Rheinland-Pfälzische Technische Universität Kaiserslautern Landau

Writing libraries for the FsLab ecosystem

Kevin Schneider

CSB

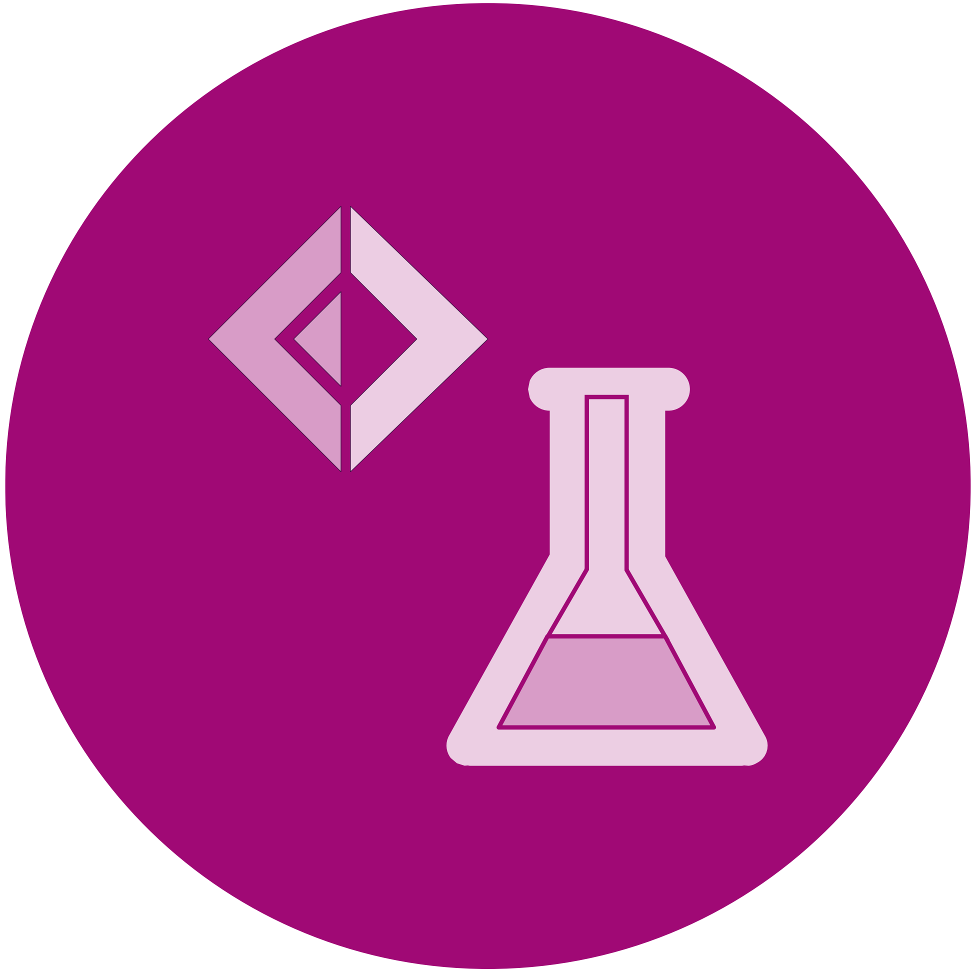
COMPUTATIONAL SYSTEMS BIOLOGY

Writing libraries for the FsLab ecosystem

- What is FsLab, what should it be?
- Plotly.NET
- Learnings from Plotly.NET

What is FsLab?

- Project incubation space for data science projects



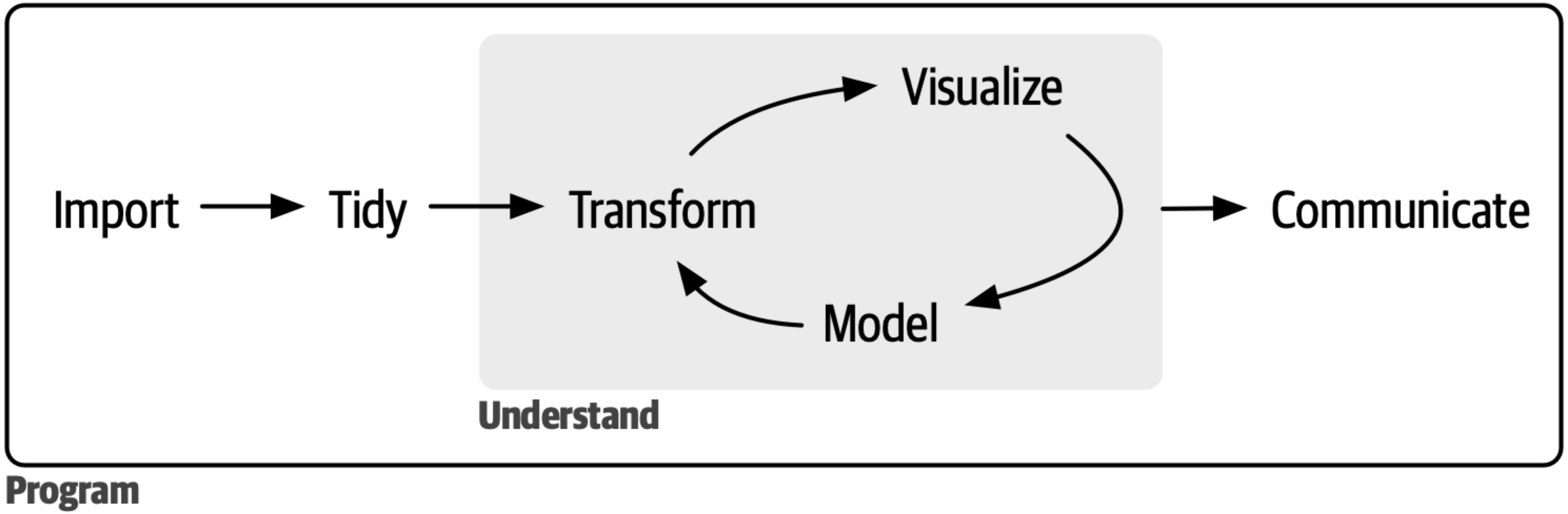
What should FsLab be?

- **cohesive, high quality data science stack for F#**
- Foster a community around it

R and the tidyverse

The tidyverse

- A group of R packages for data science
- Common design and data handling philosophy
- ‘Extended’ tidyverse: less strictly involved projects



Learnings from the tidyverse

- R:
  - Core language is explicitly designed for data analysis
  - Lacks a strong type system
  - Large community

- tidyverse:
  - Dogmatic focus on producing ‘tidy’ data
  - Visualization
  - Extension of functional programming capabilities

Learnings from the tidyverse

- FsLab should have different goals:
  - **Visualization**
  - What is a **data frame** in a strongly typed language?
  - **Data structures and algorithms** for data science
  - How should common APIs look and feel?
  - Adopt core stack + extended universe approach

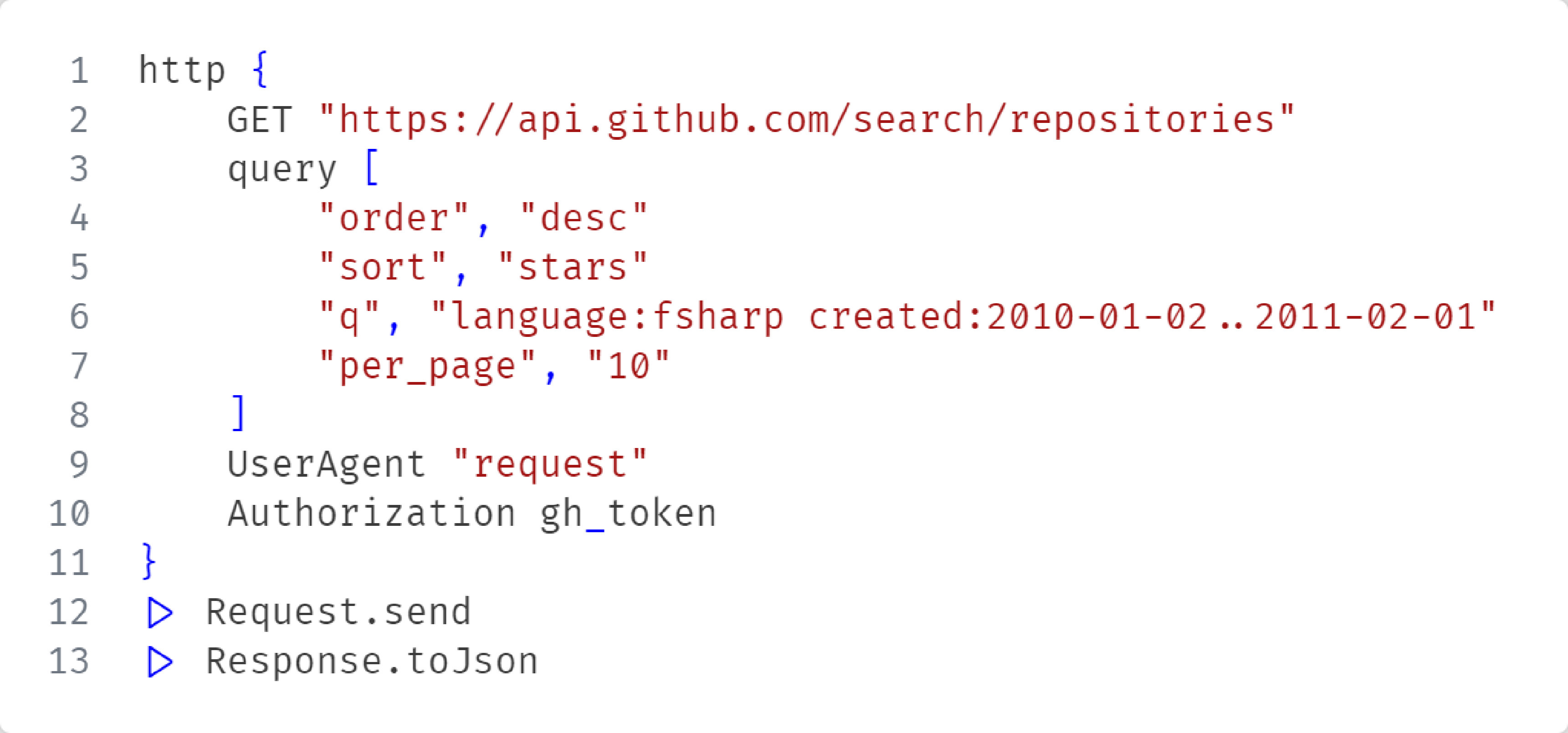
- Current stage: emergence of individual high-quality packages

Plotly.NET

- Fully-featured visualization library
- Built on plotly.js
- Type safe, multi-layered abstractions for hundreds of visualization types
- Inspired by FSharp.Charting and XPlot

GitHub stars – a proxy for success of a F# library?

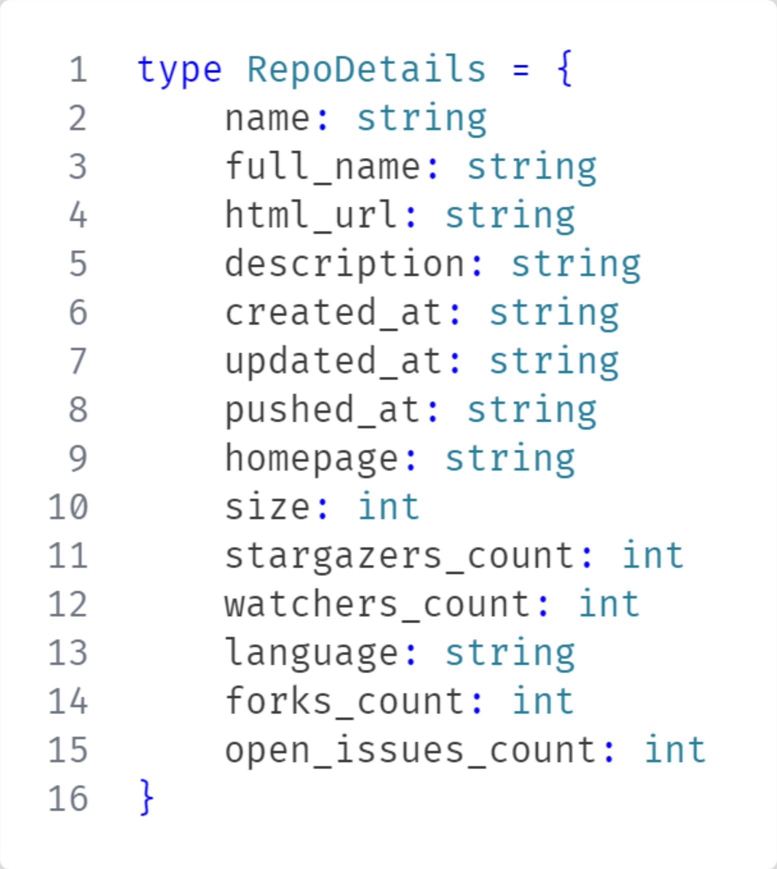
- Collect and analyze data of all public F# repositories



```
http {
    GET "https://api.github.com/search/repositories"
    query [
        "order", "desc"
        "sort", "stars"
        "q", "language:fsharp created:2010-01-02..2011-02-01"
        "per_page", "10"
    ]
    UserAgent "request"
    Authorization gh_token
}
|> Request.send
|> Response.toJson
```

