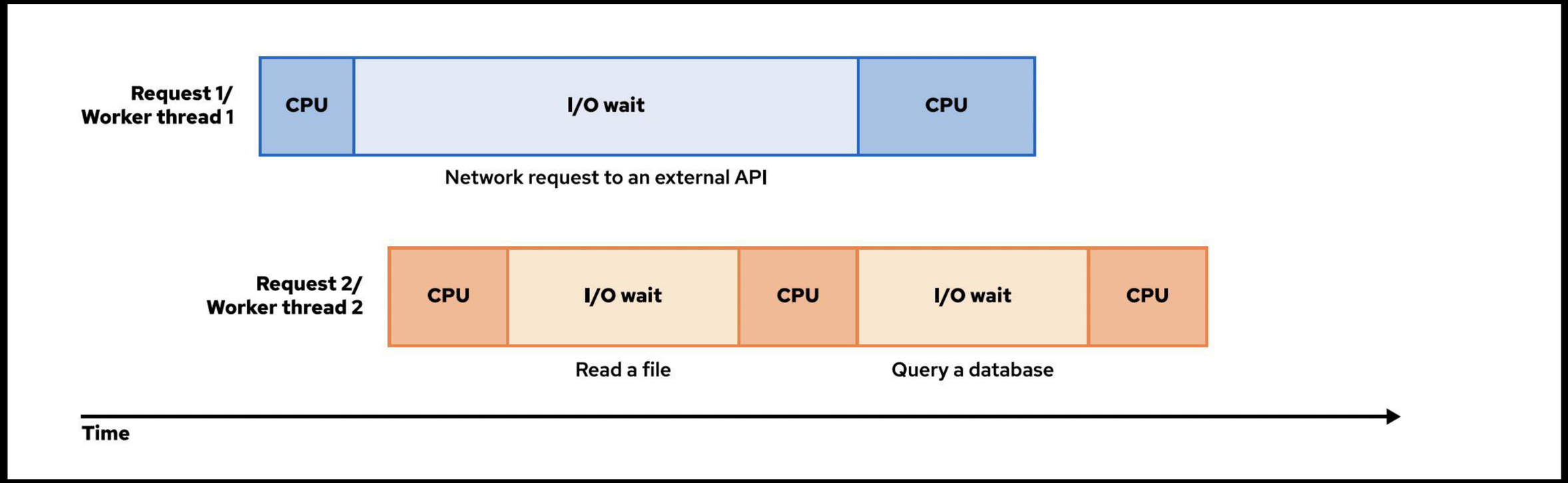
Desarrollando microservicios

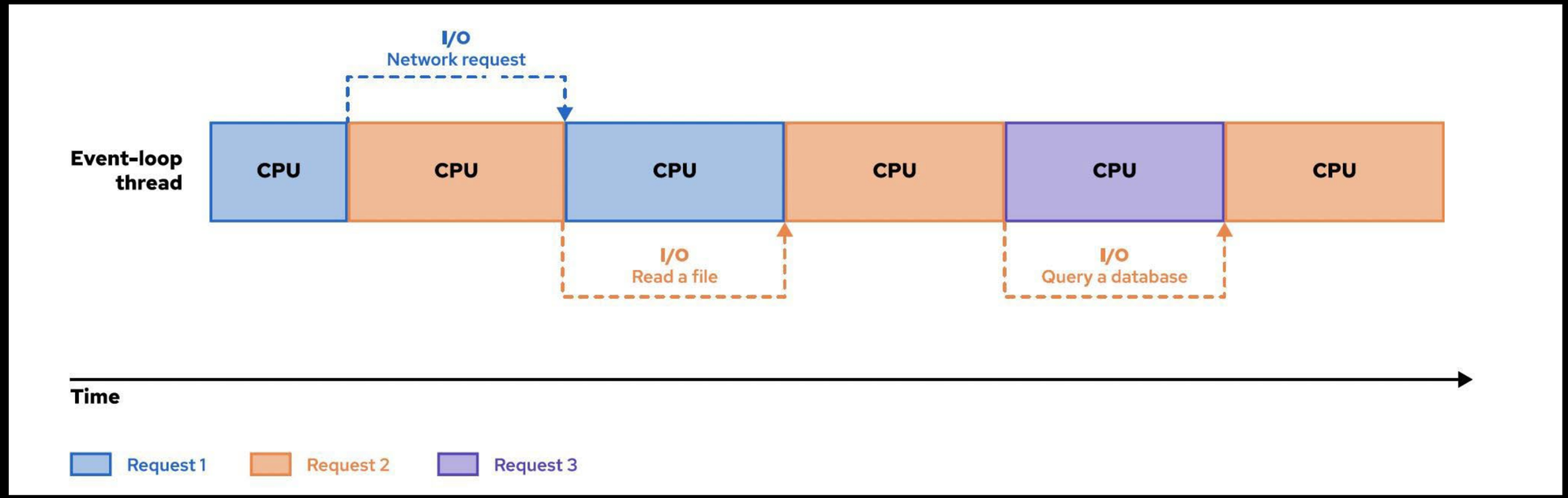
Reactivos y Asíncronos

JoeDayz | 2025

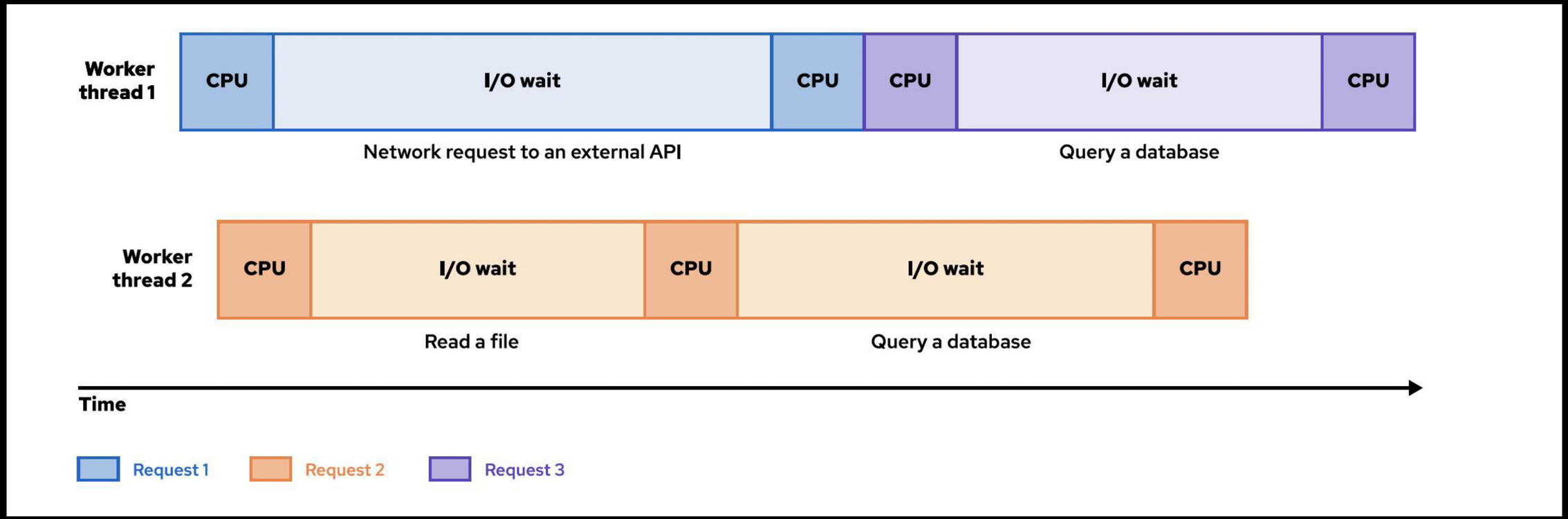
Blocking I/O


Request 1/
Worker thread 1
CPU
I/O wait
CPU
Network request to an external API
Request 2/
Worker thread 2
CPU
I/O wait
CPU
I/O wait
CPU
Read a file
Query a database
Time

