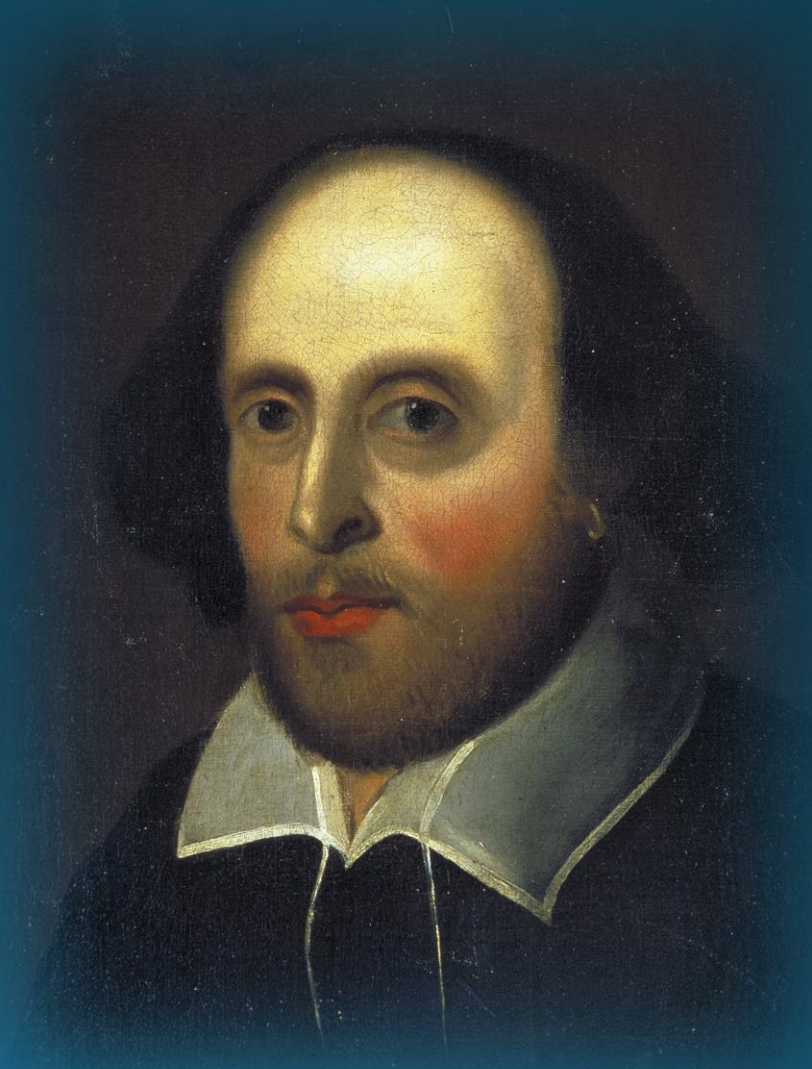
Actors in the .NET Environment

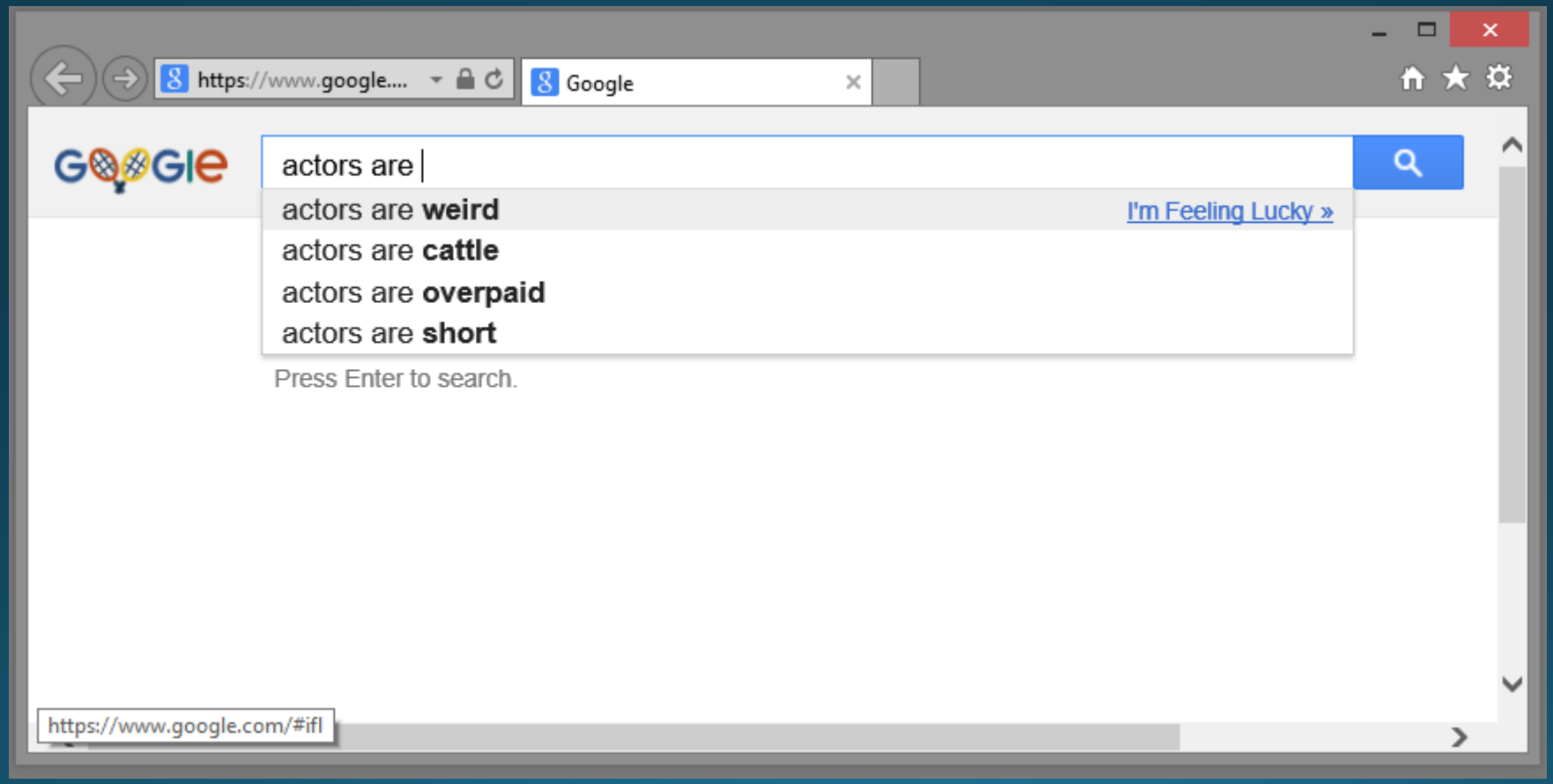
Their Exits and Their Entrances

All the world's a stage,
And all the men and women merely players;
They have their exits and their entrances,
And one man in his time plays many parts…

