# F# for the Analyst and the High-Performance Engineer

By Matthew Crews

SDI

# Who am I?


# F# for the Analyst

# Defining Types is Easy

```
type Chicken =
    {
        Name: string
        Size: float
        Age: int
    }
```

```
type Animal =
    | Chicken of Chicken
    | Turkey of Turkey
    | Goose of Goose
```

# Type Inference

```fsharp
let myFunction a b c = // int -> int -> int -> int
    a + b / c


module Chicken =

    let create name age size = // string -> int -> float -> Chicken
        {
            Name = name
            Age = age
            Size = size
        }
```

# Pattern Matching

```
let getName a = // Animal -> string
    match a with
    | Chicken
    | Turkey
    | Goose g

let checkBoun
    match x w
    | a when a < 0.0 -> Negative
```

Incomplete pattern matches on this expression. F# Compiler(25)

val x: float

Open the documentation

*Full name: x*

*Assembly: CodeExamples*

View Problem (Alt+F8)  No quick fixes available

# Type Providers

```
Name, Age, Size
Clucky, 1, 2.0
Gobble, 10, 14.0
Scramble, 2, 3.5
```

```
#r "nuget: FSharp.Data"
open FSharp.Data

type ChickenRow = CsvProvider<"ExampleData.csv">

let chickens = // CsvProvider<...>
    ChickenRow.Load("ExampleData.csv")

for row in chickens.Rows do
    printfn $"{row.Name}"
```

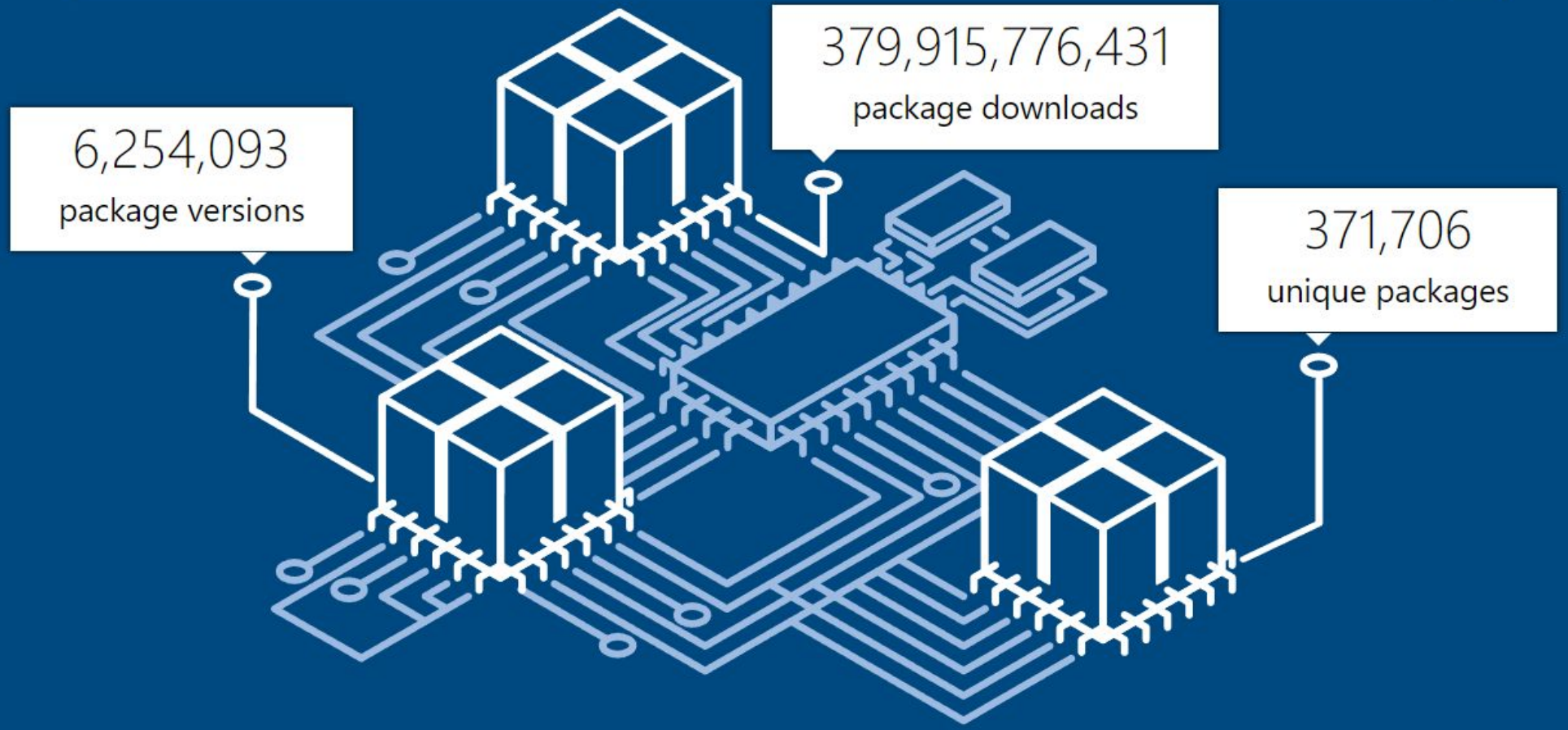
# Computation Expressions

```
let model =
    Model "Coffee" {
        structure [
            supply1 ⟶ cartoner1
            supply2 ⟶ cartoner2
            supply3 ⟶ cartoner3
            supply4 ⟶ cartoner4
            cartoner1 ⟶ cartonerMerge
            cartoner2 ⟶ cartonerMerge
            cartoner3 ⟶ cartonerMerge
            cartoner4 ⟶ cartonerMerge
            cartonerMerge ⟶ casePackerFeed
            casePackerFeed ⟶ casePacker
            casePacker ⟶ palletizerFeed
            palletizerFeed ⟶ palletizer
            palletizer ⟶ wrapperFeed
            wrapperFeed ⟶ wrapper
            wrapper ⟶ endingInventory
        ]
        limits [
            cartoner1, 2.0
            cartoner2, 2.0
            cartoner3, 2.0
            cartoner4, 2.0
            casePacker, 2.3
            palletizer, 2.6
            wrapper, 2.9
        ]
        capacities [
            supply1, infVolume
            supply2, infVolume
            supply3, infVolume
            supply4, infVolume
            casePackerFeed, 10.0
            palletizerFeed, 10.0
            wrapperFeed, 10.0
            endingInventory, infVolume
        ]
        levels [
            supply1, infVolume
            supply2, infVolume
            supply3, infVolume
            supply4, infVolume
        ]
        mergeRules [
            cartonerMerge, (MergeRule.Priority [
                cartoner1; cartoner2; cartoner3; cartoner4
            ])
        ]
```

nuget Packages Upload Statistics Documentation Downloads Blog Sign i

# Create .NET apps faster with NuGet

Search for packages...



