Downloading a Billion Files in Python

A case study in multi-threading, multi-processing, and asyncio

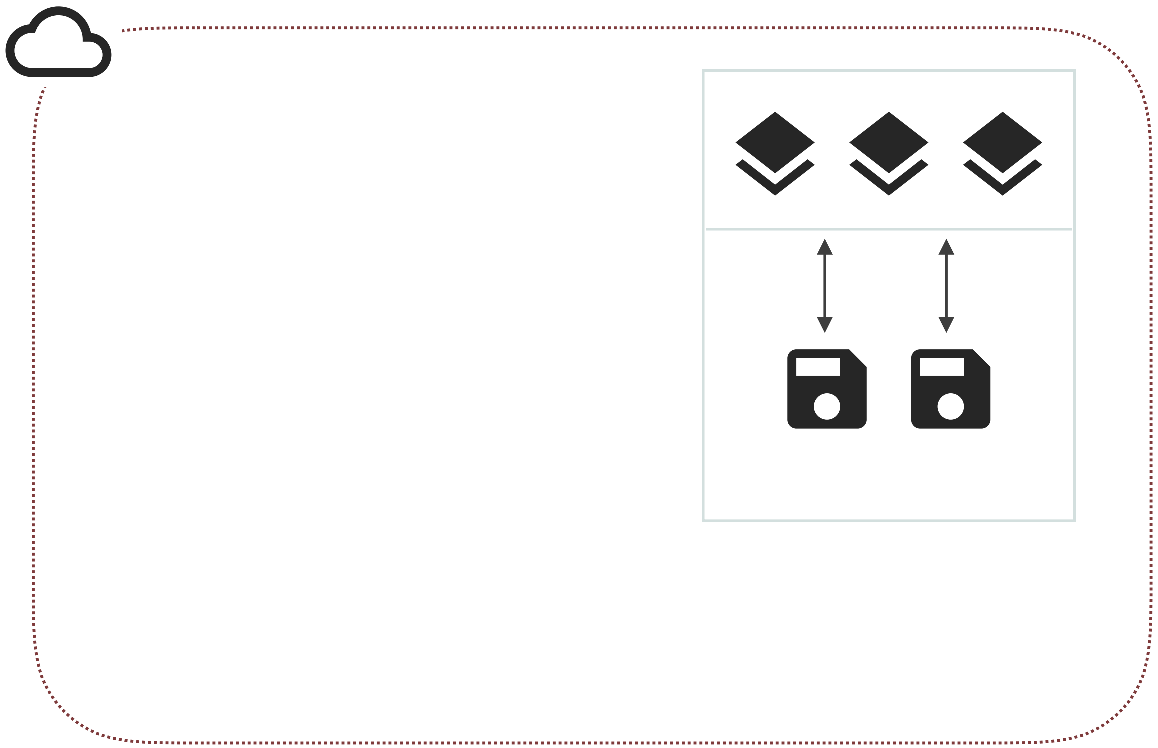
James Saryerwinnie

🐦 @jsaryer

Our Task

Our Task

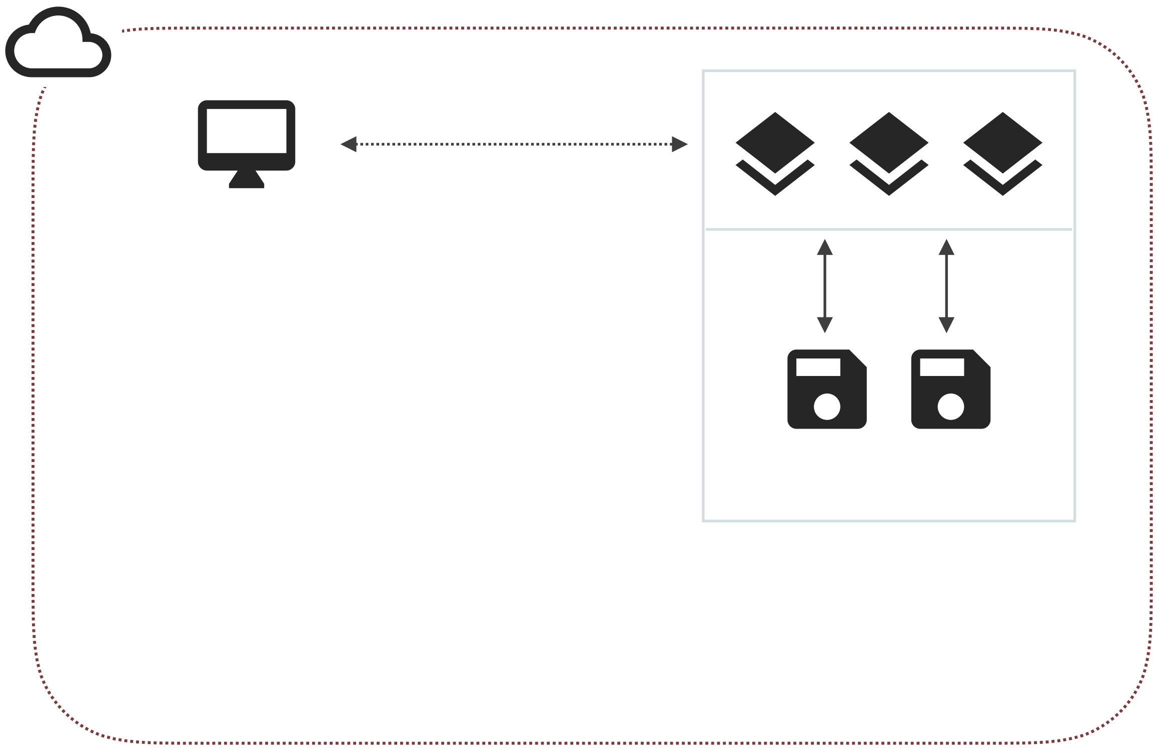
There is a remote server that stores files



Our Task

There is a remote server that stores files

The files can be accessed through a REST API

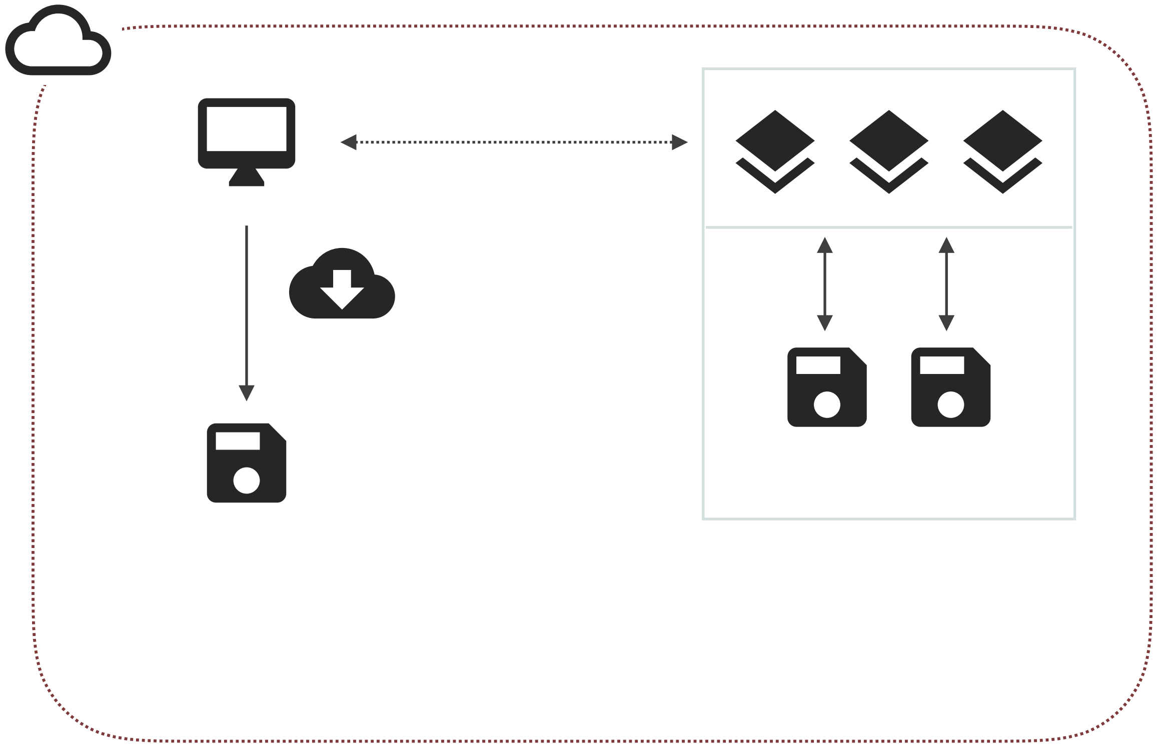


Our Task

There is a remote server that stores files

The files can be accessed through a REST API

Our task is to download all the files on the remote server to our client machine



Our Task (the details)

Our Task (the details)

What client machine will this run on?

Our Task (the details)

What client machine will this run on?

We have one machine we can use, 16 cores, 64GB memory

Our Task (the details)

What client machine will this run on?

We have one machine we can use, 16 cores, 64GB memory

What about the network between the client and server?

Our Task (the details)

What client machine will this run on?

We have one machine we can use, 16 cores, 64GB memory

What about the network between the client and server?

Our client machine is on the same network as the service with remote files

Our Task (the details)

What client machine will this run on?

We have one machine we can use, 16 cores, 64GB memory

What about the network between the client and server?

Our client machine is on the same network as the service with remote files

How many files are on the remote server?

Our Task (the details)

What client machine will this run on?

We have one machine we can use, 16 cores, 64GB memory

What about the network between the client and server?

Our client machine is on the same network as the service with remote files

How many files are on the remote server?

Approximately one billion files, 100 bytes per file

Our Task (the details)

What client machine will this run on?

We have one machine we can use, 16 cores, 64GB memory

What about the network between the client and server?

Our client machine is on the same network as the service with remote files

How many files are on the remote server?

Approximately one billion files, 100 bytes per file

When do you need this done?

Our Task (the details)

What client machine will this run on?

We have one machine we can use, 16 cores, 64GB memory

What about the network between the client and server?

Our client machine is on the same network as the service with remote files

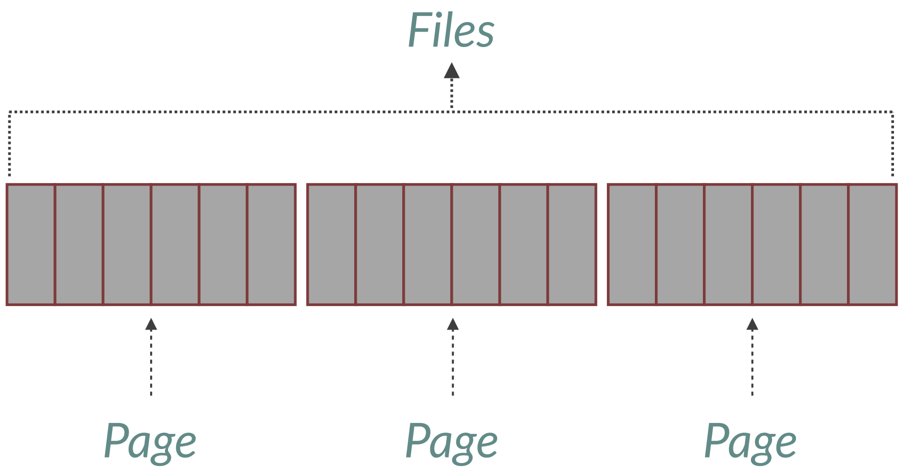
How many files are on the remote server?

Approximately one billion files, 100 bytes per file

When do you need this done?

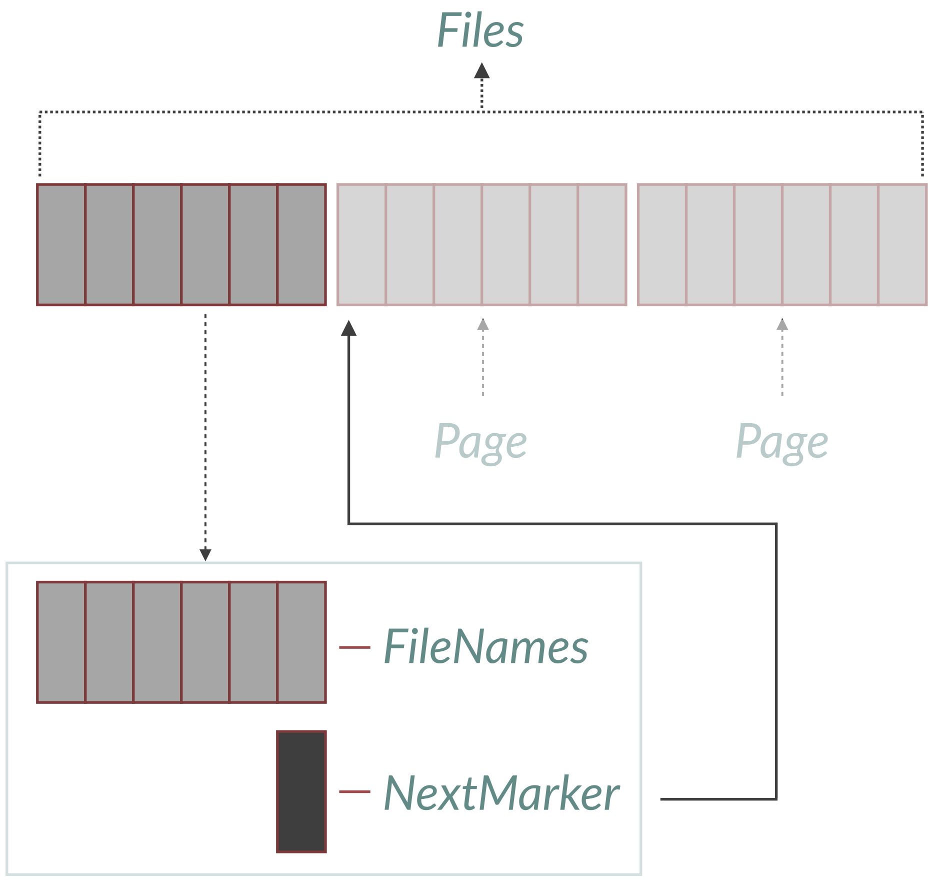
Please have this done as soon as possible

File Server Rest API



File Server Rest API

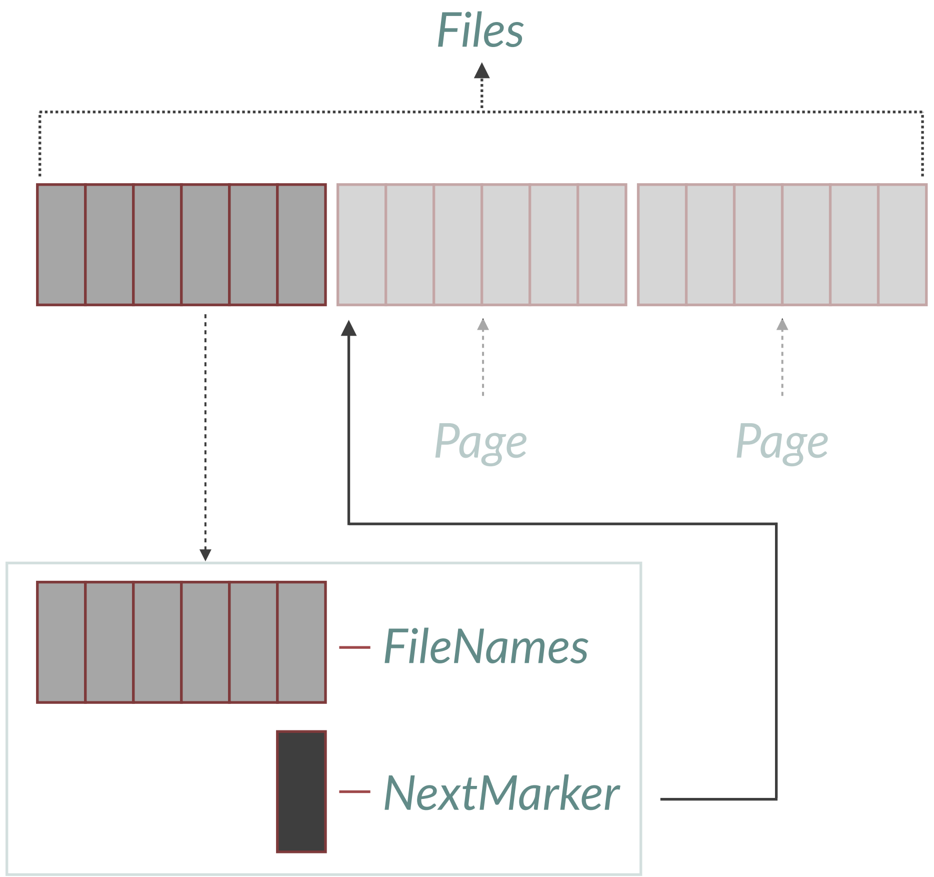
`GET /list`



File Server Rest API

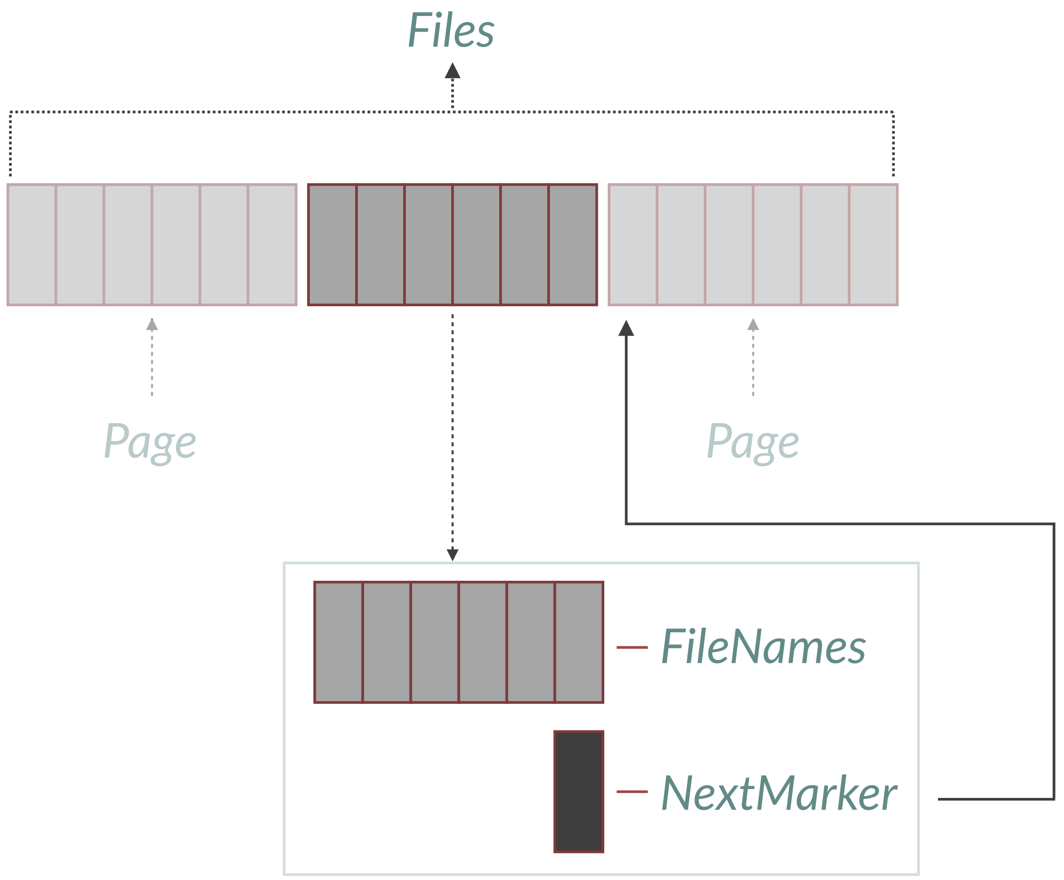
GET /list

```
{"FileNames": [
   "file1", "file2", ...],
 "NextMarker": "pagination-token"}
```



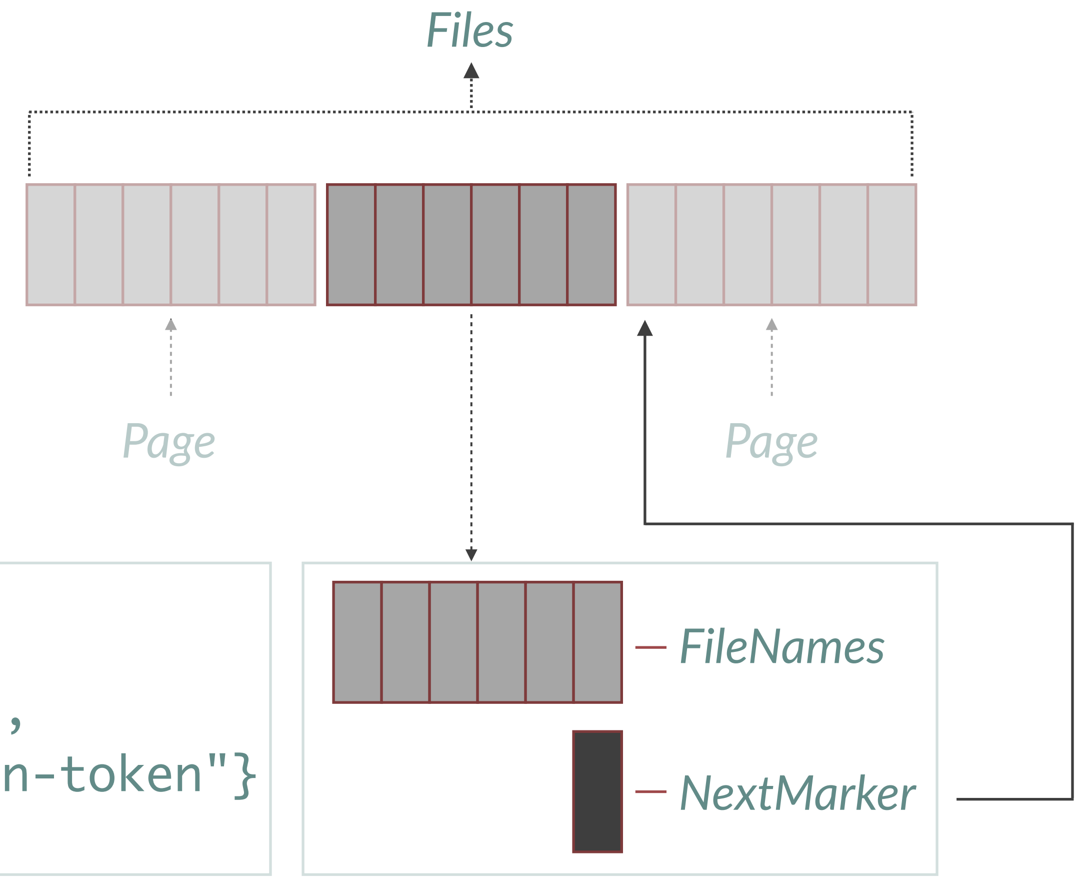
File Server Rest API

```
GET /list?next-marker=token
```



File Server Rest API

```
GET /list?next-marker=token
```



```
{"FileNames": [
    "file1", "file2", ...],
 "NextMarker": "pagination-token"}
```

File Server Rest API

GET /list	{"FileNames": ["file1", "file2", ...]}
GET /list?next-marker={token}	{"FileNames": ["file1", "file2", ...], "NextMarker": "pagination-token"}
GET /get/{filename}	(File blob content)

Caveats

This is a simplified case study.

The results shown here don't necessarily generalize.

Not an apples to apples comparison, each approach does things slightly different

Sometimes concrete examples can be helpful

Caveats

This is a simplified case study.

The results shown here don't necessarily generalize.

Not an apples to apples comparison, each approach does things slightly different

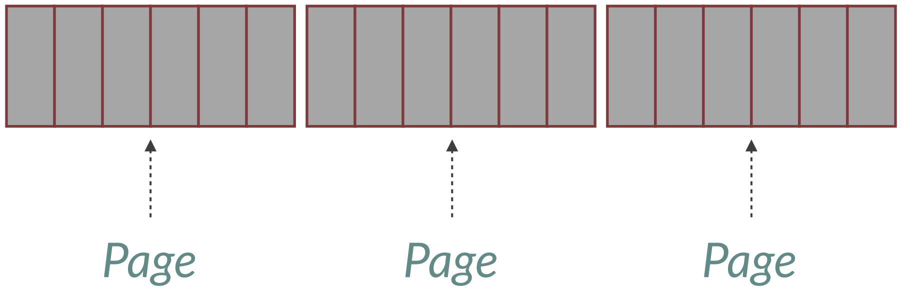
Sometimes concrete examples can be helpful

Always profile and test for yourself

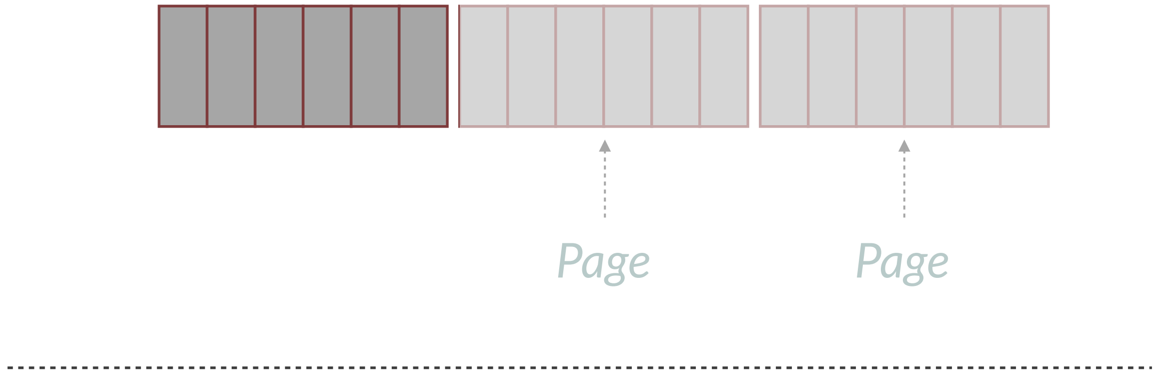
Synchronous Version

Simplest thing that could possibly work.

