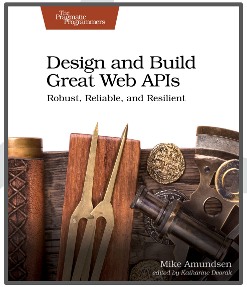
Five API Styles

Mike Amundsen

@mamund

training.amundsen.com



The Pragmatic Programmers

Design and Build Great Web APIs

Robust, Reliable, and Resilient

Mike Amundsen

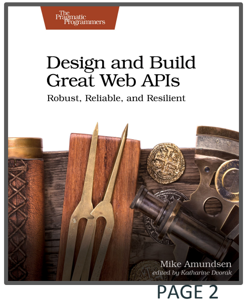
edited by Katharine Dvorak

Why API Styles?

- Architects of networked/distributed applications have many decisions to make

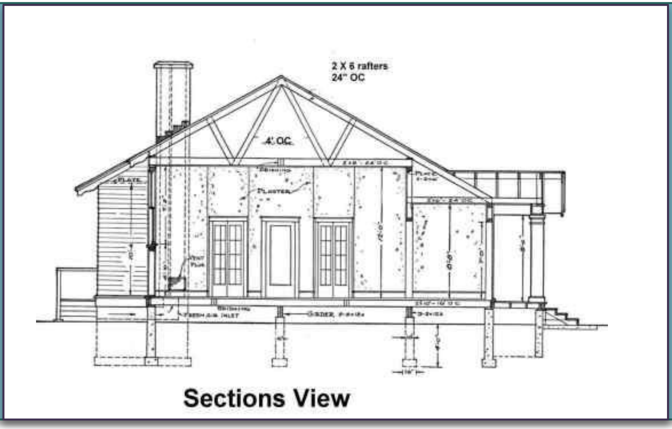
- Technology changing quickly, new implementations every year:
  - graphQL, gRPC, Kafka, HAL+Forms

- Which *models* of component interaction work best?


The Pragmatic Programmers
Design and Build Great Web APIs
Robust, Reliable, and Resilient
Mike Amundsen
edited by Katharine Dvorak

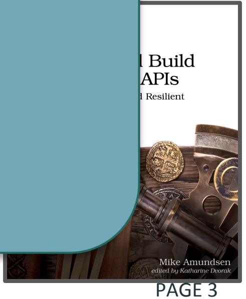
The Value of Styles for the Designer

Design a House



Sections View

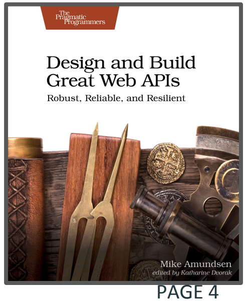
Design a *Victorian Style* House


l Build
APIs
l Resilient
Mike Amundsen
edited by Katharine Dvorak

Styles, Not Standards

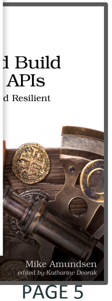
- Standards

  “Usually a formal document that establishes uniform engineering or technical criteria, methods, processes and practices.”


The Pragmatic Programmers
Design and Build Great Web APIs
Robust, Reliable, and Resilient
Mike Amundsen
edited by Katharine Dvorak

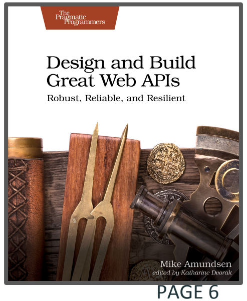
Standards

- **IETF** (HTTP, URI, Basic Auth, etc.)
- **W3C** (SOAP, HTML, RDF, etc.)
- **OASIS** (ebXML, DocBook, WS-Security, etc.)



The Value of Styles for the Designer

- Styles describe:
  - Characteristics
  - Vocabulary
  - Constraints
- The style is a loose set of rules – the rules become a guide
- Styles help designers communicate


The Pragmatic Programmers
Design and Build Great Web APIs
Robust, Reliable, and Resilient
Mike Amundsen
edited by Katharine Dvorak

Tunnel Style

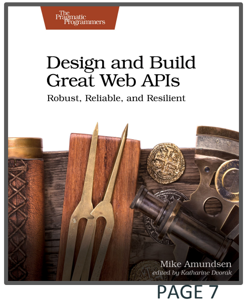
GET
PUT
POST
DELETE

URI Style

Hypermedia Style

Query Style

Event Driven Style



The Pragmatic Programmers

Design and Build Great Web APIs

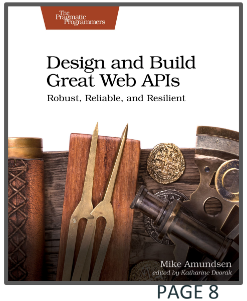
Robust, Reliable, and Resilient

Mike Amundsen
edited by Katharine Dvorak

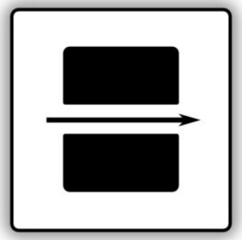
Style Implementation Considerations

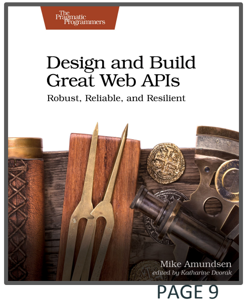
Five properties to consider:

- Scalability
- Usability
- Changeability
- Performance
- Reliability


The Pragmatic Programmers
Design and Build Great Web APIs
Robust, Reliable, and Resilient
Mike Amundsen
edited by Katharine Dvorak

Tunnel Style





The Pragmatic Programmers

Design and Build Great Web APIs

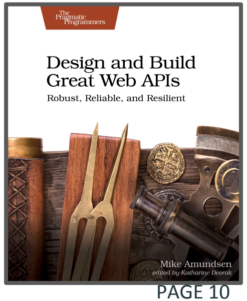
Robust, Reliable, and Resilient

Mike Amundsen

edited by Katharine Dvorak

Tunnel Style: Overview

- Application Layer Protocol with a type system and operations
- HTTP is not usually required (protocol agnostic)
- RPC Interaction
- Examples:
  - XML-RPC (1998)
  - SOAP 1.0 (1999)
  - SOAP 1.2 (2003)
  - JSON-RPC (2005)
  - gRPC (2016)



The Pragmatic Programmers

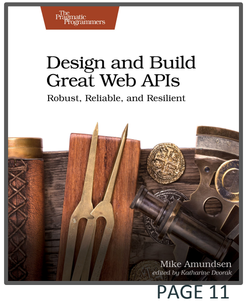
Design and Build Great Web APIs

Robust, Reliable, and Resilient

Mike Amundsen
edited by Katharine Dvorak

Tunnel Style: Characteristics

- Type and Specification Driven (XML-*, WS-*, Protocol Buffers)
- Procedure/Operation based design (“RPC”)
- Similar to imperative programming interfaces



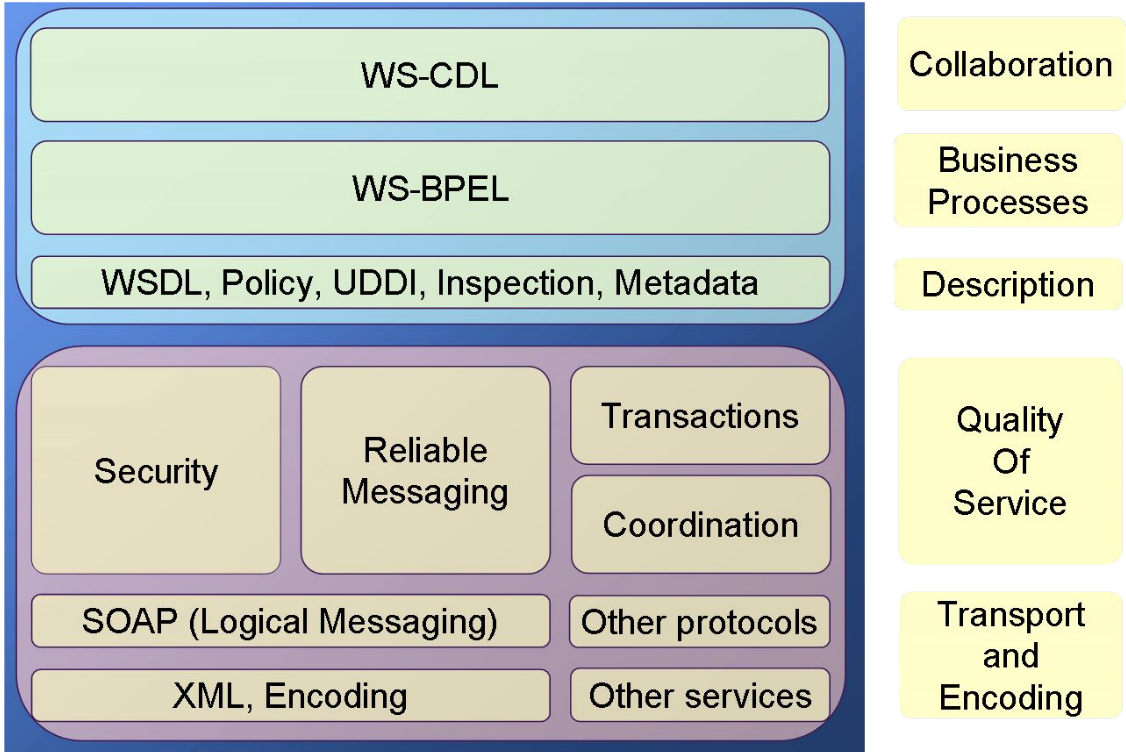
The Pragmatic Programmers

Design and Build Great Web APIs

Robust, Reliable, and Resilient

Mike Amundsen
edited by Katharine Dvorak

Tunnel Style: SOAP Stack Example



WS-CDL

WS-BPEL

WSDL, Policy, UDDI, Inspection, Metadata

Security

Reliable Messaging

Transactions

Coordination

SOAP (Logical Messaging)

Other protocols

XML, Encoding

Other services

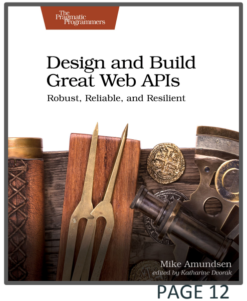
Collaboration

Business Processes

Description

Quality Of Service

Transport and Encoding



The Pragmatic Programmers

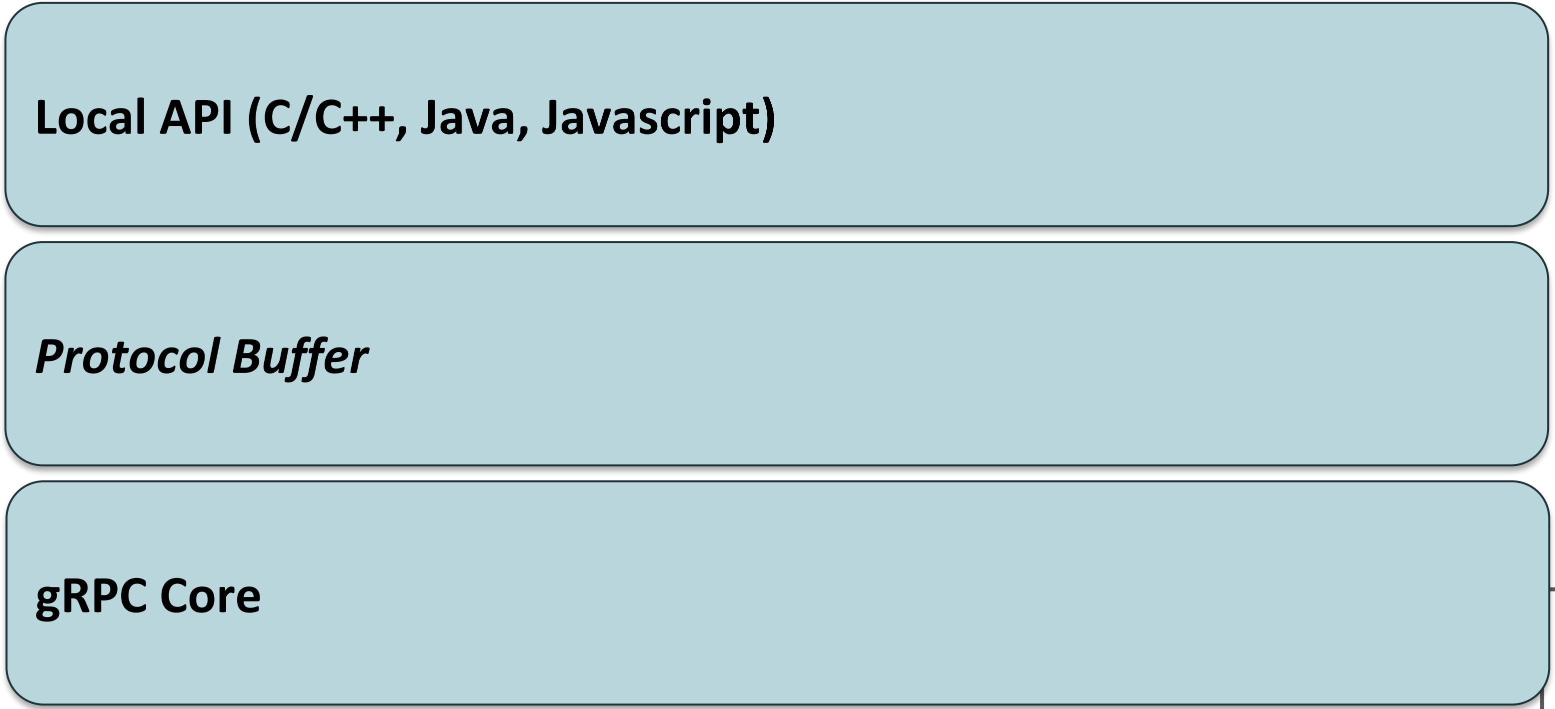
Design and Build Great Web APIs

Robust, Reliable, and Resilient

Mike Amundsen

edited by Katharine Dvorak

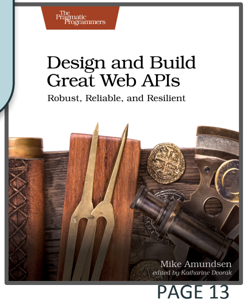
Tunnel Style: gRPC Stack Example



Local API (C/C++, Java, Javascript)