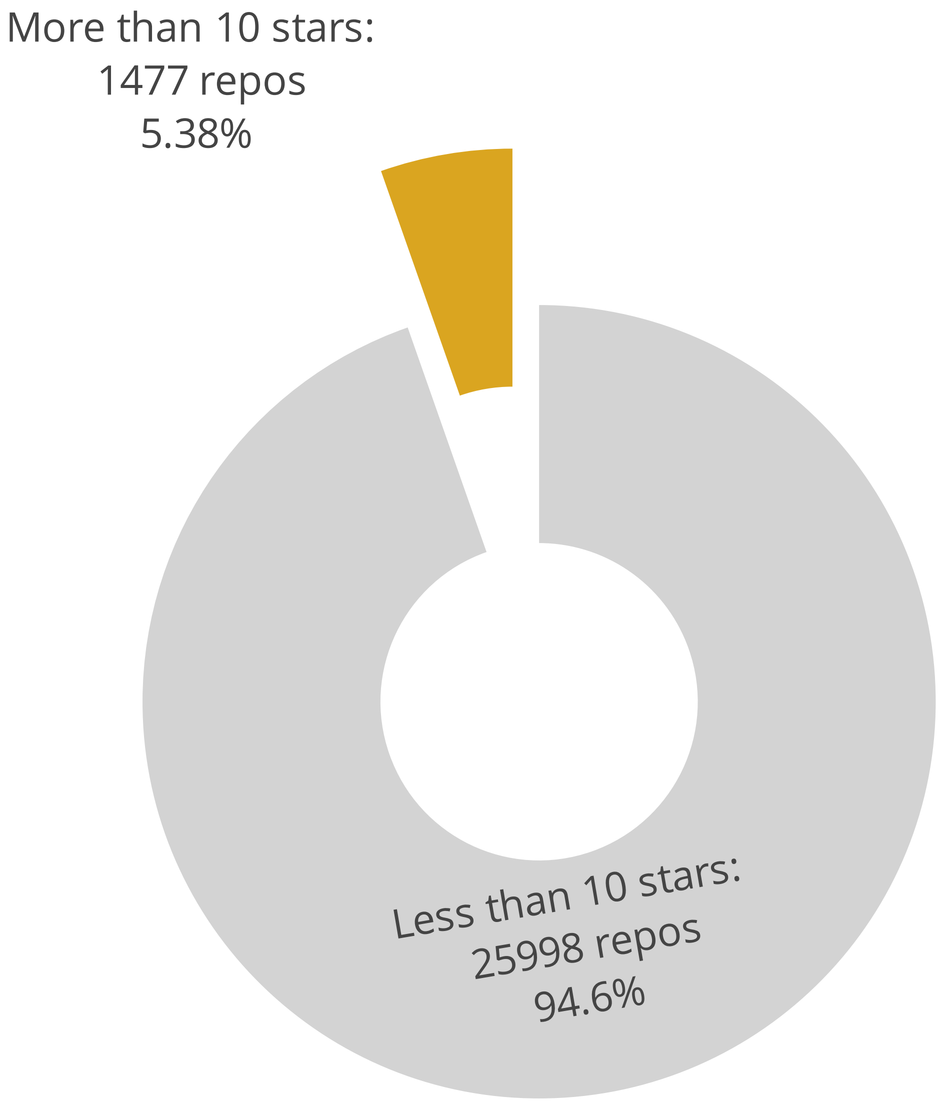
GitHub stars – a proxy for success of a F# library?

- Collect and analyze data of all public F# repositories

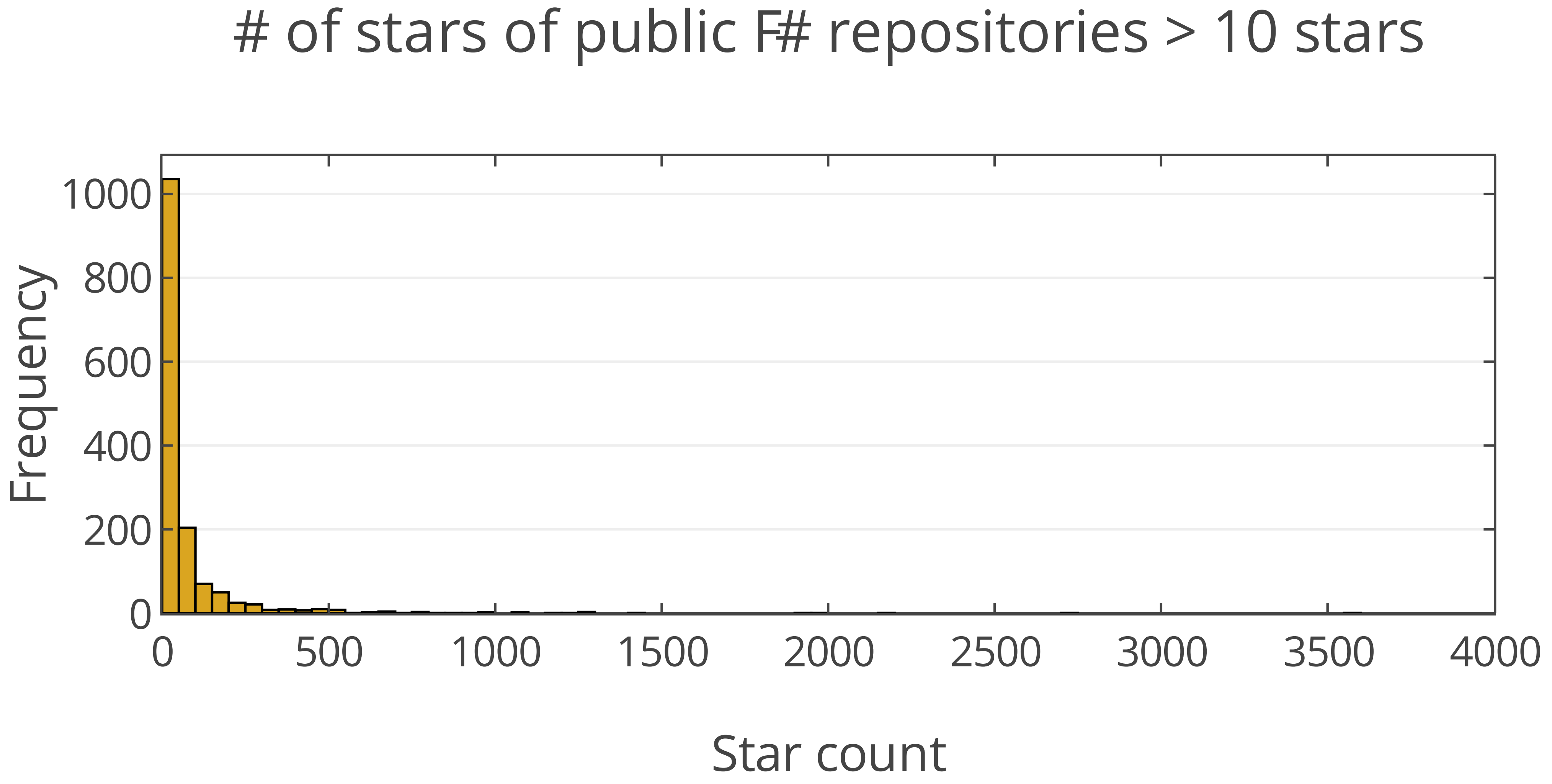


```
type RepoDetails = {
    name: string
    full_name: string
    html_url: string
    description: string
    created_at: string
    updated_at: string
    pushed_at: string
    homepage: string
    size: int
    stargazers_count: int
    watchers_count: int
    language: string
    forks_count: int
    open_issues_count: int
}
```

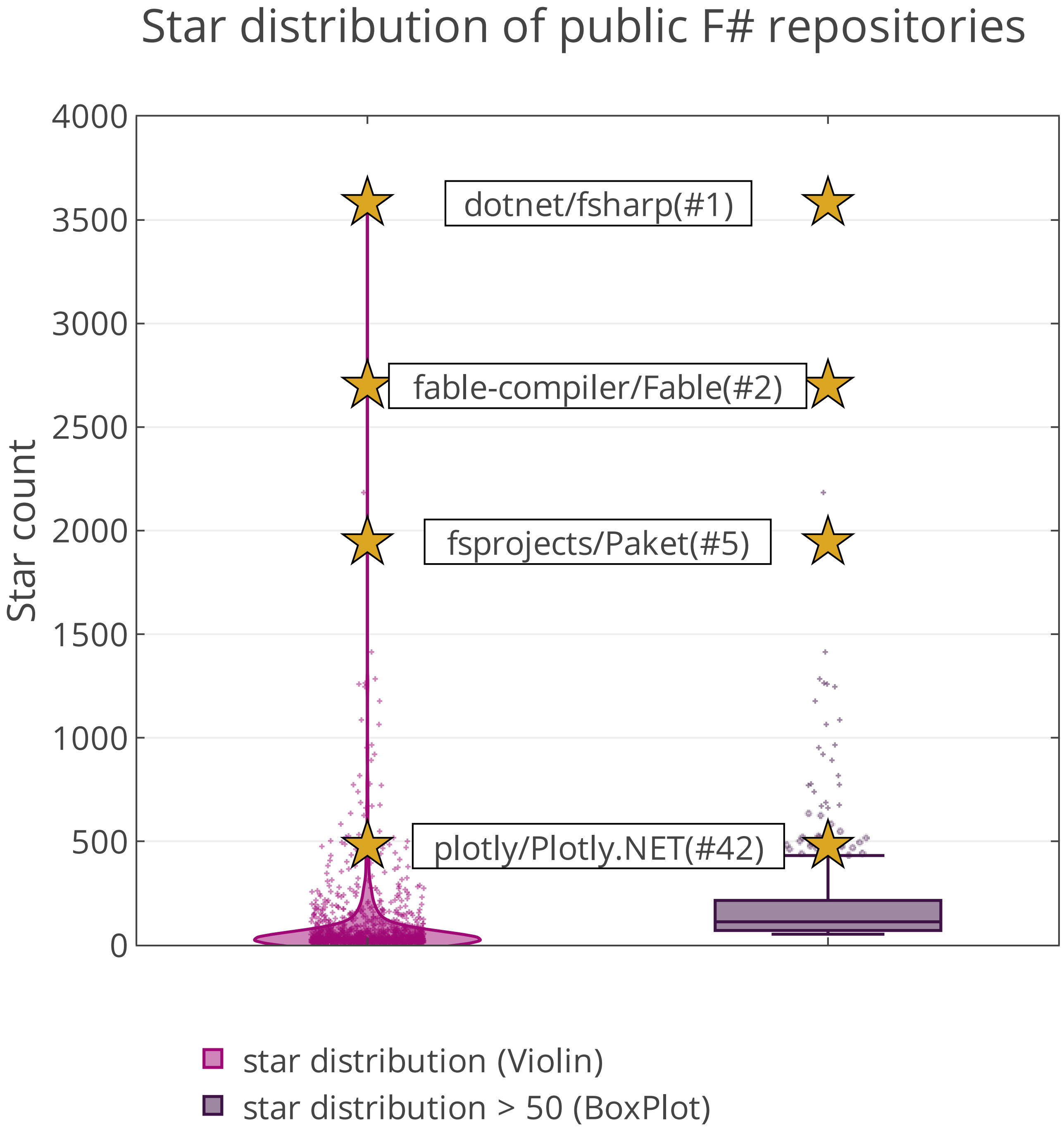
- Total repositories: 27475
- Sample: take repos > 10 stars