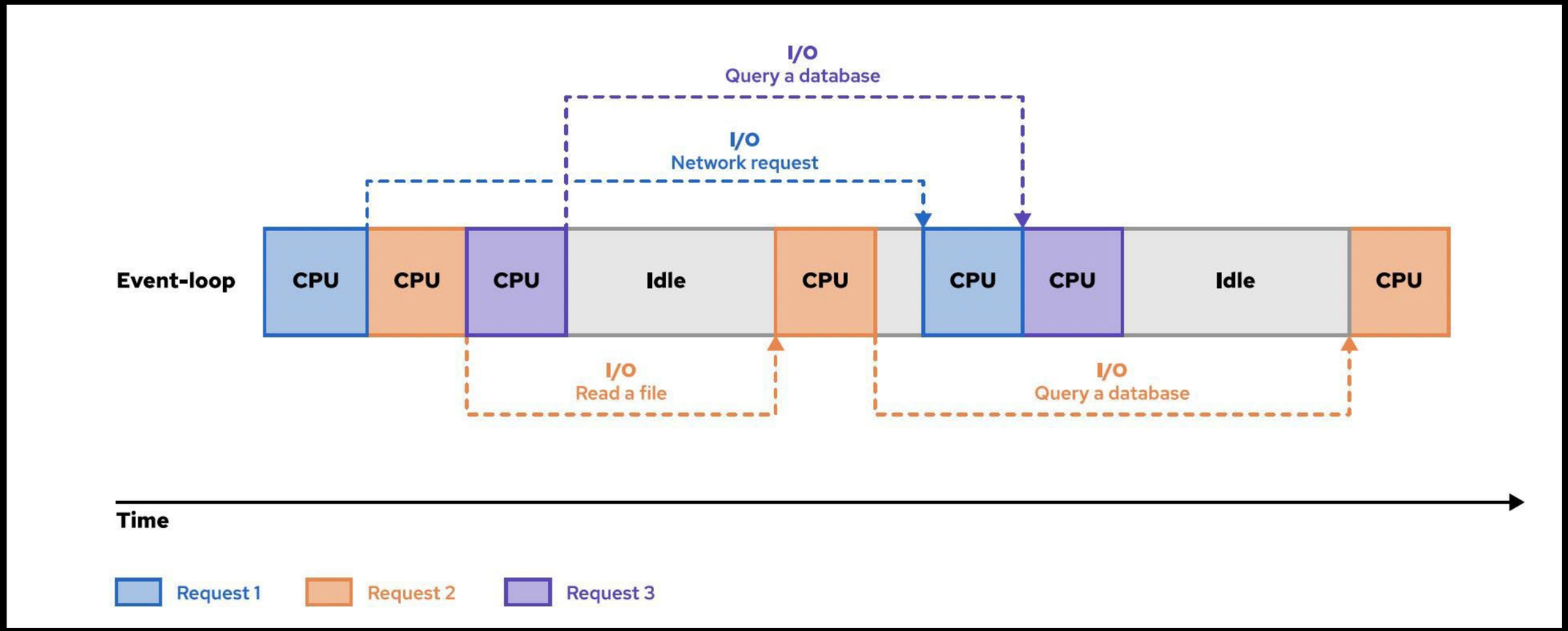
NO Blocking I/O



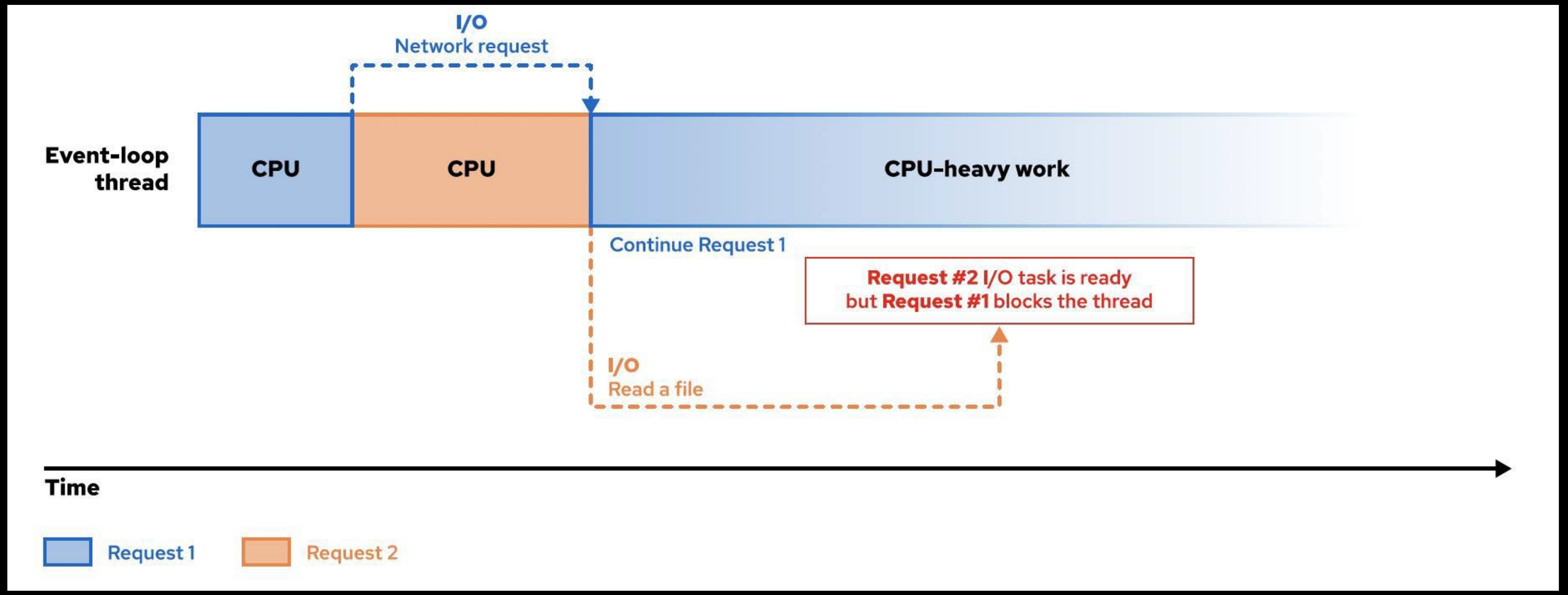
2 Worker Threads atendiendo 3 Requests



2 Event Loop Threads atendiendo 3 Requests



No bloquees el Event Loop



Extensiones Reactivas

AHORA:

```xml
<dependency>
    <groupId>io.quarkus</groupId>
    <artifactId>quarkus-rest</artifactId>
</dependency>
<dependency>
    <groupId>io.quarkus</groupId>
    <artifactId>quarkus-rest-jackson</artifactId>
</dependency>
```

