What is F#?

- Microsoft Multi-paradigm Language
- CLR – All the .NET platform benefits
- First class language in to Visual Studio 2010

What about F# makes it functional or concurrent?

- Immutable Constructs
- Higher Order Functions
- Partial Application
- Ability to do lazy evaluation
- Asynchronous Workflows

Real World Application!

Auto Insurance Rating Engine

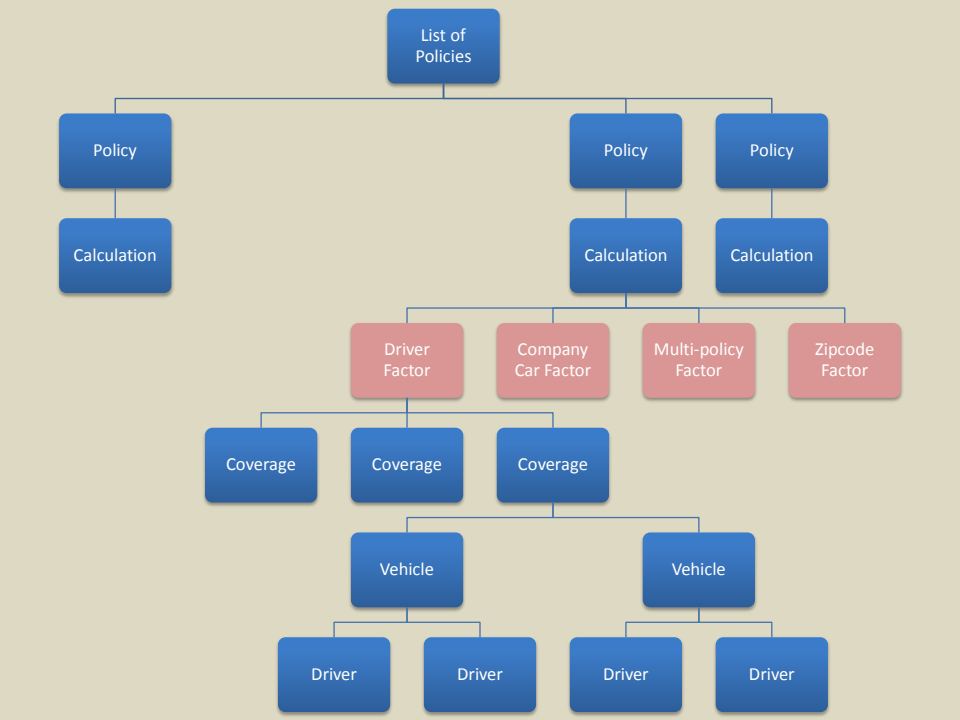
Grange Insurance

CALCULATION OF INDIVIDUAL DRIVER FACTOR
(one calculation is necessary for each driver on the policy)
Round to three (3) decimal places after each step.

STEP	Description	BI	PD	MP	UMBI	UMPD	OTC	Coll
1	Driver Class Factor							
2	Insurance Score Factor	×	×	×	×	×	×	×
3	Tier Factor	×	×	×	×	×	×	×
4	Accident Factor	×	×	×	×	×	×	×
5	DWI Factor	×	×	×	×	×	×	×
6	Major Violation Factor	×	×	×	×	×	×	×
7	Minor Violation Factor	×	×	×	×	×	×	×
8	Good Student Factor	×	×	×	×	×	×	×
9	Defensive Driver Factor	×	×	×	×	×	×	×
10	Good Partner Factor	×	×	×	×	×	×	×
11	Grange Life Insurance Factor	×	×	×	×	×	×	×
12	Away At School Factor	×	×	×	×	×	×	×
13	Individual Driver Factor	=	=	=	=	=	=	=

