API Evolution for Microservices

Holger Knoche, Wilhelm Hasselbring

Kiel University, Software Engineering Group

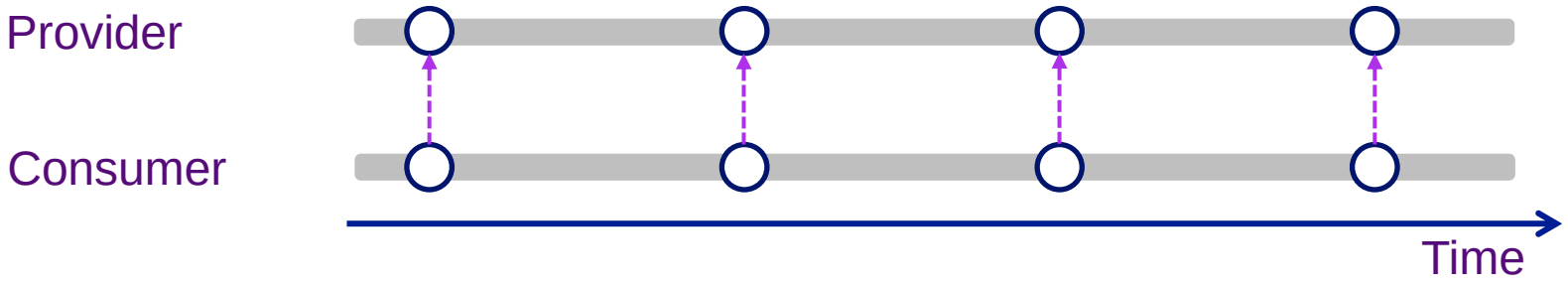
September 13, 2022 @ AK MSDO

ARBEITSKREIS
MSDO

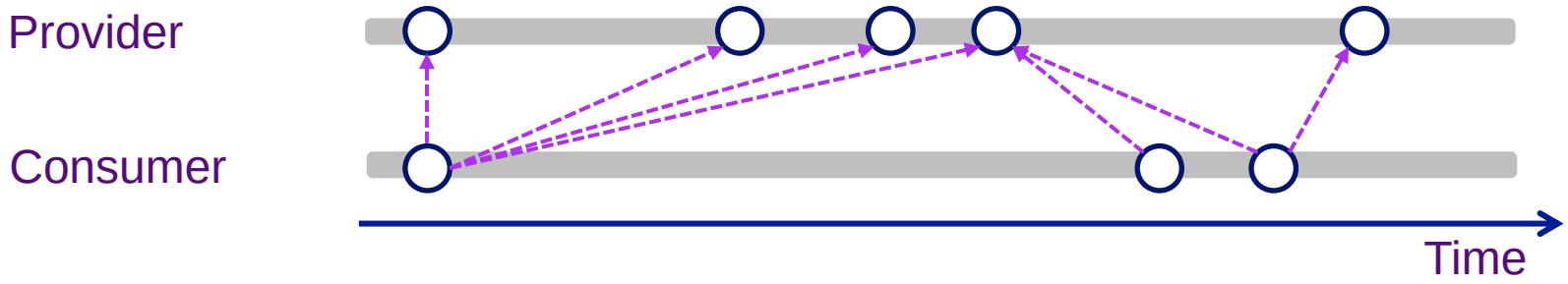
C A U
Christian-Albrechts-Universität zu Kiel

