task.Run(async () => { // Process 1
            var internal =
                ChannelManager.GetChannel<int>("out");

            while(true)
                await internal.WriteAsync(42);
        }),

        Task.Run(async () => { // Process 2
            var internal =
                ChannelManager.GetChannel<int>("out");

            while(true)
                await external.WriteAsync(
                    await internal.ReadAsync()
                );
        })
    );
```

Wiring channels automatically

Prototein Network in PyCSP

```
feeder = One2AnyChannel()
collector = Any2OneChannel()
done = Any2OneChannel()

Parallel(
    Process(producer, protein, map, place, feeder.write),
    Process(worker, feeder.read, collector.write, done.write),
    Process(worker, feeder.read, collector.write, done.write),
    Process(worker, feeder.read, collector.write, done.write),
    Process(worker, feeder.read, collector.write, done.write),
    Process(worker, feeder.read, collector.write, done.write),
    Process(barrier, 5, done.read, collector.write),
    Process(consumer , collector.read))
```

Prototein Network in CoCoL

```
var feeder = ChannelManager.GetChannel<int>();
var collector = ChannelManager.GetChannel<int>();
var done = ChannelManager.GetChannel<bool>();

Task.WhenAll(
    Producer(protein, map, place, feeder.AsWrite()),
    Worker(feeder.AsRead(), collector.AsWrite(), done.AsWrite()),
    Worker(feeder.AsRead(), collector.AsWrite(), done.AsWrite()),
    Worker(feeder.AsRead(), collector.AsWrite(), done.AsWrite()),
    Worker(feeder.AsRead(), collector.AsWrite(), done.AsWrite()),
    Worker(feeder.AsRead(), collector.AsWrite(), done.AsWrite()),
    Barrier(5, done.AsRead(), collector.AsWrite(), done.AsWrite()),
    Consumer(collector.AsRead()));
```

Prototein Network with Wiring

```
public class Worker : ProcessHelper {
    [ChannelName("feeder")]
    private IReadChannel<int> feeder;

    [ChannelName("collector")]
    private IWriteChannel<int> collector;

    [ChannelName("done")]
    private IWriteChannel<bool> done;

    ... code omitted ...
}

Task.WhenAll(
    new Producer(protein, map, place),
    new Worker(), new Worker(), new Worker(),
    new Worker(), new Worker(),
    new Barrier(5),
    new Consumer());
```

Prototein Network in PyCSP

```
feeder = One2AnyChannel()
collector = Any2OneChannel()
done = Any2OneChannel()

Parallel(
    Process(producer, protein, map, place, feeder.write),
    Process(worker, feeder.read, collector.write, done.write),
    Process(worker, feeder.read, collector.write, done.write),
    Process(worker, feeder.read, collector.write, done.write),
    Process(worker, feeder.read, collector.write, done.write),
    Process(worker, feeder.read, collector.write, done.write),
    Process(barrier, 5, done.read, collector.write),
    Process(consumer , collector.read))
```

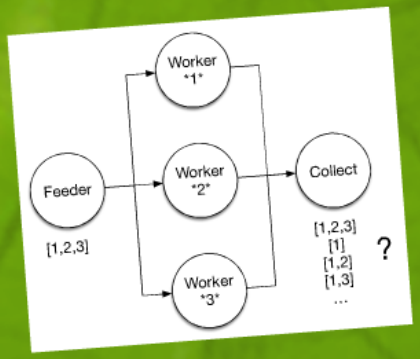
Prototein Network in CoCoL

```
var feeder = ChannelManager.GetChannel<int>();
var collector = ChannelManager.GetChannel<int>();
var done = ChannelManager.GetChannel<bool>();

Task.WhenAll(
    Producer(protein, map, place, feeder.AsWrite()),
    Worker(feeder.AsRead(), collector.AsWrite(), done.AsWrite()),
    Worker(feeder.AsRead(), collector.AsWrite(), done.AsWrite()),
    Worker(feeder.AsRead(), collector.AsWrite(), done.AsWrite()),
    Worker(feeder.AsRead(), collector.AsWrite(), done.AsWrite()),
    Worker(feeder.AsRead(), collector.AsWrite(), done.AsWrite()),
    Barrier(5, done.AsRead(), collector.AsWrite()),
    Consumer(collector.AsRead()));
```

Prototein Network with Wiring

```
public class Worker : ProcessHelper {
    [ChannelName("feeder")]
    private IReadChannel<int> feeder;

    [ChannelName("collector")]
    private IWriteChannel<int> collector;

    [ChannelName("done")]
    private IWriteChannel<bool> done;