# Innovation Rate

Time to Develop  x  Time to Execute

# F# for High-Performance

# All of performance comes down to…

# Doing Less, With Less, Predictably

# Doing Less

Minimize the number of CPU instructions

# F# is Statically Typed

F#

```fsharp
let add (a, b) =
    a + b
```

ASM

```
_.add(Int32, Int32)
    L0000: lea eax, [ecx+edx]
    L0003: ret
```

Python

```python
def add(a, b)
  return a + b
```

ASM

```
mov         qword ptr [r11+38h],r14
add         r14,2
jmp         _PyEval_EvalFrameDefault+5872h (07FFDFD04F462h)
mov         rdi,qword ptr [r12-8]
lea         rax,[__ImageBase (07FFDFCE20000h)]
mov         rsi,qword ptr [r12-10h]
sub         r12,8
mov         rdx,rdi
mov         rcx,rsi
call        qword ptr [rax+r13*8+3E7A70h]
```

```
mov     qword ptr [rsp+8],rbx
push    rdi
sub     rsp,30h
xor     r8d,r8d
mov     rdi,rdx
mov     rbx,rcx
call    binary_op1 (07FFDFCF06C70h)
mov     qword ptr [rsp+10h],rbp
mov     qword ptr [rsp+18h],rsi
mov     qword ptr [rsp+20h],rdi
push    r14
sub     rsp,20h
mov     rsi,rdx
movsxd  r9,r8d
mov     rdx,qword ptr [rcx+8]
mov     rbp,rcx
mov     rdi,qword ptr [rdx+60h]
test    rdi,rdi
je      binary_op1+2Fh (07FFDFCF06C9Fh)
mov     rdi,qword ptr [rdi+r9]
mov     rcx,qword ptr [rsi+8]
mov     qword ptr [rsp+30h],rbx
cmp     rcx,rdx
je      binary_op1+55h (07FFDFCF06CC5h)
xor     ebx,ebx
lea     r14,[_Py_NotImplementedStruct (07FFDFD2E6AF0h)]
test    rdi,rdi
je      binary_op1+0ADh (07FFDFCF06D1Dh)
test    rbx,rbx
je      binary_op1+90h (07FFDFCF06D00h)
mov     rdx,rsi
mov     rcx,rbp
call    rdi
mov     rax,qword ptr [rcx+8]
test    dword ptr [rax+0A8h],1000000h
je      long_add+24h (07FFDFCF60434h)
mov     rax,qword ptr [rdx+8]
test    dword ptr [rax+0A8h],1000000h
jne     _PyLong_Add (07FFDFCF60380h)
sub     rsp,28h
mov     r8,rdx
mov     rdx,qword ptr [rcx+10h]
lea     rax,[rdx+1]
cmp     rax,3
jae     _PyLong_Add+3Eh (07FFDFCF603BEh)
mov     r9,qword ptr [r8+10h]
lea     rax,[r9+1]
cmp     rax,3
jae     _PyLong_Add+3Eh (07FFDFCF603BEh)
mov     ecx,dword ptr [rcx+18h]
mov     eax,dword ptr [r8+18h]
imul    rcx,rdx
imul    rax,r9
add     rcx,rax
add     rsp,28h
jmp     _PyLong_FromSTwoDigits (07FFDFCF5AF10h)
push    rdi
sub     rsp,20h
lea     rax,[rcx+5]
mov     rdi,rcx
cmp     rax,105h
ja      _PyLong_FromSTwoDigits+2Fh (07FFDFCF5AF3Fh)
lea     rax,[rcx+3FFFFFFFh]
cmp     rax,7FFFFFFFh
jae     _PyLong_FromSTwoDigits+48h (07FFDFCF5AF58h)
add     rsp,20h
pop     rdi
jmp     _PyLong_FromMedium (07FFDFCF5AE50h)
mov     qword ptr [rsp+10h],rbx
push    rsi
sub     rsp,20h
movsxd  rsi,ecx
mov     edx,20h
mov     rcx,qword ptr [_PyObject (07FFDFD3B8CA8h)]
call    qword ptr [_PyObject+8h (07FFDFD3B8CB0h)]
push    rbx
sub     rsp,20h
mov     rbx,rdx
call    pymalloc_alloc (07FFDFCF73C80h)
sub     rsp,28h
lea     rax,[rdx-1]
cmp     rax,1FFh
jbe     pymalloc_alloc+17h (07FFDFCF73C97h)
mov     qword ptr [rsp+30h],rbx
mov     qword ptr [rsp+38h],rsi
mov     qword ptr [rsp+40h],rdi
lea     edi,[rdx-1]
shr     edi,4
mov     qword ptr [rsp+20h],r14
lea     r14,[__ImageBase (07FFDFCE20000h)]
lea     eax,[rdi+rdi]
lea     rsi,[rax*8+4C7120h]
mov     rdx,qword ptr [rsi+r14]
mov     rcx,qword ptr [rdx+10h]
cmp     rdx,rcx
je      pymalloc_alloc+9Fh (07FFDFCF73D1Fh)
mov     r9,qword ptr [rdx+8]
inc     dword ptr [rdx]
mov     rax,qword ptr [r9]
mov     qword ptr [rdx+8],rax
test    rax,rax
jne     pymalloc_alloc+1C1h (07FFDFCF73E41h)
mov     r14,qword ptr [rsp+20h]
mov     rax,r9
mov     rdi,qword ptr [rsp+40h]
mov     rsi,qword ptr [rsp+38h]
mov     rbx,qword ptr [rsp+30h]
add     rsp,28h
ret
test    rax,rax
jne     _PyObject_Malloc+46h (07FFDFCF73EA6h)
```

# Can you make Dynamic Typing fast?


**Unladen Swallow** [ edit ]

**Unladen Swallow** was an optimization branch of CPytho
supplementing CPython's custom virtual machine with a

Mojo 🔥 — the programming language for all AI developers.

```
def sort(v: ArraySlice[Int]):
  for i in range(len(v)):
    for j in range(len(v) - i - 1):
      if v[j] > v[j + 1]:
        swap(v[j], v[j + 1])
```

```
struct MyPair:
  var first: Int
  var second: F32

  def __init__(self, first: Int, second: F32):
    self.first = first
    self.second = second
```

# Default Equality

```fsharp
type Chicken =
    {
        Name: string
        Size: float
        Age: int
    }

let equalsTest (a: Chicken, b: Chicken) =
    a = b
```

```
_.equalsTest(Chicken, Chicken)
    L0000: push ebp
    L0001: mov ebp, esp
    L0003: cmp [ecx], cl
    L0005: test edx, edx
    L0007: je short L003d
    L0009: mov ecx, [ecx+4]
    L000c: mov edx, [edx+4]
    L000f: cmp ecx, edx
    L0011: je short L0041
    L0013: test ecx, ecx
    L0015: je short L0023
    L0017: test edx, edx
    L0019: je short L0023
    L001b: mov eax, [ecx+4]
    L001e: cmp eax, [edx+4]
    L0021: je short L0027
    L0023: xor eax, eax
    L0025: jmp short L003f
    L0027: lea eax, [ecx+8]
    L002a: mov ecx, [ecx+4]
    L002d: add ecx, ecx
    L002f: push ecx
    L0030: add edx, 8
    L0033: mov ecx, eax
    L0035: call dword ptr [0x652d738]
    L003b: jmp short L0025
    L003d: xor eax, eax
    L003f: pop ebp
    L0040: ret
    L0041: mov eax, 1
    L0046: jmp short L0025
```

