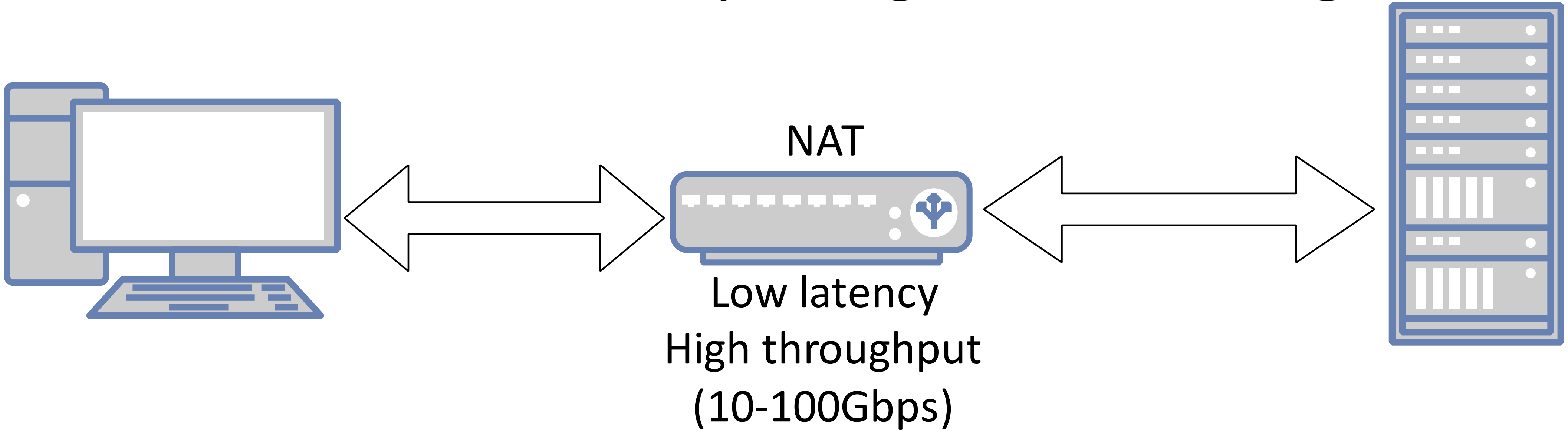
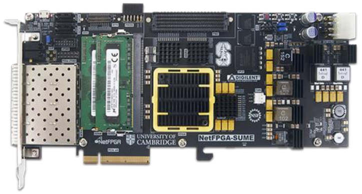
Using C# for high performance network programming





