

A Tour of Ox 1.0: Direct-Style Concurrency and Resiliency

Adam Warski, August 2025
warski.org



sbt

Before we dive into the code...

WHY?

- 1. Using FP results in better code**
- 2. Scala is the best FP language**



[Why Scala?](#)



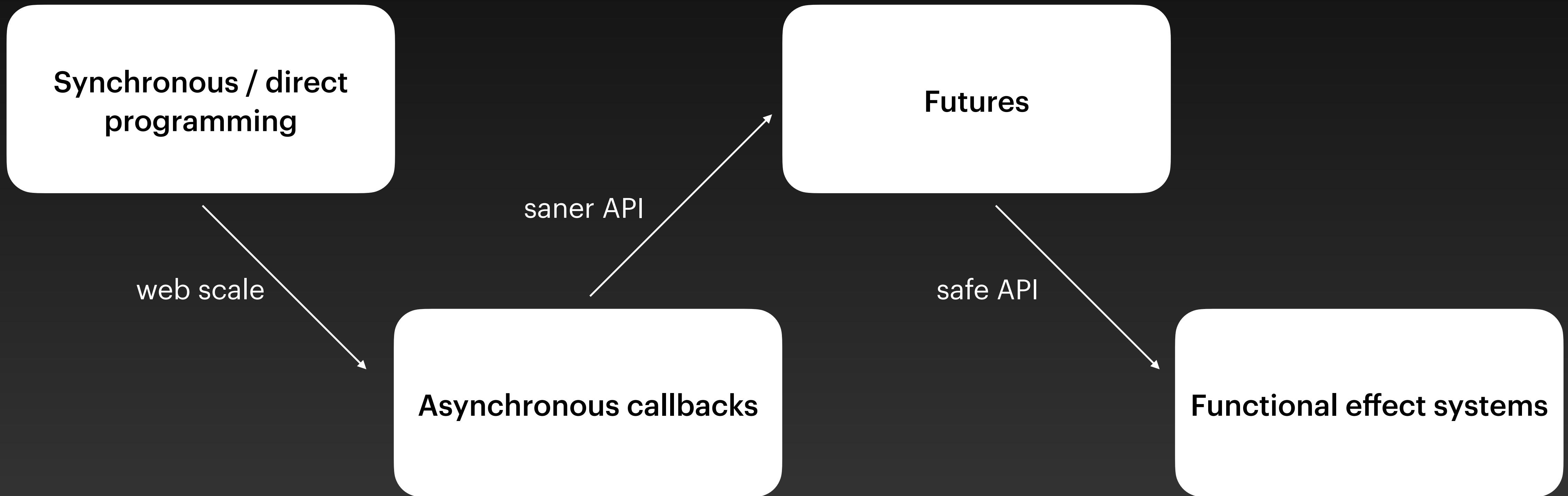
[Why Scala? \(YT\)](#)

What is FP?

Casual FP	Pure FP
Composing functions	
Light syntax for lambdas	
Higher-order functions	🍀 No mutation
Immutable data types	🍀 No side effects
ADTs + pattern matching	🍀 All computations are lazy
Expression-oriented	
Functional standard library	
Separate data & behavior	



How did we arrive at cats-effect or ZIO?



Java 21

- We now have a lightweight-thread / fiber runtime directly in the JVM
- Multiple threads/fibers are scheduled to run on a small pool of OS threads
 - cats-effect/ZIO: library level
 - virtual threads: runtime level

There's more to functional effects than fibers

The good	The bad
🍀 Error handling	💔 Syntax overhead
🍀 Resource management	💔 Custom control flow
🍀 Fearless refactoring	💔 Lost error context
🍀 Principled interruptions	💔 Virality
🍀 High-level concurrency	💔 Learning curve
🍀 Streaming	



Can we ...

- Use casual FP
- Leverage Java 21's virtual threads
- Keep some of the benefits of functional effect systems
- But avoid some of the problems



The screenshot shows a browser window displaying the Ox documentation at ox.softwaremill.com. The page has a dark theme with a light blue header bar. The left sidebar contains a search bar and several navigation sections: PROJECT INFO (Community & support, Dependency (sbt, scala-cli, etc.), Project scope, Using Ox with AI coding assistants); BASICS (A tour of ox, Direct style, Error handling); HIGH-LEVEL CONCURRENCY (Running computations in parallel, Race two computations, Parallelize collection operations, Timeout a computation); and STRUCTURED CONCURRENCY. The main content area features a large heading "Ox" with a link icon, followed by a brief description: "Safe direct-style streaming, concurrency and resiliency for Scala on the JVM. Requires JDK 21+ & Scala 3.". It then provides instructions for adding the dependency: "To start using Ox, add the `com.softwaremill.ox::core:1.0.0-RC2` dependency to your project. Then, take a look at the tour of Ox, or follow one of the topics listed in the menu to get to know Ox's API!". Below this, it says "In addition to this documentation, ScalaDocs can be browsed at <https://javadoc.io>". A section titled "A tour of ox" follows, with the sub-instruction "Run two computations [in parallel](#):". A code block is shown:

```
def computation1: Int = { sleep(2.seconds); 1 }
def computation2: String = { sleep(1.second); "2" }
val result1: (Int, String) = par(computation1, computation2)
// (1, "2")
```

Below the code, there is another instruction: "Timeout a computation:".



Ox docs

Now, let's dive into the code

A tour of Ox

```
def computation1: Int = { sleep(2.seconds); 1 }
def computation2: String = { sleep(1.second); "2" }
val result1: (Int, String) = par(computation1, computation2)
// (1, "2")
```

```
def computation3: Int = { sleep(2.seconds); 1 }
val result2: Either[TimeoutException, Int] =
  timeout(1.second) (computation3).catching[TimeoutException]
```

supervised:

```
val f1 = fork { sleep(2.seconds); 1 }
val f2 = fork { sleep(1.second); 2 }
(f1.join(), f2.join())
```

supervised:

```
val user = fork(findUser())
val order = fork(fetchOrder())
Response(user.join(), order.join())
```