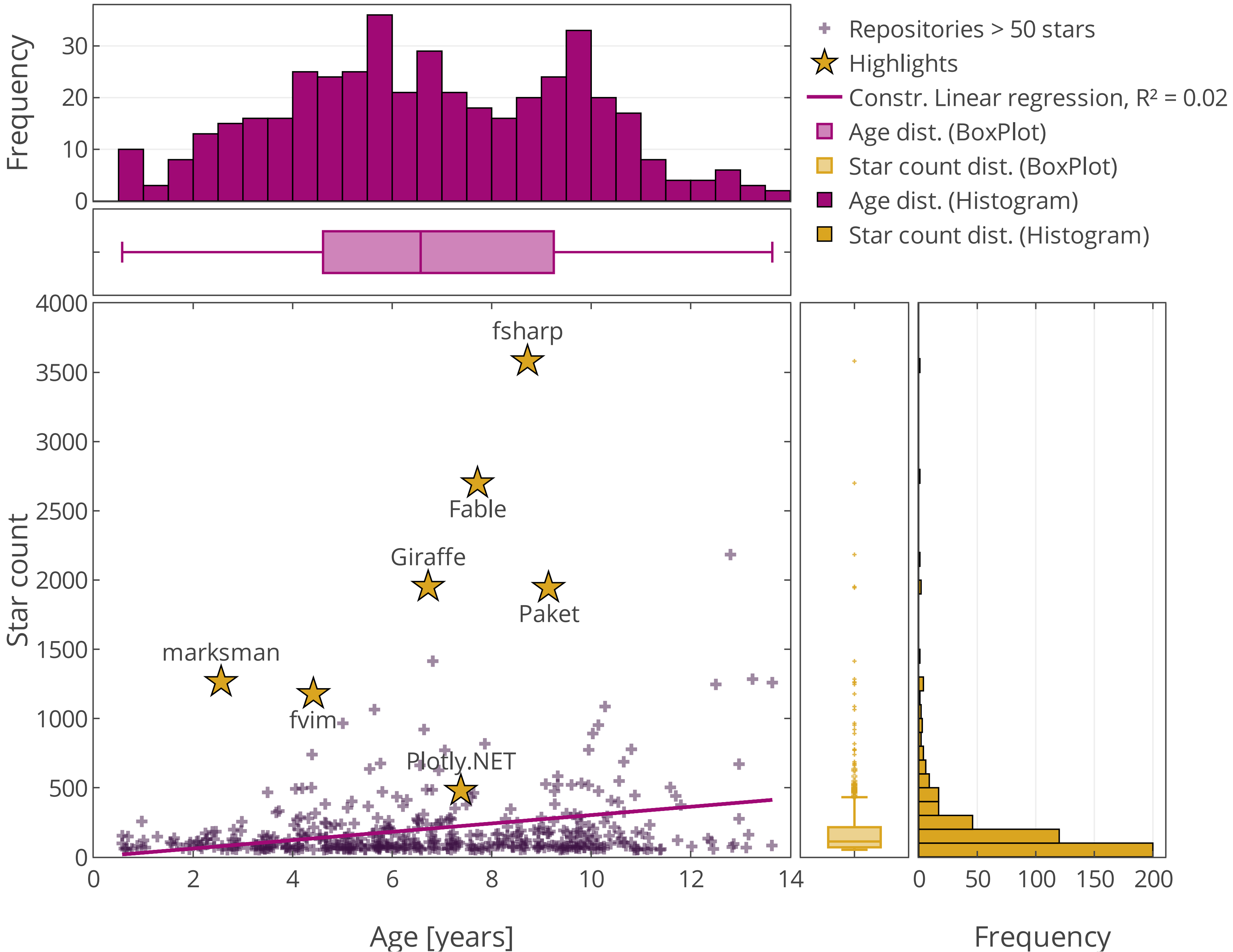
Majority of repositories have low star count



- Plotly.NET is in the suspected outlier range (>1.5IR)



Public F# repositories on GitHub - star count vs. age



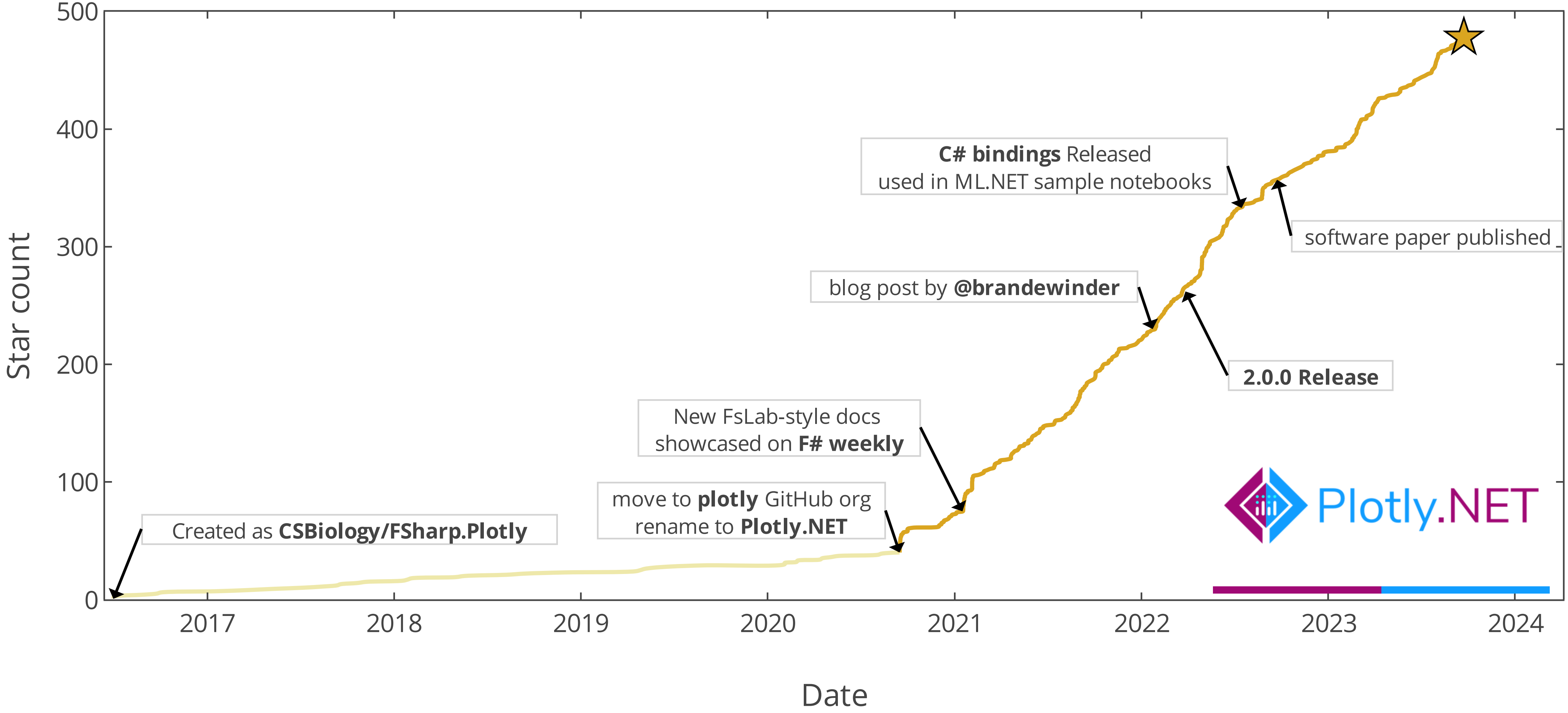
GitHub stars – a proxy for success of F# library?

- GitHub stars can be used as a proxy for success.
- Plotly.NET is a relatively successful library in the F# OSS space.

Reasons for relative success

- External:
  - Sponsoring
  - Promotion
  - Community recognition

Star count of Plotly.NET over time



Created as **CSBiology/FSharp.Plotly**

move to **plotly** GitHub org
rename to **Plotly.NET**

New FsLab-style docs
showcased on **F# weekly**

blog post by **@brandewinder**

2.0.0 Release

C# bindings Released
used in ML.NET sample notebooks

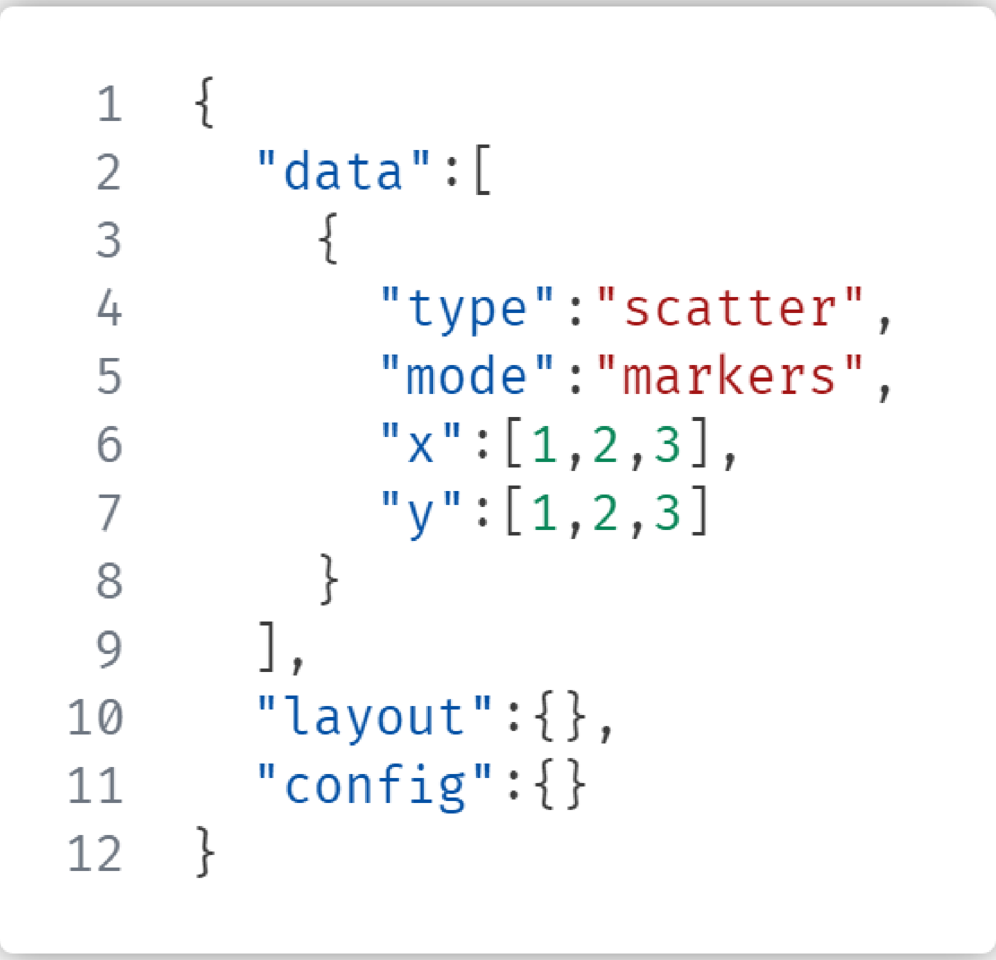
software paper published

Reasons for relative success

- External:
  - Sponsoring
  - Promotion
  - Community recognition

- Internal:
  - API design specifics
  - Documentation
  - Samples

Plotly.js foundations



```
{
  "data":[
    {
      "type":"scatter",
      "mode":"markers",
      "x":[1,2,3],
      "y":[1,2,3]
    }
  ],
  "layout":{},
  "config":{}
}
```

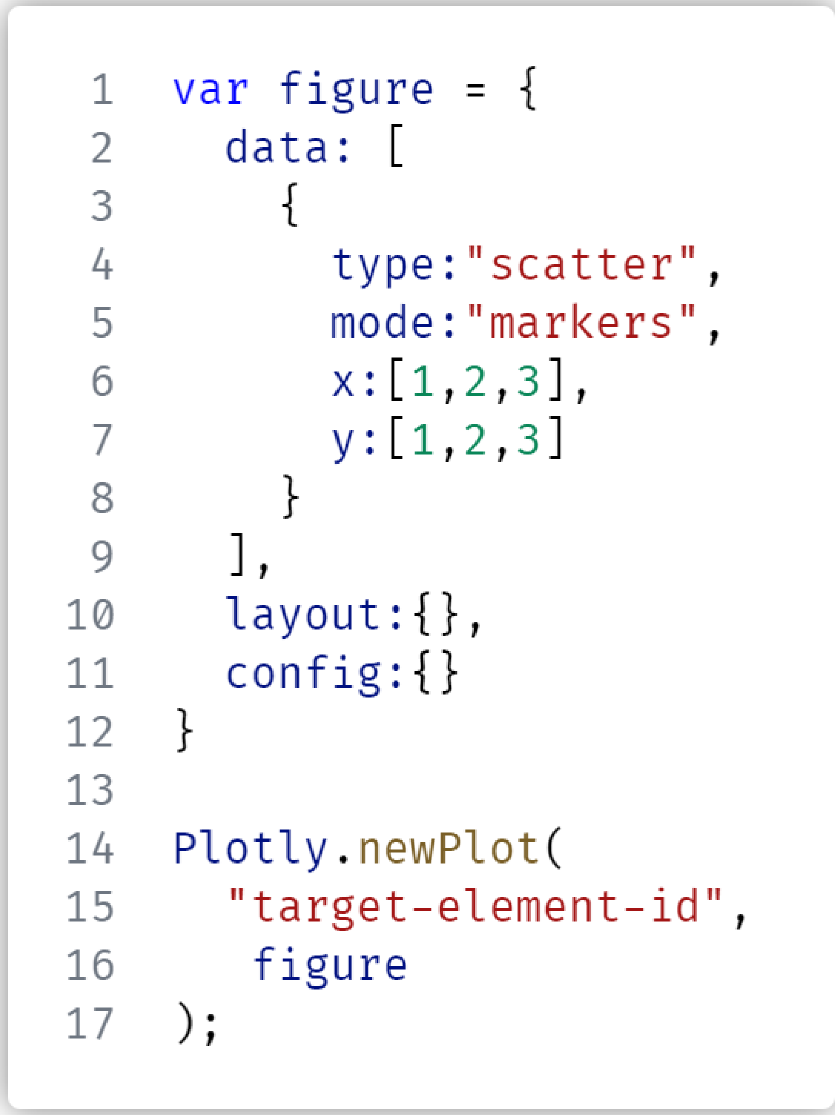
- **Declarative Json schema** for creating data visualizations