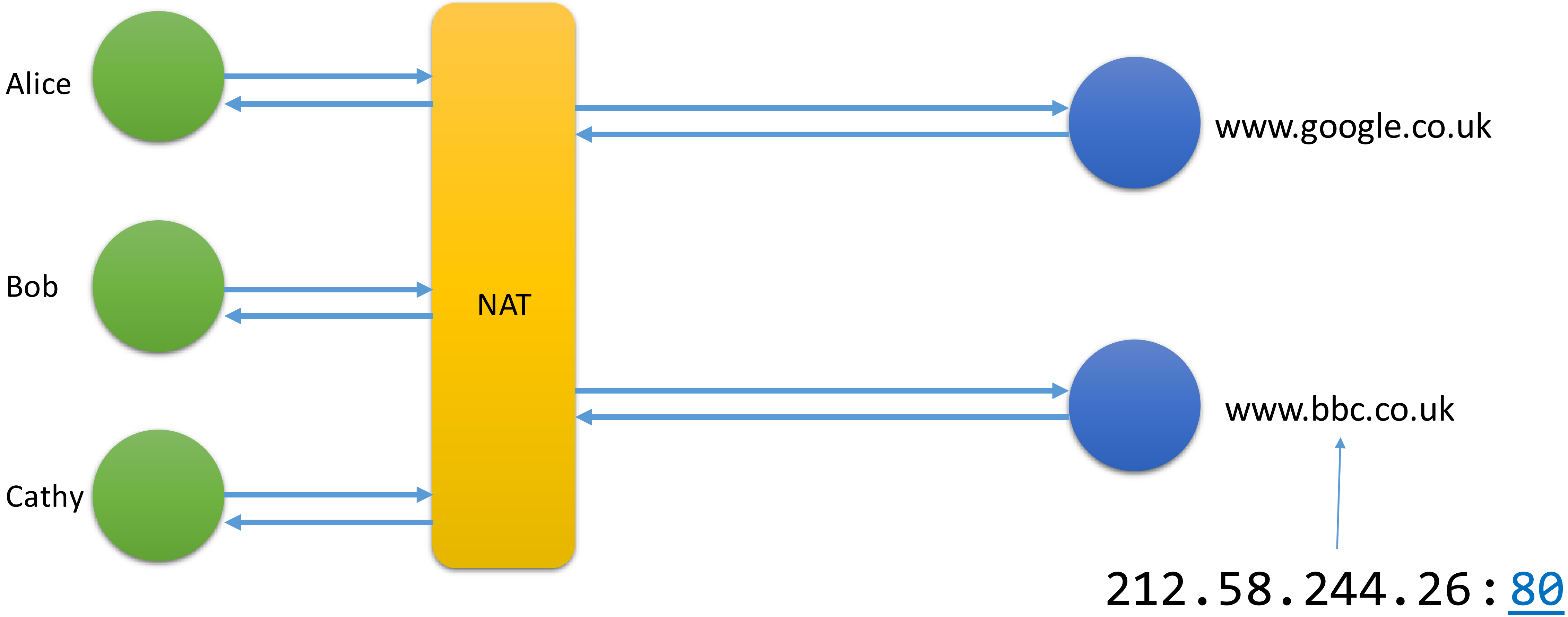
NetFPGA

Jonny Shipton

CL, University of Cambridge

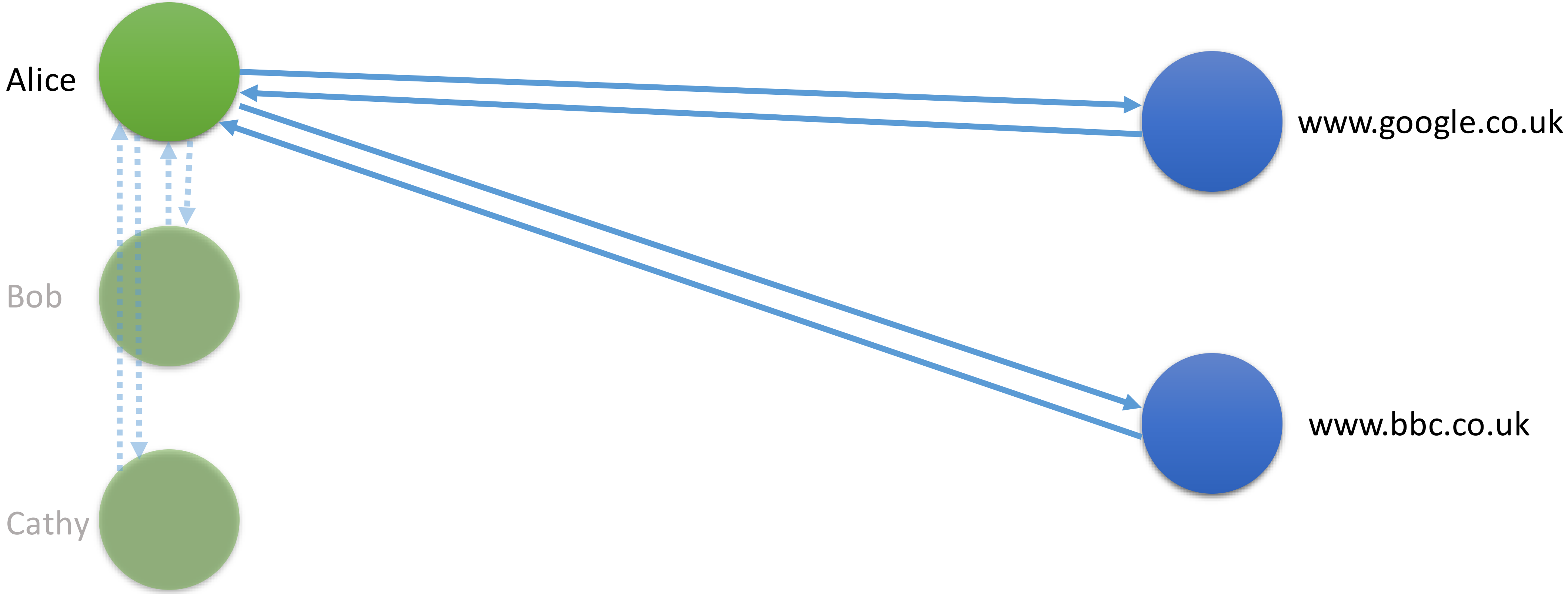
What is a NAT?