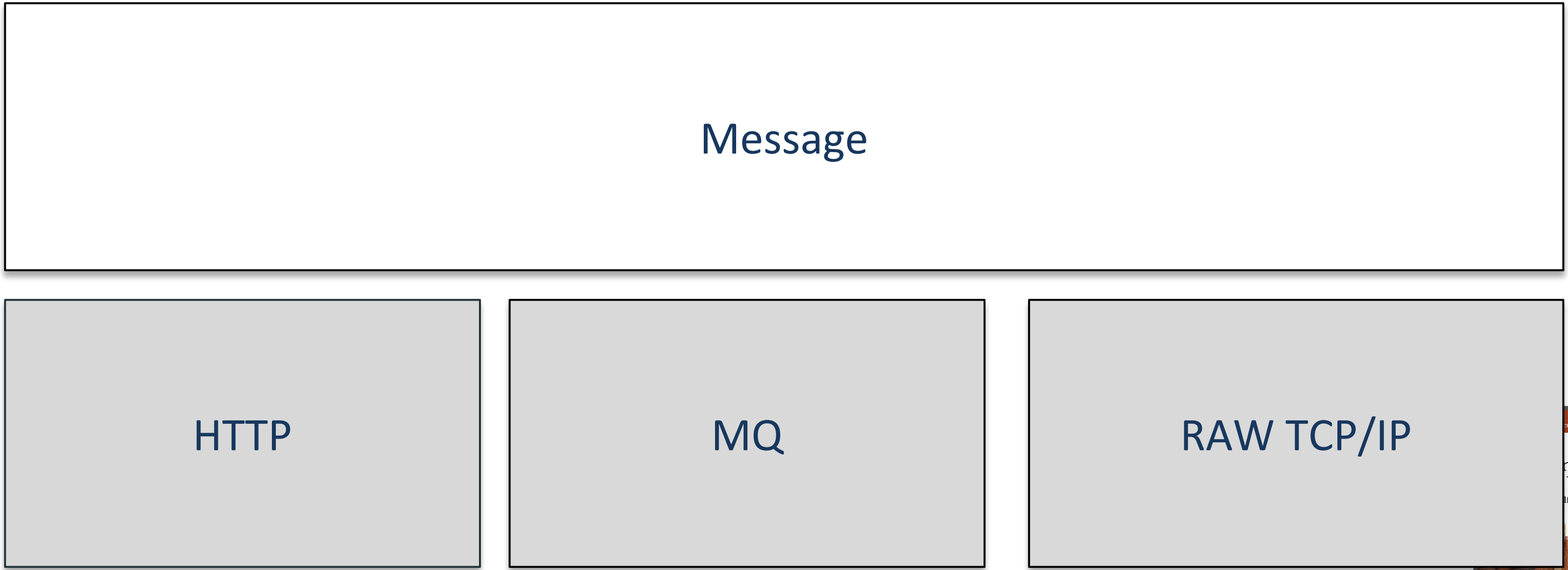
Protocol Buffer

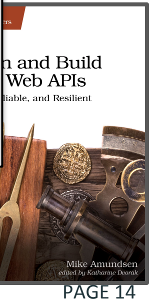
gRPC Core


The Pragmatic Programmers
Design and Build Great Web APIs
Robust, Reliable, and Resilient
Mike Amundsen
edited by Katharine Dvorak

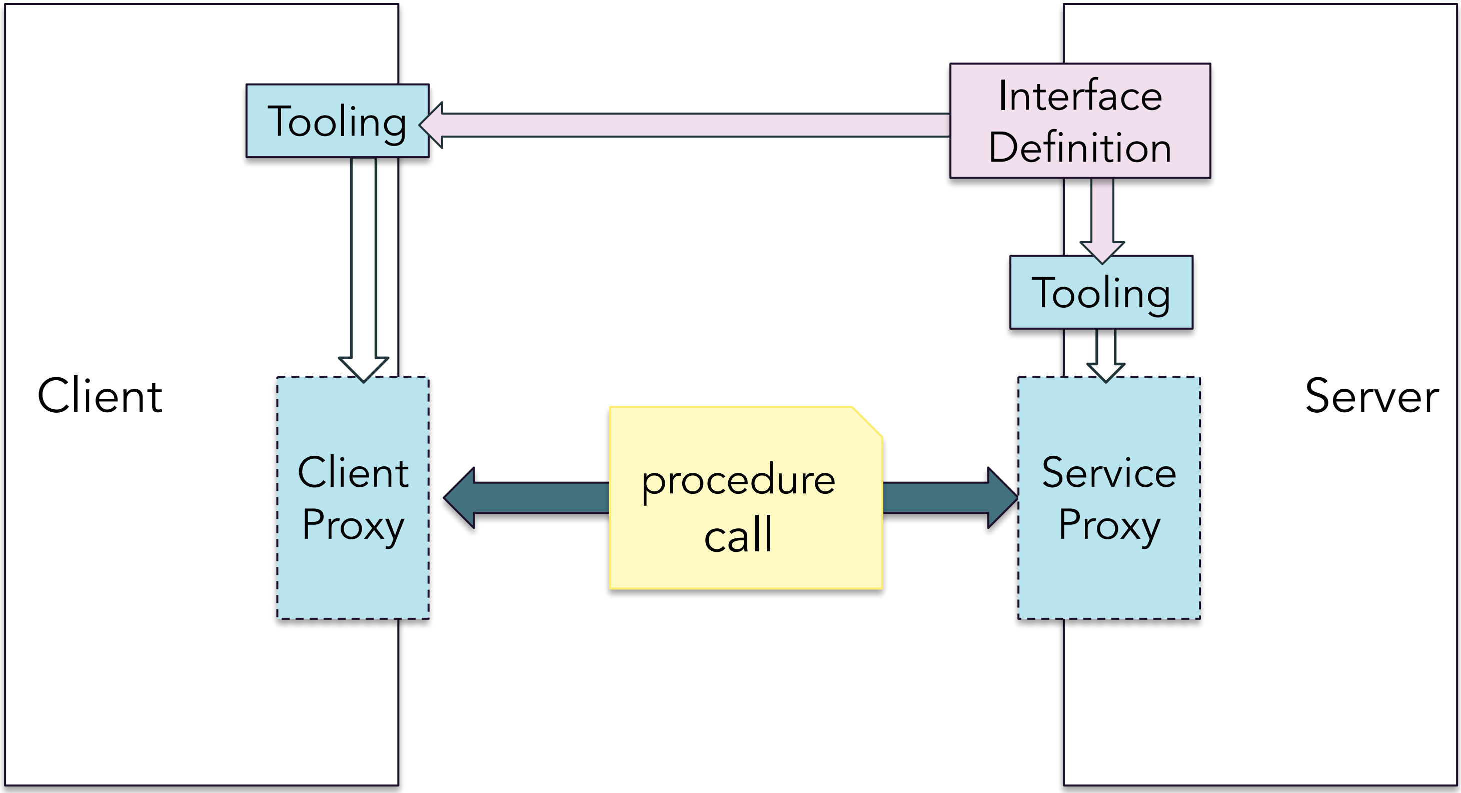
Tunnel Style: Primary Constraint

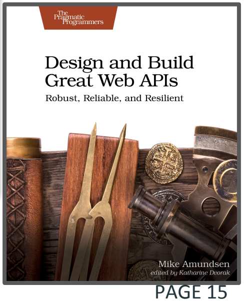
No dependencies on transport layer protocol




n and Build
Web APIs
liable, and Resilient
Mike Amundsen
edited by Katharine Dvorak

Tunnel Style: Common Implementation Model




The Pragmatic Programmers

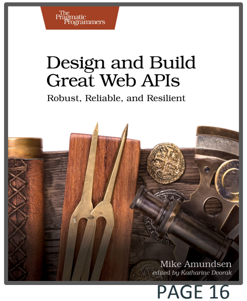
Design and Build Great Web APIs

Robust, Reliable, and Resilient

Mike Amundsen
edited by Katharine Dvorak

Tunnel Style: Benefits

- RPC style is familiar to many developers and architects
- Support for heterogeneous networks
- Messages can be optimized for point-to-point performance (reduced size, reduced latency)



The Pragmatic Programmers

Design and Build Great Web APIs

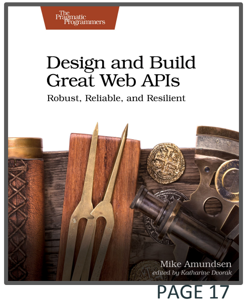
Robust, Reliable, and Resilient

Mike Amundsen

edited by Katharine Dvorak

Tunnel Style: Limitations

- Ignores HTTP features (caching, etc.)
- Limited tooling in mobile and web stacks
- Change is often costly (highly typed, tightly-coupled)



The Pragmatic Programmers

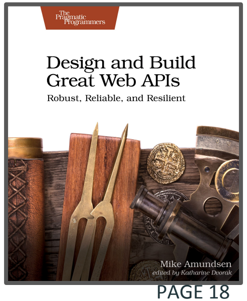
Design and Build Great Web APIs

Robust, Reliable, and Resilient

Mike Amundsen

edited by Katharine Dvorak

URI Style


The Pragmatic Programmers

Design and Build Great Web APIs

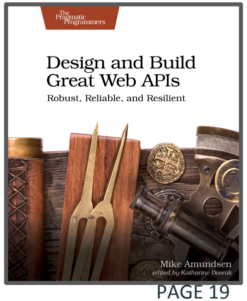
Robust, Reliable, and Resilient

Mike Amundsen

edited by Katharine Dvorak

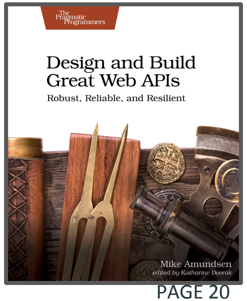
URI Style: Overview

- Uses **C**reate, **R**ead, **U**pdate, **D**elete (CRUD) interaction pattern
- URI points to a target
- Uses HTTP only


The Pragmatic Programmers
Design and Build Great Web APIs
Robust, Reliable, and Resilient
Mike Amundsen
edited by Katharine Dvorak

URI Style: Characteristics

- Use HTTP standard
- **Object** first design
- **Convention** driven
- Similar to data object interactions (DAO)


