Advanced asyncio

Solving Real-World Production Problems

Lynn Root | SRE | @roguelynn

Spotify

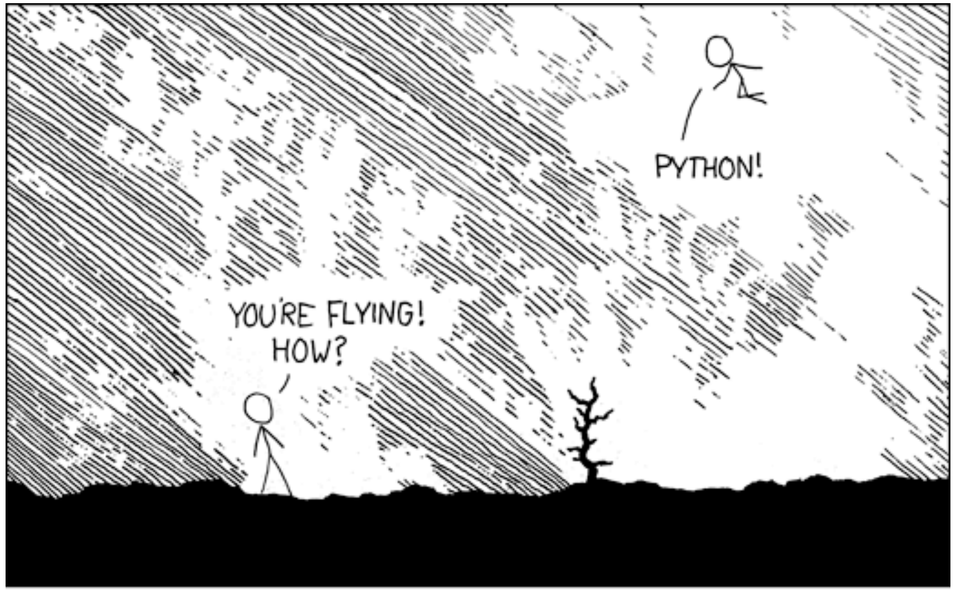
$ whoami_

agenda

- Initial setup of Mayhem Mandrill
- Development best practices
- Testing, debugging, and profiling

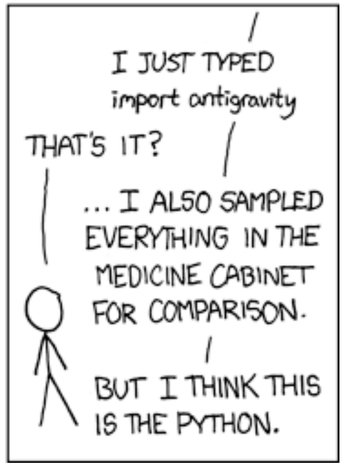
slides: ***rogue.ly/adv-aio***

async all the things


PYTHON!
YOU'RE FLYING! HOW?


I LEARNED IT LAST NIGHT! EVERYTHING IS SO SIMPLE!
HELLO WORLD IS JUST print "Hello, world!"


I DUNNO... DYNAMIC TYPING? WHITESPACE?
COME JOIN US! PROGRAMMING IS FUN AGAIN! IT'S A WHOLE NEW WORLD UP HERE!
BUT HOW ARE YOU FLYING?


I JUST TYPED import antigravity
THAT'S IT?
... I ALSO SAMPLED EVERYTHING IN THE MEDICINE CABINET FOR COMPARISON.
BUT I THINK THIS IS THE PYTHON.

```
Python 3.7.0 (default, Jul  6 2018, 11:30:06)
[Clang 9.1.0 (clang-902.0.39.2)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> import asyncio, datetime
>>> async def hello():
...     print(f'[{datetime.datetime.now()}] Hello...')
...     await asyncio.sleep(1)  # some I/O-intensive work
...     print(f'[{datetime.datetime.now()}] ...World!')
...
>>> asyncio.run(hello())
[2018-07-07 10:45:55.559856] Hello...
[2018-07-07 10:45:56.568737] ...World!
```

```
Python 3.7.0 (default, Jul  6 2018, 11:30:06)
[Clang 9.1.0 (clang-902.0.39.2)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> import asyncio, datetime
>>> async def hello():
...     print(f'[{datetime.datetime.now()}] Hello...')
...     await asyncio.sleep(1)  # some I/O intensive work
...     print(f'[{datetime.datetime.now()}] ...World!')
...
>>> asyncio.run(hello())
[2018-07-07 10:45:55.559856] Hello...
[2018-07-07 10:45:56.568737] ...World!
```

FAKE NEWS

```
Python 3.7.0 (default, Jul  6 2018, 11:30:06)
[Clang 9.1.0 (clang-902.0.39.2)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> import asyncio, datetime
>>> async def hello():
...     print(f'[{datetime.datetime.now()}] Hello...')
...     await asyncio.sleep(1)  # some I/O intensive work
...     print(f'[{datetime.datetime.now()}] ...World!')
...
>>> asyncio.run(hello())
[2018-07-07 10:45:55.559856] Hello...
[2018-07-07 10:45:56.568737] ...World!
```

FAKE NEWS

```
Python 3.7.0 (default, Jul  6 2018, 11:30:06)
[Clang 9.1.0 (clang-902.0.39.2)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> import asyncio, datetime
>>> async def hello():
...     print(f'[{datetime.datetime.now()}] Hello...')
...     await asyncio.sleep(1)  # some I/O intensive work
...     print(f'[{datetime.datetime.now()}] ...World!')
...
>>> asyncio.run(hello())
[2018-07-07 10:45:55.559856] Hello...
[2018-07-07 10:45:56.568737] ...World!
```

FAKE NEWS

building mayhem mandrill

initial setup

initial setup

concurrently publish messages

concurrently publish messages

```
async def publish(queue):
    choices = string.ascii_lowercase + string.digits
    while True:
        host_id = "".join(random.choices(choices, k=4))
        msg = Message(
           msg_id=str(uuid.uuid4()),
           inst_name=f"cattle-{host_id}"
        )

        asyncio.create_task(queue.put(msg))
        logging.info(f"Published {msg}")

        # simulate randomness of publishing messages
        await asyncio.sleep(random.random())
```

concurrently publish messages

```
async def publish(queue):
    choices = string.ascii_lowercase + string.digits
    while True:
        host_id = "".join(random.choices(choices, k=4))
        msg = Message(
           msg_id=str(uuid.uuid4()),
           inst_name=f"cattle-{host_id}"
        )

        asyncio.create_task(queue.put(msg))
        logging.info(f"Published {msg}")

        # simulate randomness of publishing messages
        await asyncio.sleep(random.random())
```

concurrently publish messages

```
async def publish(queue):
    choices = string.ascii_lowercase + string.digits
    while True:
        host_id = "".join(random.choices(choices, k=4))
        msg = Message(
           msg_id=str(uuid.uuid4()),
           inst_name=f"cattle-{host_id}"
        )

        await queue.put(msg)
        logging.info(f"Published {msg}")

        # simulate randomness of publishing messages
        await asyncio.sleep(random.random())
```

concurrently publish messages

```
async def publish(queue):
    choices = string.ascii_lowercase + string.digits
    while True:
        host_id = "".join(random.choices(choices, k=4))
        msg = Message(
           msg_id=str(uuid.uuid4()),
           inst_name=f"cattle-{host_id}"
        )

        await queue.put(msg)  # lines below are blocked
        logging.info(f"Published {msg}")

        # simulate randomness of publishing messages
        await asyncio.sleep(random.random())
```

concurrently publish messages

```
async def publish(queue):
    choices = string.ascii_lowercase + string.digits
    while True:
        host_id = "".join(random.choices(choices, k=4))
        msg = Message(
           msg_id=str(uuid.uuid4()),
           inst_name=f"cattle-{host_id}"
        )

        asyncio.create_task(queue.put(msg))
        logging.info(f"Published {msg}")

        # simulate randomness of publishing messages
        await asyncio.sleep(random.random())
```

initial setup

concurrently consume messages

concurrently consume messages

```
async def consume(queue):
    while True:
        msg = await queue.get()
        logging.info(f"Consumed {msg}")
        # unhelpful simulation of an i/o operation
        await asyncio.sleep(random.random())
```

concurrently consume messages

```
async def consume(queue):
    while True:
        msg = await queue.get()                    # <-- does not block loop
        logging.info(f"Consumed {msg}")
        # unhelpful simulation of an i/o operation
        await asyncio.sleep(random.random())
```

concurrently consume messages

```
async def consume(queue):
    while True:
        msg = await queue.get()                    # <-- only blocks coro scope
        logging.info(f"Consumed {msg}")
        # unhelpful simulation of an i/o operation
        await asyncio.sleep(random.random())
```

concurrently consume messages

```
async def consume(queue):
    while True:
        msg = await queue.get()
        logging.info(f"Consumed {msg}")
        await restart_host(msg)


async def restart_host(msg):
    # unhelpful simulation of i/o work
    await asyncio.sleep(random.random())
    msg.restarted = True
    logging.info(f"Restarted {msg.hostname}")
```

concurrently consume messages

```
async def consume(queue):
    while True:
        msg = await queue.get()
        logging.info(f"Consumed {msg}")
        asyncio.create_task(restart_host(msg))


async def restart_host(msg):
    # unhelpful simulation of i/o work
    await asyncio.sleep(random.random())
    msg.restarted = True
    logging.info(f"Restarted {msg.hostname}")
```

concurrently consume messages

```python
async def consume(queue):
    while True:
        msg = await queue.get()
        logging.info(f"Consumed {msg}")
        asyncio.create_task(restart_host(msg))


async def restart_host(msg):
    # unhelpful simulation of i/o work
    await asyncio.sleep(random.random())
    msg.restarted = True
    logging.info(f"Restarted {msg.hostname}")
```

initial setup

concurrent work

concurrent work

```
async def restart_host(msg):
    # unhelpful simulation of i/o work
    await asyncio.sleep(random.random())
    msg.restarted = True
    logging.info(f"Restarted {msg.hostname}")
```

concurrent work

```
async def restart_host(msg):
    # unhelpful simulation of i/o work
    await asyncio.sleep(random.random())
    msg.restarted = True
    logging.info(f"Restarted {msg.hostname}")

async def save(msg):
    # unhelpful simulation of i/o work
    await asyncio.sleep(random.random())
    msg.saved = True
    logging.info(f"Saved {msg} into database")
```

concurrent work

```
async def restart_host(msg):
    ...

async def save(msg):
    ...

async def consume(queue):
    while True:
        msg = await queue.get()
        logging.info(f"Pulled {msg}")

        asyncio.create_task(save(msg))
        asyncio.create_task(restart_host(msg))
```

concurrent work

```
async def restart_host(msg):
    ...

async def save(msg):
    ...

async def consume(queue):
    while True:
        msg = await queue.get()
        logging.info(f'Pulled {msg}')

        await save(msg)
        await restart_host(msg)
```

block when needed

```
async def restart_host(msg):
    ...

async def save(msg):
    ...

async def consume(queue):
    while True:
        msg = await queue.get()
        logging.info(f'Pulled {msg}')

        await save(msg)
        last_restart = await last_restart_date(msg)
        if today - last_restart > max_days:
            await restart_host(msg)
```

block when needed

```
async def handle_message(msg):
    await save(msg)
    last_restart = await last_restart_date(msg)
    if today - last_restart > max_days:
       asyncio.create_task(restart_host(msg))


async def consume(queue):
    while True:
        msg = await queue.get()
        logging.info(f"Pulled {msg}")
        asyncio.create_task(handle_message(msg))
```

block when needed

```
async def handle_message(msg):
    asyncio.create_task(save(msg))
    asyncio.create_task(restart_host(msg))


async def consume(queue):
    while True:
        msg = await queue.get()
        logging.info(f"Pulled {msg}")
        asyncio.create_task(handle_message(msg))
```

initial setup

finalization tasks

unblocking: finalization tasks

```
def cleanup(msg):
    msg.acked = True
    logging.info(f"Done. Acked {msg}")
```

unblocking: finalization tasks

```
def cleanup(msg):
    msg.acked = True
    logging.info(f"Done. Acked {msg}")

async def handle_message(msg):
    asyncio.create_task(save(msg))
    asyncio.create_task(restart_host(msg))
```

unblocking: finalization tasks

```
def cleanup(msg):
    msg.acked = True
    logging.info(f"Done. Acked {msg}")

async def handle_message(msg):
    await save(msg)
    await restart_host(msg)
    cleanup(msg)
```

