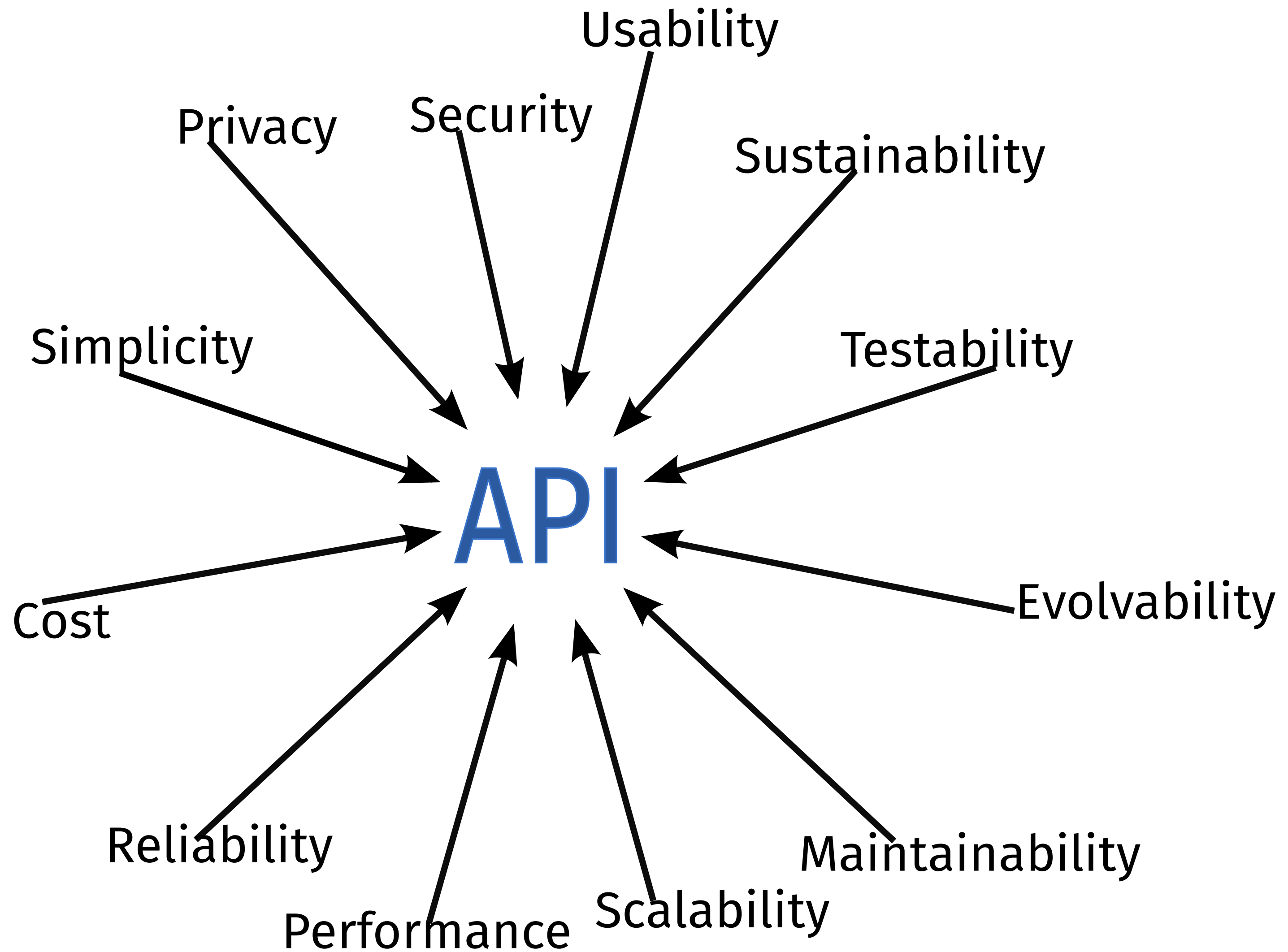


# Guiding Architectural Decision Making on Quality Aspects in Microservice APIs

Uwe Zdun, Mirko Stocker, Olaf Zimmermann,  
Cesare Pautasso, and Daniel Lübke

<http://www.pautasso.info/talks/2018/icsoc>

[@pautasso@scholar.social](mailto:pautasso@scholar.social)



# Research Questions

1  
2

What are established practices to **design** for, realize, communicate and maintain the **quality of a microservice API**?