- **data**: trace objects

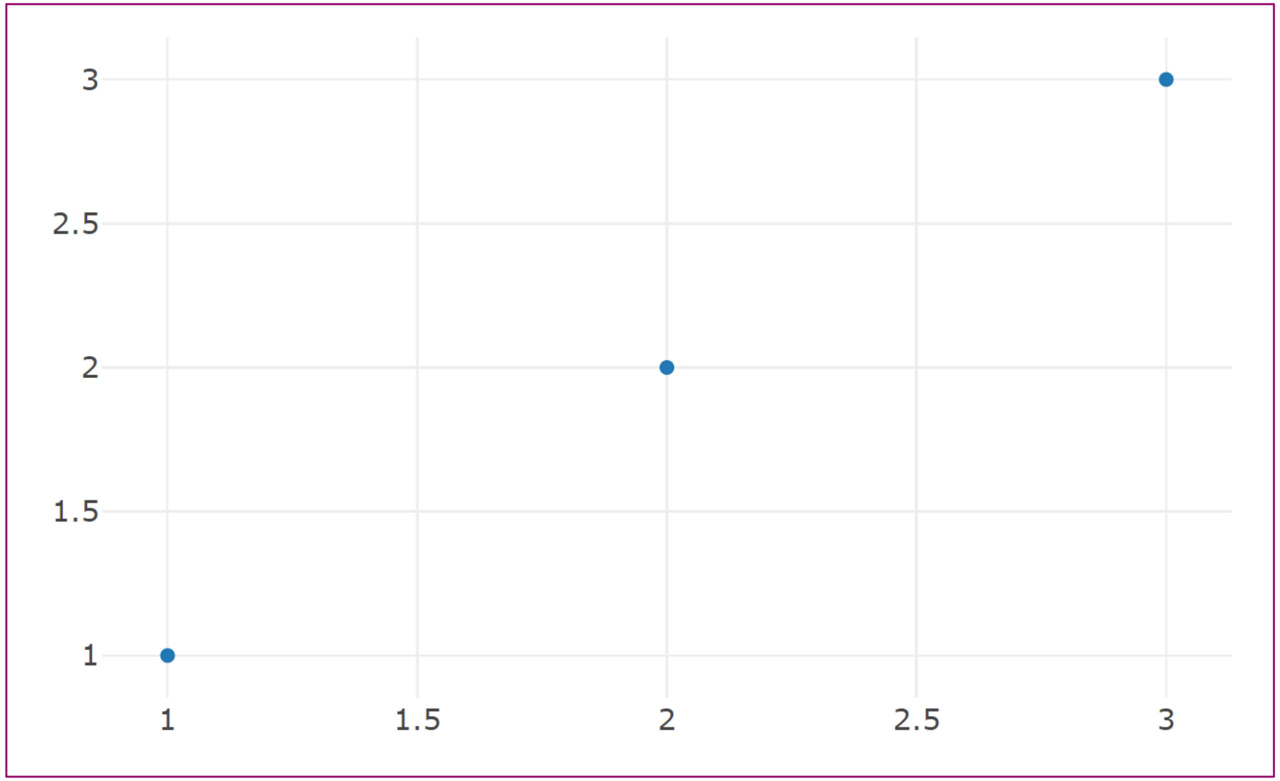
  (data + chart type)

- **layout**: overall styling
- **config**: render settings

Plotly.js foundations



```
var figure = {
  data: [
    {
      type:"scatter",
      mode:"markers",
      x:[1,2,3],
      y:[1,2,3]
    }
  ],
  layout:{},
  config:{}
}

Plotly.newPlot(
  "target-element-id",
   figure
);
```



Plotly.NET: core challenge

- JSON schema allows different (or even arbitrary) types
- F# is a statically language
- Simple type modelling not feasible

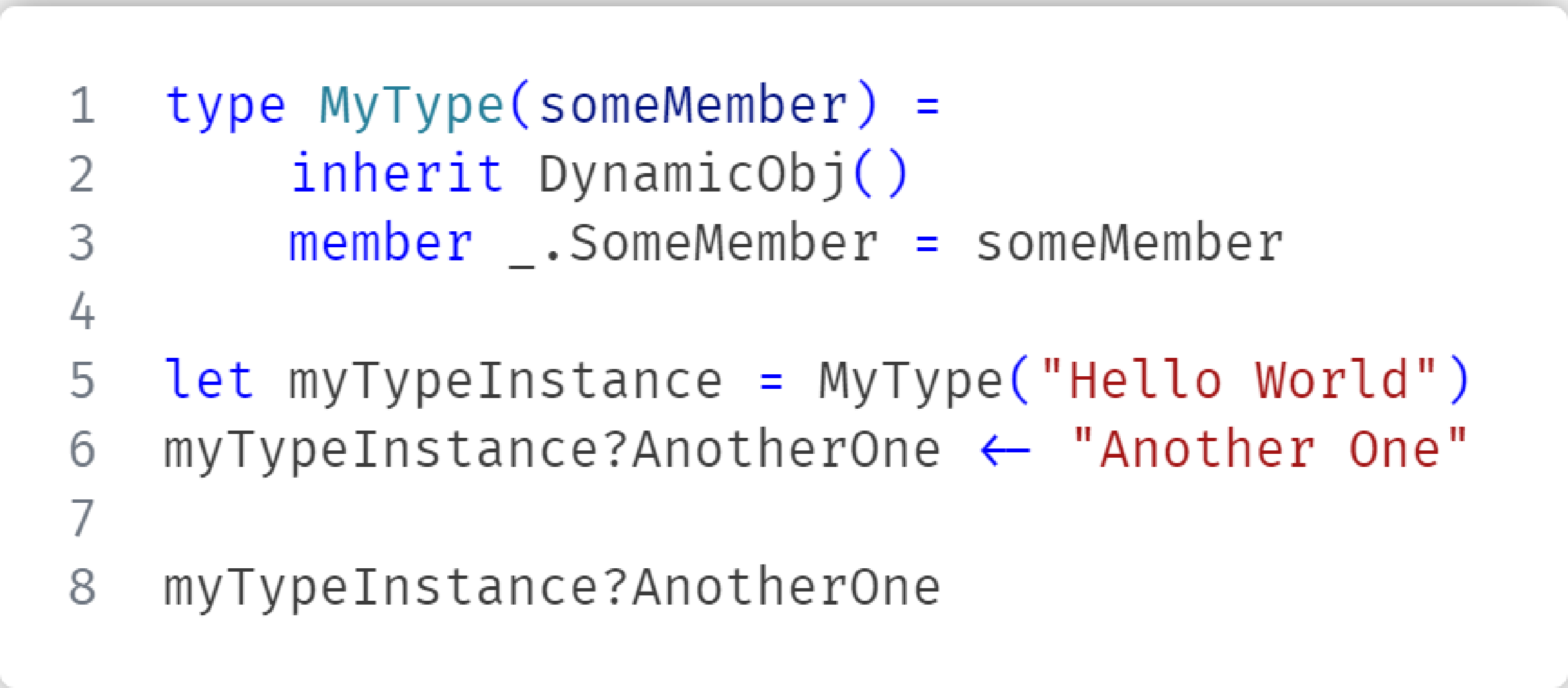
- **Solution**: Increasingly typed abstractions on a dynamic core

Increasingly typed abstractions

- Everything must be possible (low level, expert API)
- Common tasks must be easy (high-level, low friction API)
- Re-use established patterns

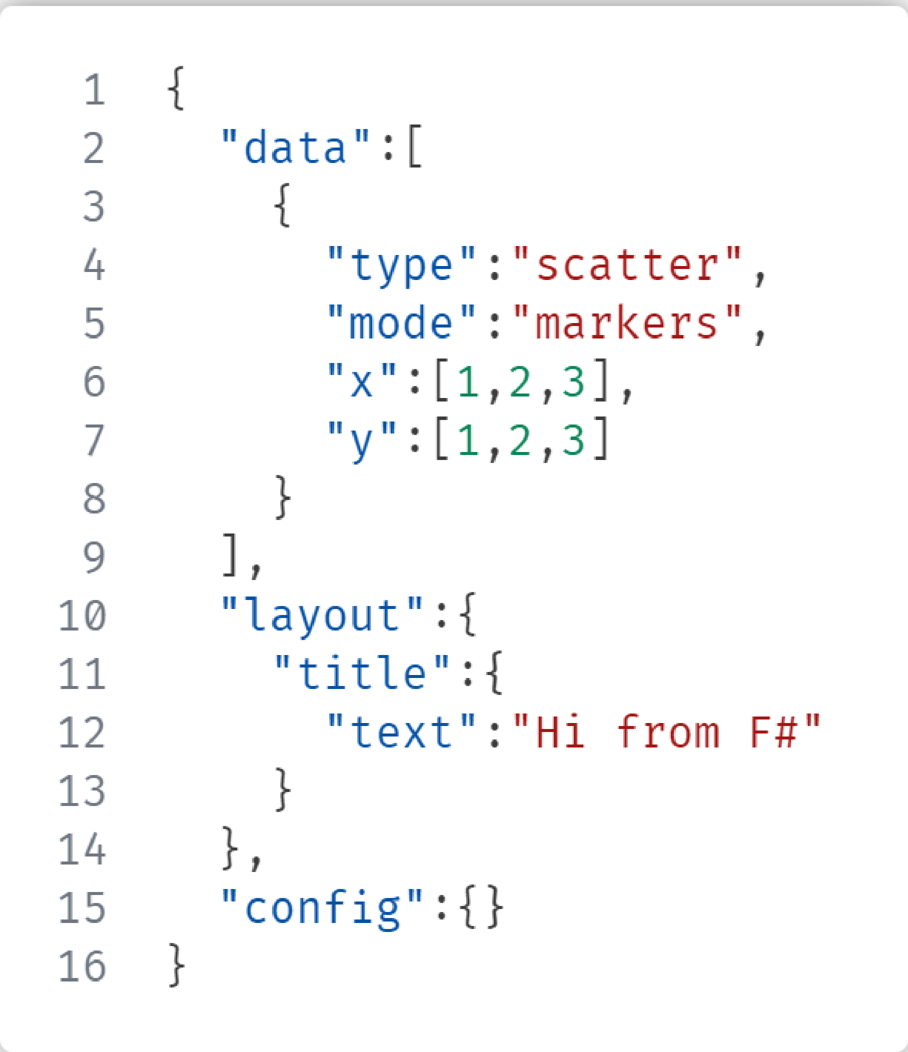
Increasingly typed abstractions: **DynamicObj**

- Low-level API
- Dynamic members on any type by inheritance



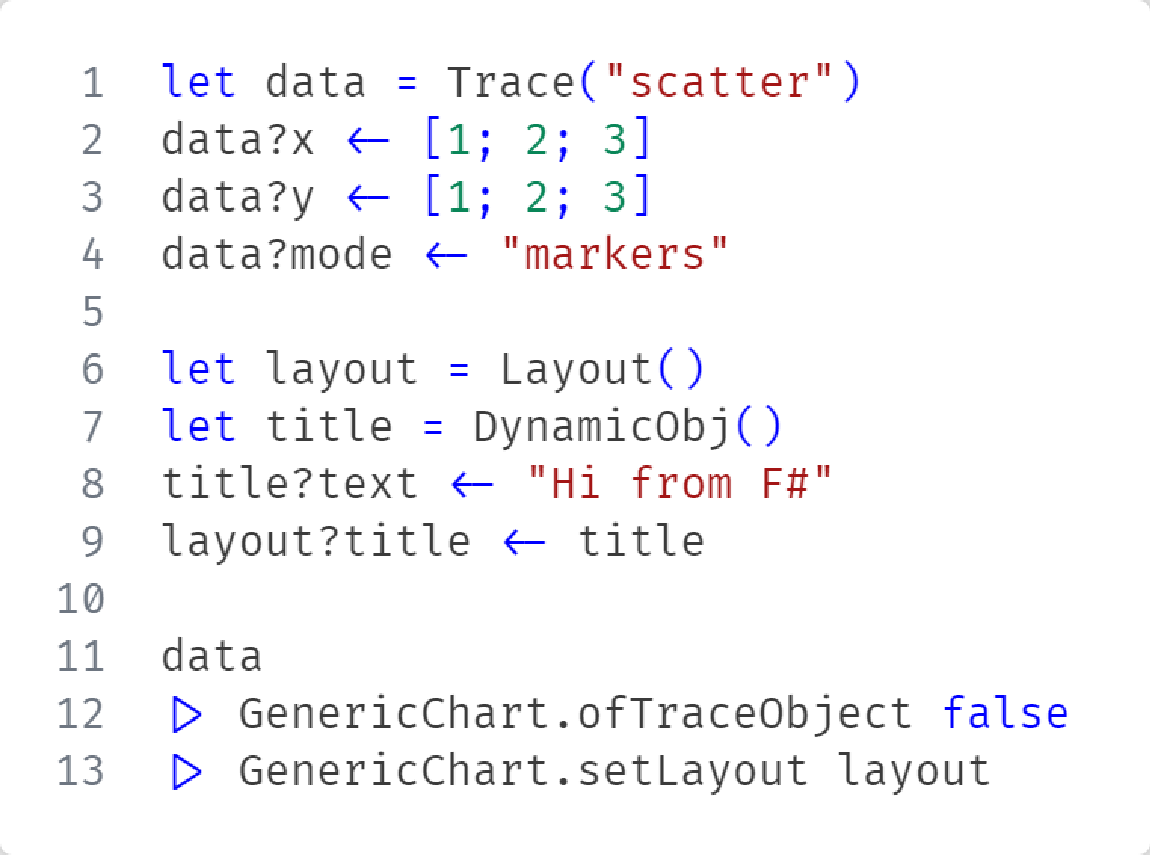
```
type MyType(someMember) =
    inherit DynamicObj()
    member _.SomeMember = someMember

let myTypeInstance = MyType("Hello World")
myTypeInstance?AnotherOne ← "Another One"

myTypeInstance?AnotherOne
```