unblocking: finalization tasks

```
def cleanup(msg, fut):
    msg.acked = True
    logging.info(f"Done. Acked {msg}")

async def handle_message(msg):
    g_future = asyncio.gather(save(msg), restart_host(msg))

    callback = functools.partial(cleanup, msg)
    g_future.add_done_callback(callback)
    await g_future
```

unblocking: finalization tasks

```
13:15:31,250 INFO: Pulled Message(inst_name='cattle-zpsk')
13:15:31,286 INFO: Restarted cattle-zpsk.example.net
13:15:31,347 INFO: Pulled Message(inst_name='cattle-998c')
13:15:31,486 INFO: Saved Message(inst_name='cattle-zpsk') into database
13:15:31,486 INFO: Done. Acked Message(inst_name='cattle-zpsk')
13:15:31,811 INFO: Pulled Message(inst_name='cattle-j9bu')
13:15:31,863 INFO: Saved Message(inst_name='cattle-998c') into database
13:15:31,903 INFO: Pulled Message(inst_name='cattle-vk5l')
13:15:32,149 INFO: Pulled Message(inst_name='cattle-1lf2')
13:15:32,239 INFO: Restarted cattle-vk5l.example.net
13:15:32,245 INFO: Restarted cattle-998c.example.net
13:15:32,245 INFO: Done. Acked Message(inst_name='cattle-998c')
13:15:32,267 INFO: Saved Message(inst_name='cattle-j9bu') into database
13:15:32,478 INFO: Pulled Message(inst_name='cattle-mflk')
13:15:32,481 INFO: Restarted cattle-j9bu.example.net
13:15:32,482 INFO: Done. Acked Message(inst_name='cattle-j9bu')
13:15:32,505 INFO: Pulled Message(inst_name='cattle-t7tv')
```

unblocking: finalization tasks

```
13:15:31,250 INFO: Pulled Message(inst_name='cattle-zpsk')
13:15:31,286 INFO: Restarted cattle-zpsk.example.net
13:15:31,347 INFO: Pulled Message(inst_name='cattle-998c')
13:15:31,486 INFO: Saved Message(inst_name='cattle-zpsk') into database
13:15:31,486 INFO: Done. Acked Message(inst_name='cattle-zpsk')
13:15:31,811 INFO: Pulled Message(inst_name='cattle-j9bu')
13:15:31,863 INFO: Saved Message(inst_name='cattle-998c') into database
13:15:31,903 INFO: Pulled Message(inst_name='cattle-vk5l')
13:15:32,149 INFO: Pulled Message(inst_name='cattle-1lf2')
13:15:32,239 INFO: Restarted cattle-vk5l.example.net
13:15:32,245 INFO: Restarted cattle-998c.example.net
13:15:32,245 INFO: Done. Acked Message(inst_name='cattle-998c')
13:15:32,267 INFO: Saved Message(inst_name='cattle-j9bu') into database
13:15:32,478 INFO: Pulled Message(inst_name='cattle-mflk')
13:15:32,481 INFO: Restarted cattle-j9bu.example.net
13:15:32,482 INFO: Done. Acked Message(inst_name='cattle-j9bu')
13:15:32,505 INFO: Pulled Message(inst_name='cattle-t7tv')
```

unblocking: finalization tasks

```
13:15:31,250 INFO: Pulled Message(inst_name='cattle-zpsk')
13:15:31,286 INFO: Restarted cattle-zpsk.example.net
13:15:31,347 INFO: Pulled Message(inst_name='cattle-998c')
13:15:31,486 INFO: Saved Message(inst_name='cattle-zpsk') into database
13:15:31,486 INFO: Done. Acked Message(inst_name='cattle-zpsk')
13:15:31,811 INFO: Pulled Message(inst_name='cattle-j9bu')
13:15:31,863 INFO: Saved Message(inst_name='cattle-998c') into database
13:15:31,903 INFO: Pulled Message(inst_name='cattle-vk5l')
13:15:32,149 INFO: Pulled Message(inst_name='cattle-1lf2')
13:15:32,239 INFO: Restarted cattle-vk5l.example.net
13:15:32,245 INFO: Restarted cattle-998c.example.net
13:15:32,245 INFO: Done. Acked Message(inst_name='cattle-998c')
13:15:32,267 INFO: Saved Message(inst_name='cattle-j9bu') into database
13:15:32,478 INFO: Pulled Message(inst_name='cattle-mflk')
13:15:32,481 INFO: Restarted cattle-j9bu.example.net
13:15:32,482 INFO: Done. Acked Message(inst_name='cattle-j9bu')
13:15:32,505 INFO: Pulled Message(inst_name='cattle-t7tv')
```

unblocking: finalization tasks

```
def cleanup(msg, fut):
    msg.acked = True
    logging.info(f"Done. Acked {msg}")

async def handle_message(msg):
    g_future = asyncio.gather(save(msg), restart_host(msg))

    callback = functools.partial(cleanup, msg)
    g_future.add_done_callback(callback)
    await g_future
```

unblocking: finalization tasks

```
async def cleanup(msg):
    msg.acked = True
    logging.info(f"Done. Acked {msg}")

async def handle_message(msg):
    await asyncio.gather(save(msg), restart_host(msg))
    await cleanup(msg)
```

adding concurrency: tl;dr

- Asynchronous != concurrent
- Serial != blocking

graceful shutdowns

graceful shutdowns

responding to signals

responding to signals

```
def main():
    queue = asyncio.Queue()
    loop = asyncio.get_event_loop()

    try:
        loop.create_task(publish(queue))
        loop.create_task(consume(queue))
        loop.run_forever()
    finally:
        logging.info("Cleaning up")
        loop.close()
```

responding to signals

```
def main():
    queue = asyncio.Queue()
    loop = asyncio.get_event_loop()

    try:
        loop.create_task(publish(queue))
        loop.create_task(consume(queue))
        loop.run_forever()
    except KeyboardInterrupt:
        logging.info("Process interrupted")
    finally:
        logging.info("Cleaning up")
        loop.close()
```

responding to signals

```
def main():
    queue = asyncio.Queue()
    loop = asyncio.get_event_loop()

    try:
        loop.create_task(publish(queue))
        loop.create_task(consume(queue))
        loop.run_forever()
    except KeyboardInterrupt:                    # <-- a.k.a. SIGINT
        logging.info("Process interrupted")
    finally:
        logging.info("Cleaning up")
        loop.close()
```

responding to signals

```
$ python mayhem.py
$ pkill -INT -f "python mayhem.py"

19:11:25,321 INFO: Pulled Message(inst_name='cattle-lrnm')
19:11:25,321 INFO: Done. Acked Message(inst_name='cattle-lrnm')
19:11:25,700 INFO: Pulled Message(inst_name='cattle-m0f6')
19:11:25,700 INFO: Done. Acked Message(inst_name='cattle-m0f6')
19:11:25,740 INFO: Saved Message(inst_name='cattle-m0f6') into database
19:11:25,840 INFO: Saved Message(inst_name='cattle-lrnm') into database
19:11:26,144 INFO: Process interrupted
19:11:26,144 INFO: Cleaning up
```

responding to signals

```
$ python mayhem.py
$ pkill -INT -f "python mayhem.py"

19:11:25,321 INFO: Pulled Message(inst_name='cattle-lrnm')
19:11:25,321 INFO: Done. Acked Message(inst_name='cattle-lrnm')
19:11:25,700 INFO: Pulled Message(inst_name='cattle-m0f6')
19:11:25,700 INFO: Done. Acked Message(inst_name='cattle-m0f6')
19:11:25,740 INFO: Saved Message(inst_name='cattle-m0f6') into database
19:11:25,840 INFO: Saved Message(inst_name='cattle-lrnm') into database
19:11:26,144 INFO: Process interrupted
19:11:26,144 INFO: Cleaning up
```

responding to signals

```
$ python mayhem.py
$ pkill -TERM -f "python mayhem.py"

19:08:25,553 INFO: Pulled Message(inst_name='cattle-npww')
19:08:25,554 INFO: Done. Acked Message(inst_name='cattle-npww')
19:08:25,655 INFO: Pulled Message(inst_name='cattle-rm7n')
19:08:25,655 INFO: Done. Acked Message(inst_name='cattle-rm7n')
19:08:25,790 INFO: Saved Message(inst_name='cattle-rm7n') into database
19:08:25,831 INFO: Saved Message(inst_name='cattle-npww') into database
[1]    78851 terminated  python mandrill/mayhem.py
```

responding to signals

```
$ python mayhem.py
$ pkill -TERM -f "python mayhem.py"

19:08:25,553 INFO: Pulled Message(inst_name='cattle-npww')
19:08:25,554 INFO: Done. Acked Message(inst_name='cattle-npww')
19:08:25,655 INFO: Pulled Message(inst_name='cattle-rm7n')
19:08:25,655 INFO: Done. Acked Message(inst_name='cattle-rm7n')
19:08:25,790 INFO: Saved Message(inst_name='cattle-rm7n') into database
19:08:25,831 INFO: Saved Message(inst_name='cattle-npww') into database
[1]    78851 terminated  python mandrill/mayhem.py
```

responding to signals

```
def main():
    queue = asyncio.Queue()
    loop = asyncio.get_event_loop()

    try:
        loop.create_task(publish(queue))
        loop.create_task(consume(queue))
        loop.run_forever()
    except KeyboardInterrupt:
        logging.info("Process interrupted")
    finally:
        logging.info("Cleaning up")
        loop.close()
```

responding to signals

```
def main():
    queue = asyncio.Queue()
    loop = asyncio.get_event_loop()  # <-- could happen here or earlier

    try:
        loop.create_task(publish(queue))
        loop.create_task(consume(queue))
        loop.run_forever()
    except KeyboardInterrupt:
        logging.info("Process interrupted")  # <-- could happen here
    finally:
        logging.info("Cleaning up")          # <-- could happen here
        loop.close()                         # <-- could happen here
```

graceful shutdowns

signal handler

signal handler

```
async def shutdown(signal, loop):
    logging.info(f"Received exit signal {signal.name}...")
    logging.info("Closing database connections")
    logging.info("Nacking outstanding messages")
    tasks = [
        t for t in asyncio.all_tasks()
        if t is not asyncio.current_task()
    ]
    [task.cancel() for task in tasks]

    logging.info(f"Cancelling {len(tasks)} outstanding tasks")
    await asyncio.gather(*tasks, return_exceptions=True)
    logging.info("Flushing metrics”)
    loop.stop()
    logging.info("Shutdown complete.")
```

signal handler

```
async def shutdown(signal, loop):
    logging.info(f"Received exit signal {signal.name}...")
    logging.info("Closing database connections")
    logging.info("Nacking outstanding messages")
      tasks = [
          t for t in asyncio.all_tasks()
          if t is not asyncio.current_task()
      ]
      [task.cancel() for task in tasks]

      logging.info(f"Cancelling {len(tasks)} outstanding tasks")
      await asyncio.gather(*tasks, return_exceptions=True)
      logging.info("Flushing metrics”)
      loop.stop()
      logging.info("Shutdown complete.")
```

signal handler

```
async def shutdown(signal, loop):
    logging.info(f"Received exit signal {signal.name}...")
    logging.info("Closing database connections")
    logging.info("Nacking outstanding messages")
    tasks = [
      t for t in asyncio.all_tasks()
      if t is not asyncio.current_task()
    ]
   [task.cancel() for task in tasks]

    logging.info(f"Cancelling {len(tasks)} outstanding tasks")
   await asyncio.gather(*tasks, return_exceptions=True)
   logging.info("Flushing metrics")
    loop.stop()
    logging.info("Shutdown complete.")
```

signal handler

```
def main():
    queue = asyncio.Queue()
    loop = asyncio.get_event_loop()

    try:
        loop.create_task(publish(queue))
        loop.create_task(consume(queue))
        loop.run_forever()
    except KeyboardInterrupt:
        logging.info("Process interrupted")
    finally:
        logging.info("Cleaning up")
        loop.close()
```

signal handler

```
def main():
    loop = asyncio.get_event_loop()
    signals = (signal.SIGHUP, signal.SIGTERM, signal.SIGINT)
    for s in signals:
        loop.add_signal_handler(
            s, lambda s=s: asyncio.create_task(shutdown(s, loop)))
    queue = asyncio.Queue()

    try:
        loop.create_task(publish(queue))
        loop.create_task(consume(queue))
        loop.run_forever()
    finally:
        logging.info("Cleaning up")
        loop.close()
```