Driver Class Factor

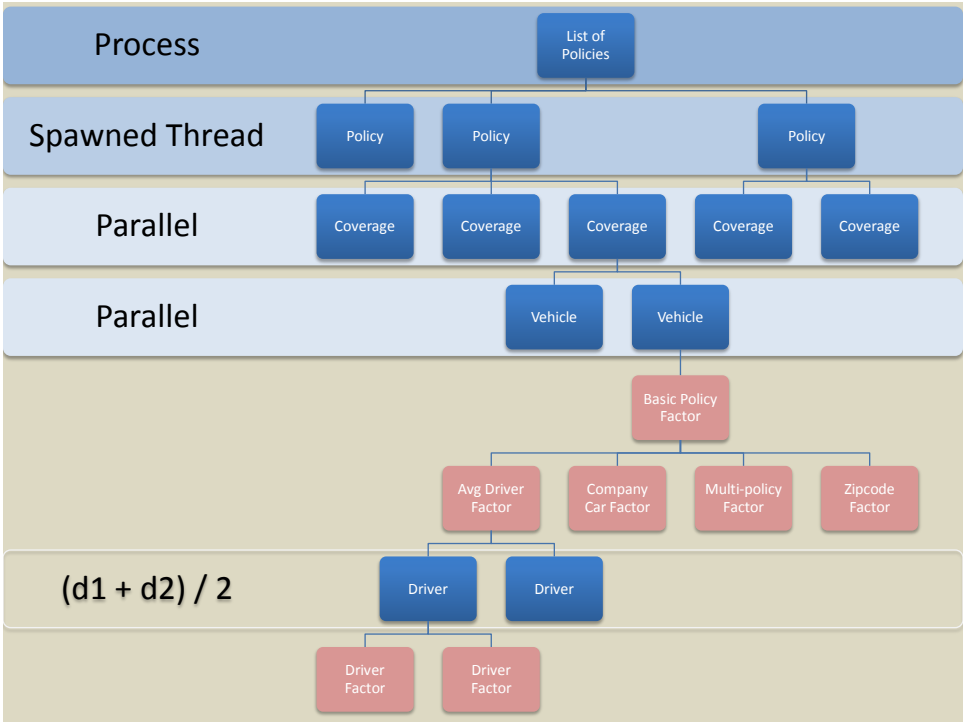
Gender	Marital Status	Age	BI	PD	MP	UMB	UMP	UNB	UNP	OTC	COLL
Female	Married	<=16	2.013	2.013	1.621	1.930	2.210			1.208	1.509
Female	Married	17	2.013	2.013	1.613	1.920	2.210			1.208	1.509
Female	Married	18	2.013	2.013	1.604	1.910	2.200			1.208	1.502
Female	Married	19	1.870	1.870	1.170	1.170	1.440			1.100	1.449
Female	Married	20	1.760	1.760	1.150	1.150	1.440			1.100	1.313
Female	Married	21	1.430	1.430	1.090	1.090	1.180			0.920	1.180
Female	Married	22	1.430	1.430	1.070	1.070	1.180			0.920	1.180
Female	Married	23	1.430	1.430	0.940	0.940	1.180			0.920	1.180
Female	Married	24	1.287	1.287	0.920	0.920	1.180			0.931	1.180
Female	Married	25	0.990	0.990	0.910	0.910	0.856			0.882	1.040
Female	Married	26	0.990	0.990	0.910	0.910	0.856			0.882	0.950
Female	Married	27	0.990	0.990	0.910	0.910	0.856			0.872	0.950
Female	Married	28	0.990	0.990	0.910	0.910	0.856			0.872	0.942
Female	Married	29	0.990	0.990	0.910	0.910	0.856			0.872	0.935
Female	Married	30	0.982	0.982	0.908	0.908	0.856			0.865	0.927
Female	Married	31	0.975	0.965	0.906	0.906	0.856			0.859	0.920
Female	Married	32	0.967	0.960	0.904	0.904	0.856			0.853	0.912
Female	Married	33	0.959	0.950	0.902	0.902	0.856			0.846	0.904
Female	Married	34	0.952	0.940	0.900	0.900	0.856			0.840	0.897
Female	Married	35	0.944	0.935	0.898	0.898	0.856			0.834	0.889
Female	Married	36	0.936	0.936	0.896	0.896	0.856			0.827	0.882
Female	Married	37	0.928	0.928	0.894	0.894	0.856			0.821	0.874
Female	Married	38	0.921	0.921	0.892	0.892	0.856			0.815	0.866
Female	Married	39	0.913	0.913	0.890	0.890	0.856			0.809	0.859
Female	Married	40	0.905	0.905	0.888	0.888	0.856			0.803	0.851



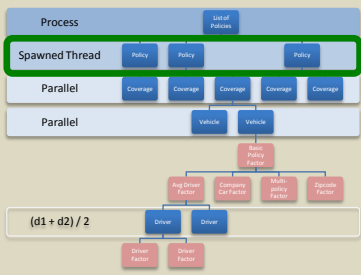
Why can't it be faster?

CALCULATION OF INDIVIDUAL DRIVER FACTOR
(one calculation is necessary for each driver on the policy)
Round to three (3) decimal places after each step.

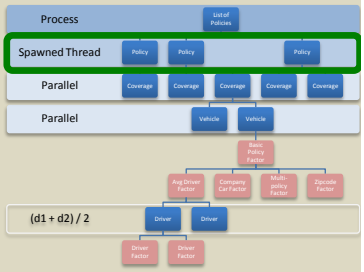
STEP	Description	BI	PD	MP	UMBI	UMPD	OTC	Coll
1	Driver Class Factor							
2	Insurance Score Factor	×	×	×	×	×	×	×
3	Tier Factor	×	×	×	×	×	×	×
4	Accident Factor	×	×	×	×	×	×	×
5	DWI Factor	×	×	×	×	×	×	×
6	Major Violation Factor	×	×	×	×	×	×	×
7	Minor Violation Factor	×	×	×	×	×	×	×
8	Good Student Factor	×	×	×	×	×	×	×
9	Defensive Driver Factor	×	×	×	×	×	×	×
10	Good Partner Factor	×	×	×	×	×	×	×
11	Grange Life Insurance Factor	×	×	×	×	×	×	×
12	Away At School Factor	×	×	×	×	×	×	×
13	Individual Driver Factor	=	=	=	=	=	=	=



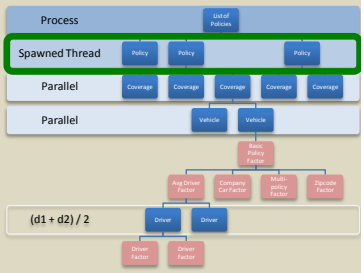

```
let rec Rate (policies: System.Collections.Generic.List<Policy>) =
    let policies = List.of_seq policies
    let rec exec = function
        |[] -> ()
        |p::tail -> async{
                      calcCoverages(p, GetCurrentRates(p), GetCoverages(p))
                    } |> Async.Spawn
                    exec tail
    exec policies
```



