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?

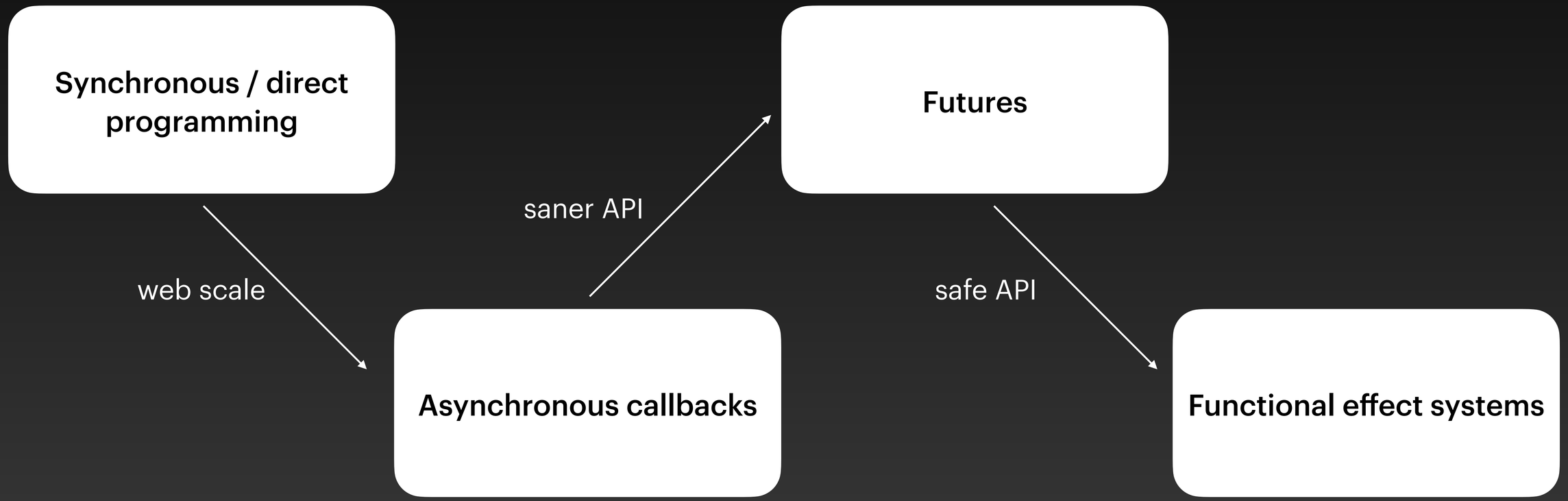
Why Scala? (YT)

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	

Lambda Days talk

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

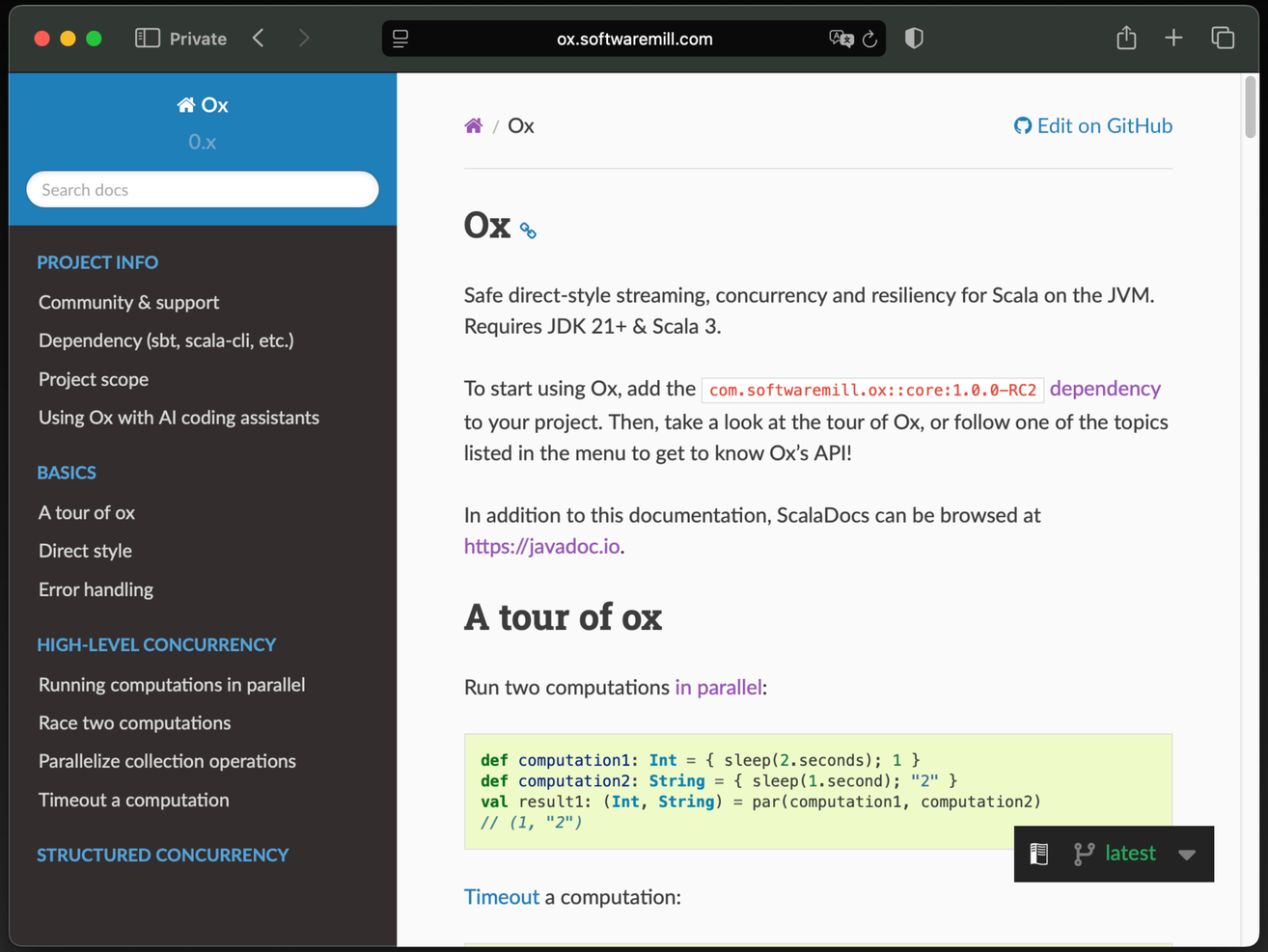
than fibers

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	

Unwrapping IO @ Scala.IO

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


Private
ox.softwaremill.com
Ox
0.x
Search docs
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
STRUCTURED CONCURRENCY
Ox
Edit on GitHub
Ox
Safe direct-style streaming, concurrency and resiliency for Scala on the JVM. Requires JDK 21+ & Scala 3.
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!
In addition to this documentation, ScalaDocs can be browsed at https://javadoc.io.
A tour of ox
Run two computations in parallel:
def computation1: Int = { sleep(2.seconds); 1 }
def computation2: String = { sleep(1.second); "2" }
val result1: (Int, String) = par(computation1, computation2)
// (1, "2")
latest
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 ...
```

```scala
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.usingEmit: 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))
```