signal handler

```
def main():
    loop = asyncio.get_event_loop()
    signals = (signal.SIGHUP, signal.SIGTERM, signal.SIGINT)
    for s in signals:
        loop.add_signal_handler(
            s, lambda s=s: asyncio.create_task(shutdown(s, loop)))
    queue = asyncio.Queue()

    try:
        loop.create_task(publish(queue))
        loop.create_task(consume(queue))
        loop.run_forever()
    finally:
        logging.info("Cleaning up")
        loop.close()
```

signal handler

```
$ python mayhem.py
# or -HUP or -INT
$ pkill -TERM -f "python mayhem.py"

19:11:25,321 INFO: Pulled Message(inst_name='cattle-lrnm')
19:11:25,321 INFO: Done. Acked Message(inst_name='cattle-lrnm')
19:11:25,700 INFO: Pulled Message(inst_name='cattle-m0f6')
19:11:25,700 INFO: Done. Acked Message(inst_name='cattle-m0f6')
19:11:25,740 INFO: Saved Message(inst_name='cattle-m0f6') into database
19:11:25,840 INFO: Saved Message(inst_name='cattle-lrnm') into database
19:11:26,143 INFO: Received exit signal SIGTERM...
19:11:26,143 INFO: Closing database connections
19:11:26,144 INFO: Cancelling 19 outstanding tasks
19:11:26,144 INFO: Flushing metrics
19:11:26,145 INFO: Cleaning up
```

signal handler

```
$ python mayhem.py
# or -HUP or -INT
$ pkill -TERM -f "python mayhem.py"

19:11:25,321 INFO: Pulled Message(inst_name='cattle-lrnm')
19:11:25,321 INFO: Done. Acked Message(inst_name='cattle-lrnm')
19:11:25,700 INFO: Pulled Message(inst_name='cattle-m0f6')
19:11:25,700 INFO: Done. Acked Message(inst_name='cattle-m0f6')
19:11:25,740 INFO: Saved Message(inst_name='cattle-m0f6') into database
19:11:25,840 INFO: Saved Message(inst_name='cattle-lrnm') into database
19:11:26,143 INFO: Received exit signal SIGTERM...
19:11:26,143 INFO: Closing database connections
19:11:26,144 INFO: Cancelling 19 outstanding tasks
19:11:26,144 INFO: Flushing metrics
19:11:26,145 INFO: Cleaning up
```

graceful shutdowns

which signals to care about

which signals to care about

	Hard Exit	Graceful	Reload/Restart
nginx	`TERM, INT`	`QUIT`	`HUP`
Apache	`TERM`	`WINCH`	`HUP`
uWSGI	`INT, QUIT`		`HUP, TERM`
Gunicorn	`INT, QUIT`	`TERM`	`HUP`
Docker	`KILL`	`TERM`	

graceful shutdowns

not-so-graceful `asyncio.shield`

ungraceful shutdown: `asyncio.shield`

```
async def cant_stop_me():
    logging.info("Hold on...")
    await asyncio.sleep(60)
    logging.info("Done!")
```

ungraceful shutdown: asyncio.shield

```
async def cant_stop_me():
    ...

def main():
    loop = asyncio.get_event_loop()
    signals = (signal.SIGHUP, signal.SIGTERM, signal.SIGINT)
    for s in signals:
        loop.add_signal_handler(
            s, lambda s=s: asyncio.create_task(shutdown(s, loop)))

    shielded_coro = asyncio.shield(cant_stop_me())
    try:
        loop.run_until_complete(shielded_coro)
    finally:
        logging.info("Cleaning up")
        loop.close()
```

ungraceful shutdown: asyncio.shield

```
async def cant_stop_me():
    ...

def main():
    loop = asyncio.get_event_loop()
    signals = (signal.SIGHUP, signal.SIGTERM, signal.SIGINT)
    for s in signals:
        loop.add_signal_handler(
            s, lambda s=s: asyncio.create_task(shutdown(s, loop)))

    shielded_coro = asyncio.shield(cant_stop_me())
    try:
        loop.run_until_complete(shielded_coro)
    finally:
        logging.info("Cleaning up")
        loop.close()
```

ungraceful shutdown: `asyncio.shield`

```
13:24:20,105 INFO: Hold on...
^C13:24:21,156 INFO: Received exit signal SIGINT...
13:24:21,156 INFO: Cancelling 2 outstanding tasks
13:24:21,156 INFO: Coroutine cancelled
13:24:21,157 INFO: Cleaning up
Traceback (most recent call last):
  File "examples/shield_test.py", line 62, in
    loop.run_until_complete(shielded_coro)
  File "/Users/lynn/.pyenv/versions/3.7.0/lib/python3.7/asyncio/base_event
    return future.result()
concurrent.futures._base.CancelledError
```

ungraceful shutdown: `asyncio.shield`

```
async def cant_stop_me():
    logging.info("Hold on...")
    await asyncio.sleep(60)
    logging.info("Done!")

async def main():
    await asyncio.shield(cant_stop_me())

asyncio.run(main())
```

ungraceful shutdown: `asyncio.shield`

```
18:27:17,587 INFO: Hold on...
^C18:27:18,982 INFO: Cleaning up
Traceback (most recent call last):
  File "shield_test_no_shutdown.py", line 23, in <module>
    loop.run_until_complete(shielded_coro)
  File "/Users/lynn/.pyenv/versions/3.6.2/lib/python3.6/asyncio/base_event
    self.run_forever()
  File "/Users/lynn/.pyenv/versions/3.6.2/lib/python3.6/asyncio/base_event
    self._run_once()
  File "/Users/lynn/.pyenv/versions/3.6.2/lib/python3.6/asyncio/base_event
    event_list = self._selector.select(timeout)
  File "/Users/lynn/.pyenv/versions/3.6.2/lib/python3.6/selectors.py", lin
    kev_list = self._kqueue.control(None, max_ev, timeout)
KeyboardInterrupt
```

ungraceful shutdown: asyncio.shield

```
async def cant_stop_me():
    logging.info("Hold on...")
    await asyncio.sleep(60)
    logging.info("Done!")

async def imma_let_you_speak(task_to_cancel):
    await asyncio.sleep(2)
    logging.info(f"interrupting {task_to_cancel}”)
    task_to_cancel.cancel()

async def main():
    shielded = asyncio.shield(cant_stop_me())
    cancel_coro = imma_let_you_speak(shielded)
    await asyncio.gather(shielded, cancel_coro)

asyncio.run(main())
```

ungraceful shutdown: asyncio.shield

```
async def cant_stop_me():
    logging.info("Hold on...")
    await asyncio.sleep(60)
    logging.info("Done!")

async def imma_let_you_speak(task_to_cancel):
    await asyncio.sleep(2)
    logging.info(f"interrupting {task_to_cancel}”)
    task_to_cancel.cancel()

async def main():
    shielded = asyncio.shield(cant_stop_me())
    cancel_coro = imma_let_you_speak(shielded)
    await asyncio.gather(shielded, cancel_coro)

asyncio.run(main())
```

ungraceful shutdown: asyncio.shield

```
18:43:53,729 INFO: Hold on...
18:43:55,730 INFO: killing <Future pending cb=[gather.<locals>._done_callb
python3.7/asyncio/tasks.py:660]>
Traceback (most recent call last):
  File "shield_test_no_shutdown.py", line 38, in <module>
    asyncio.run(main())
  File "/Users/lynn/.pyenv/versions/3.7.0/lib/python3.7/asyncio/runners.py
    return loop.run_until_complete(main)
  File "/Users/lynn/.pyenv/versions/3.7.0/lib/python3.7/asyncio/base_event
    return future.result()
concurrent.futures._base.CancelledError
```

graceful shutdown: tl;dr

- try/except/finally isn’t enough
- Define desired shutdown behavior
- Use signal handlers
- Listen for appropriate signals

exception handling

exception handling

global handler

global handler

```
async def restart_host(msg):
    # unhelpful simulation of i/o work
    await asyncio.sleep(random.random())
    msg.restarted = True
    logging.info(f"Restarted {msg.hostname}")
```

global handler

```
async def restart_host(msg):
    # unhelpful simulation of i/o work
    await asyncio.sleep(random.random())
    rand_int = random.randrange(1, 5)
    if rand_int == 2:
        raise Exception(f"Could not restart {msg.hostname}!")

    msg.restarted = True
    logging.info(f"Restarted {msg.hostname}")
```

global handler

```
$ python mayhem.py
```

global handler

```
$ python mayhem.py

13:49:33,524 INFO: Pulled Message(inst_name='cattle-hvy0')
13:49:33,924 INFO: Pulled Message(inst_name='cattle-5i2f')
13:49:33,925 ERROR: Task exception was never retrieved
future: <Task finished coro=<restart_host() done, defined at
mayhem_ex_handling.py:56> exception=Exception('Could not restart
cattle-5i2f.example.net')>
Traceback (most recent call last):
  File "mayhem_ex_handling.py", line 60, in restart_host
    raise Exception(f"Could not restart {msg.hostname}")
Exception: Could not restart cattle-5i2f.example.net
13:49:34,247 INFO: Pulled Message(inst_name='cattle-e086')
13:49:34,432 INFO: Saved Message(inst_name='cattle-hvy0') into database
13:49:34,517 INFO: Restarted cattle-hvy0.example.net
```

global handler

```
$ python mayhem.py

13:49:33,524 INFO: Pulled Message(inst_name='cattle-hvy0')
13:49:33,924 INFO: Pulled Message(inst_name='cattle-5i2f')
13:49:33,925 ERROR: Task exception was never retrieved
future: <Task finished coro=<restart_host() done, defined at
mayhem_ex_handling.py:56> exception=Exception('Could not restart
cattle-5i2f.example.net')>
Traceback (most recent call last):
  File "mayhem_ex_handling.py", line 60, in restart_host
    raise Exception(f"Could not restart {msg.hostname}")
Exception: Could not restart cattle-5i2f.example.net
13:49:34,247 INFO: Pulled Message(inst_name='cattle-e086')
13:49:34,432 INFO: Saved Message(inst_name='cattle-hvy0') into database
13:49:34,517 INFO: Restarted cattle-hvy0.example.net
```

global handler

```
$ python mayhem.py

13:49:33,524 INFO: Pulled Message(inst_name='cattle-hvy0')
13:49:33,924 INFO: Pulled Message(inst_name='cattle-5i2f')
13:49:33,925 ERROR: Task exception was never retrieved
future: <Task finished coro=<restart_host() done, defined at
mayhem_ex_handling.py:56> exception=Exception('Could not restart
cattle-5i2f.example.net')>
Traceback (most recent call last):
  File "mayhem_ex_handling.py", line 60, in restart_host
    raise Exception(f"Could not restart {msg.hostname}")
Exception: Could not restart cattle-5i2f.example.net
13:49:34,247 INFO: Pulled Message(inst_name='cattle-e086')
13:49:34,432 INFO: Saved Message(inst_name='cattle-hvy0') into database
13:49:34,517 INFO: Restarted cattle-hvy0.example.net
```

global handler

```
def exception_handler(loop, context):
    logging.error(f"Caught exception: {context['exception']}")
```

global handler

```
def exception_handler(loop, context):
    logging.error(f"Caught exception: {context['exception']}")

def main():
    loop = asyncio.get_event_loop()
    # <-- snip ->
    loop.set_exception_handler(exception_handler)
    # <-- snip -->
```

global handler

```
def exception_handler(loop, context):
    logging.error(f"Caught exception: {context['exception']}")

def main():
    loop = asyncio.get_event_loop()
    # <-- snip -->
    loop.set_exception_handler(exception_handler)
    # <-- snip -->
```

global handler

```
$ python mayhem.py
```

global handler

```
$ python mayhem.py

14:01:56,187 INFO: Pulled Message(instance_name='cattle-i490')
14:01:56,192 INFO: Restarted cattle-i490.example.net
14:01:56,241 INFO: Pulled Message(instance_name='cattle-31is')
14:01:56,331 INFO: Saved Message(instance_name='cattle-31is') into database
14:01:56,535 INFO: Pulled Message(instance_name='cattle-sx7f')
14:01:56,535 ERROR: Caught exception: Could not restart cattle-sx7f.example.n
14:01:56,730 INFO: Pulled Message(instance_name='cattle-hsh9')
14:01:56,731 INFO: Saved Message(instance_name='cattle-sx7f') into database
14:01:56,759 INFO: Pulled Message(instance_name='cattle-g20p')
14:01:56,800 INFO: Restarted cattle-31is.example.net
14:01:57,26 INFO: Saved Message(instance_name='cattle-i490') into database
14:01:57,45 INFO: Saved Message(instance_name='cattle-hsh9') into database
14:01:57,181 INFO: Saved Message(instance_name='cattle-g20p') into database
14:01:57,194 INFO: Restarted cattle-g20p.example.net
```