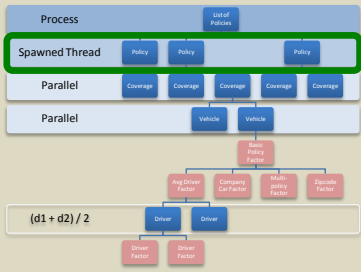
```
let Rate (policies: System.Collections.Generic.List<Policy>) =
    let policies = List.of_seq policies
    let rec exec = function
        |[] -> ()
        |p::tail -> async{
                      calcCoverages(p, GetCurrentRates(p), GetCoverages(p))
                    } |> Async.Spawn
                    exec tail
    exec policies
```



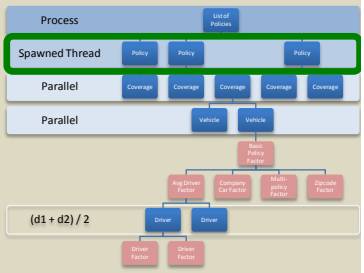

```
let rec Rate (policies: System.Collections.Generic.List<Policy>) =
    let policies = List.of_seq policies
    let rec exec = function
        |[] -> ()
        |p::tail -> async{
                      calcCoverages(p, GetCurrentRates(p), GetCoverages(p))
                    } |> Async.Spawn
                    exec tail
    exec policies
```



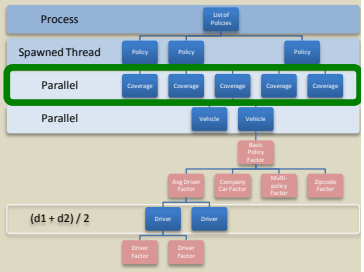
```
let rec Rate (policies: System.Collections.Generic.List<Policy>) =
    let policies = List.of_seq policies
    let rec exec = function
        |[] -> ()
        |p::tail -> async{
                      calcCoverages(p, GetCurrentRates(p), GetCoverages(p))
                    } |> Async.Spawn
                    exec tail
    exec policies
```



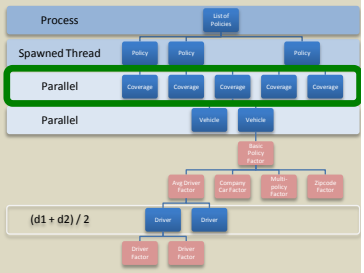

```
let rec Rate (policies: System.Collections.Generic.List<Policy>) =
    let policies = List.of_seq policies
    let rec exec = function
        |[] -> ()
        |p::tail -> async{
                      calcCoverages(p, GetCurrentRates(p), GetCoverages(p))
                    } |> Async.Spawn
                    exec tail
    exec policies
```

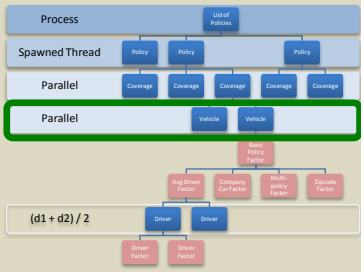
```
let rec calcCoverages (policy:Policy) rates coverages = function
    | [] -> ()
    | c::tail -> updatePolicy(policy (Async.Run(GetRates policy rates c)))
                 calcCoverages policy rates tail
```



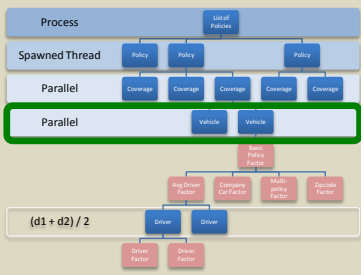

```
let rec calcCoverages (policy:Policy) rates coverages = function
    | [] -> ()
    | c::tail -> updatePolicy(policy (Async.Run(GetRates policy rates c)))
                 calcCoverages policy rates tail
```

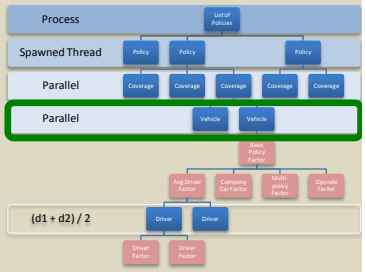
```
let GetRates (policy:Policy) rates coverage =
    async{
        let! results = Async.Parallel[
            for v in Policy.vehicles ->
                async{return (Result policy v coverage rates).Force()}
        ]
        return (coverage, Seq.reduce(+) results)}
```



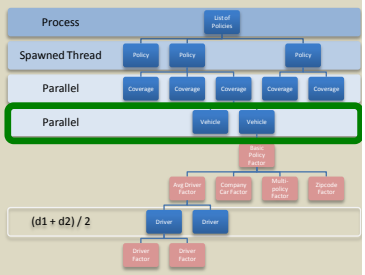

```
let GetRates (policy:Policy) rates coverage =
    async{
        let! results = Async.Parallel[
            for v in Policy.vehicles ->
                async{return (Result policy v coverage rates).Force()}
        ]
        return (coverage, Seq.reduce(+) results)}
```

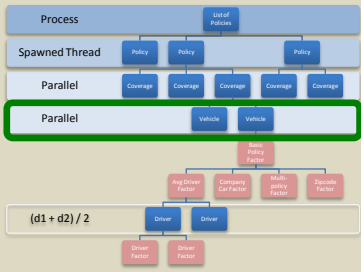
```
let GetRates (policy:Policy) rates coverage =
    async{
        let! results = Async.Parallel[
            for v in Policy.vehicles ->
                async{return (Result policy v coverage rates).Force()}
        ]
        return (coverage, Seq.reduce(+) results)}
```