Compare with Java's upcoming standard:

```
try (var scope = StructuredTaskScope.open()) {
    var user = scope.fork(() -> findUser());
    var order = scope.fork(() -> fetchOrder());

    scope.join();

    return new Response(user.get(), order.get());
}
```

```
def computationR: Int = ???  
retry(Schedule.exponentialBackoff(100.millis).maxRetries(4)  
.jitter().maxInterval(5.minutes)) (computationR)
```

```
def computationR: Int = ???  
repeat(Schedule.fixedInterval(100.millis)) (computationR)
```

```
supervised:  
  val rateLimiter = RateLimiter.fixedWindowWithStartTime(2, 1.second)  
  rateLimiter.runBlocking { /* ... */ }
```

```
useCloseable(new java.io.PrintWriter("test.txt")) { writer =>
    writer.println("Hello, world!")
}
```

```
supervised:  
  val writer = useCloseableInScope(new java.io.PrintWriter("test.txt"))  
  // ... use writer ...
```

```
object MyApp extends OxApp:  
  def run(args: Vector[String])(using Ox): ExitCode =  
    // ... your app's code ...  
    // might use fork {} to create top-level background threads  
  ExitCode.Success
```

```
val c = Channel.buffered[String] (8)
c.send("Hello,")
c.send("World")
c.done()
```

```
val c = Channel.rendezvous[Int]
val d = Channel.rendezvous[Int]
// use c & d in forks
select(c.sendClause(10), d.receiveClause)
```

```
Flow.iterate(0) (_ + 1) // natural numbers
  .filter(_ % 2 == 0)
  .map(_ + 1)
  .intersperse(5)
  // compute the running total
  .mapStateful(0) { (state, value) =>
    val newState = state + value
    (newState, newState)
  }
  .take(10)
  .runForEach(n => println(n.toString))
```

```
def sendHttpRequest(entry: String): Unit = ???  
Flow  
.fromInputStream(this.getClass().getResourceAsStream("/list.txt"))  
.linesUtf8  
.mapPar(4) (sendHttpRequest)  
.runDrain()
```

```
val f1 = Flow.tick(123.millis, "left")
val f2 = Flow.tick(312.millis, "right")
f1.merge(f2).take(100).runForeach(println)
```

```
def readNextBatch(): List[String] = ???  
Flow.usingEmitt: emit =>  
    forever:  
        readNextBatch().foreach(emit.apply)
```

```
KafkaFlow
  .subscribe(consumerSettings, sourceTopic)
  .map(in => (in.value.toLong * 2, in))
  .map((value, original) =>
    SendPacket(
      ProducerRecord[String, String](destTopic, value.toString),
      original))
  .pipe(KafkaDrain.runPublishAndCommit(producerSettings))
```

```
def lookupUser(id1: Int): Either[String, User] = ???  
def lookupOrganization(id2: Int): Either[String, Organization] = ???  
  
val result: Either[String, Assignment] = either:  
  val user = lookupUser(1).ok()  
  val org = lookupOrganization(2).ok()  
  Assignment(user, org)
```

```
val userInput: Boolean = ???  
def process = if userInput  
  then 10  
  else throw new IllegalArgumentException("boom")  
  
val result: Either[IllegalArgumentException, Int] =  
  process.catching[IllegalArgumentException]  
  
val v2: Either[Exception, Int] = Left(new RuntimeException("boom!"))  
v2.orThrow // throws RuntimeException("boom!")
```

```
def compute: Int = ???  
def computeMore(v: Int): Long = ???  
compute  
.pipe(2 * _)  
.tap(println)  
.pipe(computeMore)
```

Performance



Benchmarks use the harness from the Kyo library:

- streamlined algebraic effects
- also approaching 1.0
- peak performance
- functional effect system
- ran using Java 24, MacBook M1 Max

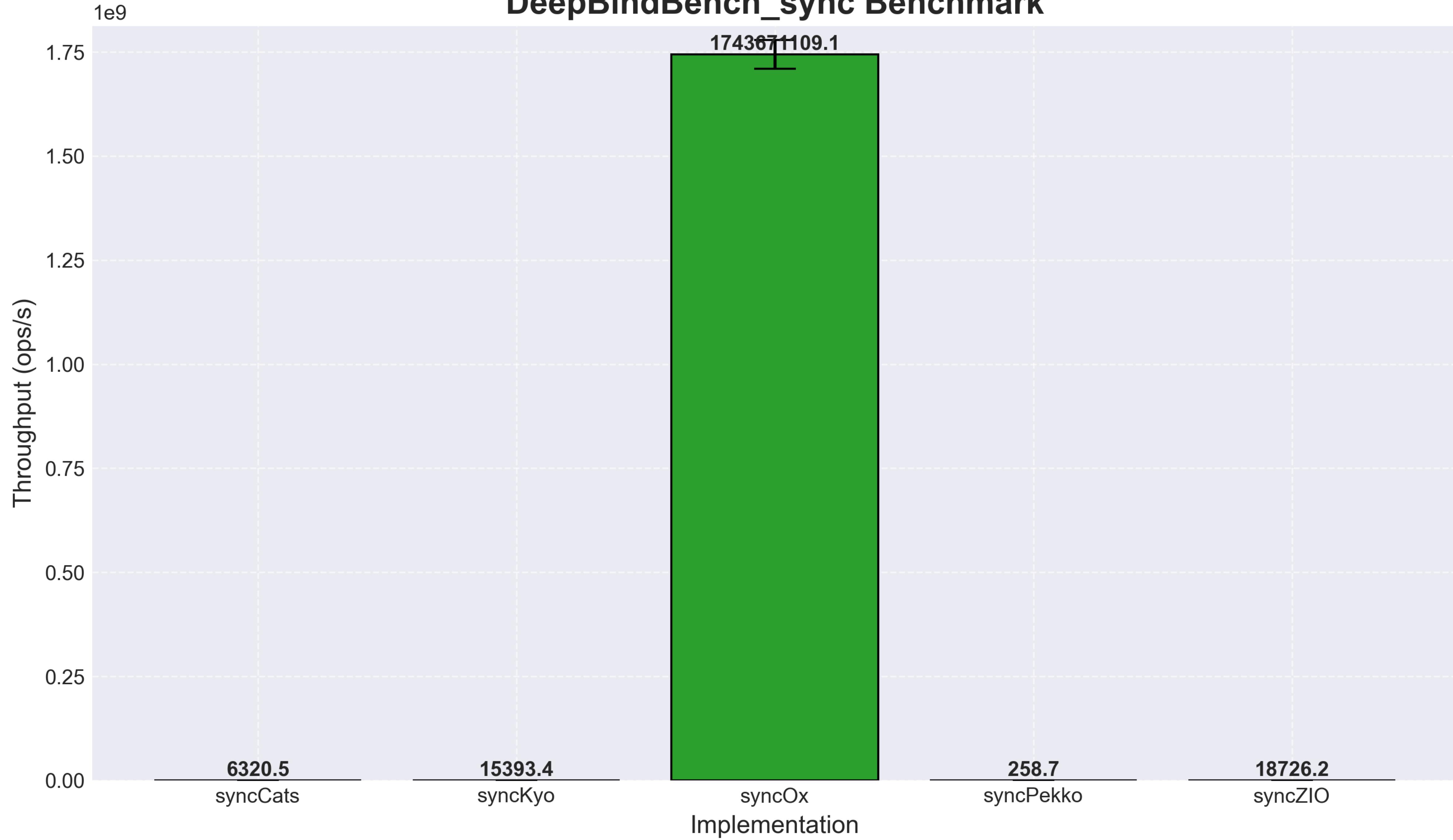


```
// DeepBindBench

def catsBench() =
  import cats.effect.~
  def loop(i: Int): IO[Unit] =
    IO.unit.flatMap { _ =>
      if i > depth then
        IO.unit
      else
        loop(i + 1)
    }
  loop(0)
end catsBench

def oxBench() =
  def loop(i: Int): Unit = if i > depth then () else loop(i + 1)
  loop(0)
```