Increasingly typed abstractions: **DynamicObj**



```
{
  "data":[
    {
      "type":"scatter",
      "mode":"markers",
      "x":[1,2,3],
      "y":[1,2,3]
    }
  ],
  "layout":{
    "title":{
      "text":"Hi from F#"
    }
  },
  "config":{}
}
```

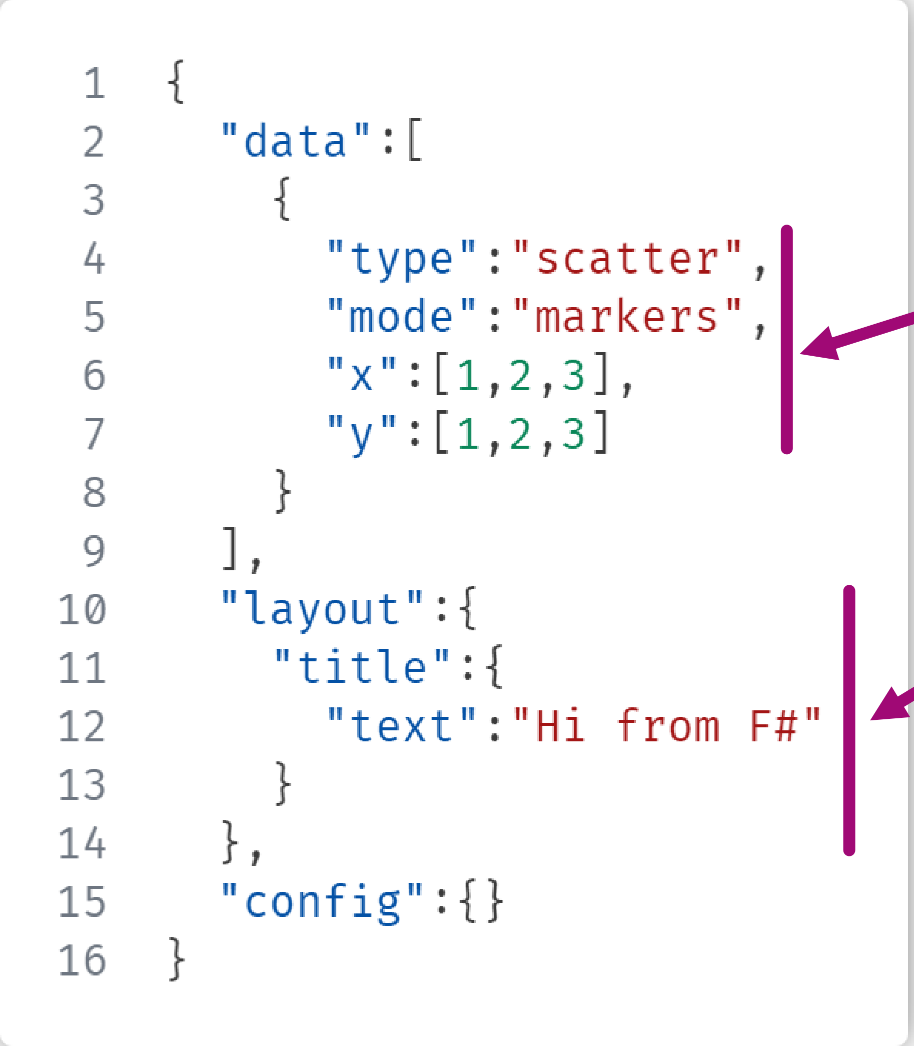
plotly.js json



```
let data = Trace("scatter")
data?x ← [1; 2; 3]
data?y ← [1; 2; 3]
data?mode ← "markers"

let layout = Layout()
let title = DynamicObj()
title?text ← "Hi from F#"
layout?title ← title

data
▷ GenericChart.ofTraceObject false
▷ GenericChart.setLayout layout
```

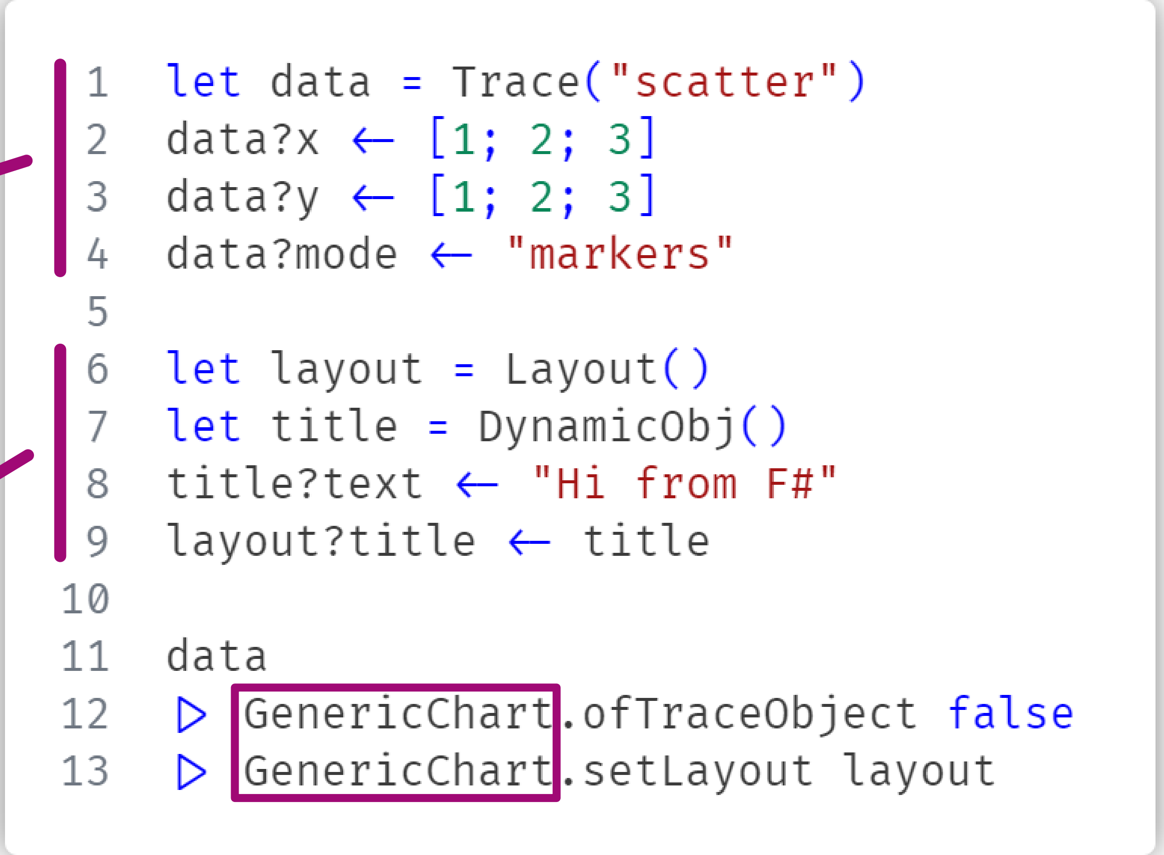
Plotly.NET: DynamicObj layer

Increasingly typed abstractions: **DynamicObj**



```
{
  "data":[
    {
      "type":"scatter",
      "mode":"markers",
      "x":[1,2,3],
      "y":[1,2,3]
    }
  ],
  "layout":{
    "title":{
      "text":"Hi from F#"
    }
  },
  "config":{}
}
```

plotly.js json

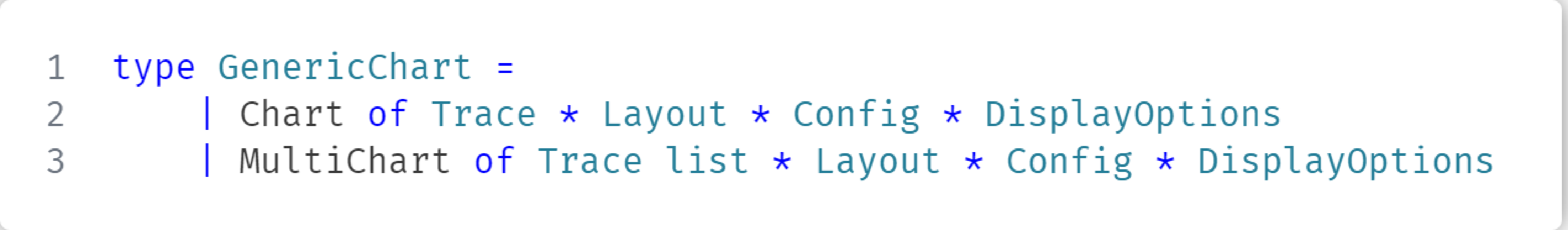


```
let data = Trace("scatter")
data?x ← [1; 2; 3]
data?y ← [1; 2; 3]
data?mode ← "markers"

let layout = Layout()
let title = DynamicObj()
title?text ← "Hi from F#"
layout?title ← title

data
▷ GenericChart.ofTraceObject false
▷ GenericChart.setLayout layout
```

Plotly.NET: DynamicObj layer

Increasingly typed abstractions: **DynamicObj**

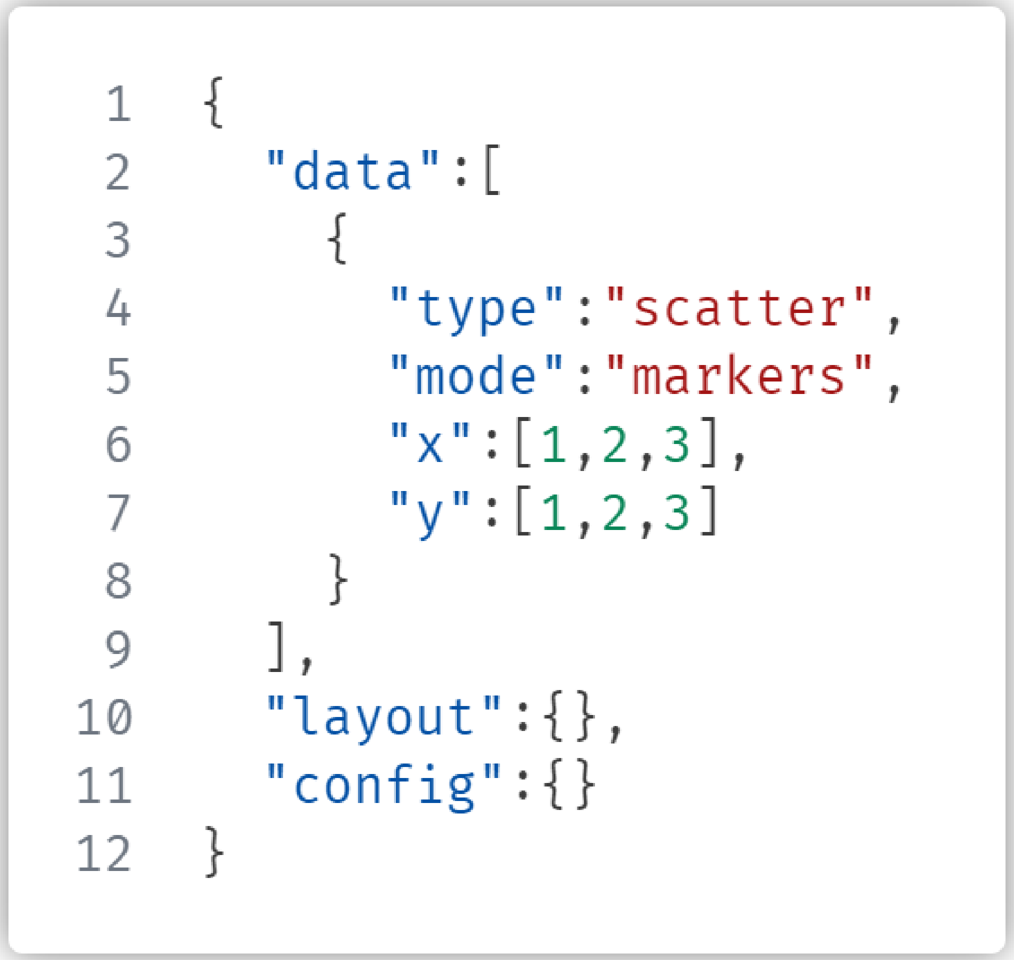
- **GenericChart**: central chart object representation



```
type GenericChart =
    | Chart of Trace * Layout * Config * DisplayOptions
    | MultiChart of Trace list * Layout * Config * DisplayOptions
```