Reality



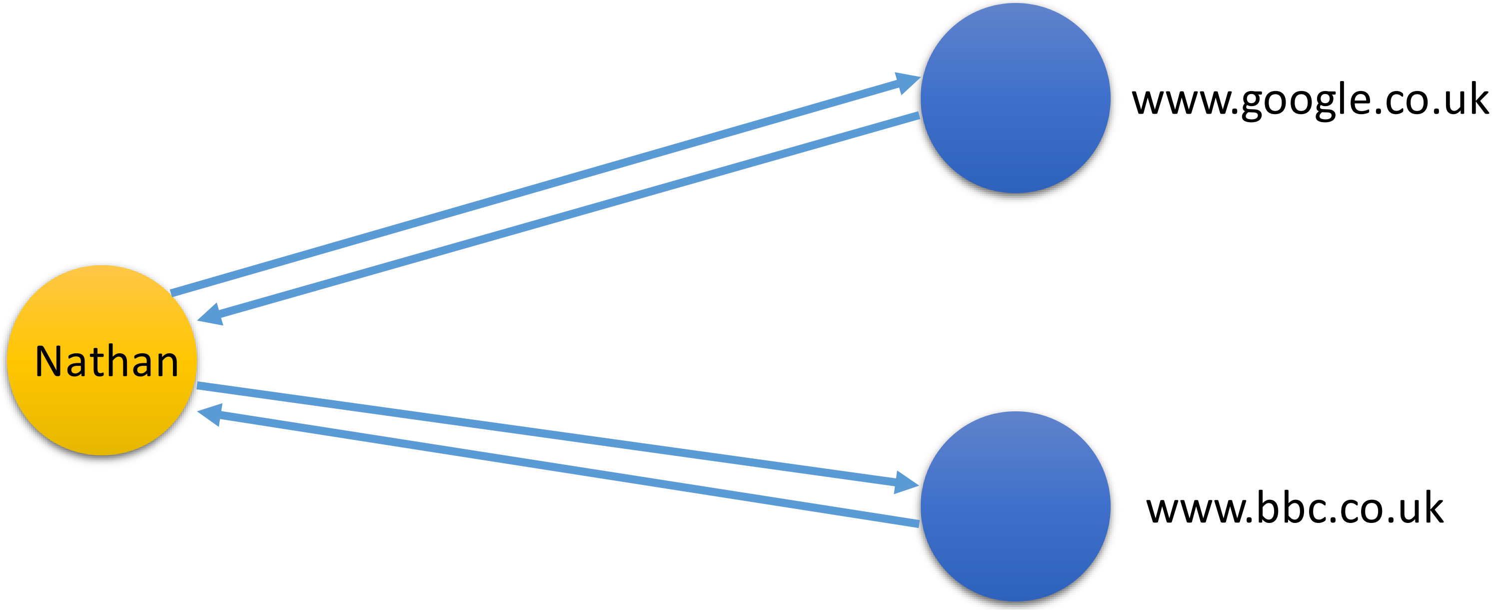
What is a NAT?

Client's POV



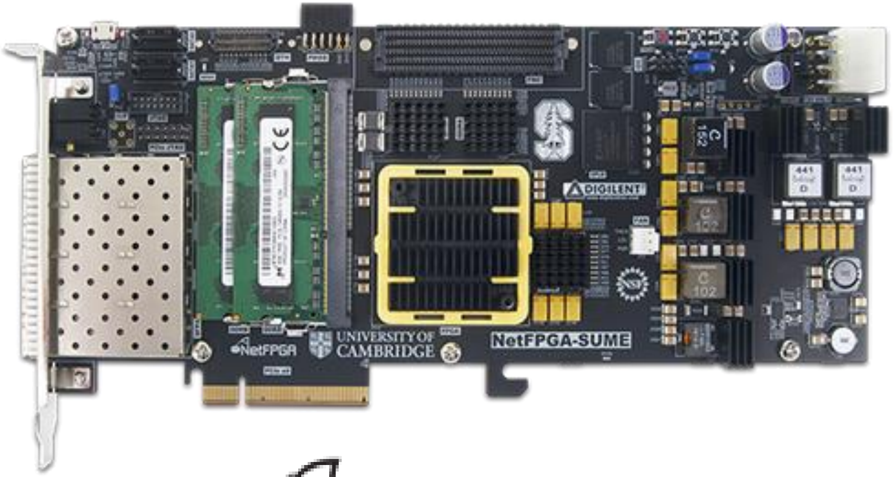
What is a NAT?