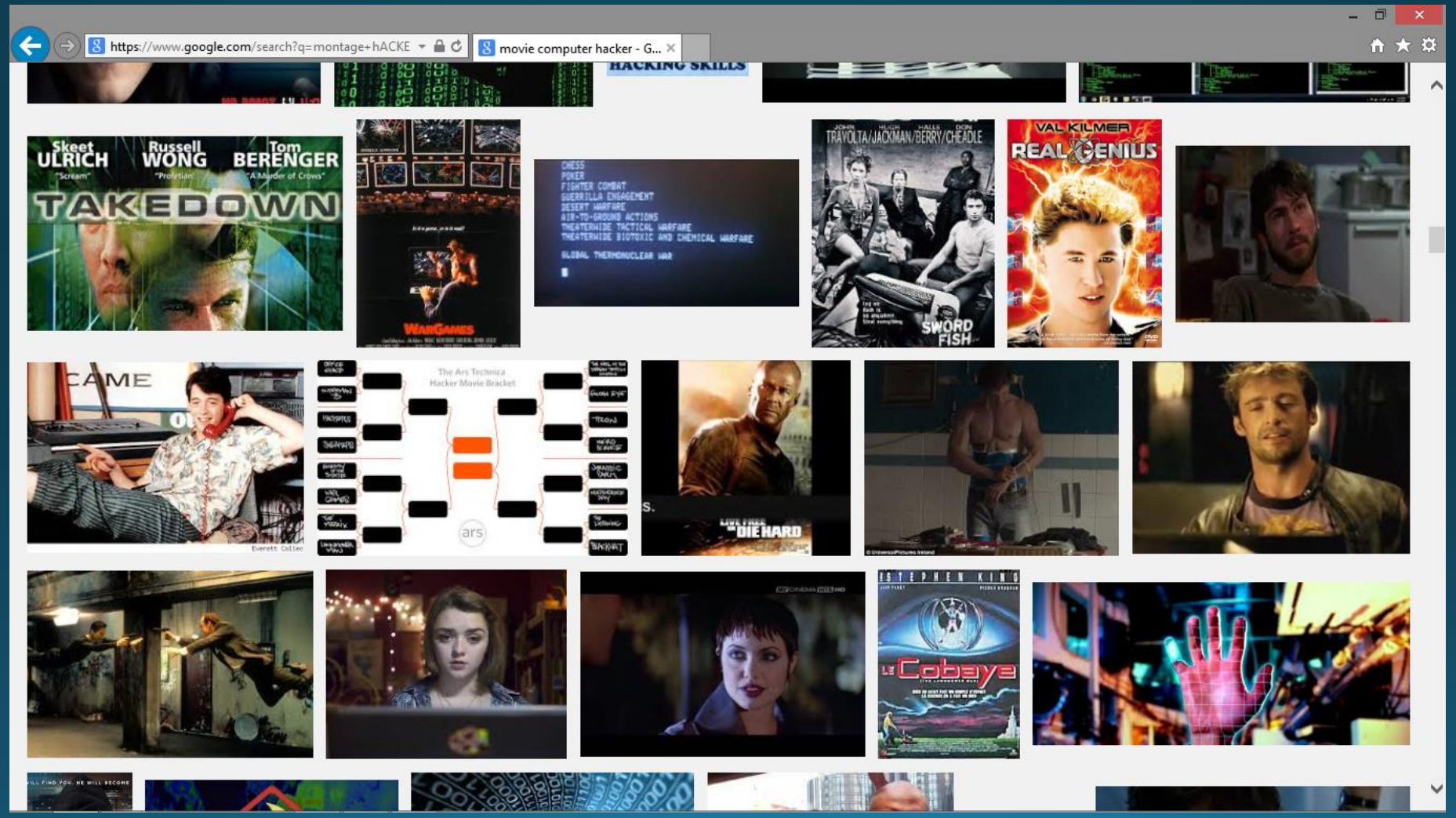
William Shakespeare, *As You Like It*, Act II, Scene VII



Actors are...


https://www.google....
Google
actors are
actors are weird
actors are cattle
actors are overpaid
actors are short
I'm Feeling Lucky »
Press Enter to search.
https://www.google.com/#ifl

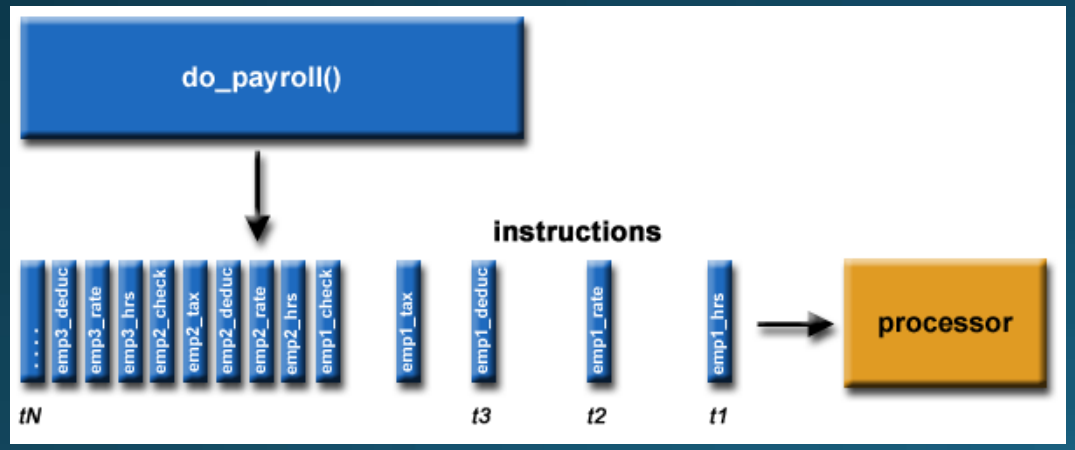
Actors and… Software?