How high is the **decision making uncertainty** in this design space?

How can we reduce this decision making uncertainty?

# Knowledge Sources

Amazon EC2 API, Amazon S3, AWS Lambda, Cloud Convert API, Confluence REST API, Facebook Graph API, File Transfer Service API, Finance Industry Web Service API, GitHub API v3, GitHub API v4, Google Calendar API, Google Compute Engine, JIRA Cloud API, LinkedIn API, Microsoft Azure, Microsoft Dynamics CRM, Microsoft Graph API, Open Weather Map, Optimizely, PayPal API, Quandl API, Salesforce API, Singlewire, Stripe API, SWIFT, Swiss Bank API, Swiss Federal Administration registry of companies web service API, Swiss Insurance API, TMForum REST API, Twitter API, YouTube Data API

Adidas API Spec, Amazon API Gateway, apistylebook.com, Basic Authentication, CHAP, EAP, EC SLA Guidelines, HTTP/1.1: Conditional Requests, JSON API Spec, Kerberos, LDAP, MuleSoft API Manager, OAuth, OpenID Connect 1.0, OWASP REST Security, Play2 Guard, REST API design book, RESTful Web Services Cookbook, RFC 7519, SAML, SLA Best Practices, SLA Whitepaper, Suggested REST Practices, TM Forum Applications Framework 3.0

# Some API Quality Design Decisions


1. Explicit Specification of Quality Objectives and Penalties
2. API Client Identification and Authentication
3. Metering and Charging for API Consumption
4. Avoid Unnecessary Data Transfer
5. Preventing API Clients from Excessive API Usage
6. Communicate Errors

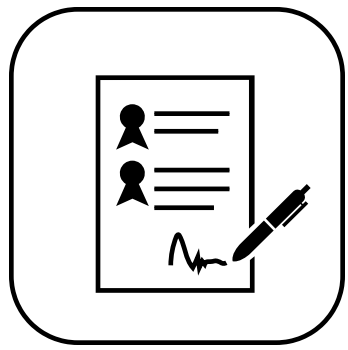
# Explicit Specification of Quality Objectives and Penalties

## Decision Criteria

- Consumer Attractiveness
- Provider cost-efficiency and business risks
- Government regulations
- Relevant quality attributes (Availability, Performance, Privacy)
- Business agility/vitality

## Options

- Introduce  Service Level Agreement:
  1. Internal use only
  2. External use
- Represent SLA with:
  1. Informal specification (natural language)
  2. Formal specification



# Service Level Agreement

How can an API client learn about the specific quality-of-service characteristics of an API and its operations? How can these characteristics and the consequences of not meeting them be defined in a measurable way?

As an API provider, define a structured Service Level Agreement (SLA) that is written in an assertive, unambiguous way: the SLA must identify the API operation that it pertains to and must contain at least one measurable Service Level Objective (SLO), such as performance or availability.




# API Client Identification and Authentication

## Decision Criteria

- Required security level
- Ease of use for clients
- User credential management
- Performance overhead

## Options

- Introduce Authentication Mechanism:
  1.  API Key
  2. API Key with Secret
  3. Authentication Protocol