ANTES:

```xml
<dependency>
    <groupId>io.quarkus</groupId>
    <artifactId>quarkus-resteasy-reactive</artifactId>
</dependency>
```

```xml
<dependency>
    <groupId>io.quarkus</groupId>
    <artifactId>quarkus-resteasy-reactive-jackson</artifactId>
</dependency>
```

REST reactivo

Manejar requests REST como tareas no bloqueantes

Métodos Bloqueantes

- Se ejecuta en worker threads. Usa esta modalidad cuando tu operación es intensiva en uso de CPU.
- Con esto se evita romper el event loop de Vert.x.

Métodos No Bloqueantes

- Estos usan el event loop de Vert.x.
- Este deberías usar para operaciones no bloqueantes o asíncronas.

REST reactivo

¿Cómo se da cuenta que debe aplicar NO Blocking?

Si tu método retorna estos tipos, se considerará una operación NO BLOQUEANTE:

- `io.smallrye.mutiny.Uni`
- `io.smallrye.mutiny.Multi`
- `java.util.concurrent.CompletionStage`
- `org.reactivestreams.Publisher`

Característica	Mutiny (Quarkus)	Reactor (WebFlux)
Resultado único	`Uni<T>`	`Mono<T>`
Flujo de múltiples valores	`Multi<T>`	`Flux<T>`
Basado en	API propia (Mutiny)	Project Reactor
Estilo	Declarativo, fluido y orientado a simplicidad	Declarativo, verboso pero muy poderoso
Cancelación	Nativa y simple (`subscription.cancel()`)	Soportada pero más verbosa
Integración con Java	Usado por defecto en Quarkus	Usado en Spring WebFlux
Filosofía	"Events → Uni/Multi" (API más humana, menos técnica)	"Publisher/Subscriber" (más pegado a Reactive Streams spec)

¿Quarkus cómo lo ejecuta: blocking o no blocking?

```
@GET
@Path( "/cart" )
public ShoppingCart getCart() {
   return storage.findCart();
}
```

```
INFO  [io.qua.htt.access-log] (executor-thread-X) ...
```

Aquí tenemos un endpoint no blocking

```
@GET
@Path( "/cart" )
public Uni<ShoppingCart> getCart() {
   Uni<ShoppingCart> cartStream = storage.findCartAsync();
   return cartStream;
}
```

```
INFO [io.qua.htt.access-log] (vert.x-eventloop-thread-X) ...
```

Usando anotaciones Bloqueantes

io.smallrye.common.annotation.Blocking

- Indica que el método es bloqueante. REST ejecuta el método en un

worker thread.

io.smallrye.common.annotation.NonBlocking

- Indica que el método es no bloqueante. REST ejecuta el método en un Event loop thread (usa Vert.x).

Esto demora minutos…

```
@GET
@Path( "/recommendations" )
public Uni<Set<Recommendations>> getRecommendations() {
  // This operation is computationally expensive
  computeRecommendations();

  Uni<Set<Recommendations>> recommendations = fetchRecommendations();

  return recommendations;
}
```

Note

If Vert.x detects that you are blocking the event-loop, then it throws an exception.

```
@GET
@Path( "/recommendations" )
@Blocking
public Uni<Set<Recommendations>> getRecommendations() {
   ...implementation omitted...
}
```

Extensiones Reactivas

```xml
<dependency>
    <groupId>io.quarkus</groupId>
    <artifactId>quarkus-rest-client</artifactId>
</dependency>

<dependency>
    <groupId>io.quarkus</groupId>
    <artifactId>quarkus-rest-client-jackson</artifactId>
</dependency>
```

Ejemplo:

```
@GET
@Path( "/users/{userId}" )
User getUser( @PathParam( "userId" ) final Long userId );
```

```
@GET
@Path( "/users/{userId}" )
Uni<User> getUser( @PathParam( "userId" ) final Long userId );
```

Recursos

Reactividad

- https://www.reactivemanifesto.org/
- https://quarkus.io/version/2.13/guides/quarkus-reactive-architecture
- https://quarkus.io/version/2.13/guides/resteasy-reactive
- https://smallrye.io/smallrye-mutiny/1.7.0/