The Pragmatic Programmers

Design and Build Great Web APIs

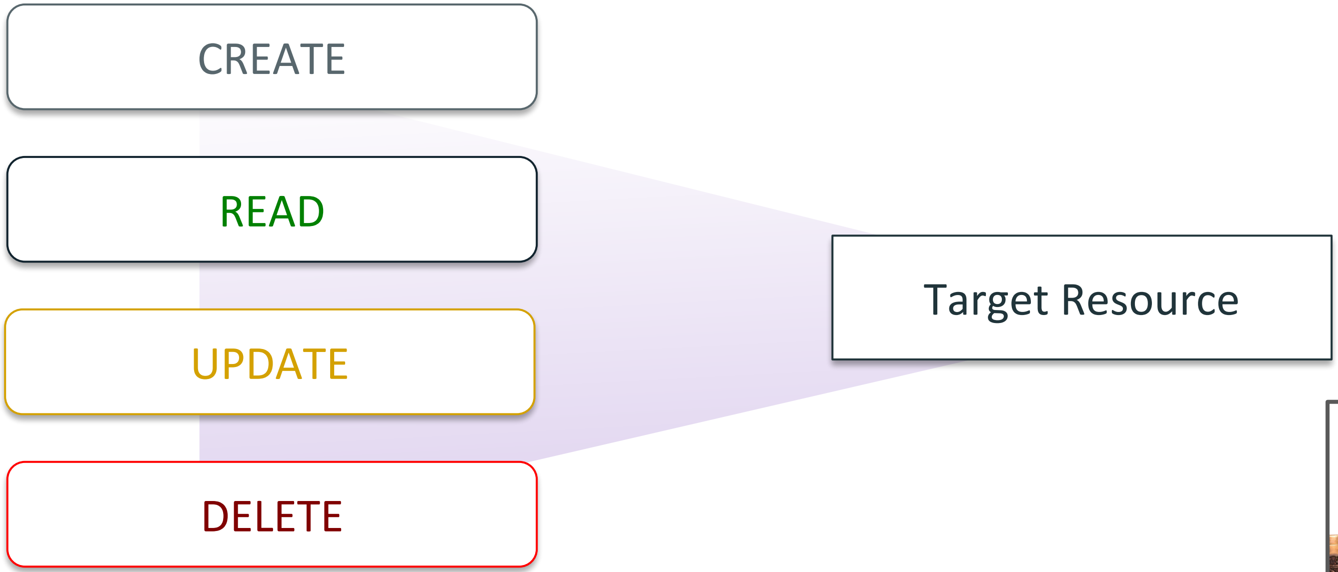
Robust, Reliable, and Resilient

Mike Amundsen
edited by Katharine Dvorak

URI Style: Primary Constraint

GET
PUT
POST
DELETE

- Interactions must use the CRUD Pattern



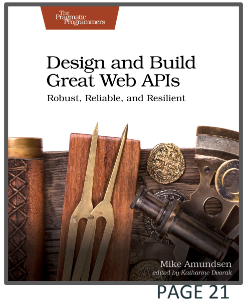
CREATE

READ

UPDATE

DELETE

Target Resource

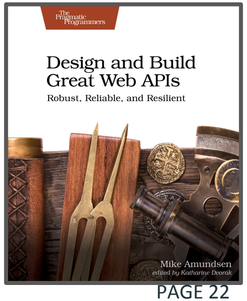

The Pragmatic Programmers
Design and Build Great Web APIs
Robust, Reliable, and Resilient
Mike Amundsen
edited by Katharine Dvorak

URI Style: Example

GET
PUT
POST
DELETE

GET `http://myapi.com/students`

- GET is used for Retrieval (Read)
- http://myapi.com/students points to a collection of student records
- This request means “retrieve a list of student records”

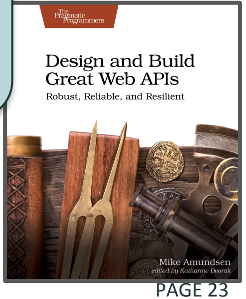

The Pragmatic Programmers
Design and Build Great Web APIs
Robust, Reliable, and Resilient
Mike Amundsen
edited by Katharine Dvorak

URI Style: Protocol Stack

MIME (for representation)
XML, JSON, CSV, etc…

URI (for identification)
RFC 3986

HTTP (transport)
RFC 7231 et al

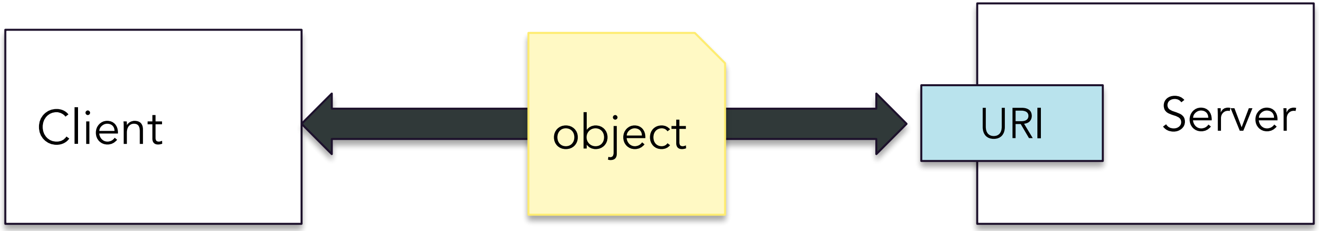

The Pragmatic Programmers

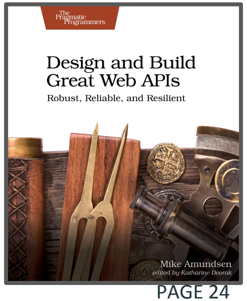
Design and Build Great Web APIs

Robust, Reliable, and Resilient

Mike Amundsen
edited by Katharine Dvorak

URI Style: Common Implementation



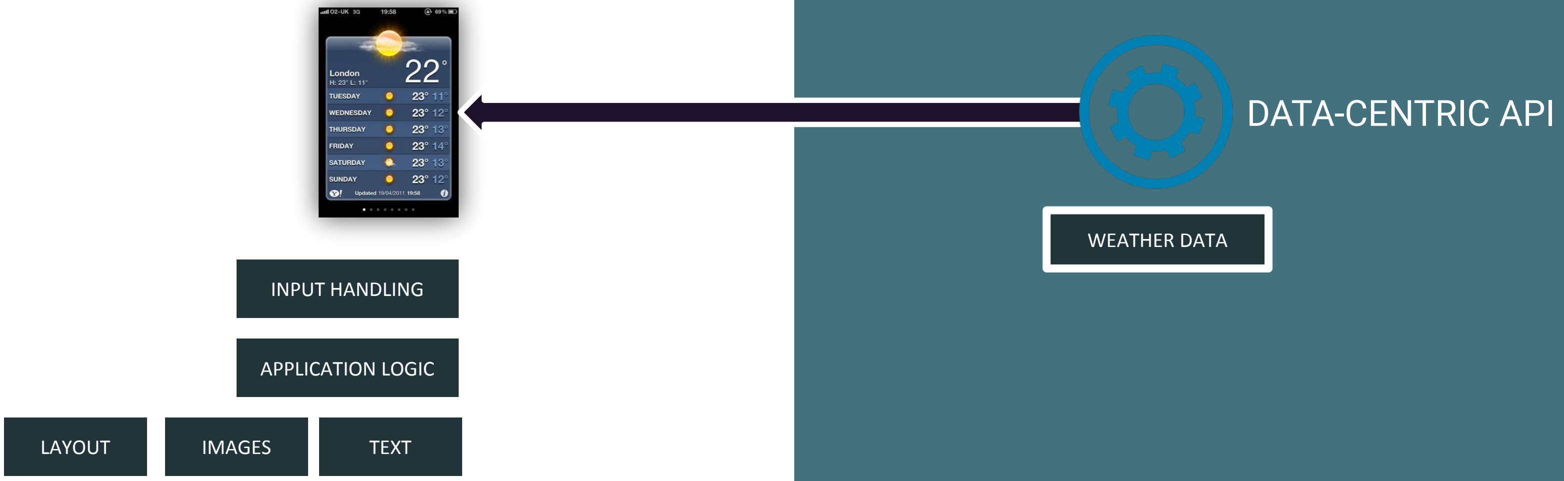

The Pragmatic Programmers
Design and Build Great Web APIs
Robust, Reliable, and Resilient
Mike Amundsen
edited by Katharine Dvorak

URI Style: Ideal for Data-Centric APIs

GET
PUT
POST
DELETE

Frontend
(client)

Backend
(server)



DATA-CENTRIC API

WEATHER DATA

INPUT HANDLING

APPLICATION LOGIC

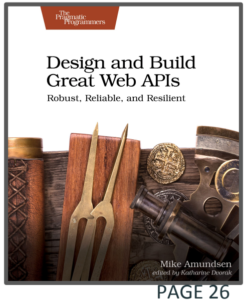
LAYOUT

IMAGES

TEXT

URI Style: Benefits

- HTTP path & query is a “well known” – improved usability for many developers
- CRUD pattern is simple and a good fit for “data service” pattern
- Large ecosystem of tools and frameworks today



The Pragmatic Programmers

Design and Build Great Web APIs

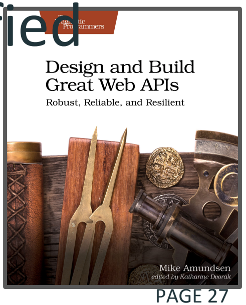
Robust, Reliable, and Resilient

Mike Amundsen
edited by Katharine Dvorak

URI Style: Limitations

GET
PUT
POST
DELETE

- CRUD pattern is limited
- URI modelling is not standard – every API is a “snowflake”
  - Internal domains can benefit from a style guide
  - External domains (partner/public) may suffer
- Can be “chatty” (esp. when the object passing pattern is used)
- API changes usually require client changes, cost is magnified by scale of client components



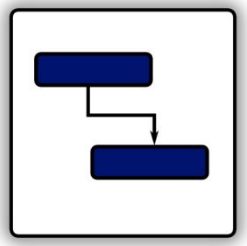
Design and Build Great Web APIs

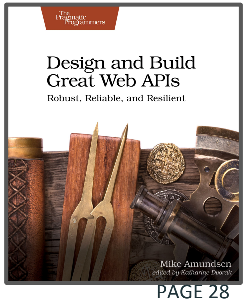
Robust, Reliable, and Resilient

Mike Amundsen

edited by Katharine Dvorak

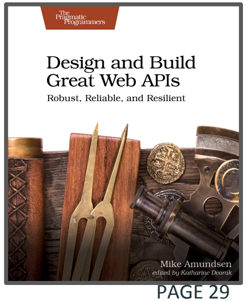
Hypermedia Style




The Pragmatic Programmers
Design and Build Great Web APIs
Robust, Reliable, and Resilient
Mike Amundsen
edited by Katharine Dvorak

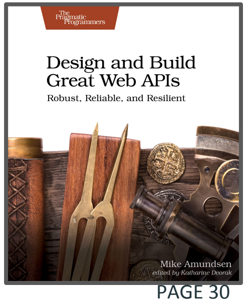
Hypermedia Style: Overview

- An API with hypermedia features
- A **browser-like** experience for machines
- Implemented with links and forms
- Example:
  - REST (Roy Fielding dissertation)


The Pragmatic Programmers
Design and Build Great Web APIs
Robust, Reliable, and Resilient
Mike Amundsen
edited by Katharine Dvorak

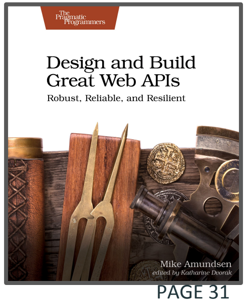
Hypermedia Style: Characteristics

- Focus on transitions
- URI is not an object key
- Messages are self-documenting


The Pragmatic Programmers
Design and Build Great Web APIs
Robust, Reliable, and Resilient
Mike Amundsen
edited by Katharine Dvorak

Hypermedia Style: Primary Constraint

- Uniform Interface
  - Identification of resources
  - Manipulation of resources through representations
  - Self-descriptive messages
  - Hypermedia as the engine of application state



The Pragmatic Programmers

Design and Build Great Web APIs

Robust, Reliable, and Resilient

Mike Amundsen
edited by Katharine Dvorak

Hypermedia Style: Protocol Stack

Profiles and Link Relations

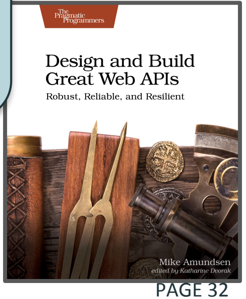
hCard, next, prev, etc..

Media Type

HTML, ATOM, HAL+JSON, Collection+JSON, etc..