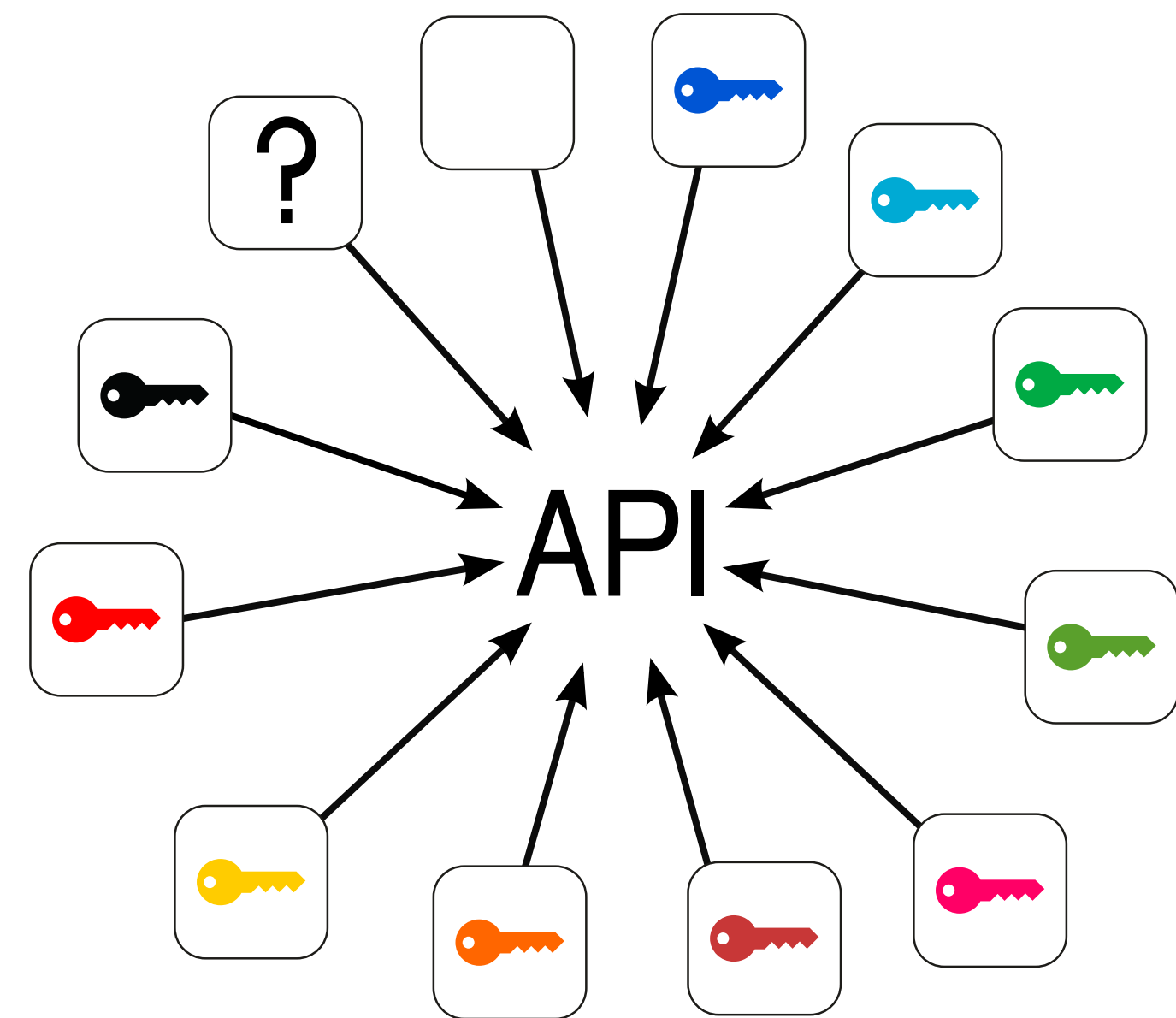




# API Key

How can an API endpoint identify and authenticate the client making a request?

As an API provider, assign each client a unique token – the API Key – that the client can present to the API endpoint for identification purposes.




# Metering and Charging for API Consumption

## Decision Criteria

- Pricing/Business Models
- Metering and Billing Accuracy
- Meter Granularity
- Security and Privacy

## Options

- Monitor Client Usage for Billing with  Rate Plan variants:
  1. Flat-rate subscription
  2. Pay per use
  3. Market-based Allocation
  4. Auction-style Allocation
  5. Freemium Model



# Rate Plan

How can the API provider meter API service consumption and charge for it?