global handler

```
$ python mayhem.py

14:01:56,187 INFO: Pulled Message(instance_name='cattle-i490')
14:01:56,192 INFO: Restarted cattle-i490.example.net
14:01:56,241 INFO: Pulled Message(instance_name='cattle-31is')
14:01:56,331 INFO: Saved Message(instance_name='cattle-31is') into database
14:01:56,535 INFO: Pulled Message(instance_name='cattle-sx7f')
14:01:56,535 ERROR: Caught exception: Could not restart cattle-sx7f
14:01:56,730 INFO: Pulled Message(instance_name='cattle-hsh9')
14:01:56,731 INFO: Saved Message(instance_name='cattle-sx7f') into database
14:01:56,759 INFO: Pulled Message(instance_name='cattle-g20p')
14:01:56,800 INFO: Restarted cattle-31is.example.net
14:01:57,26 INFO: Saved Message(instance_name='cattle-i490') into database
14:01:57,45 INFO: Saved Message(instance_name='cattle-hsh9') into database
14:01:57,181 INFO: Saved Message(instance_name='cattle-g20p') into database
14:01:57,194 INFO: Restarted cattle-g20p.example.net
```

exception handling

specific handlers

specific handlers

```
async def handle_message(msg):
    await asyncio.gather(save(msg), restart_host(msg))
    await cleanup(msg)
```

specific handlers

```
async def handle_message(msg):
    saved, restarted = await asyncio.gather(
        save(msg), restart_host(msg), return_exceptions=True)

    to_ack = True
    if isinstance(restarted, Exception):
        to_ack = False

    await cleanup(msg, to_ack)
```

specific handlers

```
async def handle_message(msg):
    saved, restarted = await asyncio.gather(
        save(msg), restart_host(msg), return_exceptions=True)

    to_ack = True
    if isinstance(restarted, Exception):
        to_ack = False

    await cleanup(msg, to_ack)
```

specific handlers

```
async def handle_message(msg):
    saved, restarted = await asyncio.gather(
        save(msg), restart_host(msg), return_exceptions=True)

    to_ack = True
    if isinstance(restarted, Exception):
        to_ack = False

    await cleanup(msg, to_ack)
```

specific handlers

```
async def handle_message(msg):
    saved, restarted = await asyncio.gather(
        save(msg), restart_host(msg), return_exceptions=True)

    to_ack = True
    if isinstance(restarted, Exception):
        to_ack = False

    await cleanup(msg, to_ack)
```

specific handlers

```
async def handle_message(msg):
    saved, restarted = await asyncio.gather(
        save(msg), restart_host(msg), return_exceptions=True)

    to_ack = True
    if isinstance(restarted, Exception):
        to_ack = False

    await cleanup(msg, to_ack)
```

exception handling

- global exception handling: **`loop.set_exception_handler`**
- individual exception handling: **`asyncio.gather`** with **`return_exceptions=True`**

threads and asyncio

threads and asyncio

running coroutines from other threads

running coroutines from other threads

```
def threaded_consume():
    threaded_pubsub_client.subscribe(TOPIC, handle_message_sync)
```

running coroutines from other threads

```
def threaded_consume():
    threaded_pubsub_client.subscribe(TOPIC, handle_message_sync)

def handle_message_sync(msg):
    msg = Message(**msg.json_data)
    logging.info(f"Pulled {msg}")
    asyncio.create_task(handle_message(msg))
```

running coroutines from other threads

```
def threaded_consume():
    threaded_pubsub_client.subscribe(TOPIC, handle_message_sync)

def handle_message_sync(msg):
    msg = Message(**msg.json_data)
    logging.info(f"Pulled {msg}")
    asyncio.create_task(handle_message(msg))
```

running coroutines from other threads

```
def threaded_consume():
    threaded_pubsub_client.subscribe(TOPIC, handle_message_sync)

def handle_message_sync(msg):
    msg = Message(**msg.json_data)
    logging.info(f"Pulled {msg}")
    asyncio.create_task(handle_message(msg))

async def run():
    loop = asyncio.get_running_loop()
    executor = concurrent.futures.ThreadPoolExecutor()
    await loop.run_in_executor(executor, threaded_consume, loop)
```

running coroutines from other threads

```
def threaded_consume():
    threaded_pubsub_client.subscribe(TOPIC, handle_message_sync)

def handle_message_sync(msg):
    msg = Message(**msg.json_data)
    logging.info(f"Pulled {msg}")
    asyncio.create_task(handle_message(msg))

async def run():
    loop = asyncio.get_running_loop()
    executor = concurrent.futures.ThreadPoolExecutor()
    await loop.run_in_executor(executor, threaded_consume, loop)
```

running coroutines from other threads

```
16:45:36,833 INFO: Pulled Message(inst_name='cattle-hvy0')
16:45:36,833 ERROR: Top-level exception occurred in callback while
processing a message
Traceback (most recent call last):
  File "/Users/lynn/.pyenv/versions/ep18-37/lib/python3.7/site-packages/
google/cloud/pubsub_v1/subscriber/_protocol/streaming_pull_manager.py",
line 63, in _wrap_callback_errors
    callback(message)
  File "mayhem.py", line 115, in callback
    asyncio.create_task(handle_message(data))
  File "/Users/lynn/.pyenv/versions/3.7.0/lib/python3.7/asyncio/tasks.py",
line 320, in create_task
    loop = events.get_running_loop()
RuntimeError: no running event loop
```

running coroutines from other threads

```
16:45:36,833 INFO: Pulled Message(inst_name='cattle-hvy0')
16:45:36,833 ERROR: Top-level exception occurred in callback while
processing a message
Traceback (most recent call last):
  File "/Users/lynn/.pyenv/versions/ep18-37/lib/python3.7/site-packages/
google/cloud/pubsub_v1/subscriber/_protocol/streaming_pull_manager.py",
line 63, in _wrap_callback_errors
    callback(message)
  File "mayhem.py", line 115, in callback
    asyncio.create_task(handle_message(data))
  File "/Users/lynn/.pyenv/versions/3.7.0/lib/python3.7/asyncio/tasks.py",
line 320, in create_task
    loop = events.get_running_loop()
RuntimeError: no running event loop
```

running coroutines from other threads

```
def threaded_consume():
    threaded_pubsub_client.subscribe(TOPIC, handle_message_sync)

def handle_message_sync(msg):
    msg = Message(**msg.json_data)
    logging.info(f"Pulled {msg}")
    asyncio.create_task(handle_message(msg))

async def run():
    loop = asyncio.get_running_loop()
    executor = concurrent.futures.ThreadPoolExecutor()
    await loop.run_in_executor(executor, threaded_consume, loop)
```

running coroutines from other threads

```
def threaded_consume(loop):
    callback = functools.partial(handle_message_sync, loop)
    threaded_pubsub_client.subscribe(TOPIC, callback)

def handle_message_sync(loop, msg):
    msg = Message(**msg.json_data)
    logging.info(f"Pulled {msg}")
    loop.create_task(handle_message(msg))

async def run():
    loop = asyncio.get_running_loop()
    executor = concurrent.futures.ThreadPoolExecutor()
    await loop.run_in_executor(executor, threaded_consume, loop)
```

running coroutines from other threads

```
def threaded_consume(loop):
    callback = functools.partial(handle_message_sync, loop)
    threaded_pubsub_client.subscribe(TOPIC, callback)

def handle_message_sync(loop, msg):
    msg = Message(**msg.json_data)
    logging.info(f"Pulled {msg}")
    loop.create_task(handle_message(msg))

async def run():
    loop = asyncio.get_running_loop()
    executor = concurrent.futures.ThreadPoolExecutor()
    await loop.run_in_executor(executor, threaded_consume, loop)
```

running coroutines from other threads

```
18:08:10,543 INFO: Pulled Message(inst_name='xbci')
18:08:10,543 INFO: Pulled Message(inst_name='e8x5')
18:08:10,544 INFO: Running something else
18:08:10,721 INFO: Saved Message(inst_name='e8x5') into database
18:08:10,828 INFO: Saved Message(inst_name='xbci') into database
18:08:10,828 ERROR: Caught exception: Could not restart xbci.example.net
18:08:11,549 INFO: Restarted e8x5.example.net
18:08:11,821 INFO: Done. Message(inst_name='e8x5')
18:08:12,108 INFO: Running something else
18:08:12,276 INFO: Done. Message(inst_name='xbci')
```

threads and asyncio

running coroutines from other threads

take 2

running coroutines from other threads

```
def threaded_consume(loop):
    callback = functools.partial(handle_message_sync, loop)
    threaded_pubsub_client.subscribe(TOPIC, callback)

def handle_message_sync(loop, msg):
    msg = Message(**msg.json_data)
    logging.info(f"Pulled {msg}")
    loop.create_task(handle_message(msg))

async def run():
    loop = asyncio.get_running_loop()
    executor = concurrent.futures.ThreadPoolExecutor()
    await loop.run_in_executor(executor, threaded_consume, loop)
```

running coroutines from other threads

```
def threaded_consume(loop):
    callback = functools.partial(handle_message_sync, loop)
    threaded_pubsub_client.subscribe(TOPIC, callback)

def handle_message_sync(loop, msg):
    msg = Message(**msg.json_data)
    logging.info(f"Pulled {msg}")
    asyncio.run_coroutine_threadsafe(handle_message(data), loop)

async def run():
    loop = asyncio.get_running_loop()
    executor = concurrent.futures.ThreadPoolExecutor()
    await loop.run_in_executor(executor, threaded_consume, loop)
```

threads and asyncio

- **`ThreadPoolExecutor`**: calling threaded code from the main event loop
- **`asyncio.run_coroutine_threadsafe`**: running a coroutine on the main event loop from another thread

testing asyncio code

testing asyncio code

simple testing with pytest

simple testing with pytest

```
async def save(msg):
    # unhelpful simulation of i/o work
    await asyncio.sleep(random.random())
    msg.saved = True
    logging.info(f"Saved {msg} into database")
```

simple testing with pytest

```python
@pytest.fixture
def message():
    return mayhem.Message(msg_id="1234", instance_name="mayhem_test")


def test_save(message):
    assert not message.saved  # sanity check
    asyncio.run(mayhem.save(message))
    assert message.saved
```

simple testing with pytest

```python
@pytest.fixture
def message():
    return mayhem.Message(msg_id="1234", instance_name="mayhem_test")


def test_save(message):
    assert not message.saved  # sanity check
    asyncio.run(mayhem.save(message))
    assert message.saved
```

simple testing with pytest

```
@pytest.fixture
def message():
    return mayhem.Message(msg_id="1234", instance_name="mayhem_test")


def test_save(message):
    assert not message.saved  # sanity check
    loop = asyncio.get_event_loop()
    loop.run_until_complete(mayhem.save(message))
    loop.close()
    assert message.saved
```

simple testing with pytest

```
@pytest.fixture
def message():
    return mayhem.Message(msg_id="1234", instance_name="mayhem_test")


def test_save(message):
    assert not message.saved  # sanity check
    loop = asyncio.get_event_loop()
    loop.run_until_complete(mayhem.save(message))
    loop.close()
    assert message.saved
```

simple testing with pytest

```python
@pytest.fixture
def message():
    return mayhem.Message(msg_id="1234", instance_name="mayhem_test")


@pytest.mark.asyncio
async def test_save(message):
    assert not message.saved  # sanity check
    await mayhem.save(message)
    assert message.saved
```

simple testing with pytest

```python
@pytest.fixture
def message():
    return mayhem.Message(msg_id="1234", instance_name="mayhem_test")


@pytest.mark.asyncio
async def test_save(message):
    assert not message.saved  # sanity check
    await mayhem.save(message)
    assert message.saved
```

simple testing with pytest

```python
@pytest.fixture
def message():
    return mayhem.Message(msg_id="1234", instance_name="mayhem_test")


@pytest.mark.asyncio
async def test_save(message):
    assert not message.saved  # sanity check
    await mayhem.save(message)
    assert message.saved
```

testing asyncio code

mocking coroutines

mocking coroutines

```
async def save(msg):
    # unhelpful simulation of i/o work
    await asyncio.sleep(random.random())
    msg.saved = True
    logging.info(f"Saved {msg} into database")
```

mocking coroutines

```
async def save(msg):
    # unhelpful simulation of i/o work
    await asyncio.sleep(random.random())
    msg.saved = True
    logging.info(f"Saved {msg} into database")
```