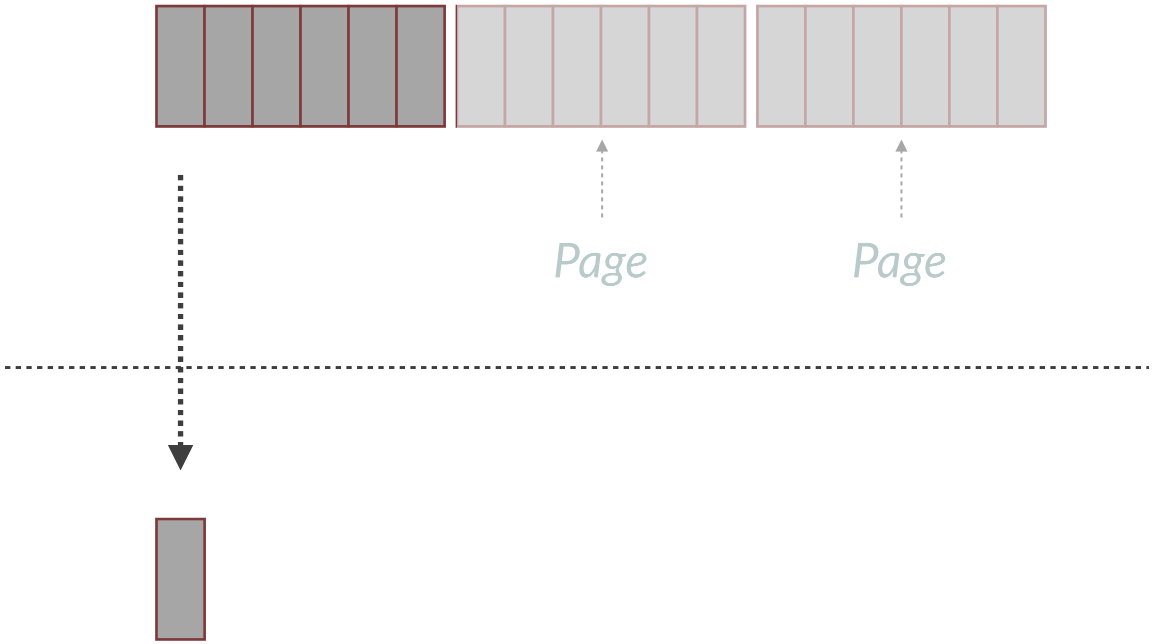
Synchronous



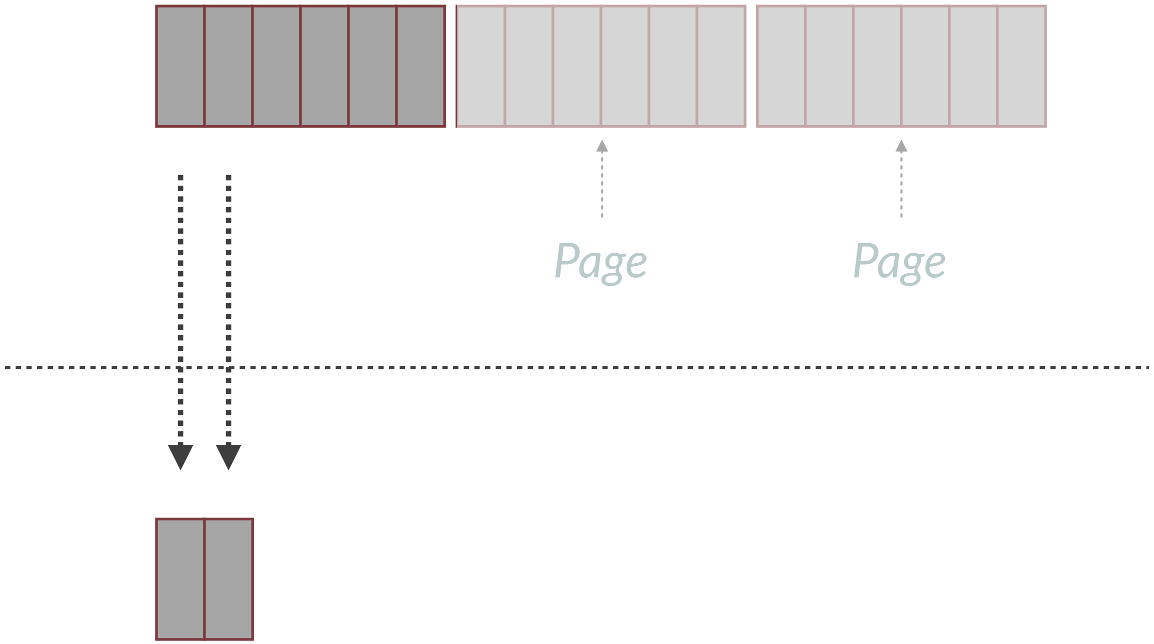
Synchronous



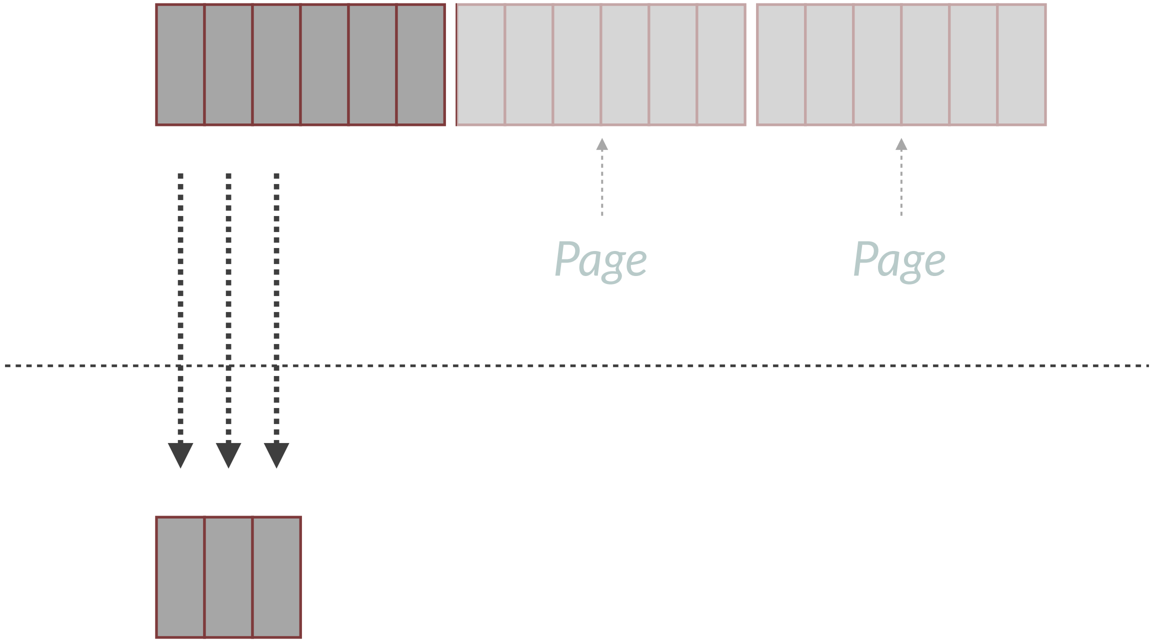
Synchronous



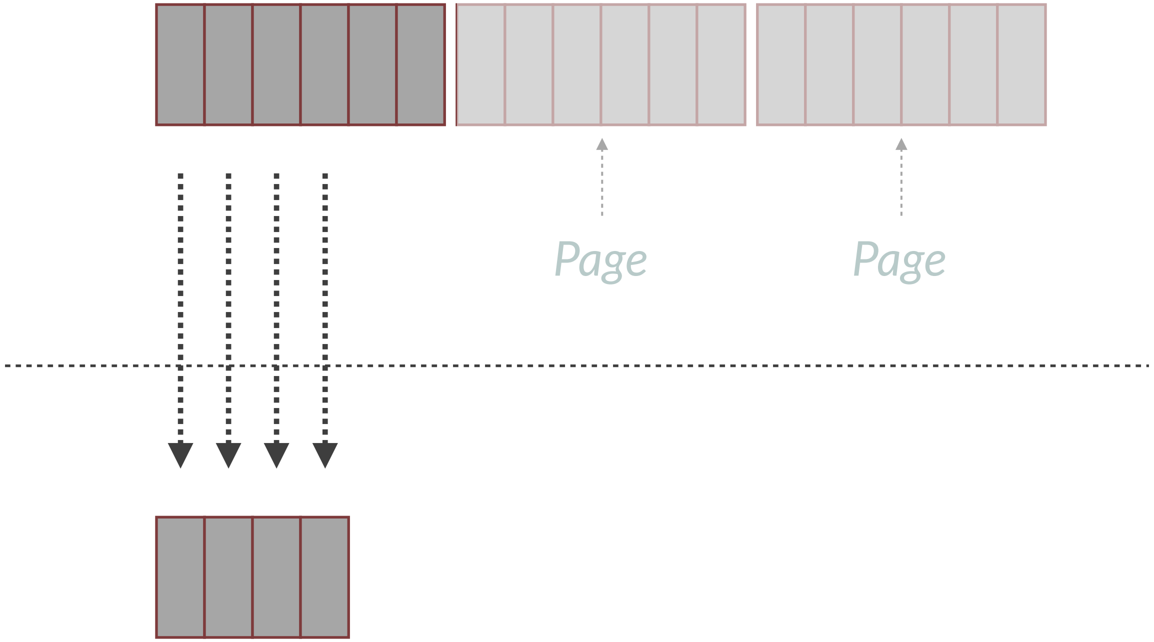
Synchronous



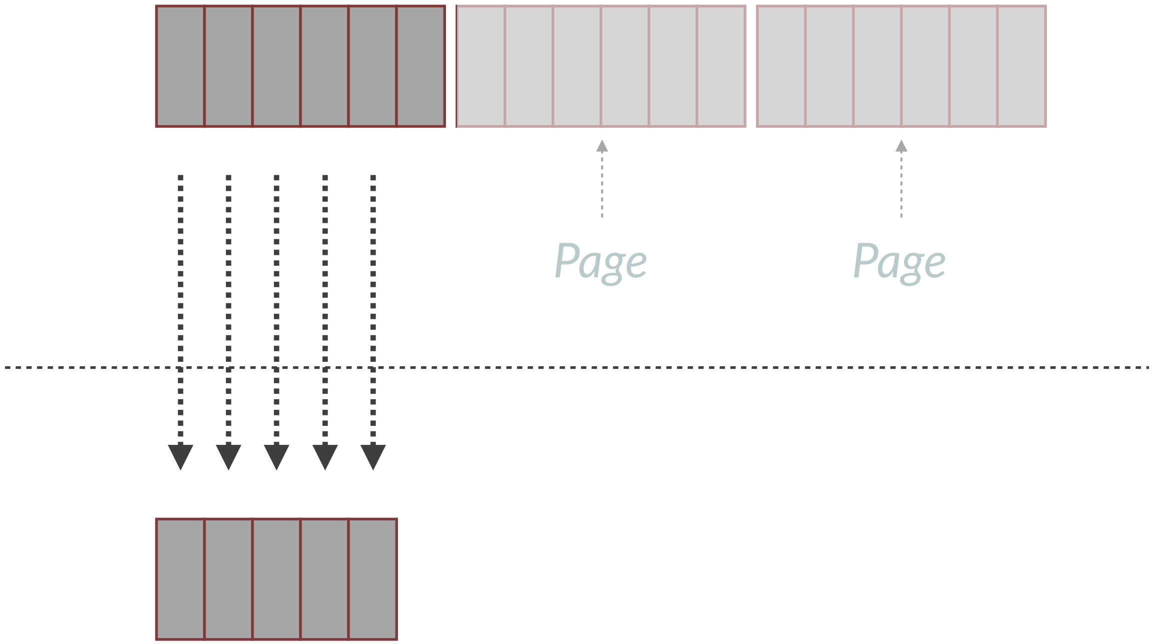
Synchronous



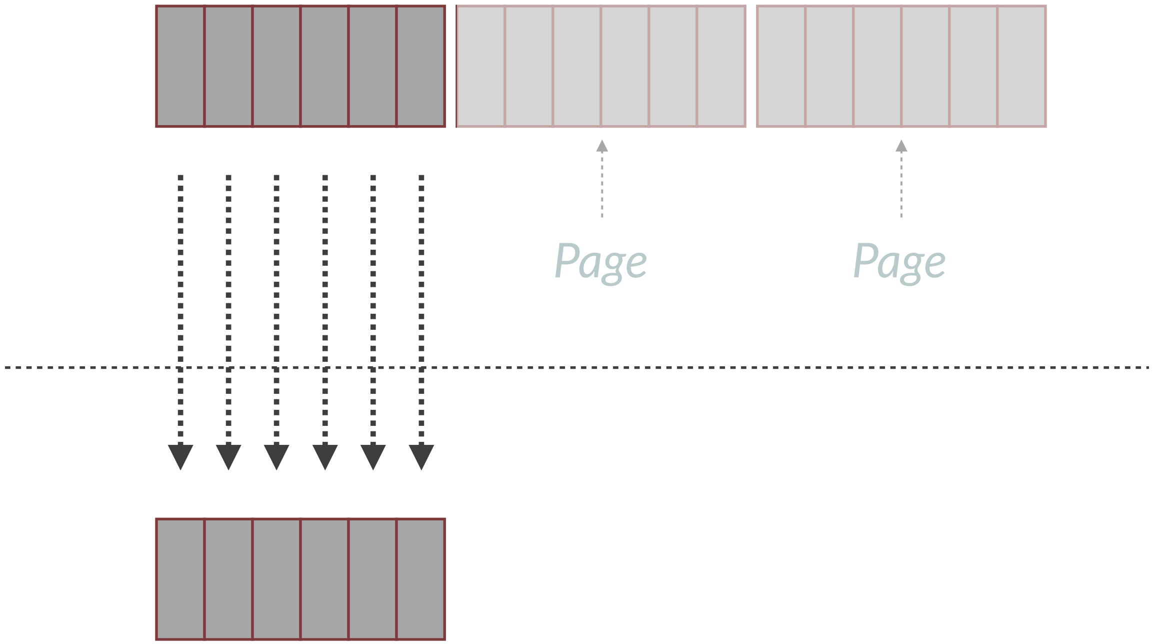
Synchronous



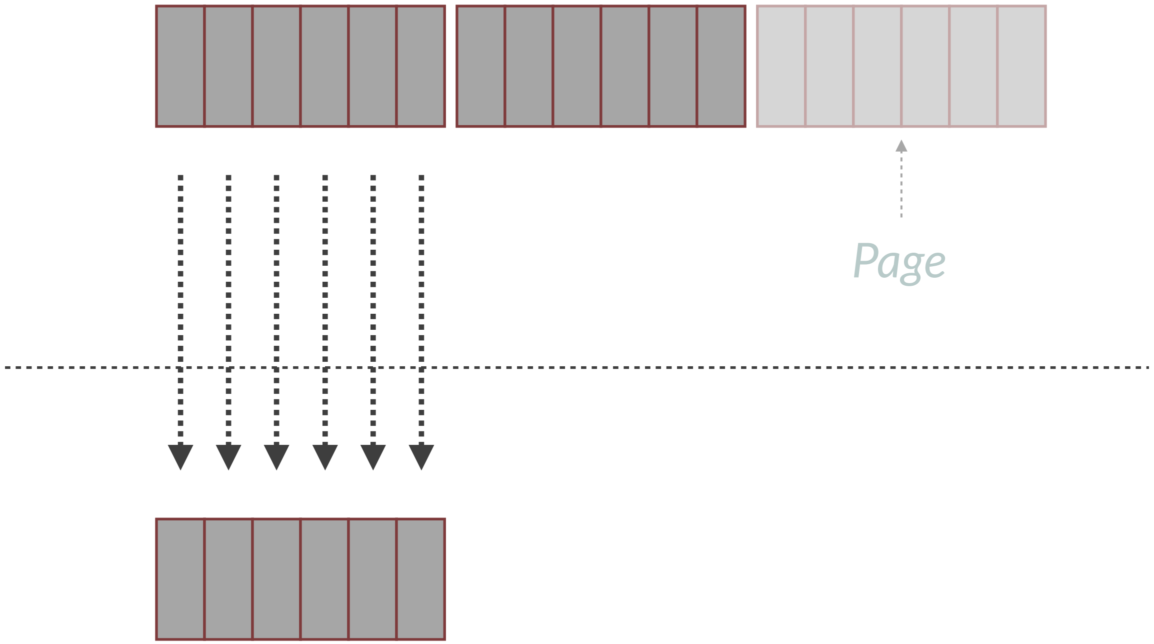
Synchronous



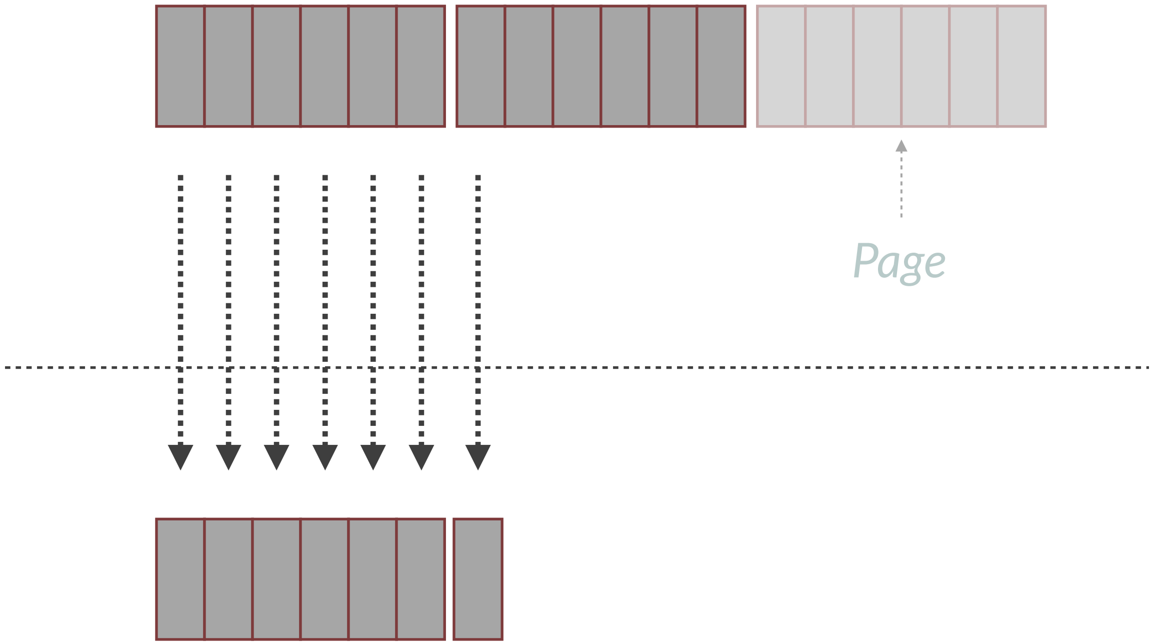
Synchronous



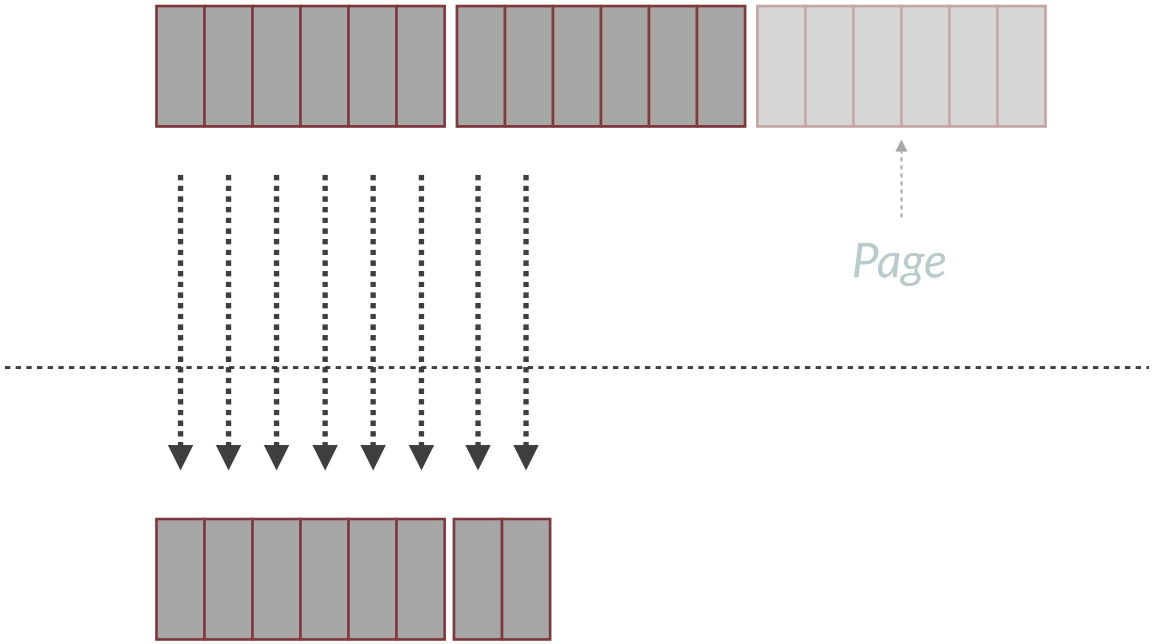
Synchronous



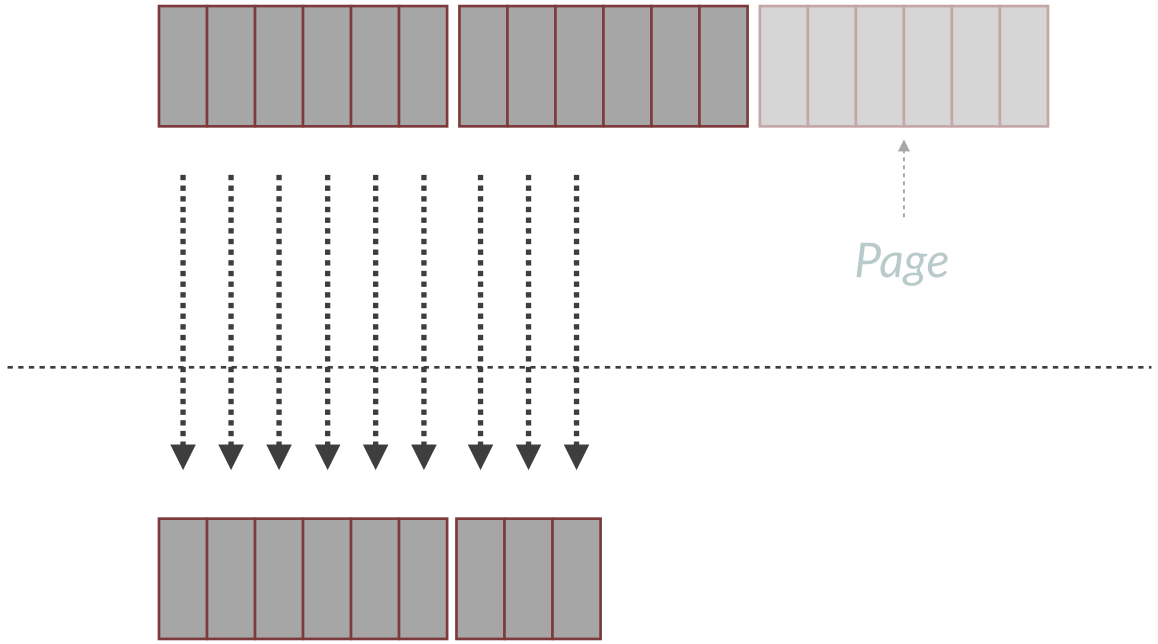
Synchronous



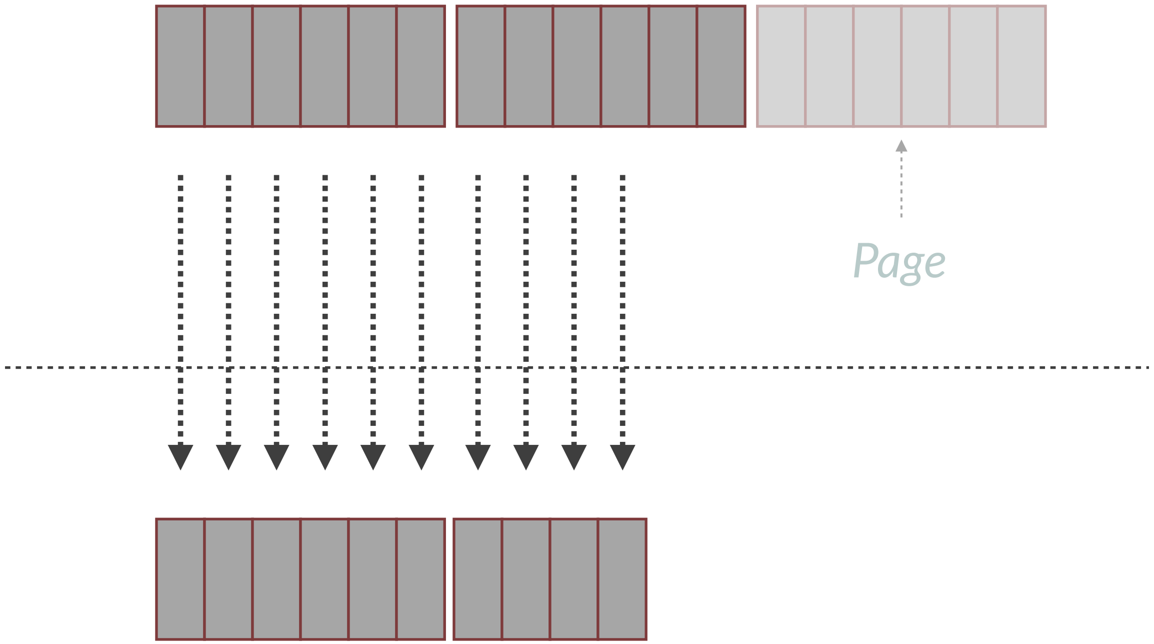
Synchronous



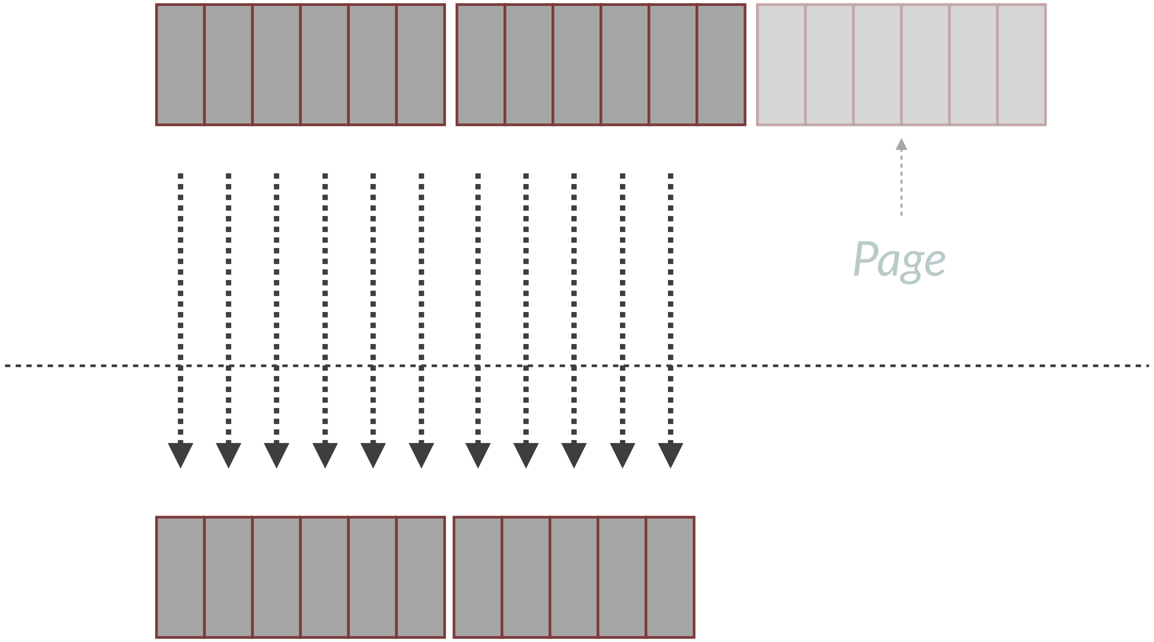
Synchronous



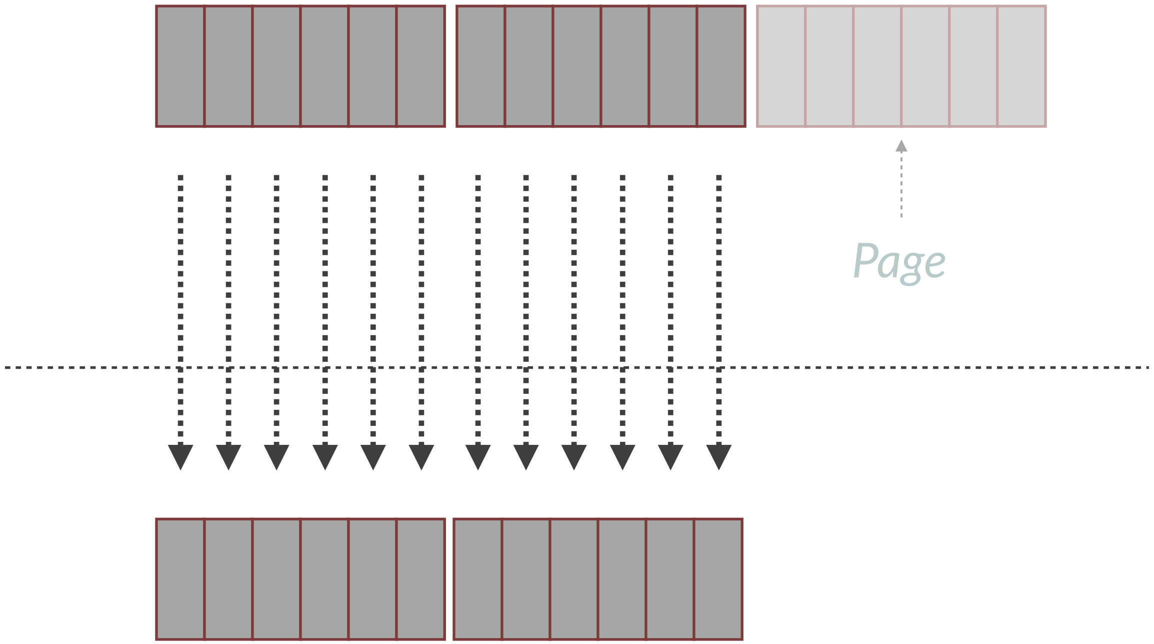
Synchronous



Synchronous



Synchronous



```
def download_files(host, port, outdir):
    hostname = f'http://{host}:{port}'
    list_url = f'{hostname}/list'
    get_url = f'{hostname}/get'

    response = requests.get(list_url)
    response.raise_for_status()
    content = json.loads(response.content)
    while True:
        for filename in content['FileNames']:
            remote_url = f'{get_url}/{filename}'
            download_file(remote_url,
                          os.path.join(outdir, filename))
        if 'NextMarker' not in content:
            break
        response = requests.get(
            f'{list_url}?next-marker={content["NextFile"]}')
        response.raise_for_status()
        content = json.loads(response.content)
```

```python
def download_files(host, port, outdir):
    hostname = f'http://{host}:{port}'
    list_url = f'{hostname}/list'
    get_url = f'{hostname}/get'

    response = requests.get(list_url)
    response.raise_for_status()
    content = json.loads(response.content)
    while True:
        for filename in content['FileNames']:
            remote_url = f'{get_url}/{filename}'
            download_file(remote_url,
                          os.path.join(outdir, filename))
        if 'NextMarker' not in content:
            break
        response = requests.get(
            f'{list_url}?next-marker={content["NextMarker"]}')
        response.raise_for_status()
        content = json.loads(response.content)
```

```python
def download_files(host, port, outdir):
    hostname = f'http://{host}:{port}'
    list_url = f'{hostname}/list'
    get_url = f'{hostname}/get'

    response = requests.get(list_url)
    response.raise_for_status()
    content = json.loads(response.content)
    while True:
        for filename in content['FileNames']:
            remote_url = f'{get_url}/{filename}'
            download_file(remote_url,
                          os.path.join(outdir, filename))
        if 'NextMarker' not in content:
            break
        response = requests.get(
            f'{list_url}?next-marker={content["NextMarker"]}')
        response.raise_for_status()
        content = json.loads(response.content)
```

```python
def download_files(host, port, outdir):
    hostname = f'http://{host}:{port}'
    list_url = f'{hostname}/list'
    get_url = f'{hostname}/get'

    response = requests.get(list_url)
    response.raise_for_status()
    content = json.loads(response.content)
    while True:
        for filename in content['FileNames']:
            remote_url = f'{get_url}/{filename}'
            download_file(remote_url,
                          os.path.join(outdir, filename))
        if 'NextMarker' not in content:
            break
        response = requests.get(
            f'{list_url}?next-marker={content["NextMarker"]}')
        response.raise_for_status()
        content = json.loads(response.content)
```

```
def download_files(host, port, outdir):
    hostname = f'http://{host}:{port}'
    list_url = f'{hostname}/list'
    get_url = f'{hostname}/get'

    response = requests.get(list_url)
    response.raise_for_status()
    content = json.loads(response.content)
    while True:
        for filename in content['FileNames']:
            remote_url = f'{get_url}/{filename}'
            download_file(remote_url,
                          os.path.join(outdir, filename))
        if 'NextMarker' not in content:
            break
        response = requests.get(
            f'{list_url}?next-marker={content["NextMarker"]}')
        response.raise_for_status()
        content = json.loads(response.content)
```

```python
def download_files(host, port, outdir):
    hostname = f'http://{host}:{port}'
    list_url = f'{hostname}/list'
    get_url = f'{hostname}/get'

    response = requests.get(list_url)
    response.raise_for_status()
    content = json.loads(response.content)
    while True:
        for filename in content['FileNames']:
            remote_url = f'{get_url}/{filename}'
            download_file(remote_url,
                          os.path.join(outdir, filename))
        if 'NextMarker' not in content:
            break
        response = requests.get(
            f'{list_url}?next-marker={content["NextFile"]}')
        response.raise_for_status()
        content = json.loads(response.content)
```

```python
def download_file(remote_url, local_filename):
    response = requests.get(remote_url)
    response.raise_for_status()
    with open(local_filename, 'wb') as f:
        f.write(response.content)
```

Synchronous Results

Synchronous Results

One request	**0.003 seconds**

Synchronous Results

One request	0.003 seconds
One billion requests	3,000,000 seconds

Synchronous Results

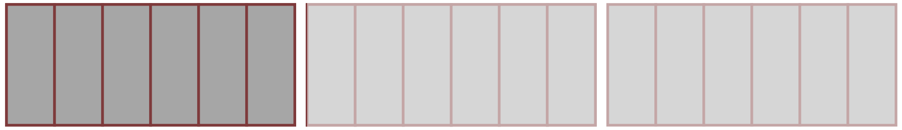
One request	0.003 seconds
One billion requests	3,000,000 seconds 833.3 hours

Synchronous Results

One request	0.003 seconds
One billion requests	3,000,000 seconds
	833.3 hours