Assign a Rate Plan for the API usage that is used to bill API clients, advertisers, or other stakeholders accordingly. Define and monitor metrics for measuring API usage, such as monitoring API usage on a per-operation level.

# Avoid Unnecessary Data Transfers

## Decision Criteria

- Client Information Needs
- Network bottlenecks
- Performance
- Security
- Development and Testing Complexity

## Options

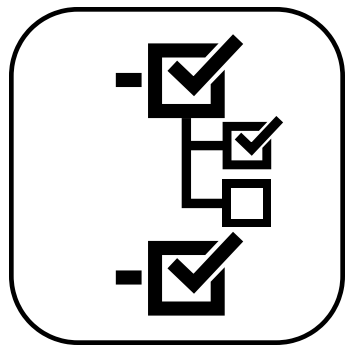
1.  Wish List
2.  Wish Template
3.  Conditional Request
4.  Request Bundle



# Wish List

How can an API client inform the API provider at runtime about the data it is interested in?

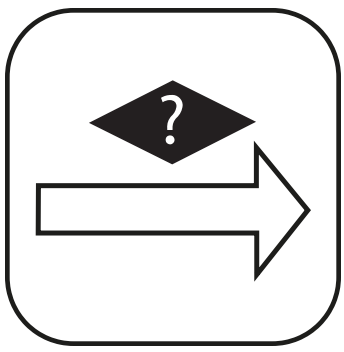
As an API client, provide a Wish List in the request that enumerates all desired data elements of the requested resource. As the API provider, deliver only those data elements in the response message that are enumerated in the Wish List.



# Wish Template

How can an API client inform the API provider at runtime about the complex data it is interested in?

Define one or more parameters in the request message that have the same structure as the desired parameter(s) in the response message.