    ... code omitted ...
}

Task.WhenAll(
    new Producer(protein, map, place),
    new Worker(), new Worker(), new Worker(),
    new Worker(), new Worker(),
    new Barrier(5),
    new Consumer());
```

Leave and Join



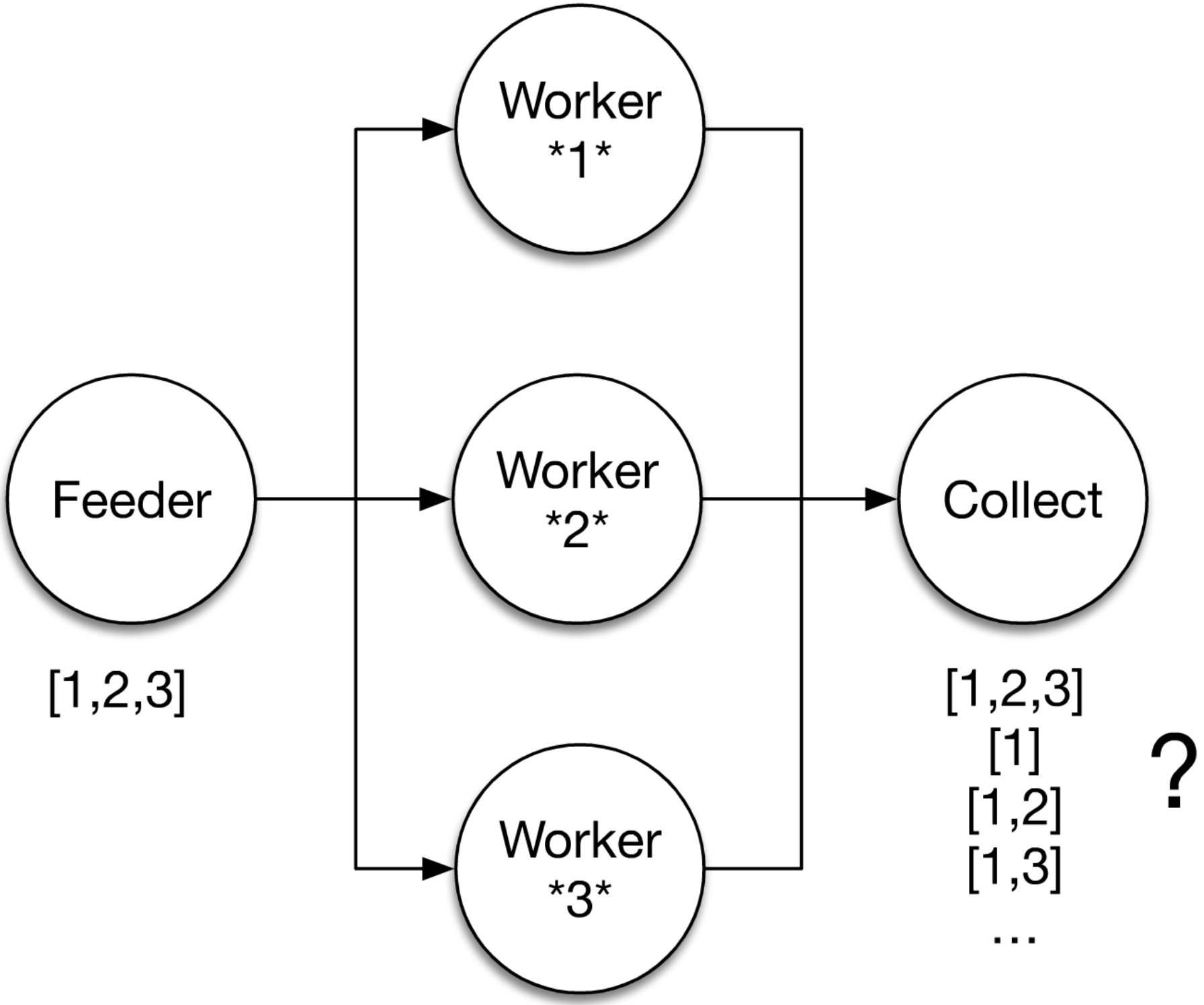
Solution:

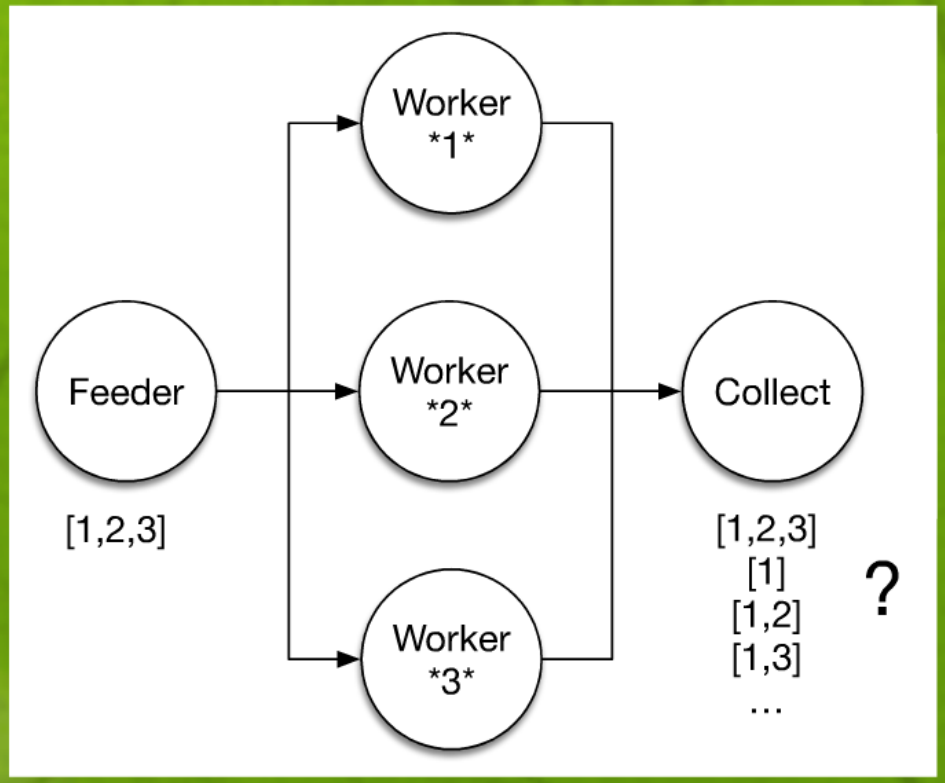
- Readers and writers "join" the channel
- Keep a counter for number of readers and number of writers
- When poisoned, "leave" the channel
- When either counter reaches zero poison the channel

*Same as in PyCSP Revisited, but differs in handling poison/retire

Adding Lambda Closures

