Advanced Rust (2026): Performance and Profiling

Extension Lecture

Lukáš Hozda

Spring 2026

MFF CUNI

Measurement Discipline

Performance Workflow

- State hypothesis before tuning.
- Build reproducible benchmark setup.
- Interpret numbers with hardware context.

Common Benchmark Traps

- Measuring debug builds.
- Tiny inputs with noise-dominated results.
- I/O in benchmark loop.
- Branch predictor warmup ignored.

Criterion Basics

```rust
use criterion::{black_box, Criterion};

fn bench_sum(c: &mut Criterion) {
    let data: Vec<i64> = (0..100_000).collect();
    c.bench_function("sum", |b| {
        b.iter(|| data.iter().copied().sum::<i64>())
    });
}
```

`black_box` Semantics

- Prevents optimizer from deleting work.
- Does not model real-world workloads by itself.

CPU Cache Effects

- Access pattern often dominates arithmetic cost.
- Contiguous scans beat random probing.
- False sharing can erase multicore scaling.

Allocation Pressure

- Reuse buffers where practical.
- Profile allocation hot paths.
- Avoid premature custom allocator complexity.

Lock Contention

- Throughput collapse can come from tiny critical sections.
- Measure lock hold times and queueing.
- Compare lock-free alternatives only with correctness proof.

Profiling Stack

- Sampling profiler for hotspots.
- `perf`/flamegraph for call-tree shape.
- Event counters for cache misses and branches.

Optimization Ordering

- Algorithmic improvements first.
- Data-layout improvements second.
- Micro-optimizations last.

Report Format

- baseline
- change
- delta
- confidence/variance
- regression risk

Final Rule

- Performance claims without reproducible methodology are not engineering evidence.