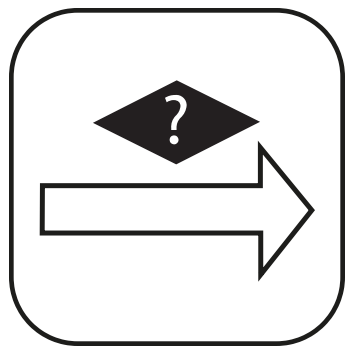


# Conditional Request

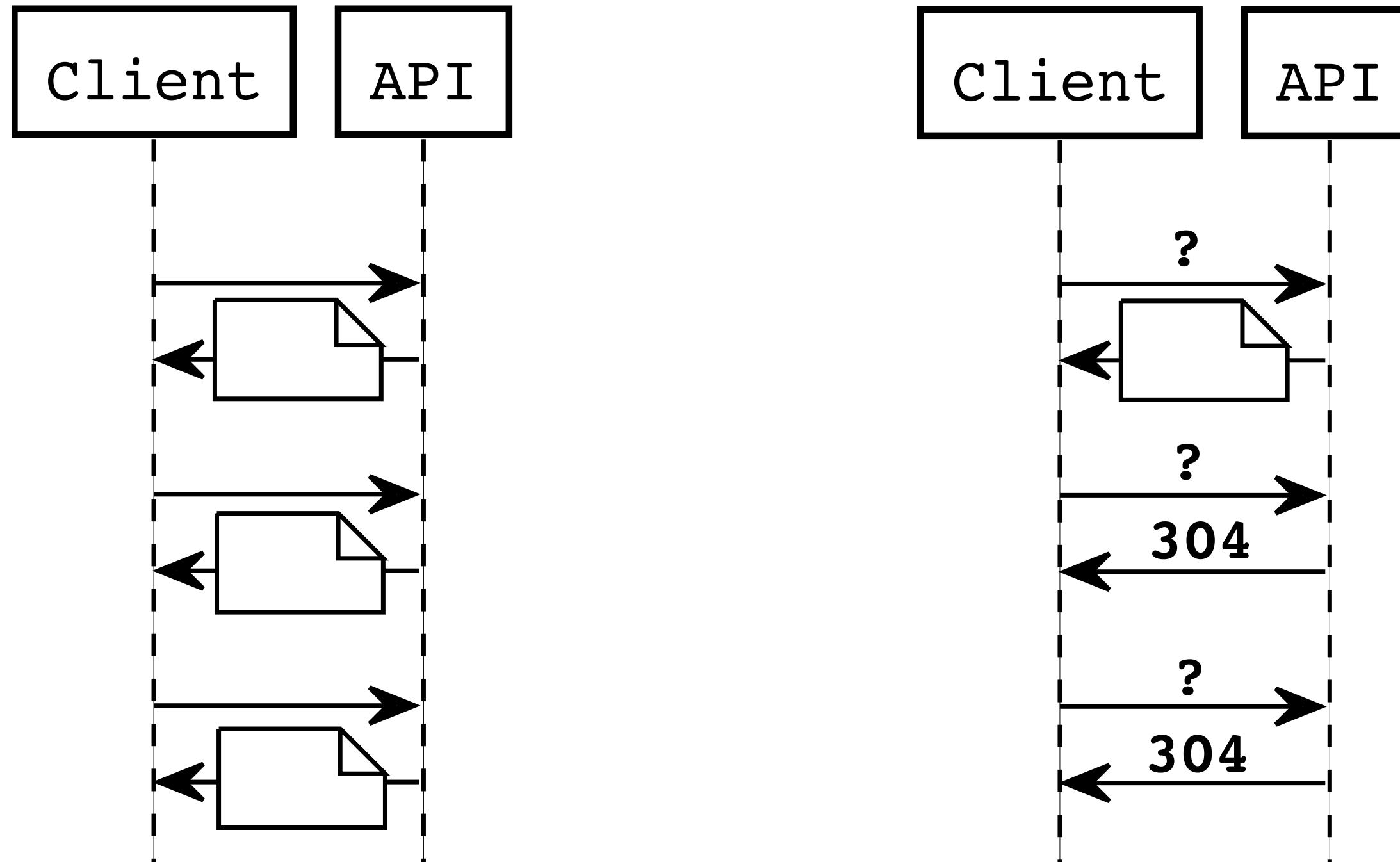
How can unnecessary server-side processing and bandwidth usage be avoided when frequently invoking API operations that return rarely changing data?

Support conditional requests by allowing client to add one or more metadata elements to the request message representations (or protocol headers) that requests are processed by the provider only if a condition is met. If the condition is not met, the provider does not reply with a full response, but returns a special status code instead.

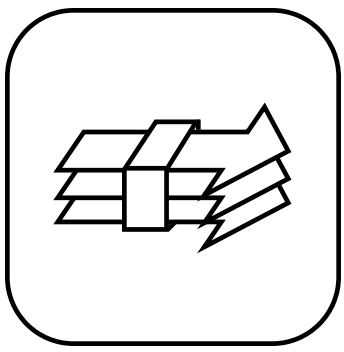




# Conditional Request



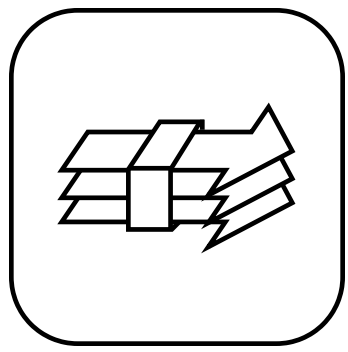
Assumption: Client can cache previous responses