Server's POV



Why C#?

	Verilog	C#
Performance	✓✓✓	✓
Scalable to 100Gbps+	✓✓✓	✓
Skill base	Small	Large
Comfy language features	☹	✓✓✓



NetFPGA

Why C#?

```
try
{
  SetSendIntf(0x01);

  var values = from uint i in Enumerable.Range(0, 10)
               join ulong v in dataplane.tdata
                 on i equals v & 0xFF
               where (v & 0xFF00) == 0x1D00
               select v.Extract_Bytes(until_byte: i);

  foreach (var v in values.Cast<int>())
  {
    var task = new Task<TCPWrapper>(() => new TCPWrapper(dataplane.tdata, v, 10));
    task.Start();
    task.ContinueWith(tcp => TCP(null, null, tcp.Result, null)).Wait();
  }
}
catch (ArgumentException ex)
{
  Console.WriteLine(ex);
}
```

Methods

LINQ

Generics

Loops

Memory Management

OOP

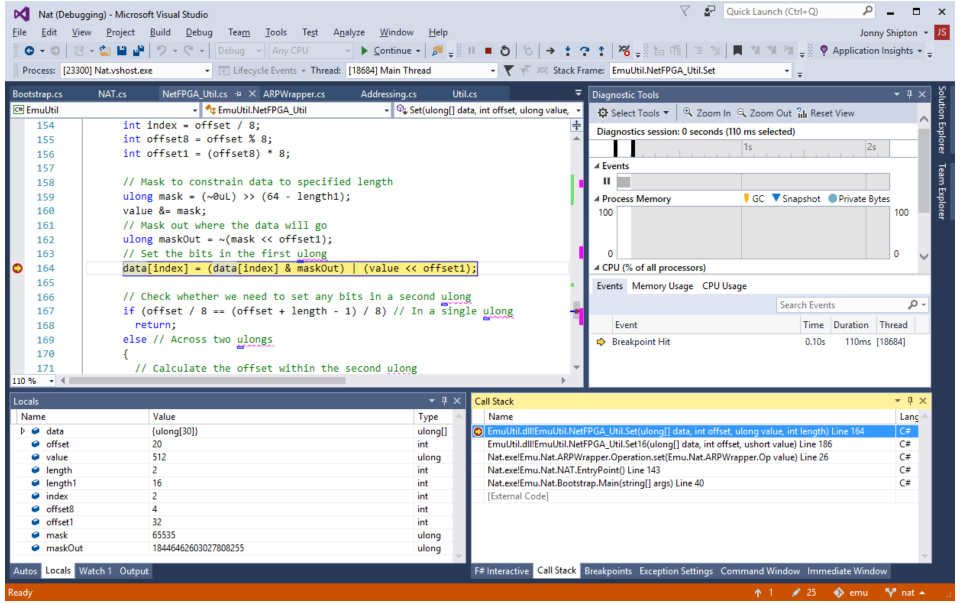
Exceptions

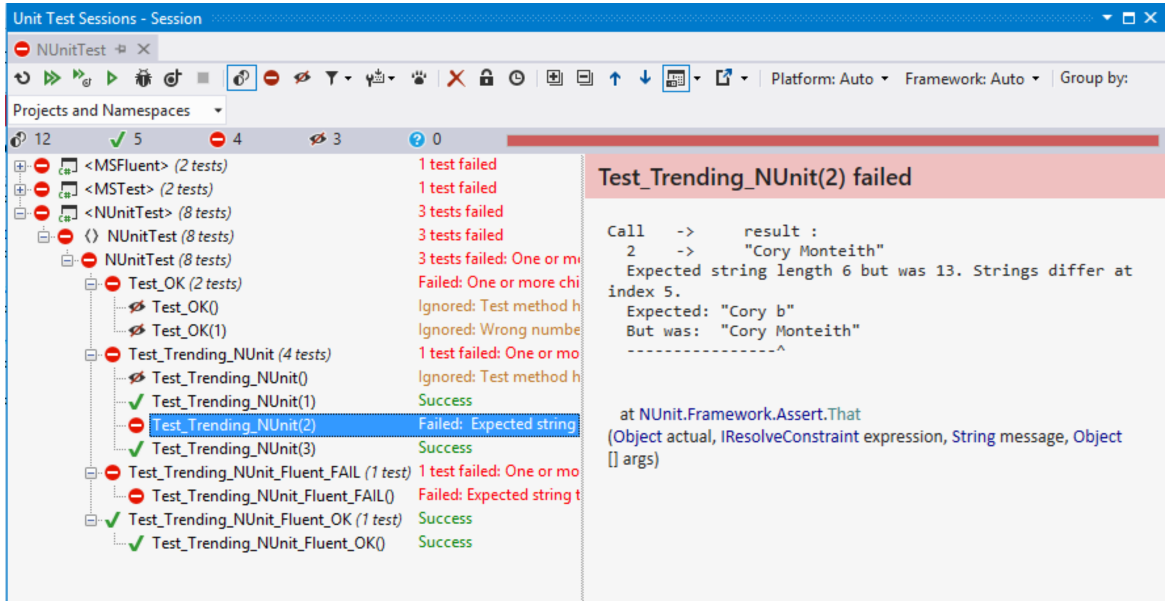
Concurrency

Lambdas

Also iterators, properties, inheritance, libraries, dynamic variables, safe expressive type system, …

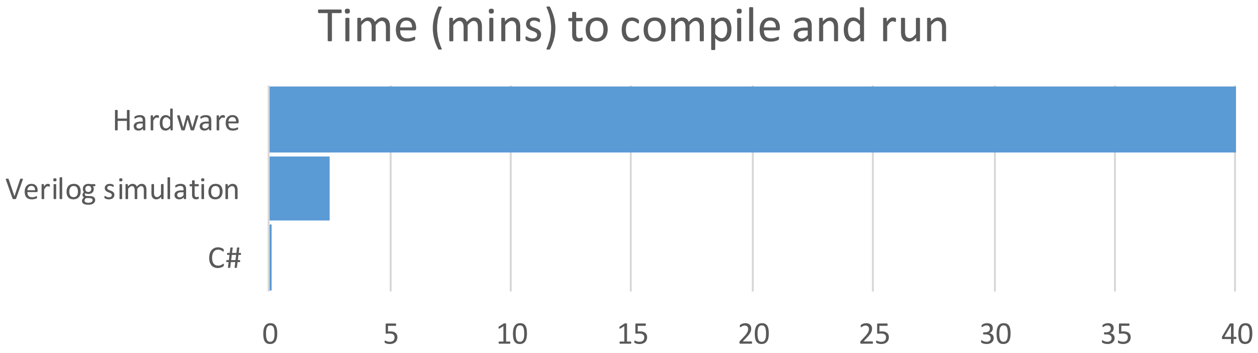
Why C#?



