CoCoL

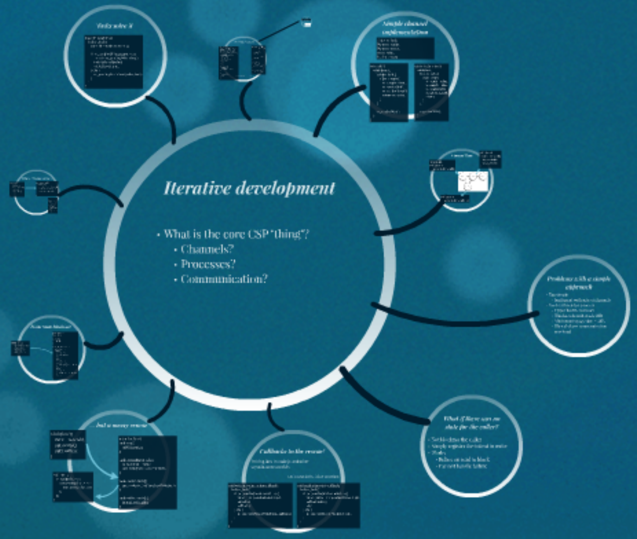
Concurrent Communication Library

Motivation

CIL (aka .Net aka CLR aka Mono) does not have a CSP library*

The CSP.Net library featured on CPA in 2008 went commercial, then extinct

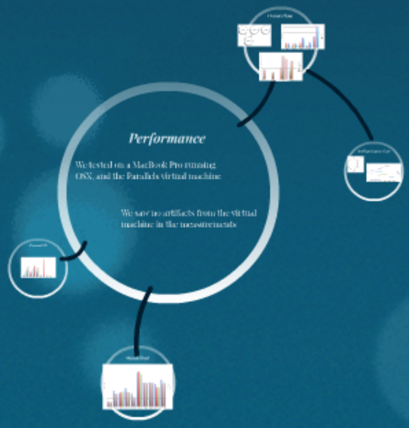
*) number of implementations existed, but none appear active



Iterative development

- What is the core CSP "thing"?
  - Channels?
  - Processes?
  - Communication?

Simple channel implementation
Problems with a simple approach
What if there was no state for the caller?
Callbacks to the rescue!
... but a messy rescue



Performance

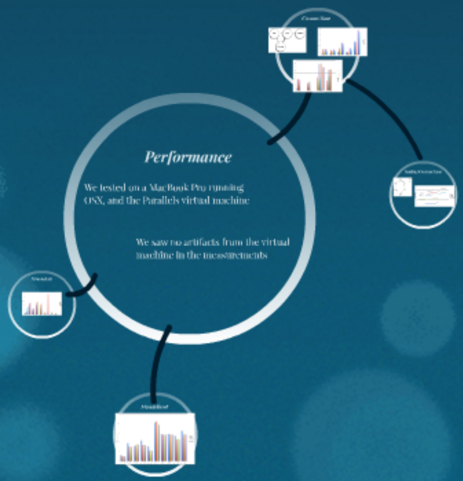
We tested on a MacBook Pro running OSX, and the Parallels virtual machine

We saw no artifacts from the virtual machine in the measurements

CPA 2015-08-25
Kenneth Skovhede
Niels Bohr Institute
University of Copenhagen

CoCoL

Concurrent Communication Library



Performance

We tested on a MacBook Pro running OSX, and the Parallels virtual machine

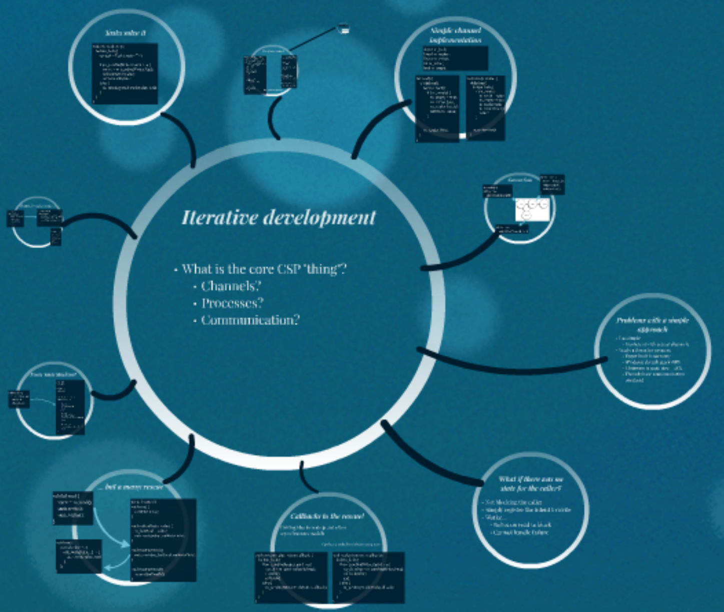
We saw no artifacts from the virtual machine in the measurements

Motivation

CIL (aka .Net aka CLR aka Mono) does not have a CSP library*

The CSP.Net library featured on CPA in 2006 went commercial, then extinct

*a number of implementations exist, but none appear to live



Iterative development

- What is the core CSP "thing"?
  - Channels?
  - Processes?
  - Communication?

Tasks solve it
Simple channel implementation
Problems with a simple approach
What if there was no state for the caller?
Callbacks to the rescue!
... but a messy rescue

CPA 2015-08-25
Kenneth Skovhede
Niels Bohr Institute
University of Copenhagen

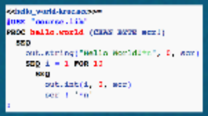
Motivation

CIL (aka .Net aka CLR aka Mono) does not have a CSP library*

The CSP.Net library featured on CPA in 2006 went commercial, then extinct

*A number of implementations exist, but none appear active

KRoC



```
<<hello_world-kroc.occ>>
#USE "course.lib"
PROC hello.world (CHAN BYTE scr!)
  SEQ
    out.string("Hello World!*n", 0, scr!)
    SEQ i = 1 FOR 10
      SEQ
        out.int(i, 0, scr!)
        scr ! '*n'
:
```

Design goals for a new CSP Library

- Fully contained in C#
    - No additional compilers or tools
    - No extra language constructs
- Simple codebase
- Fit into existing terminology & code
- Scalable
- Efficient execution

JCSP, C++CSP, PyCSP

Motivation

CIL (aka .Net aka CLR aka Mono) does not have a CSP library*

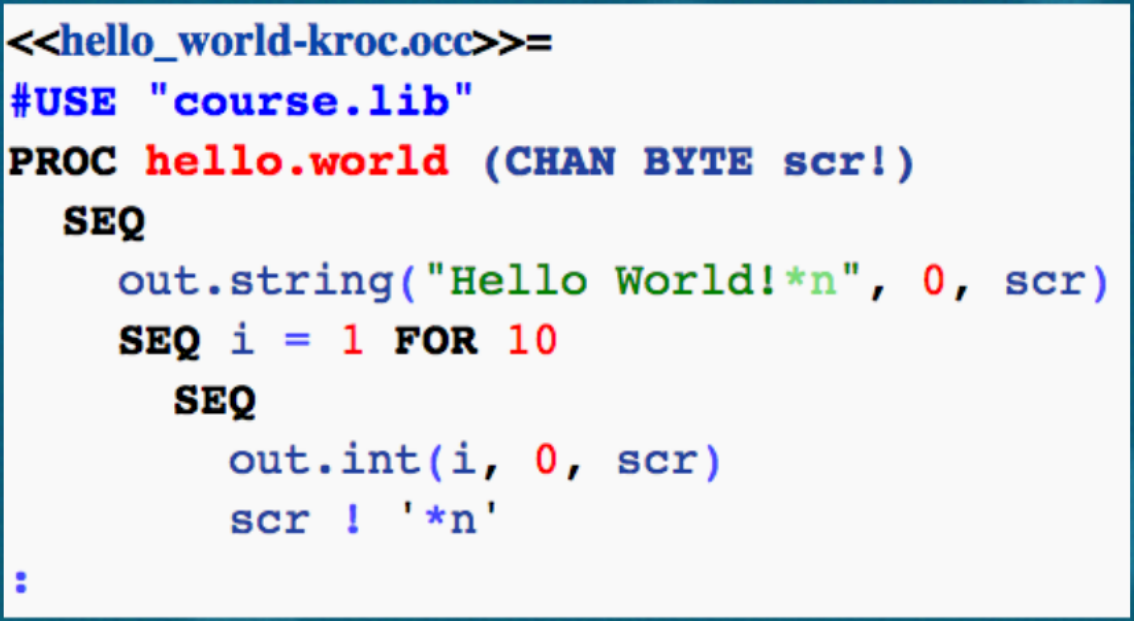
The CSP.Net library featured on CPA in 2006 went commercial, then extinct

