Introductory Rust (2026): Concurrency Foundations

Lecture 5

Lukáš Hozda

Winter 2026/27

MFF CUNI

Shared Work

Thread Basics

```rust
use std::thread;

let handle = thread::spawn(|| 40 + 2);
println!("{}", handle.join().unwrap());
```

Ownership Across Threads

- Closure for `thread::spawn` is `move` by default in practice.
- Captured data must be `Send + 'static`.

Shared State With Arc<Mutex<T>>

```rust
use std::sync::{Arc, Mutex};
use std::thread;

let counter = Arc::new(Mutex::new(0_i64));
let mut handles = Vec::new();

for _ in 0..4 {
    let c = Arc::clone(&counter);
    handles.push(thread::spawn(move || {
        *c.lock().unwrap() += 1;
    }));
}
for h in handles { h.join().unwrap(); }
println!("{}", *counter.lock().unwrap());
```

Channels

```rust
use std::sync::mpsc;

let (tx, rx) = mpsc::channel();
tx.send(10).unwrap();
println!("{}", rx.recv().unwrap());
```

- Message passing avoids shared mutable state.

Deterministic Output

- Parallel processing order may vary.
- Final external output should often be stabilized.
- Sort by ID before printing results.

Deadlocks

- Opposite lock order can block forever.
- Keep lock order global and consistent.

Data Race vs Logic Race

- Safe Rust forbids memory data races.
- Race conditions in logic are still possible.

Practical Rule

- Start with channels when possible.
- Use mutexes for unavoidable shared mutable state.
- Keep critical sections small.