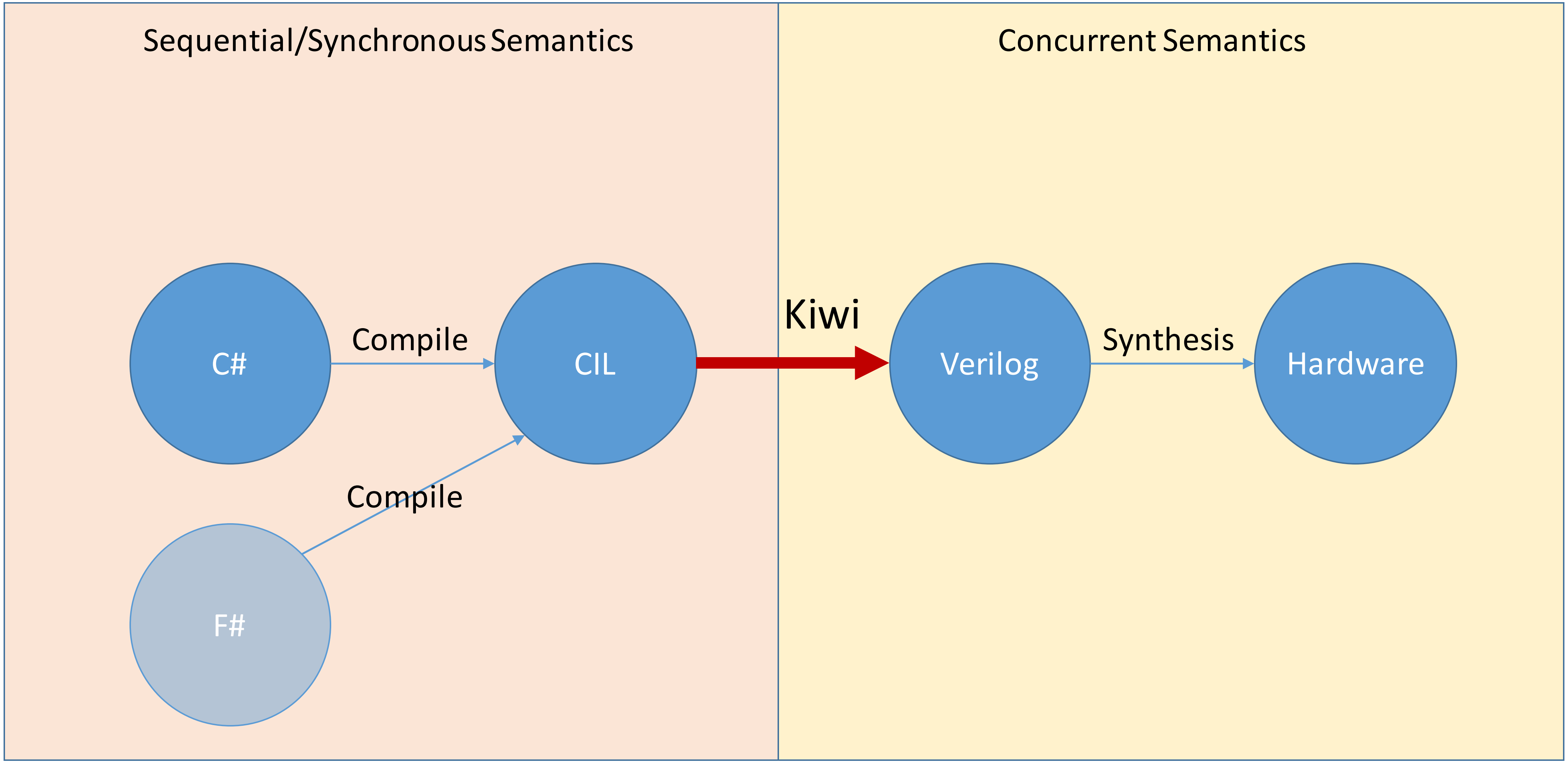


Visual Studio

ReSharper



Compile C# to Verilog?!



Look at the code

```
else if (eth.Ethertype == EthernetWrapper.EtherType.ARP)
{
  if (arp.Operation == ARPWrapper.Op.Request
    && arp.HardwareType == ARPWrapper.HwType.Ethernet
    && arp.ProtocolType == (ushort)EthernetWrapper.EtherType.IPv4
    && arp.HardwareAddressLength == 6
    && arp.ProtocolAddressLength == 4)
  {
    Kiwi.Pause(); // Reduces Kiwi compile time
    if (arp.TargetProtocolAddress == myInsideIP || arp.TargetProtocolAddress == myOutsideIP) // Request for a MAC I own
    {
      uint intfIp = arp.TargetProtocolAddress;
      ulong intfMac = macOfIntf(dataplane.GetReceiveIntf());

      // Update arp packet to use as response
      arp.Operation = ARPWrapper.Op.Reply;
      arp.TargetHardwareAddress = arp.SenderHardwareAddress;
      arp.TargetProtocolAddress = arp.SenderProtocolAddress;
      arp.SenderHardwareAddress = intfMac;
      arp.SenderProtocolAddress = intfIp;

      // Update ethernet fields
      eth.DestinationMac = arp.TargetHardwareAddress;
      eth.SourceMac = intfMac;

      Kiwi.Pause();

      // Send reply
      SetSendIntf(dataplane.GetReceiveIntf());
      SendFrame(pkt_size);
    }
  }
}
```

Look at the code

Object wrapping raw data array

```
if (arp.Operation == ARPWrapper.Op.Request ...

    ...

    arp.Operation = ARPWrapper.Op.Reply;
```

Property

Intuitive enum

```
public Op Operation { get { return (Op)data.Get16(Offset + 6); }
                      set {             data.Set16(Offset + 6, (ushort)value); } }
```

Limitations

- No truly dynamic allocation (yet)
  - Array sizes must be determined statically
  - Objects can be allocated and used within loops
- Library support less mature
- Complexity -> longer compile times
- No reflection, dynamic invoke, etc.
- Clock cycles are more precious, so we want to do more in each one
  - Asynchronous – Kiwi flattens code

Thanks to:

- Nik Sultana
- Salvator Galea
- David Greaves
- Networks-as-a-Service (Naas-project.org)
- EPSRC
- Kynesim

Kiwi

Google “Kiwi compiler”

http://www.cl.cam.ac.uk/~djg11/kiwi/
http://www.cl.cam.ac.uk/research/srg/han/hprls/orangepath/kiwic.html

NetFPGA-SUME

http://netfpga.org/site/#/systems/1netfpga-sume/details/