*A number of implementations exist, but none appear active

Design goals for a new CSP Library

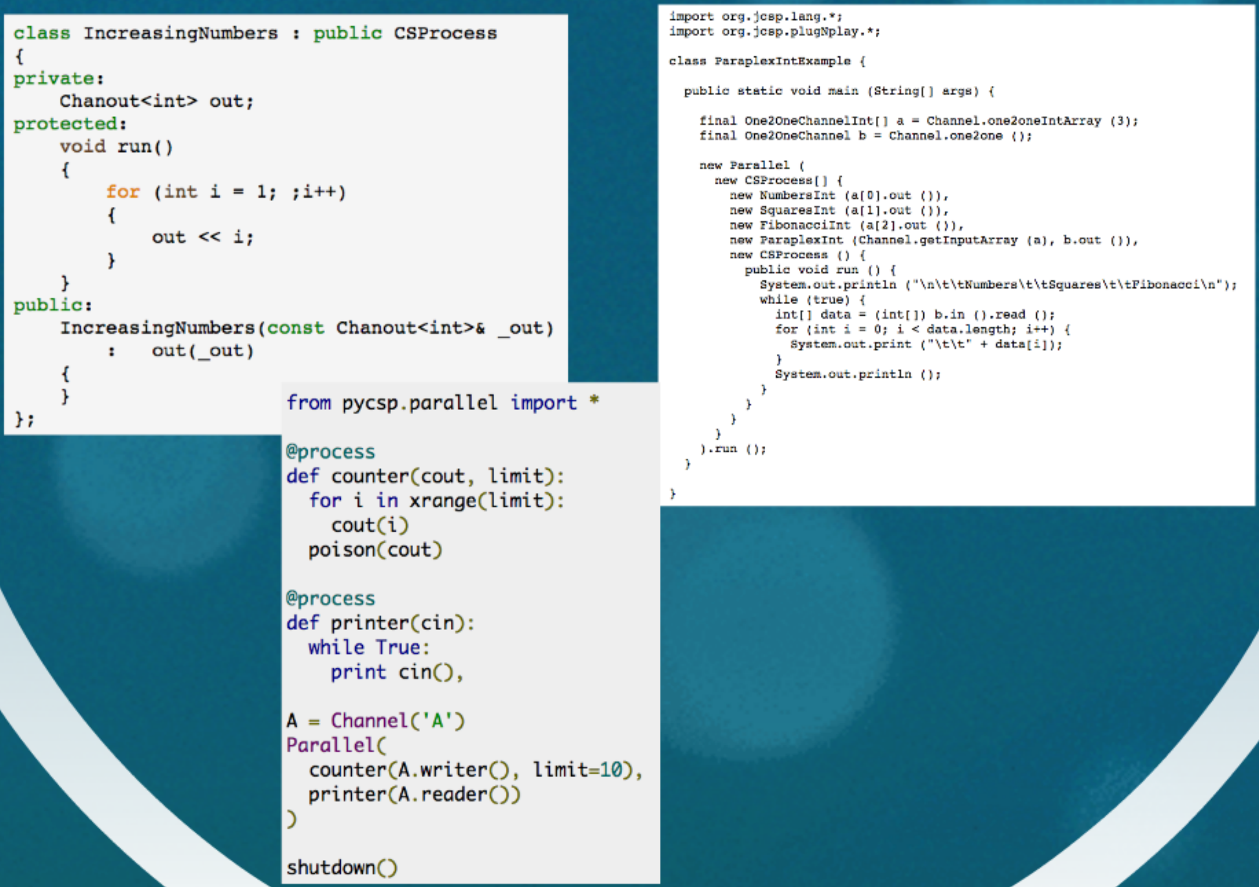
- Fully contained in C#
    - No additional compilers or tools
    - No extra language constructs
- Simple codebase
- Fit into existing terminology & code
- Scalable
- Efficient execution

KRoC



```
<<hello_world-kroc.occ>>=
#USE "course.lib"
PROC hello.world (CHAN BYTE scr!)
  SEQ
    out.string("Hello World!*n", 0, scr)
    SEQ i = 1 FOR 10
      SEQ
        out.int(i, 0, scr)
        scr ! '*n'
:
```

JCSP, C++CSP, PyCSP



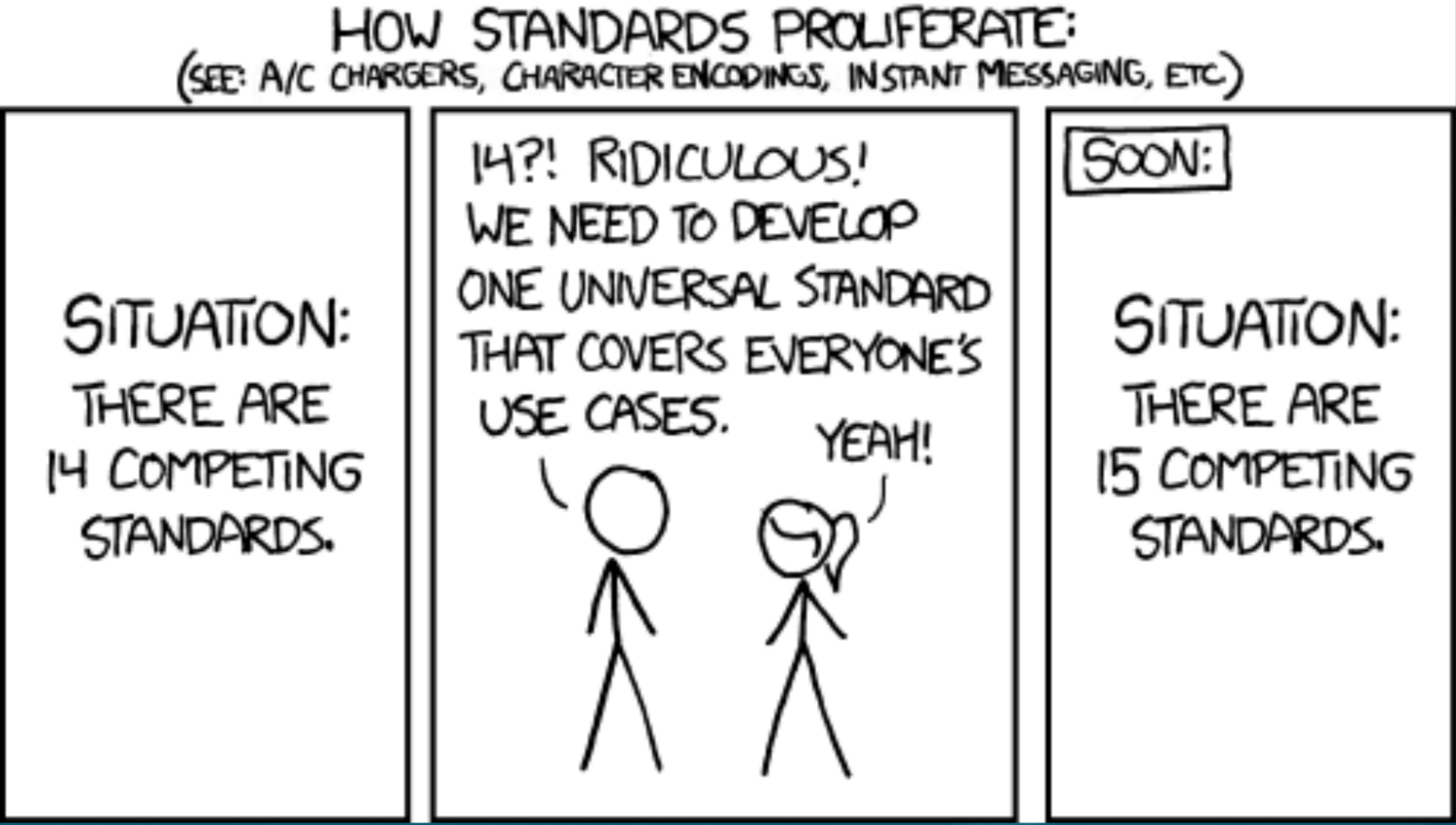
```cpp
class IncreasingNumbers : public CSProcess
{
private:
    Chanout<int> out;
protected:
    void run()
    {
        for (int i = 1; ;i++)
        {
            out << i;
        }
    }
public:
    IncreasingNumbers(const Chanout<int>& _out)
        :   out(_out)
    {
    }
};
```

```python
from pycsp.parallel import *

@process
def counter(cout, limit):
  for i in xrange(limit):
    cout(i)
  poison(cout)

@process
def printer(cin):
  while True:
    print cin(),

A = Channel('A')
Parallel(
  counter(A.writer(), limit=10),
  printer(A.reader())
)

shutdown()
```

```java
import org.jcsp.lang.*;
import org.jcsp.plugNplay.*;

class ParaplexIntExample {

  public static void main (String[] args) {

    final One2OneChannelInt[] a = Channel.one2oneIntArray (3);
    final One2OneChannel b = Channel.one2one ();

    new Parallel (
      new CSProcess[] {
        new NumbersInt (a[0].out ()),
        new SquaresInt (a[1].out ()),
        new FibonacciInt (a[2].out ()),
        new ParaplexInt (Channel.getInputArray (a), b.out ()),
        new CSProcess () {
          public void run () {
            System.out.println ("\n\t\tNumbers\t\tSquares\t\tFibonacci\n");
            while (true) {
              int[] data = (int[]) b.in ().read ();
              for (int i = 0; i < data.length; i++) {
                System.out.print ("\t\t" + data[i]);
              }
              System.out.println ();
            }
          }
        }
      }
    ).run ();
  }

}
```