Key Motivation: Independent Deployability

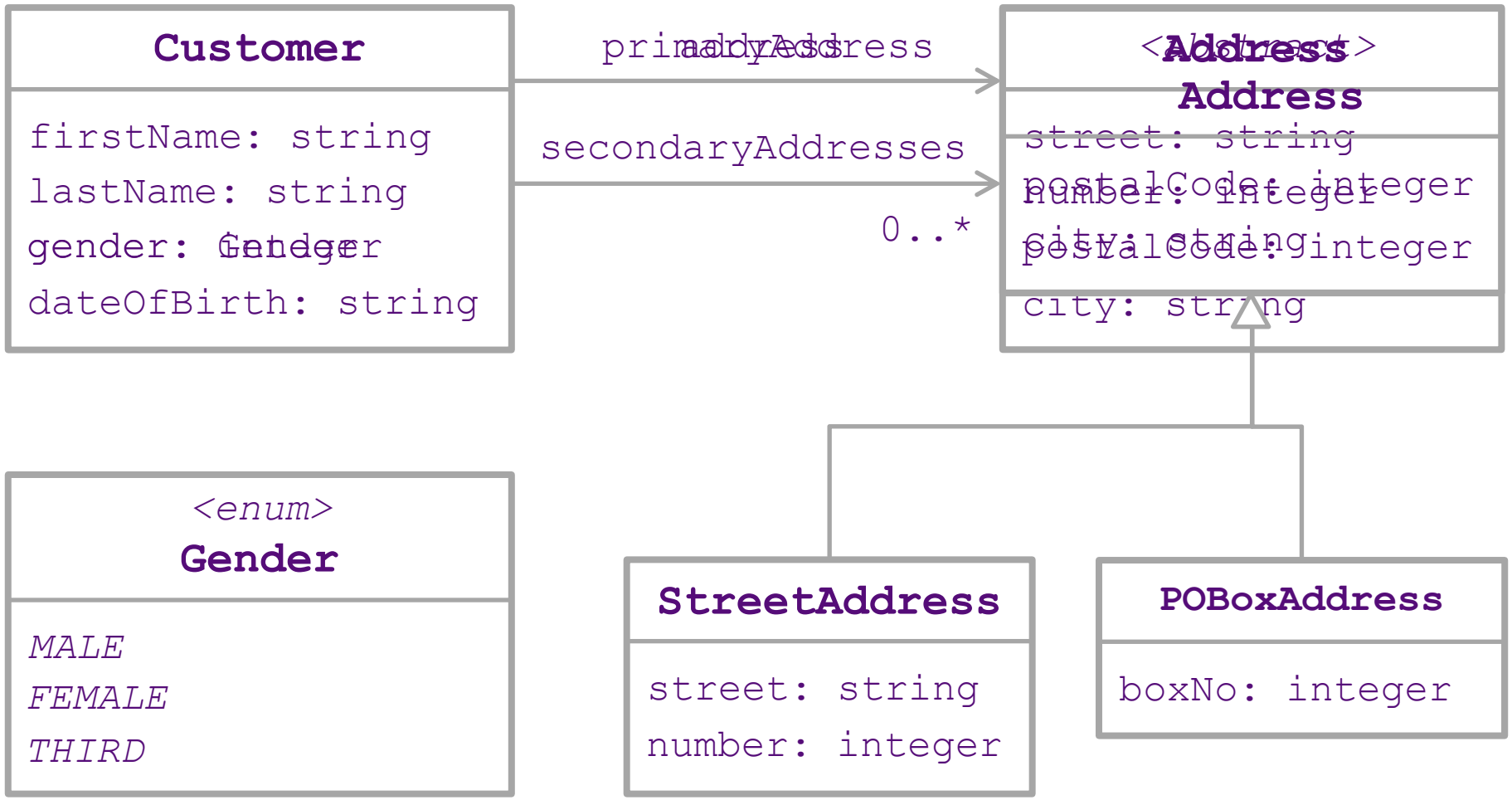
Lock-step Deployment



Independent Deployment



API Evolution Example



What About (Semantic) Versioning?

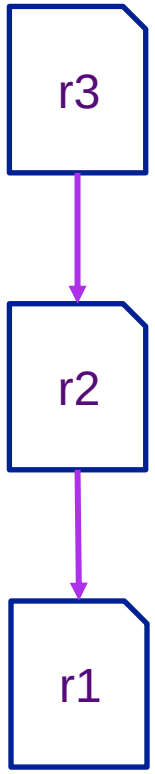
Question: How can we make sure that new versions are created when and only when necessary?

Observation: A notion of history is required to classify changes.

Question: How can we avoid unneccessary code duplication?

Observation: At some point, the versioned paths tend to converge to allow for code reuse.