	34.7 days

Multithreading

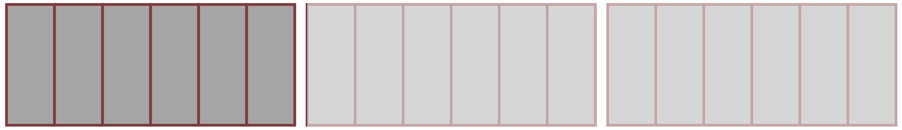
Multithreading



List Files can't be parallelized.

But Get File can be parallelized.

Multithreading

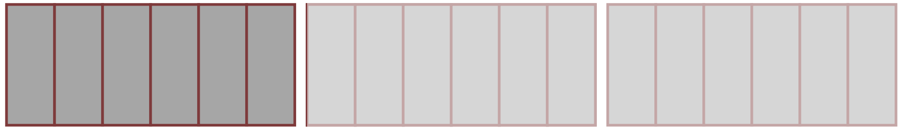


List Files can't be parallelized.

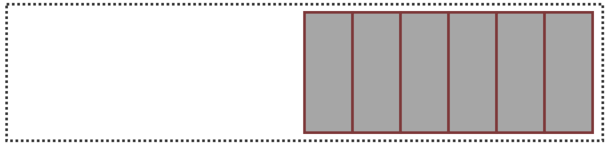
queue.Queue

But Get File can be parallelized.

Multithreading



List Files can't be parallelized.

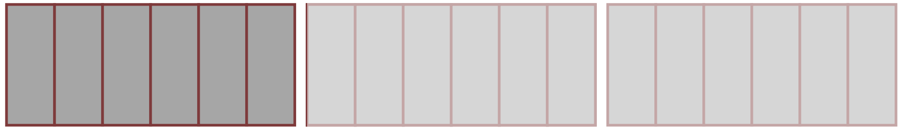


queue.Queue

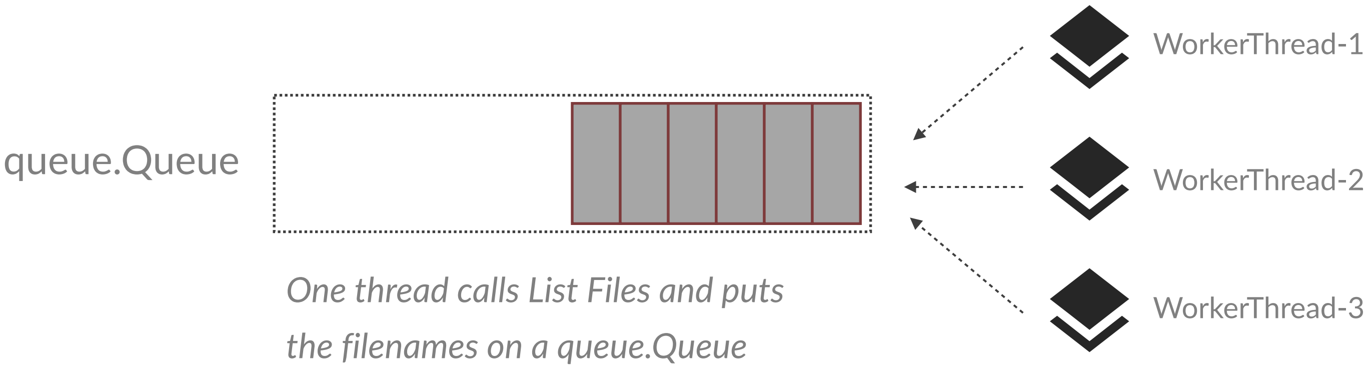
One thread calls List Files and puts the filenames on a queue.Queue

But Get File can be parallelized.

Multithreading



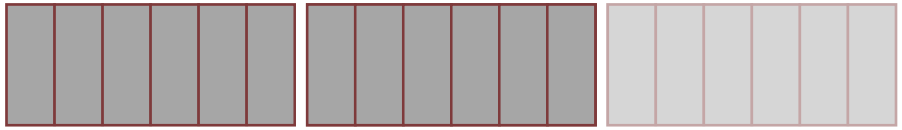
List Files can't be parallelized.



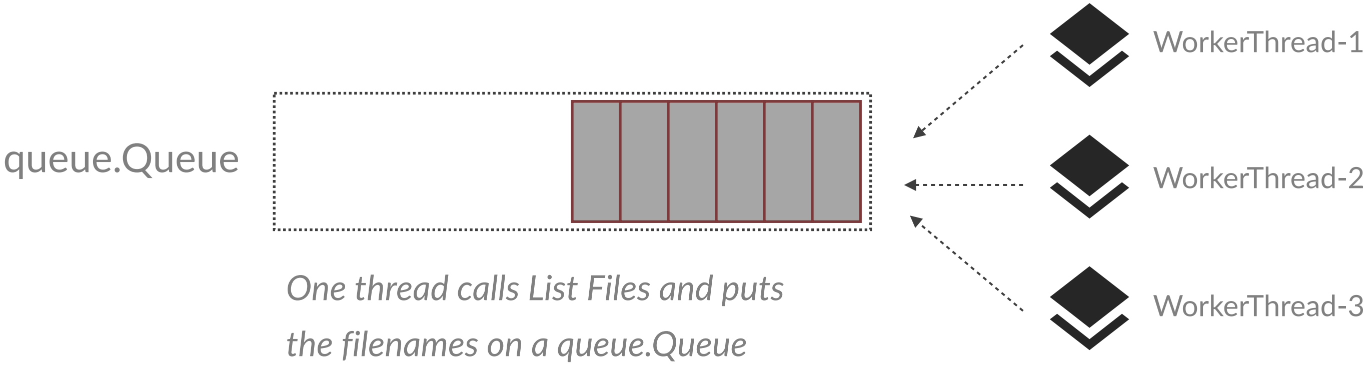
One thread calls List Files and puts the filenames on a queue.Queue

But Get File can be parallelized.

Multithreading



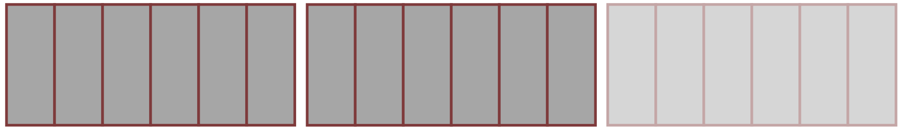
List Files can't be parallelized.



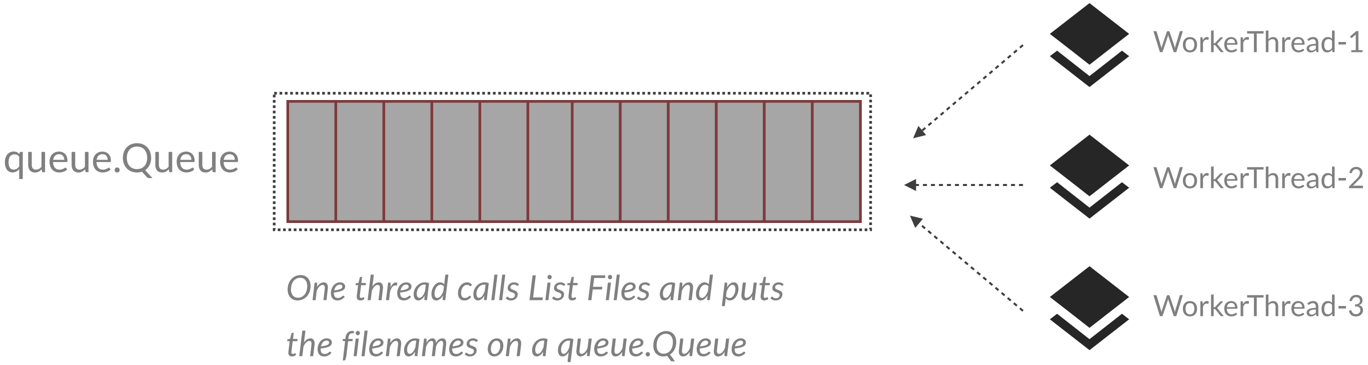
One thread calls List Files and puts the filenames on a queue.Queue

But Get File can be parallelized.

Multithreading



List Files can't be parallelized.



queue.Queue

WorkerThread-1

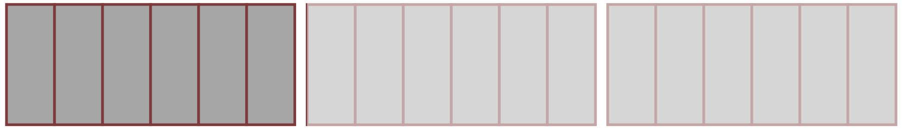
WorkerThread-2

WorkerThread-3

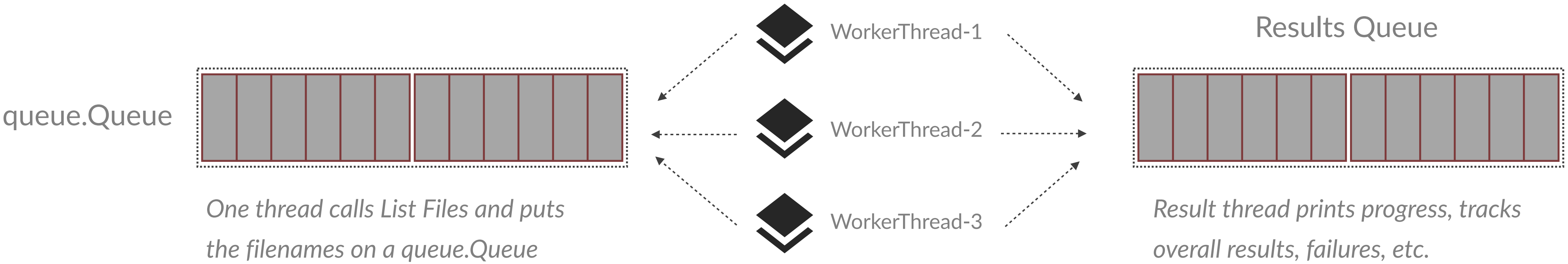
One thread calls List Files and puts the filenames on a queue.Queue

But Get File can be parallelized.

Multithreading



List Files can't be parallelized.



queue.Queue

WorkerThread-1

WorkerThread-2

WorkerThread-3

Results Queue

One thread calls List Files and puts the filenames on a queue.Queue

Result thread prints progress, tracks overall results, failures, etc.

```
def download_files(host, port, outdir, num_threads):
    #  ... same constants as before ...

    work_queue = queue.Queue(MAX_SIZE)
    result_queue = queue.Queue(MAX_SIZE)

    threads = []
    for i in range(num_threads):
        t = threading.Thread(
            target=worker_thread, args=(work_queue, result_queue))
        t.start()
        threads.append(t)
    result_thread = threading.Thread(target=result_poller,
                                     args=(result_queue,))
    result_thread.start()
    threads.append(result_thread)

    # ...
```

```python
def download_files(host, port, outdir, num_threads):
    #  ... same constants as before ...

    work_queue = queue.Queue(MAX_SIZE)
    result_queue = queue.Queue(MAX_SIZE)

    threads = []
    for i in range(num_threads):
        t = threading.Thread(
            target=worker_thread, args=(work_queue, result_queue))
        t.start()
        threads.append(t)
    result_thread = threading.Thread(target=result_poller,
                                     args=(result_queue,))
    result_thread.start()
    threads.append(result_thread)

    # ...
```

```python
response = requests.get(list_url)
response.raise_for_status()
content = json.loads(response.content)
while True:
    for filename in content['FileNames']:
        remote_url = f'{get_url}/{filename}'
        outfile = os.path.join(outdir, filename)
        work_queue.put((remote_url, outfile))
    if 'NextFile' not in content:
        break
    response = requests.get(
        f'{list_url}?next-marker={content["NextFile"]}')
    response.raise_for_status()
    content = json.loads(response.content)
```

```python
response = requests.get(list_url)
response.raise_for_status()
content = json.loads(response.content)
while True:
    for filename in content['FileNames']:
        remote_url = f'{get_url}/{filename}'
        outfile = os.path.join(outdir, filename)
        work_queue.put((remote_url, outfile))
    if 'NextFile' not in content:
        break
    response = requests.get(
        f'{list_url}?next-marker={content["NextFile"]}')
    response.raise_for_status()
    content = json.loads(response.content)
```

```python
def worker_thread(work_queue, result_queue):
    while True:
        work = work_queue.get()
        if work is _SHUTDOWN:
            return
        remote_url, outfile = work
        download_file(remote_url, outfile)
        result_queue.put(_SUCCESS)
```

```
def worker_thread(work_queue, result_queue):
    while True:
        work = work_queue.get()
        if work is _SHUTDOWN:
            return
        remote_url, outfile = work
        download_file(remote_url, outfile)
        result_queue.put(_SUCCESS)
```

Multithreaded Results - 10 threads

Multithreaded Results - 10 threads

One request	0.0036 seconds

Multithreaded Results - 10 threads

One request	0.0036 seconds
One billion requests	3,600,000 seconds 1000.0 hours 41.6 days

Multithreaded Results - 100 threads

Multithreaded Results - 100 threads

One request	0.0042 seconds

Multithreaded Results - 100 threads

One request	0.0042 seconds
One billion requests	4,200,000 seconds 1166.67 hours 48.6 days

Why?

Not necessarily IO bound due to low latency and small file size

GIL contention, overhead of passing data through queues

Things to keep in mind

The real code is more complicated, ctrl-c, graceful shutdown, etc.

Debugging is much harder, non-deterministic

The more you stray from stdlib abstractions, more likely to encounter race conditions

Can't use `concurrent.futures` `map()` because of large number of files

Multiprocessing

Our Task (the details)

What client machine will this run on?

We have one machine we can use, 16 cores, 64GB memory

What about the network between the client and server?

Our client machine is on the same network as the service with remote files

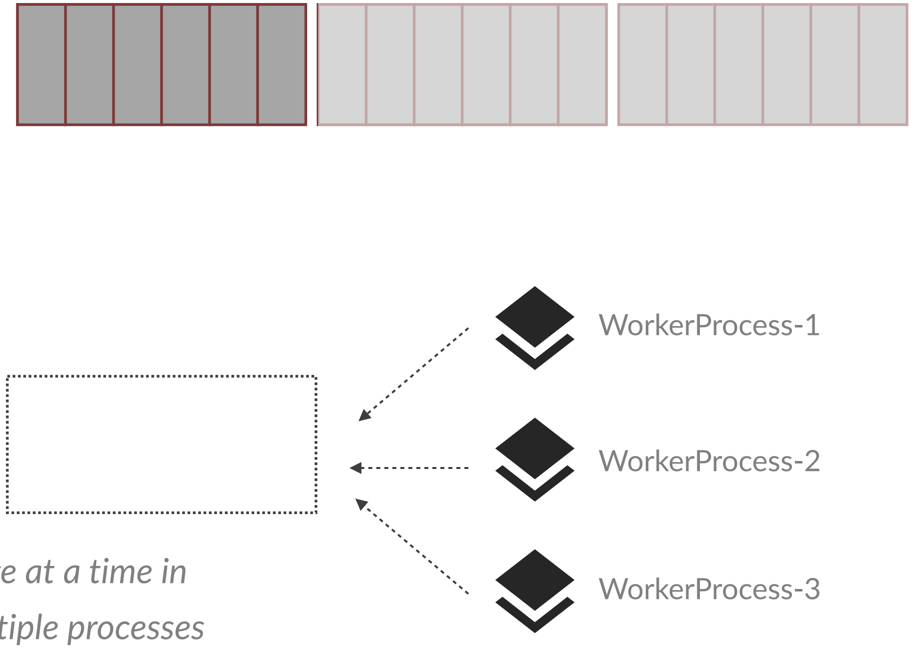
How many files are on the remote server?

Approximately one billion files, 100 bytes per file

When do you need this done?