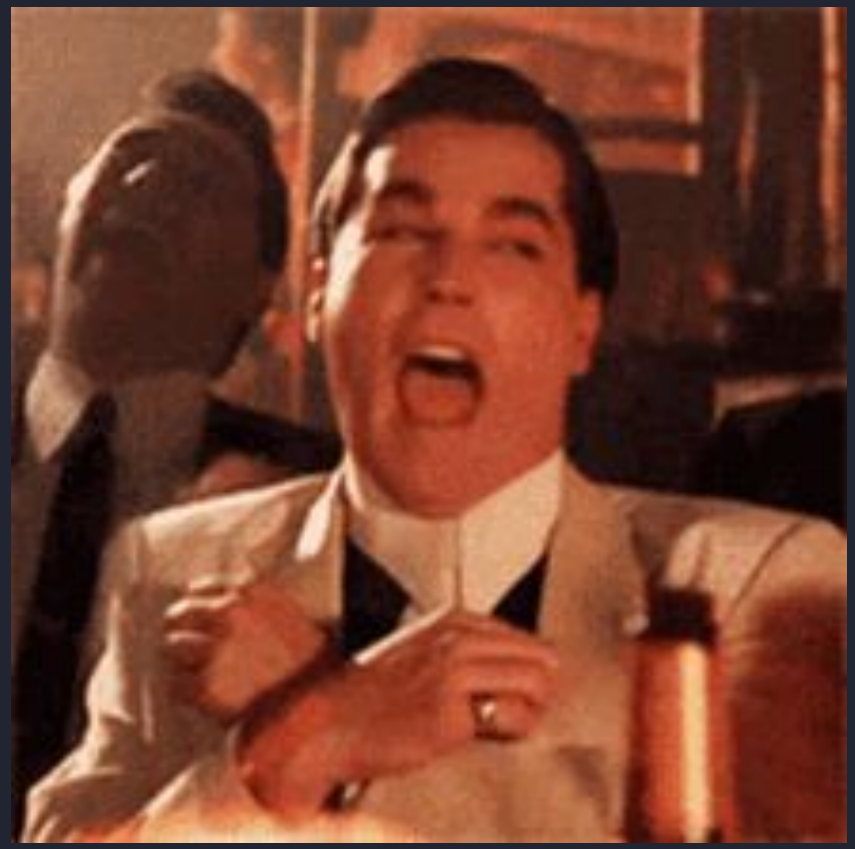
# Units of Measure

```fsharp
[<Measure>]
type ChickenId

type ChickenData =
    {
        Name: string
        Age: int
        Size: float
    }

let equalsTestUoM (a: int<ChickenId>, b: int<ChickenId>) =
    a = b
```

```
_.equalsTestUoM(Int32, Int32)
    L0000: xor eax, eax
    L0002: cmp ecx, edx
    L0004: sete al
    L0007: ret
```

We use numeric primitives to identify Entities.

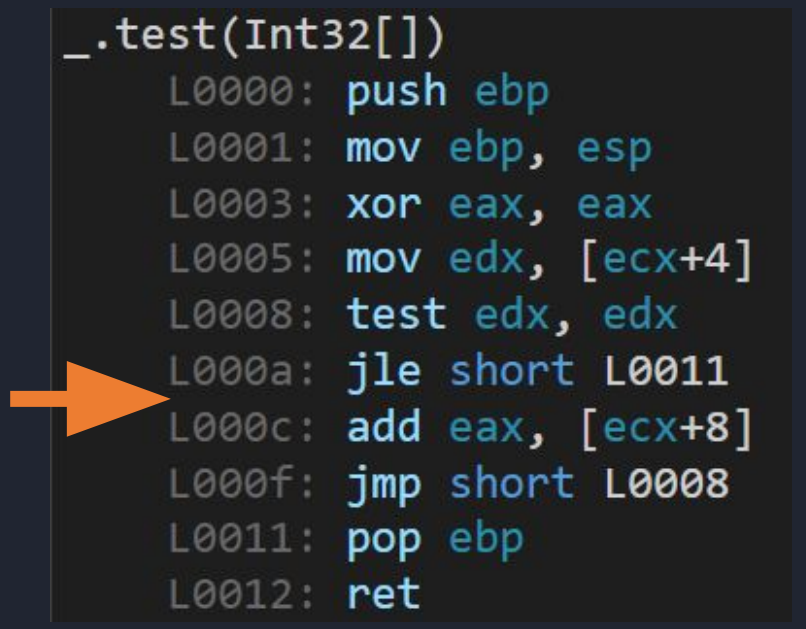
# Bounds Checking



C Devs

# Bounds Checking

```
let test (a: int[]) = // []<int> -> int
    let mutable i = 0
    let mutable acc = 0

    while i < a.Length do

        acc <- acc + a[i]

    acc
```