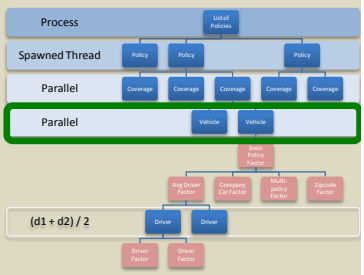

```
let GetRates (policy:Policy) rates coverage =
    async{
        let! results = Async.Parallel[
            for v in Policy.vehicles ->
                async{return (Result policy v coverage rates).Force()}
        ]
        return (coverage, Seq.reduce(+) results)}
```



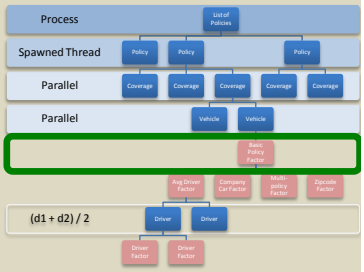
```
let GetRates (policy:Policy) rates coverage =
    async{
        let! results = Async.Parallel[
            for v in Policy.vehicles ->
                async{return (Result policy v coverage rates).Force()}
        ]
        return (coverage, Seq.reduce(+) results)}
```



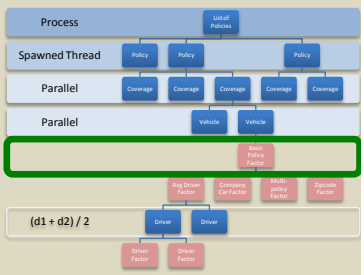

```
let GetRates (policy:Policy) rates coverage =
    async{
        let! results = Async.Parallel[
            for v in Policy.vehicles ->
                async{return (Result policy v coverage rates).Force()}
        ]
        return (coverage, Seq.reduce(+) results)}
```



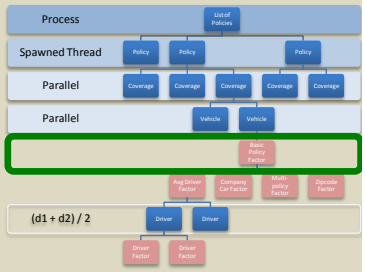
```
let Result (p:Policy)(v:Vehicle)(c:AutoCoverage)(r:CurrentRates) =
    lazy(
        let QueryRate(rq:RateQuery) =
                                    rq.Coverage <- c.ToInt()
                                    r.GetRate(rq)
                .
                .
                .
        BasicPolicyFactor()
        )
```



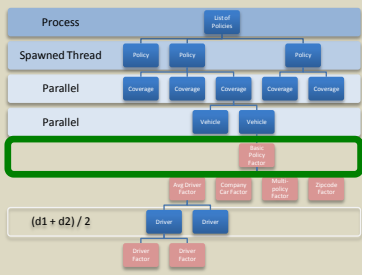

```
let Result (p:Policy)(v:Vehicle)(c:AutoCoverage)(r:CurrentRates) =
    lazy(
        let QueryRate(rq:RateQuery) =
                                    rq.Coverage <- c.ToInt()
                                    r.GetRate(rq)
                .
                .
                .
        BasicPolicyFactor()
        )
```

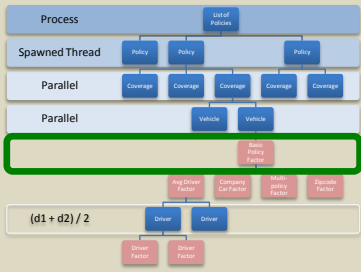
```
let Result (p:Policy)(v:Vehicle)(c:AutoCoverage)(r:CurrentRates) =
    lazy(
        let QueryRate(rq:RateQuery) =
                                    rq.Coverage <- c.ToInt()
                                    r.GetRate(rq)
                .
                .
                .
        BasicPolicyFactor()
        )
```



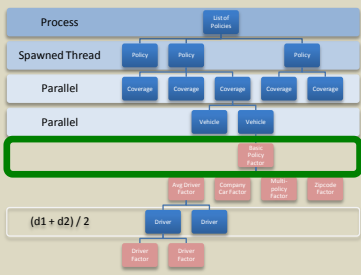

```
let Result (p:Policy)(v:Vehicle)(c:AutoCoverage)(r:CurrentRates) =
    lazy(
        let QueryRate(rq:RateQuery) =
                                    rq.Coverage <- c.ToInt()
                                    r.GetRate(rq)
                .
                .
                .
        BasicPolicyFactor()
        )
```

```
let Result (p:Policy)(v:Vehicle)(c:AutoCoverage)(r:CurrentRates) =
    lazy(
        let QueryRate(rq:RateQuery) =
                                    rq.Coverage <- c.ToInt()
                                    r.GetRate(rq)
                .
                .
                .
        BasicPolicyFactor()
        )
```