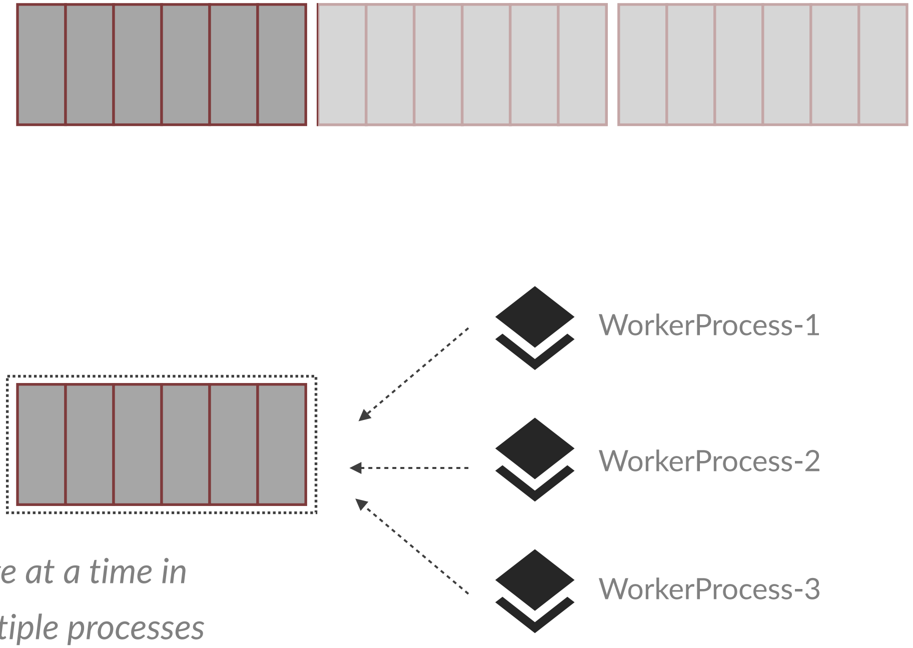
Please have this done as soon as possible

Multiprocessing



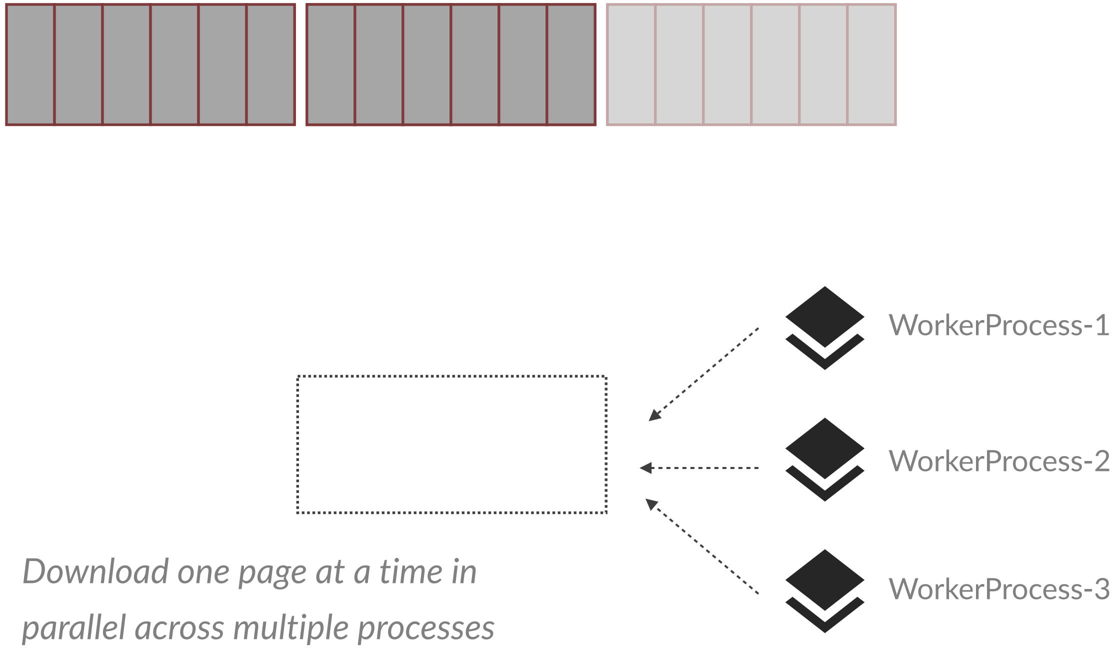
Download one page at a time in parallel across multiple processes

Multiprocessing



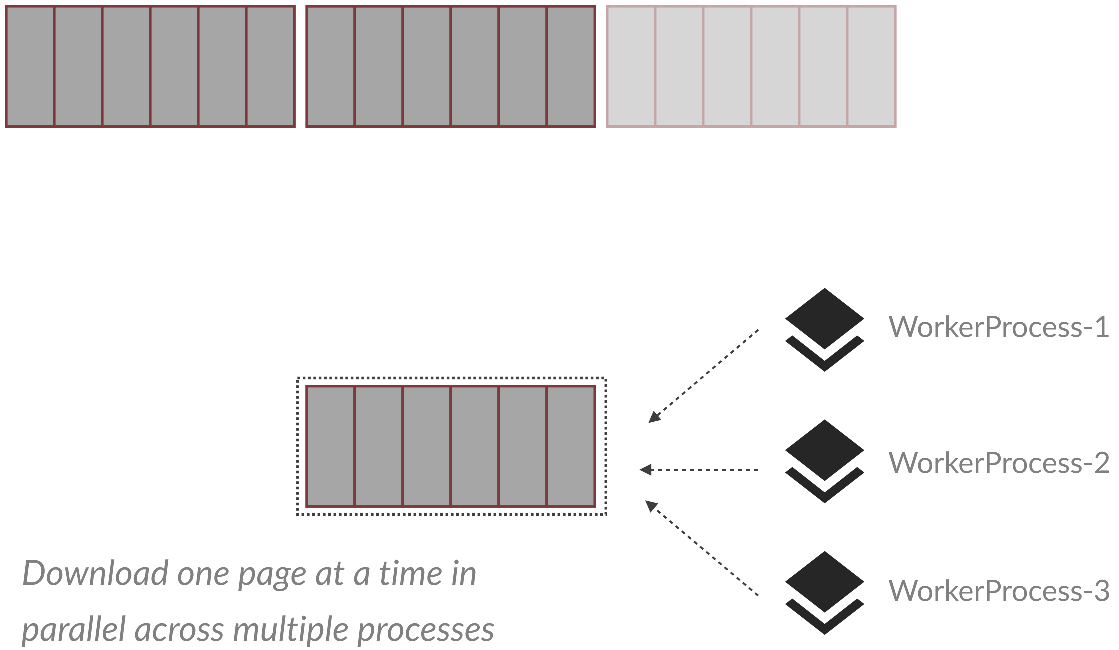
Download one page at a time in parallel across multiple processes

Multiprocessing



Download one page at a time in parallel across multiple processes

Multiprocessing



Download one page at a time in parallel across multiple processes

```python
from concurrent import futures


def download_files(host, port, outdir):
    hostname = f'http://{host}:{port}'
    list_url = f'{hostname}/list'

    all_pages = iter_all_pages(list_url)
    downloader = Downloader(host, port, outdir)
    with futures.ProcessPoolExecutor() as executor:
        for page in all_pages:
            future_to_filename = {}
            for filename in page:
                future = executor.submit(downloader.download,
                                         filename)
                future_to_filename[future] = filename
            for future in futures.as_completed(future_to_filename):
                future.result()
```

```python
from concurrent import futures


def download_files(host, port, outdir):
    hostname = f'http://{host}:{port}'
    list_url = f'{hostname}/list'

    all_pages = iter_all_pages(list_url)
    downloader = Downloader(host, port, outdir)
    with futures.ProcessPoolExecutor() as executor:
        for page in all_pages:
            future_to_filename = {}
            for filename in page:
                future = executor.submit(downloader.download,
                                         filename)
                future_to_filename[future] = filename
            for future in futures.as_completed(future_to_filename):
                future.result()
```

Start parallel downloads

```python
from concurrent import futures


def download_files(host, port, outdir):
    hostname = f'http://{host}:{port}'
    list_url = f'{hostname}/list'

    all_pages = iter_all_pages(list_url)
    downloader = Downloader(host, port, outdir)
    with futures.ProcessPoolExecutor() as executor:
        for page in all_pages:
            future_to_filename = {}
            for filename in page:
                future = executor.submit(downloader.download,
                                         filename)
                future_to_filename[future] = filename
            for future in futures.as_completed(future_to_filename):
                future.result()
```

```
from concurrent import futures


def download_files(host, port, outdir):
    hostname = f'http://{host}:{port}'
    list_url = f'{hostname}/list'

    all_pages = iter_all_pages(list_url)
    downloader = Downloader(host, port, outdir)
    with futures.ProcessPoolExecutor() as executor:
        for page in all_pages:
            future_to_filename = {}
            for filename in page:
                future = executor.submit(downloader.download,
                                         filename)
                future_to_filename[future] = filename
            for future in futures.as_completed(future_to_filename):
                future.result()
```

Wait for downloads to finish

```python
def iter_all_pages(list_url):
    session = requests.Session()
    response = session.get(list_url)
    response.raise_for_status()
    content = json.loads(response.content)
    while True:
        yield content['FileNames']
        if 'NextFile' not in content:
            break
        response = session.get(
            f'{list_url}?next-marker={content["NextFile"]}')
        response.raise_for_status()
        content = json.loads(response.content)
```

```
class Downloader:
    # ...

    def download(self, filename):
        remote_url = f'{self.get_url}/{filename}'
        response = self.session.get(remote_url)
        response.raise_for_status()
        outfile = os.path.join(self.outdir, filename)
        with open(outfile, 'wb') as f:
            f.write(response.content)
```

Multiprocessing Results - 16 processes

Multiprocessing Results - 16 processes

One request	0.00032 seconds

Multiprocessing Results - 16 processes

One request	0.00032 seconds
One billion requests	320,000 seconds 88.88 hours

Multiprocessing Results - 16 processes

One request	0.00032 seconds
One billion requests	320,000 seconds 88.88 hours 3.7 days

Things to keep in mind

Speed improvements due to truly running in parallel

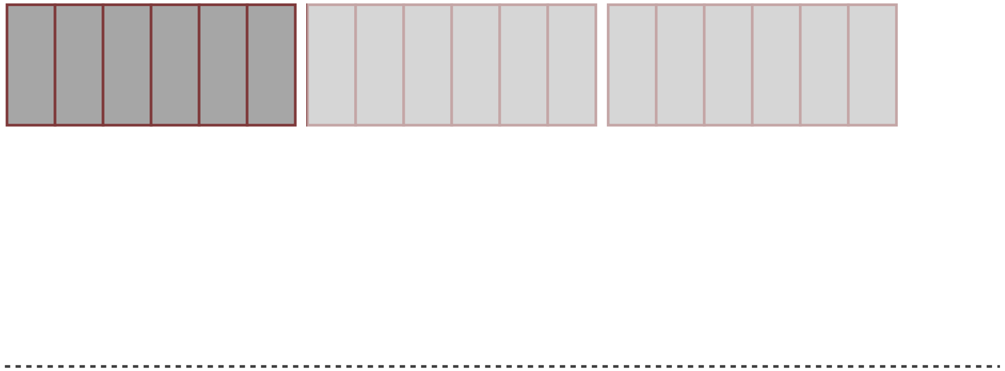
Debugging is much harder, non-deterministic, pdb doesn't work out of the box

IPC overhead between processes higher than threads

Tradeoff between entirely in parallel vs. parallel chunks

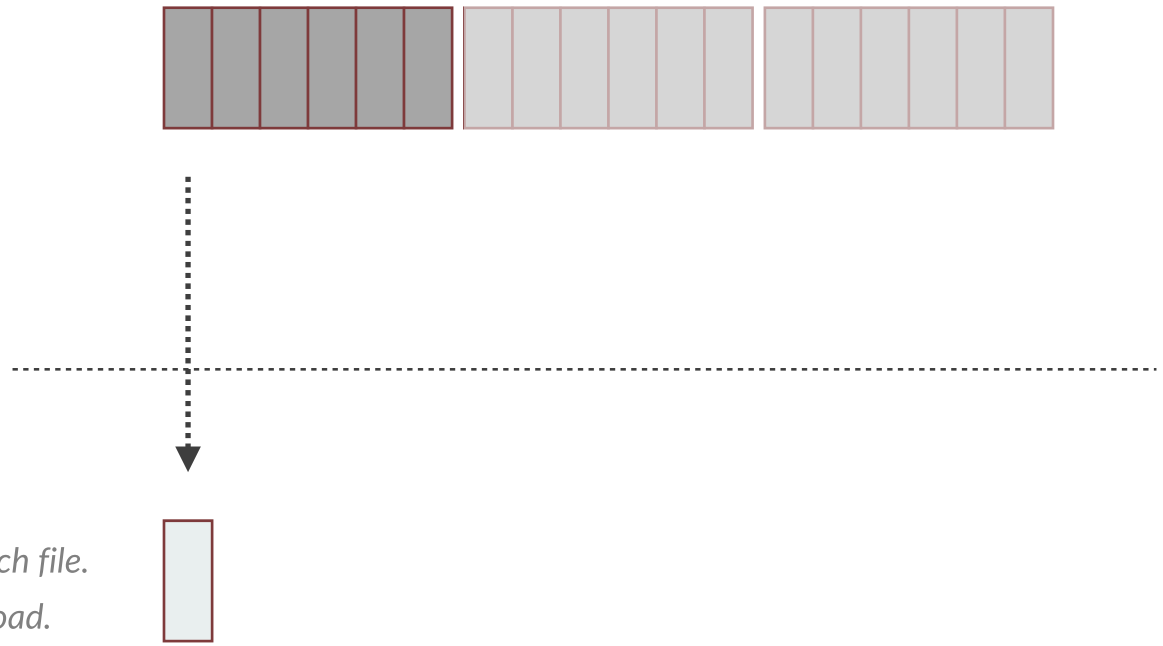
Asyncio

Asyncio



Create an `asyncio.Task` *for each file.*
This immediately starts the download.

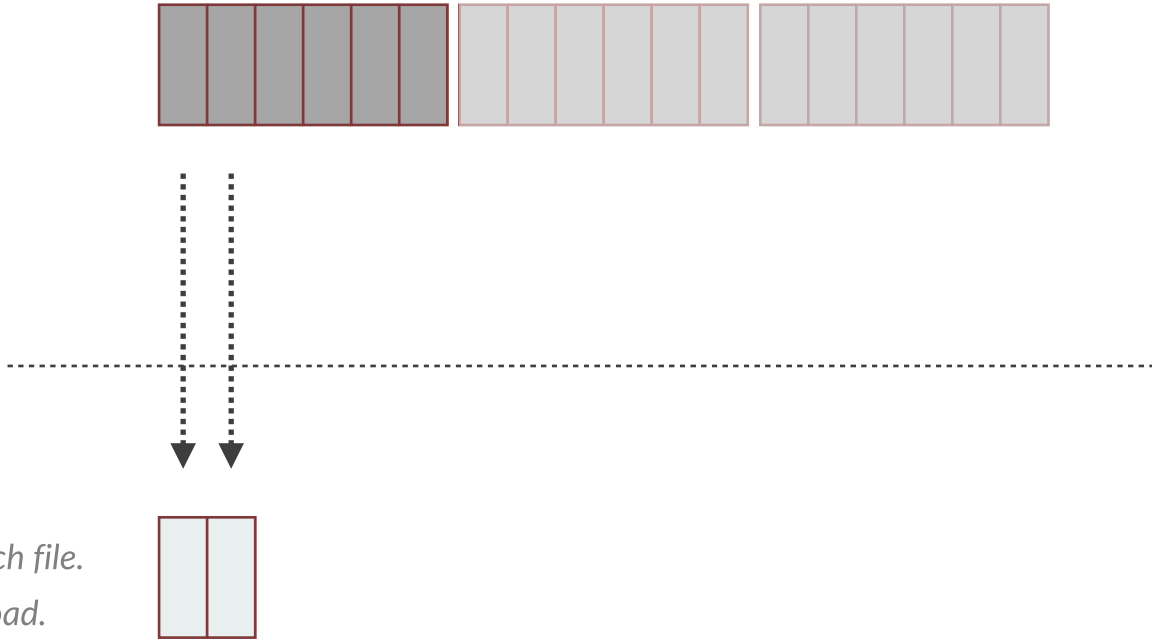
Asyncio



Create an `asyncio.Task` *for each file.*

This immediately starts the download.

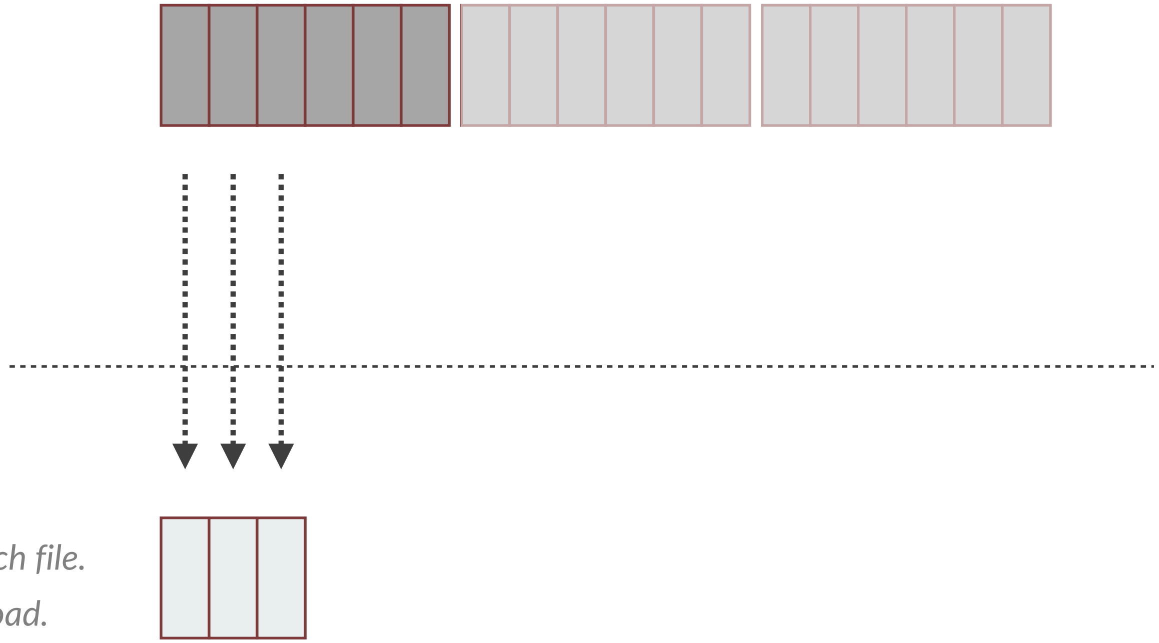
Asyncio



Create an `asyncio.Task` *for each file.*

This immediately starts the download.

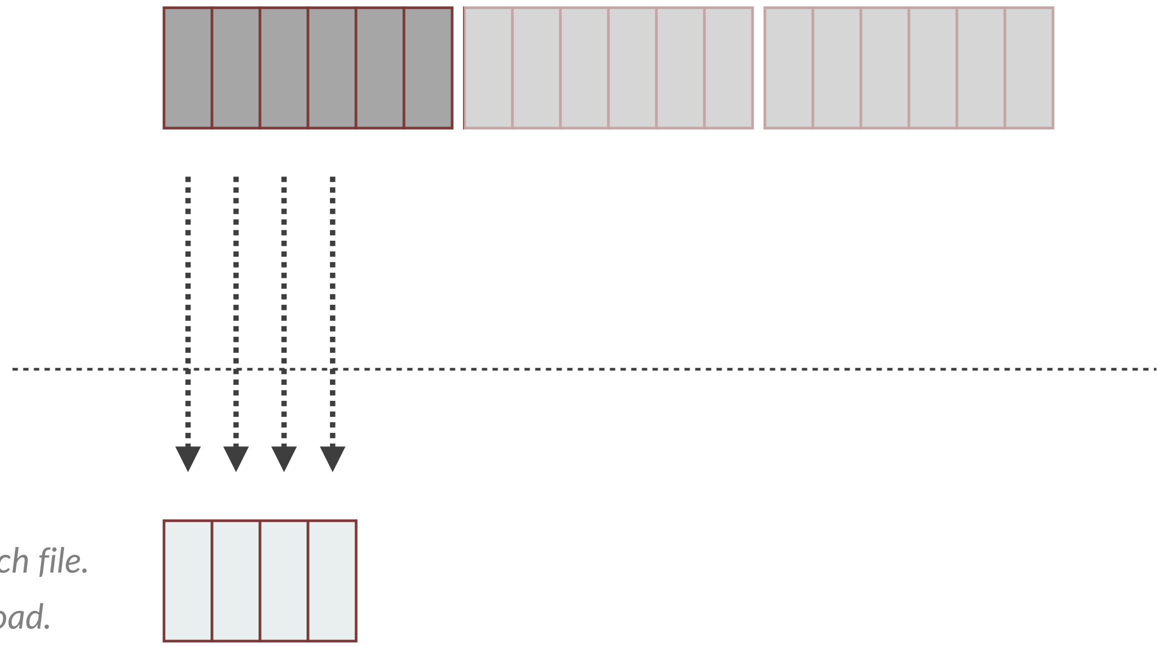
Asyncio



Create an `asyncio.Task` *for each file.*

This immediately starts the download.

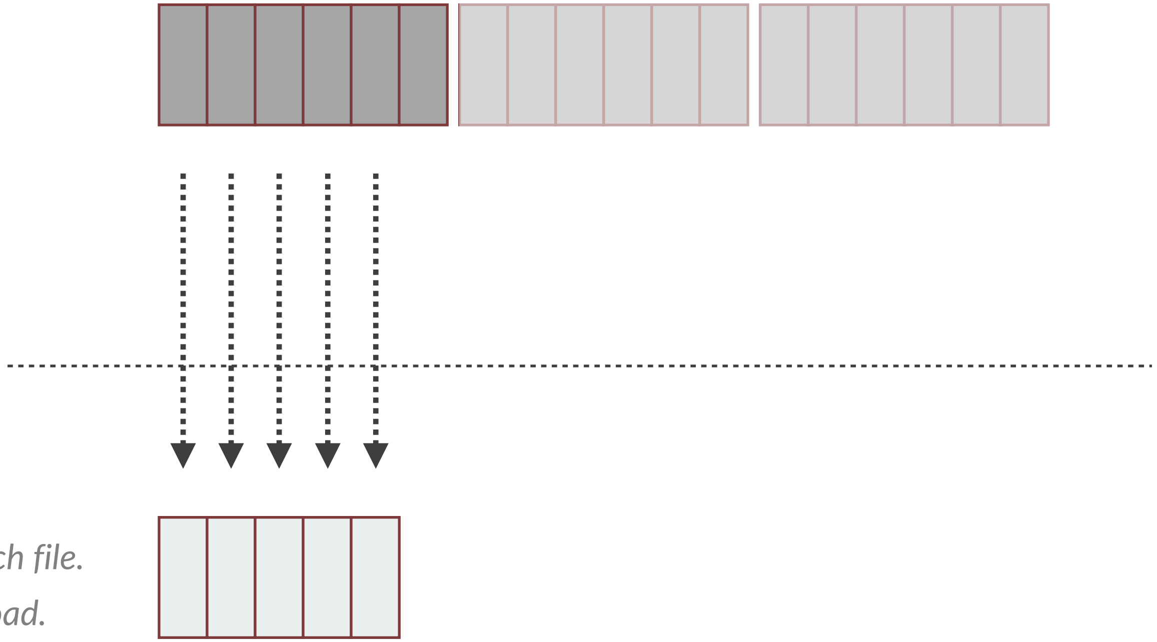
Asyncio



Create an `asyncio.Task` *for each file.*

This immediately starts the download.

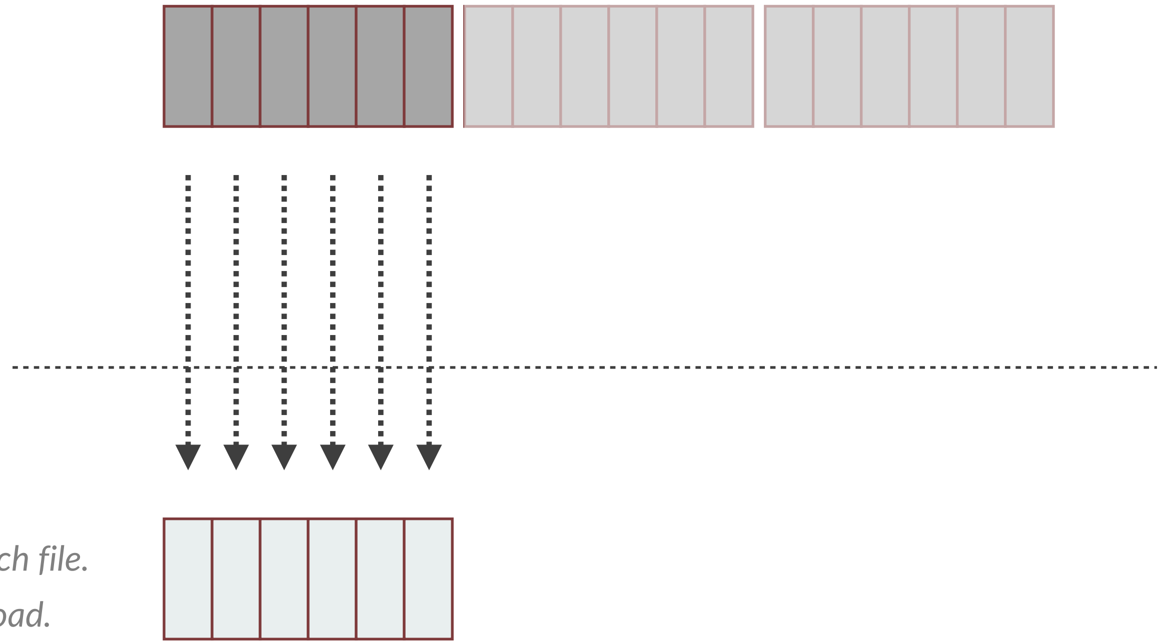
Asyncio



Create an `asyncio.Task` *for each file.*

This immediately starts the download.

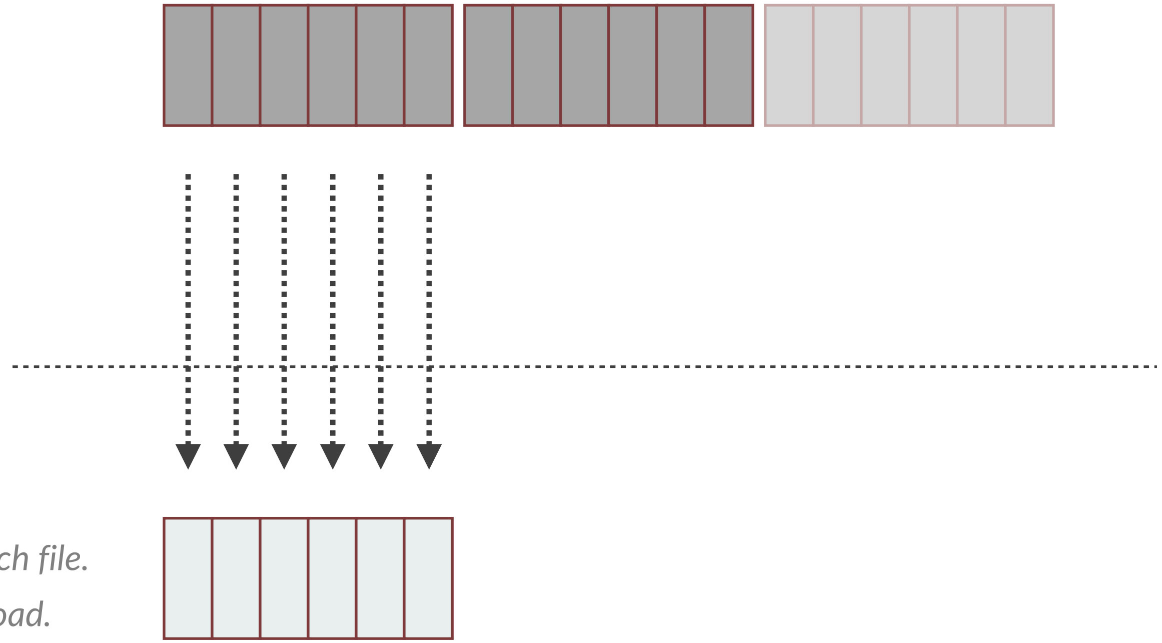
Asyncio



Create an `asyncio.Task` *for each file.*

This immediately starts the download.