Design goals for a new CSP Library

- Fully contained in C#
  - No additional compilers or tools
  - No extra language constructs
- Simple codebase
- Fit into existing terminology & code
- Scalable
- Efficient execution


HOW STANDARDS PROLIFERATE:
(SEE: A/C CHARGERS, CHARACTER ENCODINGS, INSTANT MESSAGING, ETC.)
SITUATION: THERE ARE 14 COMPETING STANDARDS.
14?! RIDICULOUS! WE NEED TO DEVELOP ONE UNIVERSAL STANDARD THAT COVERS EVERYONE'S USE CASES.
YEAH!
SOON:
SITUATION: THERE ARE 15 COMPETING STANDARDS.

Iterative development

- What is the core CSP "thing"?
    - Channels?
    - Processes?
    - Communication?

Tasks solve it

```
Task<T> read<T>() {
   lock(m_lock) {
      var task = Task.Create<T>();

      if (m_pendingWrites.Count > 0) {
         var wt = m_pendingWrites.Pop();
         task.SetResult(value);
         wt.SetResult(true);
      } else {
         m_pendingReads.Push(value, task);
      }
   }
}
```

Two-phase commit?

```
[illegible]
```

Simple channel implementation

```
object m_lock;
Event m_reader;
Event m_writer;
int m_value;
bool m_empty
```

```
int read() {
   while(true) {
      lock(m_lock) {
         if (!m_empty) {
            m_empty = true;
            m_writer.Set();
            m_reader.Reset();
            return m_value;
         }
      }

      m_reader.Wait();
   }
}
```

```
void write(int value) {
   while(true) {
      lock(m_lock) {
         if (m_empty) {
            m_result = value;
            m_empty = false;
            m_reader.Set();
            m_writer.Reset();
            return;
         }
      }

      m_writer.Wait();
   }
}
```

CommsTime

```
in.write(0);
while(true)
   out.write(in.read());
```

```
while(true) {
   var v = in.read();
   out1.write(v);
   out2.write(v);
}
```

```
while(true)
   out.write(in.read() + 1);
```

Problems with a simple approach

- Too simple
    - Inefficient with sets of channels
- Needs a thread pr process
    - Upper limit is memory
    - Windows default stack 1MB
    - Minimum is page size = 4KB
    - Threads have communication overhead

What if there was no state for the caller?

- Not blocking the caller
- Simply register the intent to write
- Works....
    - Relies on read to block
    - Cannot handle failure

Callbacks to the rescue!

Driving idea in node.js and other asynchronous models

Can be extended to deliver exceptions

```
void write(int value, Action callback) {
   lock(m_lock) {
      if (m_pendingReads.Count > 0) {
         var cb = m_pendingReads.Pop();
         cb(value);
         callback();
      } else {
         m_pendingWrites.Push(value, callback);
      }
   }
}
```

```
void read(Action<int> callback) {
   lock(m_lock) {
      if (m_pendingWrites.Count > 0) {
         var cb, value = m_pendingWrites.Pop();
         callback(value);
         cb();
      } else {
         m_pendingReads.Push(callback);
      }
   }
}
```

... but a messy rescue

```
while(true) {
   var v = in.read();
   out1.write();
   out2.write();
}
```

```
int m_lastRead;
void run() {
   onWriteOut2();
}

void onReadIn(int value) {
   m_lastRead = value;
   out1.write(value, onWriteOut1);
}

void onWriteOut1() {
   out2.write(m_lastRead, onWriteOut2);
}

void onWriteOut2() {
   in.read(onReadIn);
}
```

```
void run() {
   in.read(value => {
      out1.write(value, () => {
         out2.write(value, run)
      })
   });
}
```

Finite State Machine?

```
while(true) {
   var v = in.read();
   out1.write();
   out2.write();
}
```

```
[illegible]
```

Future, Promise, async ...

```
[illegible]
```

Iterative development

- What is the core CSP "thing"?
  - Channels?
  - Processes?
  - Communication?

in.write(0);
while(true)
  out.write(in.rea

while(tru
  out.wr

te Machine?

```
int m_state = 0;
int m_value;

void run() {
  callback();
}

void callback(int? value) {
  switch(m_state) {
    case 0:
      m_state = 1;
      in.read(callback);
      break;
    case 1:
      m_state = 2;
      m_value = value;
      out1.write(m_value, callback);
      break;
    case 2:
      m_state = 0;
      out2.write(m_value, callback);
      break;
  }
}
```

Simple channel implementation

```
object m_lock;
Event m_reader;
Event m_writer;
int m_value;
bool m_empty
```

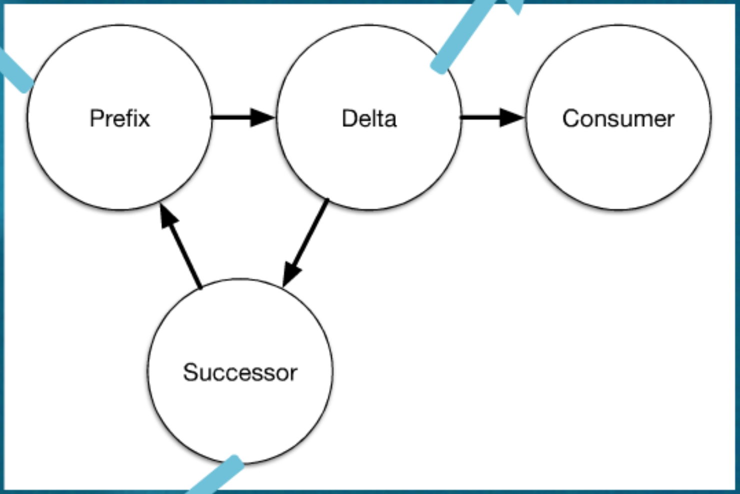
```
int read() {
   while(true) {
      lock(m_lock) {
         if (!m_empty) {
            m_empty = true;
            m_writer.Set();
            m_reader.Reset();
            return m_value;
         }
      }

      m_reader.Wait();
   }
}
```

```
void write(int value) {
   while(true) {
      lock(m_lock) {
         if (m_empty) {
            m_result = value;
            m_empty = false;
            m_reader.Set();
            m_writer.Reset();
            return;
         }
      }

      m_writer.Wait();
   }
}
```

CommsTime

```
in.write(0);
while(true)
    out.write(in.read());
```

```
while(true) {
    var v = in.read();
    out1.write(v);
    out2.write(v);
}
```



```
while(true)
    out.write(in.read() + 1)
```

Problems with a simple approach

- Too simple
    - Inefficient with sets of channels
- Needs a thread pr process
    - Upper limit is memory
    - Windows default stack 1MB
    - Minimum is page size = 4KB
    - Threads have communication overhead

What if there was no state for the caller?

- Not blocking the caller
- Simply register the intent to write
- Works....
  - Relies on read to block
  - Cannot handle failure

Callbacks to the rescue!