Transport Protocol

HTTP, COAP, etc…



The Pragmatic Programmers

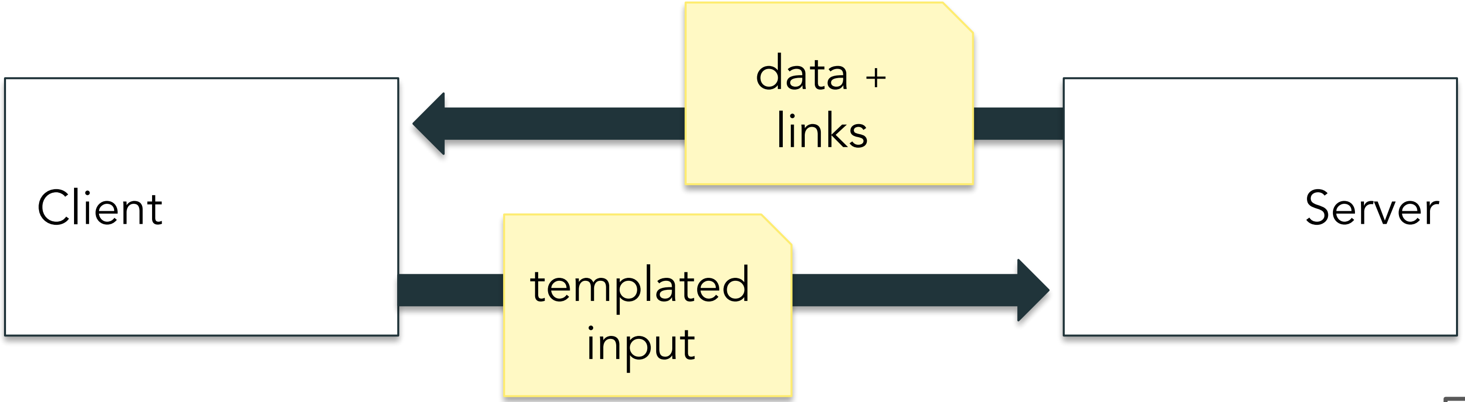
Design and Build Great Web APIs

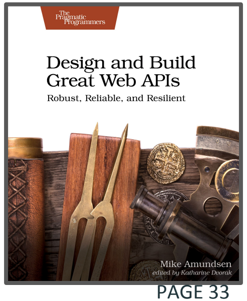
Robust, Reliable, and Resilient

Mike Amundsen

edited by Katharine Dvorak

Hypermedia Style: Common Implementation

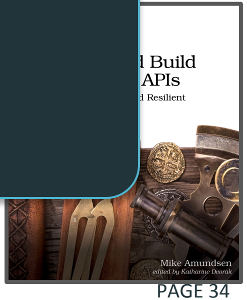



The Pragmatic Programmers
Design and Build Great Web APIs
Robust, Reliable, and Resilient
Mike Amundsen
edited by Katharine Dvorak

Hypermedia Style Example: Links

- Links tell the client what it can do next

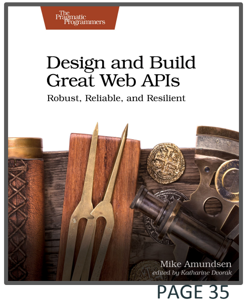
```
<html>
<body>
<h1>Student Records</h1>
<a href=“/detail?id=3”>Ronnie Mitra</a>
</body>
</html>
```



Hypermedia Style Example: Links in JSON

```
{
   “name”: “Ronnie”,
   “enrollment-year”: “2014”
}
```

A URI Style JSON Response



The Pragmatic Programmers

Design and Build Great Web APIs

Robust, Reliable, and Resilient

Mike Amundsen

edited by Katharine Dvorak

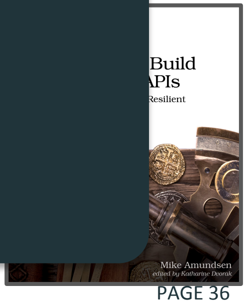
Hypermedia Style Example: Links in JSON

```
{
   “name”: “Ronnie”,
   “enrollment-year”: “2014”
}
```

A URI Style JSON Response

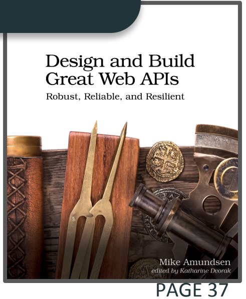
```
{
   “name”: “Ronnie”,
   “enrollment-year”: “2014”,
   “_address_details”: “/student/ronnie/address”
}
```

A Hypermedia Style JSON Response



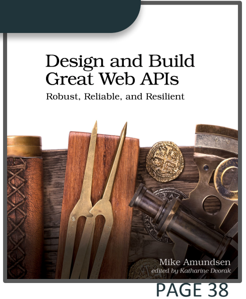
Hypermedia Style Example: Generic Data

```
{
   “name”: “Ronnie”,
   “enrollment-year”: “2014”,
   “_address_details”: “/student/ronnie/address”
}
```


Design and Build Great Web APIs
Robust, Reliable, and Resilient
Mike Amundsen
edited by Katharine Dvorak

Hypermedia Style Example: Generic Data

```
{
   “data”: [
    {“name”: “student-name”, “value”: “Ronnie”},
    {“name”: “enrollment-year”, “value”: “2014”}
    ],
   “_address_details”: “/student/ronnie/address”
}
```

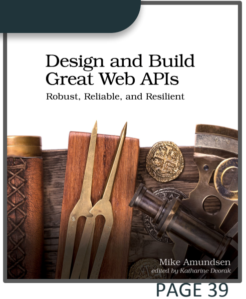


Design and Build
Great Web APIs
Robust, Reliable, and Resilient

Mike Amundsen
edited by Katharine Dvorak

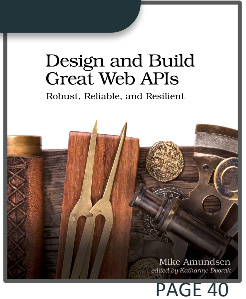
Hypermedia Style Example: Generic Links

```
{
   “data”: [
    {“name”: “student-name”, “value”: “Ronnie”},
    {“name”: “enrollment-year”, “value”: “2014”}
    ],
   “_address_details”: “/student/ronnie/address”
}
```



Hypermedia Style Example: Generic Links

```
{
   “data”: [
    {“name”: “student-name”, “value”: “Ronnie”},
    {“name”: “enrollment-year”, “value”: “2014”}
    ],
   “_links”: [
      {“rel”: “address”, “href”: “/student/ronnie/address”}
   ]
}
```

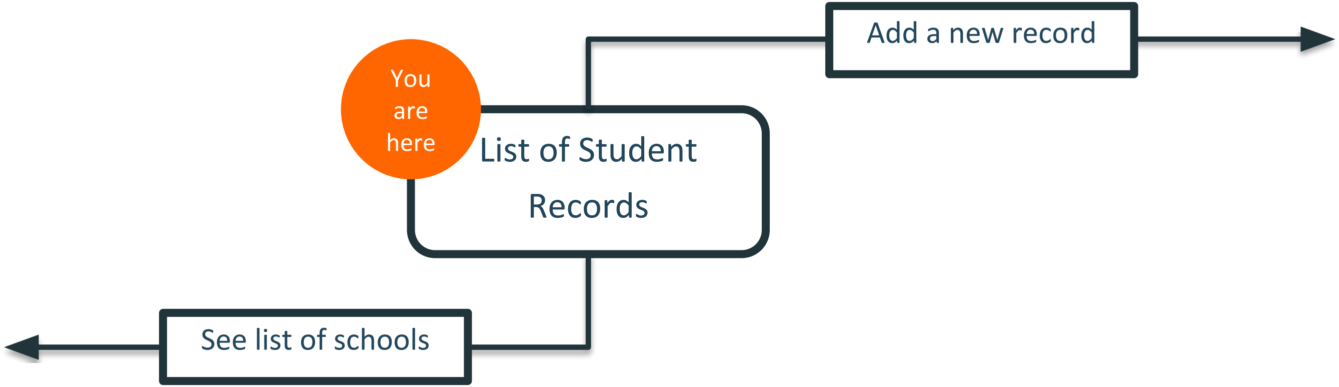


Design and Build
Great Web APIs
Robust, Reliable, and Resilient

Mike Amundsen
edited by Katharine Dvorak

Hypermedia Style: State Machine

- Each message represents the *current* state of the application
- Links tell the client what it can do *next*
- The client *changes* application state by following links