mocking coroutines

```python
@pytest.fixture
def create_coro_mock(mocker, monkeypatch):
    def _create_mock_patch_coro(to_patch=None):
        mock = mocker.Mock()

        async def _coro(*args, **kwargs):
            return mock(*args, **kwargs)

        if to_patch:  # <-- may not need/want to patch anything
            monkeypatch.setattr(to_patch, _coro)
        return mock, _coro

    return _create_mock_patch_coro
```

mocking coroutines

```python
@pytest.fixture
def create_coro_mock(mocker, monkeypatch):
    def _create_mock_patch_coro(to_patch=None):
        mock = mocker.Mock()

        async def _coro(*args, **kwargs):
            return mock(*args, **kwargs)

        if to_patch:  # <-- may not need/want to patch anything
            monkeypatch.setattr(to_patch, _coro)
        return mock, _coro

    return _create_mock_patch_coro
```

mocking coroutines

```python
@pytest.fixture
def create_coro_mock(mocker, monkeypatch):
    def _create_mock_patch_coro(to_patch=None):
        mock = mocker.Mock()

        async def _coro(*args, **kwargs):
            return mock(*args, **kwargs)

        if to_patch:  # <-- may not need/want to patch anything
            monkeypatch.setattr(to_patch, _coro)
        return mock, _coro

    return _create_mock_patch_coro
```

mocking coroutines

```python
@pytest.fixture
def create_coro_mock(mocker, monkeypatch):
    def _create_mock_patch_coro(to_patch=None):
        mock = mocker.Mock()

        async def _coro(*args, **kwargs):
            return mock(*args, **kwargs)

        if to_patch:  # <-- may not need/want to patch anything
            monkeypatch.setattr(to_patch, _coro)
        return mock, _coro

    return _create_mock_patch_coro
```

mocking coroutines

```python
@pytest.fixture
def create_coro_mock(mocker, monkeypatch):
    def _create_mock_patch_coro(to_patch=None):
        mock = mocker.Mock()

        async def _coro(*args, **kwargs):
            return mock(*args, **kwargs)

        if to_patch:  # <-- may not need/want to patch anything
            monkeypatch.setattr(to_patch, _coro)
        return mock, _coro

    return _create_mock_patch_coro
```

mocking coroutines

```python
@pytest.fixture
def mock_sleep(create_coro_mock):
    mock, _ = create_coro_mock("mayhem.asyncio.sleep")
    return mock
```

mocking coroutines

```python
@pytest.fixture
def mock_sleep(create_coro_mock):
    mock, _ = create_coro_mock("mayhem.asyncio.sleep")
    return mock


@pytest.mark.asyncio
async def test_save(mock_sleep, message):
    assert not message.saved  # sanity check
    await mayhem.save(message)
    assert message.saved
    assert 1 == mock_sleep.call_count
```

mocking coroutines

```python
@pytest.fixture
def mock_sleep(create_coro_mock):
    mock, _ = create_coro_mock("mayhem.asyncio.sleep")
    return mock


@pytest.mark.asyncio
async def test_save(mock_sleep, message):
    assert not message.saved  # sanity check
    await mayhem.save(message)
    assert message.saved
    assert 1 == mock_sleep.call_count
```

mocking coroutines

```python
@pytest.fixture
def mock_sleep(create_coro_mock):
    mock, _ = create_coro_mock("mayhem.asyncio.sleep")
    return mock


@pytest.mark.asyncio
async def test_save(mock_sleep, message):
    assert not message.saved  # sanity check
    await mayhem.save(message)
    assert message.saved
    assert 1 == mock_sleep.call_count
```

testing asyncio code

testing create_task

testing create_task

```
async def consume(queue):
    while True:
        msg = await queue.get()
        logging.info(f"Pulled {msg}")
        asyncio.create_task(handle_message(msg))
```

testing create_task

```
@pytest.fixture
def mock_queue(mocker, monkeypatch):
    queue = mocker.Mock()
    monkeypatch.setattr(mayhem.asyncio, "Queue", queue)
    return queue.return_value
```

testing create_task

```
@pytest.fixture
def mock_queue(mocker, monkeypatch):
    queue = mocker.Mock()
    monkeypatch.setattr(mayhem.asyncio, "Queue", queue)
    return queue.return_value


@pytest.fixture
def mock_get(mock_queue, create_coro_mock):
    mock, coro = create_coro_mock()
    mock_queue.get = coro
    return mock
```

testing create_task

```python
@pytest.fixture
def mock_queue(mocker, monkeypatch):
    queue = mocker.Mock()
    monkeypatch.setattr(mayhem.asyncio, "Queue", queue)
    return queue.return_value


@pytest.fixture
def mock_get(mock_queue, create_coro_mock):
    mock, coro = create_coro_mock()
    mock_queue.get = coro
    return mock
```

testing create_task

```
@pytest.mark.asyncio
async def test_consume(mock_get, mock_queue, message, create_coro_mock):
    mock_handle_message, _ = create_coro_mock("mayhem.handle_message")
    mock_get.side_effect = [message, Exception("break while loop")]

    with pytest.raises(Exception, match="break while loop"):
        await mayhem.consume(mock_queue)

    mock_handle_message.assert_called_once_with(message)
```

testing create_task

```
@pytest.mark.asyncio
async def test_consume(mock_get, mock_queue, message, create_coro_mock):
    mock_handle_message, _ = create_coro_mock("mayhem.handle_message")
    mock_get.side_effect = [message, Exception("break while loop")]

    with pytest.raises(Exception, match="break while loop"):
        await mayhem.consume(mock_queue)

    mock_handle_message.assert_called_once_with(message)
```

testing create_task

```python
@pytest.mark.asyncio
async def test_consume(mock_get, mock_queue, message, create_coro_mock):
    mock_handle_message, _ = create_coro_mock("mayhem.handle_message")
    mock_get.side_effect = [message, Exception("break while loop")]

    with pytest.raises(Exception, match="break while loop"):
        await mayhem.consume(mock_queue)

    mock_handle_message.assert_called_once_with(message)
```

testing create_task

```
@pytest.mark.asyncio
async def test_consume(mock_get, mock_queue, message, create_coro_mock):
    mock_handle_message, _ = create_coro_mock("mayhem.handle_message")
    mock_get.side_effect = [message, Exception("break while loop")]

    with pytest.raises(Exception, match="break while loop"):
        await mayhem.consume(mock_queue)

    mock_handle_message.assert_called_once_with(message)
```

testing create_task

```
@pytest.mark.asyncio
async def test_consume(mock_get, mock_queue, message, create_coro_mock):
    mock_handle_message, _ = create_coro_mock("mayhem.handle_message")
    mock_get.side_effect = [message, Exception("break while loop")]

    with pytest.raises(Exception, match="break while loop"):
        await mayhem.consume(mock_queue)

    mock_handle_message.assert_called_once_with(message)
```

testing create_task

```
================================= FAILURES =================================
_____________________________ test_consume _____________________________

<--snip-->
    @pytest.mark.asyncio
    async def test_consume(mock_get, mock_queue, message, create_coro_moc
        mock_get.side_effect = [message, Exception("break while loop")]
        mock_handle_message = create_coro_mock("mayhem.handle_message")

        with pytest.raises(Exception, match="break while loop"):
            await mayhem.consume(mock_queue)

>       mock_handle_message.assert_called_once_with(message)
E       AssertionError: Expected 'mock' to be called once. Called 0 times

test_mayhem.py:230: AssertionError
```

testing create_task

```python
@pytest.mark.asyncio
async def test_consume(mock_get, mock_queue, message, create_coro_mock):
    mock_handle_message, _ = create_coro_mock("mandrill.handle_message")
    mock_get.side_effect = [message, Exception("break while loop")]

    with pytest.raises(Exception, match="break while loop"):
        await mayhem.consume(mock_queue)

    ret_tasks = [
        t for t in asyncio.all_tasks() if t is not asyncio.current_task()
    ]
    assert 1 == len(ret_tasks)
    mock_handle_message.assert_not_called()  # <-- sanity check

    await asyncio.gather(*ret_tasks)
    mock_handle_message.assert_called_once_with(message)
```

testing create_task

```python
@pytest.mark.asyncio
async def test_consume(mock_get, mock_queue, message, create_coro_mock):
    mock_handle_message, _ = create_coro_mock("mandrill.handle_message")
    mock_get.side_effect = [message, Exception("break while loop")]

    with pytest.raises(Exception, match="break while loop"):
        await mayhem.consume(mock_queue)

    ret_tasks = [
        t for t in asyncio.all_tasks() if t is not asyncio.current_task()
    ]
    assert 1 == len(ret_tasks)
    mock_handle_message.assert_not_called()  # <-- sanity check

    await asyncio.gather(*ret_tasks)
    mock_handle_message.assert_called_once_with(message)
```

testing create_task

```python
@pytest.mark.asyncio
async def test_consume(mock_get, mock_queue, message, create_coro_mock):
    mock_handle_message, _ = create_coro_mock("mandrill.handle_message")
    mock_get.side_effect = [message, Exception("break while loop")]

    with pytest.raises(Exception, match="break while loop"):
        await mayhem.consume(mock_queue)

    ret_tasks = [
        t for t in asyncio.all_tasks() if t is not asyncio.current_task()
    ]
    assert 1 == len(ret_tasks)
    mock_handle_message.assert_not_called()  # <-- sanity check

    await asyncio.gather(*ret_tasks)
    mock_handle_message.assert_called_once_with(message)
```

testing asyncio code

testing the event loop

testing the event loop

```
def main():
    loop = asyncio.get_event_loop()
    for s in (signal.SIGHUP, signal.SIGTERM, signal.SIGINT):
        loop.add_signal_handler(
            s, lambda s=s: asyncio.create_task(shutdown(s, loop))
        )
    loop.set_exception_handler(exception_handler)
    queue = asyncio.Queue()

    try:
        loop.create_task(publish(queue))
        loop.create_task(consume(queue))
        loop.run_forever()
    finally:
        logging.info("Cleaning up")
        loop.close()
```

testing the event loop

```
@pytest.fixture
def event_loop(event_loop, mocker):
    new_loop = asyncio.get_event_loop_policy().new_event_loop()
    asyncio.set_event_loop(new_loop)
    new_loop._close = new_loop.close
    new_loop.close = mocker.Mock()

    yield new_loop

    new_loop._close()
```

testing the event loop

```
@pytest.fixture
def event_loop(event_loop, mocker):
    new_loop = asyncio.get_event_loop_policy().new_event_loop()
    asyncio.set_event_loop(new_loop)
    new_loop._close = new_loop.close
    new_loop.close = mocker.Mock()

    yield new_loop

    new_loop._close()
```

testing the event loop

```
@pytest.fixture
def event_loop(event_loop, mocker):
    new_loop = asyncio.get_event_loop_policy().new_event_loop()
    asyncio.set_event_loop(new_loop)
    new_loop._close = new_loop.close
    new_loop.close = mocker.Mock()

    yield new_loop

    new_loop._close()
```

testing the event loop

```
def test_main(create_mock_coro, event_loop, mock_queue):
    mock_consume, _ = create_mock_coro("mayhem.consume")
    mock_publish, _ = create_mock_coro("mayhem.publish")
    mock_shutdown_gather, _ = create_mock_coro("mayhem.asyncio.gather")

    def _send_signal():
        time.sleep(0.1)
        os.kill(os.getpid(), signal.SIGTERM)

    thread = threading.Thread(target=_send_signal, daemon=True)
    thread.start()

    mandrill.main()
    # <--snip-->
```

testing the event loop

```
def test_main(create_mock_coro, event_loop, mock_queue):
    mock_consume, _ = create_mock_coro("mayhem.consume")
    mock_publish, _ = create_mock_coro("mayhem.publish")
    mock_shutdown_gather, _ = create_mock_coro("mayhem.asyncio.gather")

    def _send_signal():
        time.sleep(0.1)
        os.kill(os.getpid(), signal.SIGTERM)

    thread = threading.Thread(target=_send_signal, daemon=True)
    thread.start()

    mandrill.main()
    # <--snip-->
```

testing the event loop

```
def test_main(create_mock_coro, event_loop, mock_queue):
    mock_consume, _ = create_mock_coro("mayhem.consume")
    mock_publish, _ = create_mock_coro("mayhem.publish")
    mock_shutdown_gather, _ = create_mock_coro("mayhem.asyncio.gather")

    def _send_signal():
        time.sleep(0.1)
        os.kill(os.getpid(), signal.SIGTERM)

    thread = threading.Thread(target=_send_signal, daemon=True)
    thread.start()

    mandrill.main()
    # <--snip-->
```