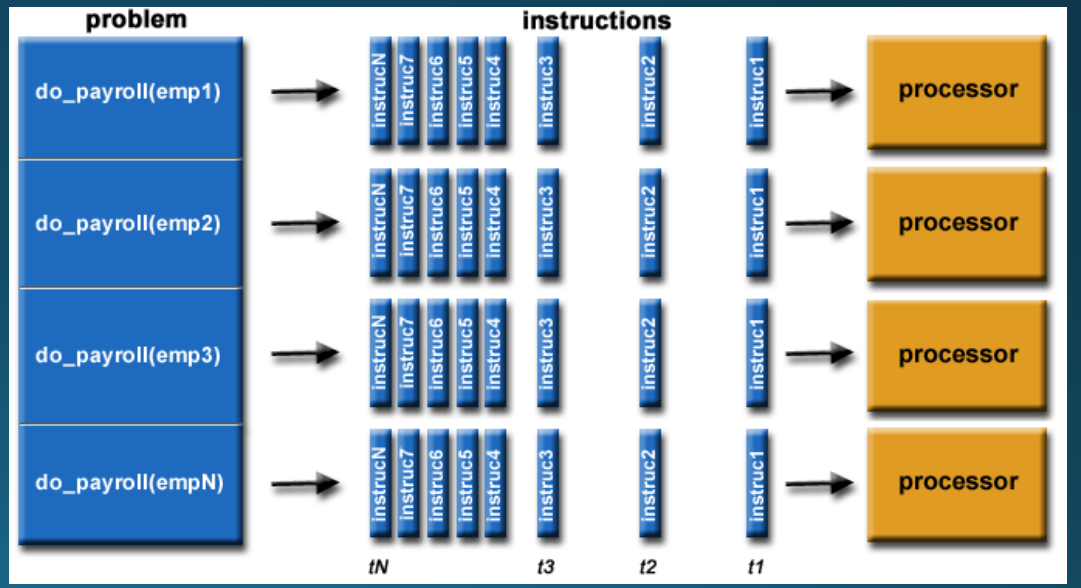
```
goto step_spag;
step_spag:
```

Parallel Programming

From this…

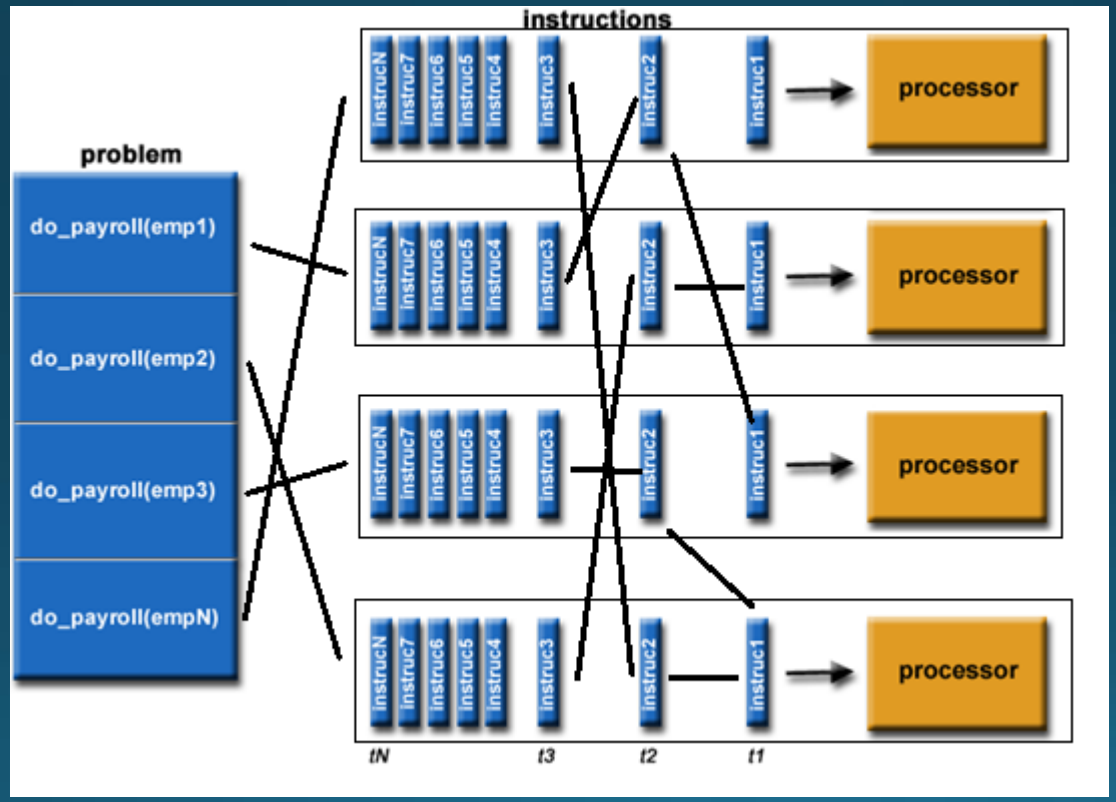


…to this:



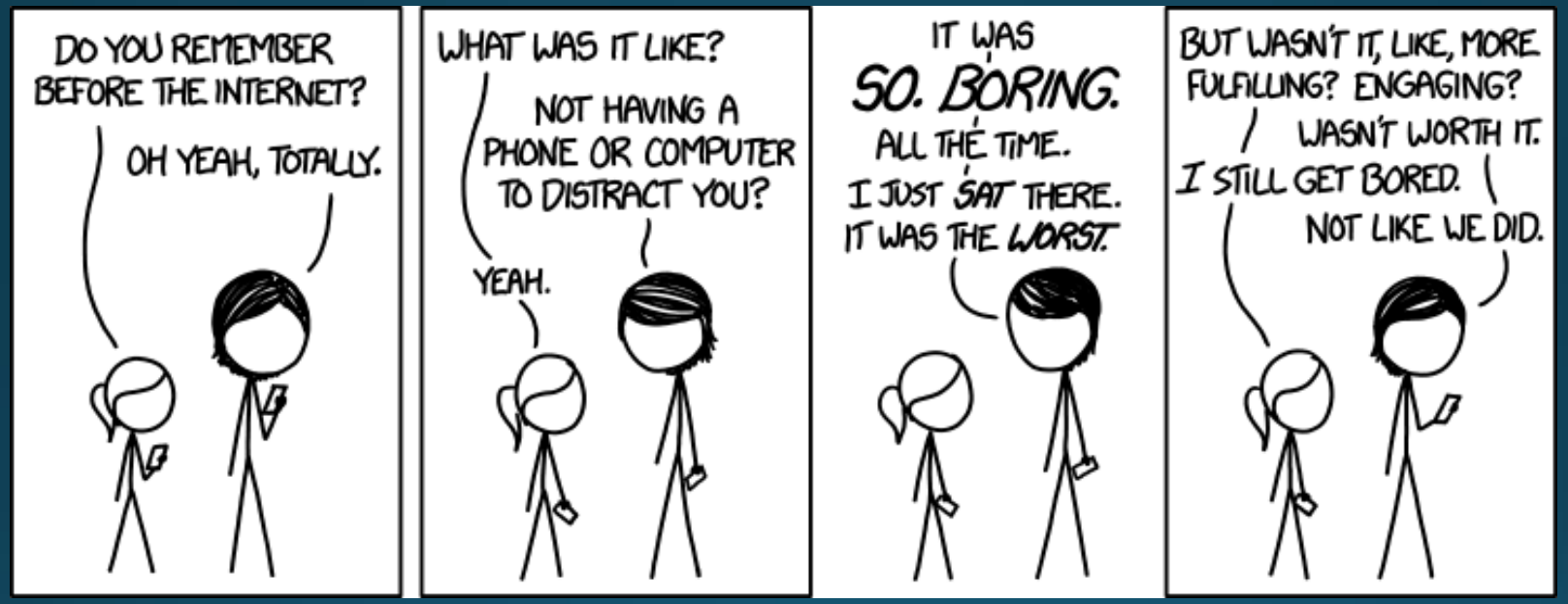
Actor Programming

…to this:



It didn’t work

The Internet



Microsoft .NET and Actors

…it's complicated.