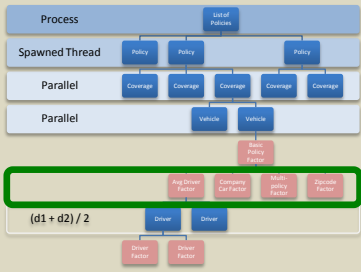



```
let Result (p:Policy)(v:Vehicle)(c:AutoCoverage)(r:CurrentRates) =
    lazy(
        let QueryRate(rq:RateQuery) =
                                    rq.Coverage <- c.ToInt()
                                    r.GetRate(rq)
                .
                .
                .
        BasicPolicyFactor()
        )
```

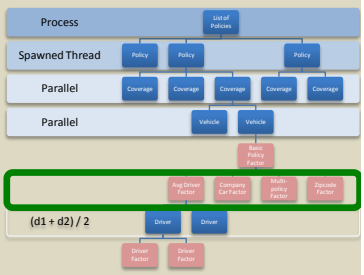


```
let BasicPolicyFactor() =
    lazy(
        Seq.map force [AverageDriverFactor();
                       MultiPolicyCombinationDiscount();
                       CompanyCarDiscount();]
                       |> Seq.reduce (*)
        )
```

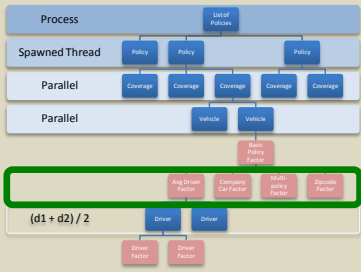
(Lazy<decimal>)




```
let BasicPolicyFactor() =
    lazy(
        Seq.map force [AverageDriverFactor();
                       MultiPolicyCombinationDiscount();
                       CompanyCarDiscount();]
                       |> Seq.reduce (*)
        )
```

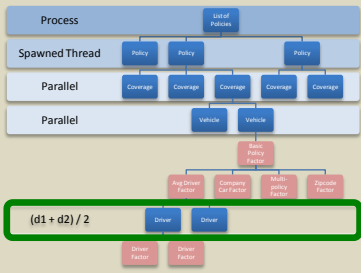



```
let BasicPolicyFactor() =
    lazy(
        Seq.map force [AverageDriverFactor();
                       MultiPolicyCombinationDiscount();
                       CompanyCarDiscount();]
                       |> Seq.reduce (*)
        )
```

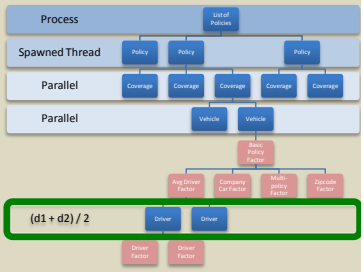



```
let AverageDriverFactor() =
    lazy(
        let mutable accum = (decimal)0
        for d in p.Drivers do
            accum <- accum + (DriverFactor d).Force()
            accum / (decimal)p.Drivers.Count)
```

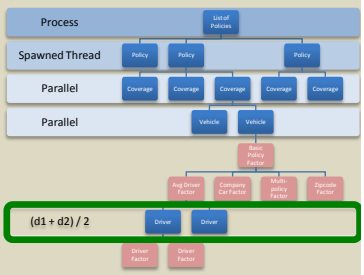
(unit -> Lazy<decimal>)



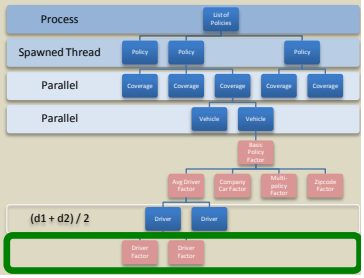
```
let AverageDriverFactor() =
    lazy(
        let mutable accum = (decimal)0
        for d in p.Drivers do
            accum <- accum + (DriverFactor d).Force()
            accum / (decimal)p.Drivers.Count)
```



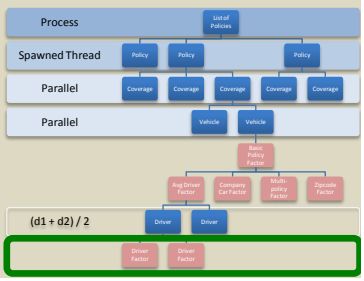

```
let AverageDriverFactor() =
    lazy(
        let mutable accum = (decimal)0
        for d in p.Drivers do
            accum <- accum + (DriverFactor d).Force()
            accum / (decimal)p.Drivers.Count)
```



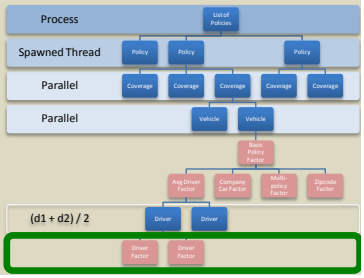
```
let DriverFactor d =
    lazy(
        [BaseRate d;
        DriverClassFactor d;
        InsuranceScoreFactor d;
        GoodStudentFactor d;
        GrangeLifeFactor d;
        AwayAtSchoolFactor d]
        |> Seq.map force
        |> Seq.reduce (*)
        )
```



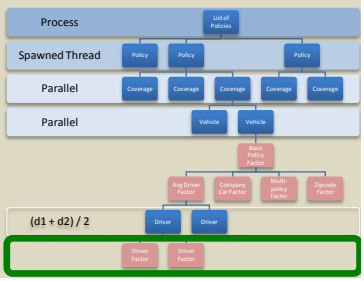

```
let DriverFactor d =
    lazy(
        [BaseRate d;
        DriverClassFactor d;
        InsuranceScoreFactor d;
        GoodStudentFactor d;
        GrangeLifeFactor d;
        AwayAtSchoolFactor d]
        |> Seq.map force
        |> Seq.reduce (*)
        )
```