# With Less

Minimize the memory footprint of your application

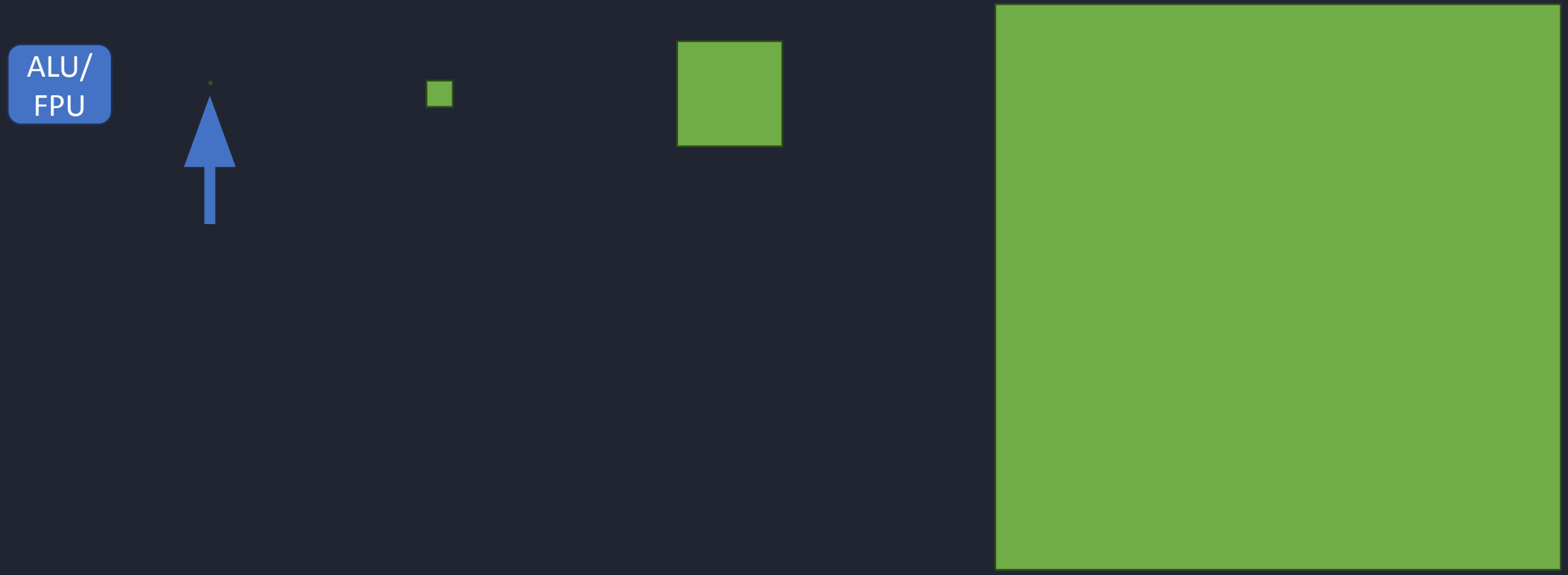
# CPU Memory Architecture

Registers
10-100
64 bit

L1 Cache
80 KB

L2 Cache
1.25 MB

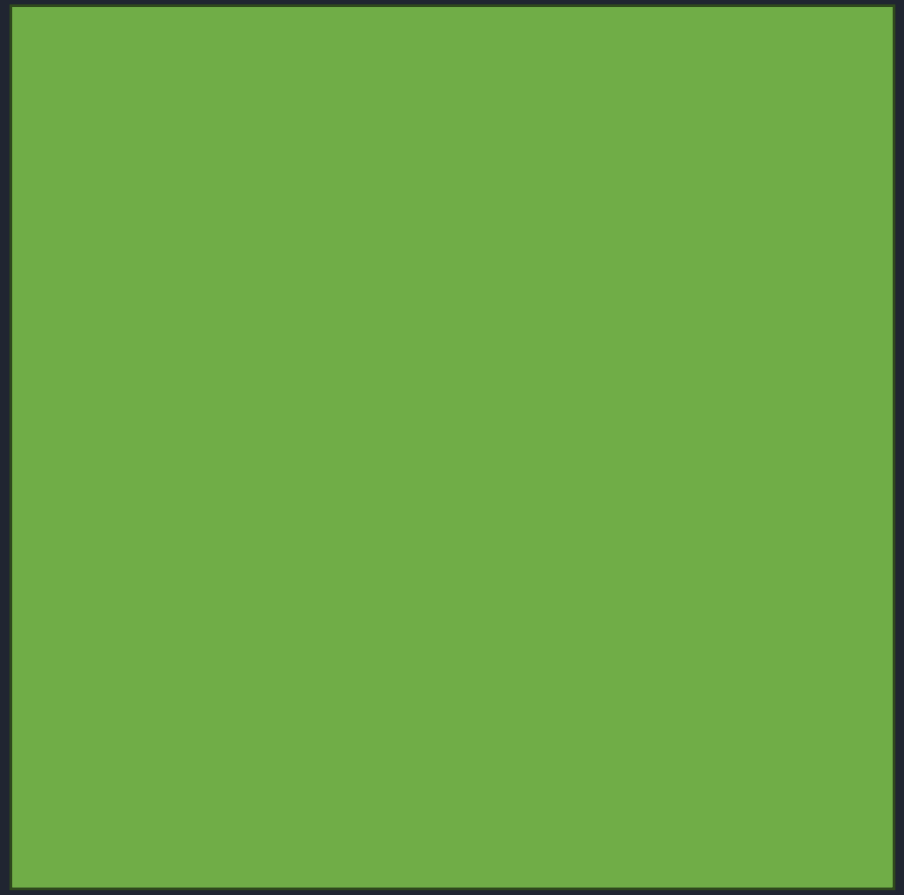
L3 Cache
30 MB Shared



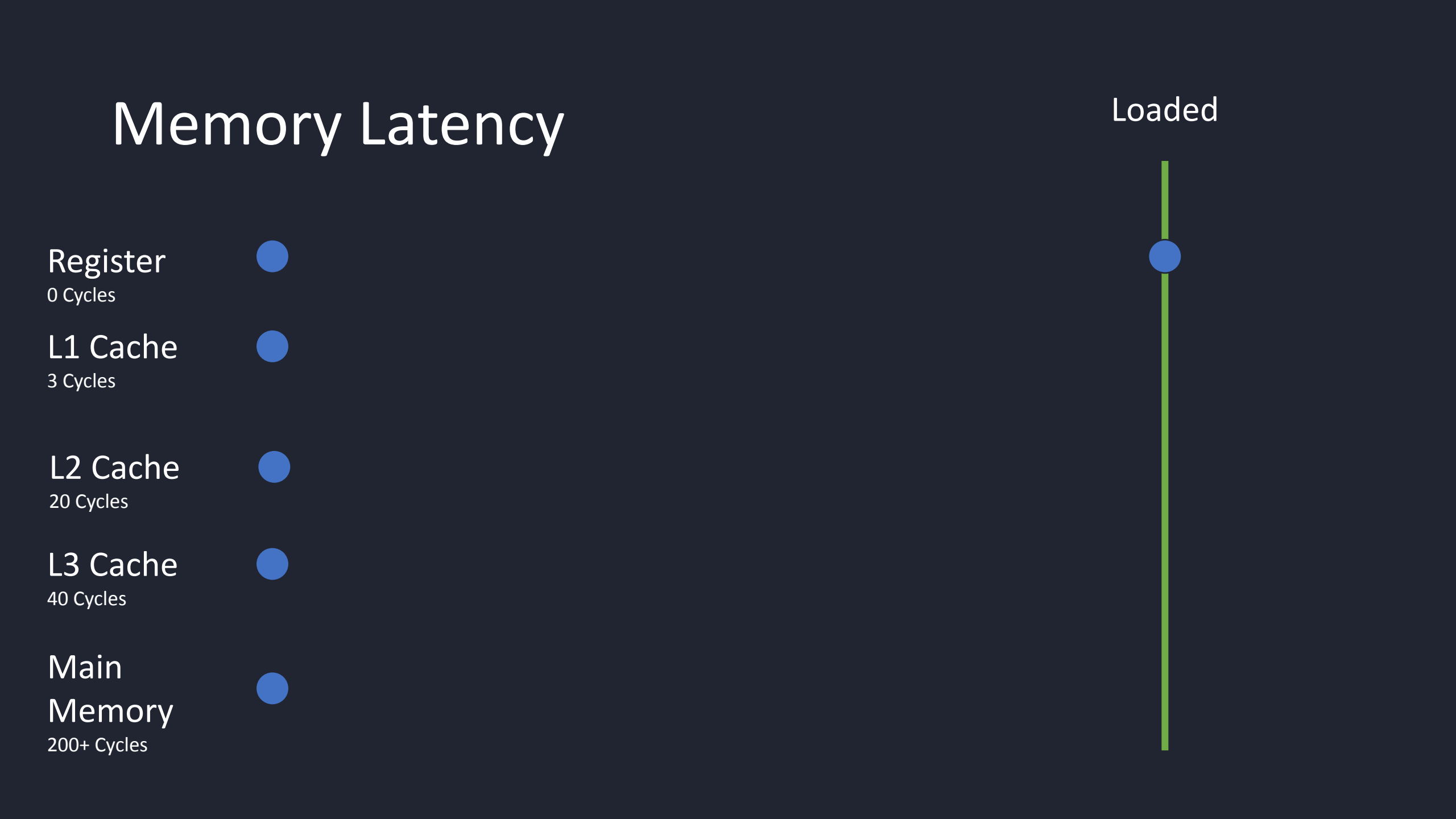
# CPU Memory Architecture

L3 Cache
30 MB

Main Memory
64 GB



# Memory Latency




Cache Space is incredibly scarce

# C Dev hearing your language has a Garbage Collector…

# What the GC Does

- Makes Allocating/Deallocating memory transparent
- Designed to be general-purpose and robust
- Makes writing horrendous code (for performance) easy