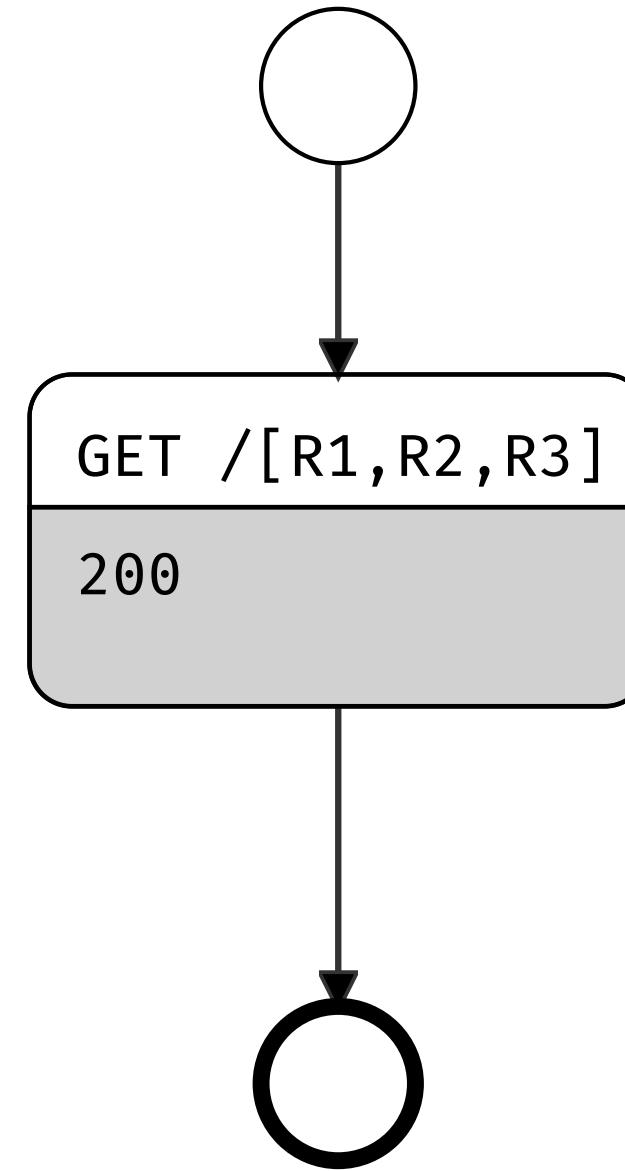
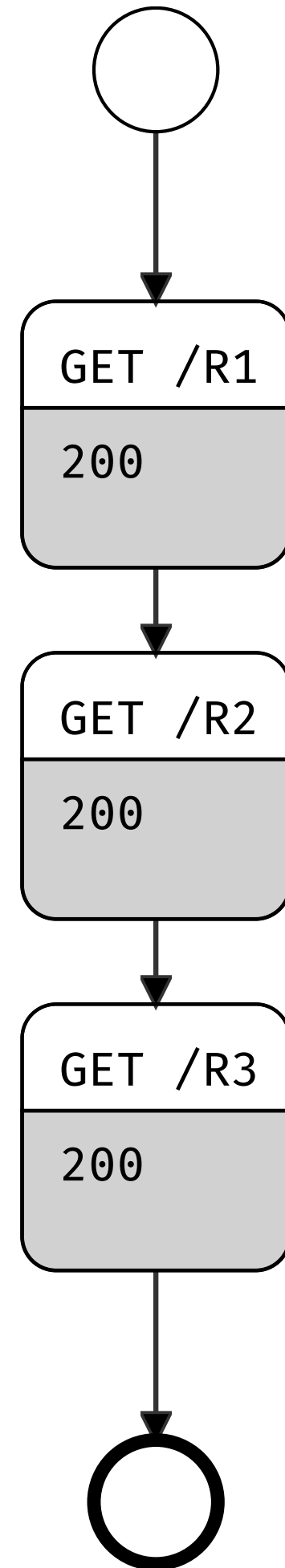
# Request Bundle

How can the number of requests and responses be reduced to save bandwidth and network capacity and to avoid unnecessary message processing?

Define a Request Bundle as a container that assembles multiple individual requests in a single request message. Add metadata such as number and identifiers of individual requests.