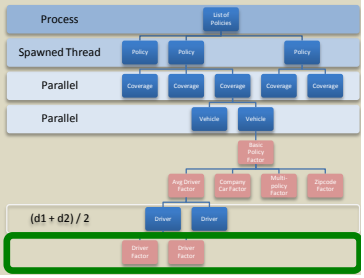
```
let DriverFactor d =
    lazy(
        [BaseRate d;
        DriverClassFactor d;
        InsuranceScoreFactor d;
        GoodStudentFactor d;
        GrangeLifeFactor d;
        AwayAtSchoolFactor d]
        |> Seq.map force
        |> Seq.reduce (*)
        )
```



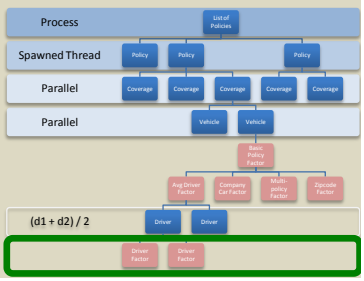

```
let InsuranceScoreFactor (d:Driver) =
    let rq = new RateQuery(RateTableType.CreditScoreFactor)
    rq.Age <- Common.CalculateDriverAge(d.BirthDate, p)
    let rec scoreCalc() =
                        match d.InsuranceScoreAttribute.ToString() with
                        | "CBRN" when score < 2 -> -2
                        | "CBRS" when score < 2 -> -3
                        | "CBRW" when score < 2 -> -4
                        | "CBRV" when score < 2 -> -5
                        | "CBRX" when score < 2 -> -6
                        | _ -> d.InsuranceScore
    rq.CreditScore <- scoreCalc()
    QueryRate rq
```



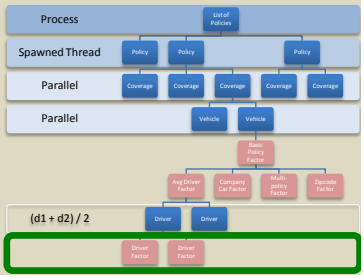
```
let InsuranceScoreFactor (d:Driver) =
    let rq = new RateQuery(RateTableType.CreditScoreFactor)
    rq.Age <- Common.CalculateDriverAge(d.BirthDate, p)
    let rec scoreCalc() =
                        match d.InsuranceScoreAttribute.ToString() with
                        | "CBRN" when score < 2 -> -2
                        | "CBRS" when score < 2 -> -3
                        | "CBRW" when score < 2 -> -4
                        | "CBRV" when score < 2 -> -5
                        | "CBRX" when score < 2 -> -6
                        | _ -> d.InsuranceScore
    rq.CreditScore <- scoreCalc()
    QueryRate rq
```



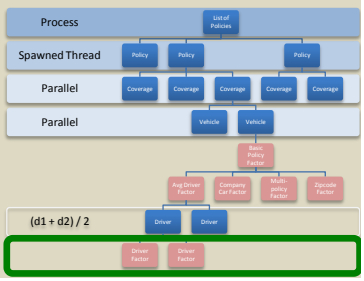

```
let InsuranceScoreFactor (d:Driver) =
    let rq = new RateQuery(RateTableType.CreditScoreFactor)
    rq.Age <- Common.CalculateDriverAge(d.BirthDate, p)
    let rec scoreCalc() =
                        match d.InsuranceScoreAttribute.ToString() with
                        | "CBRN" when score < 2 -> -2
                        | "CBRS" when score < 2 -> -3
                        | "CBRW" when score < 2 -> -4
                        | "CBRV" when score < 2 -> -5
                        | "CBRX" when score < 2 -> -6
                        | _ -> d.InsuranceScore
    rq.CreditScore <- scoreCalc()
    QueryRate rq
```



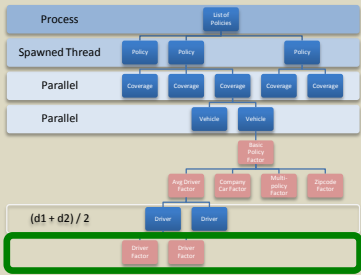
```
let InsuranceScoreFactor (d:Driver) =
    let rq = new RateQuery(RateTableType.CreditScoreFactor)
    rq.Age <- Common.CalculateDriverAge(d.BirthDate, p)
    let rec scoreCalc() =
                        match d.InsuranceScoreAttribute.ToString() with
                        | "CBRN" when score < 2 -> -2
                        | "CBRS" when score < 2 -> -3
                        | "CBRW" when score < 2 -> -4
                        | "CBRV" when score < 2 -> -5
                        | "CBRX" when score < 2 -> -6
                        | _ -> d.InsuranceScore
    rq.CreditScore <- scoreCalc()
    QueryRate rq
```