# What the GC Does NOT do

- Understand your memory access patterns
- Does not co-locate data that is frequently accessed together
- Understand Allocation/Deallocation patterns (Arena Allocator)

# Row

```fsharp
[<Struct>]
type Row<[<Measure>] 'Measure, 'T>(values: 'T[]) =

    [<EditorBrowsable(EditorBrowsableState.Never)>]
    member _._values : 'T[] = values // []<'T>

    member r.Item
        with inline get (i: int<'Measure>) =
            r._values[int i]

        and inline set (index: int<'Measure>) value =
            r._values[int index] <- value

    member inline r.Length = LanguagePrimitives.Int32WithMeasure<'Measure> r._values.Length
```

# Bar

```fsharp
[<Struct>]
type Bar<[<Measure>] 'Measure, 'T> (values: 'T[]) =

    [<EditorBrowsable(EditorBrowsableState.Never)>]
    member _._values = values // []<'T>

    member inline b.Length = LanguagePrimitives.Int32WithMeasure<'Measure> b._values.Length

    member b.Item // 'T
        with inline get(i: int<'Measure>) =
            b._values[int i]
```

# Row and Bar Usage

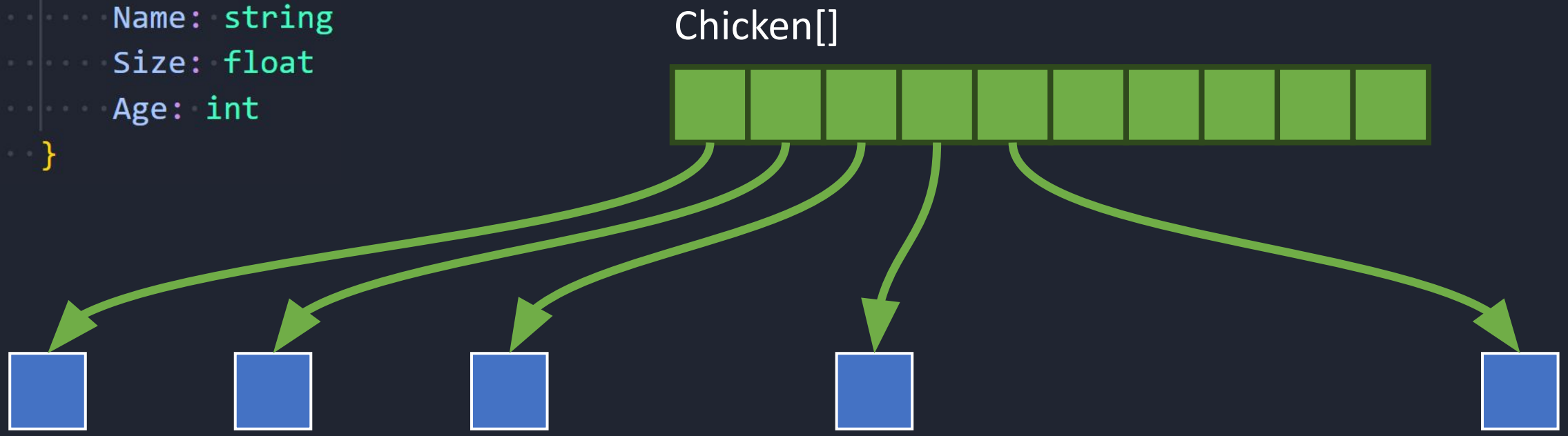
```
let chickenAges = Bar<ChickenId, _> [|0; 1; 5; 0; 2|]
let chickenIndex = 1<ChickenId>
// The Compiler enforces the correct units on the indexing Int
let chickenAge = chickenAges[chickenIndex]
```

# Row and Bar Usage

```fsharp
let chickenAges = Bar<ChickenId, _> [|0; 1; 5; 0; 2|]
let chickenIndex = 1<ChickenId>
// The Compiler enforces the correct units on the indexing Int
let chickenAge = chickenAges[chickenIndex]

// Non-Chicken Index
let cowIndex = 1<CowId>
// This a compiler error since the units of the indexing value
// does not match the expectation of the collection
let cowAge = chickenAges[cowIndex]
```

# Array of Objects

```
type Chicken =
    {
        Name: string
        Size: float
        Age: int
    }
```




Each element could be on a different Cache Line
and therefore incur a cache miss on load

# Struct of Arrays (SoA)

```
type Flock =
    {
        Names: Bar<ChickenId, string>
        Ages: Row<ChickenId, int>
        Sizes: Row<ChickenId, float>
    }
```


