# Reducing JIT Runtime Overhead in .NET & C#

Zyad Khaled

## The Problem in .NET

C# code compiles to **IL (Intermediate Language)**, not machine code. At runtime, the **CLR JIT compiler** converts IL to native code, which creates overhead - especially at startup when many methods need compilation.

## How .NET Reduces JIT Overhead

### 1. Tiered Compilation (Default since .NET Core 3.0)

- **Tier 0:** Quick JIT compilation with minimal optimization
- **Tier 1:** Re-compile hot methods with full optimizations
- Methods are marked "hot" after 30 calls
- Result: Fast startup + optimized steady-state performance

### 2. Ready-to-Run (R2R) / AOT Compilation

- Pre-compile IL to native code **before deployment**
- Use `dotnet publish -p:PublishReadyToRun=true`
- JIT still used for fallback and optimization
- Benefit: Eliminates most startup JIT overhead

### 3. Native AOT (.NET 7+)

- Compile **entire app** to native machine code
- No JIT at runtime, no IL shipping
- Smaller deployment, faster startup, lower memory
- Trade-off: Loses some dynamic features (reflection limits)

### 4. NGen (Legacy Tool)

- Pre-compile assemblies to native images on installation
- Used in .NET Framework (before .NET Core)
- Mostly replaced by R2R in modern .NET

### 5. Code Caching

- CLR caches compiled methods in memory
- Methods compiled once per app run
- No recompilation for repeated calls

### 6. Profile-Guided Optimization (PGO)

- **Dynamic PGO** (enabled in .NET 6+)
- Collects runtime profiling data
- JIT uses this data to optimize hot paths better
- Works with Tiered Compilation

## Summary

**.NET Strategy:** Use R2R/AOT for startup, Tiered JIT + PGO for runtime optimization, caching to avoid recompilation.