```
private static Task Worker () {
    return AutomationExtensions.RunTask(
        new {
            feed = ChannelMarker.ForRead<int>("feeder"),
            coll = ChannelMarker.ForWrite<int>("collector")
        },

        async (self) => {
            while (true) {
                var data = await self.feed.ReadAsync();
                ... code omitted ...
                await self.coll.WriteAsync(result);
            }
        }
    );
}

Task.WhenAll(
    Producer(protein, map, place),
    Worker(), Worker(), Worker(), Worker(), Worker(),
    Consumer());
```


Worker
1
Feeder
Worker
2
Collect
[1,2,3]
Worker
3
[1,2,3]
[1]
[1,2]
[1,3]
...
?



Solution:

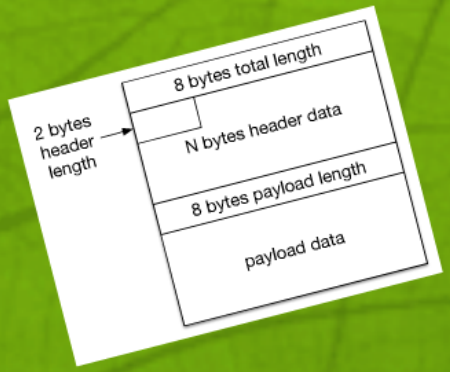
- Readers and writers "join" the channel
- Keep a counter for number of readers and number of writers
- When poisoned, "leave" the channel
- When either counter reaches zero poison the channel

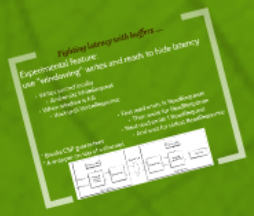
*Same as in PyCSP Revisited, but differs in handling poison/retire

Adding Lambda Closures

```
private static Task Worker () {
    return AutomationExtensions.RunTask(
        new {
            feed = ChannelMarker.ForRead<int>("feeder"),
            coll = ChannelMarker.ForWrite<int>("collector")
        },

        async (self) => {
            while (true) {
                var data = await self.feed.ReadAsync();
                ... code omitted ...