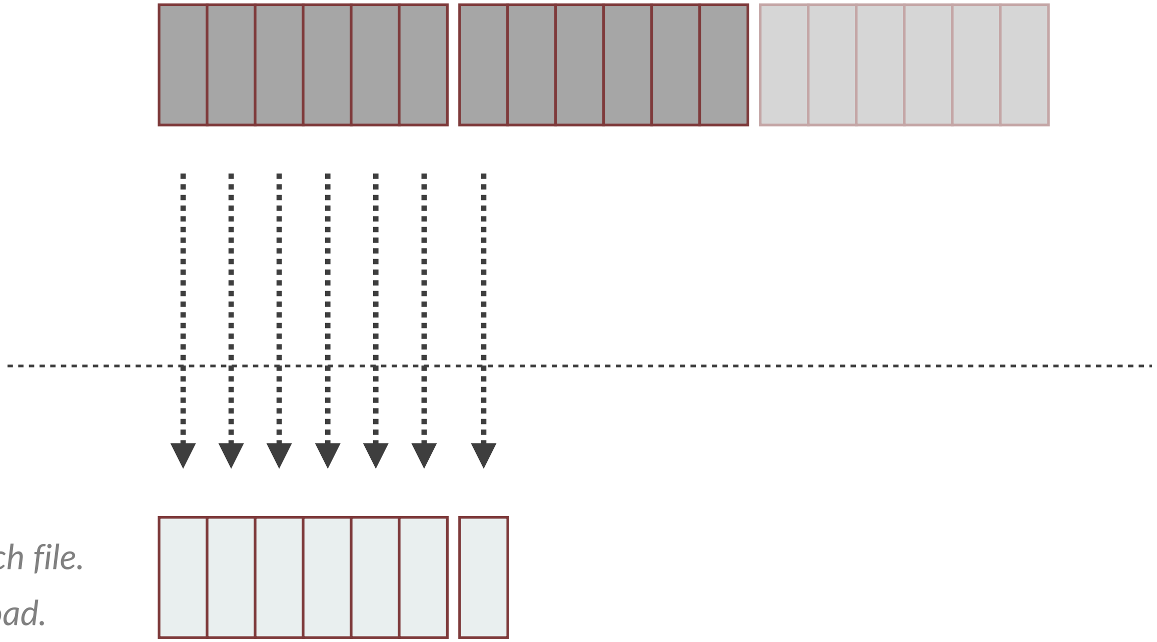
Asyncio



Create an `asyncio.Task` *for each file.*
This immediately starts the download.

Move on to the next page and start creating tasks.

Asyncio

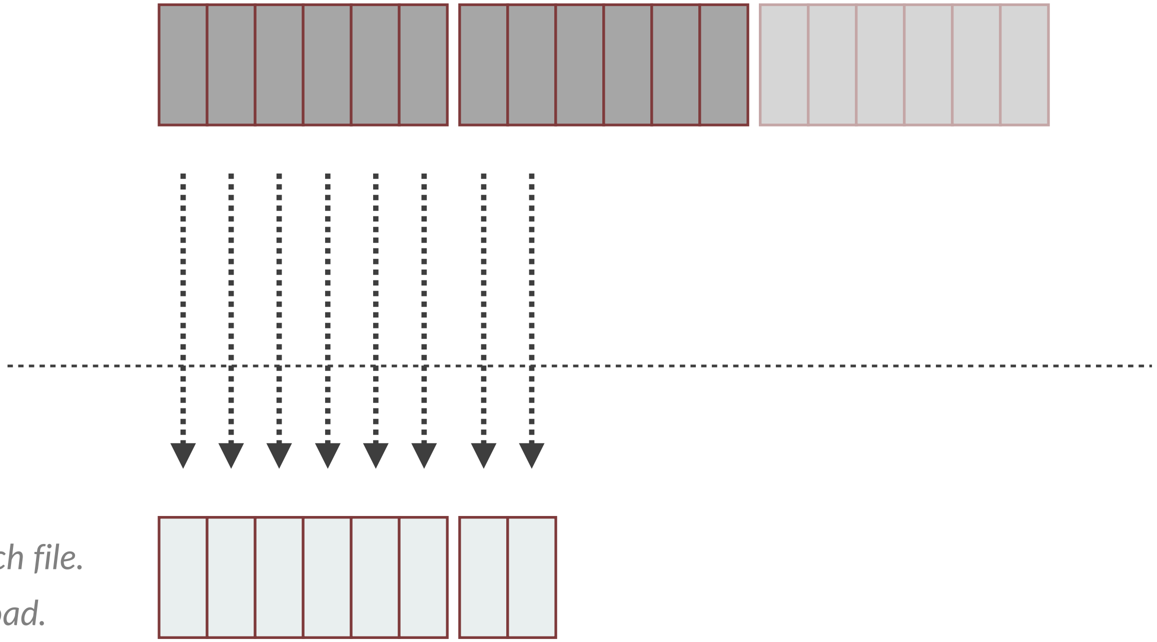


Create an `asyncio.Task` *for each file.*

This immediately starts the download.

Move on to the next page and start creating tasks.

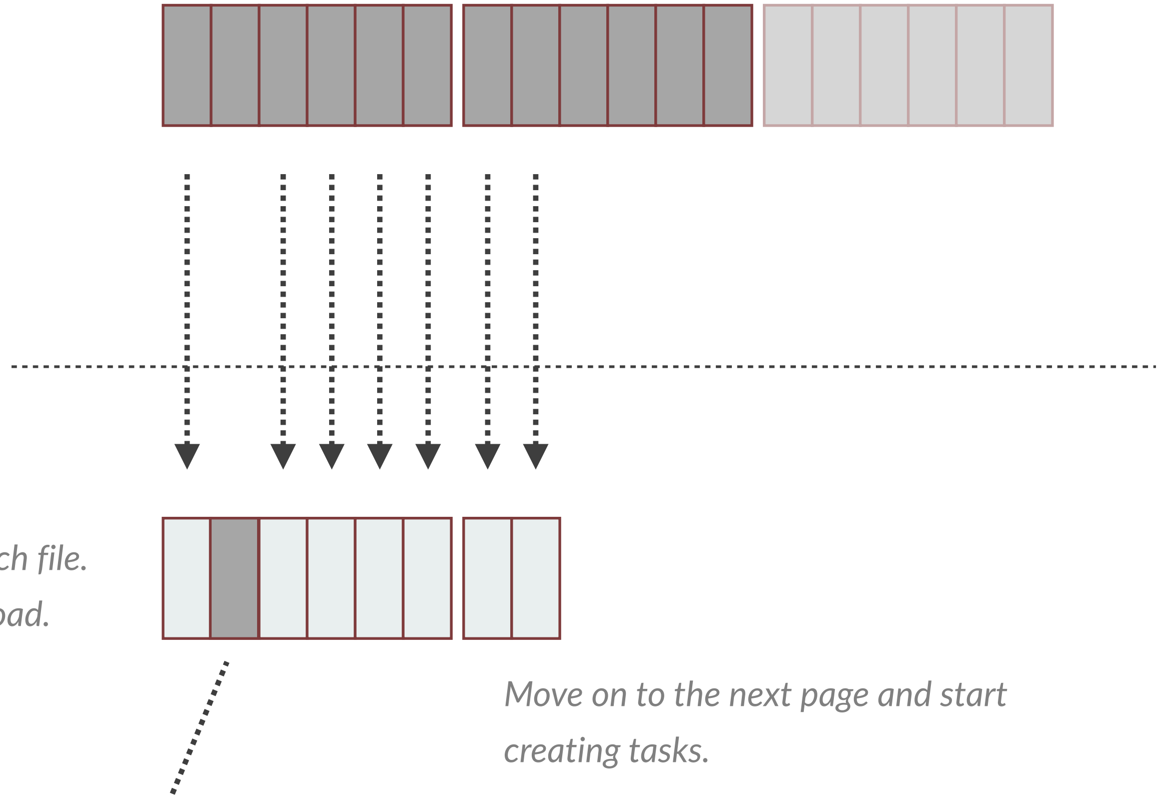
Asyncio



Create an `asyncio.Task` *for each file.*
This immediately starts the download.

Move on to the next page and start creating tasks.

Asyncio

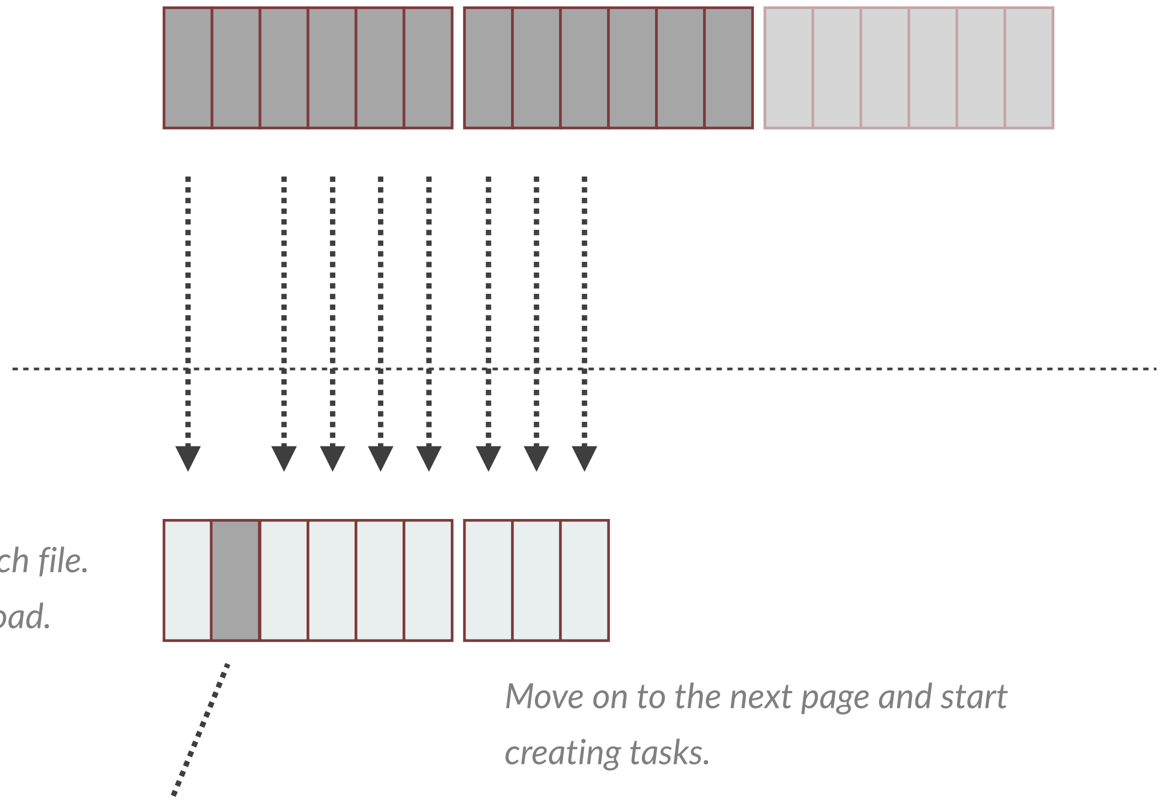


Create an `asyncio.Task` *for each file.*
This immediately starts the download.

Move on to the next page and start creating tasks.

Meanwhile tasks from the first page will finish downloading their file.

Asyncio

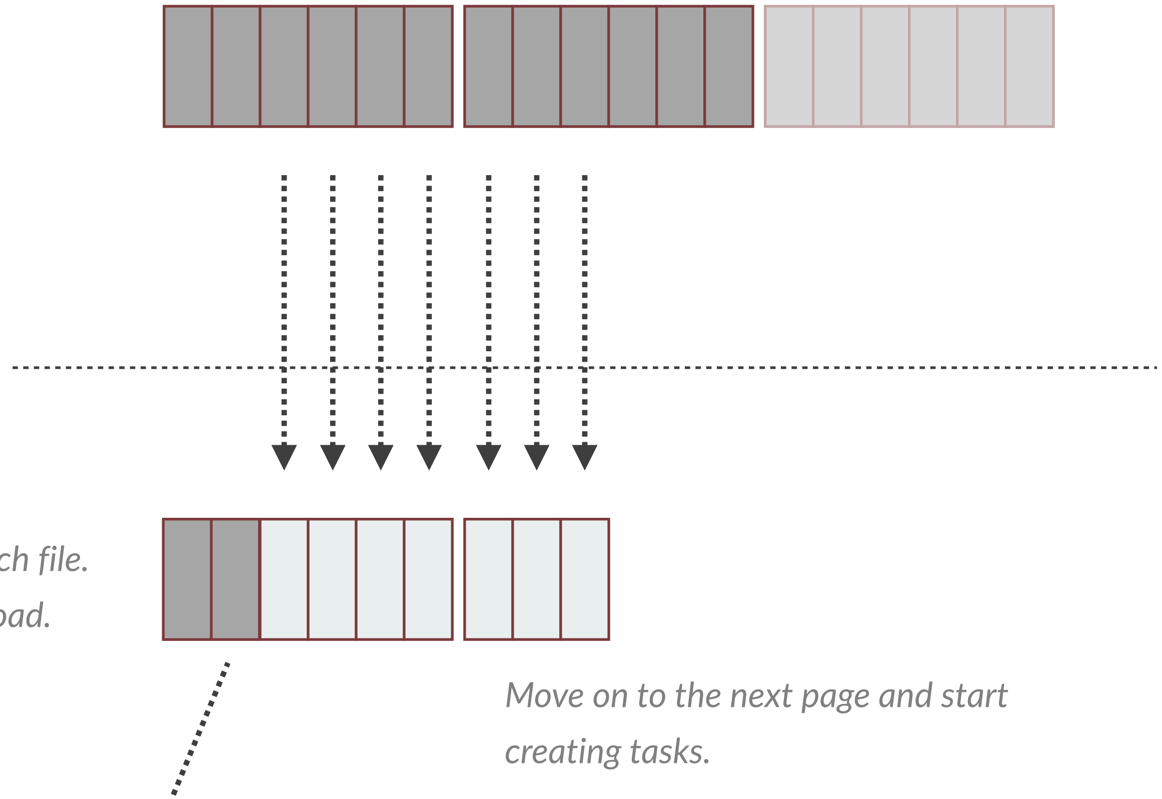


Create an `asyncio.Task` *for each file.*
This immediately starts the download.

Move on to the next page and start creating tasks.

Meanwhile tasks from the first page will finish downloading their file.

Asyncio

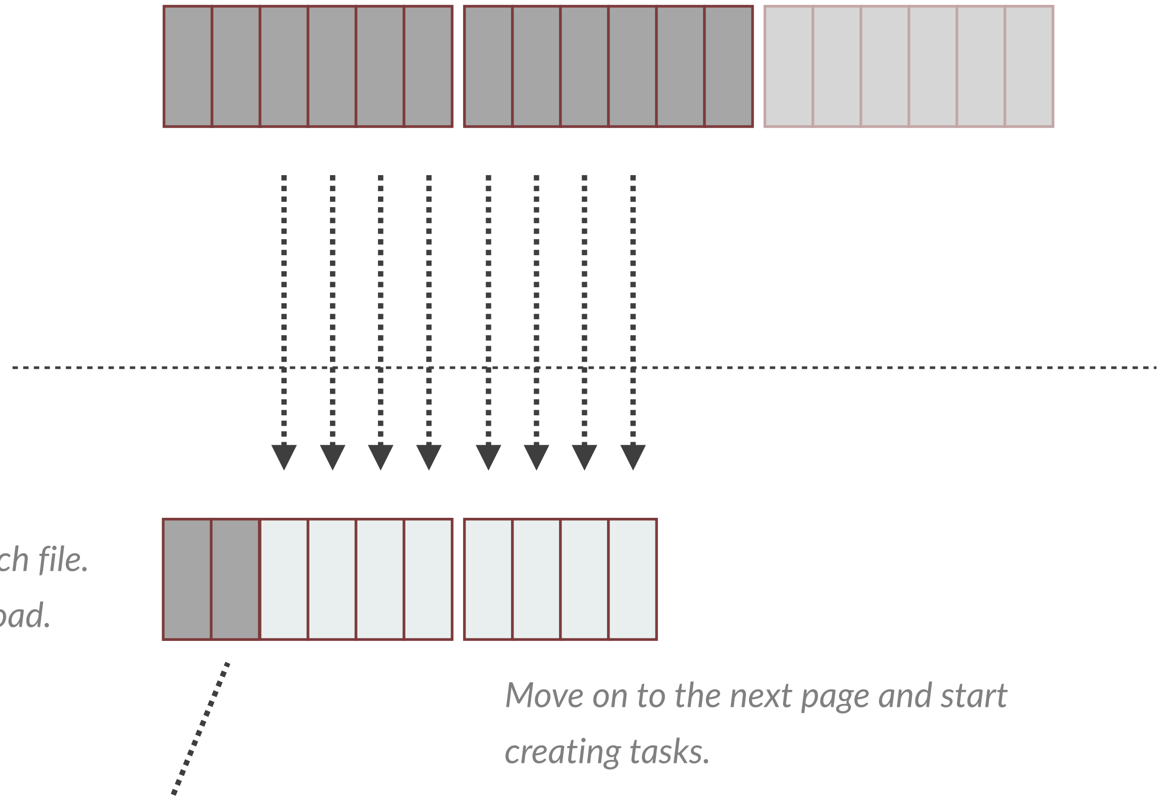


Create an `asyncio.Task` *for each file. This immediately starts the download.*

Move on to the next page and start creating tasks.

Meanwhile tasks from the first page will finish downloading their file.

Asyncio

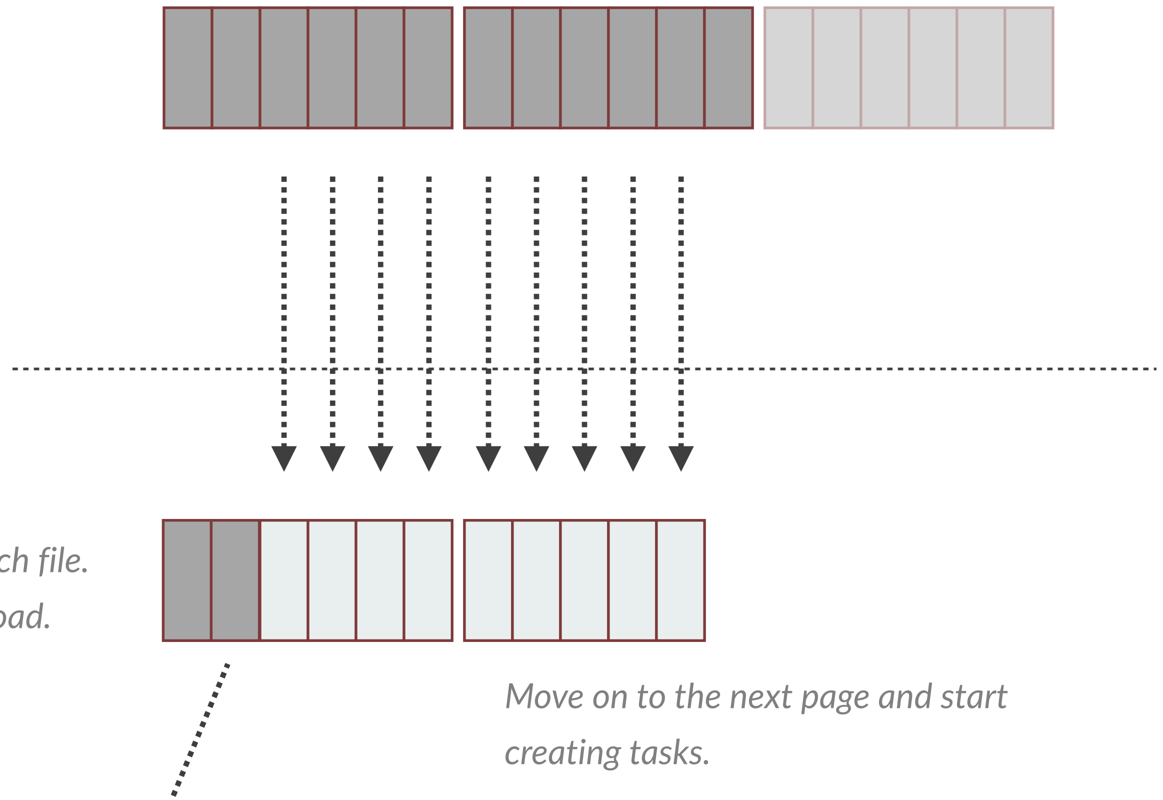


Create an `asyncio.Task` for each file.
This immediately starts the download.

Move on to the next page and start creating tasks.

Meanwhile tasks from the first page will finish downloading their file.

Asyncio

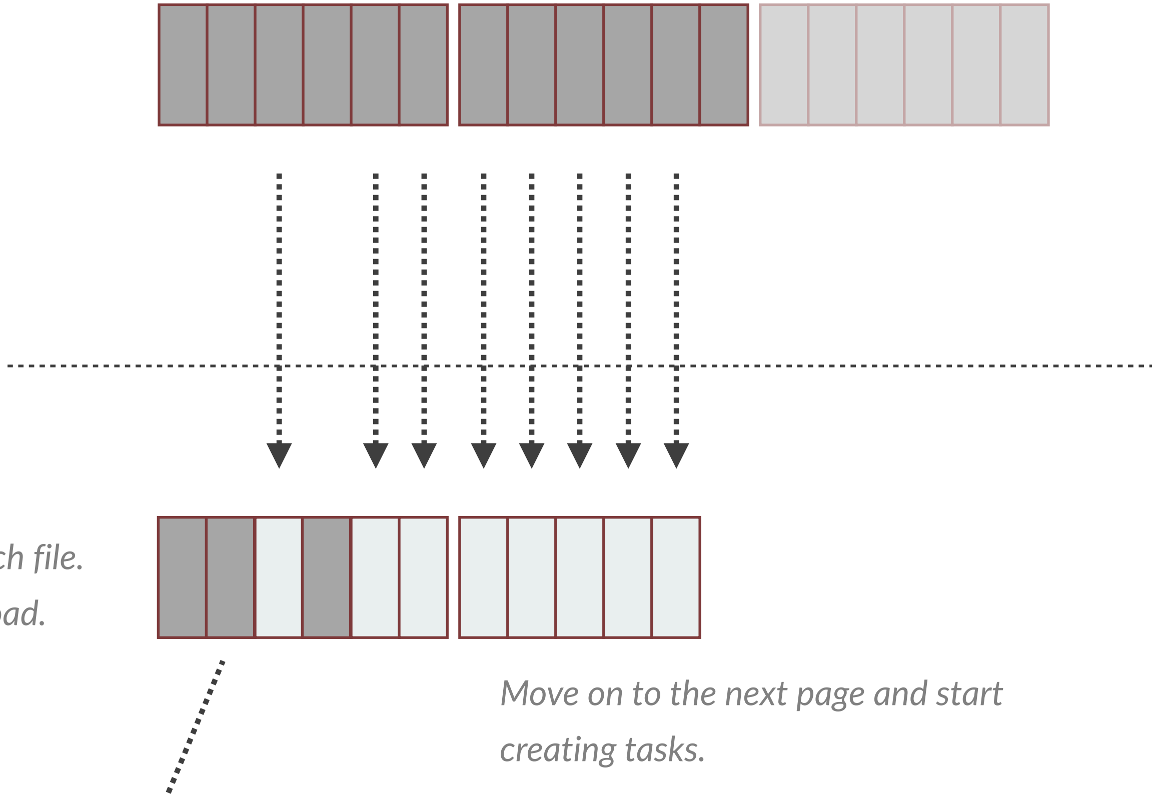


Create an `asyncio.Task` *for each file.*
This immediately starts the download.

Move on to the next page and start creating tasks.

Meanwhile tasks from the first page will finish downloading their file.

Asyncio

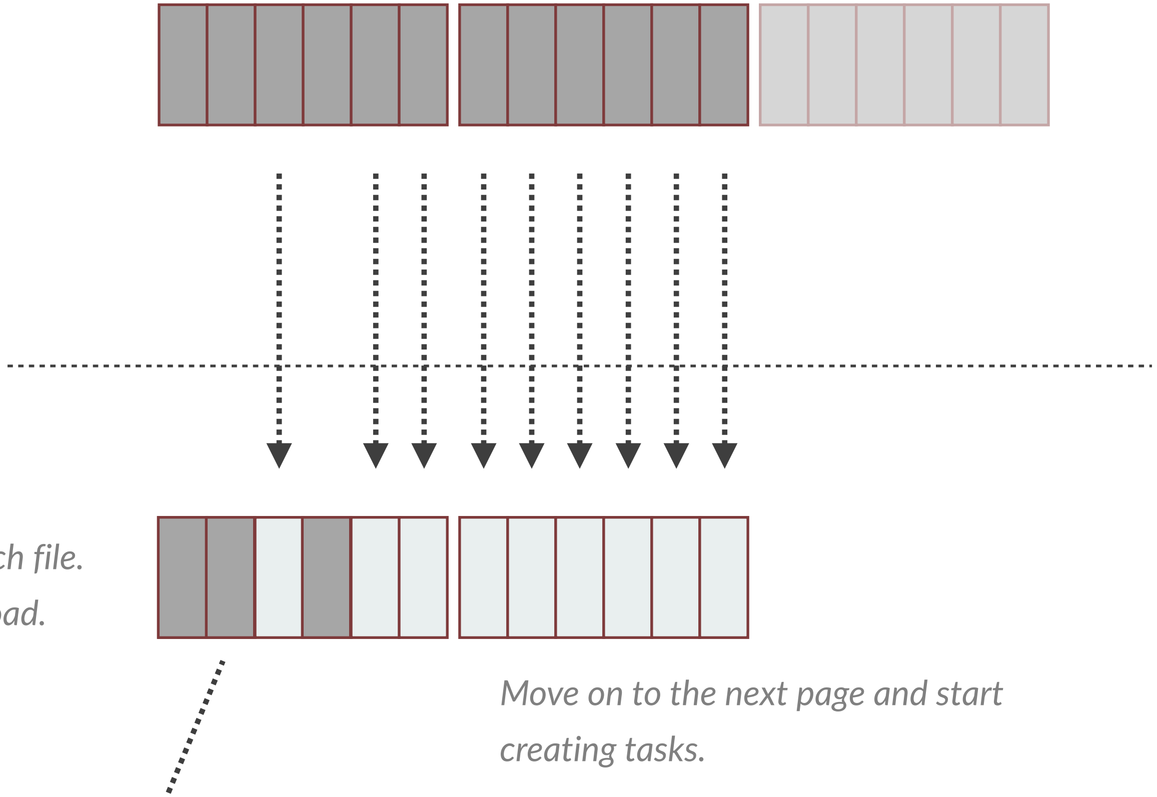


Create an `asyncio.Task` *for each file.*
This immediately starts the download.