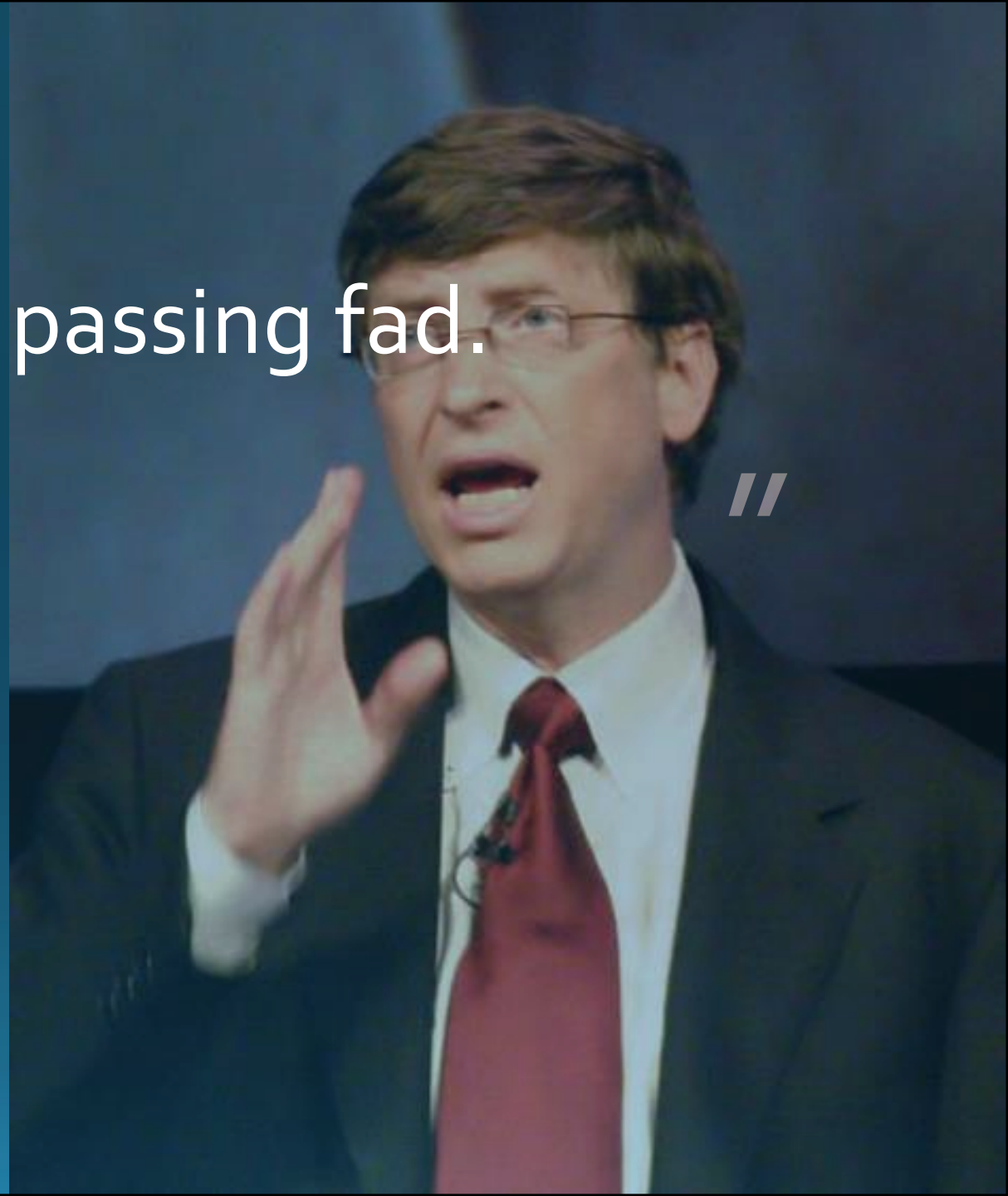
“

The Internet is a passing fad.

”

Bill Gates, 1993

- No, he didn’t say this. He’s not stupid.
- It certainly felt that way at the time.



Microsoft Azure Reliable Actors

- Part of Azure Service Fabric
- Always-on Services based upon ‘microservices’
  - Many independent units of state and logic
  - Single-threaded objects that will still scale
  - You don’t want to manage their granularity or concurrency
  - You don’t want to manage their messages

Reliable Actor Interface and Proxy

```
public interface ICalculatorActor : IActor
{
    Task<double> AddAsync(double valueOne, double valueTwo);
    Task<double> SubtractAsync(double valueOne, double valueTwo);
}
```

```
public class CalculatorActor : Actor, ICalculatorActor
{
    public Task<double> AddAsync(double valueOne, double valueTwo)
    {
        return Task.FromResult(valueOne + valueTwo);
    }

    public Task<double> SubtractAsync(double valueOne, double valueTwo)
    {
        return Task.FromResult(valueOne - valueTwo);
    }
}
```

```
ActorId actorId = ActorId.NewId();
string applicationName = "fabric:/CalculatorActorApp";
ICalculatorActor calculatorActor = ActorProxy.Create<ICalculatorActor>(actorId, applicationName);
double result = calculatorActor.AddAsync(2, 3).Result;
```

Akka and the Reactive Manifesto

Reactive Systems are:

- ***Responsive**: The system responds in a timely manner if at all possible.*
- ***Resilient**: The system stays responsive in the face of failure.*
- ***Elastic**: The system stays responsive under varying workload.*
- ***Message Driven**: The system uses asynchronous messages to establish a boundary between components.*

Akka.NET – ACT ALL THE THINGS

UntypedActor, message, and IActorRef

```
public class CalculatorActor: UntypedActor
{
    protected override void OnReceive(object message)
    {
        CalcMessage msg = (CalcMessage) message;
        if (msg.Action == "add")
        {
            Console.WriteLine(msg.X + msg.Y);
        }
    }
}
```

```
public class CalcMessage
{
    public CalcMessage(double x, double y, string act)
    {
        X = x;
        Y = y;
        Action = act;
    }
    public double X { get; private set; }
    public double Y { get; private set; }
    public string Action;
}
```

```
ActorSystem MyActorSystem = ActorSystem.Create("MySystem");

var calculator = MyActorSystem.ActorOf<CalculatorActor>();

var addMessage = new CalcMessage(1, 1, "add");

calculator.Tell(addMessage);
```