                await self.coll.WriteAsync(result);
            }
        }
    );
}

Task.WhenAll(
    Producer(protein, map, place),
    Worker(), Worker(), Worker(), Worker(), Worker(),
    Consumer());
```

Network support





Design ideas

Drop-in replacement channel,
user should not take special care

3-tier setup:

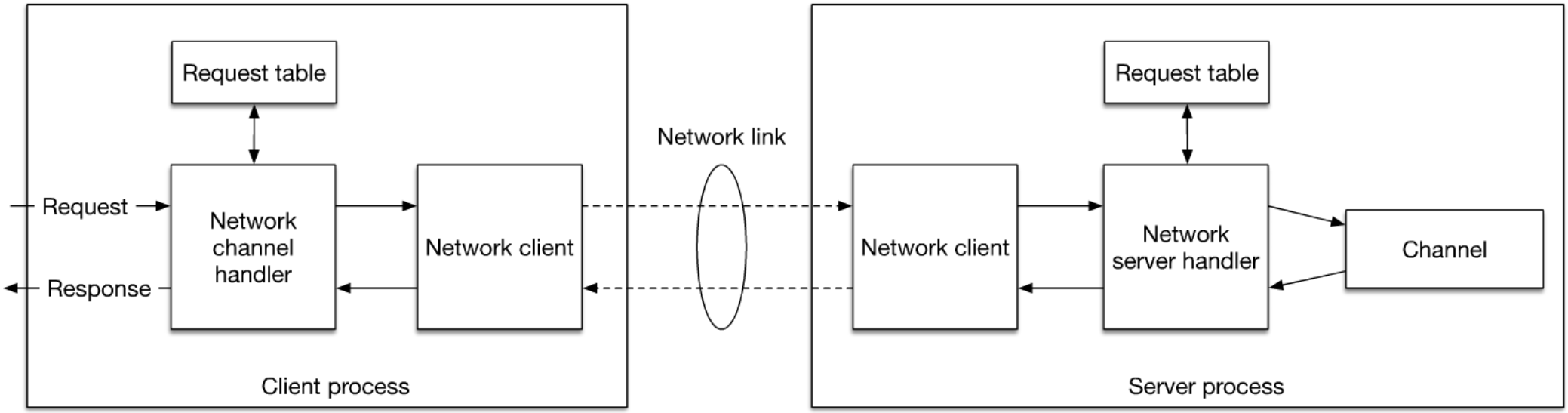
- Name-/discovery-servers
- Channel hosting servers
- Clients

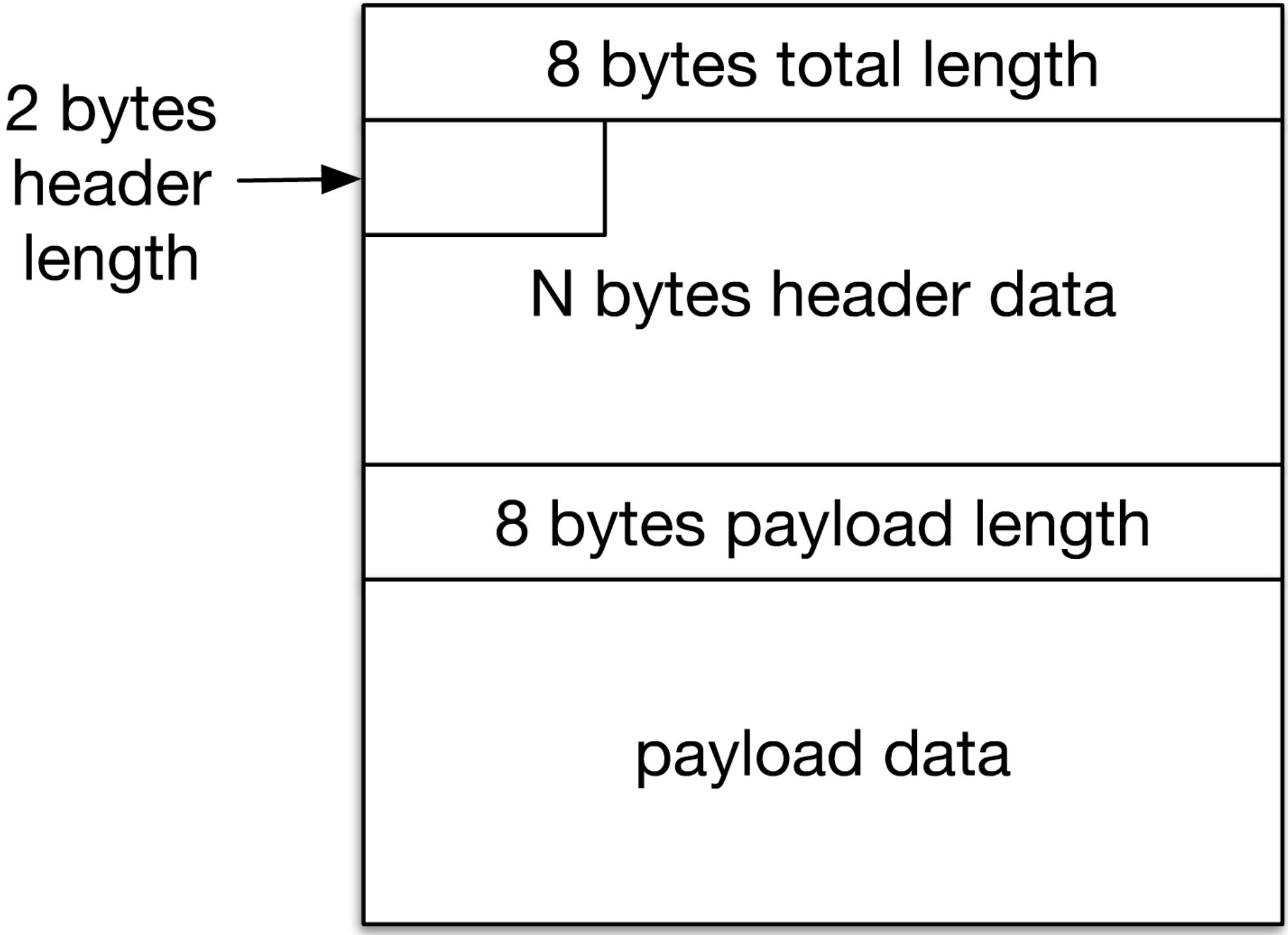
* Currently has name+hosting in one

Channel hosting server should only relay to/from
network to a normal hosted channel:
no special logic for a network channel

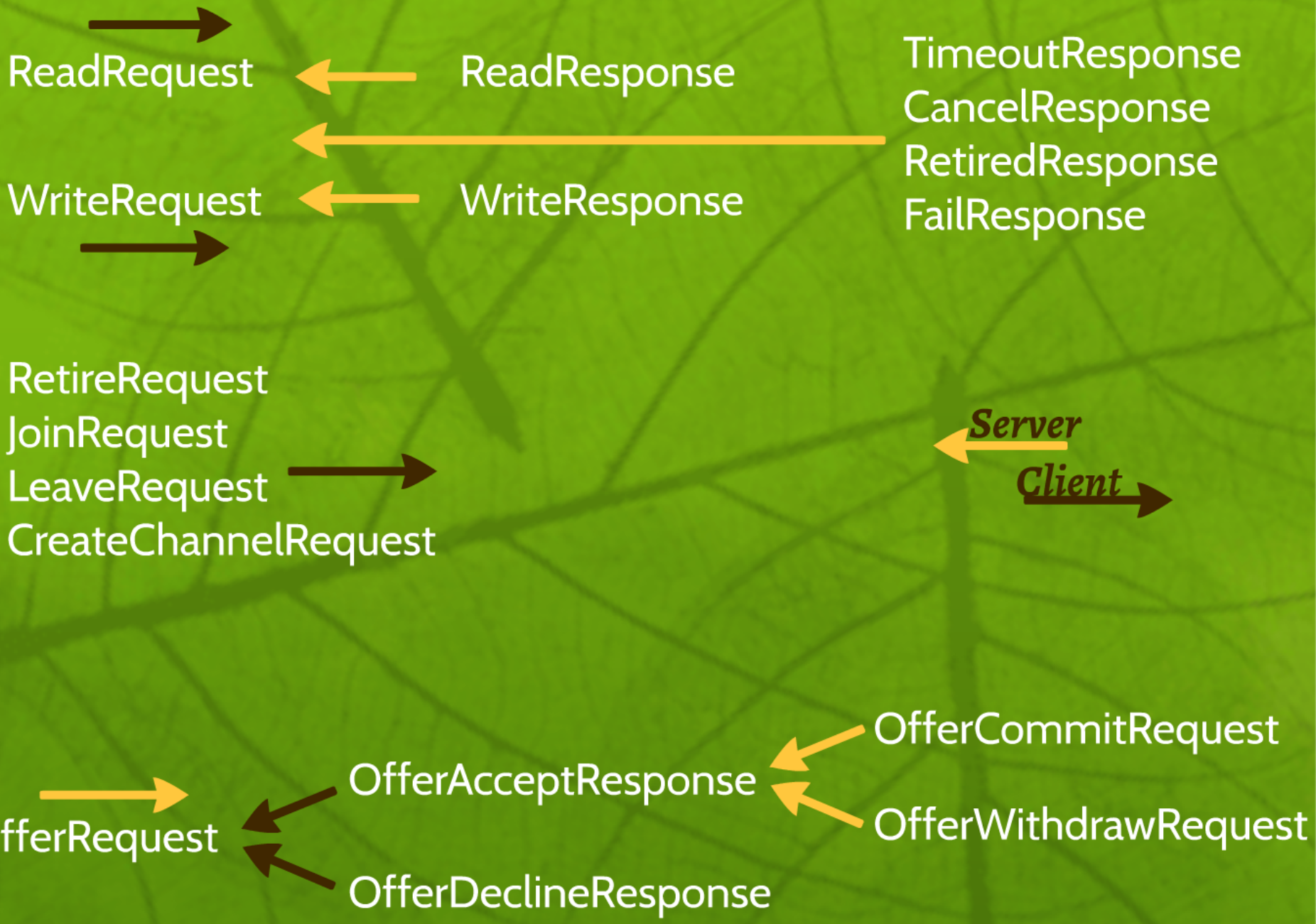
Timeout handled on hosting server

Two-phase commit support,
enables ALT'ing with any type channel-mix


Request table