Our Approach: Overview



API Definition
History

Provider

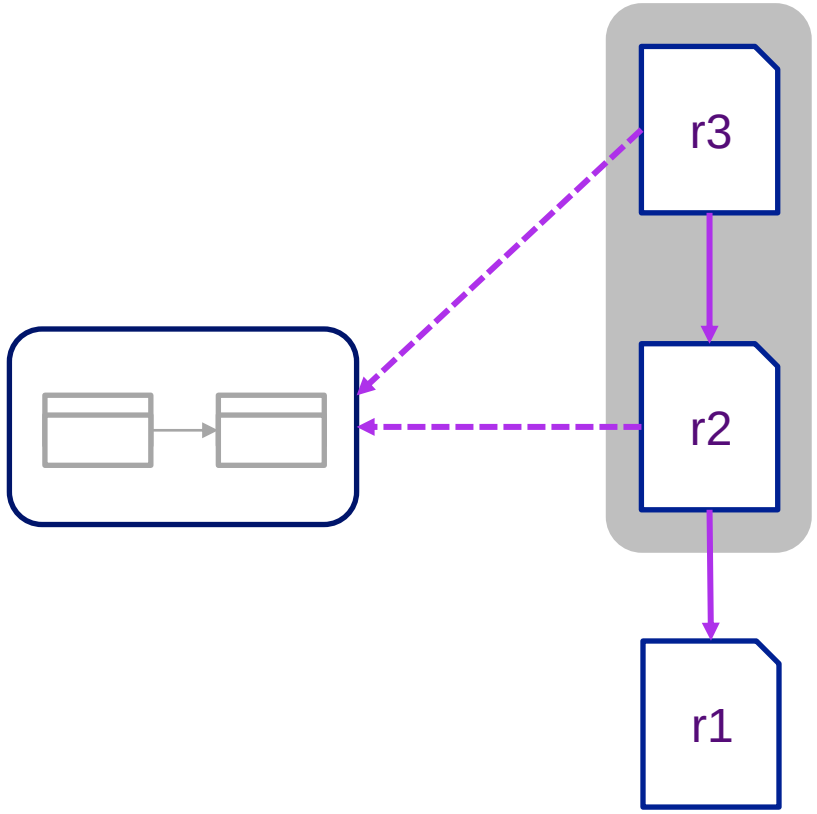
Consumer

Our Approach: Provider API Definitions


```
record Customer {
 string firstName
 string lastName
 int32 gender
 Address address
}
```

```
record Customer {
 string firstName
 string lastName
 string dateOfBirth
 Address primaryAddress replaces address
 Address* secondaryAddresses
 Gender gender as genderNew
}
```