DeepBindBench_sync Benchmark



```
// ForkJoinBench
val range = (0 until 10000).toList

def catsBench() =
  import cats.*; import cats.effect.*

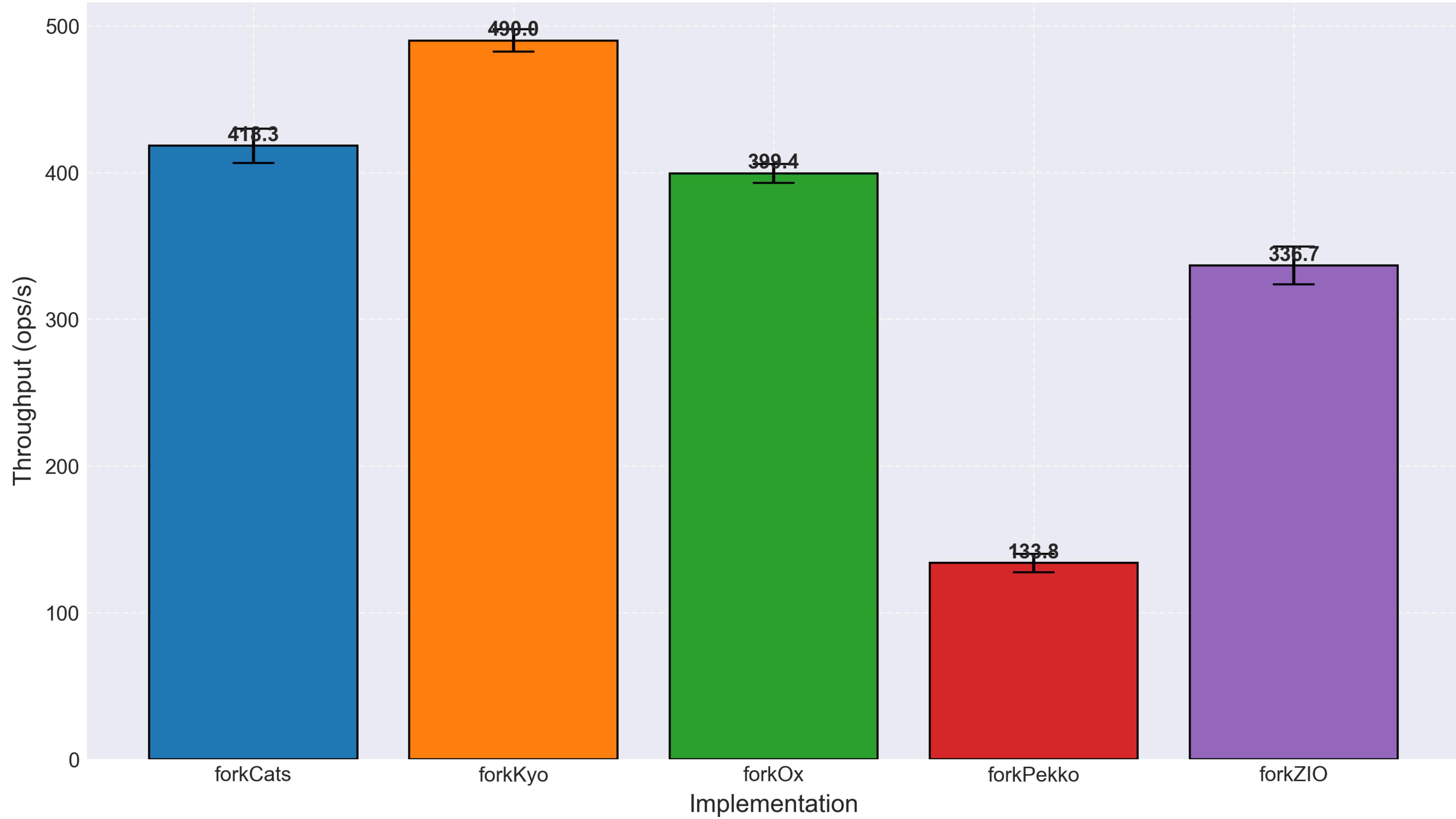
  val forkFiber      = IO.unit.start
  val forkAllFibers = Traverse[List].traverse(range) (_ => forkFiber)

  forkAllFibers.flatMap(fibers =>
    Traverse[List].traverse(fibers) (_.join).void)

def oxBench() =
  import ox.*

supervised:
  val forks = range.map(_ => fork(()))
  forks.foreach(_.join())
```