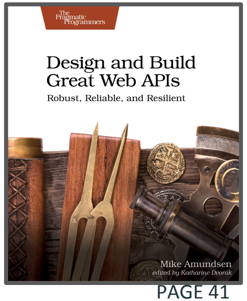


You are here

List of Student Records

Add a new record

See list of schools



The Pragmatic Programmers

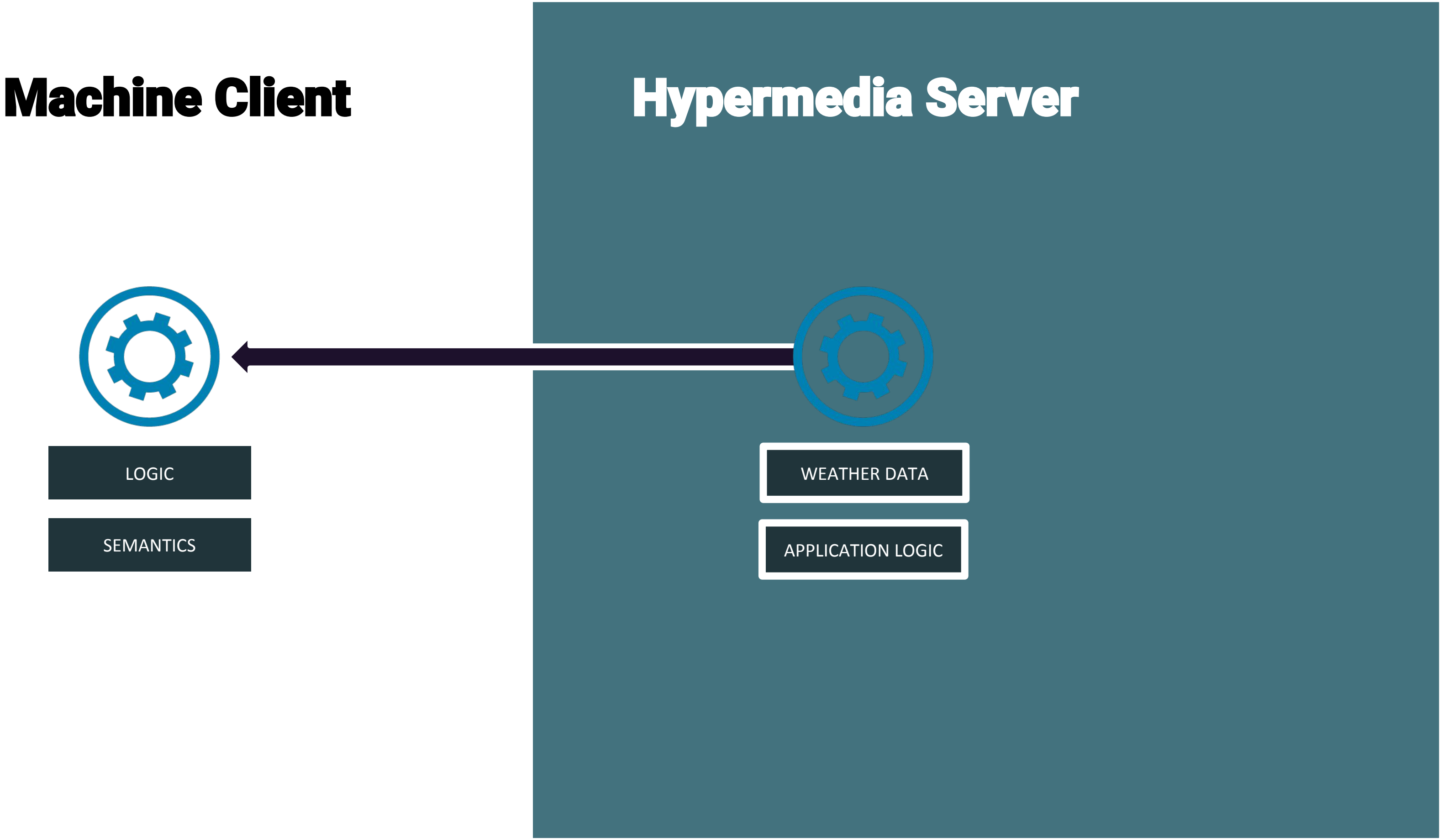
Design and Build Great Web APIs

Robust, Reliable, and Resilient

Mike Amundsen

edited by Katharine Dvorak

Hypermedia Style: Server to Server



Machine Client

LOGIC

SEMANTICS

Hypermedia Server

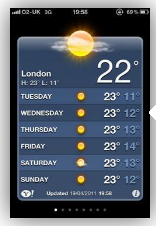
WEATHER DATA

APPLICATION LOGIC

Hypermedia Style: Mobile Client

Human Facing Client

Hypermedia Server



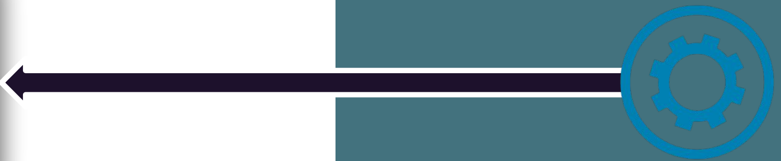
O2-UK 3G 19:58 69 %

London 22°

H: 23° L: 11°

TUESDAY	23°	11°
WEDNESDAY	23°	12°
THURSDAY	23°	13°
FRIDAY	23°	14°
SATURDAY	23°	13°
SUNDAY	23°	12°

Updated 19/04/2011 19:58



WEATHER DATA

APPLICATION LOGIC

INPUT HANDLING

LAYOUT

IMAGES

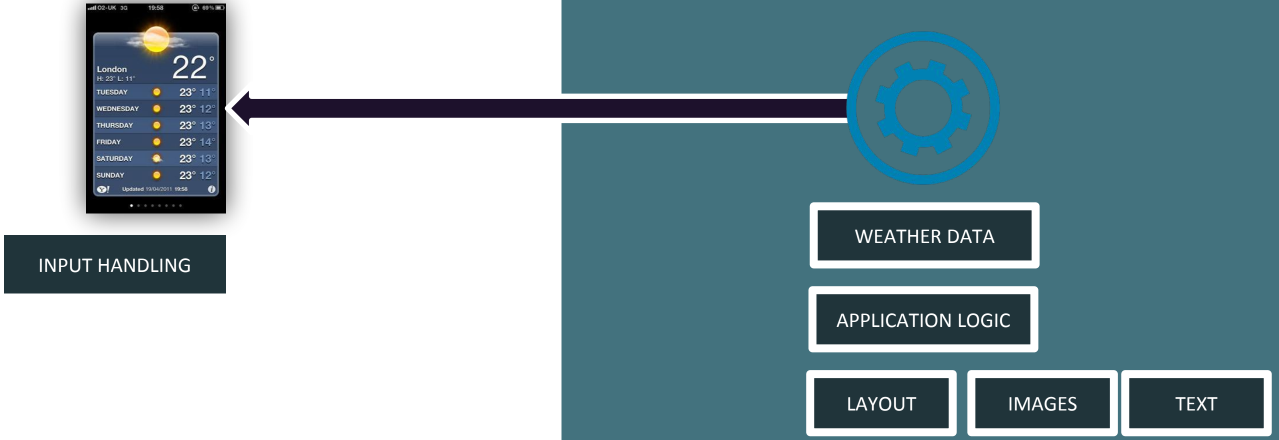
TEXT

SEMANTICS

Hypermedia Style: “Browser” Client

Human Facing Client

Hypermedia Server



O2-UK 3G 19:58 69 %

London
H: 23° L: 11°
22°

TUESDAY	23°	11°
WEDNESDAY	23°	12°
THURSDAY	23°	13°
FRIDAY	23°	14°
SATURDAY	23°	13°
SUNDAY	23°	12°

Updated 19/04/2011 19:58

INPUT HANDLING

WEATHER DATA

APPLICATION LOGIC

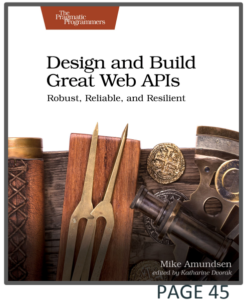
LAYOUT

IMAGES

TEXT

Hypermedia Style: Benefits

- Applications are easier to change (less client code changes required)
- Favours long running and large scale applications
- Takes advantage of the WWW architecture


The Pragmatic Programmers

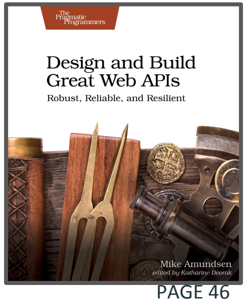
Design and Build Great Web APIs

Robust, Reliable, and Resilient

Mike Amundsen
edited by Katharine Dvorak

Hypermedia Style: Limitations

- Short-term benefits are limited – big up front cost today
- Assumed “esoteric”, “too hard”, etc.
- Clients are non-trivial to build
- Messages are verbose – not optimized for message size



The Pragmatic Programmers

Design and Build Great Web APIs

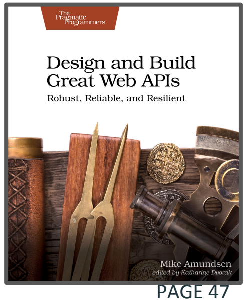
Robust, Reliable, and Resilient

Mike Amundsen

edited by Katharine Dvorak

Query Style

?



The Pragmatic Programmers

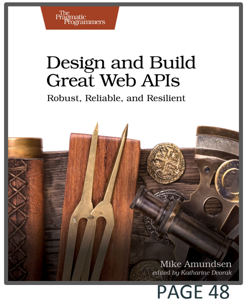
Design and Build Great Web APIs

Robust, Reliable, and Resilient

Mike Amundsen
edited by Katharine Dvorak

Query Style: Overview

- Interaction optimized and standardized for *querying* functions
- Learn the location of API and run queries against the data
- Suitable for any transport protocol that supports client-server interactions (usually supports HTTP)
- Examples:
  - graphQL
  - sparQL
  - ODBC
  - ql.io



The Pragmatic Programmers

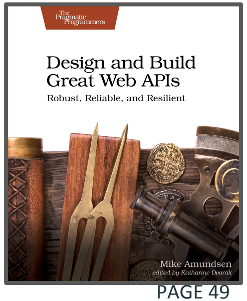
Design and Build Great Web APIs

Robust, Reliable, and Resilient

Mike Amundsen
edited by Katharine Dvorak

Query Style: Characteristics

- Treats the API as a *data source*
- A *Query Language* is defined and standardized (not just generic support)
- Focus is on reading and writing data


The Pragmatic Programmers

Design and Build Great Web APIs

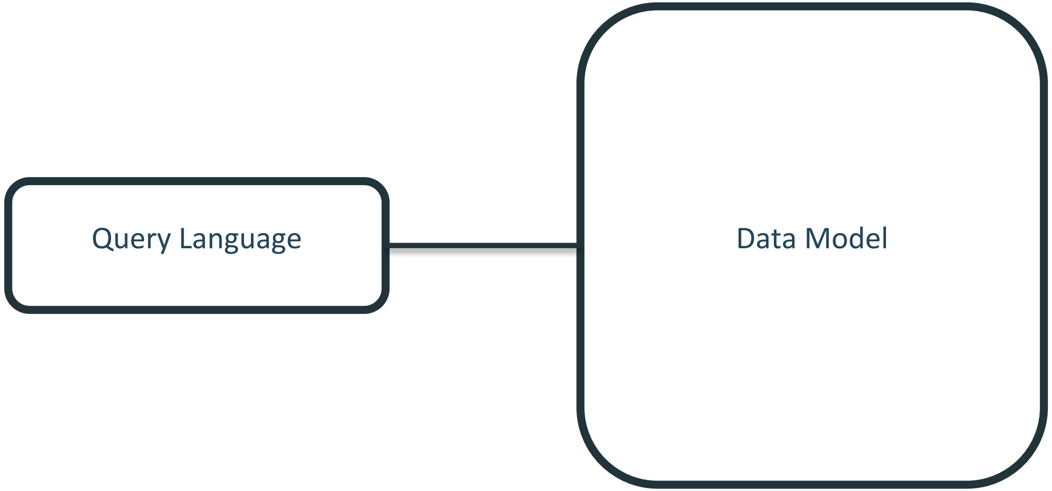
Robust, Reliable, and Resilient

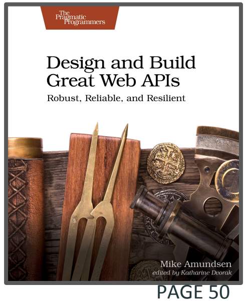
Mike Amundsen

edited by Katharine Dvorak

Query Style: Primary Constraint

- Interactions and data model are constrained by the *query language*





The Pragmatic Programmers

Design and Build Great Web APIs

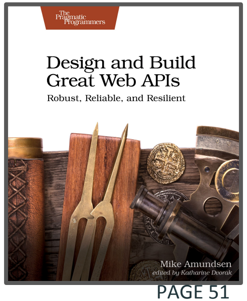
Robust, Reliable, and Resilient

Mike Amundsen
edited by Katharine Dvorak

Query Style Example: GraphQL

POST `http://myapi.com/graphql`

```
{
 “query”: “
    {
      student {
        name
        age
      }
    }
  ”
}
```


The Pragmatic Programmers
Design and Build Great Web APIs
Robust, Reliable, and Resilient
Mike Amundsen
edited by Katharine Dvorak

Query Style: Protocol Stack

?

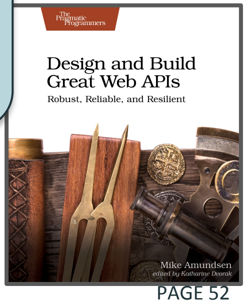
Query Interface Language

X, Y, etc…

Data Model

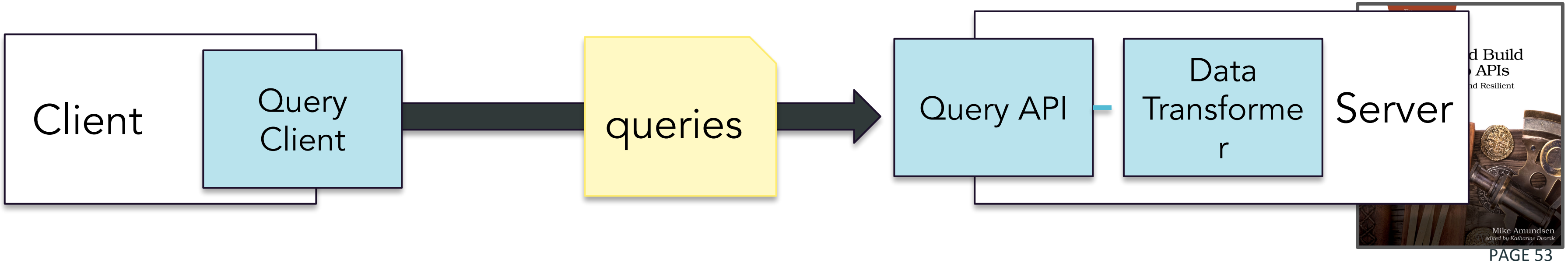
Relational, CRUD, etc…

Client-Server Transport Protocol


The Pragmatic Programmers
Design and Build Great Web APIs
Robust, Reliable, and Resilient
Mike Amundsen
edited by Katharine Dvorak

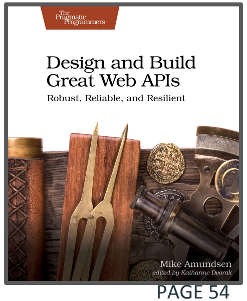
Query Style: Common Implementation

- Implement data transformation components on the server to support the standardized *Data Model*
- Bind a *Query Language* API to the data transformation
- Client implements a client query library
- Client uses query client to work with data (in RPC fashion)



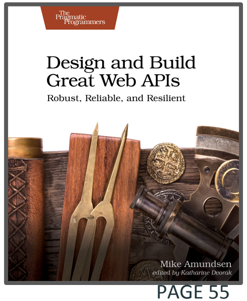
Query Style: Benefits

- All clients and servers that support query standard can interact easily
- Standardizing on language makes tooling possible:
  - Data inspection tools
  - Frameworks and libraries for clients and servers
- Ideal choice for data-centric apps (e.g. mobile apps)


The Pragmatic Programmers
Design and Build Great Web APIs
Robust, Reliable, and Resilient
Mike Amundsen
edited by Katharine Dvorak

Query Style: Limitations

- Features are limited to query language functions
  - How do you mutate data?
  - What is the performance profile?
  - How can you perform non-query operations?
- Difficult to use if data model doesn't match the client's needs
- Changes to data model may require client code changes



The Pragmatic Programmers

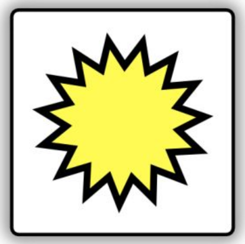
Design and Build Great Web APIs

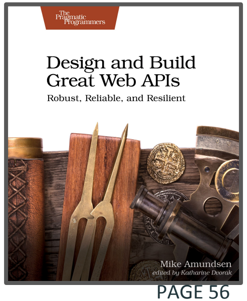
Robust, Reliable, and Resilient

Mike Amundsen

edited by Katharine Dvorak

Event-Driven Style





The Pragmatic Programmers

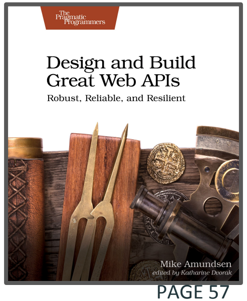
Design and Build Great Web APIs

Robust, Reliable, and Resilient

Mike Amundsen
edited by Katharine Dvorak

Event-Driven Style: Overview

- Fire and receive “events”
- Asynchronous interactions (one-way)
- Sender/Receiver instead of Client/Server
- Examples:
  - Message Oriented Middleware (e.g. MQ)
  - Reactive Programming
  - Event Streaming



The Pragmatic Programmers

Design and Build Great Web APIs

Robust, Reliable, and Resilient

Mike Amundsen

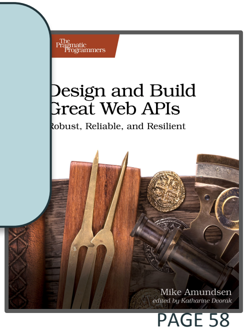
edited by Katharine Dvorak

Event-Driven Style: Protocol Stack

Event
Custom Design

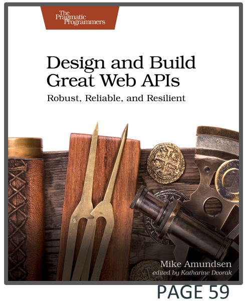
Transport Protocol
HTTP, MQ, TCP/IP, etc…

