Advanced Rust (2026): Borrow-Checker-Driven API Design

Lecture 2

Lukáš Hozda

Spring 2026

MFF CUNI

Borrowing As Architecture

Why This Topic Matters

- Borrow checker is not syntax police, it is architecture feedback.
- Most hard compile errors are design errors, not annotation errors.
- Good API shape reduces both cloning and lifetime noise.

Ownership Roles

- Separate *owner* and *view* roles.
- Owners control allocation and mutation rights.
- Views expose read-only access with explicit lifetimes.

```rust
struct Document {
    bytes: Vec<u8>,
}

struct DocumentView<'a> {
    bytes: &'a [u8],
}
```

Return Types Decide Coupling

- Returning `String` transfers ownership.
- Returning `&str` ties caller to callee lifetime.
- Returning iterators can delay allocation decisions.

```
fn line_owned(input: &str) -> String { input.to_owned() }
fn line_view(input: &str) -> &str { input }
fn words<'a>(input: &'a str) -> impl Iterator<Item = &'a str> {
    input.split_whitespace()
}
```

`&self` vs `&mut self`

- `&self`: multiple shared readers.
- `&mut self`: unique mutable access.
- `&mut self` in public API is a strong design statement.

```rust
struct Counter { value: i64 }

impl Counter {
    fn get(&self) -> i64 { self.value }
    fn add(&mut self, delta: i64) { self.value += delta; }
}
```

Two-Phase Access Pattern

- Typical mistake: hold mutable borrow too long.
- Compute indices/keys before borrowing mutably.
- Short borrow scopes are easier to compose.

```rust
let idx = values.iter().position(|x| *x == needle);
if let Some(i) = idx {
    values[i] += 1;
}
```

Slices Over Ownership Transfer

- `&[T]` communicates read-only bulk access.
- `&mut [T]` communicates in-place transformation.
- Prefer slices for algorithmic helpers.

```rust
fn normalize(xs: &mut [i64]) {
    let min = xs.iter().copied().min().unwrap_or(0);
    for x in xs { *x -= min; }
}
```

Lifetime Elision Is Not Magic

- Elision follows fixed rules.
- When multiple references exist, explicit lifetimes clarify intent.

```rust
fn pick_left<'a>(left: &'a str, _right: &str) -> &'a str {
    left
}
```

Borrowing In Struct APIs

- Store owned data in structs.
- Expose borrowed views in methods.

```rust
struct LogBuffer {
    lines: Vec<String>,
}

impl LogBuffer {
    fn as_slice(&self) -> &[String] { &self.lines }
}
```

Iterator Lifetimes

- Iterators often borrow source collections.
- `impl Iterator + '_` keeps API concise and accurate.

```rust
fn evens(xs: &[i64]) -> impl Iterator<Item = i64> + '_ {
    xs.iter().copied().filter(|x| x % 2 == 0)
}
```

Interior Mutability Boundaries

- `Cell` and `RefCell` move checks to runtime.
- Use for local invariants, not global escape hatch.
- Panic risk (`RefCell`) must be part of correctness story.

```rust
use std::cell::RefCell;

let value = RefCell::new(10_i64);
*value.borrow_mut() += 1;
```

Borrow Checker Diagnostic Reading

- Start from first error in chain.
- Identify long-lived borrow that blocks operation.
- Refactor scopes before adding clones.

Design Checklist

- API names indicate ownership flow.
- Borrow duration is minimal and intentional.
- Clones are explicit and justified.
- Lifetimes appear only where semantics require them.