Values are contiguous in memory which minimizes the probability of a cache miss

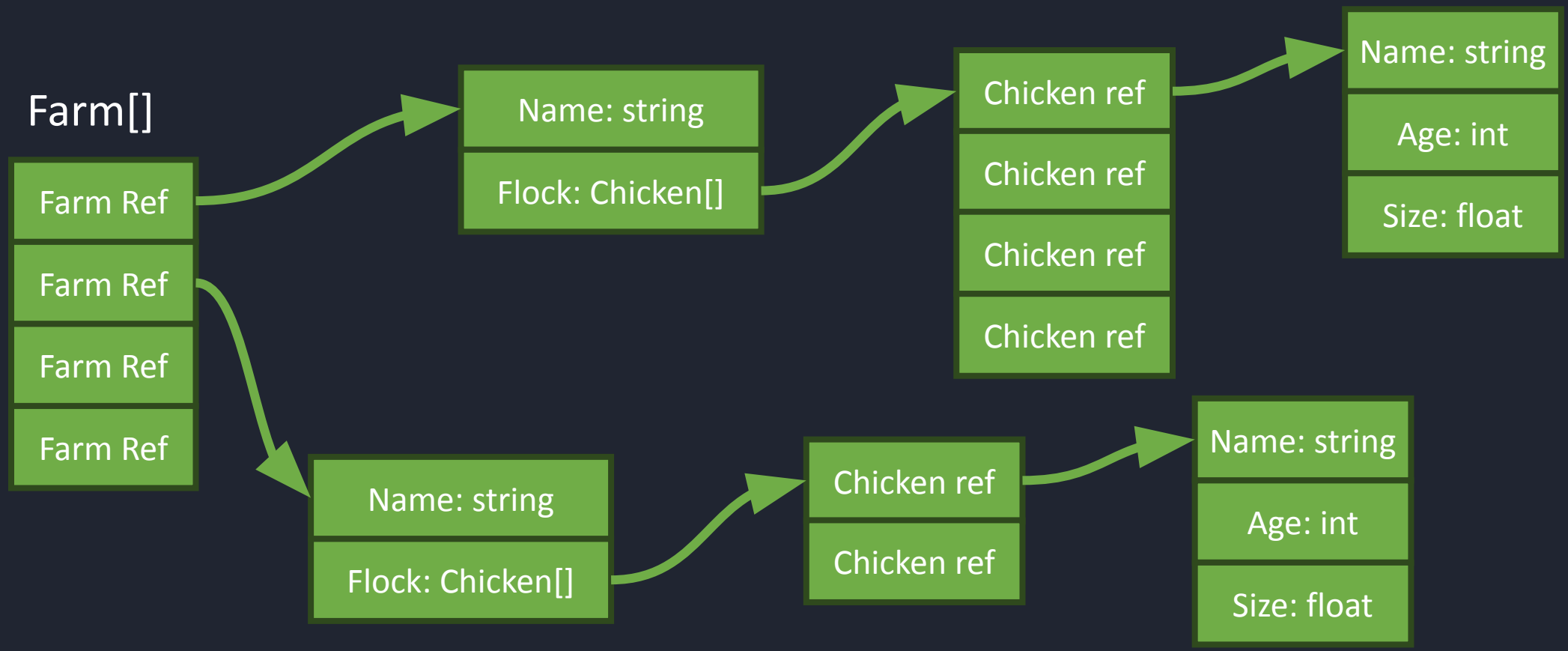
# One to Many Relationships

# One to Many Relationships

```
type Chicken =
    {
        Name: string
        Size: float
        Age: int
    }

type Farm =
    {
        Name: string
        Flock: Chicken[]
    }
```

```
let cooperative =[| // []<Farm>
    {
        Name = "Farm 1"
        Flock = [|
            {
                Name = "Clucky"
                Age = 1
                Size = 1.0
            }
            {
                Name = "Drumstick"
                Age = 2
                Size = 1.0
            }
        |]
    }
|]
```

# One to Many Relationships



# Farm Array Oriented Layout

## Arrays for Farms

Names

| Name1 | Name1 | Name1 | ••• | NameN |
|---|---|---|---|---|

Chickens

| 1<ChickenId> | 0<ChickenId> | 7<ChickenId> | ••• | 3<ChickenId> |
|---|---|---|---|---|

Ranges

| Range1 | Range2 | Range2 | ••• | RangeN |
|---|---|---|---|---|

## Arrays for Chickens

Names

| Name1 | Name1 | Name1 | ••• | NameN |
|---|---|---|---|---|

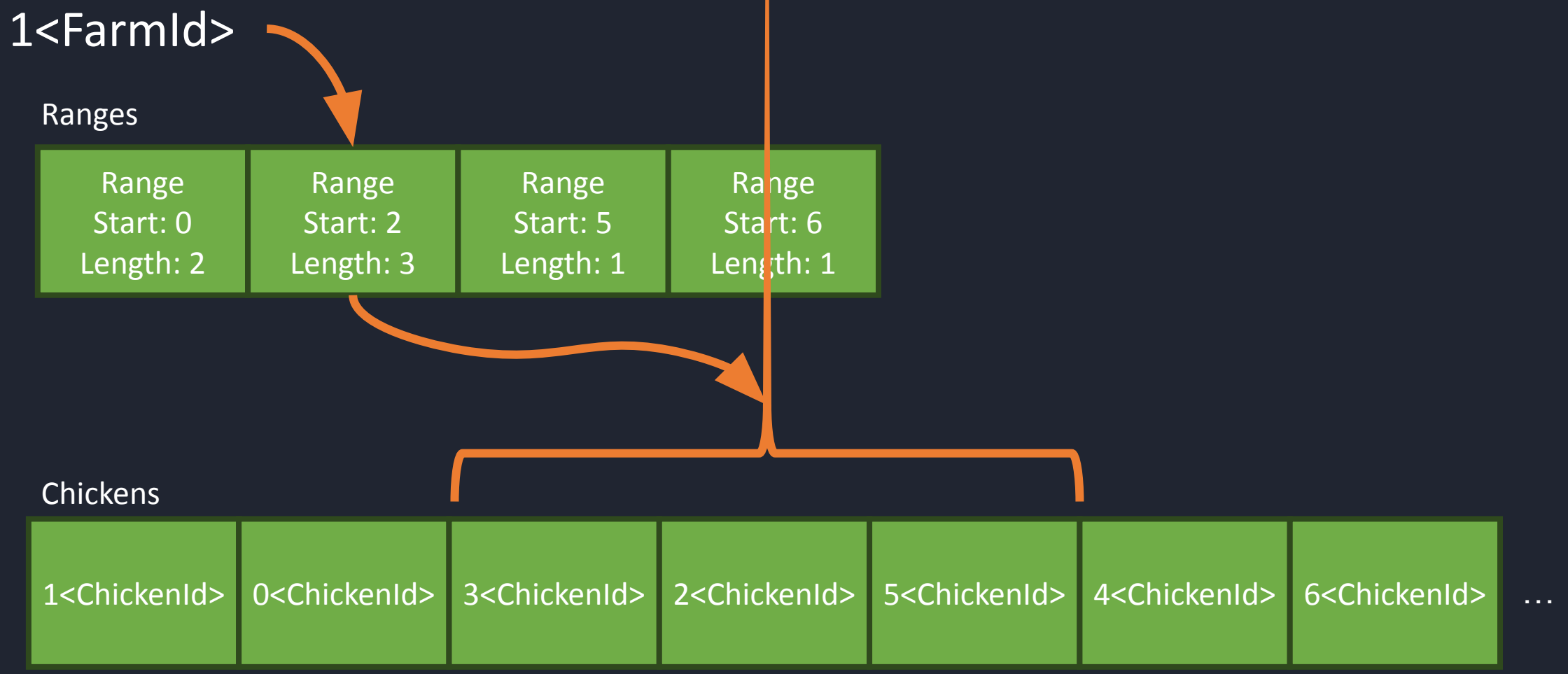
Ages

| 2 | 3 | 1 | ••• | 2 |
|---|---|---|---|---|

Sizes

| 2.0 | 1.5 | 3.3 | ••• | 1.9 |
|---|---|---|---|---|

# One to Many with Range and BarGroup



# Benchmark

- 100 Random Farms
- 100,000 Random Chickens randomly assigned to Farms
- Double the size of the Chickens

# Benchmark

```
// * Summary *

BenchmarkDotNet v0.13.8, Windows 11 (10.0.22621.2134/22H2/2022Update/SunValley2)
13th Gen Intel Core i9-13900K, 1 CPU, 32 logical and 24 physical cores
.NET SDK 8.0.100-preview.7.23376.3
  [Host]     : .NET 8.0.0 (8.0.23.37506), X64 RyuJIT AVX2 DEBUG
  DefaultJob : .NET 8.0.0 (8.0.23.37506), X64 RyuJIT AVX2


| Method      | Mean         | Error     | StdDev    |
|------------ |------------:|----------:|----------:|
| ObjectStyle | 1,021.14 us | 22.755 us | 65.287 us |
| ArrayStyle  |    24.01 us |  0.468 us |  0.481 us |
```