ForkJoinBench Benchmark



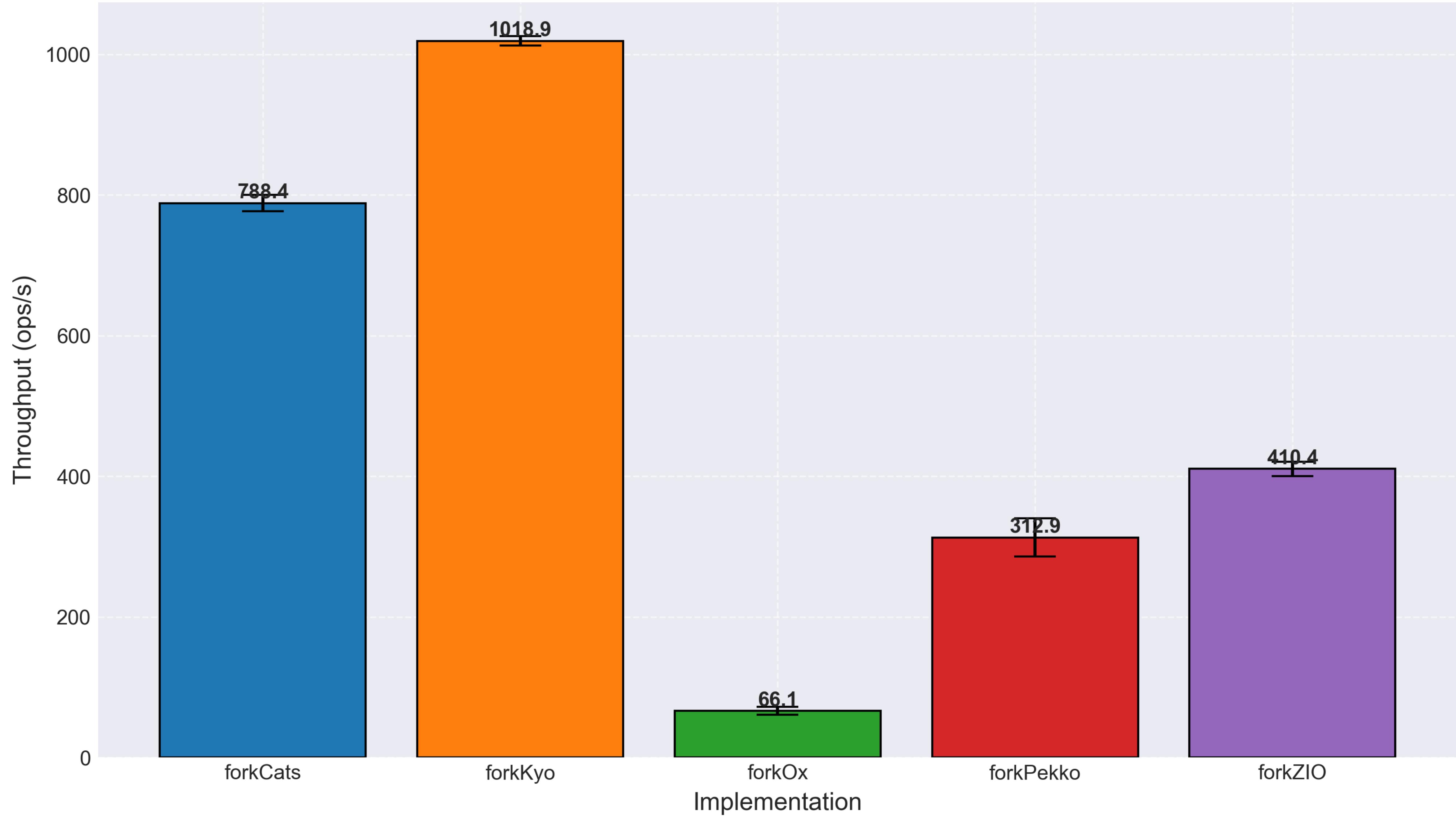
```
// ForkChainedBench
def catsBench() =
  def iterate(deferred: Deferred[IO, Unit], n: Int): IO[Any] =
    if n <= 0 then deferred.complete(())
    else IO.unit.flatMap(_ => iterate(deferred, n - 1).start)

  for
    deferred <- IO.deferred[Unit]
    _           <- iterate(deferred, 10000).start
    _           <- deferred.get
  yield 0

def oxBench() =
  def iterate(p: Channel[Unit], n: Int) (using Ox): Unit =
    if n <= 0 then p.send(())
    else forkDiscard(iterate(p, n - 1))

supervised:
  val p = Channel.buffered[Unit](1)
  forkDiscard(iterate(p, 10000))
  p.receive()
  0
```

ForkChainedBench Benchmark



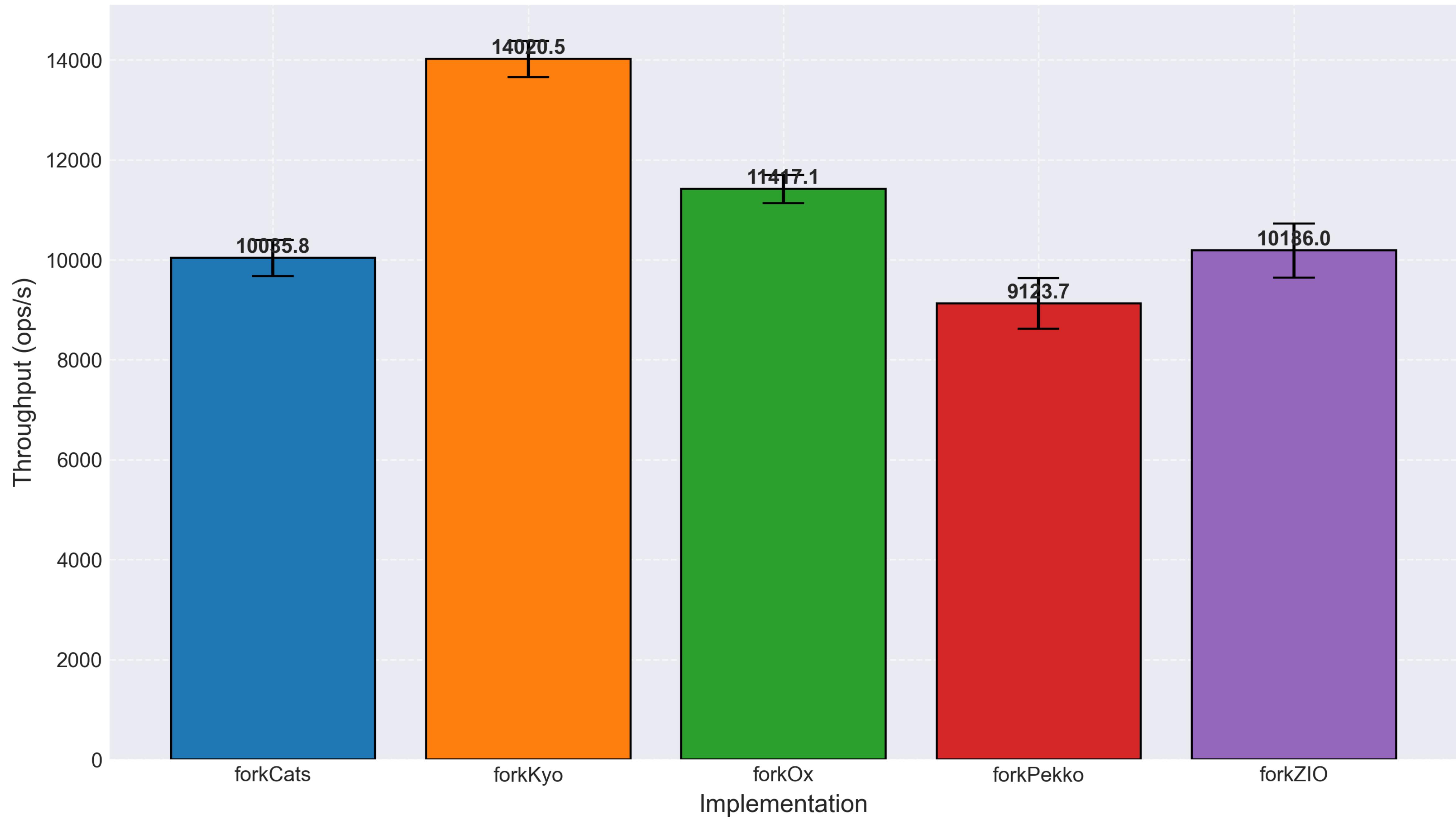
```
// HttpClientBench
lazy val catsClient =
  given LoggerFactory[IO] = Slf4jFactory.create[IO]
  EmberClientBuilder.default[IO].build.allocated.unsafeRunSync()._1

val catsUrl = Uri.fromString(url).toOption.get

def catsBench() = catsClient.expect[String](catsUrl)

def oxBench() =
  import sttp.client4.quick.*
  quickRequest.get(uri"$url").send().body
```