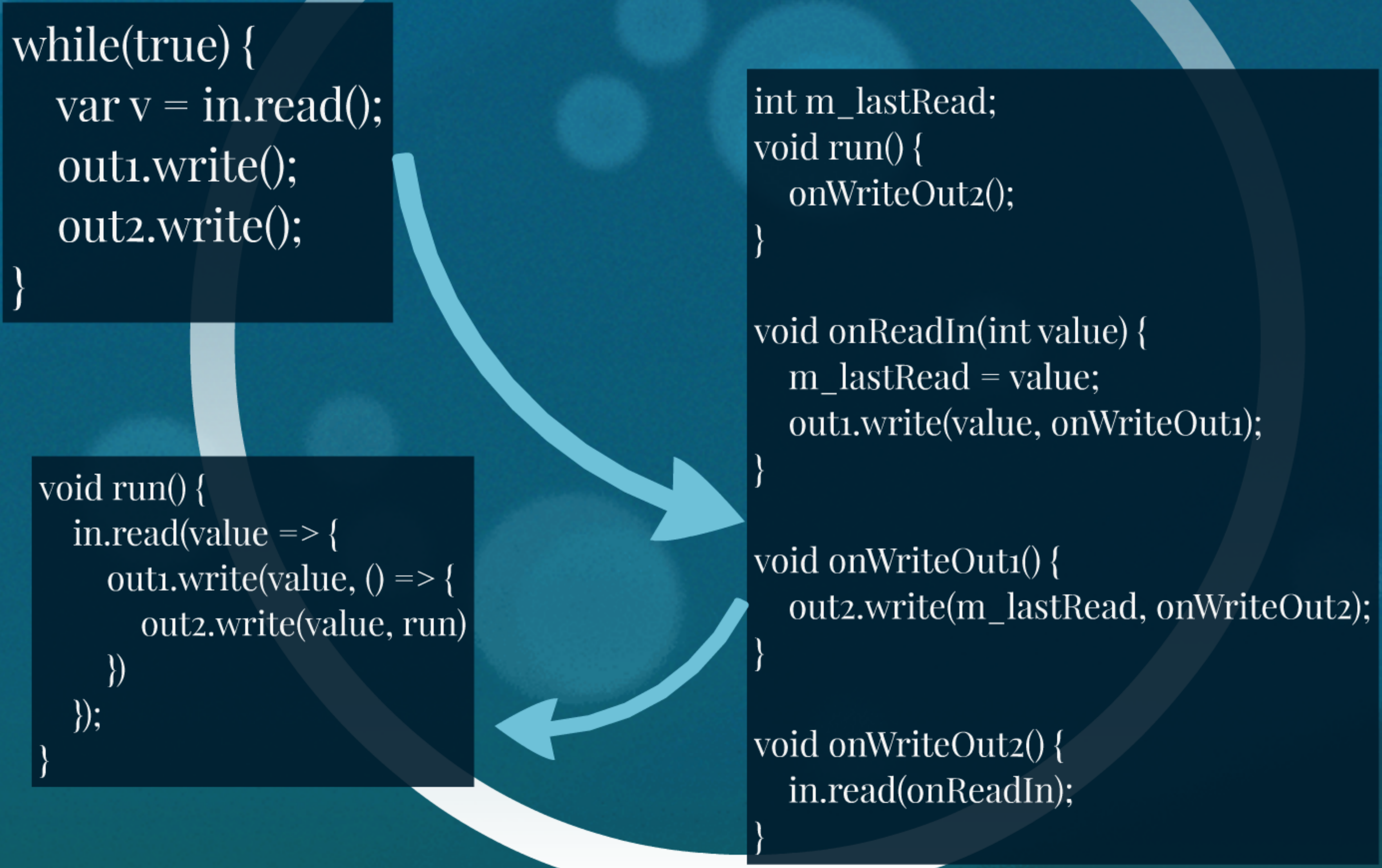
Driving idea in node.js and other asynchronous models

Can be extended to deliver exceptions

```
void write(int value, Action callback) {
   lock(m_lock) {
      if (m_pendingReads.Count > 0) {
         var cb = m_pendingReads.Pop();
         cb(value);
         callback();
      } else {
         m_pendingWrites.Push(value, callback);
      }
   }
}
```

```
void read(Action<int> callback) {
   lock(m_lock) {
      if (m_pendingWrites.Count > 0) {
         var cb, value = m_pendingWrites.Pop();
         callback(value);
         cb();
      } else {
         m_pendingReads.Push(callback);
      }
   }
}
```

… but a messy rescue



```
while(true) {
   var v = in.read();
   out1.write();
   out2.write();
}
```

```
int m_lastRead;
void run() {
   onWriteOut2();
}

void onReadIn(int value) {
   m_lastRead = value;
   out1.write(value, onWriteOut1);
}

void onWriteOut1() {
   out2.write(m_lastRead, onWriteOut2);
}

void onWriteOut2() {
   in.read(onReadIn);
}
```

```
void run() {
   in.read(value => {
      out1.write(value, () => {
         out2.write(value, run)
      })
   });
}
```

Finite State Machine?

```
while(true) {
   var v = in.read();
   out1.write();
   out2.write();
}
```

```
int m_state = 0;
int m_value;

void run() {
   callback();
}

void callback(int? value) {
   switch (m_state) {
      case 0:
         m_state = 1;
         in.read(callback);
         break;
      case 1:
         m_state = 2;
         m_value = value;
         out1.write(m_value, callback);
         break;
      case 2:
         m_state = 0;
         out2.write(m_value, callback);
         break;
   }
}
```

Future, Promise, async …

```
void run() {
   while(true) {
      var v = in.read();
      out1.write();
      out2.write();
   }
}
```

```
void async run() {
   while(true) {
      var value = await in.read();
      await out1.write(value);
      await out2.write(value);
   }
}
```

```
int m_state = 0;
int m_value;

void callback(int? value) {
   switch (m_state) {
      case 0:
         m_state = 1;
         in.read(callback);
         break;
      case 1:
         m_state = 2;
         m_value = value;
         out1.write(m_value, callback);
         break;
      case 2:
         m_state = 0;
         out2.write(m_value, callback);
         break;
   }
}
```

Tasks solve it

```
Task<T> read<T>() {
  lock(m_lock) {
    var task = Task.Create<T>();

    if (m_pendingWrites.Count > 0) {
      var wt = m_pendingWrites.Pop();
      task.SetResult(value);
      wt.SetResult(true);
    } else {
      m_pendingReads.Push(value, task);
    }
  }
}
```

Two-phase commit

```
while (m_pendingWrites.Count > 0) {
   var wt = m_pendingWrites.Peek();
   var writerAccepts = wt.Offer();
   var readerAccepts = task.Offer();
   if (writerAccepts && readerAccepts) {
      m_pendingWrites.Pop();
      wt.Commit();
      task.Commit();
      task.SetResult(value);
      wt.SetResult(true);
      return;
   }

   if (writerAccepts) {
      wt.WithDraw();
      m_pendingWrites.Pop();
   }

   if (readerAccepts) {
      task.WithDraw();
      return;
   }
}
```

```
bool Offer(object caller) {
   Monitor.Enter(m_lock);

   if (!m_taken)
      return true;

   Monitor.Exit(m_lock);
   return false;
}

void Commit() {
   m_taken = true;
   Monitor.Exit(m_lock);
}

void WithDraw() {
   Monitor.Exit(m_lock);
}
```

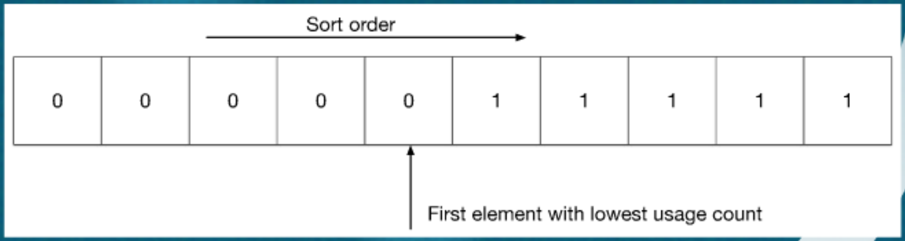
Many optimizations implemented ...

External choice

```
Task<T> readFromAny(IEnumerable<Channel<T>> channels) {
   var twophase = new TwoPhaseCommit();
   return Task.WhenAny(from c in channels select c.read(twophase));
}
```

Timeouts are handled with a shared timer, skip is done with timeout=0

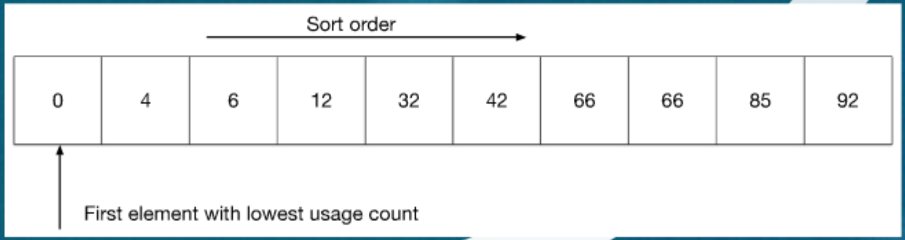
Fair select relies on a "usage counter"