testing the event loop

```
def test_main(create_mock_coro, event_loop, mock_queue):
    mock_consume, _ = create_mock_coro("mayhem.consume")
    mock_publish, _ = create_mock_coro("mayhem.publish")
    mock_shutdown_gather, _ = create_mock_coro("mayhem.asyncio.gather")

    def _send_signal():
        time.sleep(0.1)
        os.kill(os.getpid(), signal.SIGTERM)

    thread = threading.Thread(target=_send_signal, daemon=True)
    thread.start()

    mandrill.main()
    # <--snip-->
```

testing the event loop

```
def test_main(create_mock_coro, event_loop, mock_queue):
    # <--snip-->
    mayhem.main()

    assert signal.SIGTERM in event_loop._signal_handlers
    assert mayhem.handle_exception == event_loop.get_exception_handler()

    mock_consume.assert_called_once_with(mock_queue)
    mock_publish.assert_called_once_with(mock_queue)
    mock_shutdown_gather.assert_called_once_with()

    # asserting the loop is stopped but not closed
    assert not event_loop.is_running()
    assert not event_loop.is_closed()

    event_loop.close.assert_called_once_with()
```

testing the event loop

```python
def test_main(create_mock_coro, event_loop, mock_queue):
    # <--snip-->
    mayhem.main()

    assert signal.SIGTERM in event_loop._signal_handlers
    assert mayhem.handle_exception == event_loop.get_exception_handler()

    mock_consume.assert_called_once_with(mock_queue)
    mock_publish.assert_called_once_with(mock_queue)
    mock_shutdown_gather.assert_called_once_with()

    # asserting the loop is stopped but not closed
    assert not event_loop.is_running()
    assert not event_loop.is_closed()

    event_loop.close.assert_called_once_with()
```

testing the event loop

```python
def test_main(create_mock_coro, event_loop, mock_queue):
    # <--snip-->
    mayhem.main()

    assert signal.SIGTERM in event_loop._signal_handlers
    assert mayhem.handle_exception == event_loop.get_exception_handler()

    mock_consume.assert_called_once_with(mock_queue)
    mock_publish.assert_called_once_with(mock_queue)
    mock_shutdown_gather.assert_called_once_with()

    # asserting the loop is stopped but not closed
    assert not event_loop.is_running()
    assert not event_loop.is_closed()

    event_loop.close.assert_called_once_with()
```

testing the event loop

```
def test_main(create_mock_coro, event_loop, mock_queue):
    mock_consume, _ = create_mock_coro("mayhem.consume")
    mock_publish, _ = create_mock_coro("mayhem.publish")
    mock_shutdown_gather, _ = create_mock_coro("mayhem.asyncio.gather")

    def _send_signal():
        time.sleep(0.1)
        os.kill(os.getpid(), signal.SIGTERM)

    thread = threading.Thread(target=_send_signal, daemon=True)
    thread.start()

    mandrill.main()
    # <--snip-->
```

testing the event loop

```python
@pytest.mark.parametrize("sig_to_test", ("SIGINT", "SIGTERM", "SIGHUP"))
def test_main(sig_to_test, create_mock_coro, event_loop, mock_queue):
    mock_consume, _ = create_mock_coro("mayhem.consume")
    mock_publish, _ = create_mock_coro("mayhem.publish")
    mock_shutdown_gather, _ = create_mock_coro("mayhem.asyncio.gather")

    def _send_signal():
        time.sleep(0.1)
        os.kill(os.getpid(), sig_to_test)

    thread = threading.Thread(target=_send_signal, daemon=True)
    thread.start()

    mandrill.main()

    assert sig_to_test in event_loop._signal_handlers
    # <--snip-->
```

testing the event loop

```python
@pytest.mark.parametrize("sig_to_test", ("SIGINT", "SIGTERM", "SIGHUP"))
def test_main(sig_to_test, create_mock_coro, event_loop, mock_queue):
    mock_consume, _ = create_mock_coro("mayhem.consume")
    mock_publish, _ = create_mock_coro("mayhem.publish")
    mock_shutdown_gather, _ = create_mock_coro("mayhem.asyncio.gather")

    def _send_signal():
        time.sleep(0.1)
        os.kill(os.getpid(), sig_to_test)

    thread = threading.Thread(target=_send_signal, daemon=True)
    thread.start()

    mandrill.main()

    assert sig_to_test in event_loop._signal_handlers
    # <--snip-->
```

testing asyncio code

- `pytest-asyncio` + mocked coroutines
- `asynctest` for ~~the former Java developers~~ stdlib's `unittest`

debugging asyncio code

debugging asyncio code

manual debugging

manual debugging

```
async def monitor_tasks():
    while True:
        tasks = [
            t for t in asyncio.all_tasks()
            if t is not asyncio.current_task()
        ]
        [t.print_stack() for t in tasks]
        await asyncio.sleep(2)
```

manual debugging

```
async def monitor_tasks():
    while True:
        tasks = [
            t for t in asyncio.all_tasks()
            if t is not asyncio.current_task()
        ]
        [t.print_stack() for t in tasks]
        await asyncio.sleep(2)
```

manual debugging

```
Stack for <Task pending coro=<handle_message() running at mayhem.py:107> w
  File "mayhem.py", line 107, in handle_message
    save_coro, restart_coro, return_exceptions=True
Stack for <Task pending coro=<handle_message() running at mayhem.py:107> w
  File "mayhem.py", line 107, in handle_message
    save_coro, restart_coro, return_exceptions=True
Stack for <Task pending coro=<cleanup() running at mayhem.py:78> wait_for=
  File "mayhem.py", line 78, in cleanup
    await asyncio.sleep(random.random())
Stack for <Task pending coro=<consume() running at mayhem.py:115> wait_for
  File "mayhem.py", line 115, in consume
    msg = await queue.get()
Stack for <Task pending coro=<restart_host() running at mayhem.py:62> wait
  File "mayhem.py", line 62, in restart_host
    await asyncio.sleep(random.randrange(1, 3))
```

manual debugging

```
Stack for <Task pending coro=<handle_message() running at mayhem.py:107> w
  File "mayhem.py", line 107, in handle_message
    save_coro, restart_coro, return_exceptions=True
Stack for <Task pending coro=<handle_message() running at mayhem.py:107> w
  File "mayhem.py", line 107, in handle_message
    save_coro, restart_coro, return_exceptions=True
Stack for <Task pending coro=<cleanup() running at mayhem.py:78> wait_for=
  File "mayhem.py", line 78, in cleanup
    await asyncio.sleep(random.random())
Stack for <Task pending coro=<consume() running at mayhem.py:115> wait_for
  File "mayhem.py", line 115, in consume
    msg = await queue.get()
Stack for <Task pending coro=<restart_host() running at mayhem.py:62> wait
  File "mayhem.py", line 62, in restart_host
    await asyncio.sleep(random.randrange(1, 3))
```

manual debugging

```
async def monitor_tasks():
    while True:
        tasks = [
            t for t in asyncio.all_tasks()
            if t is not asyncio.current_task()
        ]
        [t.print_stack(limit=5) for t in tasks]
        await asyncio.sleep(2)
```

debugging asyncio code

using debug mode

using debug mode: traceback context

```
$ PYTHONASYNCIODEBUG=1 python mayhem.py
```

using debug mode: traceback context

```
12:57:52,830 ERROR: Task exception was never retrieved
```

using debug mode: traceback context

```
12:57:52,830 ERROR: Task exception was never retrieved
future: <Task finished coro=<handle_message() done, defined at
mayhem.py:97> exception=Exception('Could not restart cattle-
ykdc.example.net') created at /Users/lynn/.pyenv/versions/3.7.2/lib/
python3.7/asyncio/tasks.py:325>
```

using debug mode: traceback context

```
12:57:52,830 ERROR: Task exception was never retrieved
future: <Task finished ...>
source_traceback: Object created at (most recent call last):
  File "mayhem.py", line 164, in <module>
    main()
  File "mayhem.py", line 157, in main
    loop.run_forever()
  File "/Users/lynn/.pyenv/versions/3.7.2/lib/python3.7/asyncio/base_event
    self._run_once()
  File "/Users/lynn/.pyenv/versions/3.7.2/lib/python3.7/asyncio/base_event
    handle._run()
  File "/Users/lynn/.pyenv/versions/3.7.2/lib/python3.7/asyncio/events.py"
    self._context.run(self._callback, *self._args)
  File "mayhem.py", line 117, in consume
    asyncio.create_task(handle_message(msg))
  File "/Users/lynn/.pyenv/versions/3.7.2/lib/python3.7/asyncio/tasks.py",
    return loop.create_task(coro)
```

using debug mode: traceback context

```
12:57:52,830 ERROR: Task exception was never retrieved
future: <Task finished ...>
source_traceback: Object created at ...
Traceback (most recent call last):
  File "mayhem.py", line 107, in handle_message
    save_coro, restart_coro
  File "mayhem.py", line 60, in restart_host
    raise Exception(f"Could not restart {msg.hostname}")
Exception: Could not restart cattle-ykdc.example.net
```

using debug mode: thread safety

```
$ PYTHONASYNCIODEBUG=1 python mayhem.py
```

using debug mode: thread safety

```
$ PYTHONASYNCIODEBUG=1 python mayhem.py

20:21:59,954 ERROR: Top-level exception occurred in callback while
processing a message
Traceback (most recent call last):
  File "/Users/lynn/.pyenv/versions/pycon19/lib/python3.7/site-packages/
google/cloud/pubsub_v1/subscriber/_protocol/streaming_pull_manager.py",
line 63, in _wrap_callback_errors
    callback(message)
  File "mayhem.py", line 174, in callback
    loop.create_task(handle_message(pubsub_msg))
    # <-- snip -->
RuntimeError: Non-thread-safe operation invoked on an event loop other
than the current one
```

using debug mode: thread safety

```
$ PYTHONASYNCIODEBUG=1 python mayhem.py

20:21:59,954 ERROR: Top-level exception occurred in callback while
processing a message
Traceback (most recent call last):
  File "/Users/lynn/.pyenv/versions/pycon19/lib/python3.7/site-packages/
google/cloud/pubsub_v1/subscriber/_protocol/streaming_pull_manager.py",
line 63, in _wrap_callback_errors
    callback(message)
  File "mayhem.py", line 174, in callback
    loop.create_task(handle_message(pubsub_msg))
    # <-- snip -->
RuntimeError: Non-thread-safe operation invoked on an event loop other
than the current one
```

using debug mode: slow coros

```
async def save(msg):
    time.sleep(1 + random.random())
    msg.saved = True
    logging.info(f"Saved {msg} into database")
```

using debug mode: slow coros

```
async def save(msg):
    time.sleep(1 + random.random())
    msg.saved = True
    logging.info(f"Saved {msg} into database")
```

using debug mode: slow coros

```
$ PYTHONASYNCIODEBUG=1 python mayhem.py
```

using debug mode: slow coros

```
$ PYTHONASYNCIODEBUG=1 python mayhem.py

20:00:57,781 INFO: Pulled Message(inst_name='cattle-okxa')
20:00:57,782 INFO: Extended deadline 3s for Message(inst_name='cattle-okxa')
20:00:59,416 INFO: Saved Message(inst_name='cattle-okxa') into database
20:00:59,417 WARNING: Executing <Task finished coro=<save() done, defined at
mayhem.py:68> result=None
    created at /Users/lynn/.pyenv/versions/3.7.2/lib/python3.7/asyncio/
tasks.py:719> took 1.634 seconds
20:00:59,418 INFO: Pulled Message(instance_name='cattle-pmbv')
```

using debug mode: slow coros

```
$ PYTHONASYNCIODEBUG=1 python mayhem.py

20:00:57,781 INFO: Pulled Message(inst_name='cattle-okxa')
20:00:57,782 INFO: Extended deadline 3s for Message(inst_name='cattle-okxa')
20:00:59,416 INFO: Saved Message(inst_name='cattle-okxa') into database
20:00:59,417 WARNING: Executing <Task finished coro=<save() done, defined at
mayhem.py:68> result=None
    created at /Users/lynn/.pyenv/versions/3.7.2/lib/python3.7/asyncio/
tasks.py:719> took 1.634 seconds
20:00:59,418 INFO: Pulled Message(instance_name='cattle-pmbv')
```

using debug mode: slow coros

```
def main():
    queue = asyncio.Queue()
    loop = asyncio.get_event_loop()
    # float, in seconds
    loop.slow_callback_duration = 0.5

    # <-- snip -->
```

using debug mode: slow coros

```
def main():
    queue = asyncio.Queue()
    loop = asyncio.get_event_loop()
    # float, in seconds
    loop.slow_callback_duration = 0.5

    # <-- snip -->
```