Our Approach: Overview



Internal Representation

API Definition
History

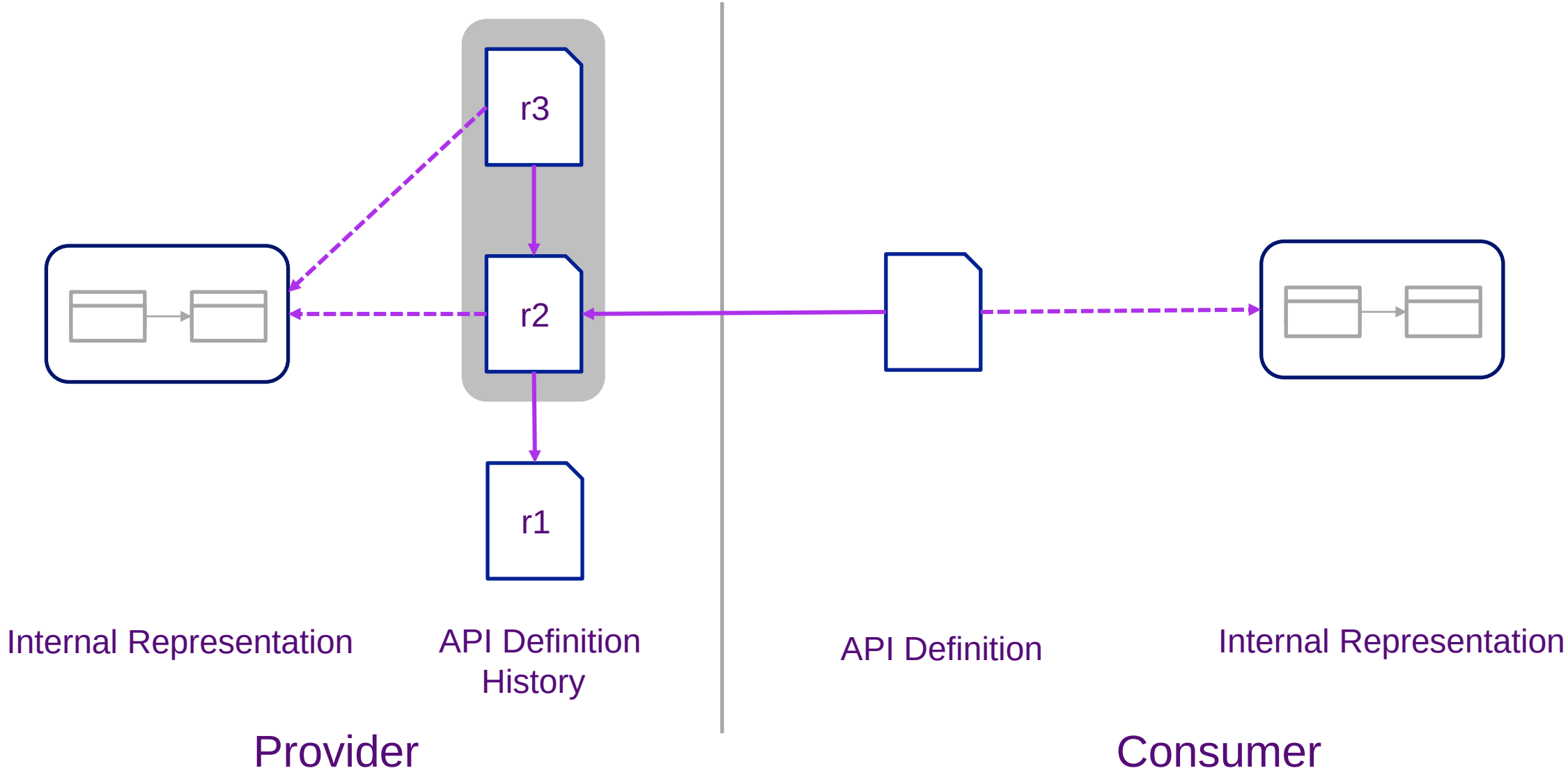
Provider

Consumer

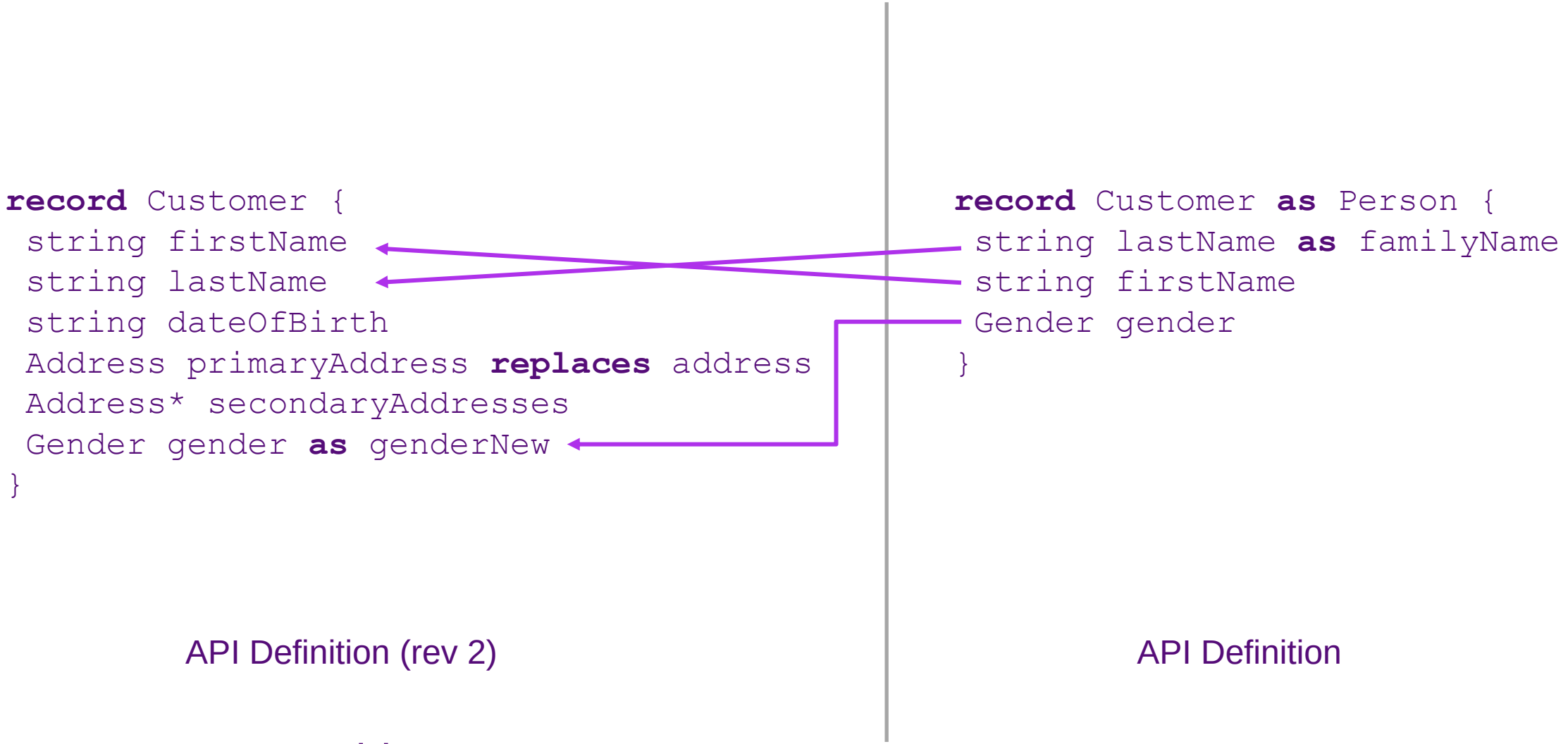
Our Approach: Internal Representations


```
class Customer {
 // Appear in all revisions
 String firstName;
 String lastName;
 // opt-in: Newer clients will not provide this field
 Optional<int> gender;
 // opt-in: Older clients will not provide this field
 Optional<String> dateOfBirth;
 // Appears in all revisions, but has the latest name
 Address primaryAddress;
 // Implicitly optional
 List<Address> secondaryAddresses;
 // Explicit internal name to avoid name clashes
 Optional<Gender> genderNew;
}
```