Request
Response
Network channel handler
Network client
Network link
Request table
Network client
Network server handler
Channel
Client process
Server process


8 bytes total length
2 bytes
header
length
N bytes header data
8 bytes payload length
payload data

Message types



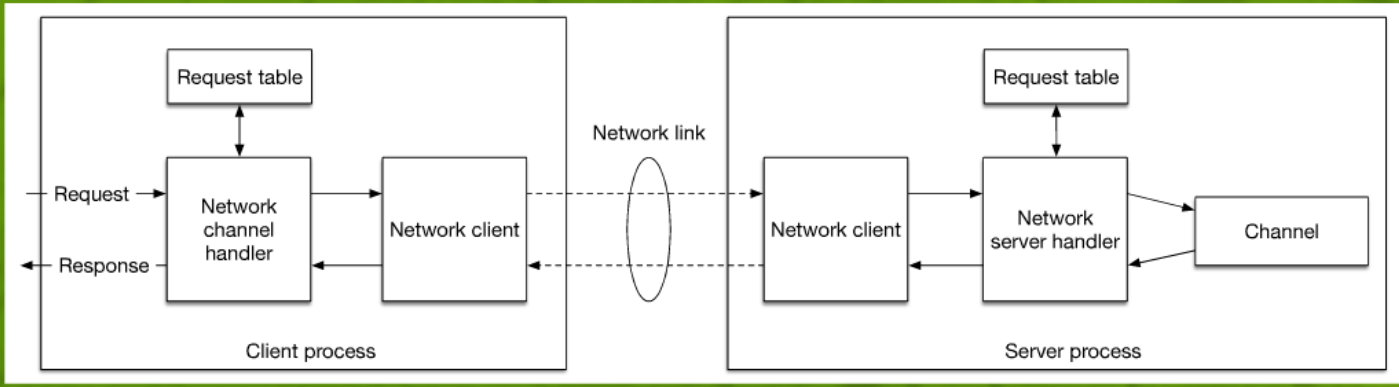
Fighting latency with buffers ....

Experimental feature:
use "windowing" writes and reads to hide latency

- Writes succed locally
  - And emits WriteRequest
- When window is full
  - Wait until WriteResponse

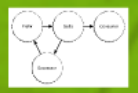
- First read emits N ReadRequests
  - Then waits for ReadResponse
- Next read emits 1 ReadRequest
  - And wait for oldest ReadResponse

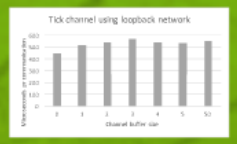
* Breaks CSP guarantees
* A wrapper on top of a channel