```scala
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)
```

```scala
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!")
```

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

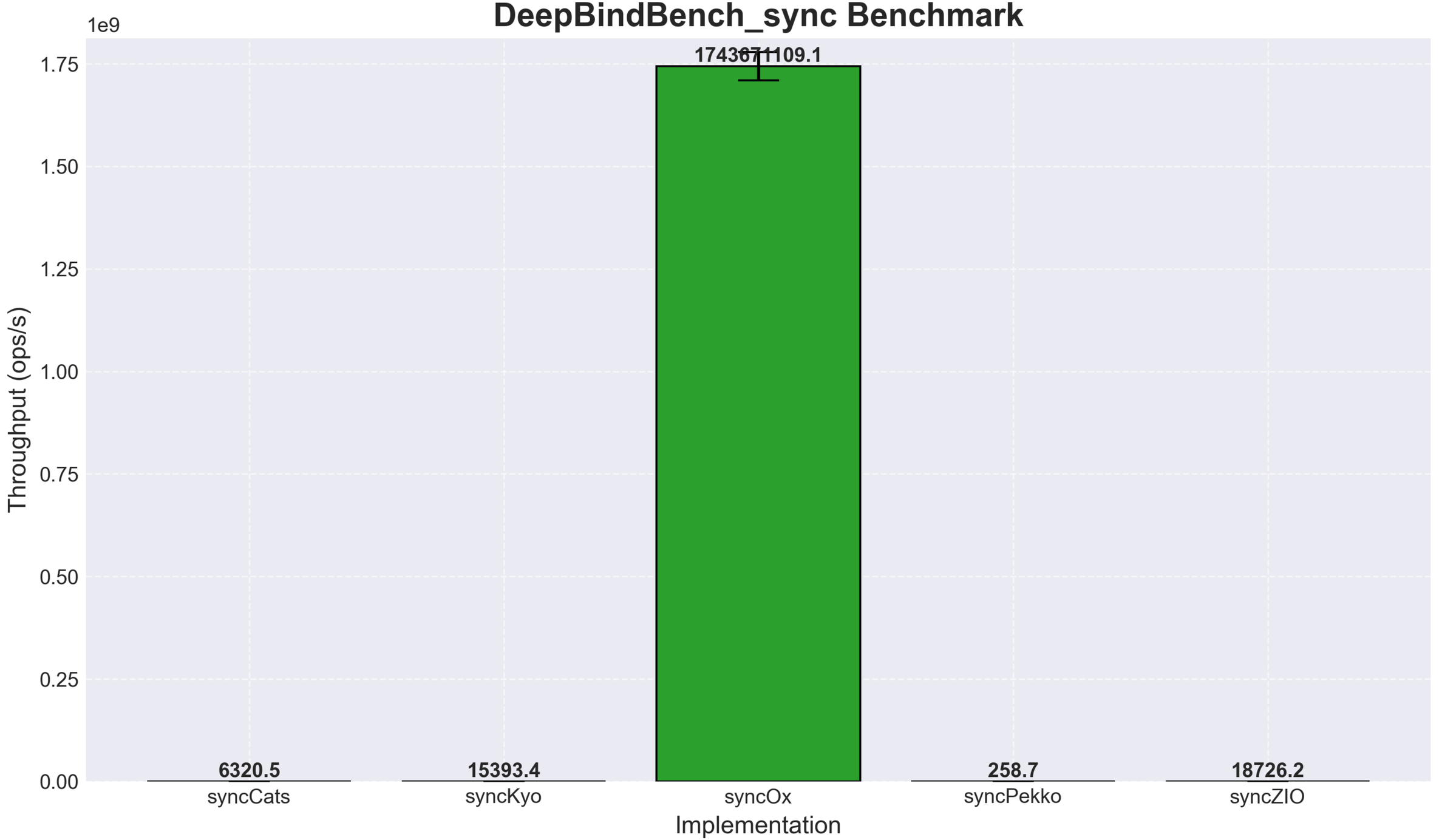
Performance

Benchmarks use the harness from the Kyo library:

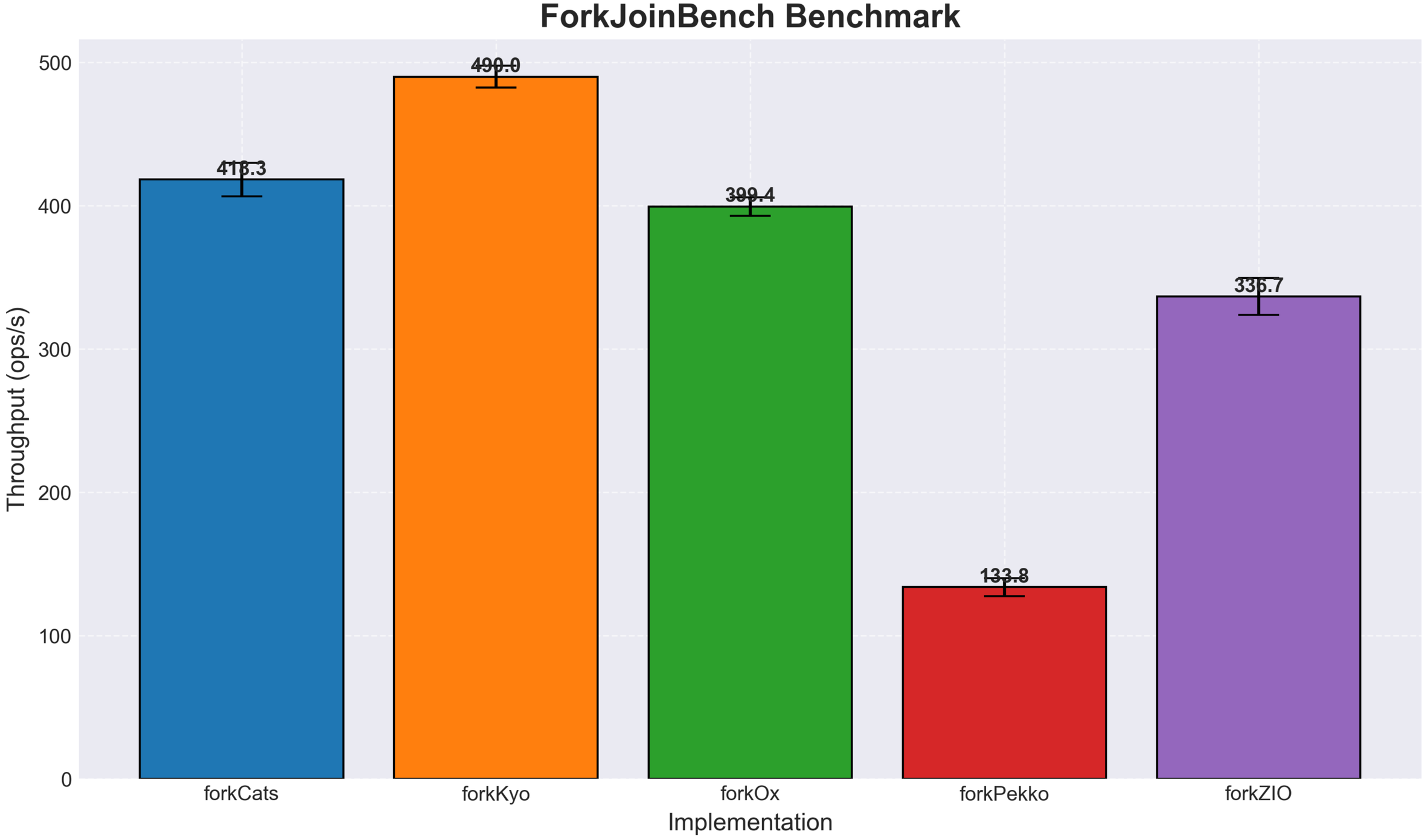
- streamlined algebraic effects
- also approaching 1.0
- peak performance
- functional effect system
- ran using Java 24, MacBook M1 Max

Kyo docs

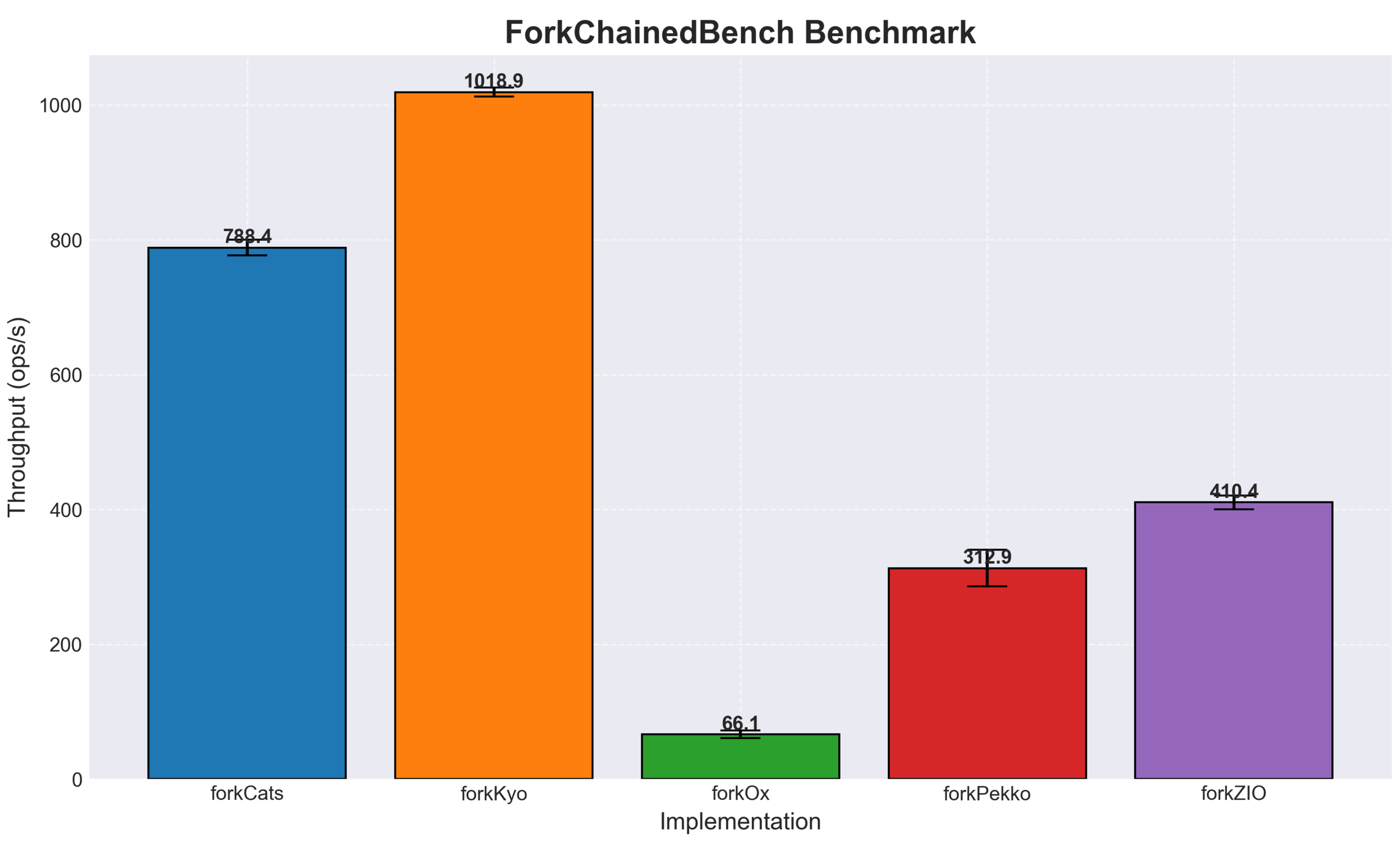
```
// DeepBindBench