Move on to the next page and start
creating tasks.

Meanwhile tasks from the first page
will finish downloading their file.

Asyncio

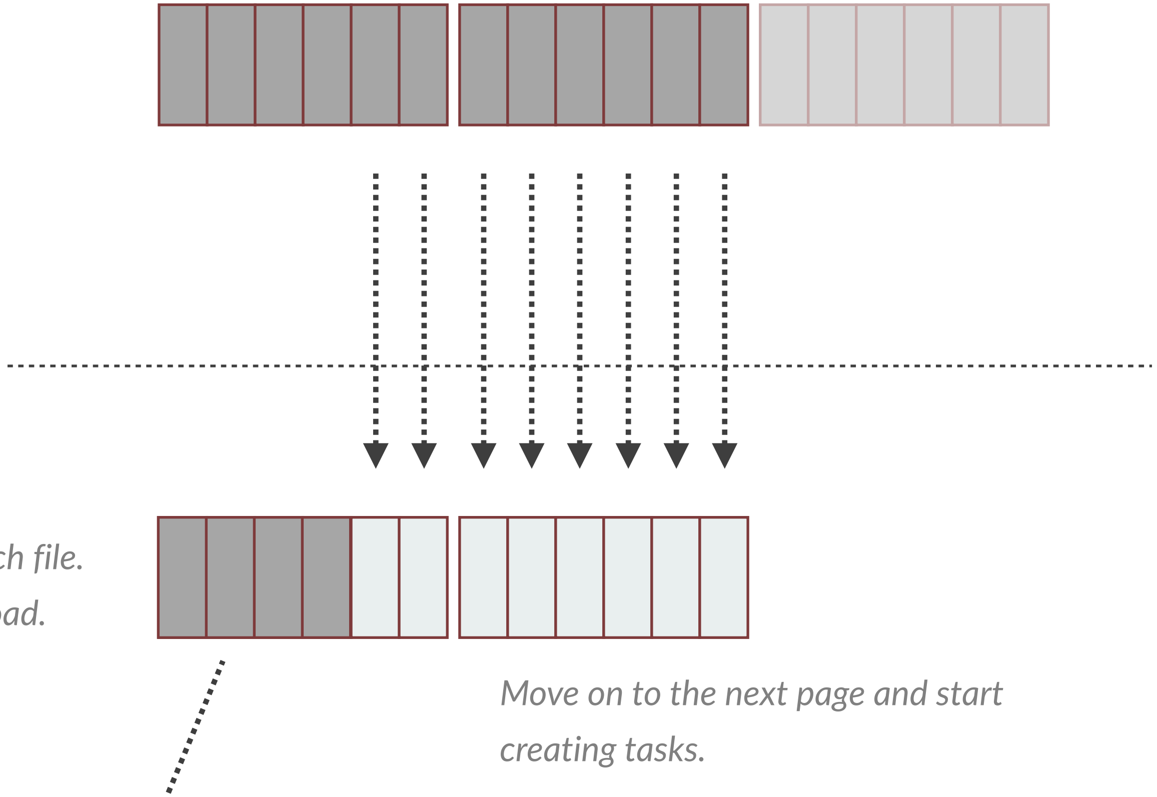


Create an `asyncio.Task` *for each file.*
This immediately starts the download.

Move on to the next page and start creating tasks.

Meanwhile tasks from the first page will finish downloading their file.

Asyncio

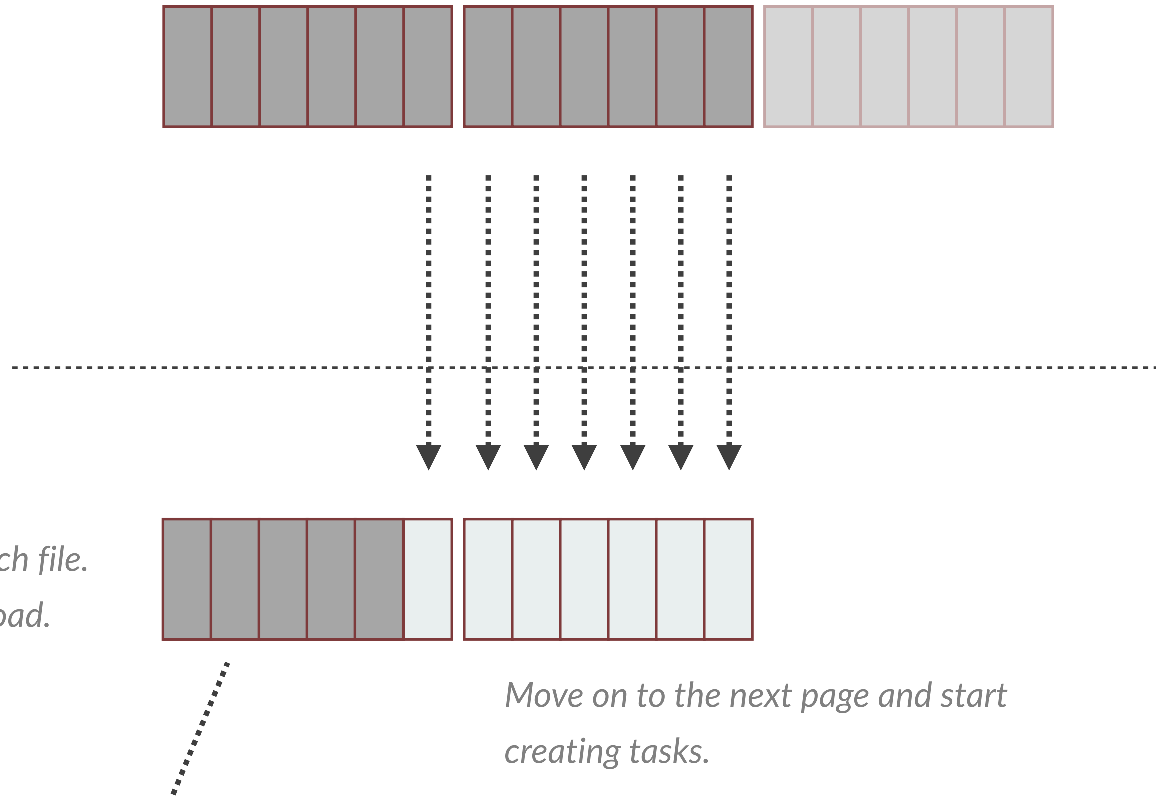


Create an `asyncio.Task` *for each file. This immediately starts the download.*

Move on to the next page and start creating tasks.

Meanwhile tasks from the first page will finish downloading their file.

Asyncio

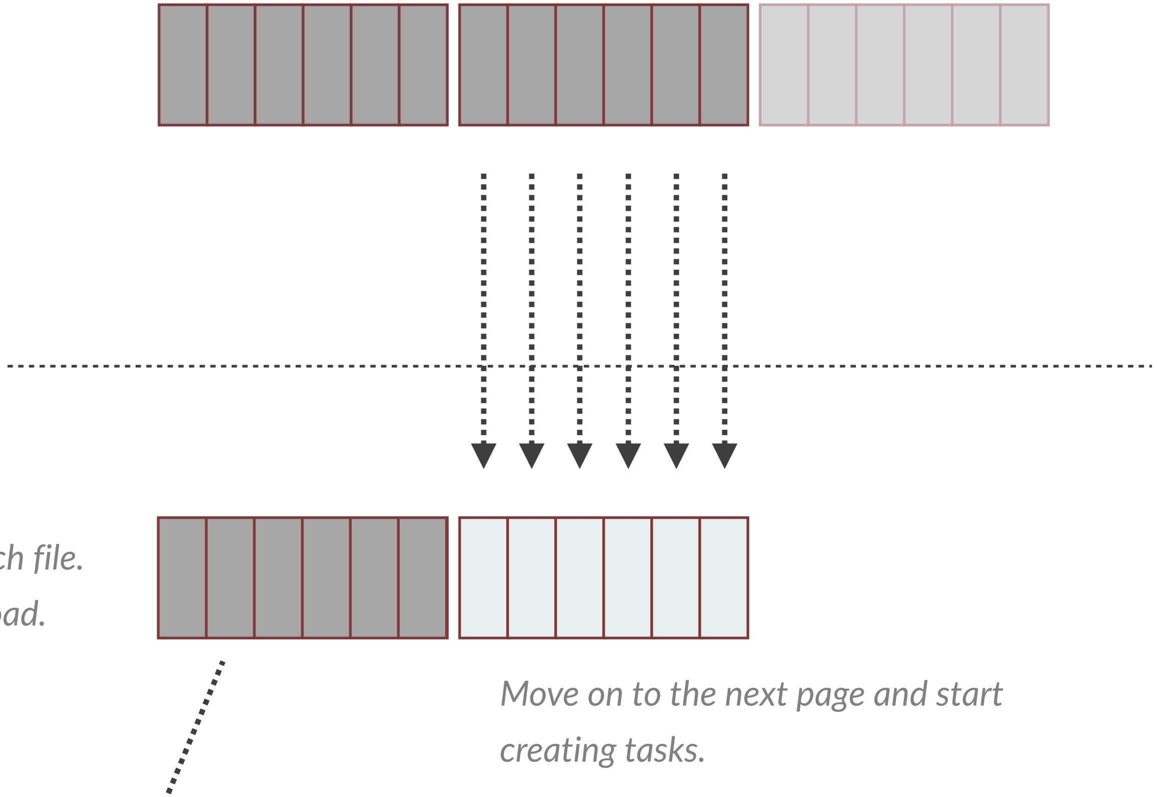


Create an `asyncio.Task` *for each file.*
This immediately starts the download.

Move on to the next page and start creating tasks.

Meanwhile tasks from the first page will finish downloading their file.

Asyncio

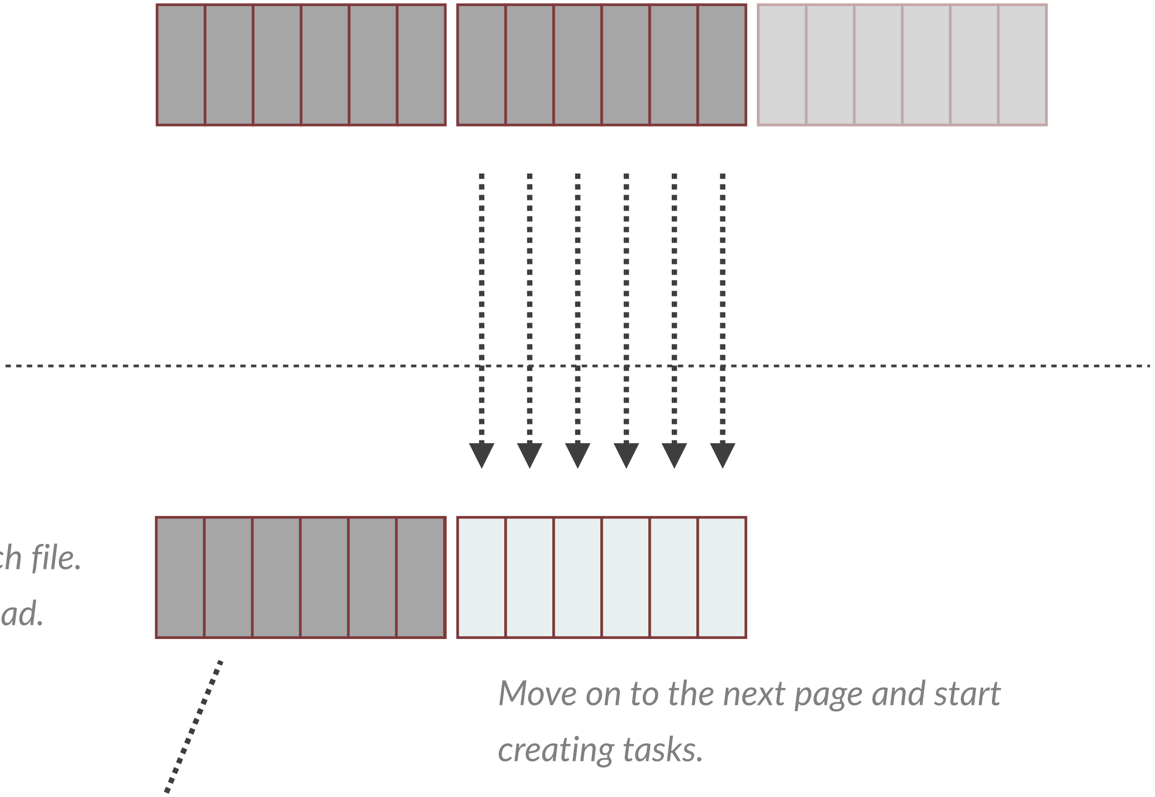


Create an `asyncio.Task` *for each file.*
This immediately starts the download.

Move on to the next page and start creating tasks.

Meanwhile tasks from the first page will finish downloading their file.

Asyncio

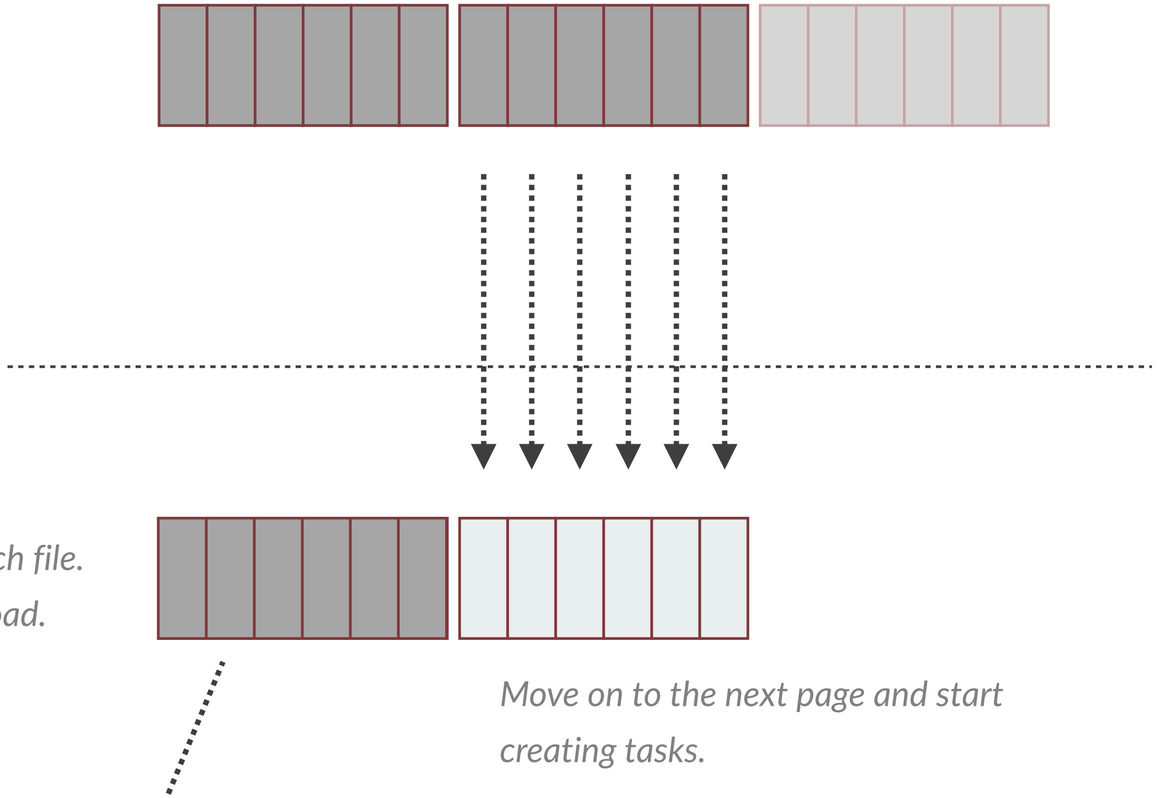


Create an `asyncio.Task` *for each file.*
This immediately starts the download.

Move on to the next page and start creating tasks.

Meanwhile tasks from the first page will finish downloading their file.

Asyncio



Create an `asyncio.Task` for each file.
This immediately starts the download.

Move on to the next page and start creating tasks.

Meanwhile tasks from the first page will finish downloading their file.

All in a single process

All in a single thread

Switch tasks when waiting for IO

Should keep CPU busy

```python
import asyncio
from aiohttp import ClientSession
import uvloop

async def download_files(host, port, outdir):
    hostname = f'http://{host}:{port}'
    list_url = f'{hostname}/list'
    get_url = f'{hostname}/get'
    semaphore = asyncio.Semaphore(MAX_CONCURRENT)
    task_queue = asyncio.Queue(MAX_SIZE)
    asyncio.create_task(results_worker(task_queue))
    async with ClientSession() as session:
        async for filename in iter_all_files(session, list_url):
            remote_url = f'{get_url}/{filename}'
            task = asyncio.create_task(
                download_file(session, semaphore, remote_url,
                              os.path.join(outdir, filename))
            )
            await task_queue.put(task)
```

```python
import asyncio
from aiohttp import ClientSession
import uvloop

async def download_files(host, port, outdir):
    hostname = f'http://{host}:{port}'
    list_url = f'{hostname}/list'
    get_url = f'{hostname}/get'
    semaphore = asyncio.Semaphore(MAX_CONCURRENT)
    task_queue = asyncio.Queue(MAX_SIZE)
    asyncio.create_task(results_worker(task_queue))
    async with ClientSession() as session:
        async for filename in iter_all_files(session, list_url):
            remote_url = f'{get_url}/{filename}'
            task = asyncio.create_task(
                download_file(session, semaphore, remote_url,
                              os.path.join(outdir, filename))
            )
            await task_queue.put(task)
```

```python
import asyncio
from aiohttp import ClientSession
import uvloop

async def download_files(host, port, outdir):
    hostname = f'http://{host}:{port}'
    list_url = f'{hostname}/list'
    get_url = f'{hostname}/get'
    semaphore = asyncio.Semaphore(MAX_CONCURRENT)
    task_queue = asyncio.Queue(MAX_SIZE)
    asyncio.create_task(results_worker(task_queue))
    async with ClientSession() as session:
        async for filename in iter_all_files(session, list_url):
            remote_url = f'{get_url}/{filename}'
            task = asyncio.create_task(
                download_file(session, semaphore, remote_url,
                              os.path.join(outdir, filename))
            )
            await task_queue.put(task)
```

```python
import asyncio
from aiohttp import ClientSession
import uvloop

async def download_files(host, port, outdir):
    hostname = f'http://{host}:{port}'
    list_url = f'{hostname}/list'
    get_url = f'{hostname}/get'
    semaphore = asyncio.Semaphore(MAX_CONCURRENT)
    task_queue = asyncio.Queue(MAX_SIZE)
    asyncio.create_task(results_worker(task_queue))
    async with ClientSession() as session:
        async for filename in iter_all_files(session, list_url):
            remote_url = f'{get_url}/{filename}'
            task = asyncio.create_task(
                download_file(session, semaphore, remote_url,
                              os.path.join(outdir, filename))
            )
            await task_queue.put(task)
```

```python
import asyncio
from aiohttp import ClientSession
import uvloop

async def download_files(host, port, outdir):
    hostname = f'http://{host}:{port}'
    list_url = f'{hostname}/list'
    get_url = f'{hostname}/get'
    semaphore = asyncio.Semaphore(MAX_CONCURRENT)
    task_queue = asyncio.Queue(MAX_SIZE)
    asyncio.create_task(results_worker(task_queue))
    async with ClientSession() as session:
        async for filename in iter_all_files(session, list_url):
            remote_url = f'{get_url}/{filename}'
            task = asyncio.create_task(
                download_file(session, semaphore, remote_url,
                              os.path.join(outdir, filename))
            )
            await task_queue.put(task)
```

```
async def iter_all_files(session, list_url):
    async with session.get(list_url) as response:
        if response.status != 200:
            raise RuntimeError(f"Bad status code: {response.status}")
        content = json.loads(await response.read())
    while True:
        for filename in content['FileNames']:
            yield filename
        if 'NextFile' not in content:
            return
        next_page_url = f'{list_url}?next-marker={content["NextFile"]}'
        async with session.get(next_page_url) as response:
            if response.status != 200:
                raise RuntimeError(f"Bad status code: {response.status}")
            content = json.loads(await response.read())
```

```python
async def iter_all_files(session, list_url):
    async with session.get(list_url) as response:
        if response.status != 200:
            raise RuntimeError(f"Bad status code: {response.status}")
        content = json.loads(await response.read())
    while True:
        for filename in content['FileNames']:
            yield filename
        if 'NextFile' not in content:
            return
        next_page_url = f'{list_url}?next-marker={content["NextFile"]}'
        async with session.get(next_page_url) as response:
            if response.status != 200:
                raise RuntimeError(f"Bad status code: {response.status}")
            content = json.loads(await response.read())
```

```python
async def download_file(session, semaphore, remote_url, local_filename):
    async with semaphore:
        async with session.get(remote_url) as response:
            contents = await response.read()
            # Sync version.
            with open(local_filename, 'wb') as f:
                f.write(contents)
            return local_filename
```

```python
async def download_file(session, semaphore, remote_url, local_filename):
    async with semaphore:
        async with session.get(remote_url) as response:
            contents = await response.read()
            # Sync version.
            with open(local_filename, 'wb') as f:
                f.write(contents)
            return local_filename
```

Asyncio Results

Asyncio Results

One request	0.00056 seconds

Asyncio Results

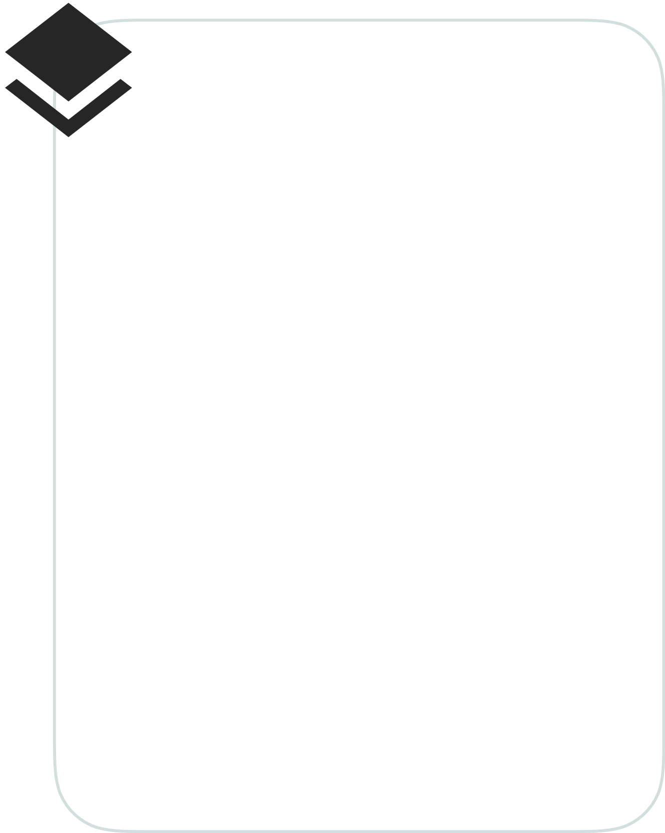
One request	0.00056 seconds
One billion requests	560,000 seconds 155.55 hours 6.48 days

Summary

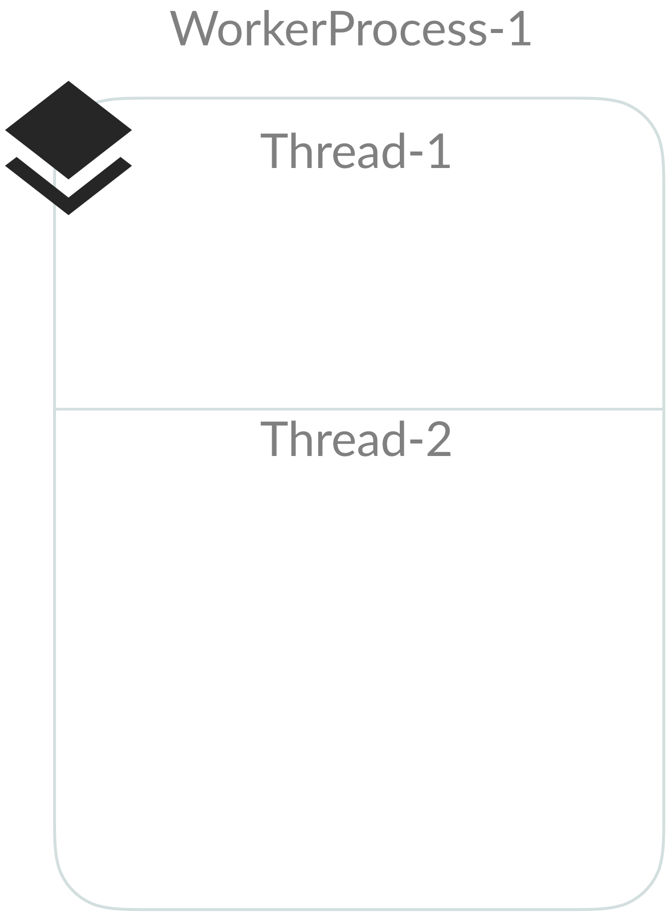
Approach	Single Request Time (s)	Days
Synchronous	0.003	34.7
Multithread	0.0036	41.6
Multiprocess	0.00032	3.7
Asyncio	0.00056	6.5