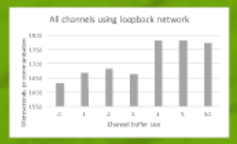


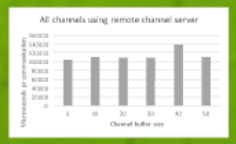
Experiments

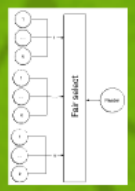
CommsTime



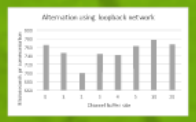


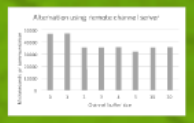




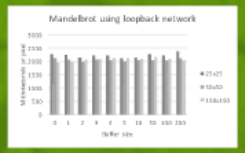


StressedAlt

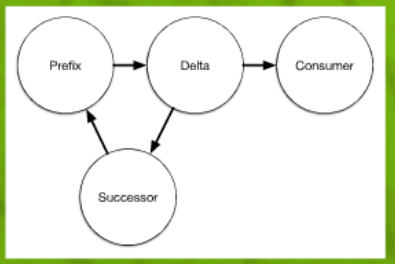


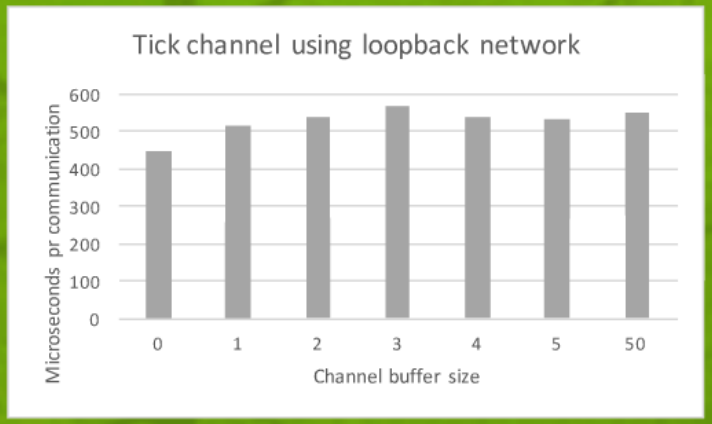


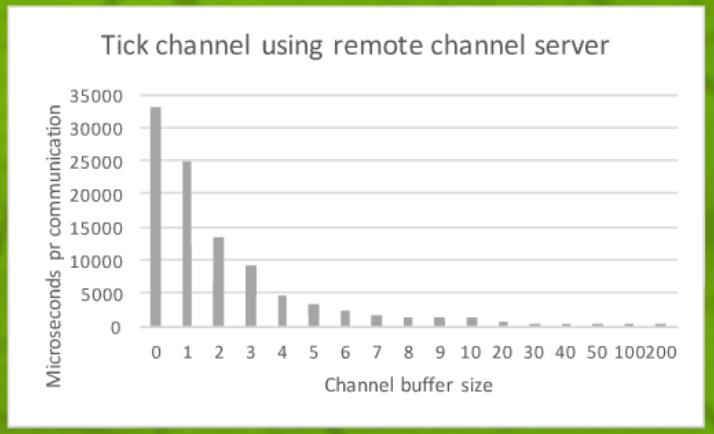
Mandelbrot

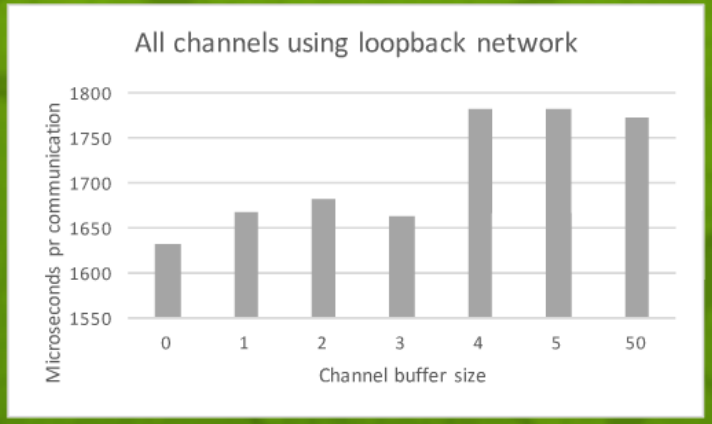


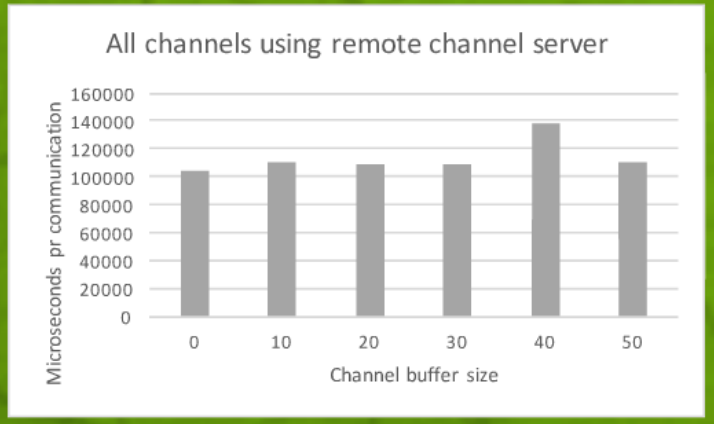
CommsTime


Prefix
Delta
Consumer
Successor

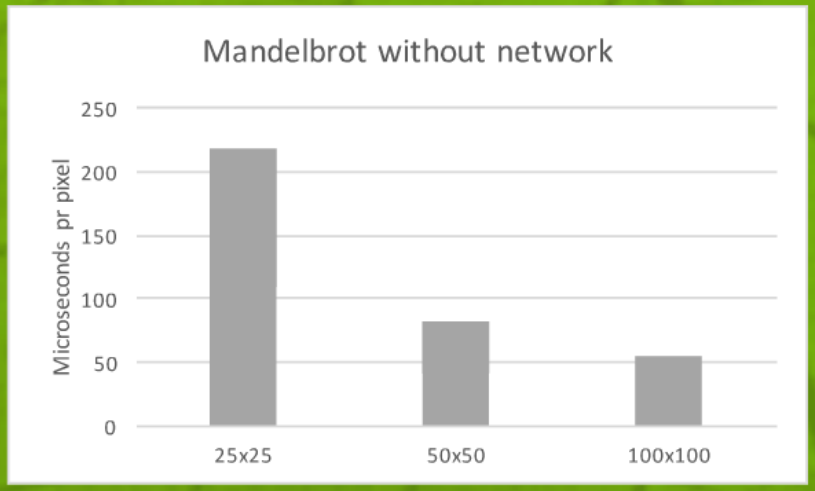

Tick channel using loopback network
Microseconds pr communication
600
500
400
300
200
100
0
0
1
2
3
4
5
50
Channel buffer size

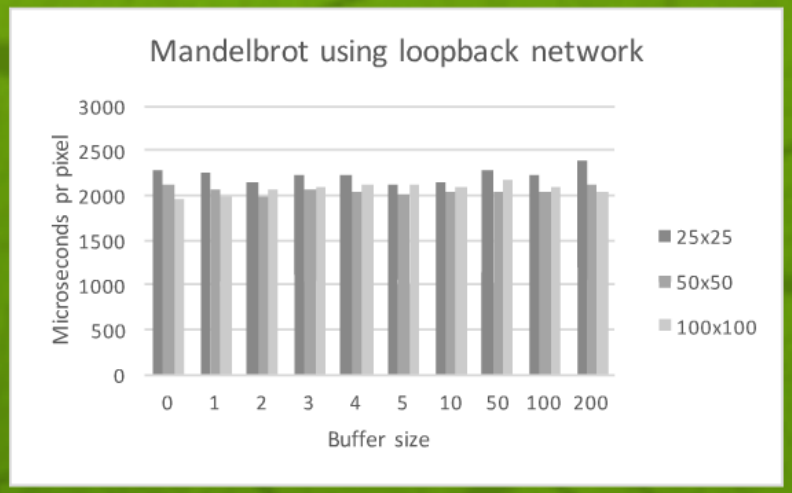