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
1e9
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
Throughput (ops/s)
1743671109.1
6320.5
15393.4
258.7
18726.2
syncCats
syncKyo
syncOx
syncPekko
syncZIO
Implementation

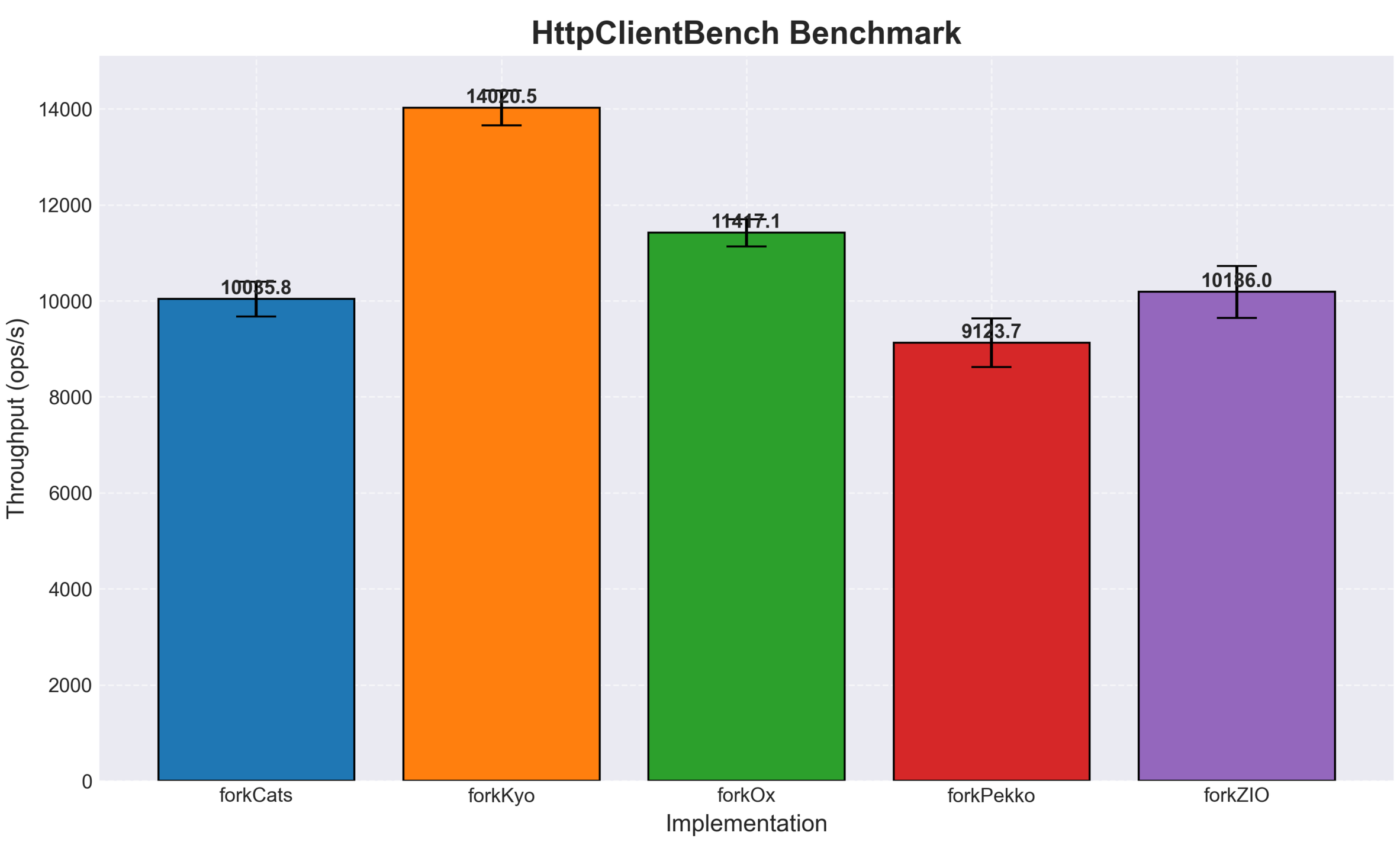
```scala
// ForkJoinBench
val range = (0 until 10000).toList

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
Throughput (ops/s)
0
100
200
300
400
500
418.3
490.0
399.4
133.8
336.7
forkCats
forkKyo
forkOx
forkPekko
forkZIO
Implementation

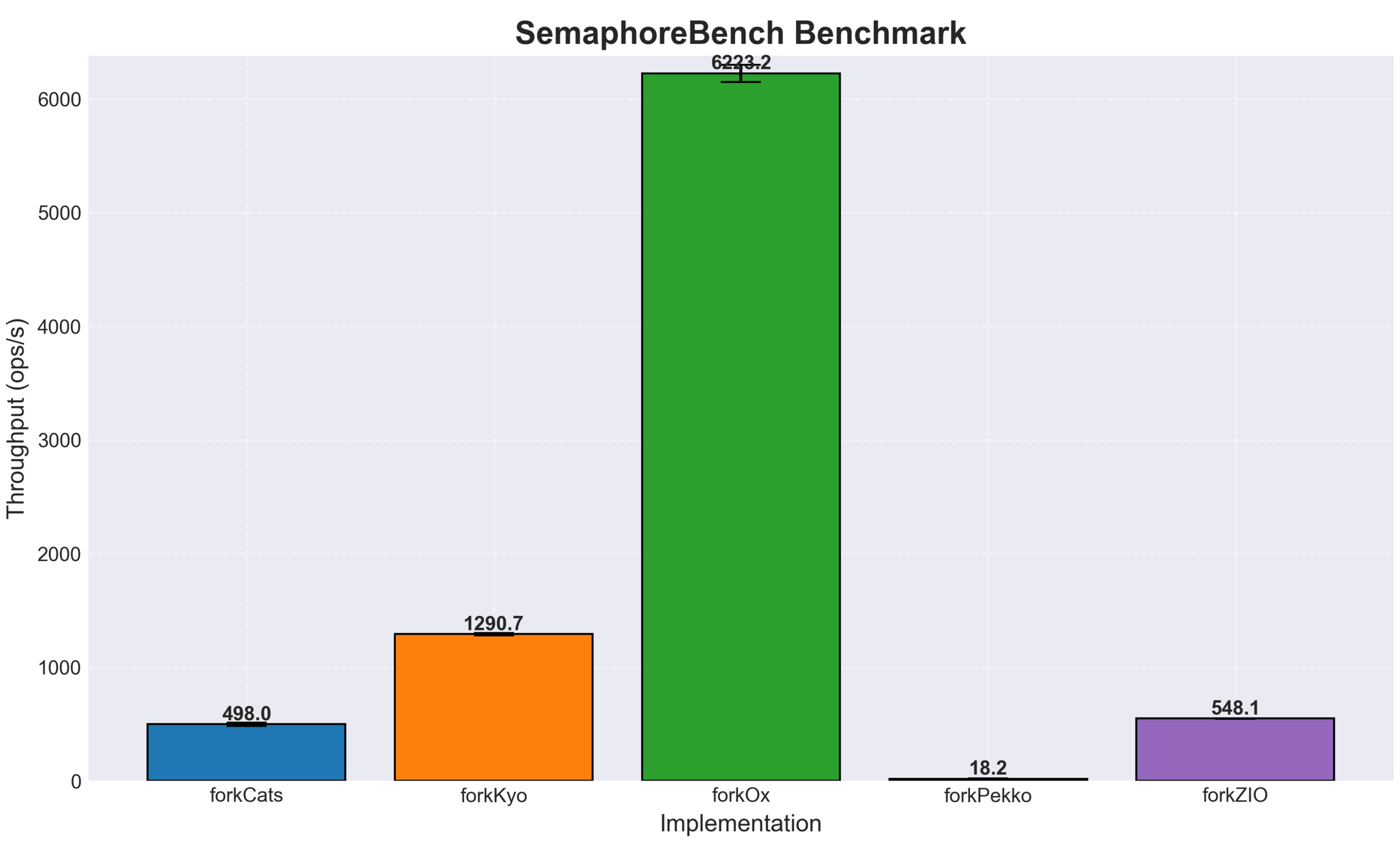
```
// ForkChainedBench
def catsBench() =
  def iterate(deferred: Deferred[IO, Unit], n: Int): IO[Any] =
    if n <= 0 then deferred.complete(())
    else IO.unit.flatMap(_ => iterate(deferred, n - 1).start)

  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
Throughput (ops/s)
0
200
400
600
800
1000
788.4
1018.9
66.1
312.9
410.4
forkCats
forkKyo
forkOx
forkPekko
forkZIO
Implementation