debugging asyncio code

debugging in production

debugging in production

```
# <-- snip -->
from aiodebug import log_slow_callbacks
# <-- snip -->
def main():
    queue = asyncio.Queue()
    loop = asyncio.get_event_loop()
    log_slow_callbacks.enable(0.05)
```

debugging in production

```
# <-- snip -->
from aiodebug import log_slow_callbacks
# <-- snip -->
def main():
    queue = asyncio.Queue()
    loop = asyncio.get_event_loop()
    log_slow_callbacks.enable(0.05)
```

debugging in production

```
# <-- snip -->
from aiodebug import log_slow_callbacks
# <-- snip -->
def main():
    queue = asyncio.Queue()
    loop = asyncio.get_event_loop()
    log_slow_callbacks.enable(0.05)
```

debugging in production

```
# <-- snip -->
from aiodebug import log_slow_callbacks
from aiodebug import monitor_loop_lag
# <-- snip -->
def main():
    queue = asyncio.Queue()
    loop = asyncio.get_event_loop()
    log_slow_callbacks.enable(0.05)
    monitor_loop_lag.enable(my_statsd_client)
```

debugging asyncio code

- manual: **`task.print_stack()`**
- proper: built-in debug mode
- f-it, we'll do it live: **`aiodebug`**

profiling asyncio code

profiling asyncio code

cProfile

cProfile

```
$ timeout -s INT 5s python -m cProfile -s tottime mayhem.py
```

cProfile

```
$ timeout -s INT 5s python -m cProfile -s tottime mayhem.py
   ncalls  tottime  percall ... filename:lineno(function)
      134    4.785    0.036 ... {method 'control' of 'select.kqueue' objec
       17    0.007    0.000 ... {built-in method _imp.create_dynamic}
       62    0.007    0.000 ... {built-in method marshal.loads}
      132    0.003    0.000 ... base_events.py:1679(_run_once)
  217/216    0.003    0.000 ... {built-in method builtins.__build_class__}
      361    0.003    0.000 ... {built-in method posix.stat}
       62    0.002    0.000 ... <frozen importlib._bootstrap_external>:914
       42    0.002    0.000 ... {built-in method builtins.compile}
      195    0.001    0.000 ... <frozen importlib._bootstrap_external>:135
       50    0.001    0.000 ... {method 'write' of '_io.TextIOWrapper' obj
      122    0.001    0.000 ... _make.py:1217(__repr__)
       50    0.001    0.000 ... __init__.py:293(__init__)
       18    0.001    0.000 ... enum.py:134(__new__)
    72/15    0.001    0.000 ... sre_parse.py:475(_parse)
       62    0.001    0.000 ... {method 'read' of '_io.FileIO' objects}
```

cProfile

```
$ timeout -s INT 5s python -m cProfile -s tottime mayhem.py
   ncalls  tottime  percall ... filename:lineno(function)
      134    4.785    0.036 ... {method 'control' of 'select.kqueue' objec
       17    0.007    0.000 ... {built-in method _imp.create_dynamic}
       62    0.007    0.000 ... {built-in method marshal.loads}
      132    0.003    0.000 ... base_events.py:1679(_run_once)
  217/216    0.003    0.000 ... {built-in method builtins.__build_class__}
      361    0.003    0.000 ... {built-in method posix.stat}
       62    0.002    0.000 ... <frozen importlib._bootstrap_external>:914
       42    0.002    0.000 ... {built-in method builtins.compile}
      195    0.001    0.000 ... <frozen importlib._bootstrap_external>:135
       50    0.001    0.000 ... {method 'write' of '_io.TextIOWrapper' obj
      122    0.001    0.000 ... _make.py:1217(__repr__)
       50    0.001    0.000 ... __init__.py:293(__init__)
       18    0.001    0.000 ... enum.py:134(__new__)
    72/15    0.001    0.000 ... sre_parse.py:475(_parse)
       62    0.001    0.000 ... {method 'read' of '_io.FileIO' objects}
```

cProfile

```
$ timeout -s INT 5s python -m cProfile -s filename mayhem.py
   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
        1    0.000    0.000    4.704    4.704 mayhem.py:141(main)
       18    0.000    0.000    0.002    0.000 mayhem.py:56(restart_host)
       22    0.000    0.000    0.003    0.000 mayhem.py:67(save)
       33    0.000    0.000    0.003    0.000 mayhem.py:74(cleanup)
       22    0.000    0.000    0.002    0.000 mayhem.py:83(extend)
       11    0.000    0.000    0.000    0.000 mayhem.py:91(handle_results)
       12    0.000    0.000    0.002    0.000 mayhem.py:41(publish)
       22    0.000    0.000    0.002    0.000 mayhem.py:97(handle_message)
       12    0.000    0.000    0.003    0.000 mayhem.py:114(consume)
       11    0.000    0.000    0.000    0.000 mayhem.py:37(__attrs_post_in
        1    0.000    0.000    0.000    0.000 mayhem.py:26(Message)
        1    0.000    0.000    0.000    0.000 mayhem.py:130(<listcomp>)
        2    0.000    0.000    0.001    0.000 mayhem.py:126(shutdown)
        1    0.000    0.000    0.000    0.000 mayhem.py:148(<lambda>)
```

cProfile

```
$ timeout -s INT 5s python -m cProfile -s filename mayhem.py
   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
        1    0.000    0.000    4.704    4.704 mayhem.py:141(main)
       18    0.000    0.000    0.002    0.000 mayhem.py:56(restart_host)
       22    0.000    0.000    0.003    0.000 mayhem.py:67(save)
       33    0.000    0.000    0.003    0.000 mayhem.py:74(cleanup)
       22    0.000    0.000    0.002    0.000 mayhem.py:83(extend)
       11    0.000    0.000    0.000    0.000 mayhem.py:91(handle_results)
       12    0.000    0.000    0.002    0.000 mayhem.py:41(publish)
       22    0.000    0.000    0.002    0.000 mayhem.py:97(handle_message)
       12    0.000    0.000    0.003    0.000 mayhem.py:114(consume)
       11    0.000    0.000    0.000    0.000 mayhem.py:37(__attrs_post_in
        1    0.000    0.000    0.000    0.000 mayhem.py:26(Message)
        1    0.000    0.000    0.000    0.000 mayhem.py:130(<listcomp>)
        2    0.000    0.000    0.001    0.000 mayhem.py:126(shutdown)
        1    0.000    0.000    0.000    0.000 mayhem.py:148(<lambda>)
```

profiling asyncio code

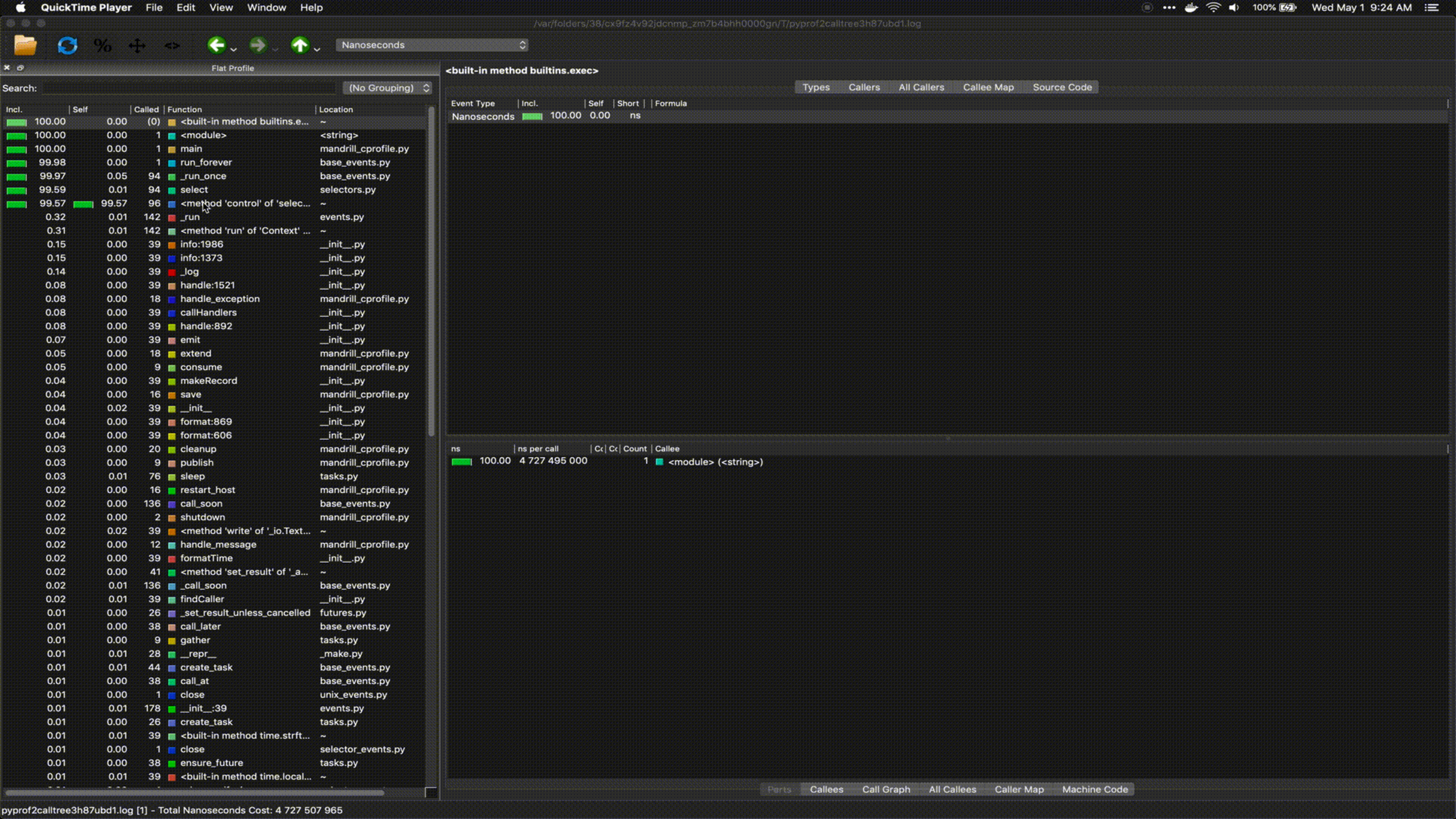
cProfile with KCacheGrind

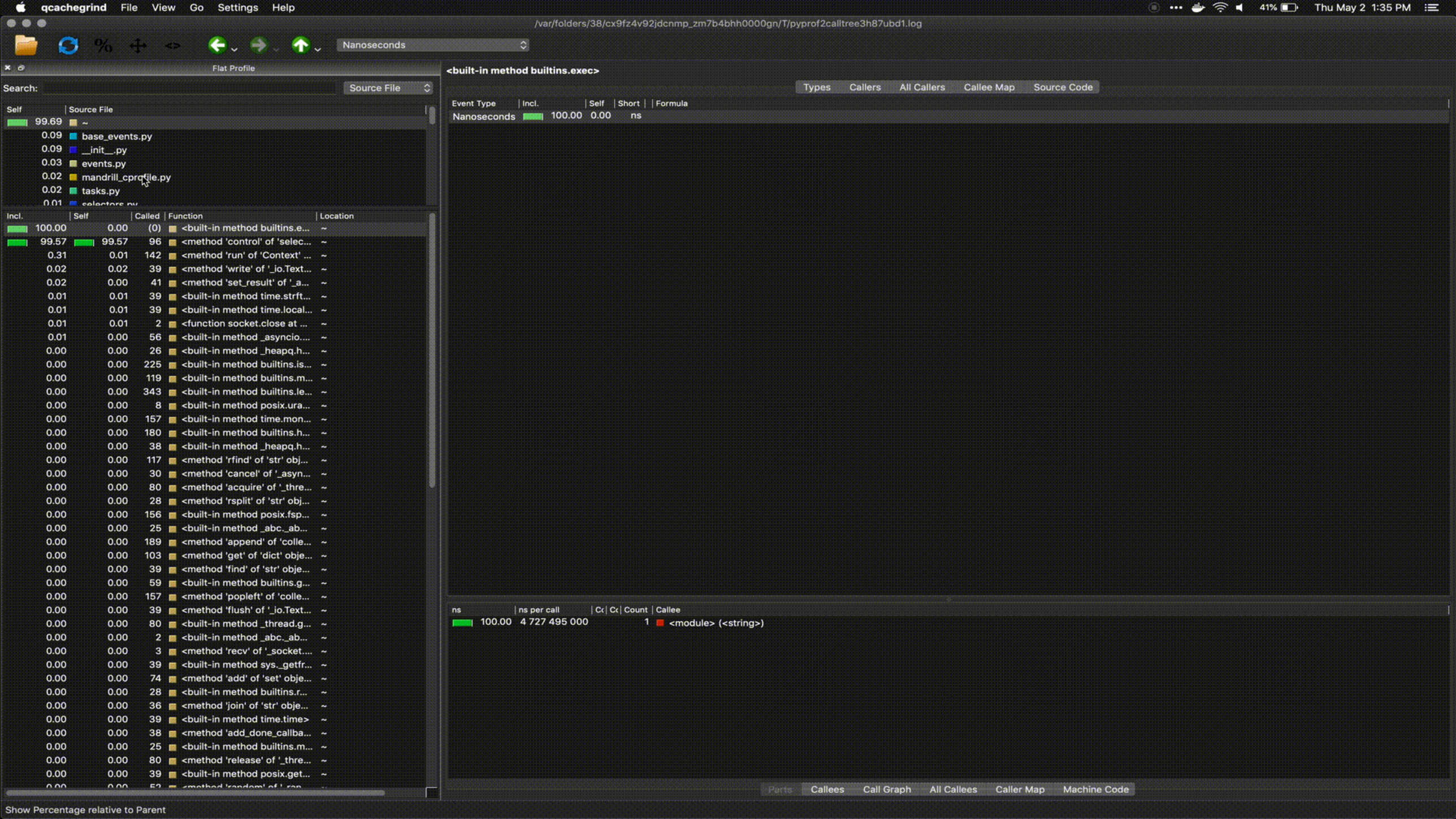
cProfile with (K|Q)CacheGrind

```
$ timeout -s INT 5s python -m cProfile -o mayhem.prof mayhem.py
```