Tick channel using remote channel server
Microseconds pr communication
35000
30000
25000
20000
15000
10000
5000
0
0 1 2 3 4 5 6 7 8 9 10 20 30 40 50 100200
Channel buffer size

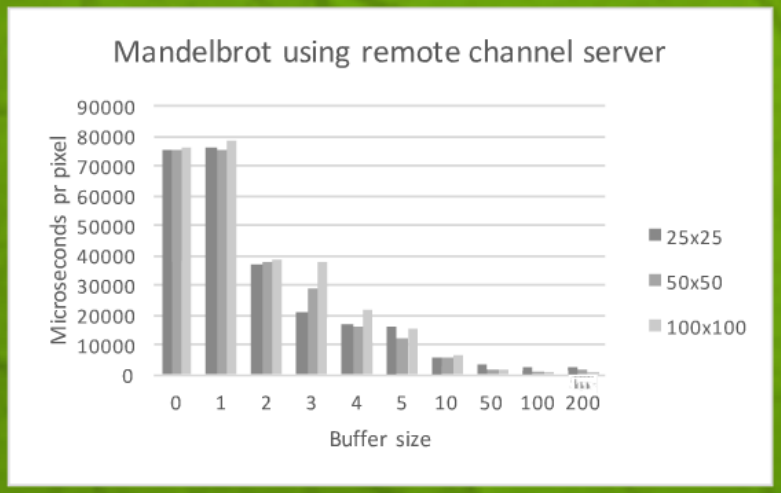

All channels using loopback network
Microseconds pr communication
1800
1750
1700
1650
1600
1550
0
1
2
3
4
5
50
Channel buffer size

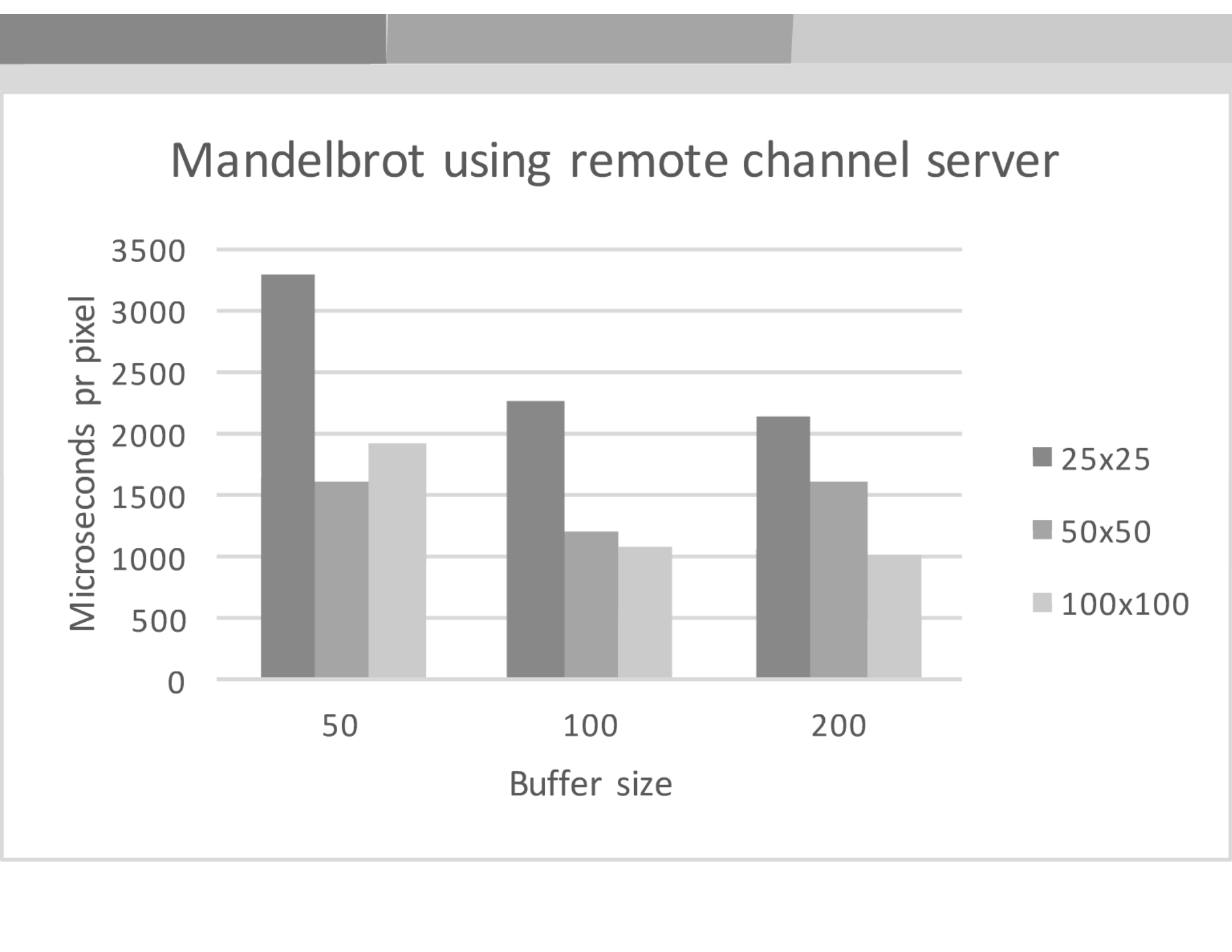

All channels using remote channel server
Microseconds pr communication
160000
140000
120000
100000
80000
60000
40000
20000
0
0
10
20
30
40
50
Channel buffer size

Mandelbrot

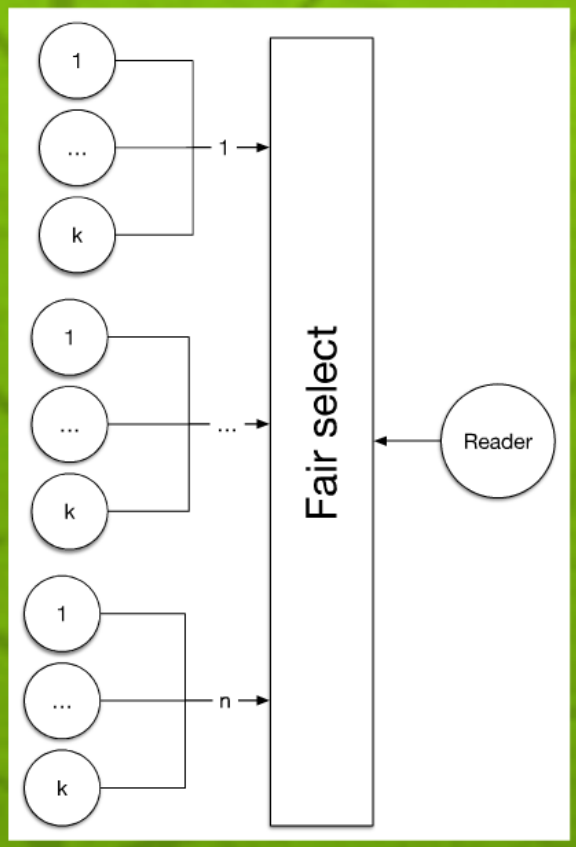

Mandelbrot without network
Microseconds pr pixel
250
200
150
100
50
0
25x25
50x50
100x100

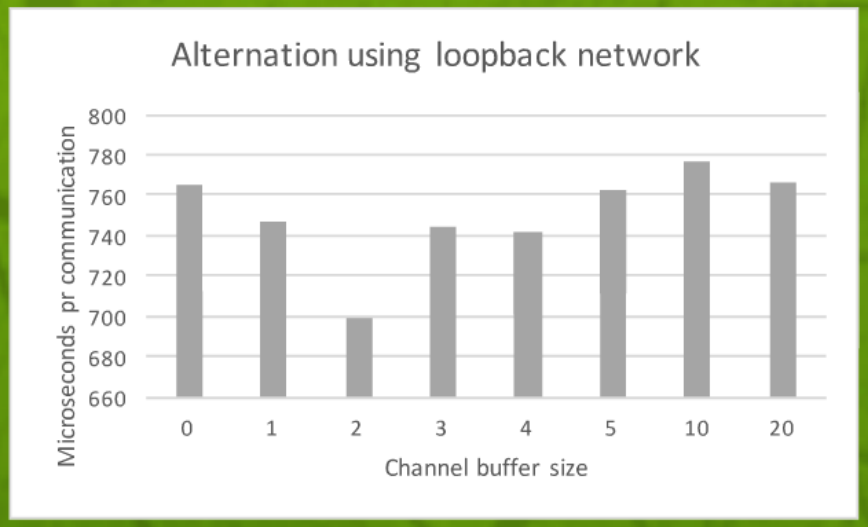

Mandelbrot using loopback network
Microseconds pr pixel