Asyncio and Multiprocessing

Asyncio and Multiprocessing and Multithreading



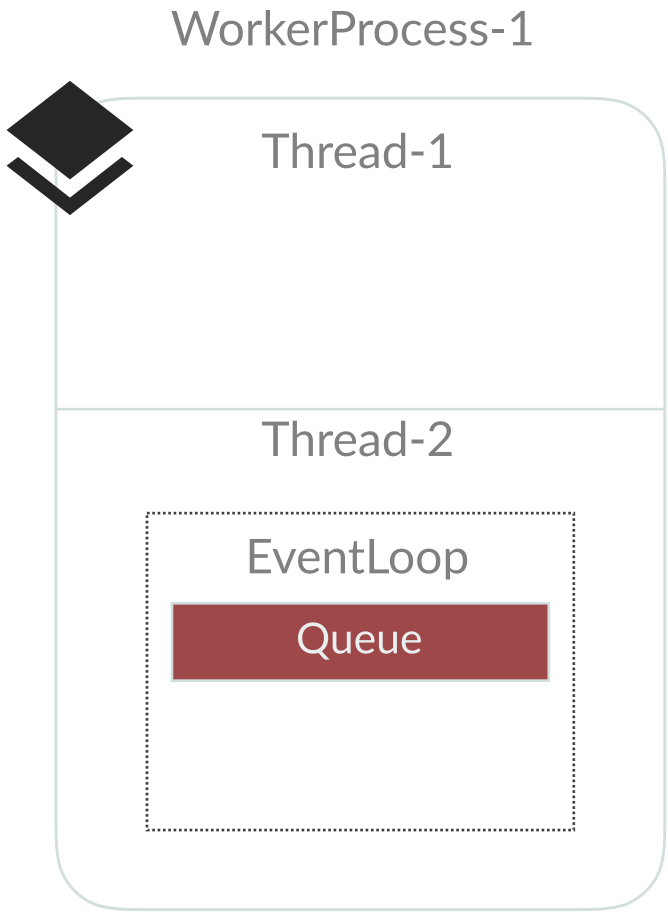
WorkerProcess-1



WorkerProcess-1

Thread-1

Thread-2



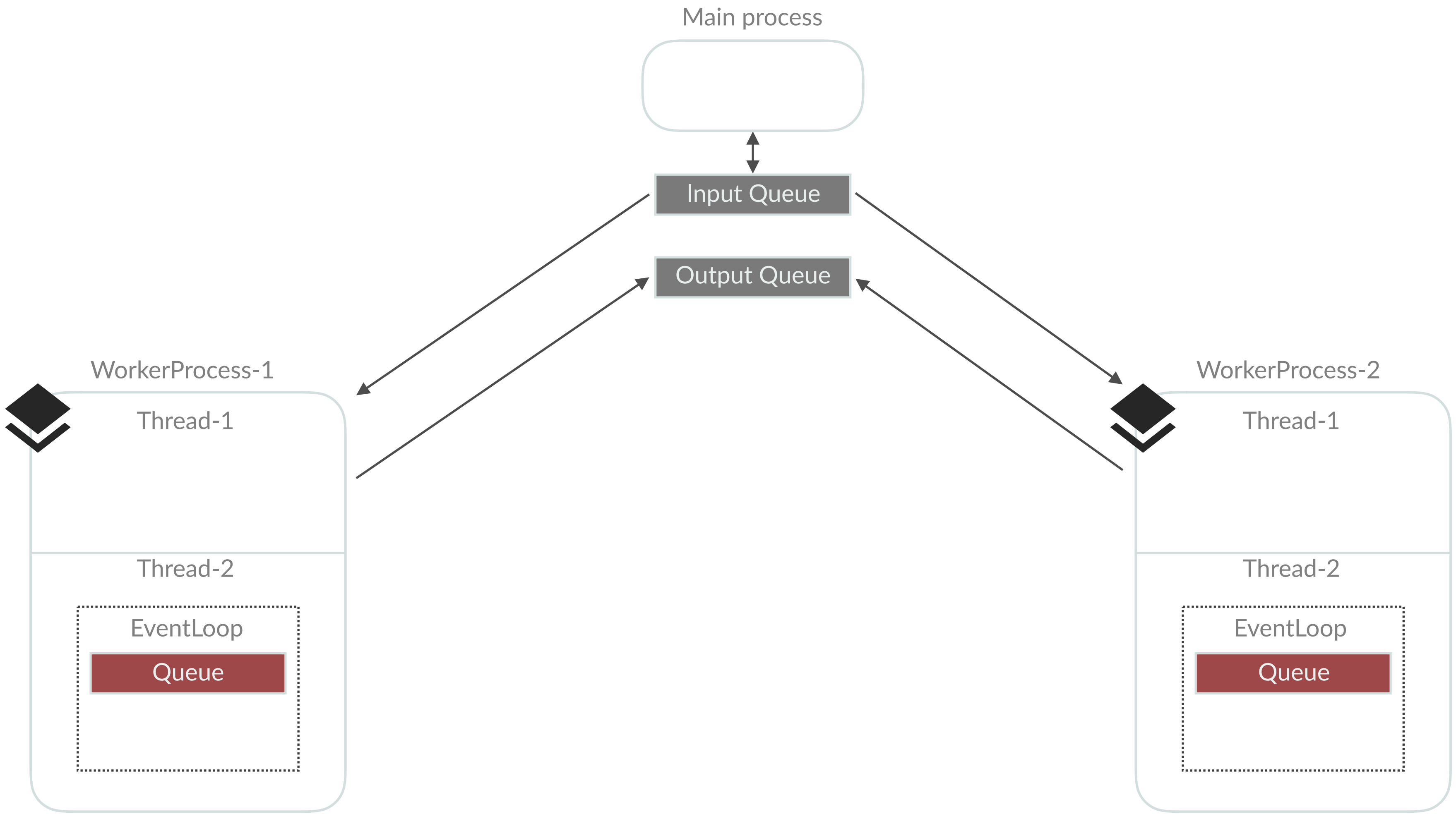
WorkerProcess-1

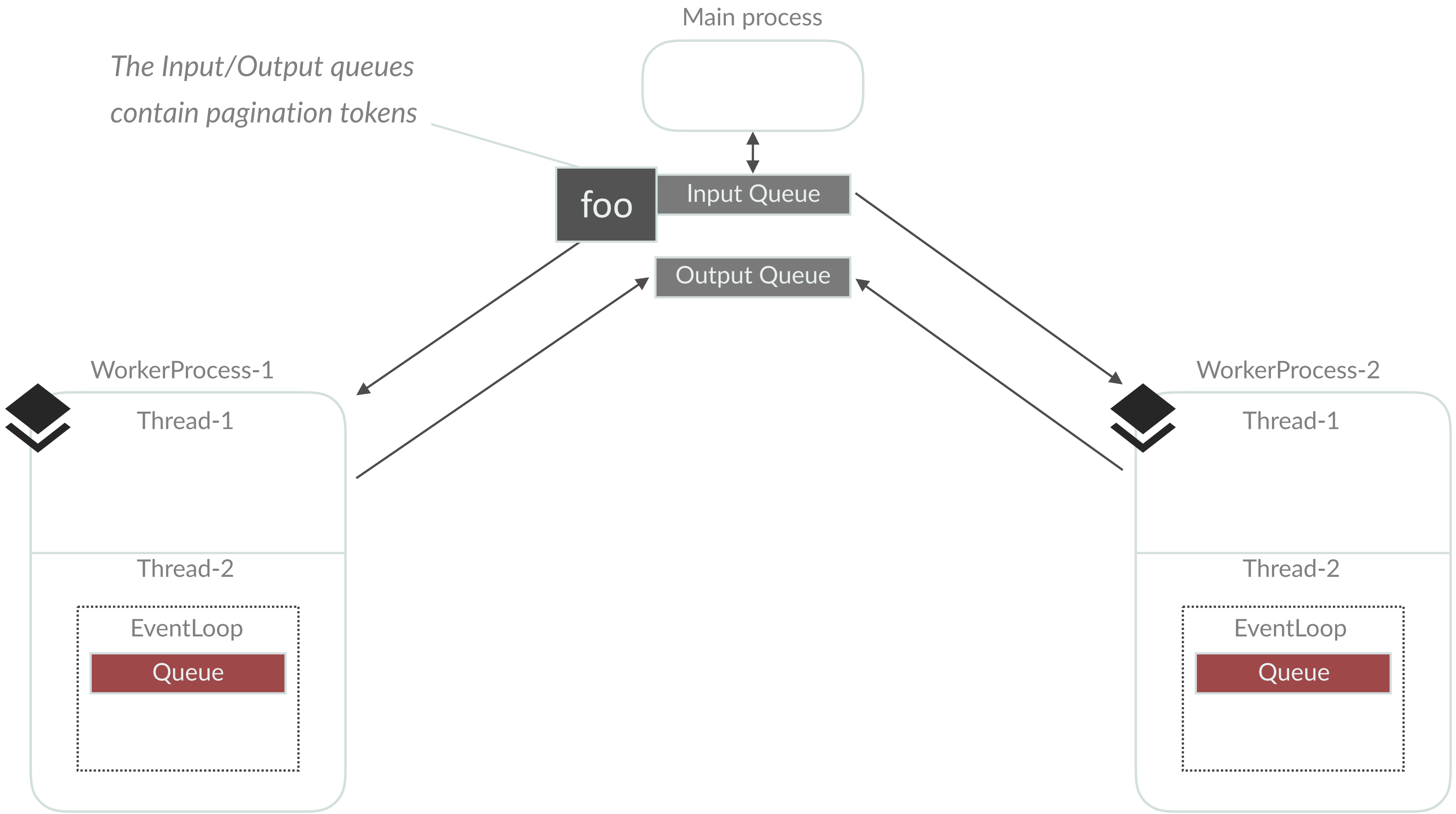
Thread-1

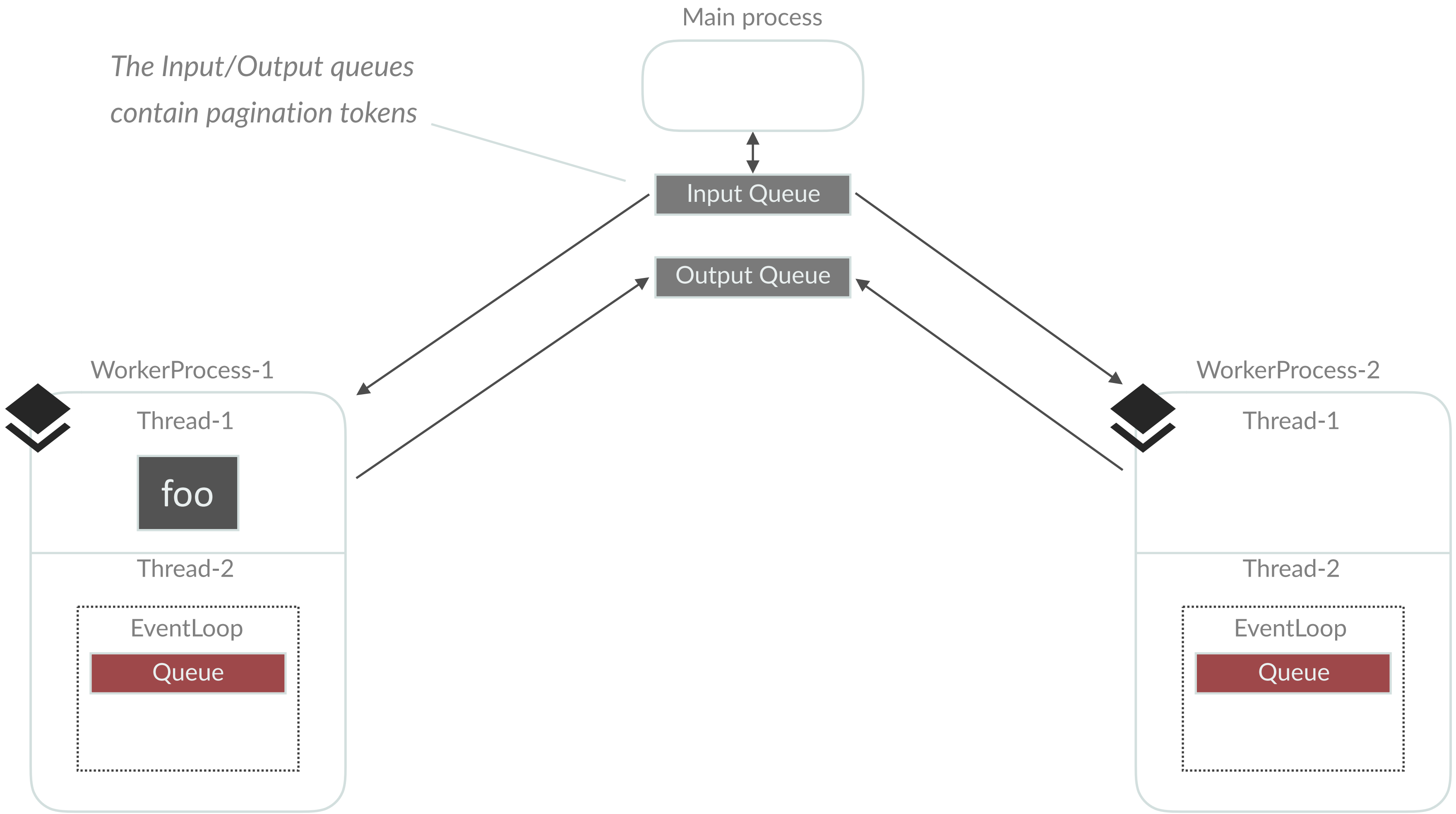
Thread-2

EventLoop

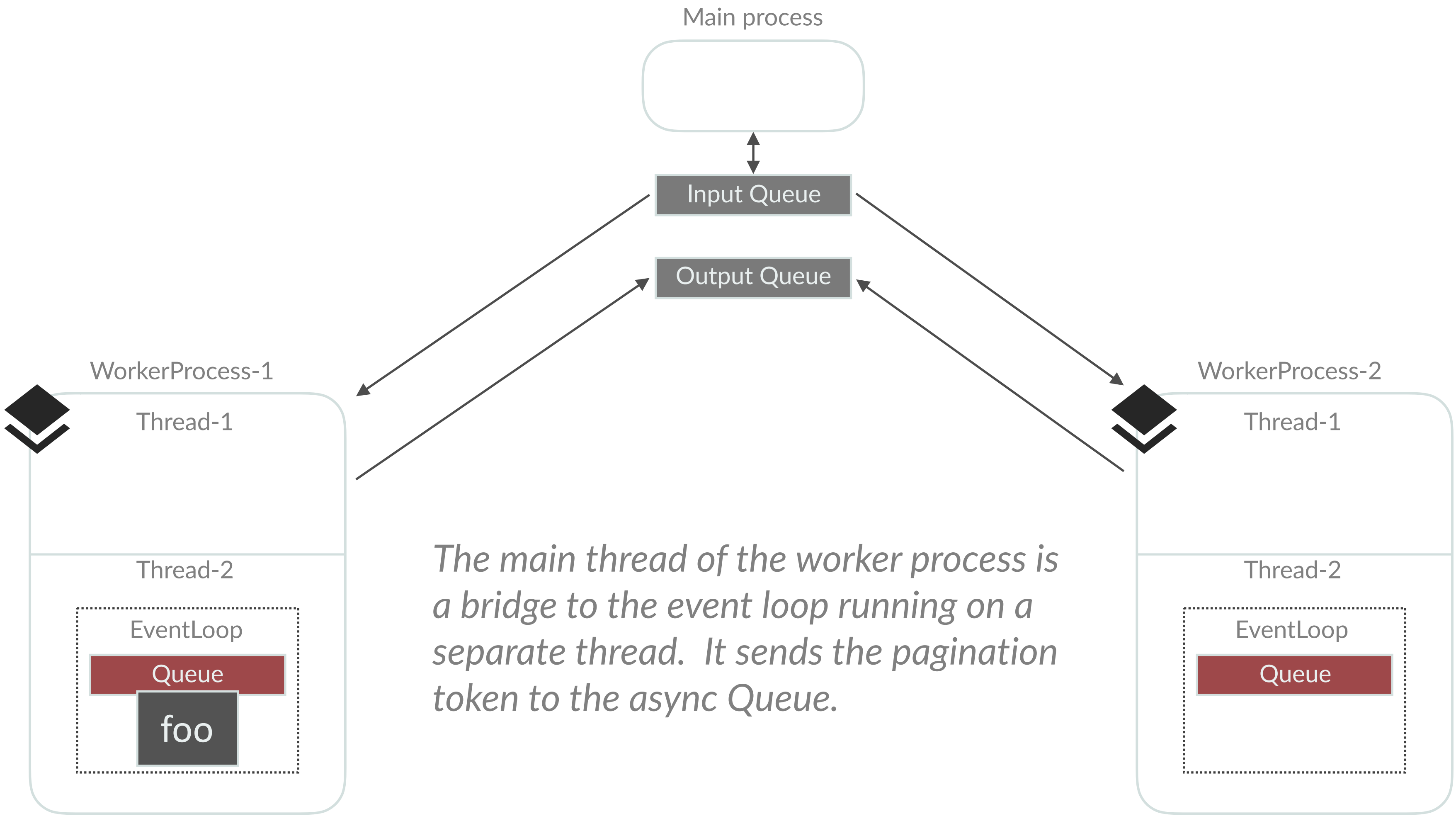
Queue


Main process
Input Queue
Output Queue
WorkerProcess-1
Thread-1
Thread-2
EventLoop
Queue
WorkerProcess-2
Thread-1
Thread-2
EventLoop
Queue

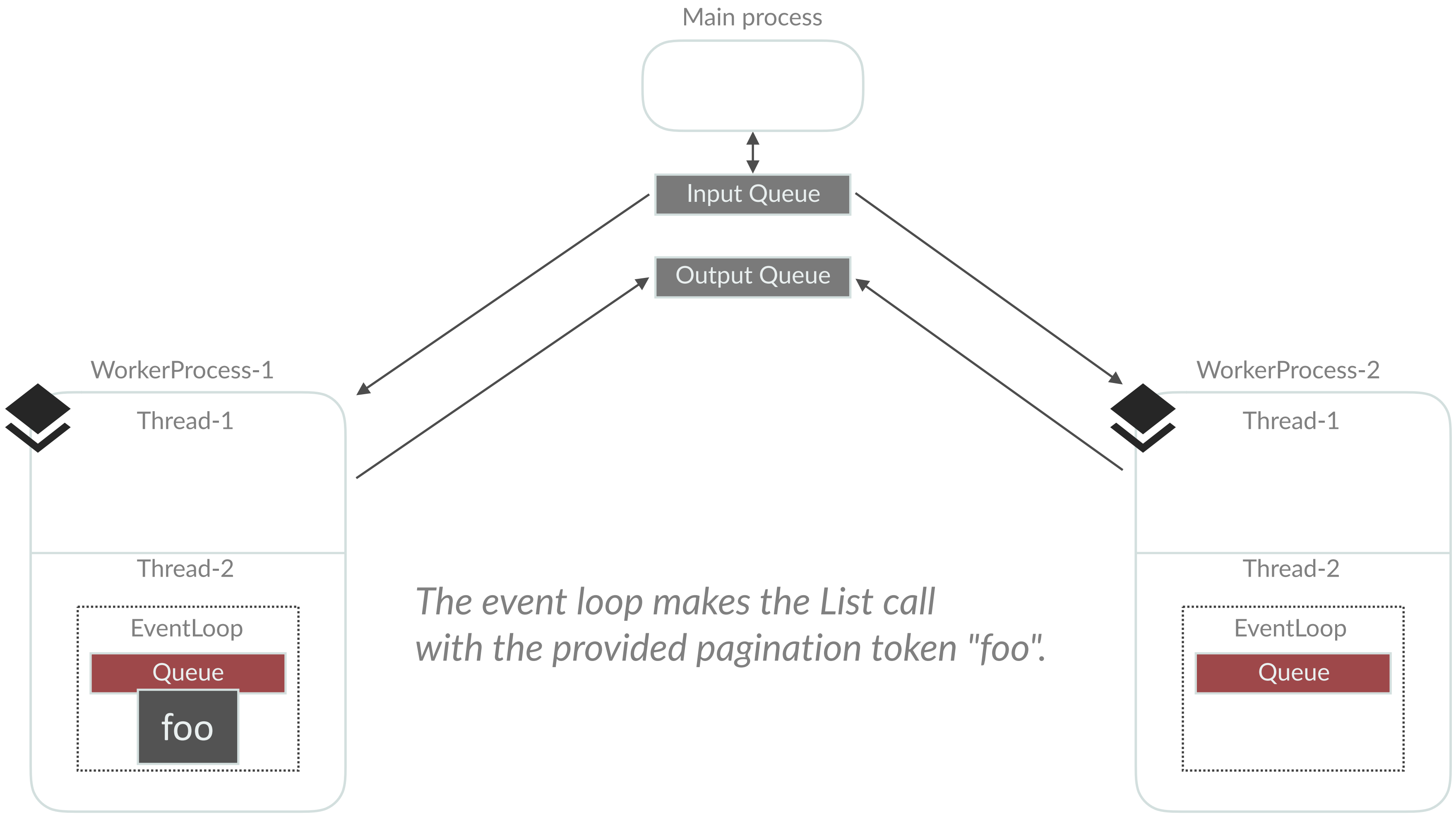

Main process
The Input/Output queues
contain pagination tokens
foo
Input Queue
Output Queue
WorkerProcess-1
Thread-1
Thread-2
EventLoop
Queue
WorkerProcess-2
Thread-1
Thread-2
EventLoop
Queue



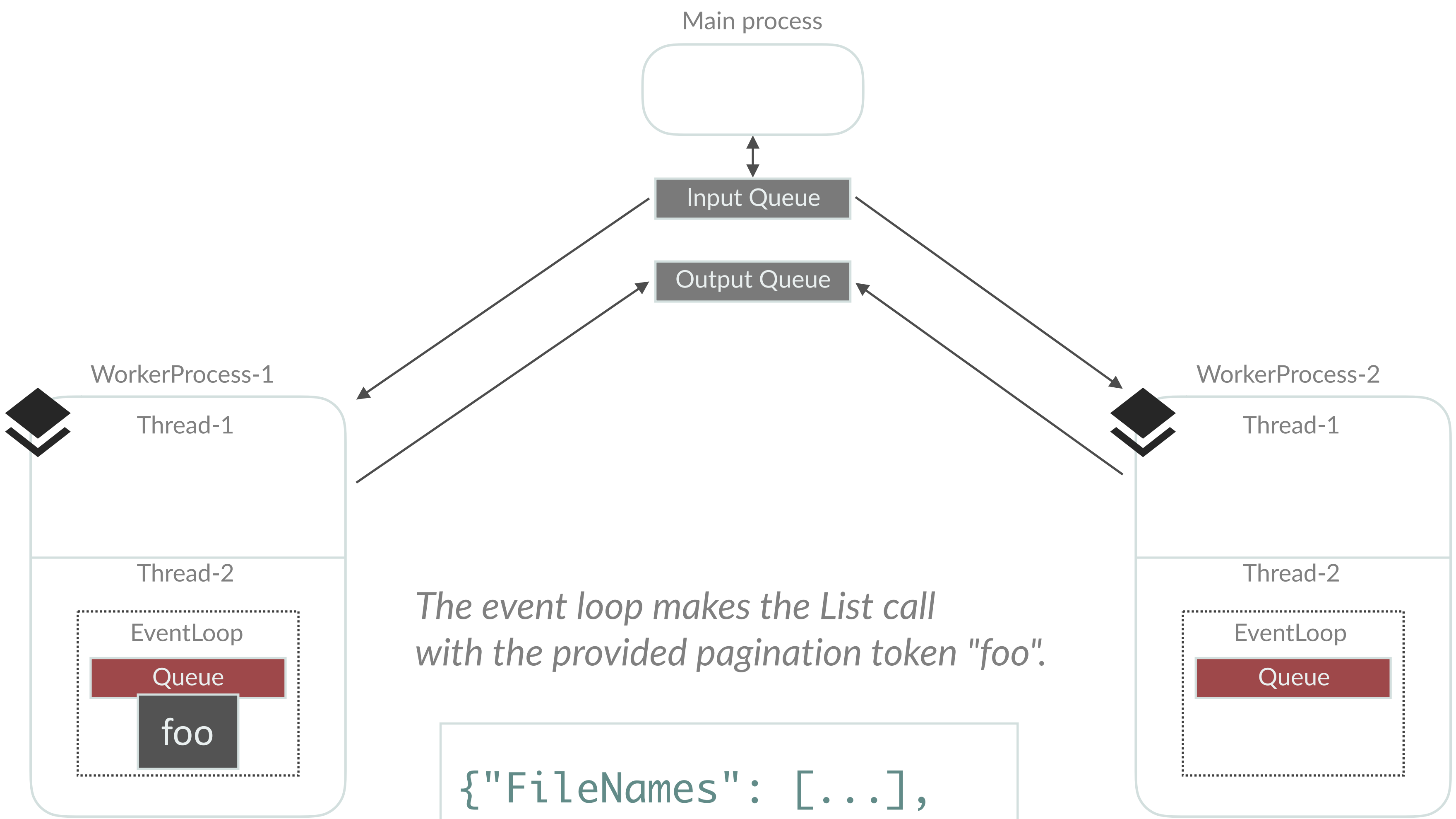
The Input/Output queues contain pagination tokens



The main thread of the worker process is a bridge to the event loop running on a separate thread. It sends the pagination token to the async Queue.