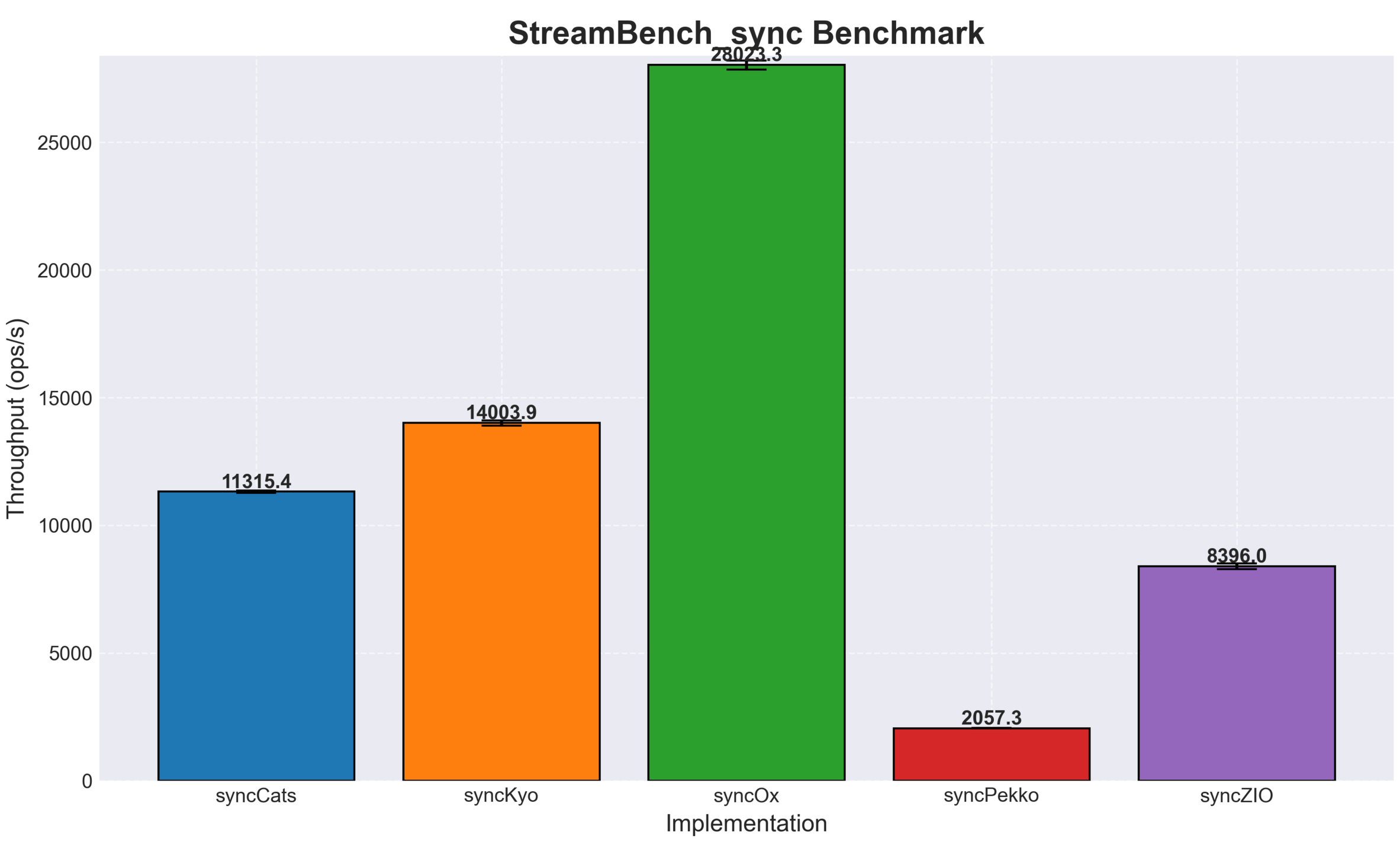
```scala
// HttpClientBench
lazy val catsClient =
  given LoggerFactory[IO] = Slf4jFactory.create[IO]
  EmberClientBuilder.default[IO].build.allocated.unsafeRunSync()._1

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
Throughput (ops/s)
0
2000
4000
6000
8000
10000
12000
14000
10035.8
14020.5
11417.1
9123.7
10186.0
forkCats
forkKyo
forkOx
forkPekko
forkZIO
Implementation

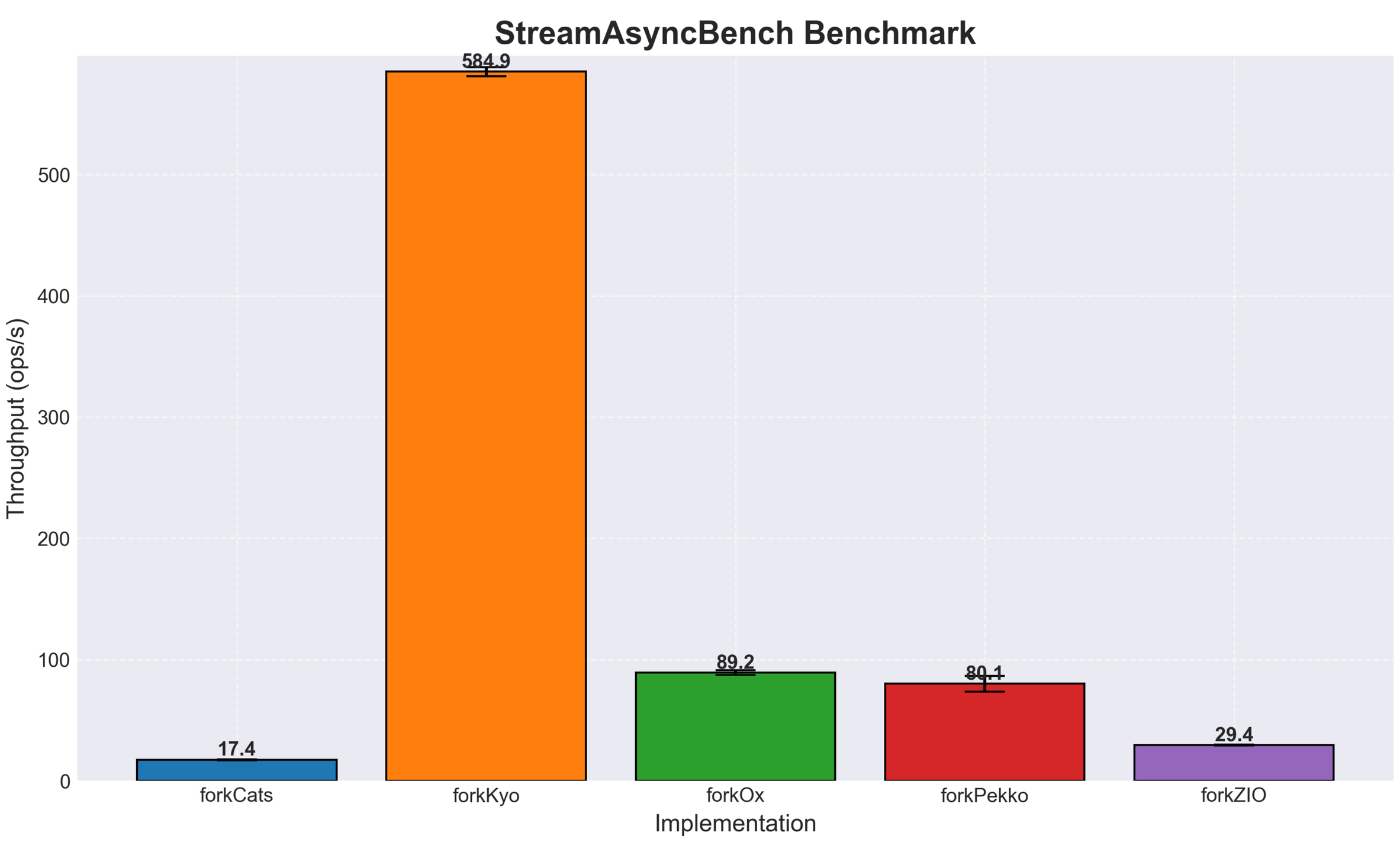
```
// SemaphoreBench
def catsBench() =
  def loop(s: Semaphore[IO], i: Int): IO[Unit] =
    if i >= depth then
      IO.unit
    else
      s.acquire.flatMap(_ => s.release).flatMap(_ => loop(s, i + 1))