Increasingly typed abstractions: **DynamicObj**

- Low-level API
- Dynamic members on any type by inheritance
- Any plotly object can be created like this
- Knowledge of plotly json schema necessary

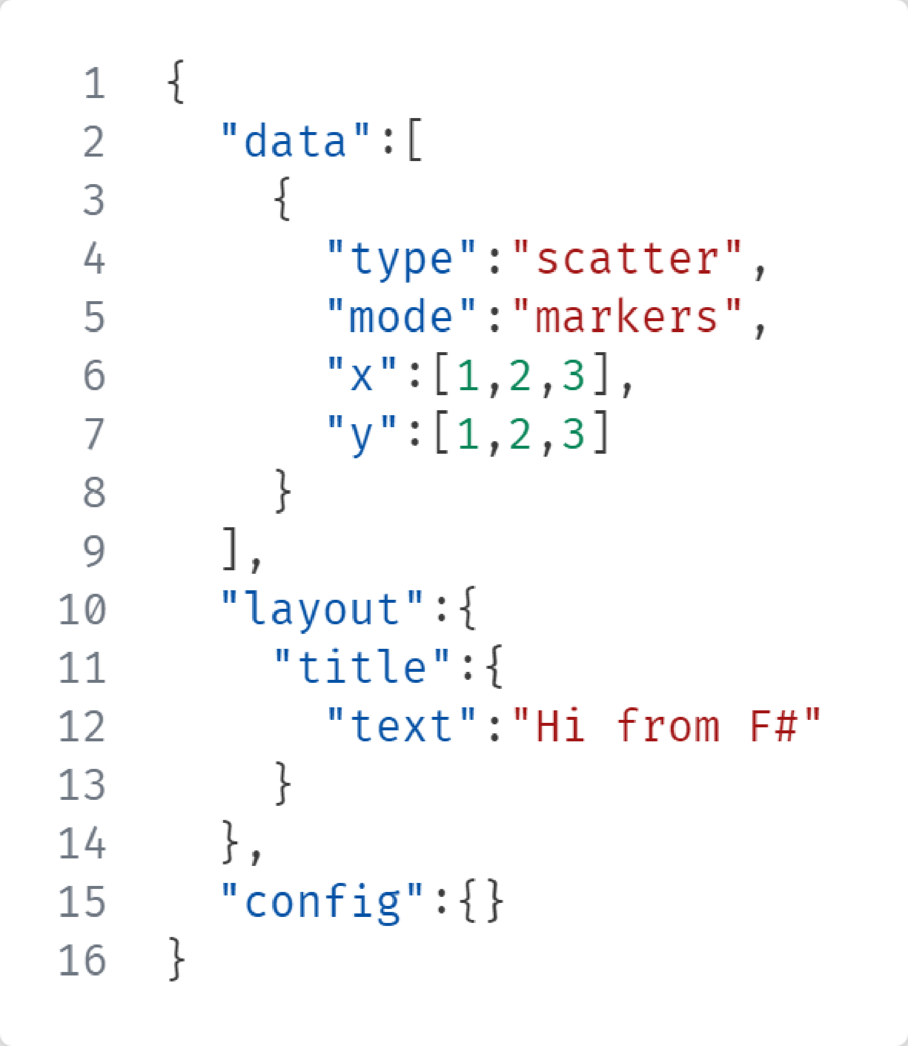


```
{
  "data":[
    {
      "type":"scatter",
      "mode":"markers",
      "x":[1,2,3],
      "y":[1,2,3]
    }
  ],
  "layout":{},
  "config":{}
}
```

Increasingly typed abstractions: **Object mappings**

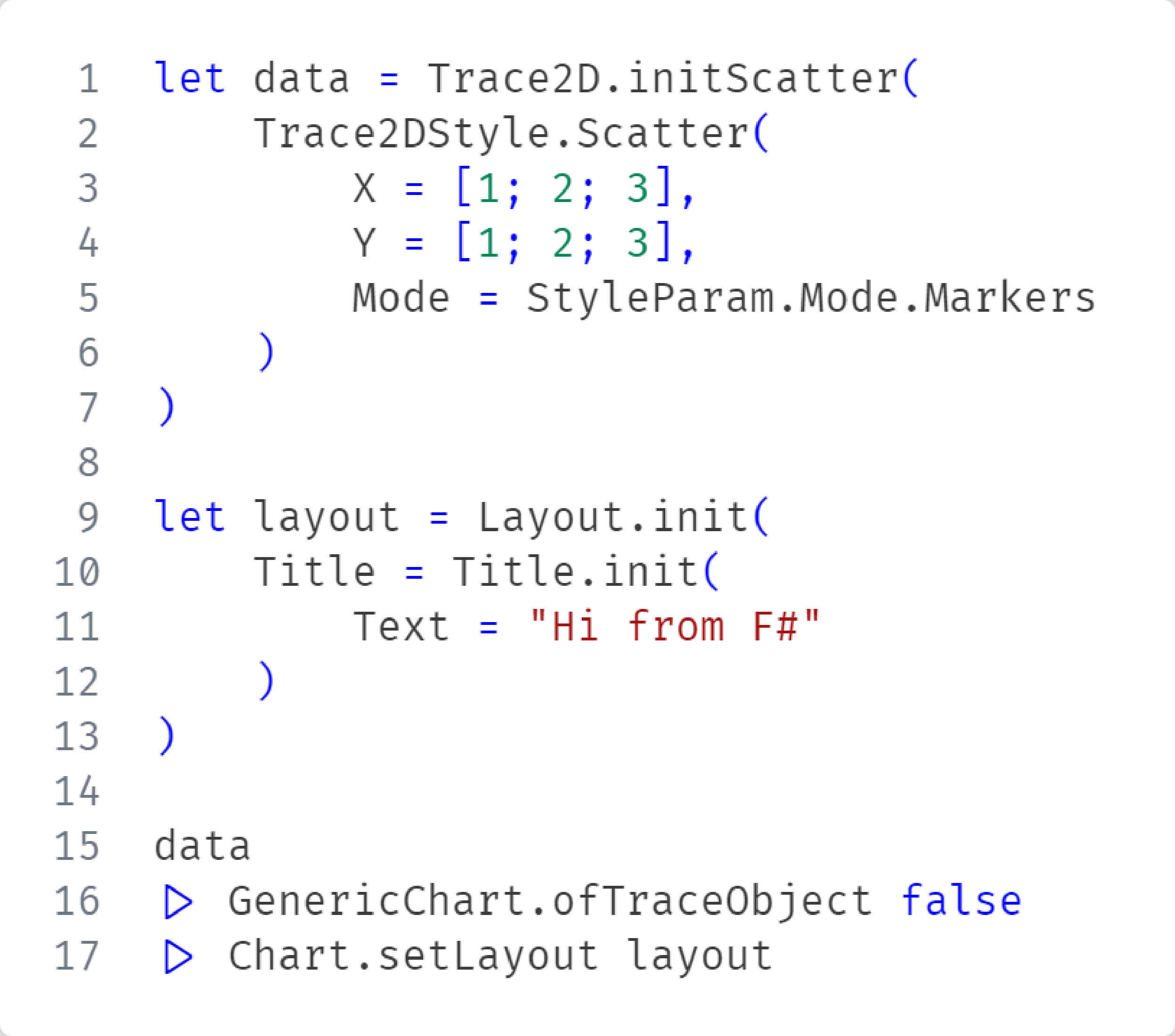
- Type-safe attributes
- Typed style parameters

Increasingly typed abstractions: **Object mappings**



```json
{
  "data":[
    {
      "type":"scatter",
      "mode":"markers",
      "x":[1,2,3],
      "y":[1,2,3]
    }
  ],
  "layout":{
    "title":{
      "text":"Hi from F#"
    }
  },
  "config":{}
}
```

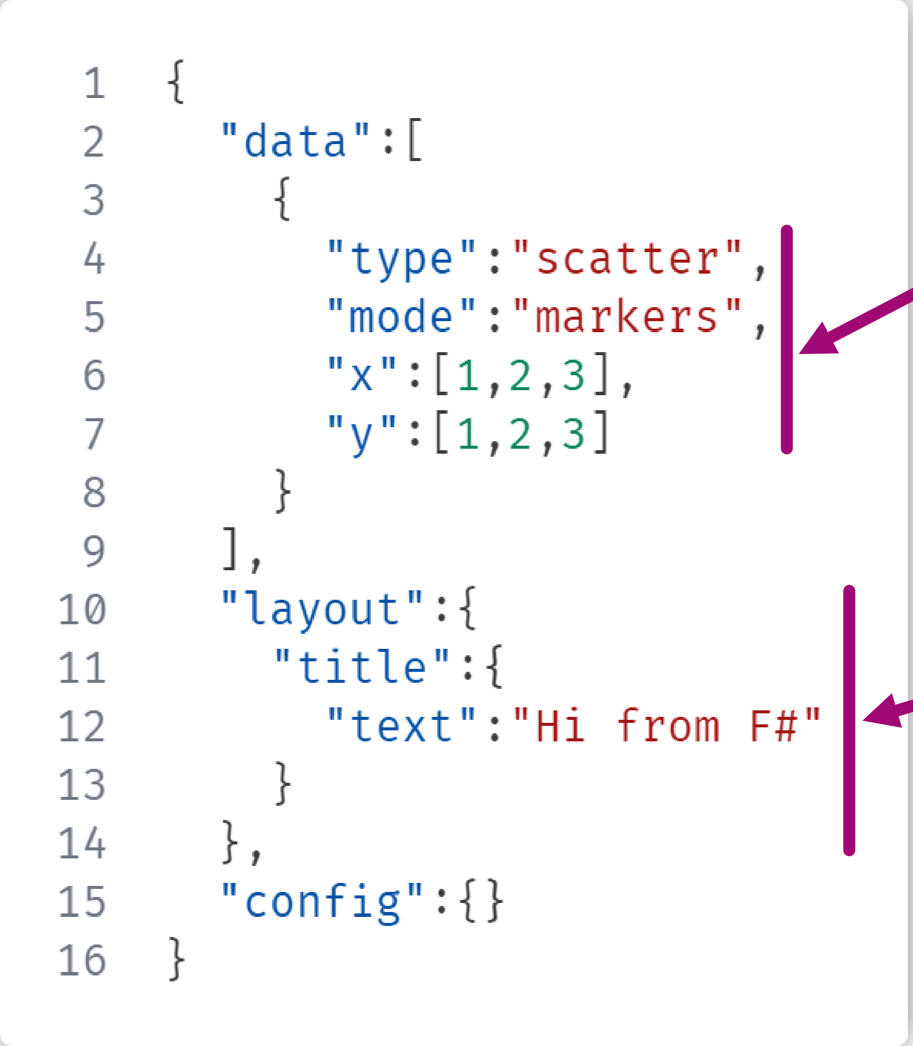
plotly.js json



```fsharp
let data = Trace2D.initScatter(
    Trace2DStyle.Scatter(
        X = [1; 2; 3],
        Y = [1; 2; 3],
        Mode = StyleParam.Mode.Markers
    )
)

let layout = Layout.init(
    Title = Title.init(
        Text = "Hi from F#"
    )
)

data
▷ GenericChart.ofTraceObject false
▷ Chart.setLayout layout
```