# Benchmark

```
// * Summary *

BenchmarkDotNet v0.13.8, Windows 11 (10.0.22621.2134/22H2/2022Update/SunValley2)
13th Gen Intel Core i9-13900K, 1 CPU, 32 logical and 24 physical cores
.NET SDK 8.0.100-preview.7.23376.3
  [Host]     : .NET 8.0.0 (8.0.23.37506), X64 RyuJIT AVX2 DEBUG
  DefaultJob : .NET 8.0.0 (8.0.23.37506), X64 RyuJIT AVX2


| Method          | Mean       | Error      | StdDev     | Median     |
|---------------- |-----------:|-----------:|-----------:|-----------:|
| ObjectStyle     | 980.407 us | 21.5794 us | 63.6272 us | 963.441 us |
| ArrayStyle      |  24.736 us |  0.4845 us |  0.5184 us |  24.665 us |
| ArraySIMDStyle |   9.864 us |  0.1066 us |  0.0998 us |   9.852 us |
```

# Range and BarGroup

```
[<Struct>]
type BarGroup<[<Measure>] 'Measure, 'Value> internal (ranges: Bar<'Measure, Range>, values: 'Value[]) =

    /// WARNING: This member is public for the purposes of inlining but it is not meant for public consumption.
    [<EditorBrowsable(EditorBrowsableState.Never)>]
    member _._values = values // []<'Value>
    /// WARNING: This member is public for the purposes of inlining but it is not meant for public consumption.
    [<EditorBrowsable(EditorBrowsableState.Never)>]
    member _._ranges = ranges // Bar<'Measure,Range>

    member bg.Item // ReadOnlySpan<'Value>
        with inline get(i: int<'Measure>) =
            let ranges = bg._ranges
            if i < ranges.Length then
                let range = ranges[i]
                ReadOnlySpan(bg._values, range.Start, range.Length)

            else
                ReadOnlySpan()
```

# Other Tools

- BitSet
- ArrayPool
- StackAlloc
- ObjectPools
- StructPools

# Memory Layout is Crucial for Performance

- The GC is your friend
- The GC doesn’t understand your program’s logic, just memory
- Ideal performance is only achieved by YOU laying out your data to be CPU-friendly

# Predictably

Write logic that makes it easy for the CPU to predict logic and data access


PAGE of PENTACLES
ACE of CUPS

# Entity Component System (ECS)

- Entity
  - The elements of your program
  - Farms, Chickens, Users, etc.
- Component
  - The data associated with Entities
  - Location, Velocity, Status, etc.
- System
  - The logic that acts on Components
  - Movement, State Change, AI, etc.

# Entities

```
module Ids =

    [<Measure>]
    type FarmId

    [<Measure>]
    type ChickenId

    [<Measure>]
    type TurkeyId

    [<Measure>]
    type GooseId

    [<Measure>]
    type AnimalId
```

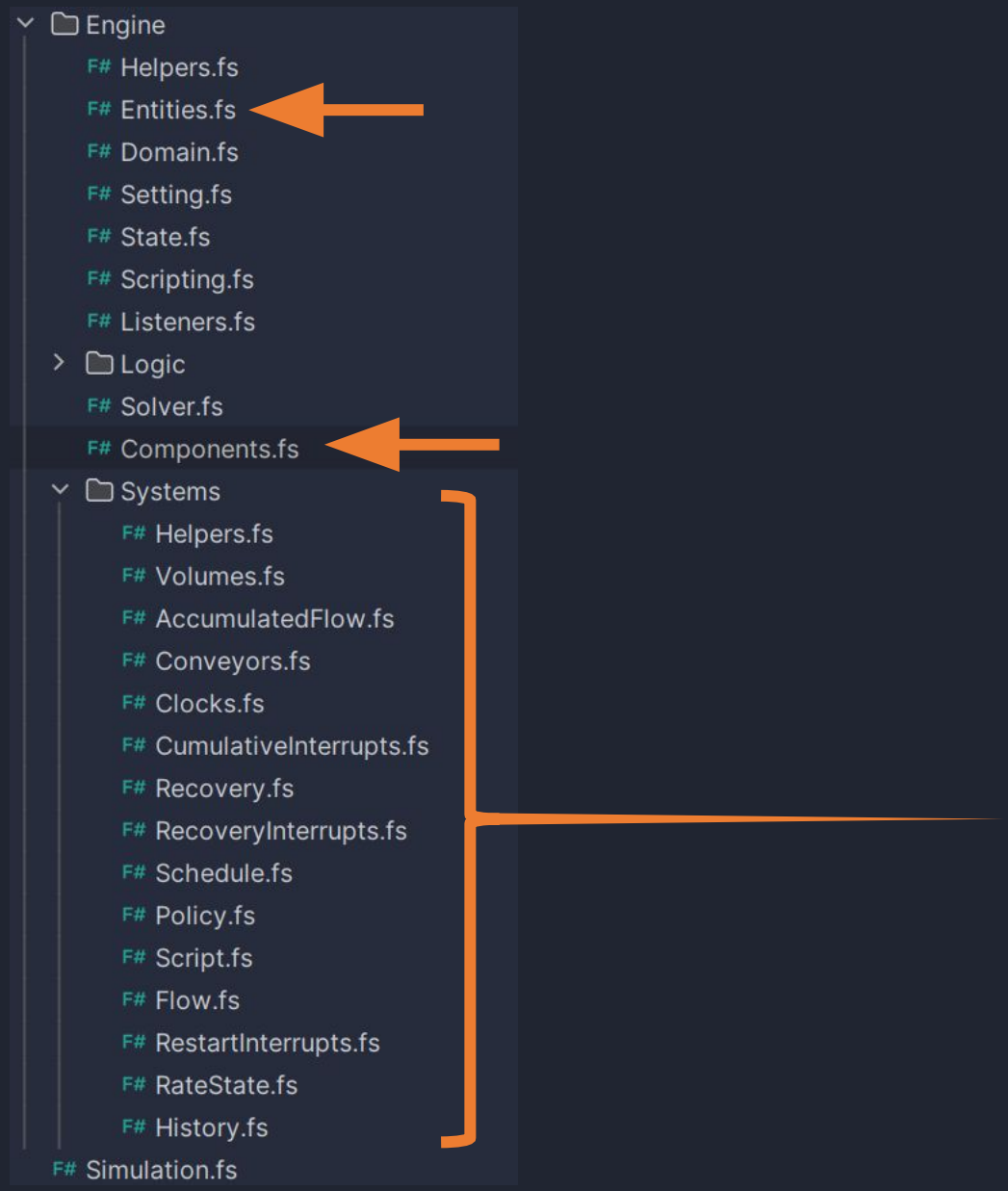
# Components

```fsharp
module Components =

    type Flock =
        {
            Names: Bar<ChickenId, string>
            Ages: Row<ChickenId, int>
            Sizes: Row<ChickenId, float>
        }

    [<NoComparison; NoEquality>]
    type Rafter =
        {
            Names: Bar<TurkeyId, string>
            Ages: Row<TurkeyId, int>
            Sizes: Row<TurkeyId, float>
            IsActive: BitSet<TurkeyId>
        }

    type Cooperative = {
        Names: Bar<FarmId, string>
        Chickens: BarGroup<FarmId, int<ChickenId>>
        Turkeys: BarGroup<TurkeyId, int<TurkeyId>>
    }
```