  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)
```

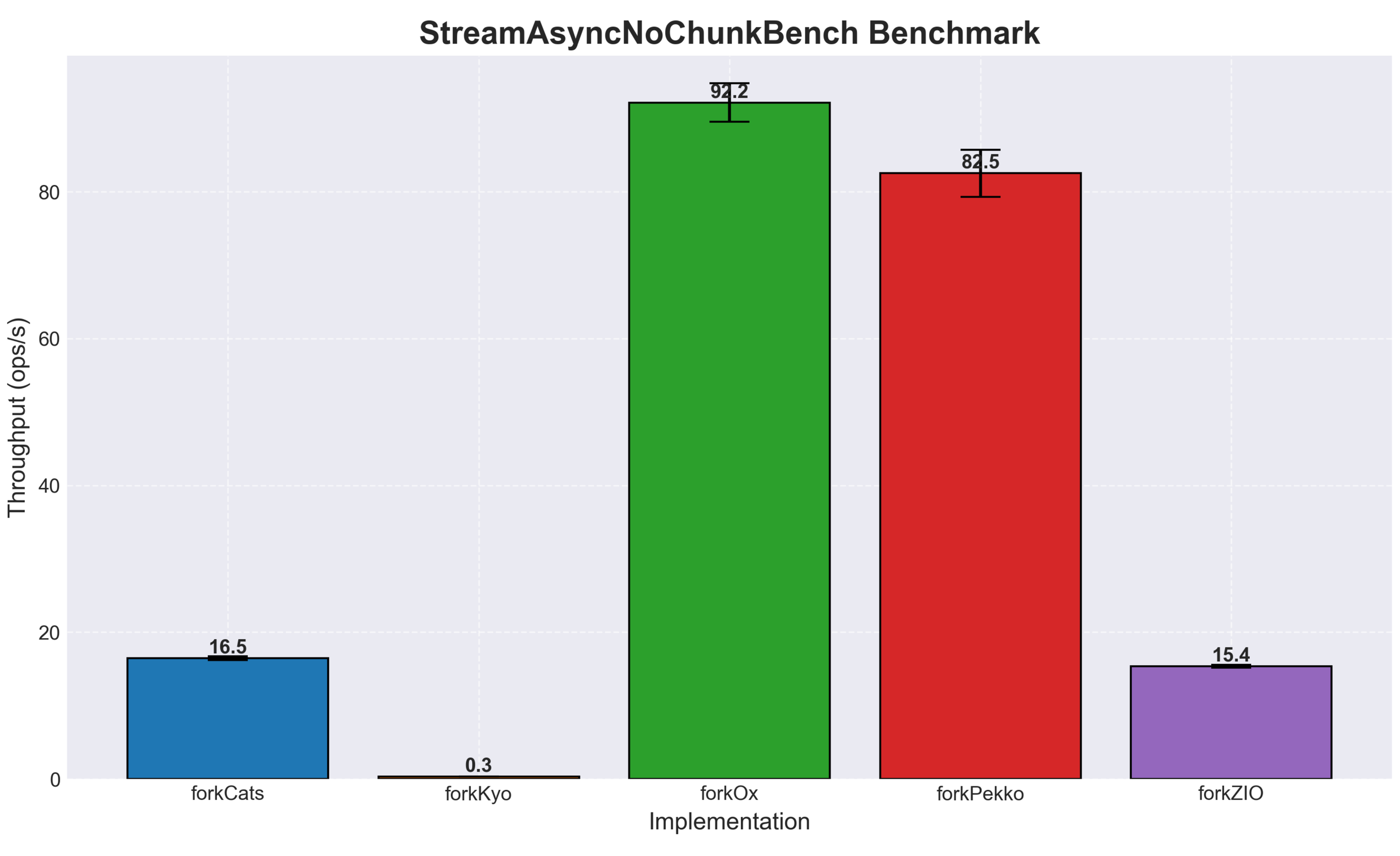

SemaphoreBench Benchmark
Throughput (ops/s)
0
1000
2000
3000
4000
5000
6000
498.0
1290.7
6223.2
18.2
548.1
forkCats
forkKyo
forkOx
forkPekko
forkZIO
Implementation

```scala
// 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
Throughput (ops/s)
0
5000
10000
15000
20000
25000
11315.4
14003.9
28023.3
2057.3
8396.0
syncCats
syncKyo
syncOx
syncPekko
syncZIO
Implementation

```
// 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()
```

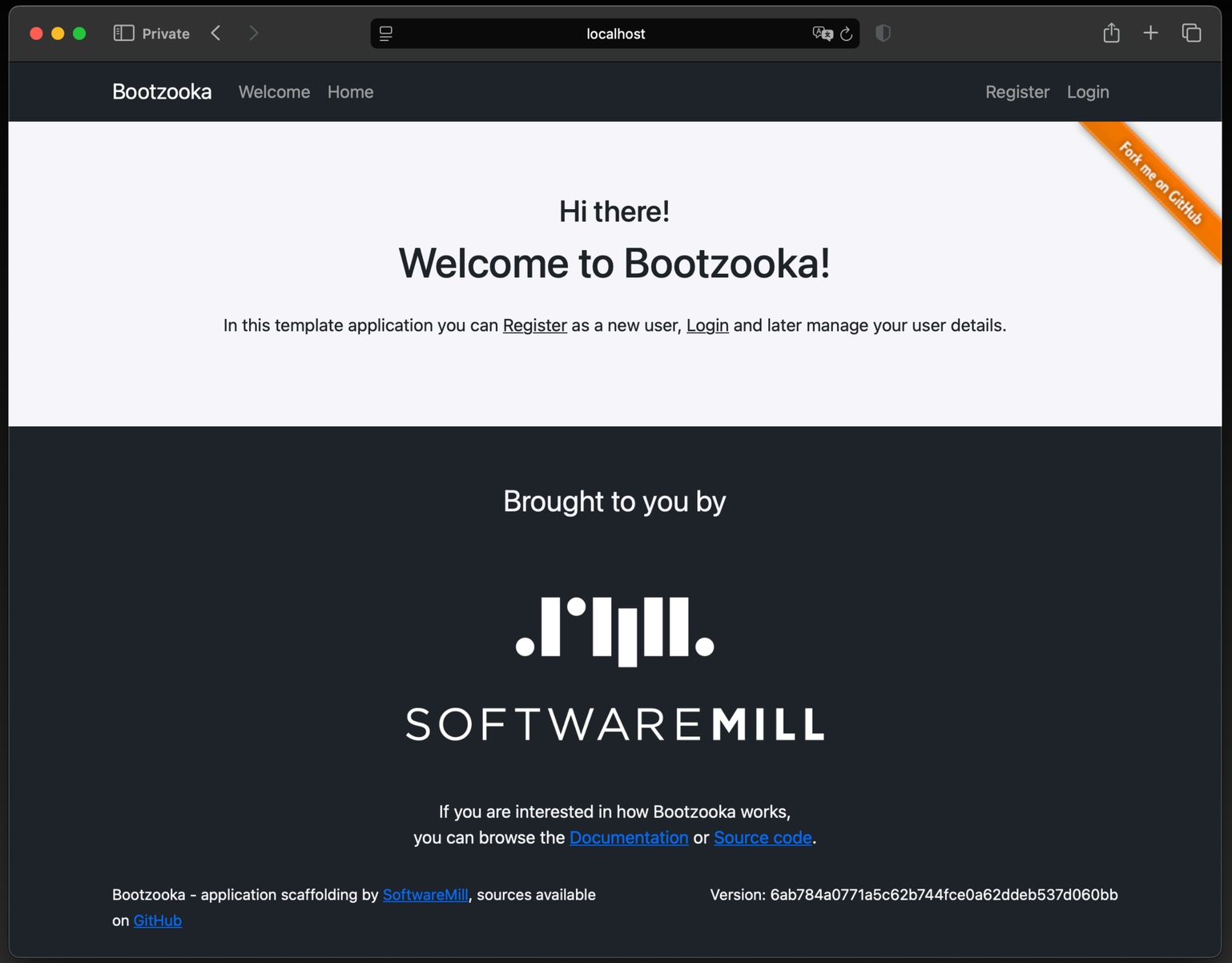

StreamAsyncBench Benchmark
Throughput (ops/s)
0
100
200
300
400
500
17.4
584.9
89.2
80.1
29.4
forkCats
forkKyo
forkOx
forkPekko
forkZIO
Implementation