HttpClientBench Benchmark



```
// SemaphoreBench
def catsBench() =
  def loop(s: Semaphore[IO], i: Int): IO[Unit] =
    if i >= depth then
      IO.unit
    else
      s.acquire.flatMap(_ => s.release).flatMap(_ => loop(s, i + 1))
```

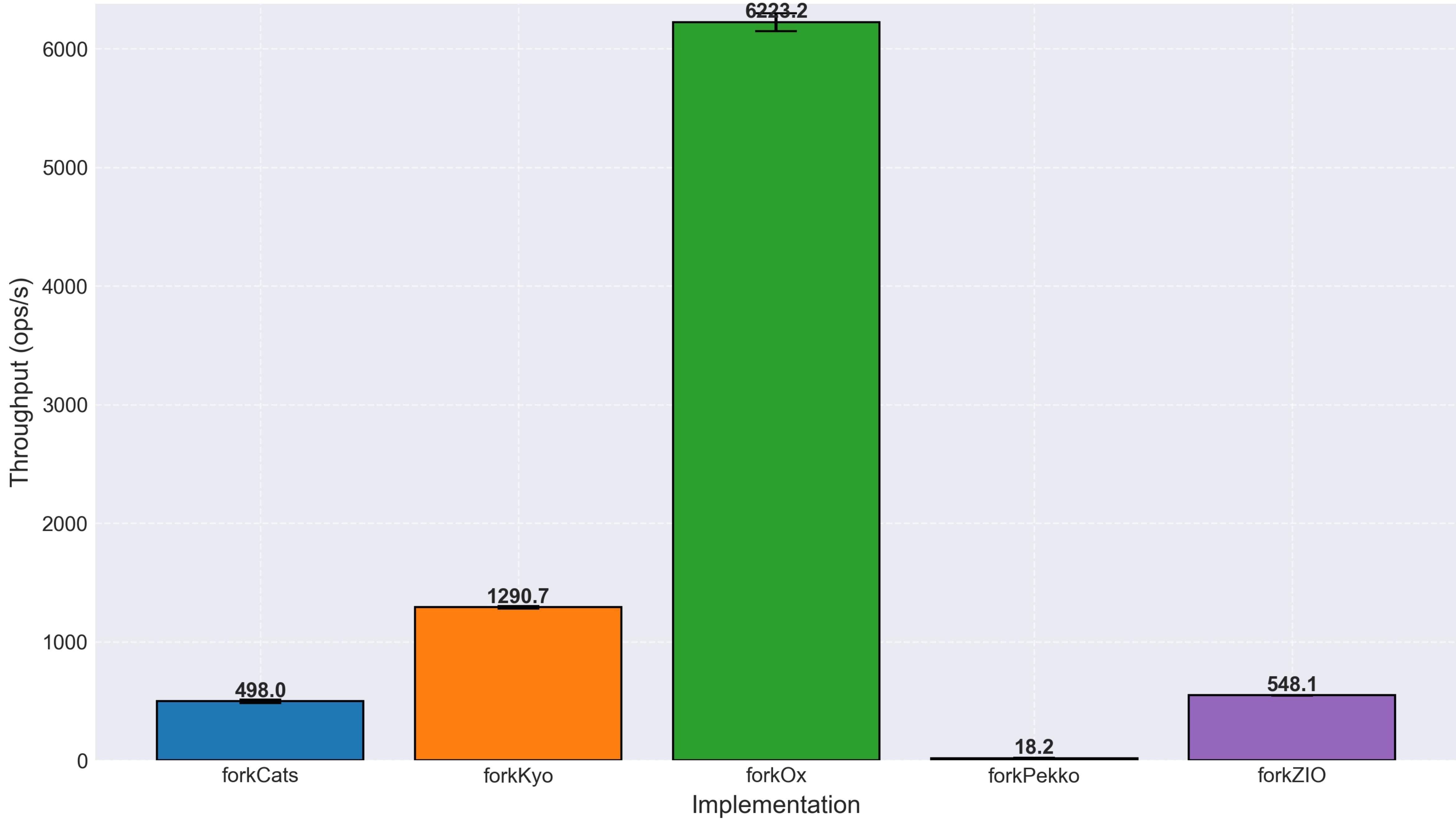
```
Semaphore[IO](1).flatMap(loop(_, 0))
```

```
def oxBench() =
  import ox./*
  import java.util.concurrent.Semaphore
```

```
def loop(s: Semaphore, i: Int): Unit =
  if i >= depth then ()
  else
    s.acquire()
    s.release()
    loop(s, i + 1)
```

```
loop(new Semaphore(1), 0)
```

SemaphoreBench Benchmark



```
// StreamBench
```

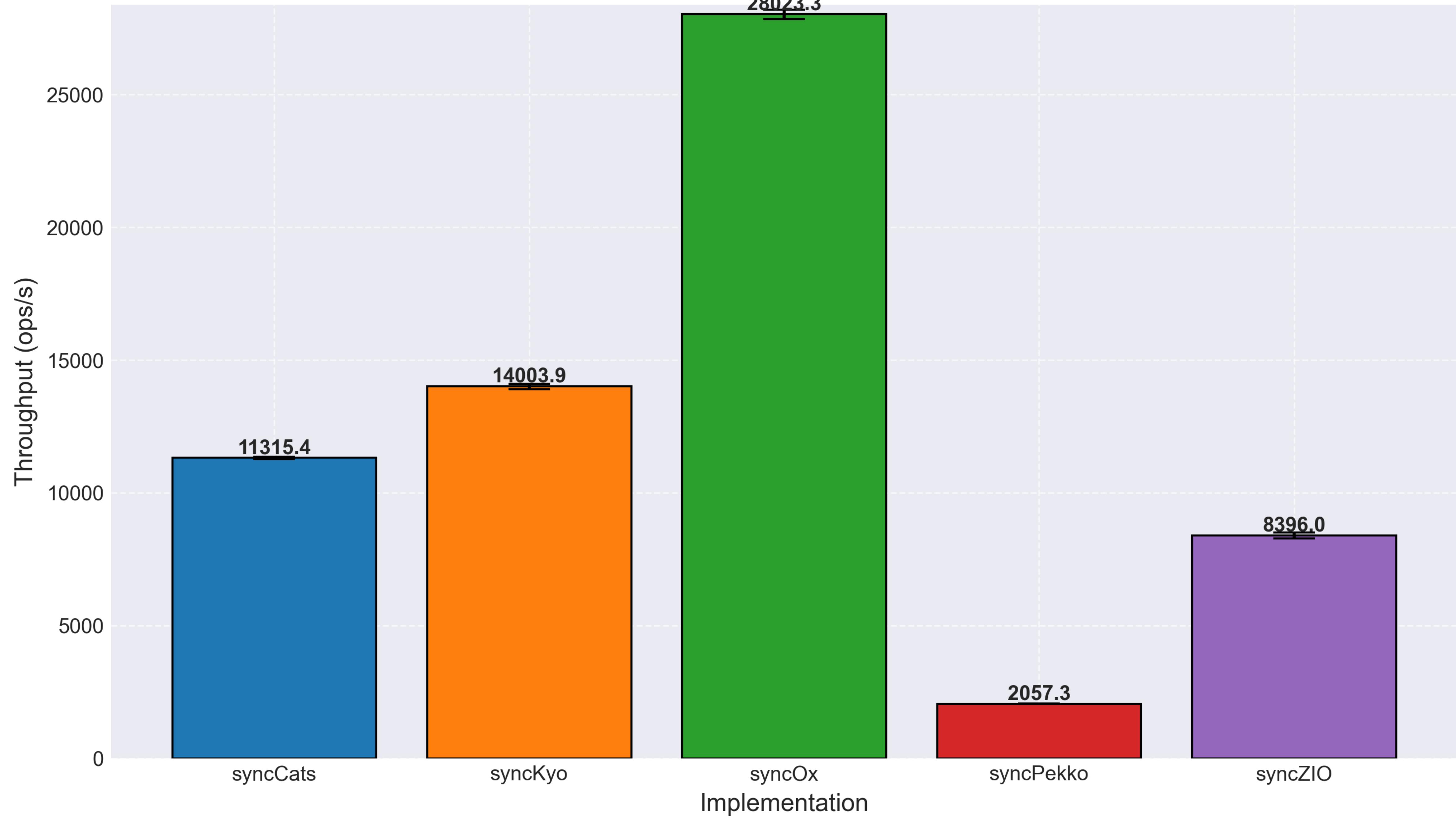
```
def catsBench() =  
  import cats.effect.*  
  import fs2.*
```

```
Stream.emits(seq)  
  .filter(_ % 2 == 0)  
  .map(_ + 1)  
  .covary[IO]  
  .compile  
  .fold(0) (_ + _)
```

```
def oxBench() =  
  import ox.flow.*
```

```
Flow.fromIterable(seq)  
  .filter(_ % 2 == 0)  
  .map(_ + 1)  
  .runFold(0) (_ + _)
```