Sort order

0	0	0	0	0	1	1	1	1	1

First element with lowest usage count



Sort order

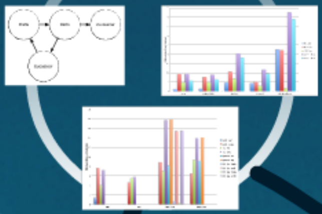
0	4	6	12	32	42	66	66	85	92

First element with lowest usage count

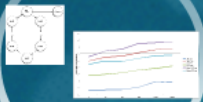
Performance

We tested on a MacBook Pro running OSX, and the Parallels virtual machine

We saw no artifacts from the virtual machine in the measurements



CommsTime



Scaling CommsTime

StressedAlt

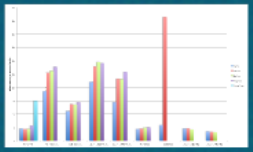
Mandelbrot

Performance

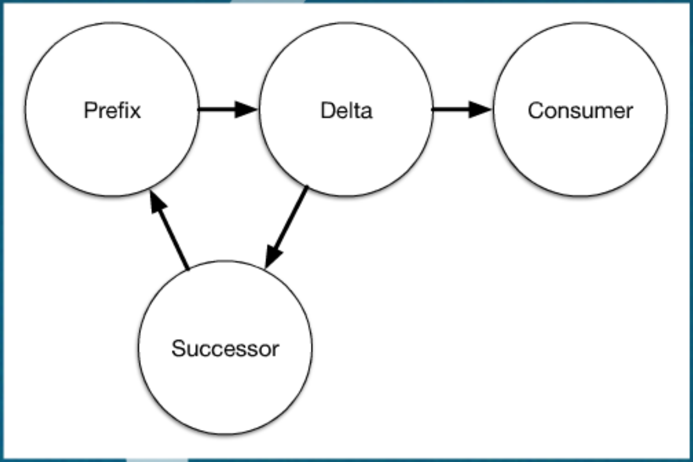
We tested on a MacBook Pro running OSX, and the Parallels virtual machine

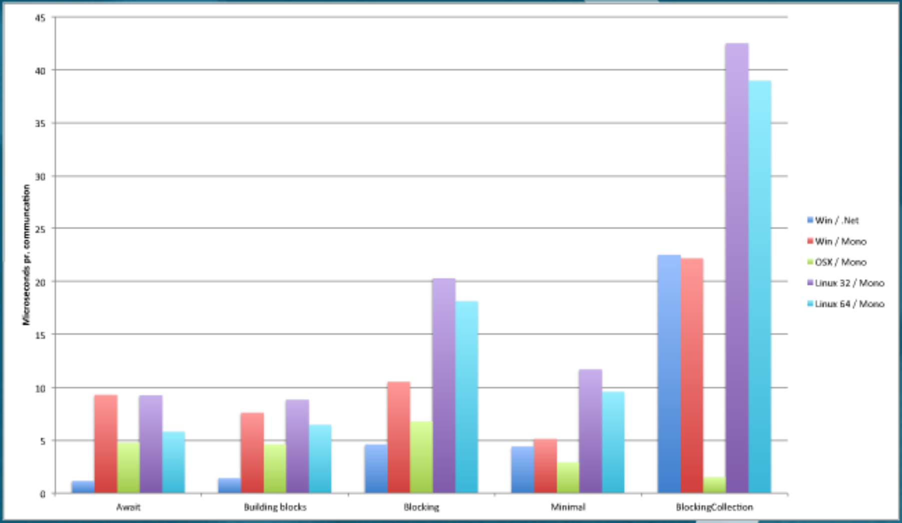
We saw no artifacts from the virtual machine in the measurements