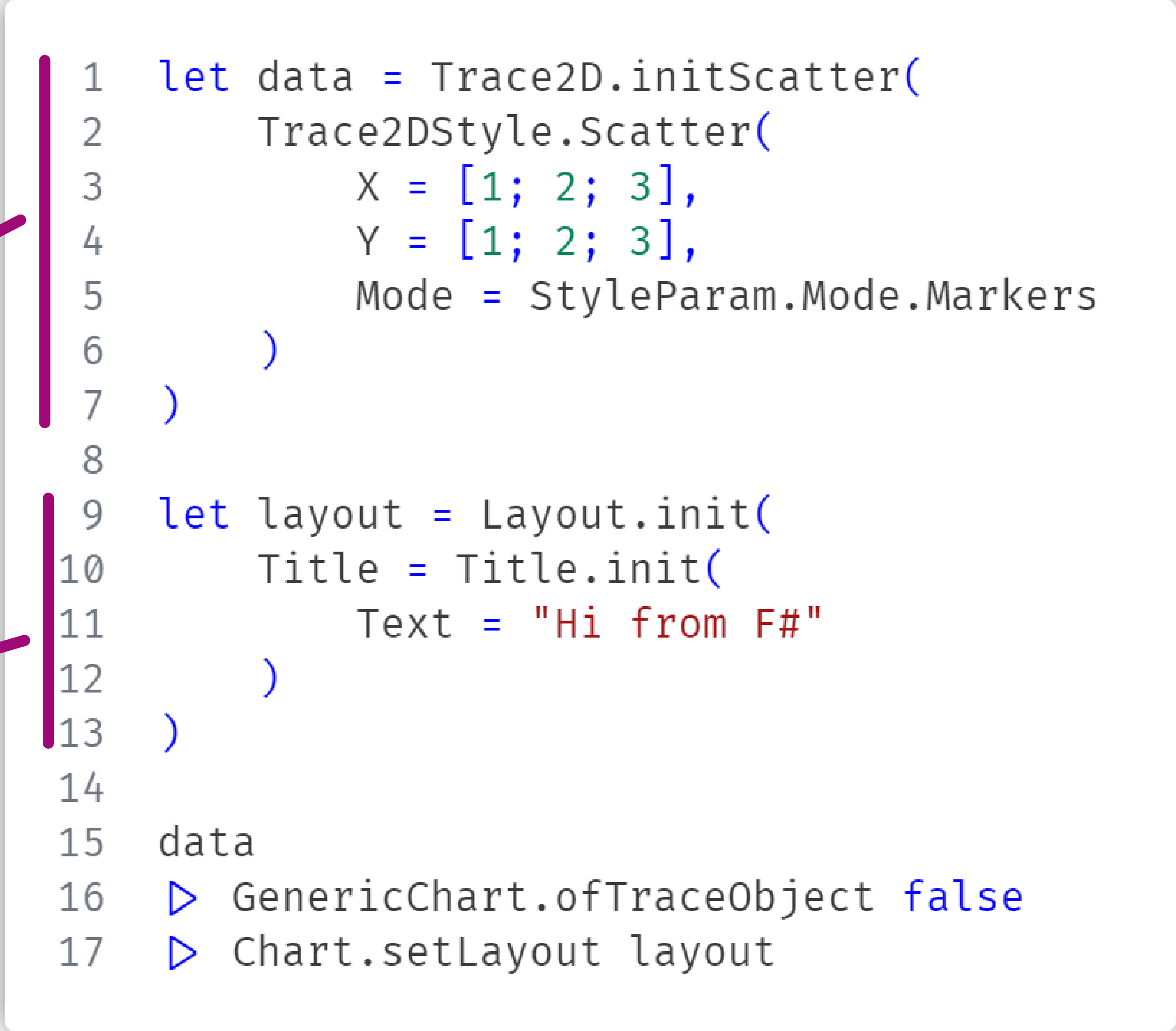
Plotly.NET: Object mappings

Increasingly typed abstractions: **Object mappings**



```json
{
  "data":[
    {
      "type":"scatter",
      "mode":"markers",
      "x":[1,2,3],
      "y":[1,2,3]
    }
  ],
  "layout":{
    "title":{
      "text":"Hi from F#"
    }
  },
  "config":{}
}
```

plotly.js json

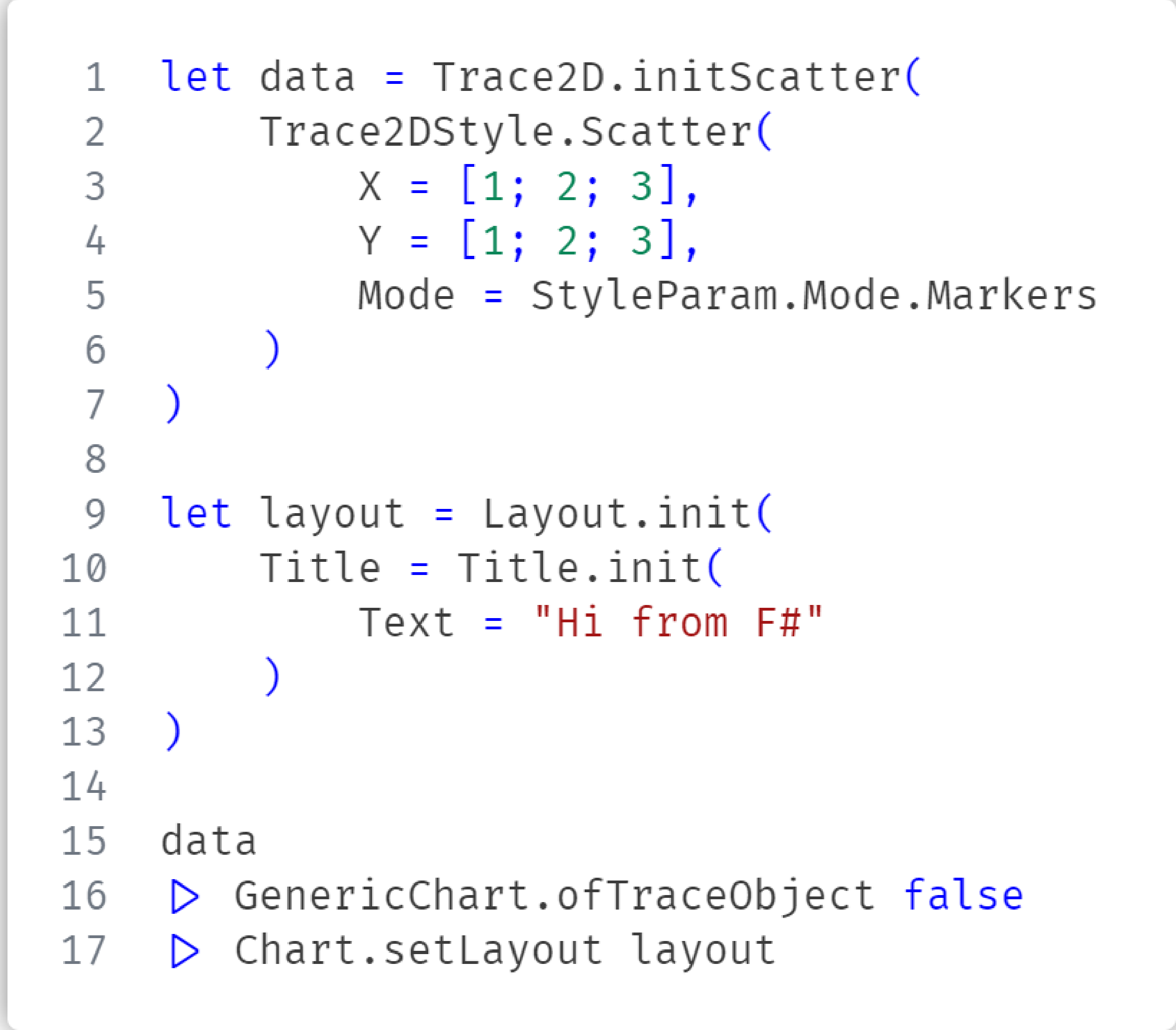


```fsharp
let data = Trace2D.initScatter(
    Trace2DStyle.Scatter(
        X = [1; 2; 3],
        Y = [1; 2; 3],
        Mode = StyleParam.Mode.Markers
    )
)

let layout = Layout.init(
    Title = Title.init(
        Text = "Hi from F#"
    )
)

data
▷ GenericChart.ofTraceObject false
▷ Chart.setLayout layout
```

Plotly.NET: Object mappings

Increasingly typed abstractions: **Object mappings**

- Type-safe attributes
- Typed style parameters
- Declarative syntax
- Less knowledge of plotly json schema necessary
- **Subset** of allowed types in JSON schema



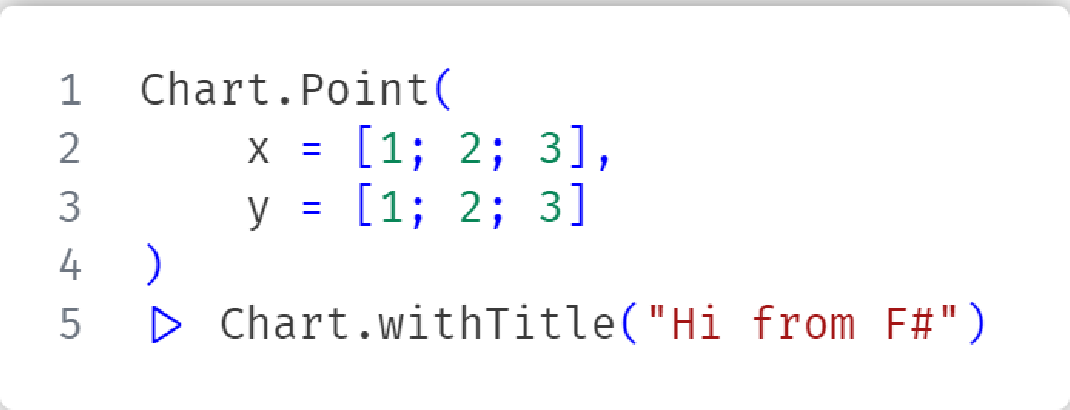
```
let data = Trace2D.initScatter(
    Trace2DStyle.Scatter(
        X = [1; 2; 3],
        Y = [1; 2; 3],
        Mode = StyleParam.Mode.Markers
    )
)

let layout = Layout.init(
    Title = Title.init(
        Text = "Hi from F#"
    )
)

data
|> GenericChart.ofTraceObject false
|> Chart.setLayout layout
```

Plotly.NET: Object mappings

Increasingly typed abstractions: **Chart API**

- Every visualization is a **`Chart`**
- Chart creation (e.g., **`Chart.Point`**, **`Chart.Histogram`**)

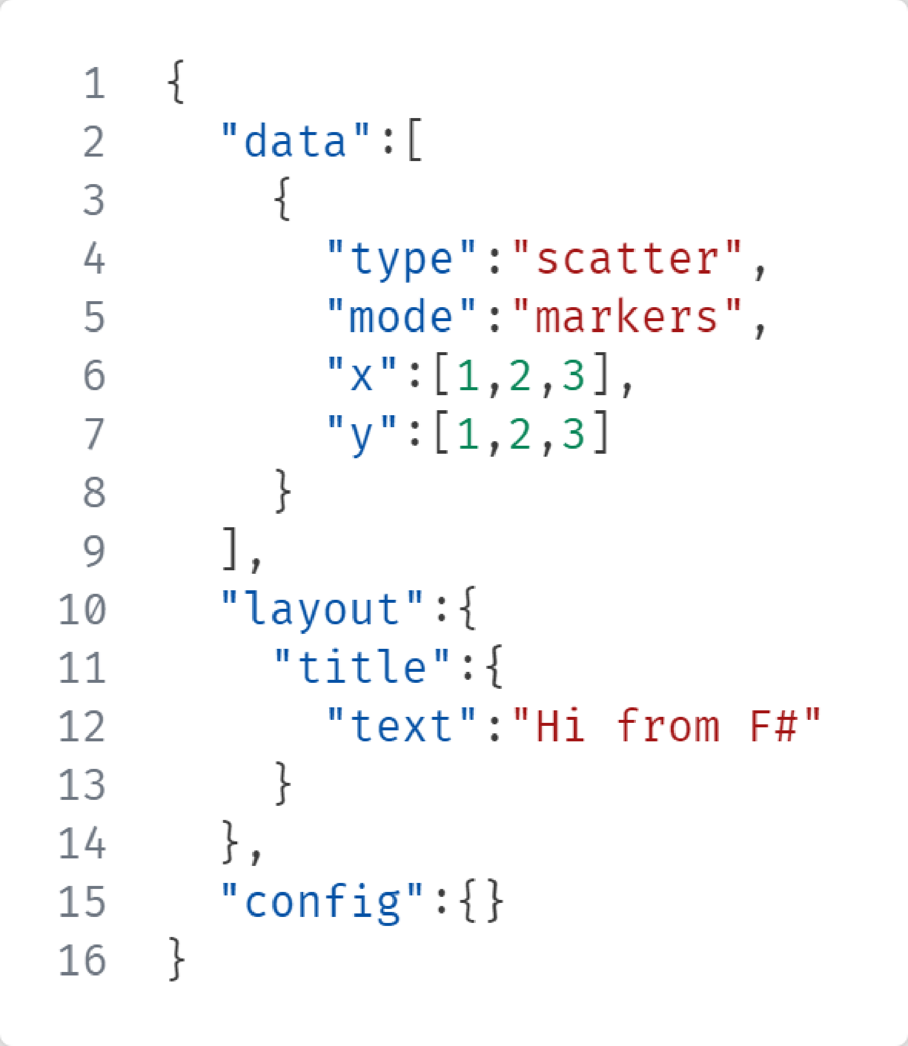
Increasingly strongly typed abstractions: **Chart API**



```
Chart.Point(
    x = [1; 2; 3],
    y = [1; 2; 3]
)
|> Chart.withTitle("Hi from F#")
```