StreamAsyncNoChunkBench Benchmark
Throughput (ops/s)
0
20
40
60
80
16.5
0.3
92.2
82.5
15.4
forkCats
forkKyo
forkOx
forkPekko
forkZIO
Implementation

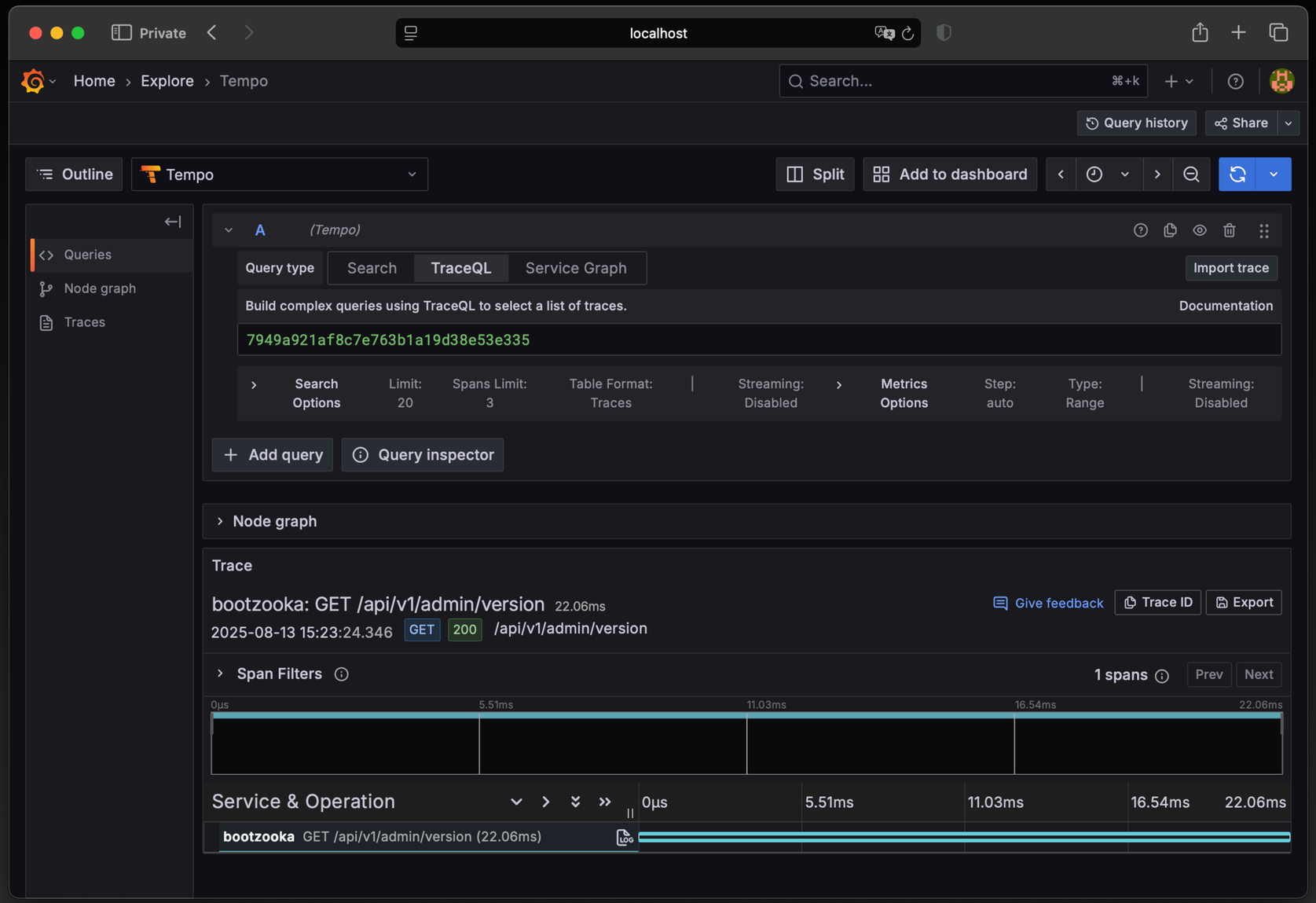
Ecosystem

```scala
@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}")
```

```scala
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()
```