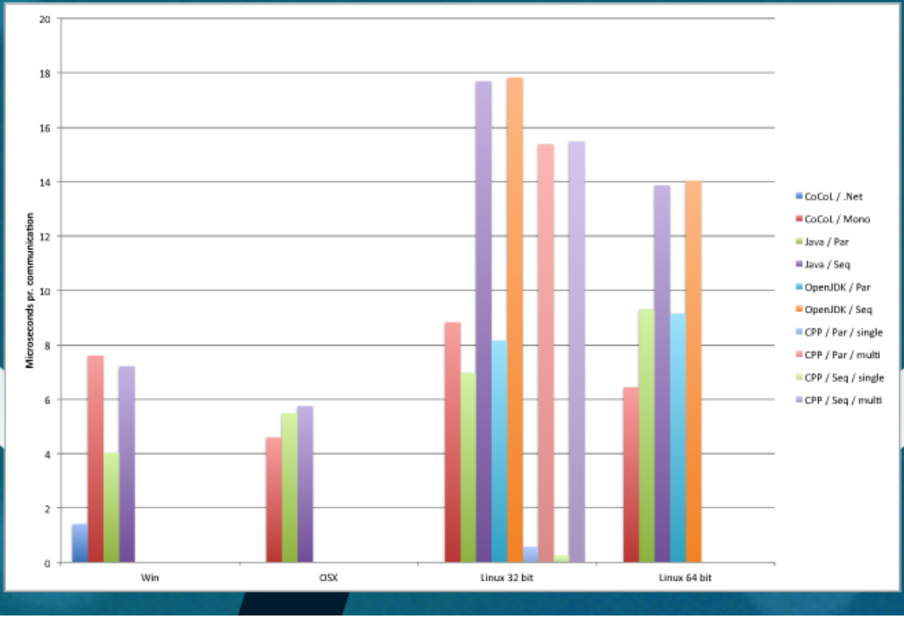
StressedAlt

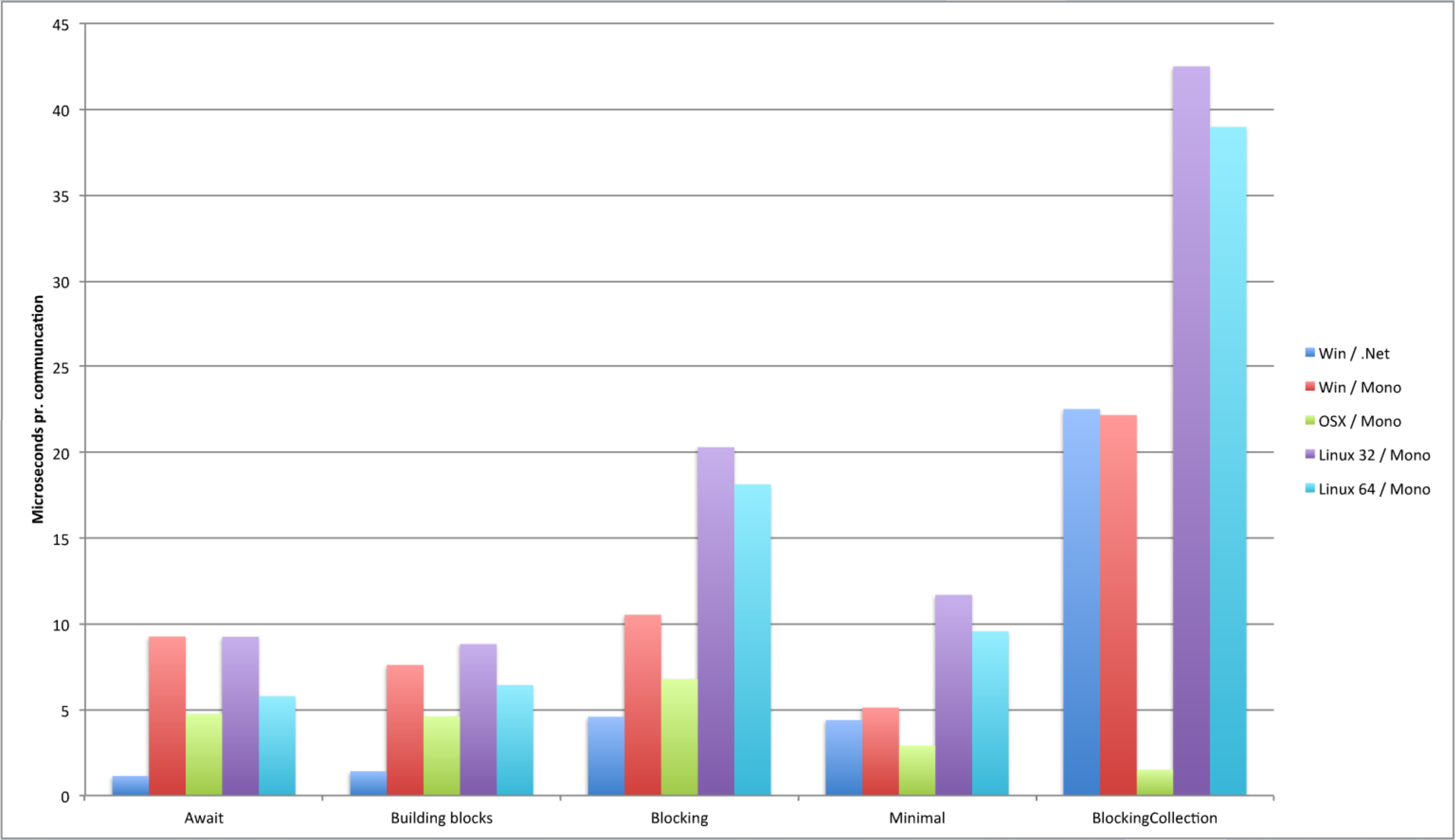


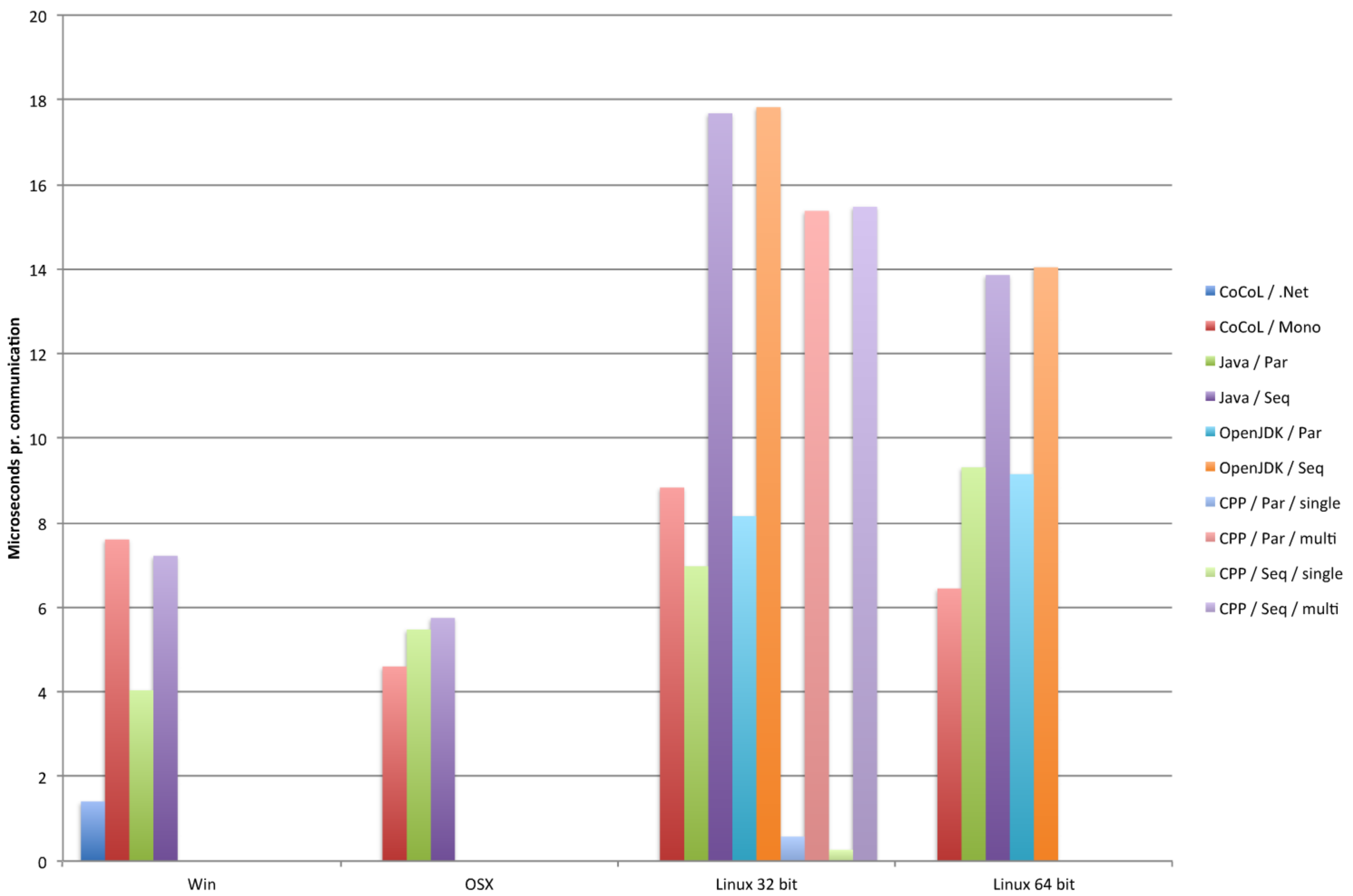
CommsTime


Prefix
Delta
Consumer
Successor


45
40
35
30
25
20
15
10
5
0
Microseconds pr. communcation
Await
Building blocks
Blocking
Minimal
BlockingCollection
Win / .Net
Win / Mono
OSX / Mono
Linux 32 / Mono
Linux 64 / Mono

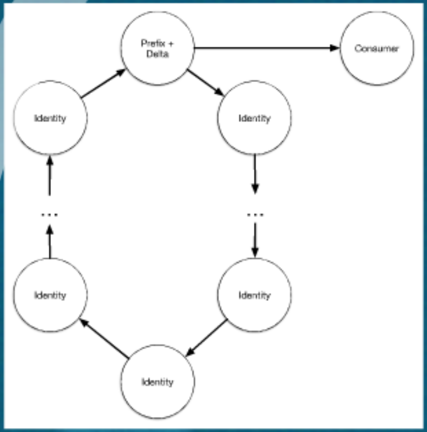

20
18
16
14
12
10
8
6
4
2
0
Microseconds pr. communication
Win
OSX
Linux 32 bit
Linux 64 bit
CoCoL / .Net
CoCoL / Mono
Java / Par
Java / Seq
OpenJDK / Par
OpenJDK / Seq
CPP / Par / single
CPP / Par / multi
CPP / Seq / single
CPP / Seq / multi

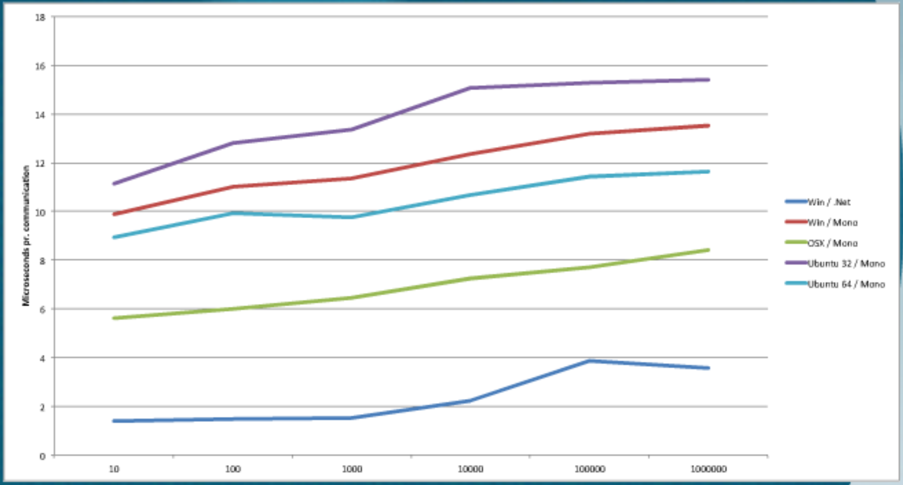

Microseconds pr. communcation
45
40
35
30
25
20
15
10
5
0
Await
Building blocks
Blocking
Minimal
BlockingCollection
Win / .Net
Win / Mono
OSX / Mono
Linux 32 / Mono
Linux 64 / Mono