3000
2500
2000
1500
1000
500
0
0 1 2 3 4 5 10 50 100 200
Buffer size
25x25
50x50
100x100

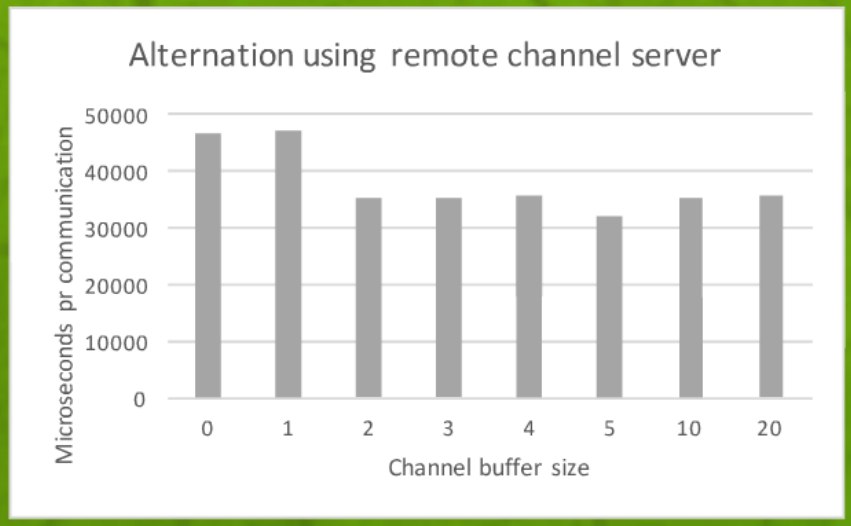

Mandelbrot using remote channel server
Microseconds pr pixel
90000
80000
70000
60000
50000
40000
30000
20000
10000
0
0 1 2 3 4 5 10 50 100 200
Buffer size
25x25
50x50
100x100


Mandelbrot using remote channel server
Microseconds pr pixel
3500
3000
2500
2000
1500
1000
500
0
50
100
200
Buffer size
25x25
50x50
100x100

StressedAlt







Future work

Error handling,
transaction logs

Skeleton methods

Channel mobility

Thanks!

Source code with benchmarks and examples on GitHub:
https://github.com/kenkendk/cocol

Extensions to the Concurrent Communications Library

Kenneth Skovhede
Niels Bohr Institute
University of Copenhagen

CPA 2016 - Copenhagen