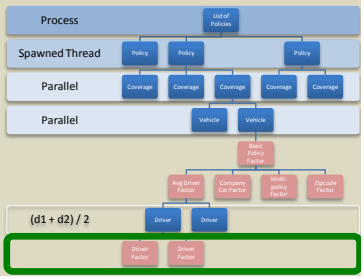

```
let InsuranceScoreFactor (d:Driver) =
    let rq = new RateQuery(RateTableType.CreditScoreFactor)
    rq.Age <- Common.CalculateDriverAge(d.BirthDate, p)
    let rec scoreCalc() =
                        match d.InsuranceScoreAttribute.ToString() with
                        | "CBRN" when score < 2 -> -2
                        | "CBRS" when score < 2 -> -3
                        | "CBRW" when score < 2 -> -4
                        | "CBRV" when score < 2 -> -5
                        | "CBRX" when score < 2 -> -6
                        | _ -> d.InsuranceScore
    rq.CreditScore <- scoreCalc()
    QueryRate rq
```



```
let DriverFactor d =
    lazy(
        [BaseRate d;
        DriverClassFactor d;
        InsuranceScoreFactor d;
        GoodStudentFactor d;
        GrangeLifeFactor d;
        AwayAtSchoolFactor d]
        |> Seq.map force
        |> Seq.reduce (*)
        )
```

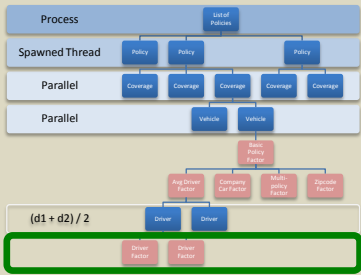



```
let DriverFactor d =
    lazy(
        [BaseRate d;
        DriverClassFactor d;
        InsuranceScoreFactor d;
        GoodStudentFactor d;
        GrangeLifeFactor d;
        AwayAtSchoolFactor d]
        |> Seq.map force
        |> Seq.reduce (*)
        )
```



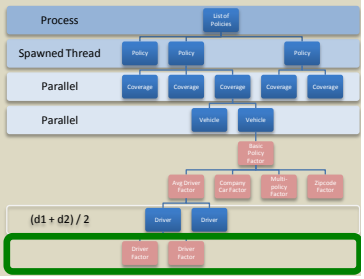
```
let DriverFactor d =
    lazy(
        [BaseRate d;
        DriverClassFactor d;
        InsuranceScoreFactor d;
        GoodStudentFactor d;
        GrangeLifeFactor d;
        AwayAtSchoolFactor d]
        |> Seq.map force
        |> Seq.reduce (*)
        )
```

```
let force (x:Lazy<x>) = x.Force()
```

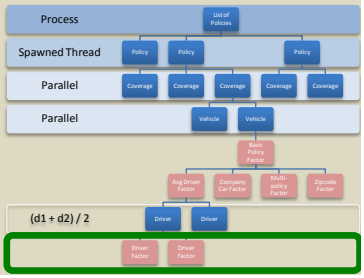



```
let DriverFactor d =
    lazy(
        [BaseRate d;
        DriverClassFactor d;
        InsuranceScoreFactor d;
        GoodStudentFactor d;
        GrangeLifeFactor d;
        AwayAtSchoolFactor d]
        |> Seq.map force
        |> Seq.reduce (*)
        )
```

decimal list

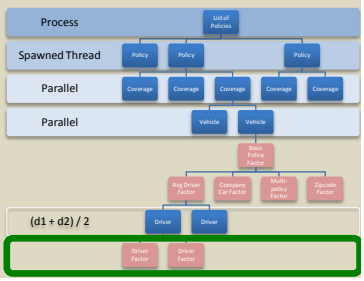


```
let DriverFactor d =
    lazy(
        [BaseRate d;
        DriverClassFactor d;
        InsuranceScoreFactor d;
        GoodStudentFactor d;
        GrangeLifeFactor d;
        AwayAtSchoolFactor d]
        |> Seq.map force
        |> Seq.reduce (*)
        )
```

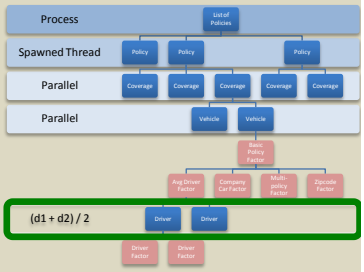



```
let DriverFactor d =
    lazy(
        [BaseRate d;
        DriverClassFactor d;
        InsuranceScoreFactor d;
        GoodStudentFactor d;
        GrangeLifeFactor d;
        AwayAtSchoolFactor d]
        |> Seq.map force
        |> Seq.reduce (*)
        )
```

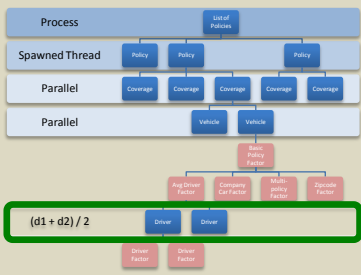
decimal



```
let AverageDriverFactor() =
    lazy(
        let mutable accum = (decimal)0
        for d in p.Drivers do
            accum <- accum + (DriverFactor d).Force()
            accum / (decimal)p.Drivers.Count
        )
```

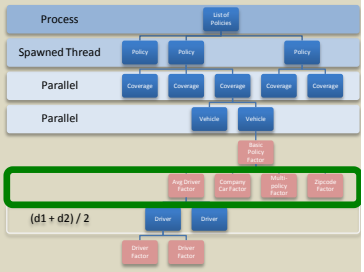



```
let AverageDriverFactor() =
    lazy(
        let mutable accum = (decimal)0
        for d in p.Drivers do
            accum <- accum + (DriverFactor d).Force()
            accum / (decimal)p.Drivers.Count
        )
```