Event Infrastructure


The Pragmatic Programmers
Design and Build Great Web APIs
Robust, Reliable, and Resilient
Mike Amundsen
edited by Katharine Dvorak

Event-Driven Style: Constraints and Characteristics

- Senders have no knowledge of receivers (e.g. write to queue or publish to topic)
- Event receivers “react” to events
- Events represent change to a state
- Events can have multiple receivers (subscribers)



The Pragmatic Programmers

Design and Build Great Web APIs

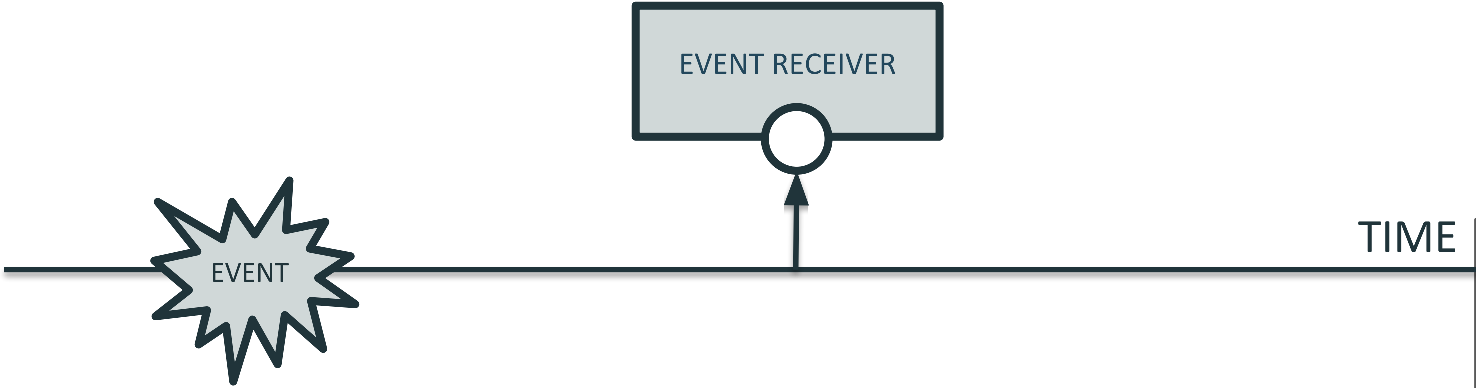
Robust, Reliable, and Resilient

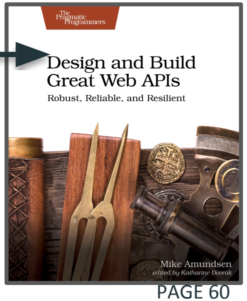
Mike Amundsen

edited by Katharine Dvorak

Event-Driven Style: Primary Constraint

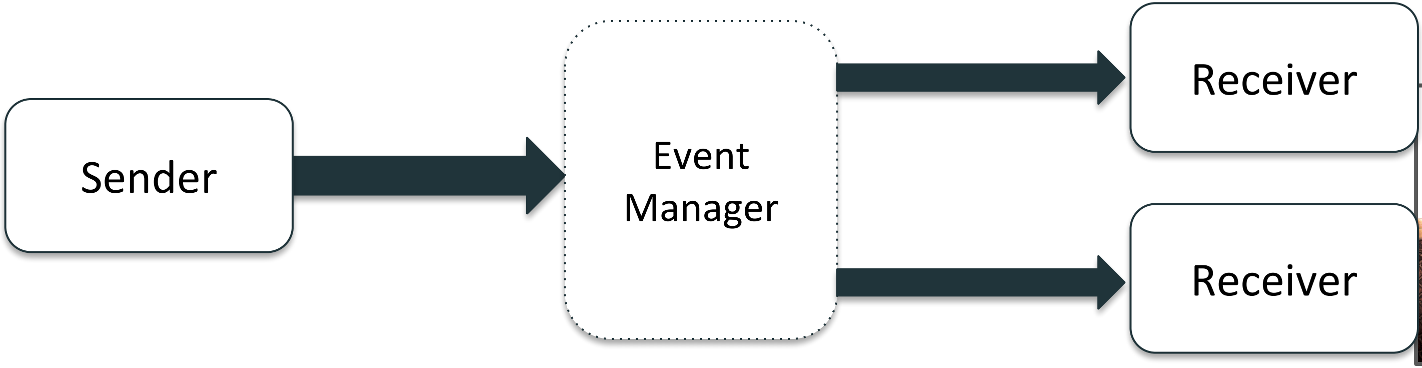
- Events occur in the past
- You can't change the past!

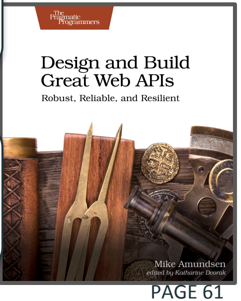



The Pragmatic Programmers
Design and Build Great Web APIs
Robust, Reliable, and Resilient
Mike Amundsen
edited by Katharine Dvorak

Event-Driven Style: Common Implementation

- Identify state change events
- Register event listener(s)
- Sender sends notification when state changes
- Event manager transmits notifications
- Receiver(s) handle events




The Pragmatic Programmers

Design and Build Great Web APIs

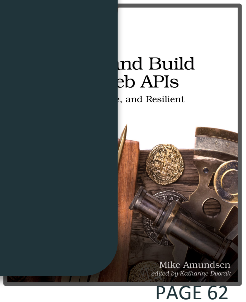
Robust, Reliable, and Resilient

Mike Amundsen
edited by Katharine Dvorak

Event-Driven Style: Notification Design

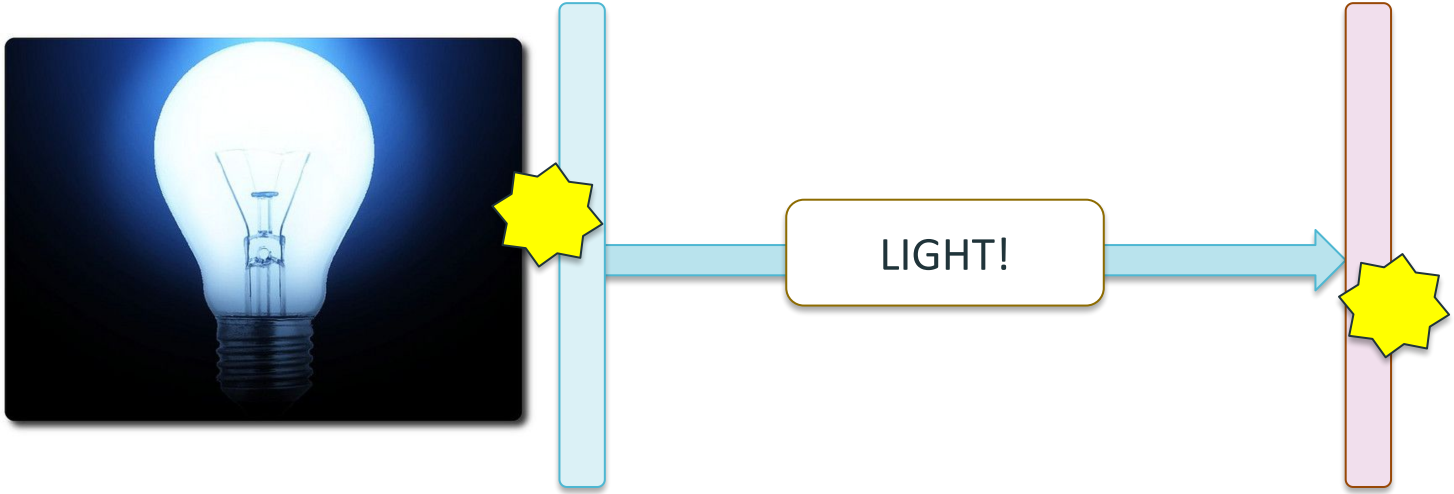
- Event data may include:
  - Target or source of event
  - Type of event
  - Event details
  - Contextual information

```
{
  “event” : {
  “name”: “RecordAdded”,
  “source”: “StudentRecords”,
  “location” : “/students/1883”,
  “editing” : “true”
  }
}
```



Event-Driven Style: Internet of Things

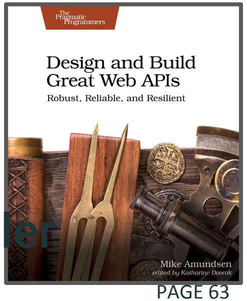
- Increased use of event driven style in IoT
- The real world is based on events
- Pervasive technology is primarily event based



LIGHT!

Sensor

Event Handler



The Pragmatic Programmers

Design and Build Great Web APIs

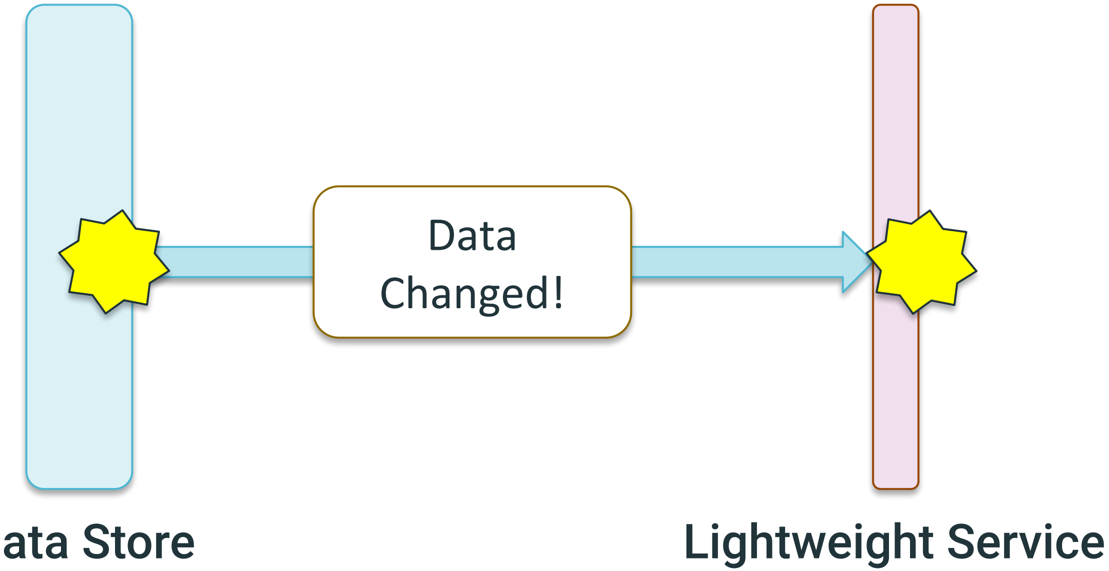
Robust, Reliable, and Resilient

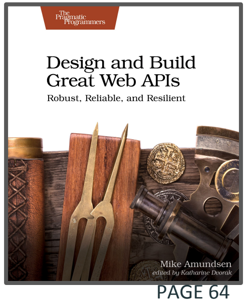
Mike Amundsen

edited by Katharine Dvorak

Event-Driven Style: Microservices

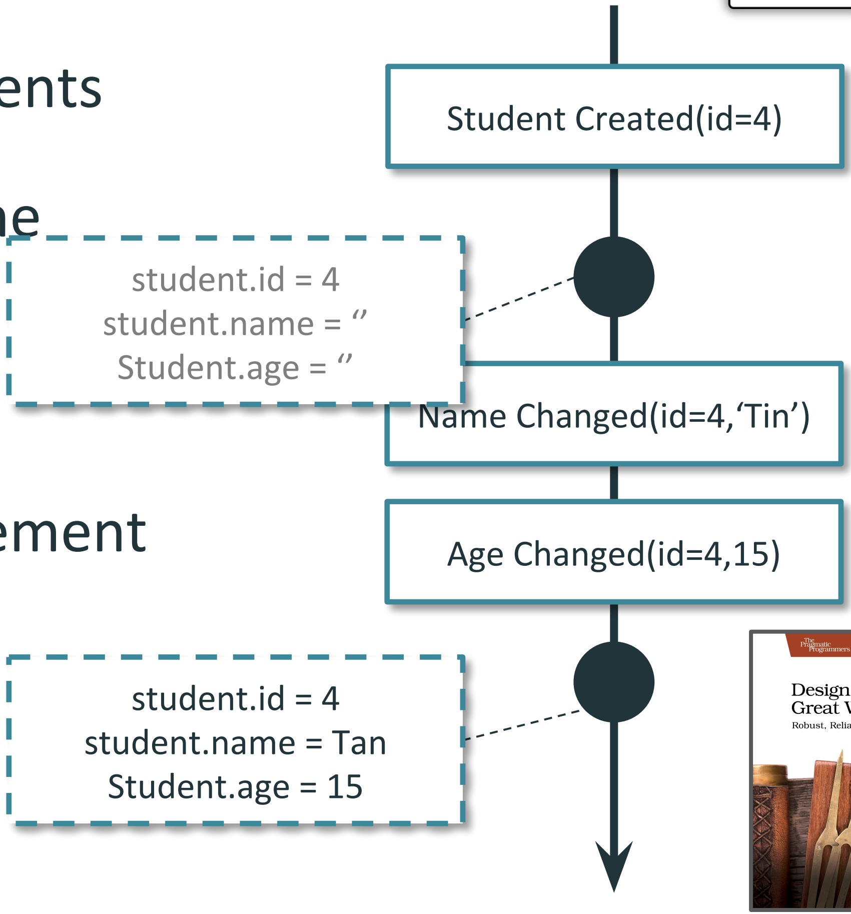
- Inter-Service communication (behind firewall)
- Cache freshness
- Data synchronization / eventual consistency