# Request Bundle





$$\text{Cost}([R_i]) < \text{SUM}(\text{Cost}(R_i))$$

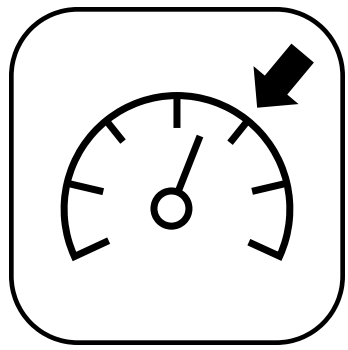
# Prevent Excessive API Usage

## Decision Criteria

- Guarantee Service Performance, Scalability, and Sustainability
- Make Clients Aware of Limits
- Impact/Severity of Abuse
- Negotiation Complexity

## Options

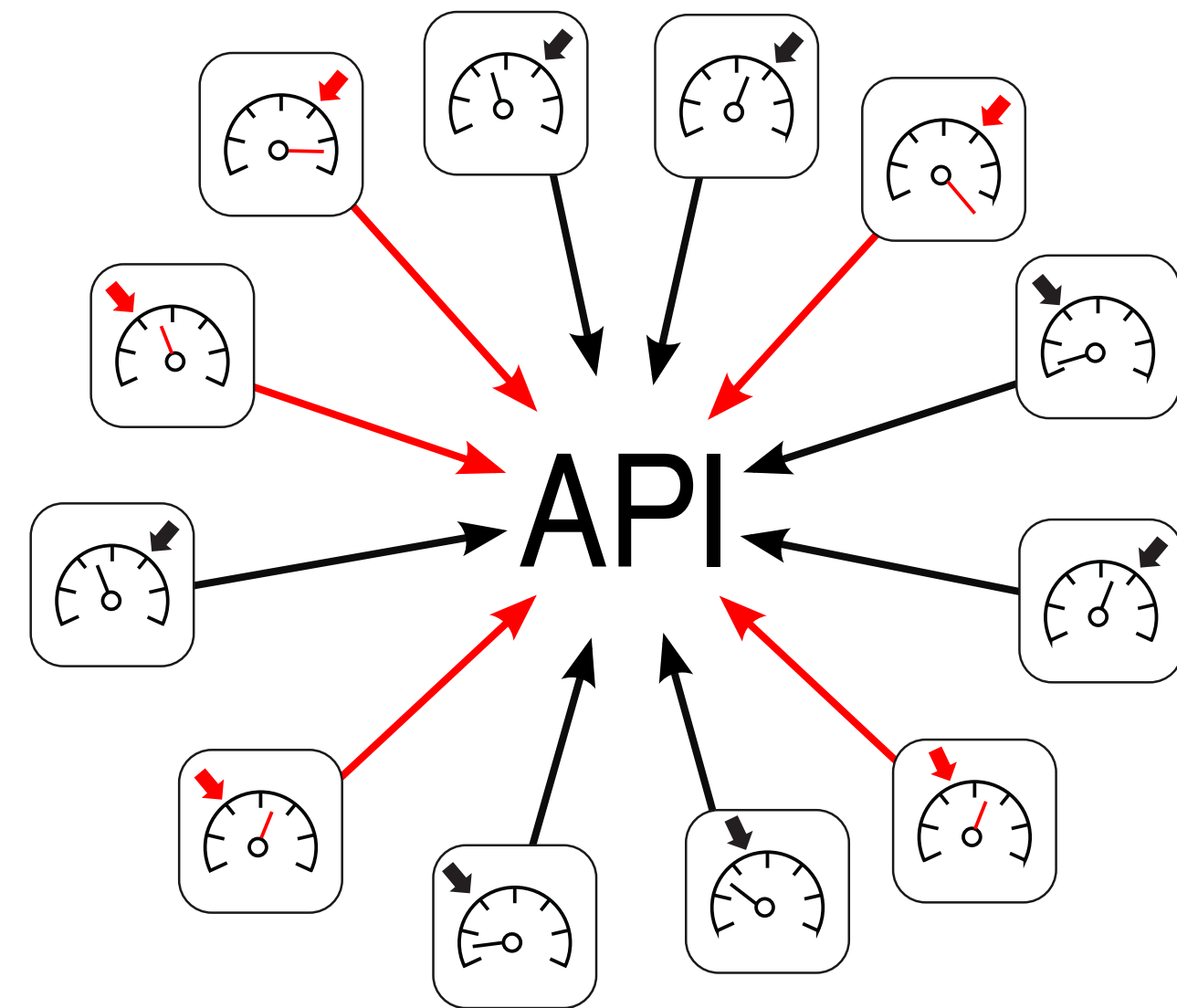
- Monitor Client Usage:
  1. for Billing with  Rate Plan
  2. within  Rate Limit
  3. or both