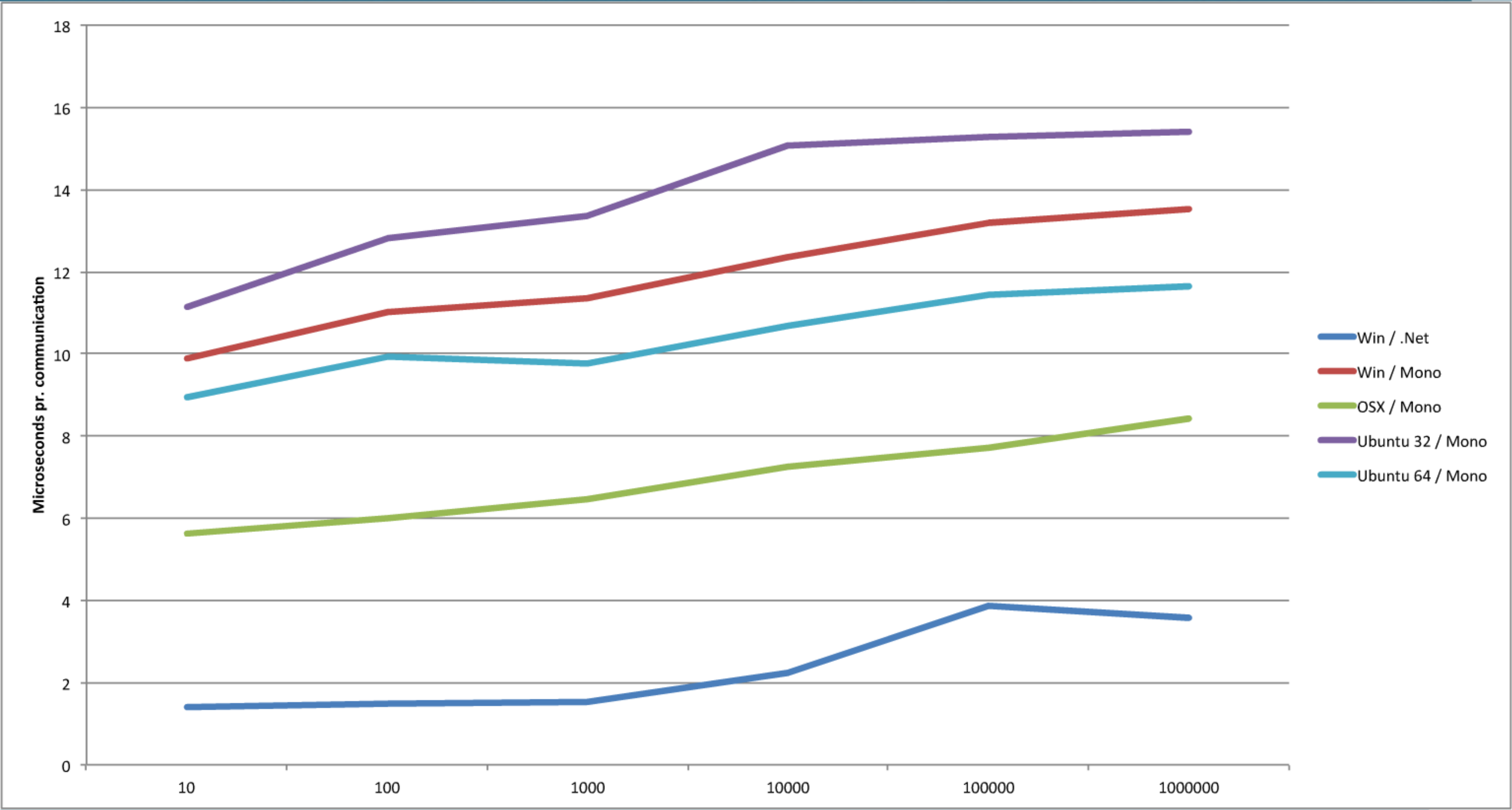


20
18
16
14
12
10
8
6
4
2
0
Microseconds pr. communication
Win
OSX
Linux 32 bit
Linux 64 bit
CoCoL / .Net
CoCoL / Mono
Java / Par
Java / Seq
OpenJDK / Par
OpenJDK / Seq
CPP / Par / single
CPP / Par / multi
CPP / Seq / single
CPP / Seq / multi