The Pragmatic Programmers
Design and Build Great Web APIs
Robust, Reliable, and Resilient
Mike Amundsen
edited by Katharine Dvorak

Event-Driven Style: Event Sourcing / Event Store

- Persist data state change events
- The history of all events is the “present” state of data
- Makes *distributed data* architectures easier to implement



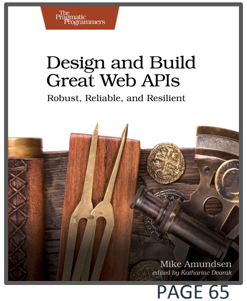
Student Created(id=4)

student.id = 4
student.name = ‘’
Student.age = ‘’

Name Changed(id=4,‘Tin’)

Age Changed(id=4,15)

student.id = 4
student.name = Tan
Student.age = 15



The Pragmatic Programmers

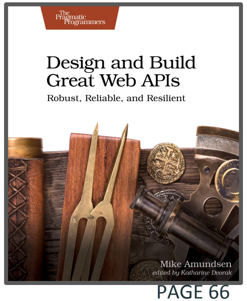
Design and Build Great Web APIs

Robust, Reliable, and Resilient

Mike Amundsen
edited by Katharine Dvorak

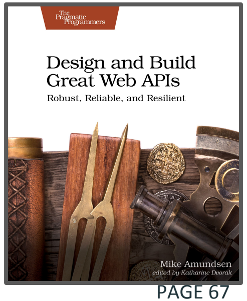
Event-Driven Style: Benefits

- Components and data can be *de-coupled* and *de-centralized*
- Ideal for transmitting many changes continuously over time (e.g. streaming)
- De-centralized messaging system offers added reliability
- “Reactive” event style offers improved perceived UI performance


The Pragmatic Programmers
Design and Build Great Web APIs
Robust, Reliable, and Resilient
Mike Amundsen
edited by Katharine Dvorak

Event-Driven Style: Limitations

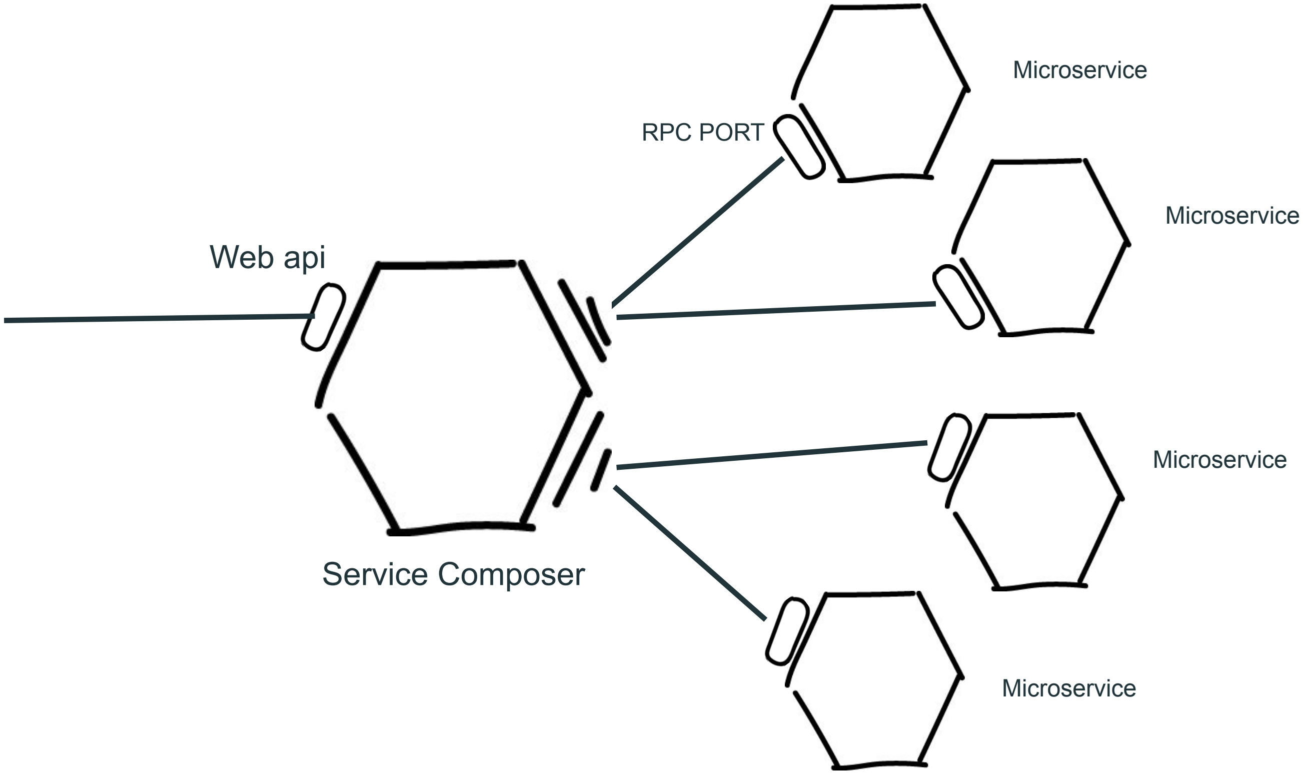
- Can only record what has *already* happened (e.g. how do you perform validation?)
- Increases the complexity of the architecture and infrastructure
- Performance, scalability and reliability limited by event infrastructure

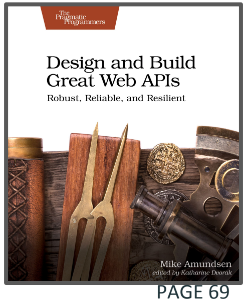

The Pragmatic Programmers
Design and Build Great Web APIs
Robust, Reliable, and Resilient
Mike Amundsen
edited by Katharine Dvorak

Styles as Metaphors

Style	Metaphor
Tunnel Style	Procedural programming
URI Style	Data Access Objects
Hypermedia	Browsing the web
Query	Database Query Languages
Event Based	Event based programming (e.g. GUI)

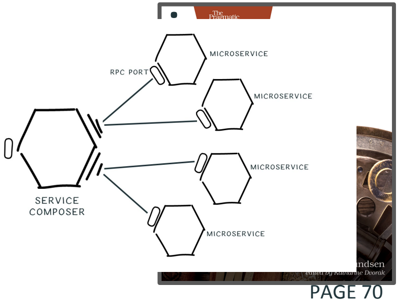
A Tunnel Style Example: Microservices Composition



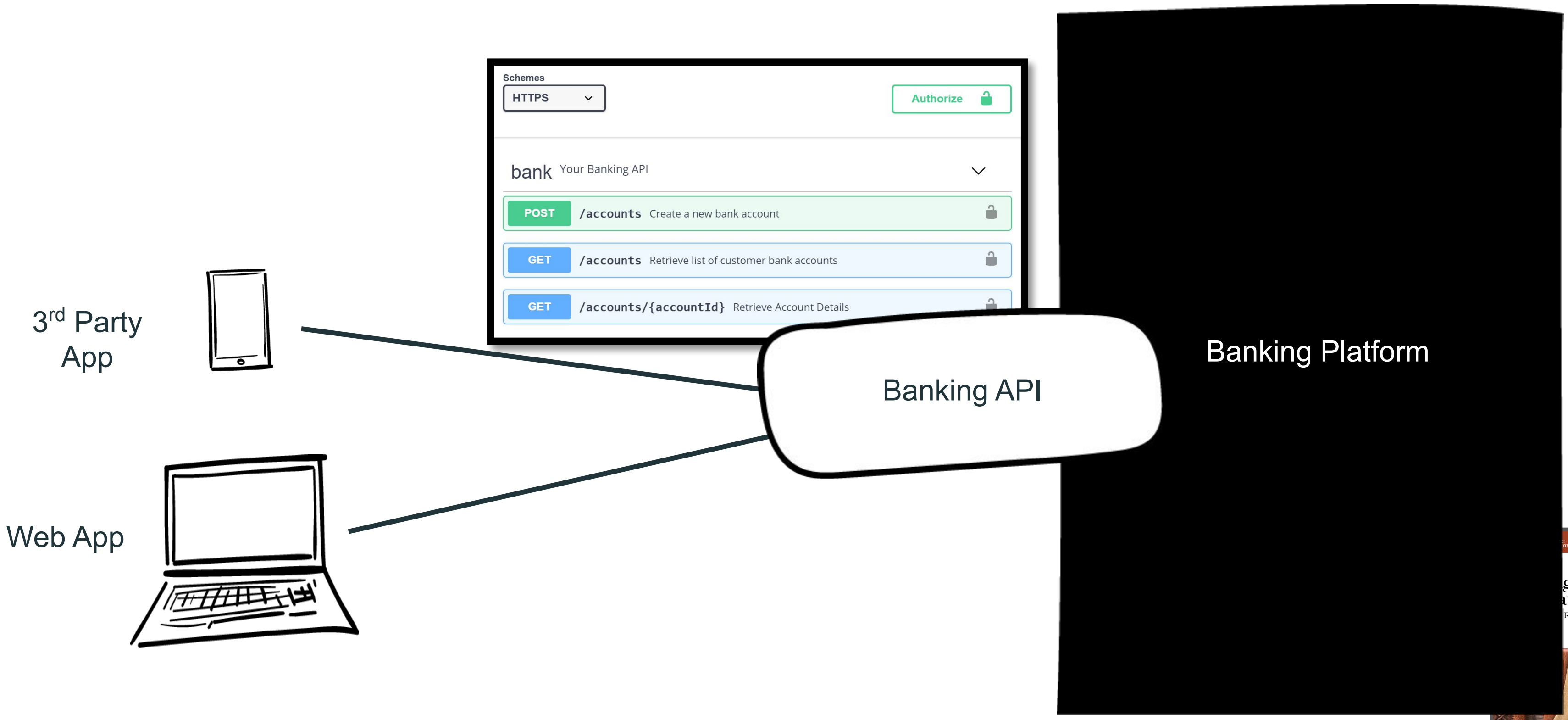

The Pragmatic Programmers
Design and Build Great Web APIs
Robust, Reliable, and Resilient
Mike Amundsen
edited by Katharine Dvorak

A Tunnel Style Example: Microservices Composition

- Publishing new interfaces is cheap and easy for service teams
- Service composer team is only client and prepared to rebuild their component after any service changes (warning: potential bottleneck)
- External component is shielded from change
- RPC implementation can be chosen for optimized speed (e.g. GRPC/Thrift)



URI Style Example: Public Banking API



Schemes

HTTPS

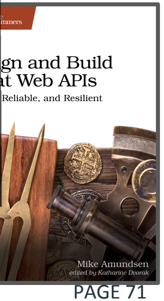
Authorize

bank Your Banking API

POST /accounts Create a new bank account

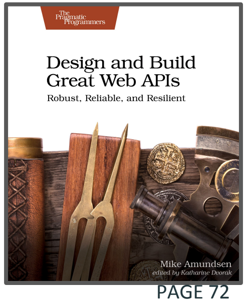
GET /accounts Retrieve list of customer bank accounts

