Chords in C#
Introduction

- Polyphonic C# is an extension to the C# language
- Extension is aimed at providing in-language concurrency support

Agenda:
1. Extension Syntax
2. Rules
3. How it works
4. Translation of new constructs to traditional C#
5. An example (Stock Server : Active Object)
New Syntax

No return: Just schedule this method for execution in another thread

```csharp
async methodName(argumentType stuff) {
    //stuff to do
}
```
New Syntax

Chord body executes when all of the methods in the header have been called
Syntax Rules – Method Declarations

“ref” and “out” parameter modifiers cannot be used in async methods because by the time this method is executed, who knows what the caller is doing?

```csharp
async method(out int i) {
    //stuff
}
```
Syntax Rules – Chord Declarations

Only one synchronous method is allowed in a chord declaration. Otherwise, in which thread is the body executed? This decision could have behavioral effects.
Syntax Rules – Chord Declarations

If a method header has a return value, the body can return a value of that type. If no type is provided then an empty return statement can be used.
Syntax Rules – Chord Declarations

All formals appearing in method-headers must have distinct identifiers. Otherwise, there would be translation issues… we’ll see how chords are translated into conventional C# later!

```csharp
void get(int i) & async calculate(int i) {
    //stuff
}
```
Syntax Rules – Chord Declarations

Two method headers within the same chord declaration may not have both the same method name and argument types. Otherwise, which function to call at runtime?

```csharp
void set(int i) & async set(int k) {
    //stuff
}
```
Syntax Rules – Chord Declarations

All method headers within a chord declaration must be either instance declarations or static declaration: Never a mix.
Syntax Rules – Within a Class

All method headers with the same member name and argument type signature must have the same return type and identical sets of attributes and modifiers.

class c {
    string work(int i) & async set(int k) {
        //stuff
    }
    int work(int i) & async set(int k) {
        //stuff
    }
}
Syntax Rules – Within a Class

When methods are overridden, all their chords must also be completely overridden.

class C {
    virtual void f() & virtual async g() {/*stuff1*/}
    virtual void f() & virtual async h() {/*stuff2*/}
}

class D:C {
    override async g() {/*stuff3*/} \(\textcolor{red}{\text{WRONG}}\)
    override void f() & override async g() {/*new stuff 1*/}
    override void f() & override async h() {/*new stuff2*/} \(\textcolor{green}{\text{Correct!}}\)
}
Chord Methodology

```csharp
class Token
{
    public Token(int initial_tokens)
    {
        for (int i = 0; i < initial_tokens; i++) Release();
    }

    public int Grab(int id)
    {
        return id;
    }

    public async void Release()
    {
        await Task.Delay(1000); // Simulate processing time
    }
}
```

Thread A

Thread B

Thread Q

void Q

Scan
Chord Translation - BitMask

```csharp
struct BitMask {
    private int v; // = 0;
    public void set(int m) { v |= m; }
    public void clear(int m) { v &= ~m; }
    public bool match(int m) { return (~v & m) == 0; }
}
```
Chord Translation - VoidQ

class voidQ {
    private int n;
    public void Q(){ n = 0; }
    public void add() {n++;}
    public void get() {n--;
    public bool empty {get{return n==0;}}
}

Chord Translation - ThreadQ

class threadQ {
    private bool signalled = false;
    private int count = 0;
    public bool empty {get{return count == 0;}}
    public void yield(object myCurrentLock){
        count++;
        Monitor.Exit(myCurrentLock);
        lock(this){
            while(!signaled) Monitor.Wait(this);
            signaled = false;
        }
        Monitor.Enter(myCurrentLock);
        count--;
    }
    public void wakeup(){
        lock(this){
            signaled = true;
            Monitor.Pulse(this);
        }
    }
}
Chord Translation – Token Example Class

class Token {
    private const int mGrab = 1 << 0;
    private const int mRelease = 1 << 1;
    private threadQ GrabQ = new threadQ();
    private voidQ ReleaseQ = new voidQ();
    private const int mGrabRelease = mGrab | mRelease;
    private BitMask s = new BitMask();
    private object mlock = ReleaseQ;
    private void scan() {
        if (s.match(mGrabRelease)) {GrabQ.wakeup(); return;}
    }
    public Token(int initial tokens) {
        for (int i = 0; i < initial tokens; i++) Release();
    }
    [OneWay] public void Release() {
        lock(mlock) {
            ReleaseQ.add();
            if (!s.match(mRelease)) {
                s.set(mRelease);
                scan();}
        }
    }
    public int Grab(int id) {
        Monitor.Enter(mlock);
        if (!s.match(mGrab)) goto now;
        later:
        GrabQ.yield(mGrab); if (GrabQ.empty) s.clear(mGrab);
        now:
        if (s.match(mRelease)) {
            ReleaseQ.get(); if (ReleaseQ.empty) s.clear(mRelease);
            scan();
            Monitor.Exit(mlock);
            { return id; // source code for the chord }
        }
        else{
            s.set(mGrab); goto later; }
    }
}
Example (From Paper)

```csharp
public abstract class ActiveObject {
    protected bool done;
    abstract protected void ProcessMessage();

    public ActiveObject()
    {
        done = false;
        mainLoop();
    }

    async mainLoop()
    {
        while (!done)
        {
            ProcessMessage();
        }
    }
}

public class StockServer : ActiveObject{
    private ArrayList clients = new ArrayList();

    public async AddClient(Client c) {
        clients.Add(c);
    }

    public async WireQuote(Quote q) {
        foreach(Client c in clients){
            c.UpdateQuote(q);
        }
    }
}
```