StreamBench sync Benchmark



```
// StreamAsyncBench
```

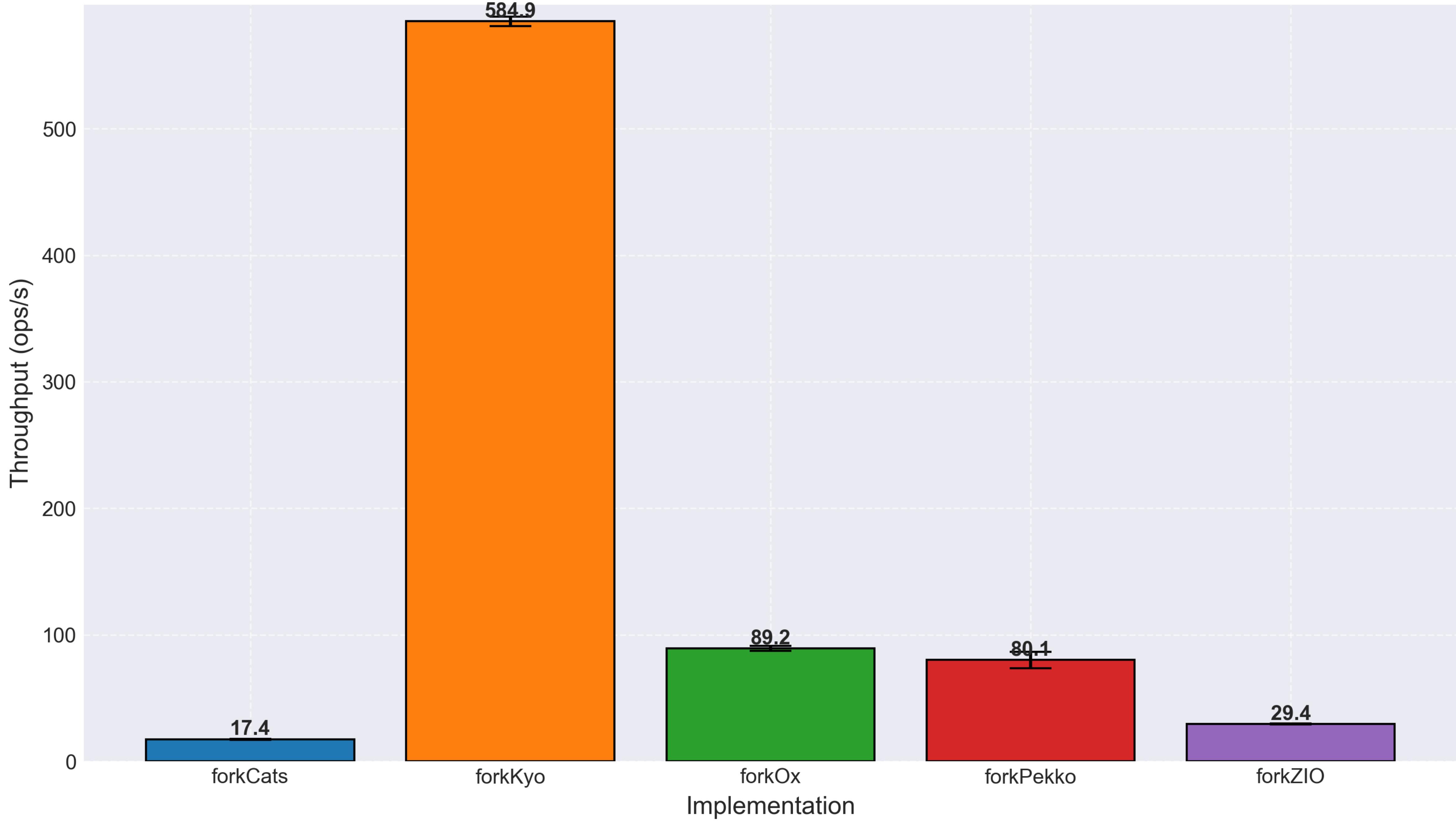
```
def catsBench() =  
  import cats.effect.*  
  import fs2.*
```

```
Stream.emits(seq)  
  .parEvalMap(6)(v => IO(v + 1))  
  .compile  
  .drain
```