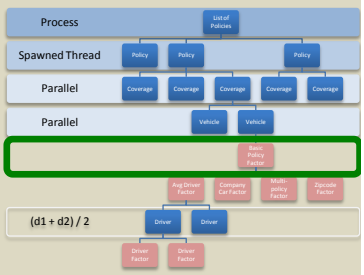



```
let BasicPolicyFactor() =
    lazy(
        Seq.map force [AverageDriverFactor();
                       MultiPolicyCombinationDiscount();
                       CompanyCarDiscount();]
                       |> Seq.reduce (*)
        )
```

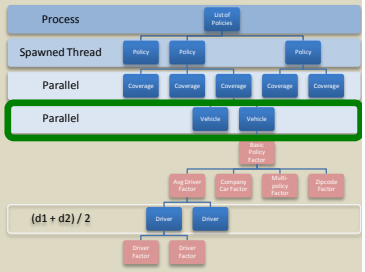
(unit -> Lazy<decimal>)



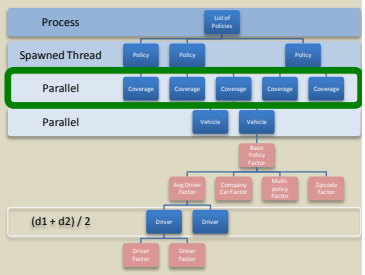

```
let Result (p:Policy)(v:Vehicle)(c:AutoCoverage)(r:CurrentRates) =
    lazy(
        let QueryRate(rq:RateQuery) =
                                rq.Coverage <- c.ToInt()
                                r.GetRate(rq)
                .
                .
                .
        BasicPolicyFactor()
        )
```



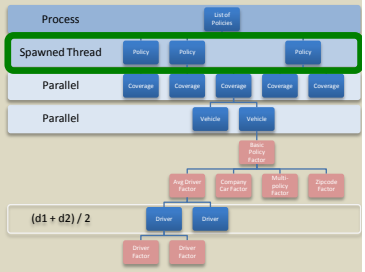
```
let GetRates (policy:Policy) rates coverage =
    async{
        let! results = Async.Parallel[
            for v in Policy.vehicles ->
                async{return (Result policy v coverage rates).Force()}
        ]
        return (coverage, Seq.reduce(+) results)}
```

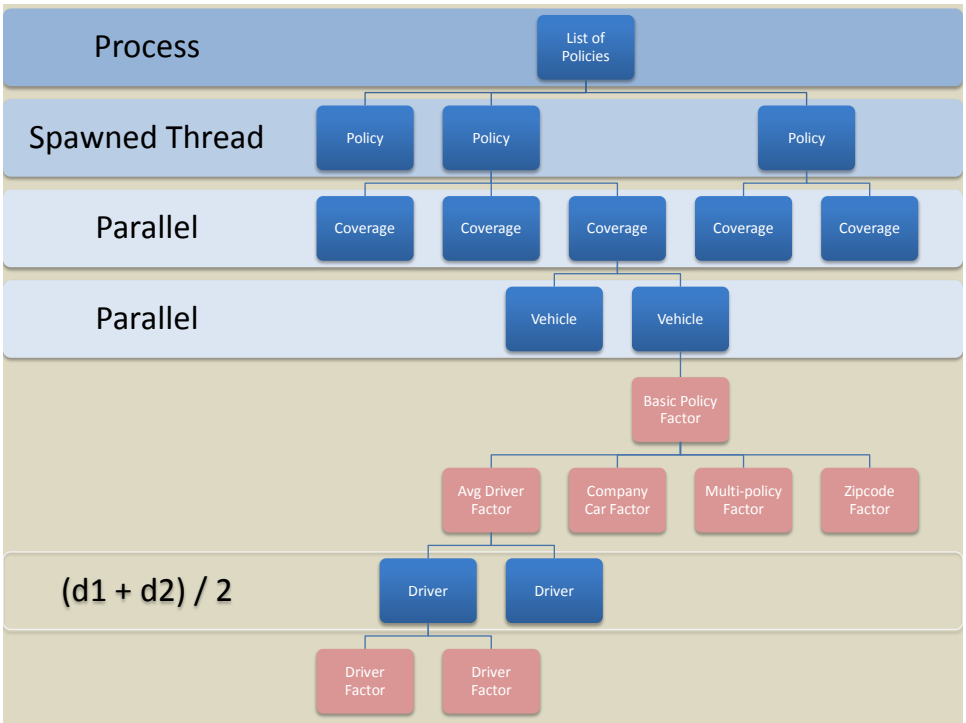



```
let CalcCoverages (policy:Policy) rates coverages = function
    | [] -> ()
    | c::tail -> updatePolicy(policy, (Async.Run(GetRates policy rates c)))
```



```
let rec Rate (policies: System.Collections.Generic.List<Policy>) =
 let policies = List.of_seq policies
 let rec exec = function
        |[] -> ()
        |p::tail -> async{
                        calcCoverages(p, GetCurrentRates(p), GetCoverages(p))
                        } |> Async.Spawn
                    exec tail
 exec policies
```





	1	5	10	20	40	50
C#	3	10	15	38	78	95
F#	1	4	7	15	20	35

```
#light
let RealWorld F# = Concurrent applications
```

Amanda Laucher
Amanda.Laucher@SophicGroup.net

pandamonial.com
Twitter: pandamonial