Our Approach: Overview

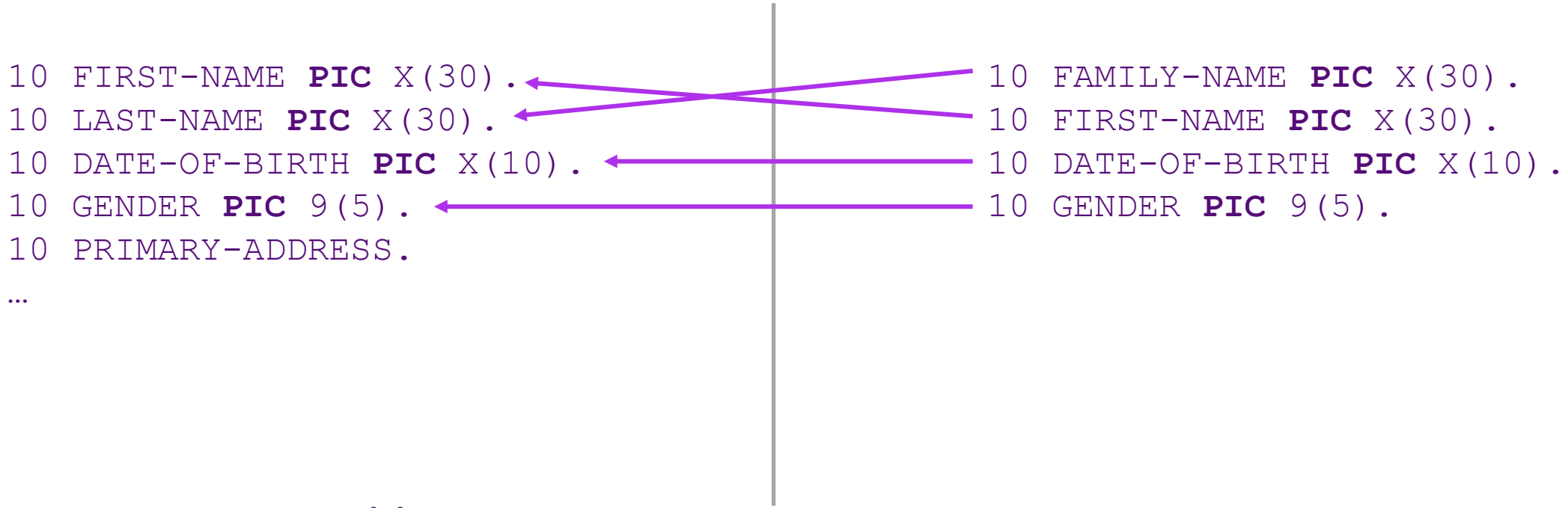


Our Approach: Consumer API Definitions



Our Approach: Extensions for COBOL

- All type occurrences must have explicit lengths (e.g., `string(30)`)
- The types `string(30)` and `string(40)` are different



Feasibility Evaluation

Goals of the Approach

The approach should:

1. …be easy to use for developers
2. …have little impact on a developer‘s work
3. …support aged implementation technologies
4. …have an acceptable performance impact

Performed Experiments

- Survey among developers
- PoC implementation

Feasibility Evaluation

Results

- The approach is generally considered usable in practice, but concerns about the programming model (optionals)
- Performance is not an issue for remote invocations, but may be for in-process invocations

Current and Future Work

- Extending and improving the prototype
- Formal treatment of the evolution semantics

- Investigate programming model and tooling
- Explore opportunities to integrate into OpenAPI, …

Code is available at

`https://github.com/holgerknoche/gutta-apievolution`