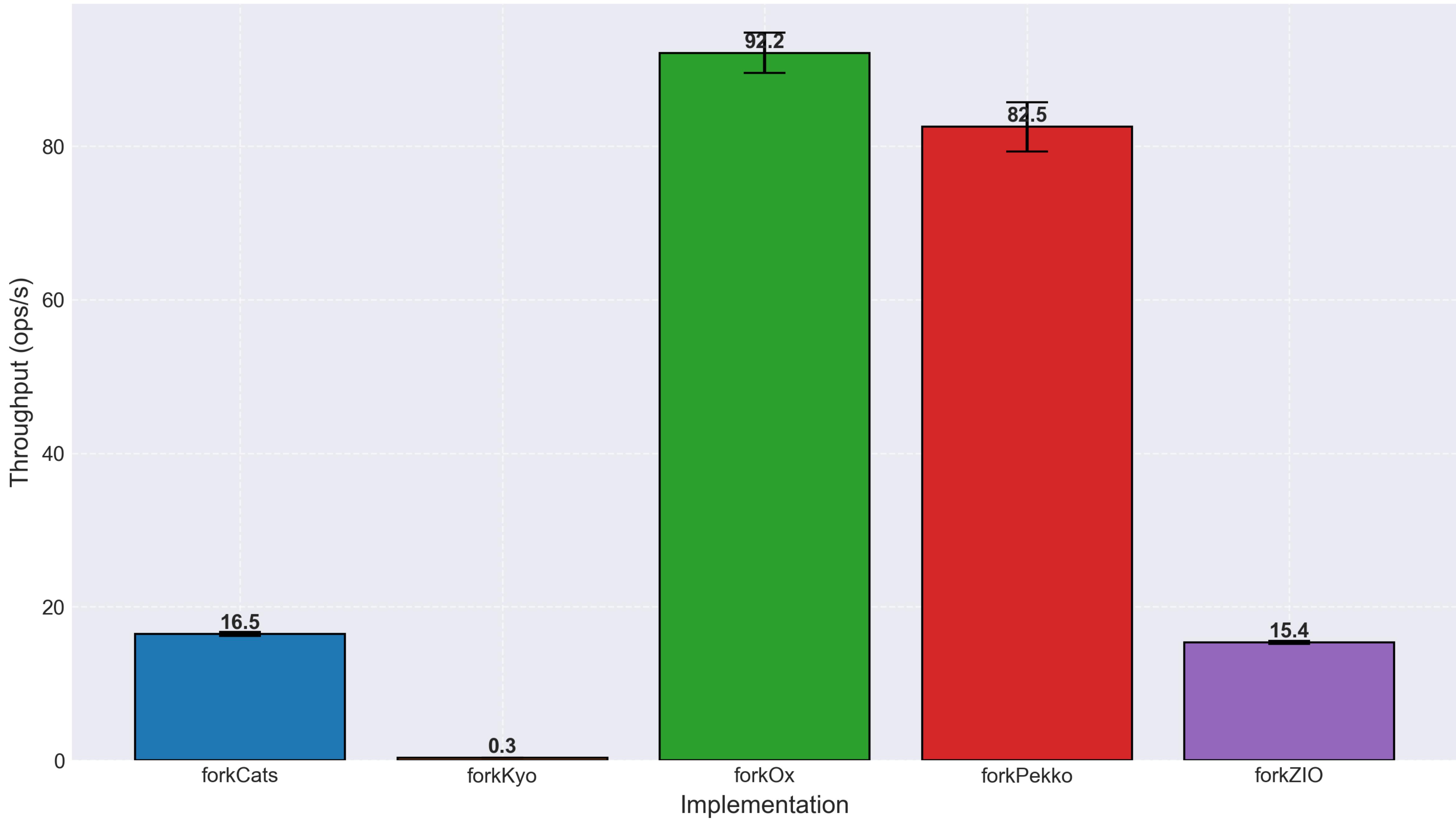
```
def oxBench() =  
  import ox.flow.*
```

```
Flow.fromIterable(seq)  
  .mapPar(6)(v => v + 1)  
  .runDrain()
```

StreamAsyncBench Benchmark



StreamAsyncNoChunkBench Benchmark



Ecosystem

```
@main def streamOx: Unit =  
  val backend = DefaultSyncBackend()  
  supervised:  
    releaseAfterScope(backend.close())  
  
  val result = basicRequest  
    .post(uri"https://httpbin.org/post")  
    .body(  
      Flow.fromValues(  
        Chunk.fromArray("Hello, ".getBytes()),  
        Chunk.fromArray("world!".getBytes()))).runToInputStream()  
    )  
    .response(asInputStream(is => Flow  
      .fromInputStream(is)  
      .decodeStringUtf8.runToList().mkString))  
    .send(backend)  
  
  println(s"Received: ${result.body}")
```

```
val wsEndpoint =  
endpoint.get  
.in("ws")  
.out(webSocketBody[String, CodecFormat.TextPlain,  
String, CodecFormat.TextPlain] (OxStreams))  
  
val wsPipe: Pipe[String, String] = in =>  
val responseFlow: Flow[String] = Flow.tick(1.second)  
.map(_ => System.currentTimeMillis()).map(_.toString)  
  
in.drain().merge(responseFlow, propagateDoneLeft = true)  
  
val wsServerEndpoint = wsEndpoint.handleSuccess(_ => wsPipe)  
  
@main def webSocketNettySyncServer(): Unit =  
NettySyncServer()  
.host("0.0.0.0")  
.port(8080)  
.addEndpoint(wsServerEndpoint)  
.startAndWait()
```

Private < > localhost ⌂ Register Login

Bootzooka Welcome Home

Hi there!

Welcome to Bootzooka!

In this template application you can [Register](#) as a new user, [Login](#) and later manage your user details.

Fork me on GitHub

Brought to you by

.I'IIII.

SOFTWARE MILL

If you are interested in how Bootzooka works,
you can browse the [Documentation](#) or [Source code](#).

Bootzooka - application scaffolding by [SoftwareMill](#), sources available
on [GitHub](#)

Version: 6ab784a0771a5c62b744fce0a62ddeb537d060bb



Bootzooka docs

Private < > localhost

Home > Explore > Tempo

Search... +k

Query history Share

Outline Tempo Split Add to dashboard < ⏪ ⏩ > ⏴ ⏵ ⏴ ⏵

A (Tempo)

Queries Node graph Traces

Query type Search TraceQL Service Graph Import trace Documentation

Build complex queries using TraceQL to select a list of traces.

7949a921af8c7e763b1a19d38e53e335

> Search Options Limit: 20 Spans Limit: 3 Table Format: Traces | Streaming: Disabled Metrics Options Step: auto Type: Range | Streaming: Disabled

+ Add query Query inspector

> Node graph

Trace

bootzooka: GET /api/v1/admin/version 22.06ms Give feedback Trace ID Export

2025-08-13 15:23:24.346 GET 200 /api/v1/admin/version

> Span Filters 1 spans Prev Next

0µs 5.51ms 11.03ms 16.54ms 22.06ms

Service & Operation 0µs 5.51ms 11.03ms 16.54ms 22.06ms

bootzooka GET /api/v1/admin/version (22.06ms) Log

This screenshot shows the Tempo UI interface for monitoring and tracing requests. The main area displays a trace for a 'bootzooka' request with a single span from 0µs to 22.06ms. The trace details section includes a 'TraceQL' tab, search options, and a 'Metrics' section. The service & operation view at the bottom shows the corresponding log entry for the request.