# Systems

```
namespace Systems

module Chickens =

    let age // Row<ChickenId,int> -> Row<ChickenId,float> -> TimeSpan -> unit
        (ages: Row<ChickenId, int>)
        (sizes: Row<ChickenId, float>)
        (timeStep: TimeSpan)
        =

        // Logic for aging Chickens
```

# Systems


Engine
Helpers.fs
Entities.fs
Domain.fs
Setting.fs
State.fs
Scripting.fs
Listeners.fs
Logic
Solver.fs
Components.fs
Systems
Helpers.fs
Volumes.fs
AccumulatedFlow.fs
Conveyors.fs
Clocks.fs
CumulativeInterrupts.fs
Recovery.fs
RecoveryInterrupts.fs
Schedule.fs
Policy.fs
Script.fs
Flow.fs
RestartInterrupts.fs
RateState.fs
History.fs
Simulation.fs

# Why ECS for Predictability

- ECS Encourages you to group work by the data it acts on
- Systems act in bulk, which encourages small loops that are easy for CPUs to predict
  - Many Small Loops are faster than Fewer Large Loops

# Real World Innovation

*Proceedings of the 2008 Winter Simulation Conference*
*S. J. Mason, R. R. Hill, L. Mönch, O. Rose, T. Jefferson, J. W. Fowler eds.*

**DISCRETE RATE SIMULATION USING LINEAR PROGRAMMING**

Cecile Damiron
Anthony Nastasi

6830 Via Del Oro, Suite 230
Imagine That Inc.
San Jose, CA 95119 USA

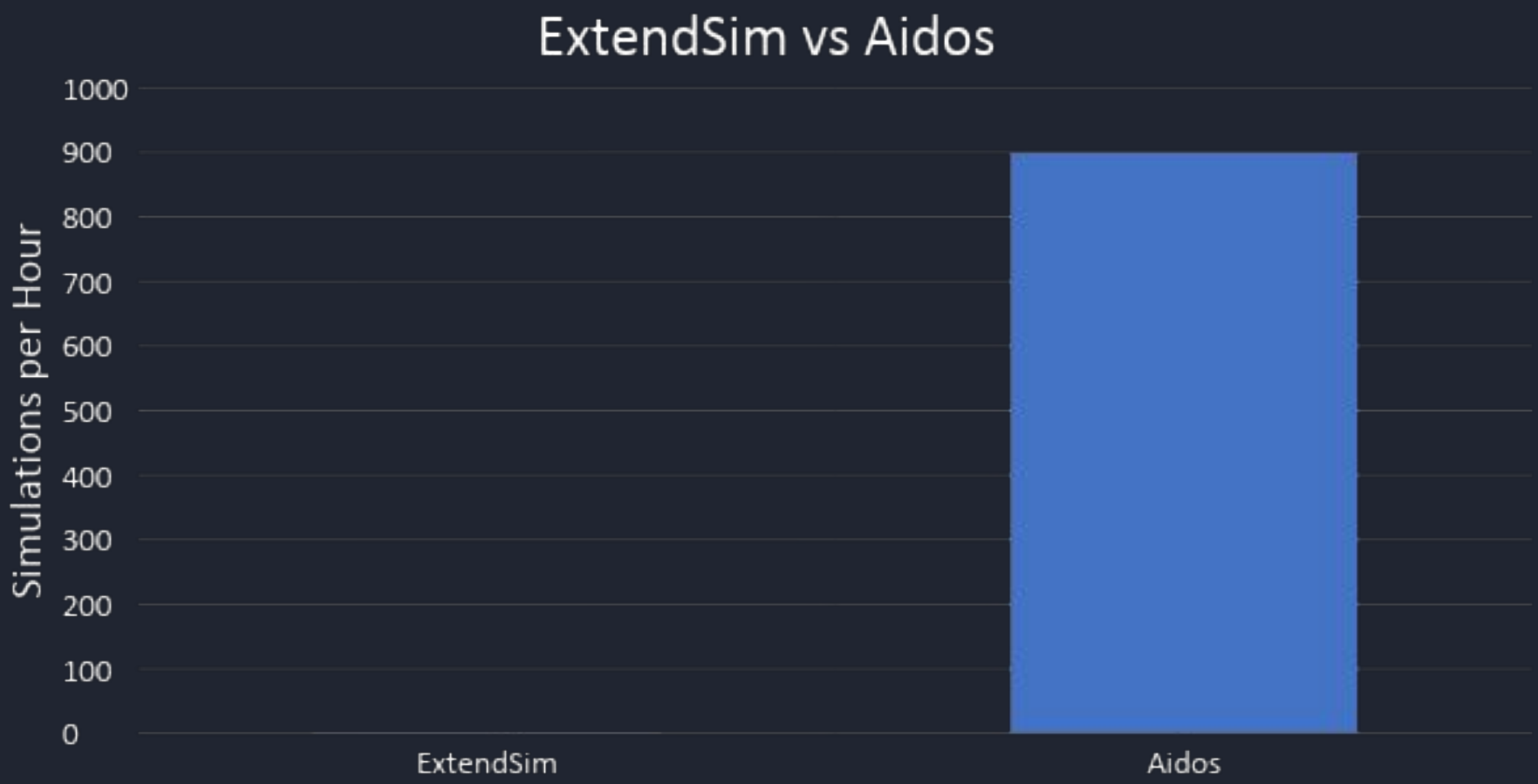
**ABSTRACT**

Discrete Rate Simulation (DRS) is a modeling methodology that uses event based logic to simulate linear continuous processes and hybrid systems. These systems are controlling its movement. Typically, these control mechanisms include valves for modulating flow rates and branching functions for merging and diverging the flow streams.

Hybrid systems, while similar to flow systems in that flow moves continuously at certain rates between loca-

# Real World Innovation

| Engine | Time for Simulation [sec] | Simulations per Hour |
|---|---|---|
| ExtendSim | 2,685 | 1.34 |
| Aidos | 4 | 900 |


ExtendSim vs Aidos
Simulations per Hour
1000
900
800
700
600
500
400
300
200
100
0
ExtendSim
Aidos

# Takeaways…

- F# lets you write correct code quickly
- F# can achieve C-levels of performance
- Units of Measure are essential
- The .NET Runtime is fast and getting faster

# Thank you!

SDI

F# Community

Hack & Craft

Rheinland-Pfälzische Technische Universität Kaiserslautern-Landau

FSLab

F# Software Foundation

Contact: hi@fastfsharp.com