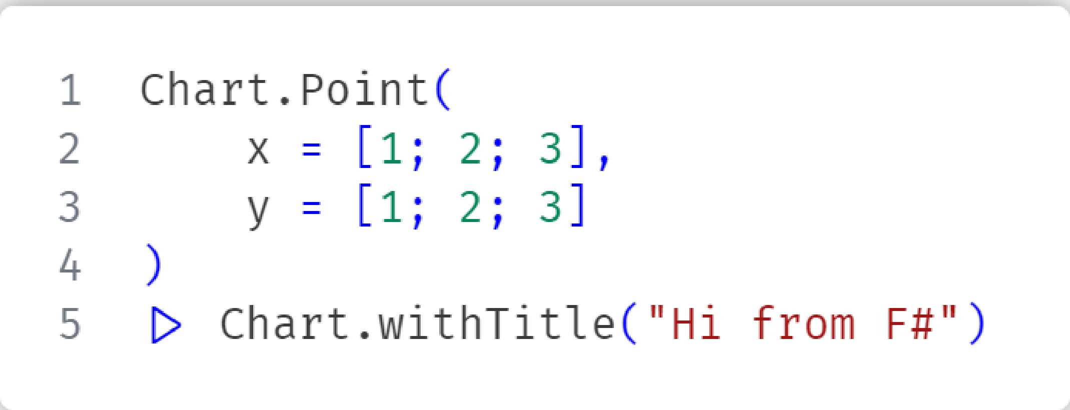
Plotly.NET:
Chart API

Increasingly strongly typed abstractions: **Chart API**

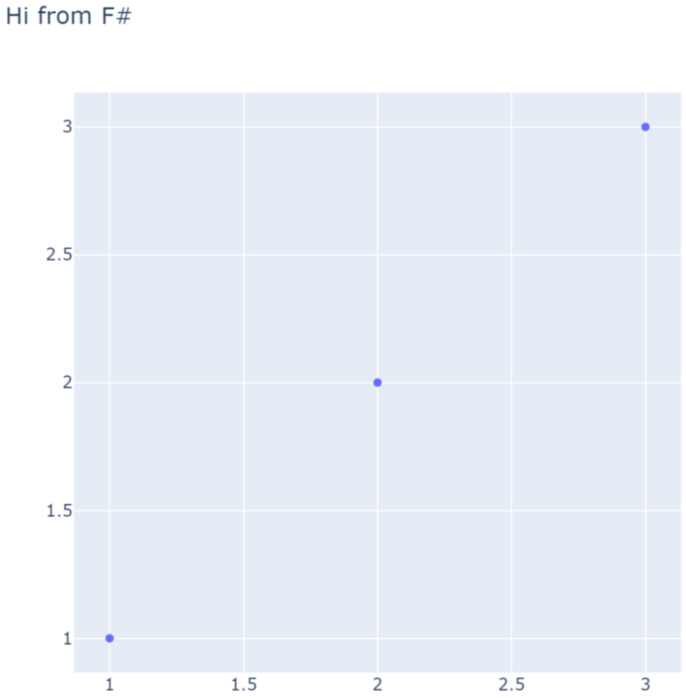


```json
{
  "data":[
    {
      "type":"scatter",
      "mode":"markers",
      "x":[1,2,3],
      "y":[1,2,3]
    }
  ],
  "layout":{
    "title":{
      "text":"Hi from F#"
    }
  },
  "config":{}
}
```

plotly.js json



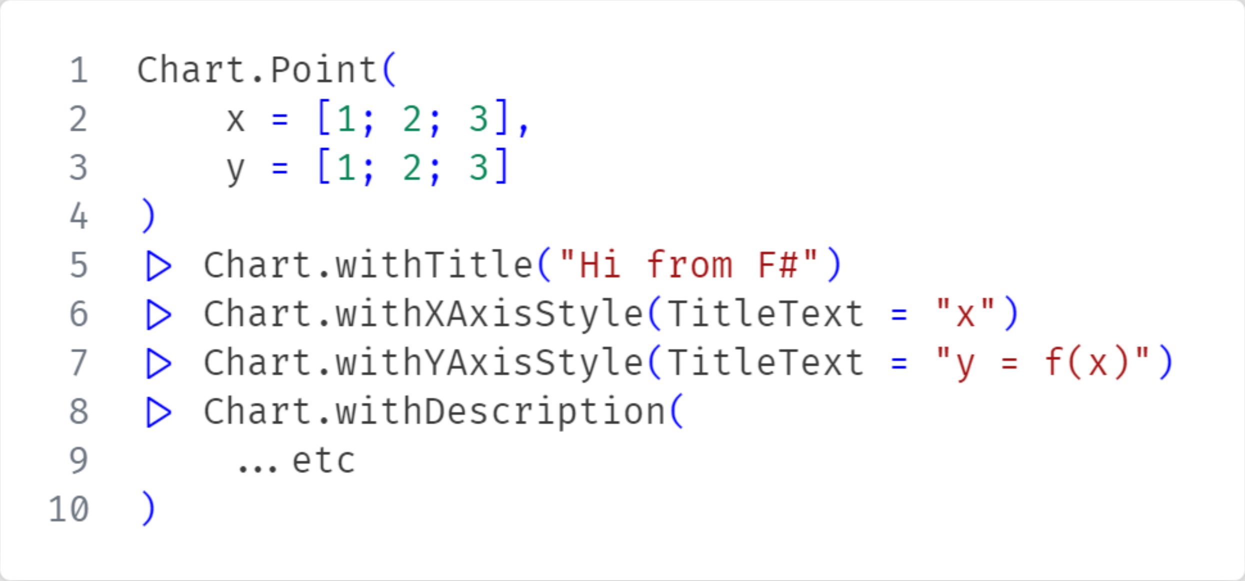
```fsharp
Chart.Point(
    x = [1; 2; 3],
    y = [1; 2; 3]
)
|> Chart.withTitle("Hi from F#")
```



Plotly.NET:
Chart API

Increasingly strongly typed abstractions: **Chart API**

- Every visualization is a **`Chart`**
- Chart creation (e.g., **`Chart.Point`**, **`Chart.Histogram`**)
- Incremental chart styling pipelines (e.g., **`Chart.withTitle`**)



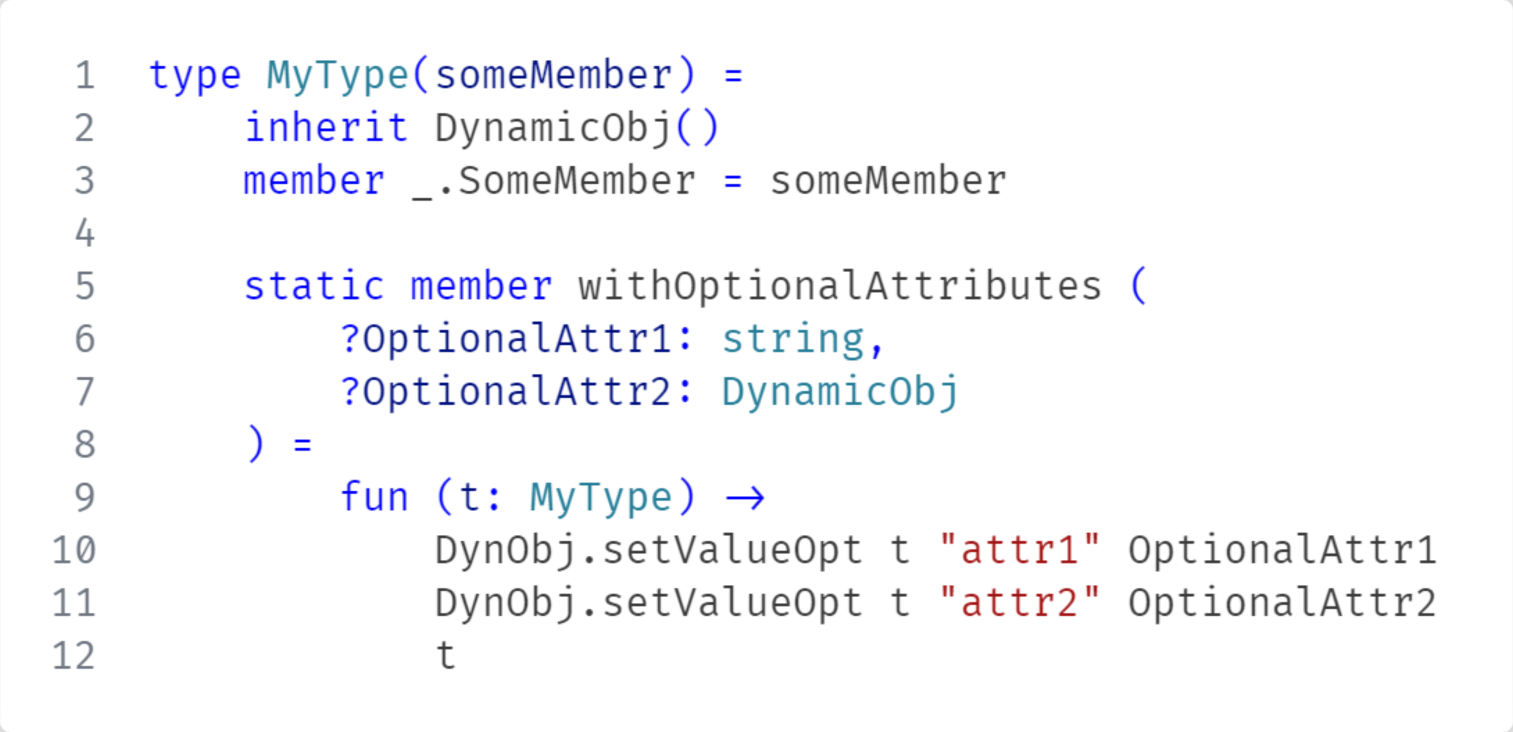
```
Chart.Point(
    x = [1; 2; 3],
    y = [1; 2; 3]
)
|> Chart.withTitle("Hi from F#")
|> Chart.withXAxisStyle(TitleText = "x")
|> Chart.withYAxisStyle(TitleText = "y = f(x)")
|> Chart.withDescription(
     ...etc
)
```

Increasingly strongly typed abstractions: **Chart API**

- Every visualization is a **`Chart`**
- Chart creation (e.g., **`Chart.Point`**, **`Chart.Histogram`**)
- Incremental chart styling pipelines
- **No plotly.js knowledge needed**
- Can be adapted to support other chart backends

functional pipelining with **optional parameters**

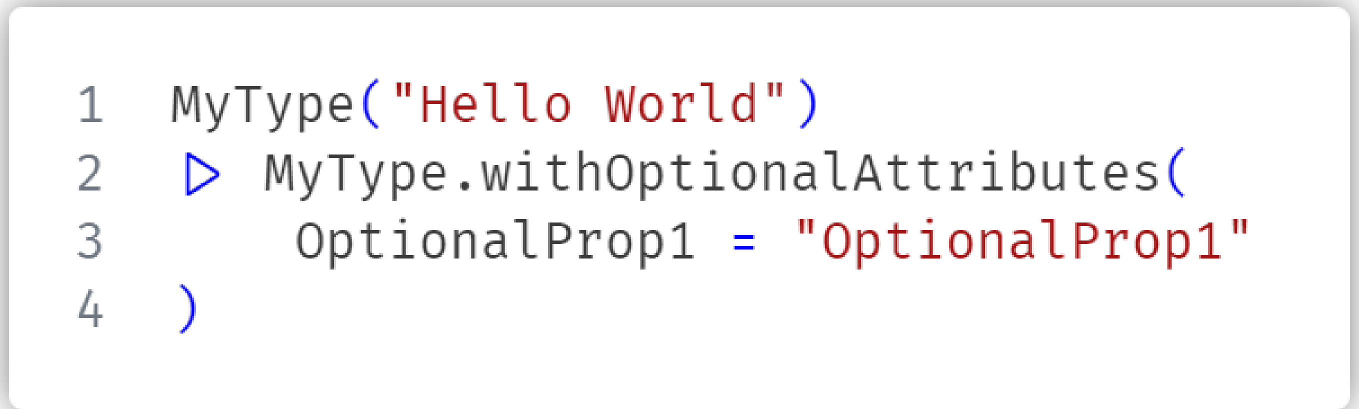
- Virtually all **attributes** in the plotly json schema are **optional**
- **There are many (Layout: >70)**



```
type MyType(someMember) =
    inherit DynamicObj()
    member _.SomeMember = someMember

    static member withOptionalAttributes (
        ?OptionalAttr1: string,
        ?OptionalAttr2: DynamicObj
    ) =
        fun (t: MyType) →
            DynObj.setValueOpt t "attr1" OptionalAttr1
            DynObj.setValueOpt t "attr2" OptionalAttr2
            t
```

functional pipelining with **optional parameters**

- Virtually all **attributes** in the plotly json schema are **optional**
- **There are many (Layout: >70)**



```
MyType("Hello World")
▷ MyType.withOptionalAttributes(
    OptionalProp1 = "OptionalProp1"
)
```

Learnings from Plotly.NET

- Community visibility is important (external success factors):
  - Recognizable GitHub organization
  - Blog posts
  - Regularly inform about project state and progress

Learnings from Plotly.NET

- Pragmatic API choices
  - Focus on understandable APIs
  - Use ‘impure’ language features
  - High-level API layer is important

- Exhaustive documentation:
  - Focus on API reference, add examples step-by-step

Learnings from Plotly.NET: closing remarks

- Do not dismiss C# compatibility
  - huge potential userbase
  - C# bindings can be auto-generated

- We need more F# people in review pools for academic journals
  - Review process took > 3 months

Thank you for your attention!

Plotly.NET