# Rate Limit

How can the API provider prevent API clients from excessive API usage?

Introduce and enforce a Rate Limit to safeguard against API clients that overuse the API.



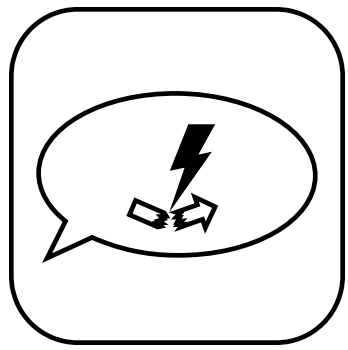
# Communicate Errors

## Decision Criteria

- Help Fixing Defects
- Increase:
  - Robustness, Reliability
  - Maintainability, Evolvability
- Expose Security Attack Vector
- Internationalization Required

## Options

- Protocol-level Error Codes
- Protocol-independent  
 Error Reporting



# Error Reporting

How can the provider inform the client about communication and processing faults without being restricted by technology specifics such as protocol headers and predefined status codes?

Reply with an error code (either an integer or a string constant) that indicates the fault in a machine-readable way. In addition, add a textual description of the error for the API client.



# API Quality Design Space

## 1. Explicit Specification of Quality Objectives and Penalties

Internal SLA

External SLA

Informal SLA

Formal SLA

## 2. API Client Identification and Authentication

API Key

API Key with Secret

Authentication Protocol

## 3. Metering and Charging for API Consumption

Flat-Rate Plan

Pay per use

Market-based  
Allocation

Auction-style  
Allocation

Freemium Model

## 4. Avoid Unnecessary Data Transfer

Wish List

Wish Template

Conditional Request

Request Bundle

## 5. Preventing API Clients from Excessive API Usage

Bill with Rate Plan

Rate Limits

## 6. Communicate Errors

Protocol-level Error Codes

Protocol-independent Error Reporting

# Uncertainty Estimation

**With Model**

**Without Model**

**Possible Decision Outcomes**

**#Allowed Option Combinations**

$2^{(\#Options)}$

**Decision Criteria Assessments**

**1 + #Undecided Criteria**

$\#Criteria \times \#Options$

# Uncertainty Reduction

API-Level Decisions: **5**, Operation-Level Decisions: **1**  
Decision Criteria: **47**, Decision Options: **40**

	With Model	Without Model
Possible Decision Outcomes	268'435'456	482'754'917'909
Decision Criteria Assessments	12	122

# Conclusions

- Organizing API Design Knowledge into Architectural Design Decision Models can lead to Significant Uncertainty Reduction
- 55 unique sources were analyzed to distill 6 architectural design decisions, with 40 decision options and 47 decision criteria
- Let us know how we can help you apply our decision guidance model to your API!

# References

- Uwe Zdun, Mirko Stocker, Olaf Zimmermann, Cesare Pautasso, Daniel Lübke, [Guiding Architectural Decision Making on Quality Aspects of Microservice APIs](#), Proc. of the 16th International Conference on Service-Oriented Computing (ICSOC 2018), Hangzhou, Zhejiang, China, November 2018
- Mirko Stocker, Olaf Zimmermann, Daniel Lübke, Uwe Zdun, Cesare Pautasso, [Interface Quality Patterns --- Crafting and Consuming Message-Based Remote APIs](#), Proc. of the 23rd European Conference on Pattern Languages of Programs (EuroPLOP), Kloster Irsee, Germany, July 2018