Scaling CommsTime

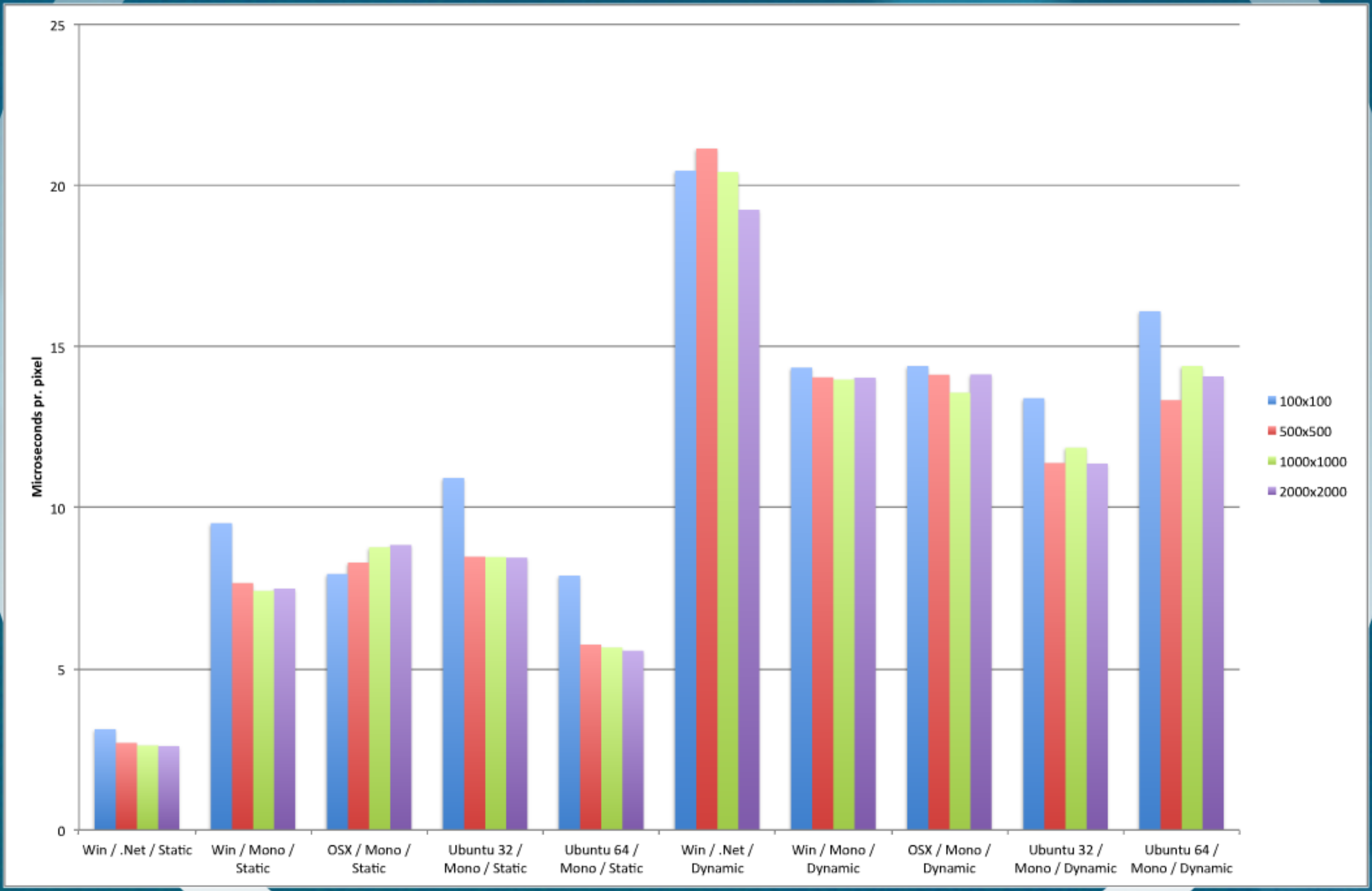


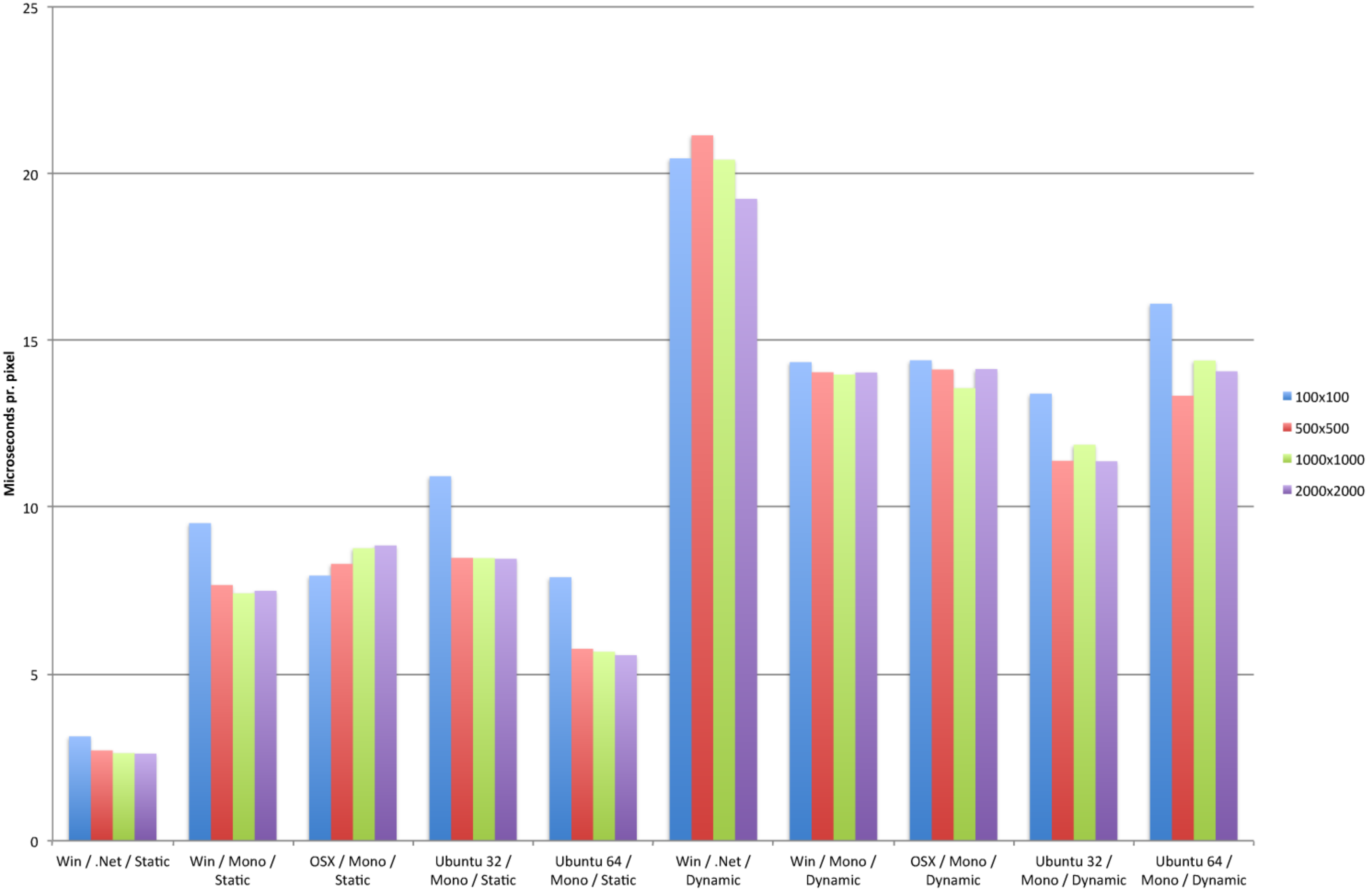




Microseconds pr. communication
18
16
14
12
10
8
6
4
2
0
10
100
1000
10000
100000
1000000
Win / .Net
Win / Mono
OSX / Mono
Ubuntu 32 / Mono
Ubuntu 64 / Mono

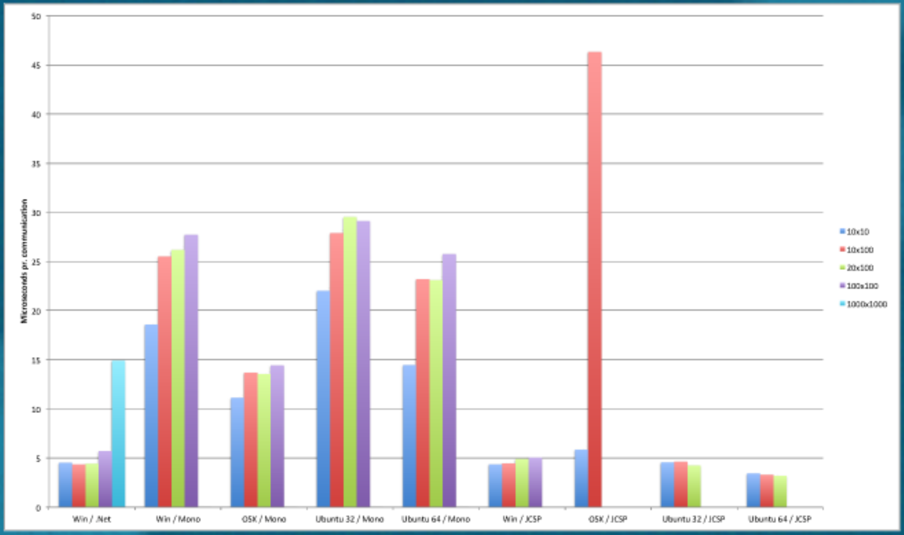
Mandelbrot

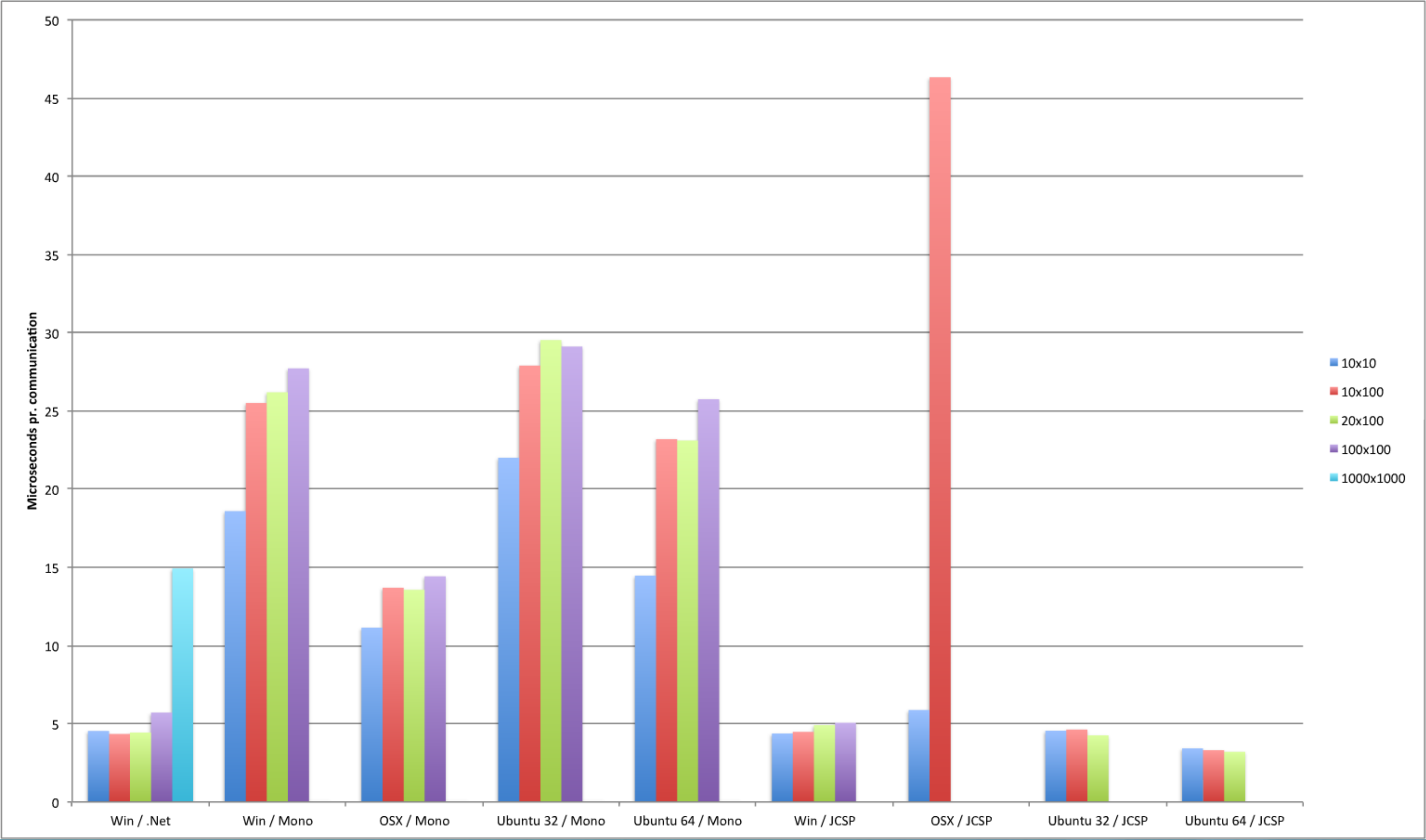

Microseconds pr. pixel
0
5
10
15
20
25
Win / .Net / Static
Win / Mono / Static
OSX / Mono / Static
Ubuntu 32 / Mono / Static
Ubuntu 64 / Mono / Static
Win / .Net / Dynamic
Win / Mono / Dynamic
OSX / Mono / Dynamic
Ubuntu 32 / Mono / Dynamic
Ubuntu 64 / Mono / Dynamic
100x100
500x500
1000x1000
2000x2000



Microseconds pr. pixel
25
20
15
10
5
0
Win / .Net / Static
Win / Mono / Static
OSX / Mono / Static
Ubuntu 32 / Mono / Static
Ubuntu 64 / Mono / Static
Win / .Net / Dynamic
Win / Mono / Dynamic
OSX / Mono / Dynamic
Ubuntu 32 / Mono / Dynamic
Ubuntu 64 / Mono / Dynamic
100x100
500x500
1000x1000
2000x2000

StressedAlt




50
45
40
35
30
25
20
15
10
5
0
Microseconds pr. communication
Win / .Net
Win / Mono
OSX / Mono
Ubuntu 32 / Mono
Ubuntu 64 / Mono
Win / JCSP
OSX / JCSP
Ubuntu 32 / JCSP
Ubuntu 64 / JCSP
10x10
10x100
20x100
100x100
1000x1000

All code, including examples / benchmarks:

https://github.com/kenkendk/cocol

Small code footprint:

Channel is appx 300 SLOC
Entire library is 1500 SLOC