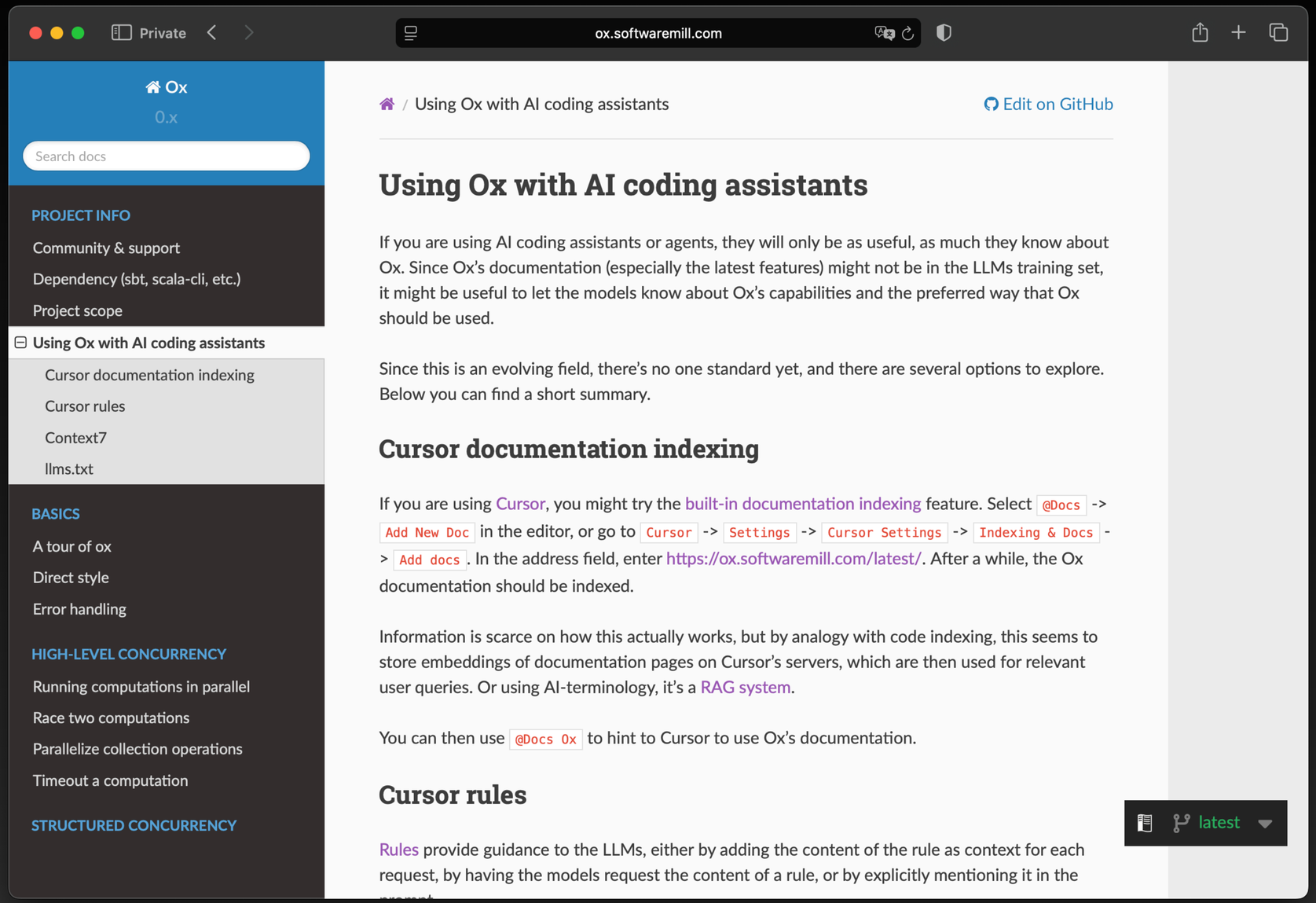


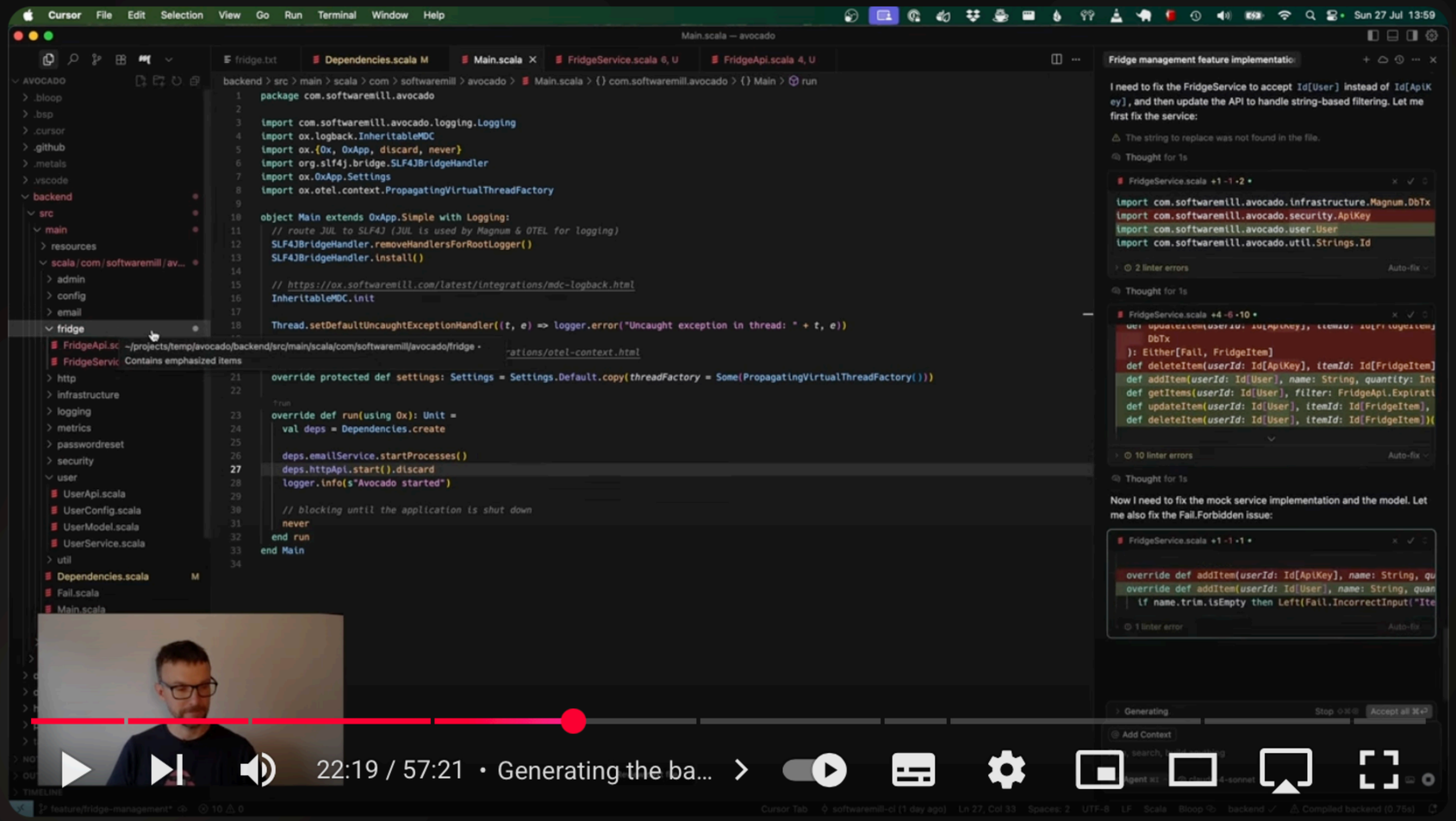
Bootzooka docs


Private
localhost
Home › Explore › Tempo
Search... ⌘+k
Query history
Share
Outline
Tempo
Split
Add to dashboard
Queries
Node graph
Traces
A (Tempo)
Query type
Search
TraceQL
Service Graph
Import trace
Build complex queries using TraceQL to select a list of traces.
Documentation
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
2025-08-13 15:23:24.346 GET 200 /api/v1/admin/version
Give feedback
Trace ID
Export
Span Filters
1 spans
Prev
Next
0µs
5.51ms
11.03ms
16.54ms
22.06ms
Service & Operation
bootzooka GET /api/v1/admin/version (22.06ms)

Plus, any blocking Scala or Java API

AI


Private
ox.softwaremill.com
Ox
0.x
Search docs
PROJECT INFO
Community & support
Dependency (sbt, scala-cli, etc.)
Project scope
Using Ox with AI coding assistants
Cursor documentation indexing
Cursor rules
Context7
llms.txt
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
STRUCTURED CONCURRENCY
Using Ox with AI coding assistants
Edit on GitHub
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
latest



Type-safe vibe-coding a CRUD application

SoftwareMill

3.44K subscribers

Subscribe

33

Share

Save

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 reasearch on Caprese
No capture checking	Leverages capture checking
Only JVM 21+	JVM, Native, JS (?)
Concurrency, resiliency, streaming	Only concurrency
`T, =>T`	`Future[T]`
Let it crash	Failed futures
Error handling using `Either`s	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

SOFTWAREMILL

Accelerating product development

SoftwareMill

println("Thank you!")

warski.org