Choices

Reliable Actors

- Service Fabric
- Azure only
- Orleans & Halo
- Still pre-release

- Child-proofed

Akka.NET

- ActorSystem
- Wherever .NET runs
- Open Source
- Existing Akka base

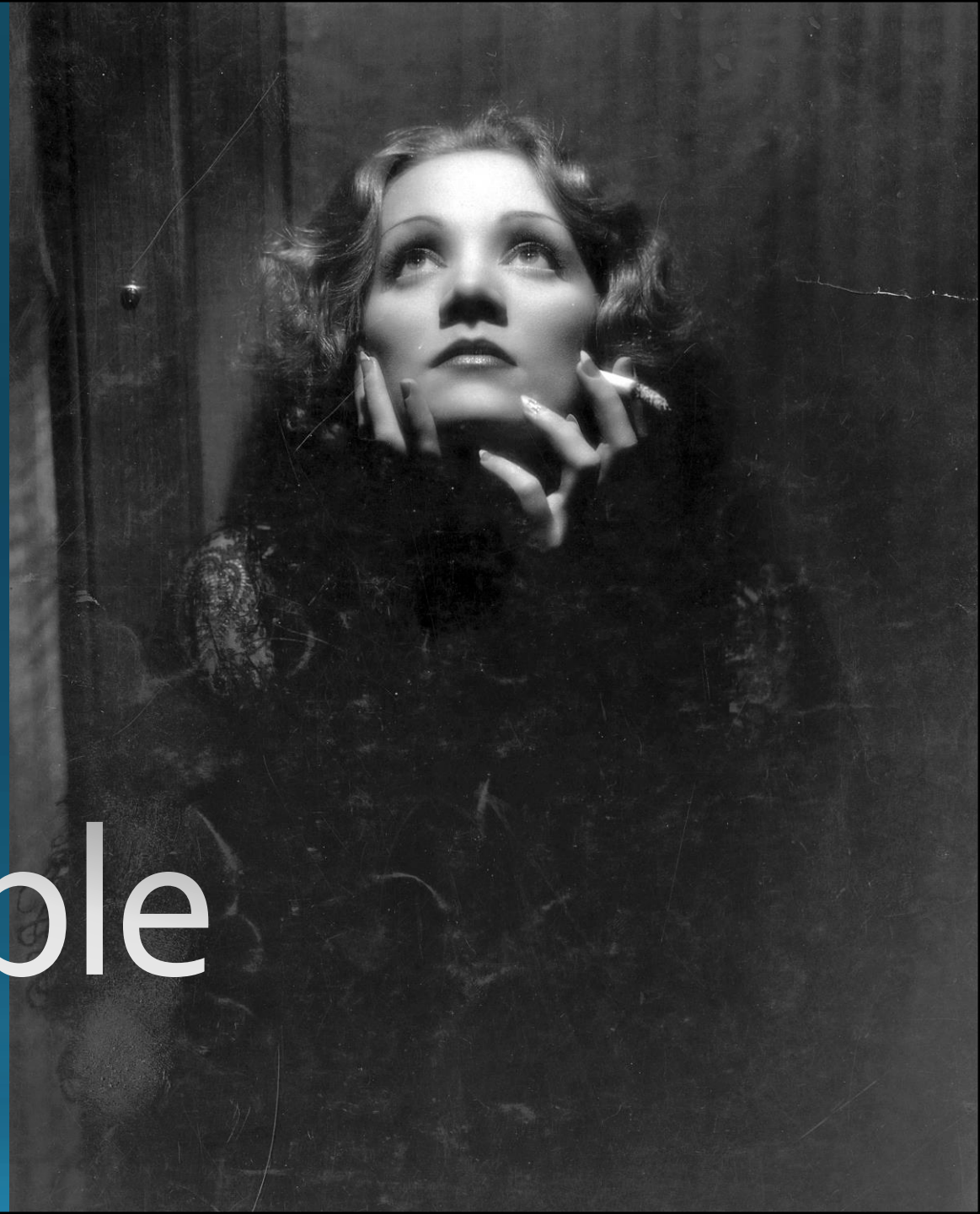
- No safety net

Hot lines under a rain of drum
Cigarette props in action

Dialogue dub, now here's the rub
She's acting her reaction

-- Bauhaus, "She's In Parties" (1983)

Actor Example



Thanks For Your Interest

- All material for this talk will be available at https://carteritrellis.github.io/CodeCamp2015
- If you found this at all interesting, come talk to me – we're hiring.

Carter Wickstrom

Software Engineer

https://www.linkedin.com/in/carterwickstrom

https://twitter.com/carterwickstrom