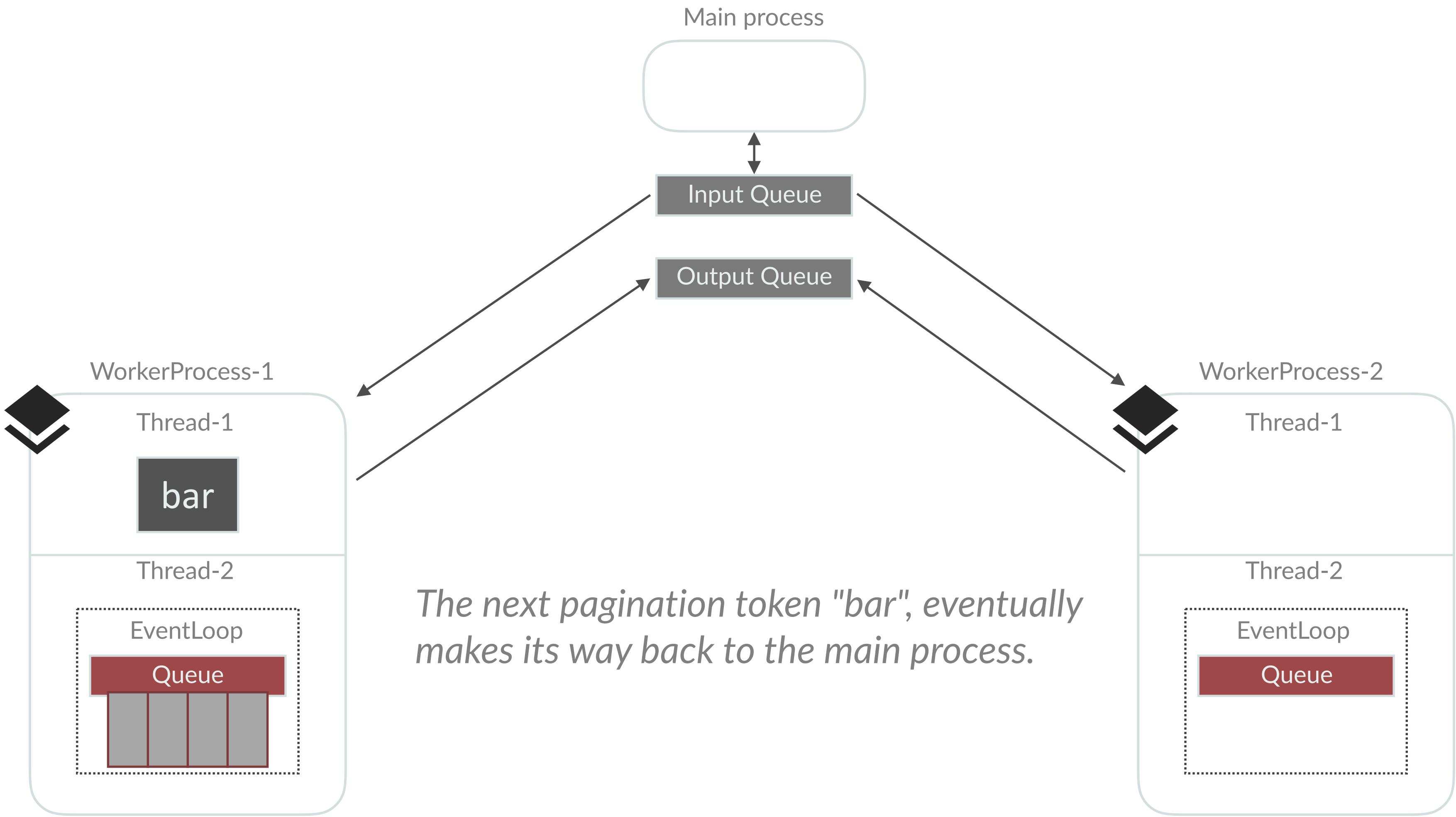


The event loop makes the List call with the provided pagination token "foo".

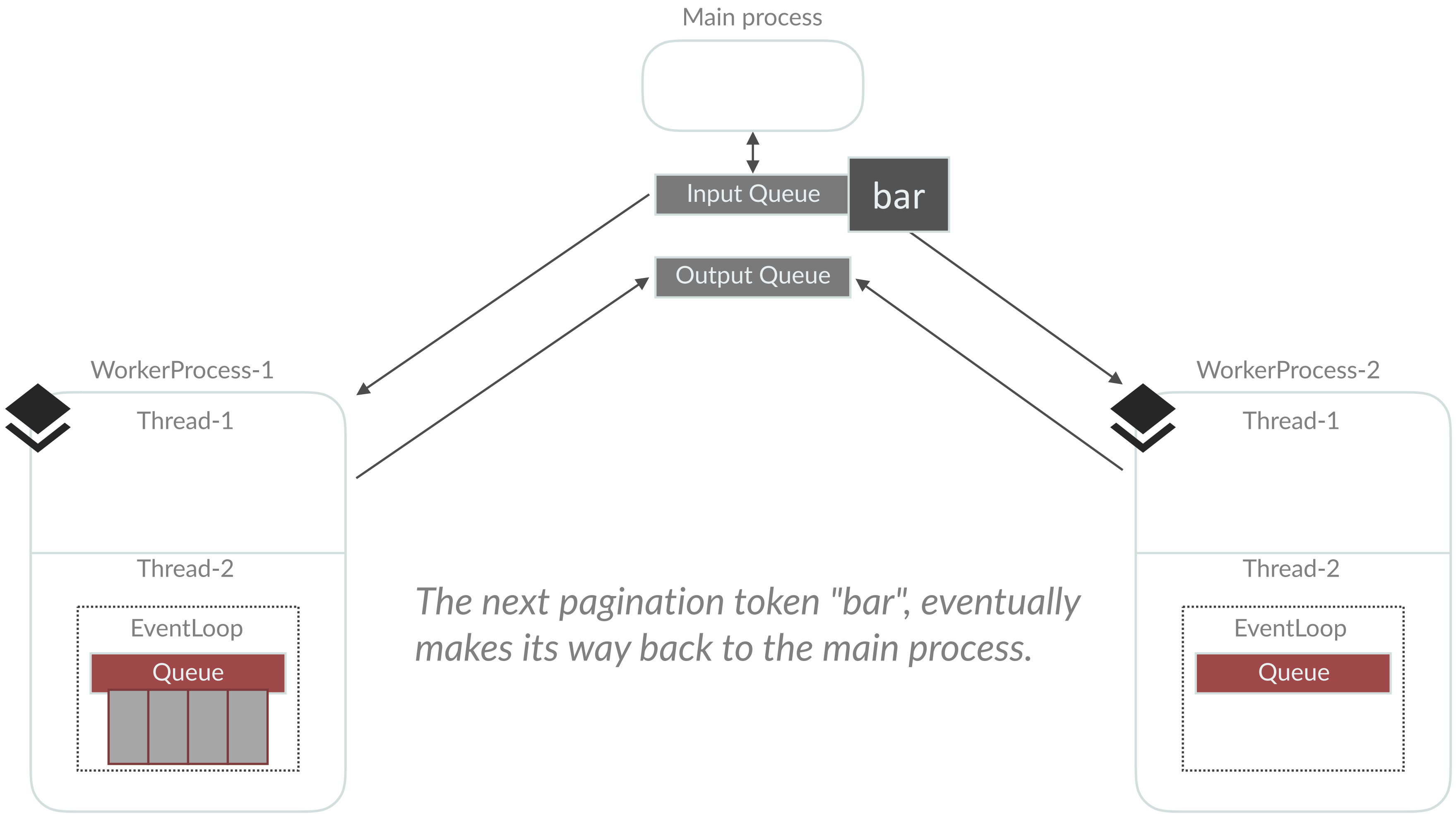


The event loop makes the List call with the provided pagination token "foo".

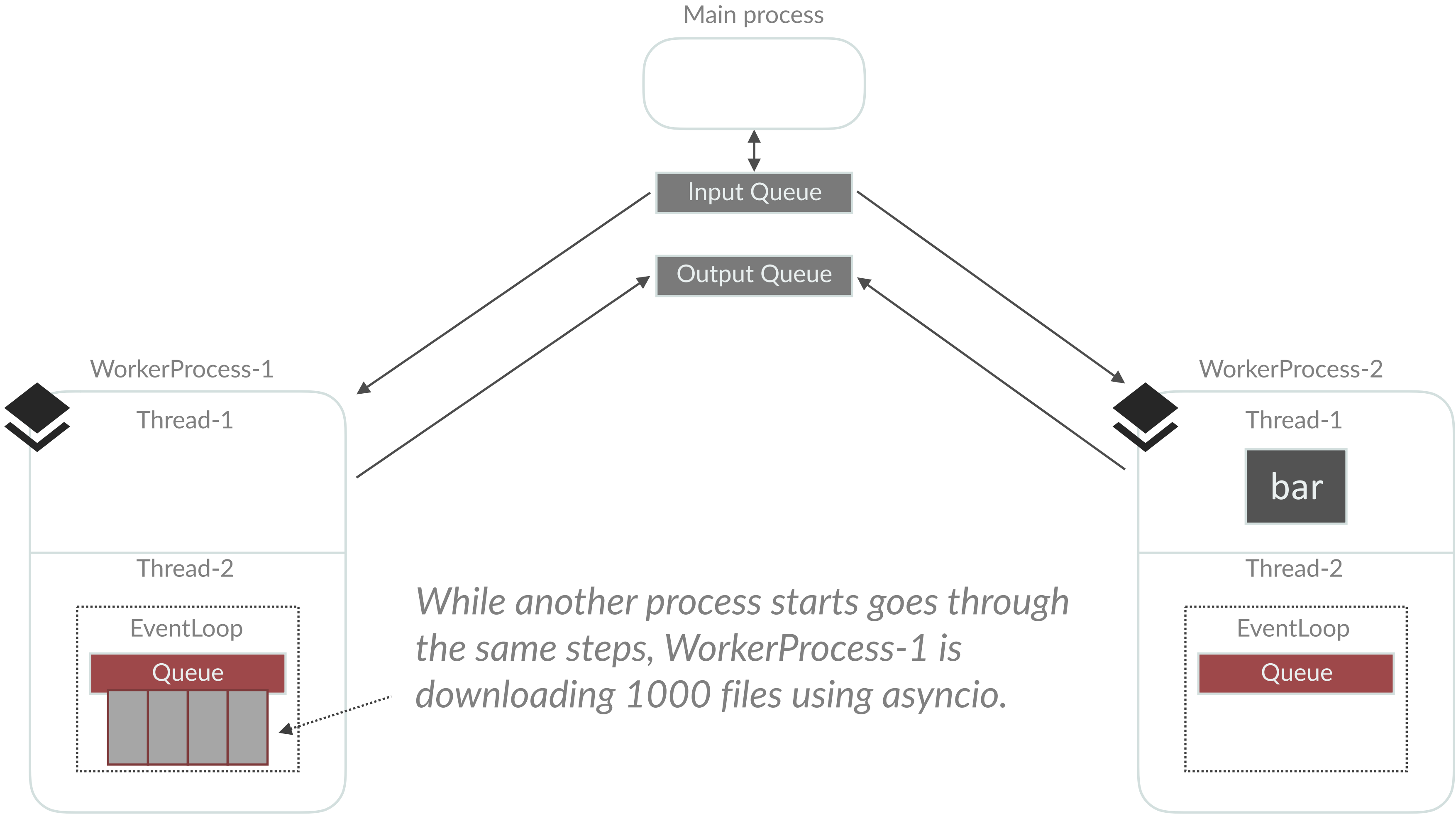
```
{"FileNames": [...],
 "NextMarker": "bar"}
```



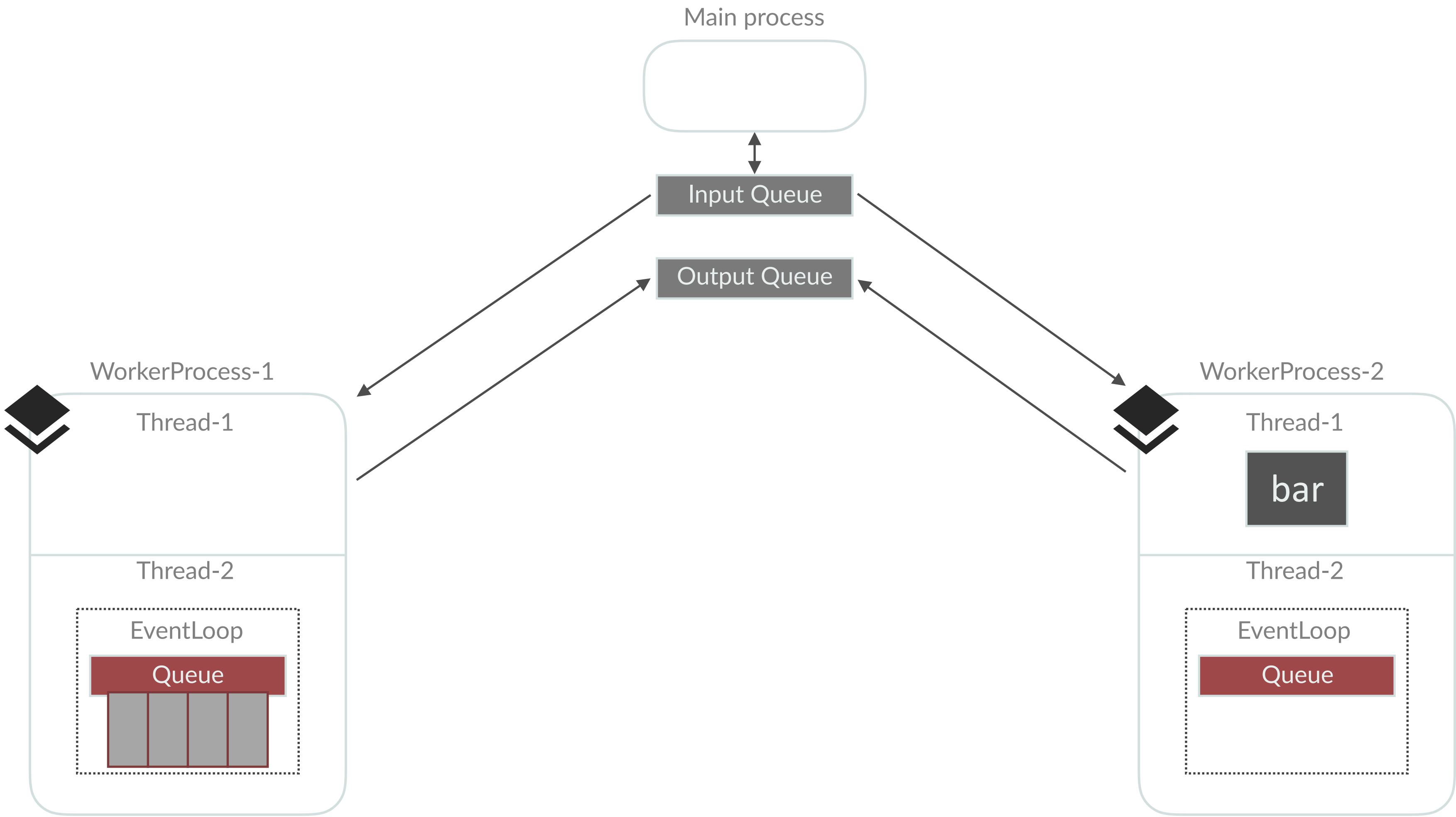
The next pagination token "bar", eventually makes its way back to the main process.



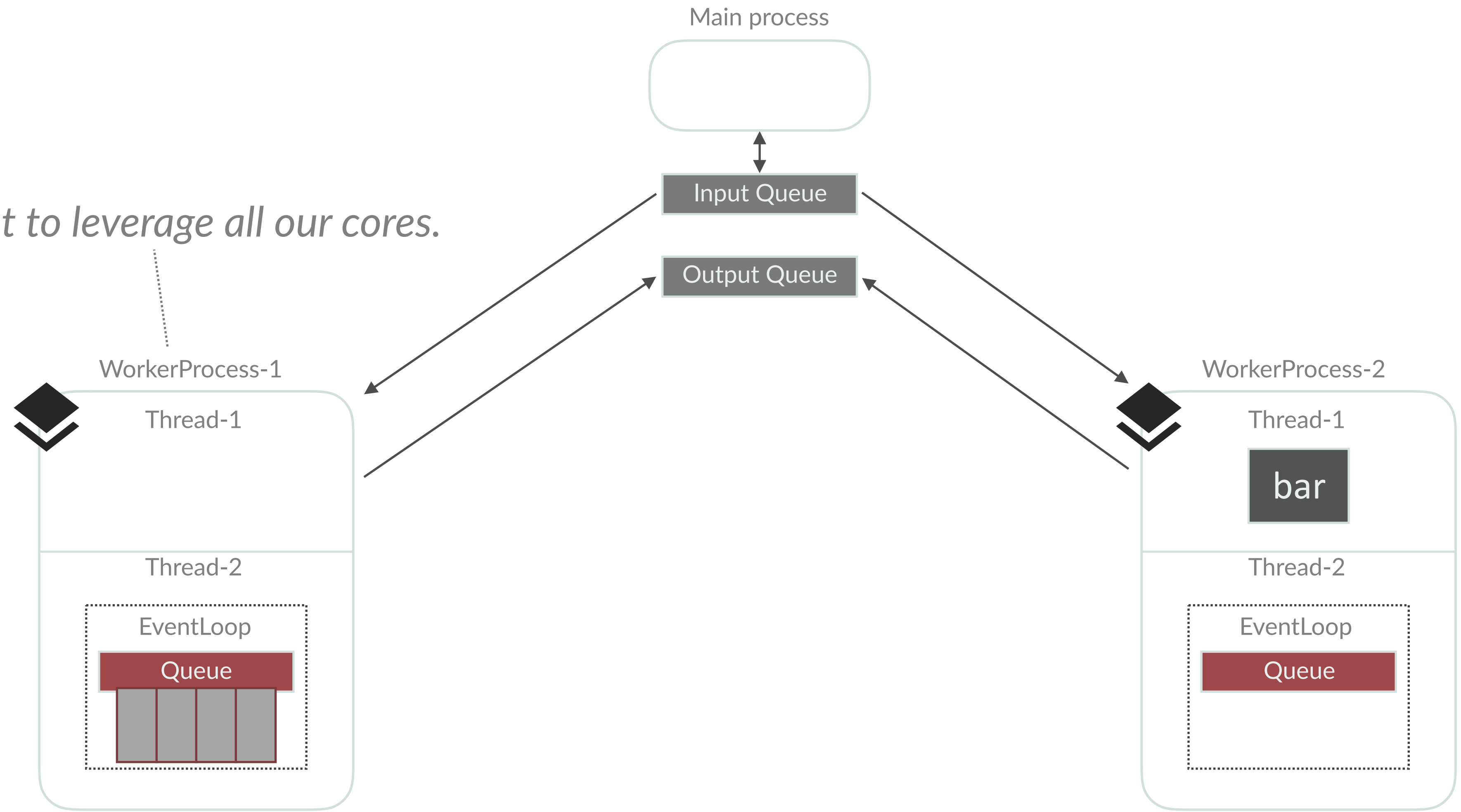
The next pagination token "bar", eventually makes its way back to the main process.

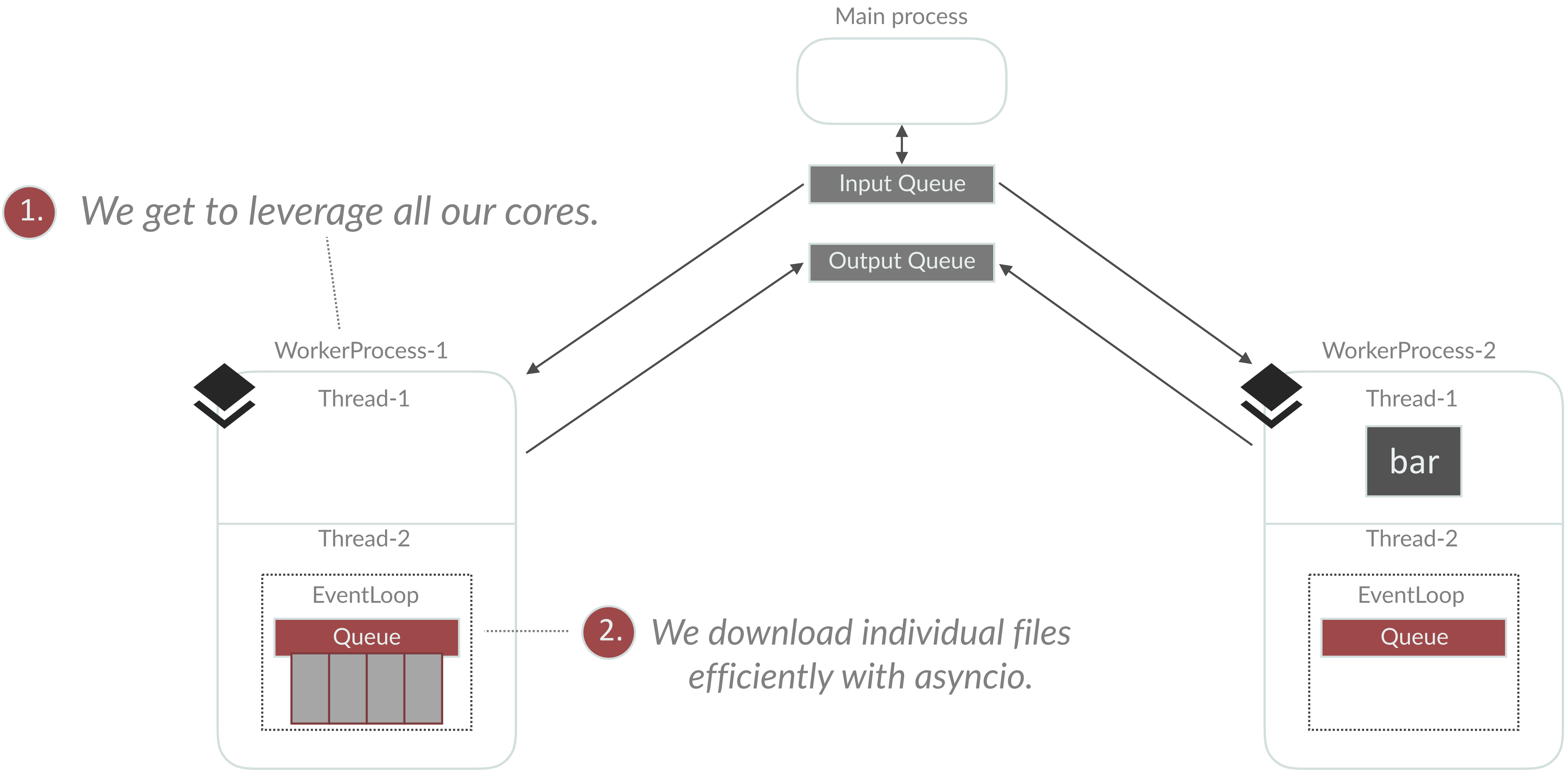


While another process starts goes through the same steps, WorkerProcess-1 is downloading 1000 files using asyncio.

Main process
Input Queue
Output Queue
WorkerProcess-1
Thread-1
Thread-2
EventLoop
Queue
WorkerProcess-2
Thread-1
bar
Thread-2
EventLoop
Queue

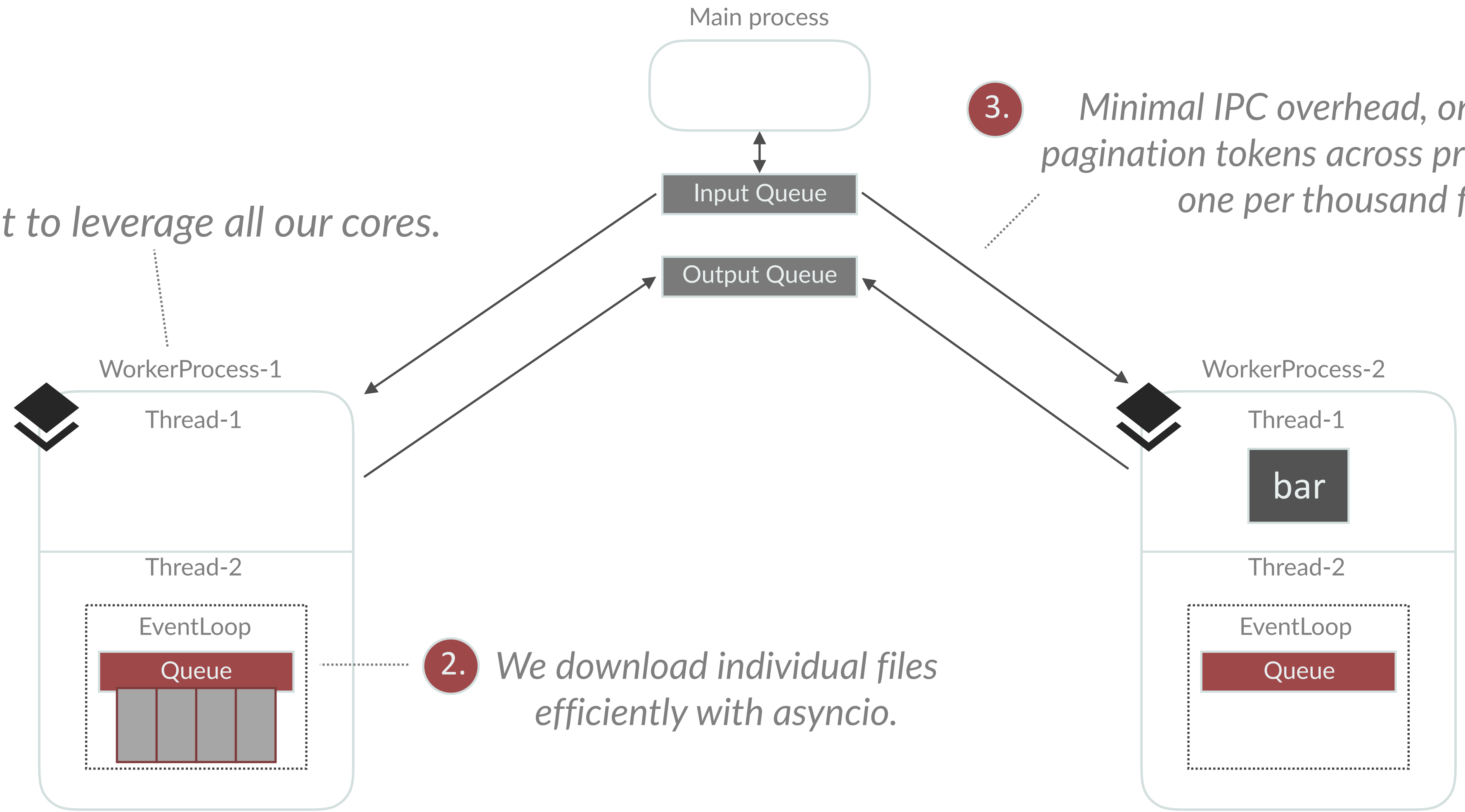
1. *We get to leverage all our cores.*





1. *We get to leverage all our cores.*

2. *We download individual files efficiently with asyncio.*



1. *We get to leverage all our cores.*

2. *We download individual files efficiently with asyncio.*

3. *Minimal IPC overhead, only passing pagination tokens across processes, only one per thousand files.*

Combo Results

Combo Results

One request	0.0000303 seconds

Combo Results

One request	0.0000303 seconds
One billion requests	30,300 seconds

Combo Results

One request	0.0000303 seconds
One billion requests	30,300 seconds 8.42 hours

Summary

Approach	Single Request Time (s)	Days
Synchronous	0.003	34.7
Multithread	0.0036	41.6
Multiprocess	0.00032	3.7
Asyncio	0.00056	6.5
Combo	0.0000303	0.35

Lessons Learned

Multiple orders of magnitude difference based on approach used

Tradeoff between simplicity and speed

Need to have max bounds when using queueing or any task scheduling

Thanks!

James Saryerwinnie

@jsaryer