Plus, any blocking Scala or Java API

AI

The screenshot shows a web browser window with the URL ox.softwaremill.com in the address bar. The page content is about using Ox with AI coding assistants.

Left sidebar (Project Info):

- Community & support
- Dependency (sbt, scala-cli, etc.)
- Project scope

Left sidebar (Using Ox with AI coding assistants):

- Cursor documentation indexing
- Cursor rules
- Context7
- llms.txt

Left sidebar (Basics):

- A tour of ox
- Direct style
- Error handling

Left sidebar (High-level Concurrency):

- Running computations in parallel
- Race two computations
- Parallelize collection operations
- Timeout a computation

Left sidebar (Structured Concurrency):

Page header:

ox.softwaremill.com

Page title:

Using Ox with AI coding assistants

Page content:

Using Ox with AI coding assistants

If you are using AI coding assistants or agents, they will only be as useful, as much they know about Ox. Since Ox's documentation (especially the latest features) might not be in the LLMs training set, it might be useful to let the models know about Ox's capabilities and the preferred way that Ox should be used.

Since this is an evolving field, there's no one standard yet, and there are several options to explore. Below you can find a short summary.

Cursor documentation indexing

If you are using [Cursor](#), you might try the [built-in documentation indexing](#) feature. Select `@Docs` -> `Add New Doc` in the editor, or go to `Cursor` -> `Settings` -> `Cursor Settings` -> `Indexing & Docs` -> `Add docs`. In the address field, enter <https://ox.softwaremill.com/latest/>. After a while, the Ox documentation should be indexed.

Information is scarce on how this actually works, but by analogy with code indexing, this seems to store embeddings of documentation pages on Cursor's servers, which are then used for relevant user queries. Or using AI-terminology, it's a [RAG system](#).

You can then use `@Docs Ox` to hint to Cursor to use Ox's documentation.

Cursor rules

[Rules](#) provide guidance to the LLMs, either by adding the content of the rule as context for each request, by having the models request the content of a rule, or by explicitly mentioning it in the request.

Bottom right corner:

latest ▾

PL

Search

Main.scala — avocado

```
backend > src > main > scala > com > softwaremill > avocado > Main.scala > {} com.softwaremill.avocado > {} Main > run
1 package com.softwaremill.avocado
2
3 import com.softwaremill.avocado.logging.Logging
4 import ox.logback.InheritableMDC
5 import ox.{Ox, OxApp, discard, never}
6 import org.slf4j.bridge.SLF4JBridgeHandler
7 import ox.OxApp.Settings
8 import ox.otel.context.PropagatingVirtualThreadFactory
9
10 object Main extends OxApp.Simple with Logging:
11   // route JUL to SLF4J (JUL is used by Magnum & OTEL for logging)
12   SLF4JBridgeHandler.removeHandlersForRootLogger()
13   SLF4JBridgeHandler.install()
14
15   // https://ox.softwaremill.com/latest/integrations/mdc-logback.html
16   InheritableMDC.init
17
18 Thread.setDefaultUncaughtExceptionHandler((t, e) => logger.error("Uncaught exception in thread: " + t, e))
19
20 FridgeApi.scala ~/projects/temp/avocado/backend/src/main-scala/com/softwaremill/avocado/fridge ·
21 override protected def settings: Settings = Settings.Default.copy(threadFactory = Some(PropagatingVirtualThreadFactory()))
22
23 run
24
25 override def run(using Ox): Unit =
26   val deps = Dependencies.create
27
28   deps.emailService.startProcesses()
29   deps.httpApi.start().discard
30   logger.info(s"Avocado started")
31
32   // blocking until the application is shut down
33   never
34 end run
end Main
```

Fridge management feature implementation

I need to fix the FridgeService to accept Id[User] instead of Id[ApiKey], and then update the API to handle string-based filtering. Let me first fix the service:

The string to replace was not found in the file.

Thought for 1s

FridgeService.scala +1 -1 -2 ·

```
import com.softwaremill.avocado.infrastructure.Magnum.DbTx
import com.softwaremill.avocado.security.ApiKey
import com.softwaremill.avocado.user.User
import com.softwaremill.avocado.util.Strings.Id
```

2 linter errors

Thought for 1s

FridgeService.scala +4 -6 -10 ·

```
def updateItem(userId: Id[ApiKey], itemId: Id[FridgeItem])
def deleteItem(userId: Id[ApiKey], itemId: Id[FridgeItem])
def addItem(userId: Id[User], name: String, quantity: Int)
def getItems(userId: Id[User], filter: FridgeApi.ExpirationFilter)
def updateItem(userId: Id[User], itemId: Id[FridgeItem], quantity: Int)
def deleteItem(userId: Id[User], itemId: Id[FridgeItem])(implicit dbTx: DbTx)
```

10 linter errors

Thought for 1s

FridgeService.scala +1 -1 -1 ·

```
override def addItem(userId: Id[ApiKey], name: String, quantity: Int)
override def addItem(userId: Id[User], name: String, quantity: Int)
if name.trim.isEmpty then Left(Fail.IncorrectInput("Item name cannot be empty")) else Right(Unit)
```

1 linter error

Generating

Add Context

Agent list

4-sonnet

...

22:19 / 57:21 · Generating the ba... >

Subscribe

33

Share

Save

...

Type-safe vibe-coding a CRUD application



SoftwareMill
3.44K subscribers



YouTube

In summary

Comparing Ox to functional effect systems

The better	The same	The worse
🍀 Casual FP	⭐ Fearless concurrency	💔 Pure FP
🍀 Simple syntax	⭐ Streaming	💔 Principled error handling
🍀 Lower learning curve	⭐ Structured concurrency	💔 Principled interruptions
🍀 Exceptions retain context		💔 Dedicated resource type
🍀 No virality		💔 Uniform computations
🍀 Built-in control flow		💔 Fearless refactoring

Comparing Ox & Gears

Ox	Gears
Ships now	Coupled to research on Caprese
No capture checking	Leverages capture checking
Only JVM 21+	JVM, Native, JS (?)
Concurrency, resiliency, streaming	Only concurrency
$T, \Rightarrow T$	Future [T]
Let it crash	Failed futures
Error handling using Eithers	Do be determined
Ox capability	Async, Async.Spawn capabilities

What's next?

- More streaming integrations!
 - what's after Kafka?
- Non-pinning file I/O?
 - io_uring
- Wrapped concurrency primitives?
- Own boundary-break?



[GitHub](#)

Ox in a nutshell

- Concurrency, resiliency, streaming, error handling for JVM 21+
- Makes direct-style feasible
- Blocking I/O
- A thin layer on top of Virtual Threads
- FP all the way up to, but excluding, computations as values
- OSS, Apache2 licensed



Accelerating product development



[SoftwareMill](#)

```
println("Thank you!")
```



warski.org