GET /accounts/{accountId} Retrieve Account Details

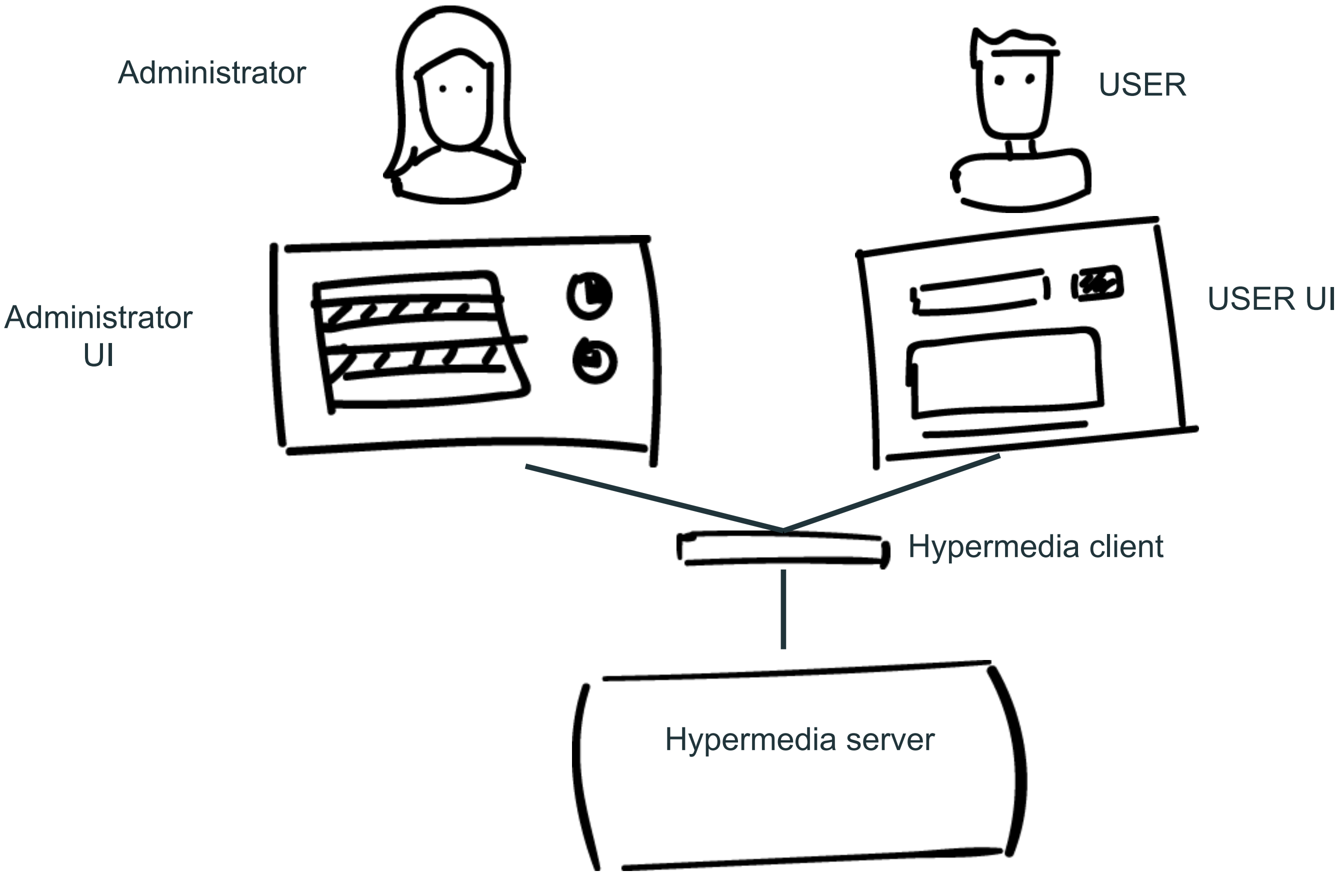


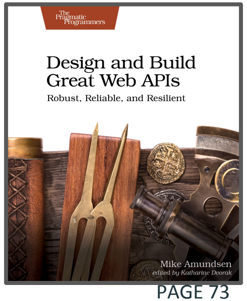
URI Style Example: Public Banking API

- Developers outside the bank will find the URI style *familiar*
- Many of the interactions are well suited for the CRUD pattern
  - View transactions
  - View balance
  - Create payment
- Little commercial motivation to make it easy to change API providers


The Pragmatic Programmers
Design and Build Great Web APIs
Robust, Reliable, and Resilient
Mike Amundsen
edited by Katharine Dvorak

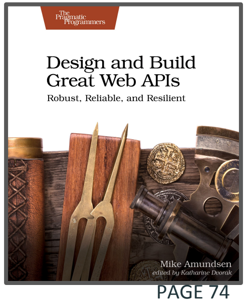
Hypermedia Style Example: UX Fragmentation



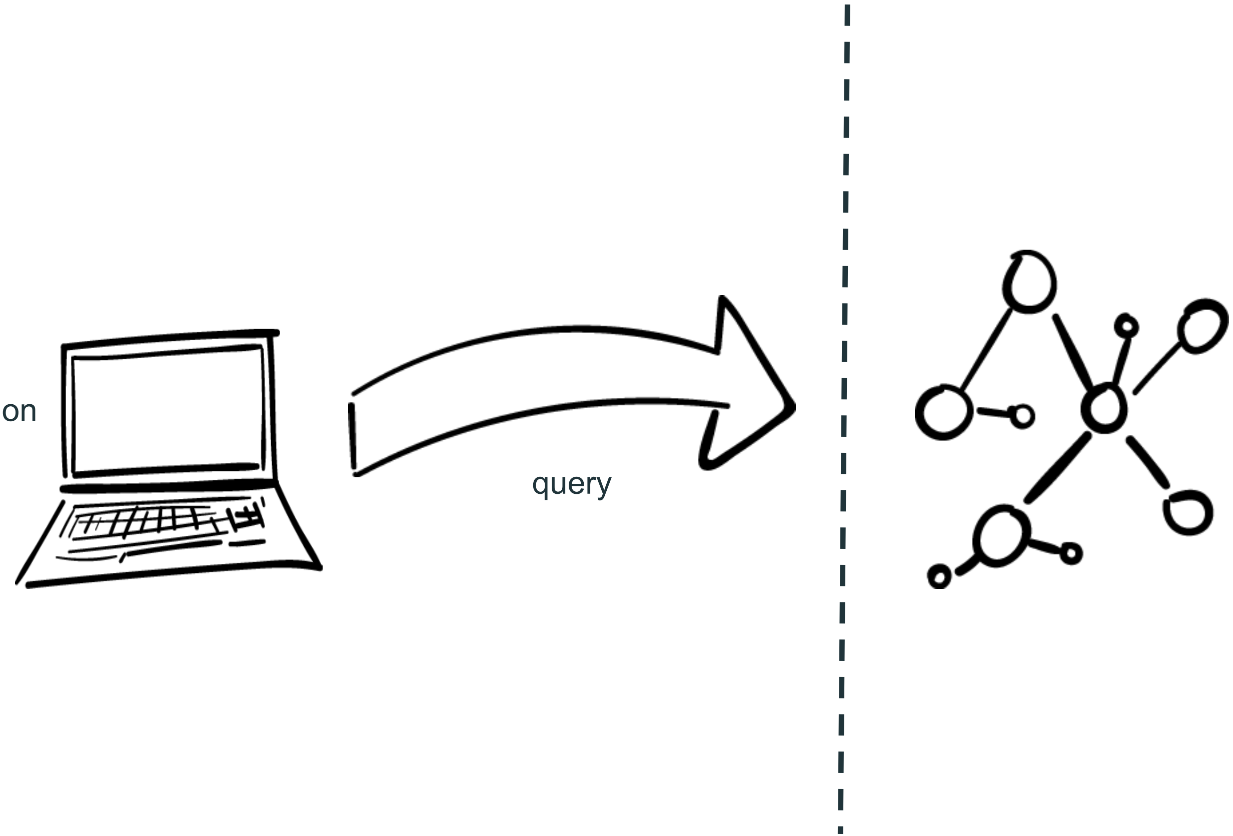

The Pragmatic Programmers
Design and Build Great Web APIs
Robust, Reliable, and Resilient
Mike Amundsen
edited by Katharine Dvorak

Hypermedia Style Example: UX Fragmentation

- Manage and deploy a single client application
- Change UI and workflow without re-deploying client
- Works best when client development owned by organization
- Works best when cheap UX generation is a market differentiator


The Pragmatic Programmers
Design and Build Great Web APIs
Robust, Reliable, and Resilient
Mike Amundsen
edited by Katharine Dvorak

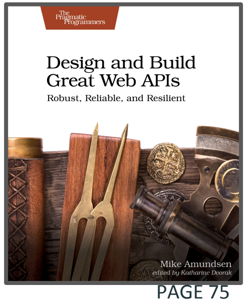
Query Style Example: Social Graph Data



Client application

query

Graph data model



The Pragmatic Programmers

Design and Build Great Web APIs

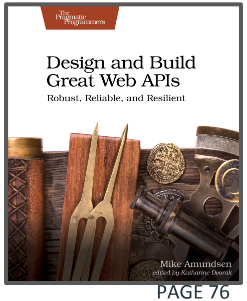
Robust, Reliable, and Resilient

Mike Amundsen

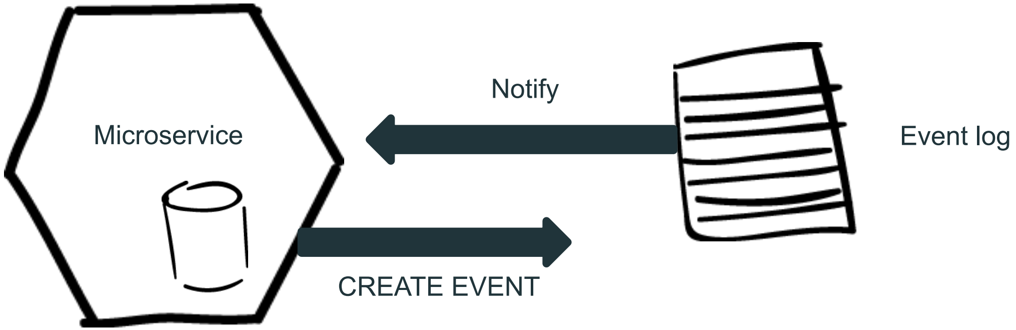
edited by Katharine Dvorak

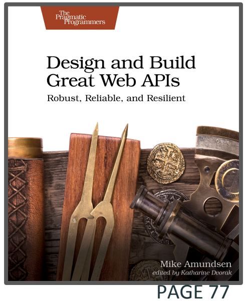
Query Style Example: Social Graph Data

- Query language optimized for data type
- Client development is easier
- Backend is optimized for fast reads and complex queries


The Pragmatic Programmers
Design and Build Great Web APIs
Robust, Reliable, and Resilient
Mike Amundsen
edited by Katharine Dvorak

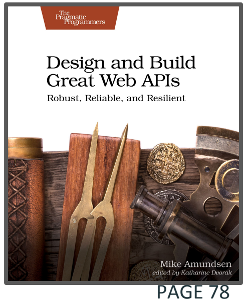
Event Driven Example: Decentralized Data




The Pragmatic Programmers
Design and Build Great Web APIs
Robust, Reliable, and Resilient
Mike Amundsen
edited by Katharine Dvorak

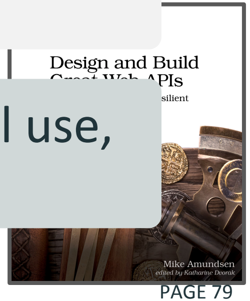
Event Driven Example: Decentralized Data

- Makes it easier to manage and deploy services independently
- Distance between components is small (intranet, not internet)
- Data can be “stale”
- Libraries/SDK/Sidecars are provided to reduce dev. cost


The Pragmatic Programmers
Design and Build Great Web APIs
Robust, Reliable, and Resilient
Mike Amundsen
edited by Katharine Dvorak

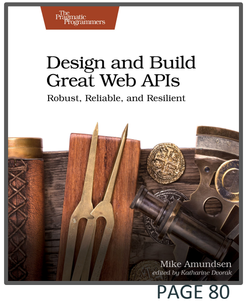
General Advice for Styles

Tunnel Style	▪ Typed interaction, gRPC gaining popularity for internal use
URI Style	▪ The default style for web based APIs
Hypermedia Style	▪ Most scalable and change-friendly, but least conventional
Query Style	▪ Gaining popularity, ideal for internal, data-centric apps
Event Driven Style	▪ Loose coupled, centralized – good for internal use, not a good choice for public APIs



Use Your Head

- Implementations may borrow from multiple styles
- Your system will probably contain more than one API style and needs will change over time
- Start with a style that makes sense for your situation – not necessarily the one you are “supposed” to use

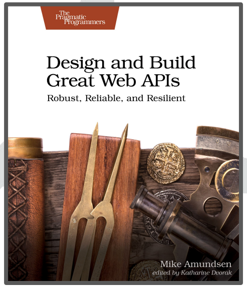

The Pragmatic Programmers
Design and Build Great Web APIs
Robust, Reliable, and Resilient
Mike Amundsen
edited by Katharine Dvorak

Five API Styles

Mike Amundsen

@mamund

training.amundsen.com



The Pragmatic Programmers

Design and Build Great Web APIs

Robust, Reliable, and Resilient

Mike Amundsen

edited by Katharine Dvorak