cProfile with (K|Q)CacheGrind

```
$ timeout -s INT 5s python -m cProfile -o mayhem.prof mayhem.py

$ pyprof2calltree --kcachegrind -i mayhem.prof
```


QuickTime Player File Edit View Window Help
Wed May 1 9:24 AM
/var/folders/38/cx9fz4v92jdcnmp_zm7b4bhh0000gn/T/pyprof2calltree3h87ubd1.log
Nanoseconds
Flat Profile
Search:
(No Grouping)
Incl. | Self | Called | Function | Location
100.00 0.00 (0) <built-in method builtins.e... ~
100.00 0.00 1 <module> <string>
100.00 0.00 1 main mandrill_cprofile.py
99.98 0.00 1 run_forever base_events.py
99.97 0.05 94 _run_once base_events.py
99.59 0.01 94 select selectors.py
99.57 99.57 96 <method 'control' of 'selec... ~
0.32 0.01 142 _run events.py
0.31 0.01 142 <method 'run' of 'Context' ... ~
0.15 0.00 39 info:1986 __init__.py
0.15 0.00 39 info:1373 __init__.py
0.14 0.00 39 _log __init__.py
0.08 0.00 39 handle:1521 __init__.py
0.08 0.00 18 handle_exception mandrill_cprofile.py
0.08 0.00 39 callHandlers __init__.py
0.08 0.00 39 handle:892 __init__.py
0.07 0.00 39 emit __init__.py
0.05 0.00 18 extend mandrill_cprofile.py
0.05 0.00 9 consume mandrill_cprofile.py
0.04 0.00 39 makeRecord __init__.py
0.04 0.00 16 save mandrill_cprofile.py
0.04 0.02 39 __init__ __init__.py
0.04 0.00 39 format:869 __init__.py
0.04 0.00 39 format:606 __init__.py
0.03 0.00 20 cleanup mandrill_cprofile.py
0.03 0.00 9 publish mandrill_cprofile.py
0.03 0.01 76 sleep tasks.py
0.02 0.00 16 restart_host mandrill_cprofile.py
0.02 0.00 136 call_soon base_events.py
0.02 0.00 2 shutdown mandrill_cprofile.py
0.02 0.02 39 <method 'write' of '_io.Text... ~
0.02 0.00 12 handle_message mandrill_cprofile.py
0.02 0.00 39 formatTime __init__.py
0.02 0.00 41 <method 'set_result' of '_a... ~
0.02 0.01 136 _call_soon base_events.py
0.02 0.01 39 findCaller __init__.py
0.01 0.00 26 _set_result_unless_cancelled futures.py
0.01 0.00 38 call_later base_events.py
0.01 0.00 9 gather tasks.py
0.01 0.01 28 __repr__ _make.py
0.01 0.01 44 create_task base_events.py
0.01 0.00 38 call_at base_events.py
0.01 0.00 1 close unix_events.py
0.01 0.01 178 __init__:39 events.py
0.01 0.00 26 create_task tasks.py
0.01 0.01 39 <built-in method time.strft... ~
0.01 0.00 1 close selector_events.py
0.01 0.00 38 ensure_future tasks.py
0.01 0.01 39 <built-in method time.local... ~
<built-in method builtins.exec>
Types Callers All Callers Callee Map Source Code
Event Type | Incl. | Self | Short | Formula
Nanoseconds 100.00 0.00 ns
ns | ns per call | Cc | Cc | Count | Callee
100.00 4 727 495 000 1 <module> (<string>)
Parts Callees Call Graph All Callees Caller Map Machine Code
pyprof2calltree3h87ubd1.log [1] - Total Nanoseconds Cost: 4 727 507 965


qcachegrind File View Go Settings Help
41% Thu May 2 1:35 PM
/var/folders/38/cx9fz4v92jdcnmp_zm7b4bhh0000gn/T/pyprof2calltree3h87ubd1.log
Nanoseconds
Flat Profile
Search:
Source File
Self | Source File
99.69 ~
0.09 base_events.py
0.09 __init__.py
0.03 events.py
0.02 mandrill_cprofile.py
0.02 tasks.py
Incl. | Self | Called | Function | Location
100.00 0.00 (0) <built-in method builtins.e... ~
99.57 99.57 96 <method 'control' of 'selec... ~
0.31 0.01 142 <method 'run' of 'Context' ... ~
0.02 0.02 39 <method 'write' of '_io.Text... ~
0.02 0.00 41 <method 'set_result' of '_a... ~
0.01 0.01 39 <built-in method time.strft... ~
0.01 0.01 39 <built-in method time.local... ~
0.01 0.01 2 <function socket.close at ... ~
0.01 0.00 56 <built-in method _asyncio.... ~
0.00 0.00 26 <built-in method _heapq.h... ~
0.00 0.00 225 <built-in method builtins.is... ~
0.00 0.00 119 <built-in method builtins.m... ~
0.00 0.00 343 <built-in method builtins.le... ~
0.00 0.00 8 <built-in method posix.ura... ~
0.00 0.00 157 <built-in method time.mon... ~
0.00 0.00 180 <built-in method builtins.h... ~
0.00 0.00 38 <built-in method _heapq.h... ~
0.00 0.00 117 <method 'rfind' of 'str' obj... ~
0.00 0.00 30 <method 'cancel' of '_asyn... ~
0.00 0.00 80 <method 'acquire' of '_thre... ~
0.00 0.00 28 <method 'rsplit' of 'str' obj... ~
0.00 0.00 156 <built-in method posix.fsp... ~
0.00 0.00 25 <built-in method _abc._ab... ~
0.00 0.00 189 <method 'append' of 'colle... ~
0.00 0.00 103 <method 'get' of 'dict' obje... ~
0.00 0.00 39 <method 'find' of 'str' obje... ~
0.00 0.00 59 <built-in method builtins.g... ~
0.00 0.00 157 <method 'popleft' of 'colle... ~
0.00 0.00 39 <method 'flush' of '_io.Text... ~
0.00 0.00 80 <built-in method _thread.g... ~
0.00 0.00 2 <built-in method _abc._ab... ~
0.00 0.00 3 <method 'recv' of '_socket.... ~
0.00 0.00 39 <built-in method sys._getfr... ~
0.00 0.00 74 <method 'add' of 'set' obje... ~
0.00 0.00 28 <built-in method builtins.r... ~
0.00 0.00 36 <method 'join' of 'str' obje... ~
0.00 0.00 39 <built-in method time.time> ~
0.00 0.00 38 <method 'add_done_callba... ~
0.00 0.00 25 <built-in method builtins.m... ~
0.00 0.00 80 <method 'release' of '_thre... ~
0.00 0.00 39 <built-in method posix.get... ~
<built-in method builtins.exec>
Types Callers All Callers Callee Map Source Code
Event Type | Incl. | Self | Short | Formula
Nanoseconds 100.00 0.00 ns
ns | ns per call | Cc | Cc | Count | Callee
100.00 4 727 495 000 1 <module> (<string>)
Parts Callees Call Graph All Callees Caller Map Machine Code
Show Percentage relative to Parent

profiling asyncio code

line_profiler

line_profiler

```
@profile
async def save(msg):
    # unhelpful simulation of i/o work
    await asyncio.sleep(random.random())
    msg.saved = True
    logging.info(f"Saved {msg} into database")
```

line_profiler

```
@profile
async def save(msg):
    # unhelpful simulation of i/o work
    await asyncio.sleep(random.random())
    msg.saved = True
    logging.info(f"Saved {msg} into database")
```

line_profiler

```
$ timeout -s INT 5s kernprof -o mayhem.prof --line-by-line mayhem.py
$ python -m line_profiler mayhem.prof
```

line_profiler

```
$ timeout -s INT 5s kernprof -o mayhem.prof --line-by-line mayhem.py
$ python -m line_profiler mayhem.prof

Timer unit: 1e-06 s

Total time: 0.002202 s
File: mayhem.py
Function: save at line 69

Line #   Hits     Time  Per Hit   % Time  Line Contents
==============================================================
    69                                    @profile
    70                                    async def save(msg):
    71        8    259.0     32.4     11.8        await asyncio.sleep(random.r
    72        8     26.0      3.2      1.2        msg.saved = True
    73        8   1917.0    239.6     87.1        logging.info(f"Saved {msg} i
```

line_profiler

```
$ timeout -s INT 5s kernprof -o mayhem.prof --line-by-line mayhem.py
$ python -m line_profiler mayhem.prof

Timer unit: 1e-06 s

Total time: 0.002202 s
File: mayhem.py
Function: save at line 69

Line #    Hits         Time  Per Hit   % Time  Line Contents
==============================================================
    69                                           @profile
    70                                           async def save(msg):
    71         8     259.0     32.4     11.8          await asyncio.sleep(random.r
    72         8      26.0      3.2      1.2          msg.saved = True
    73         8    1917.0    239.6     87.1          logging.info(f"Saved {msg} i
```

line_profiler

```
$ timeout -s INT 5s kernprof -o mayhem.prof --line-by-line mayhem.py
$ python -m line_profiler mayhem.prof

Timer unit: 1e-06 s

Total time: 0.002202 s
File: mayhem.py
Function: save at line 69

Line #    Hits         Time  Per Hit   % Time  Line Contents
==============================================================
    69                                           @profile
    70                                           async def save(msg):
    71         8     259.0     32.4     11.8          await asyncio.sleep(random.r
    72         8      26.0      3.2      1.2          msg.saved = True
    73         8    1917.0    239.6     87.1          logging.info(f"Saved {msg} i
```

line_profiler

```
import aiologger

logger = aiologger.Logger.with_default_handlers()
```

line_profiler

```
$ timeout -s INT 5s kernprof -o mayhem.prof --line-by-line mayhem.py
$ python -m line_profiler mayhem.prof

Timer unit: 1e-06 s

Total time: 0.0011 s
File: mayhem.py
Function: save at line 69

Line #   Hits      Time  Per Hit    % Time  Line Contents
==============================================================
    69                                      @profile
    70                                      async def save(msg):
    71       7    269.0      38.4      24.5      await asyncio.sleep(random.r
    72       5     23.0       4.6       2.1      msg.saved = True
    73       5    808.0     161.6      73.5      await logger.info(f"Saved {m
```

line_profiler

```
$ timeout -s INT 5s kernprof -o mayhem.prof --line-by-line mayhem.py
$ python -m line_profiler mayhem.prof

Timer unit: 1e-06 s

Total time: 0.0011 s
File: mayhem.py
Function: save at line 69

Line #    Hits         Time  Per Hit   % Time  Line Contents
==============================================================
    69                                           @profile
    70                                           async def save(msg):
    71         7    269.0     38.4     24.5          await asyncio.sleep(random.r
    72         5     23.0      4.6      2.1          msg.saved = True
    73         5    808.0    161.6     73.5          await logger.info(f"Saved {m
```

line_profiler

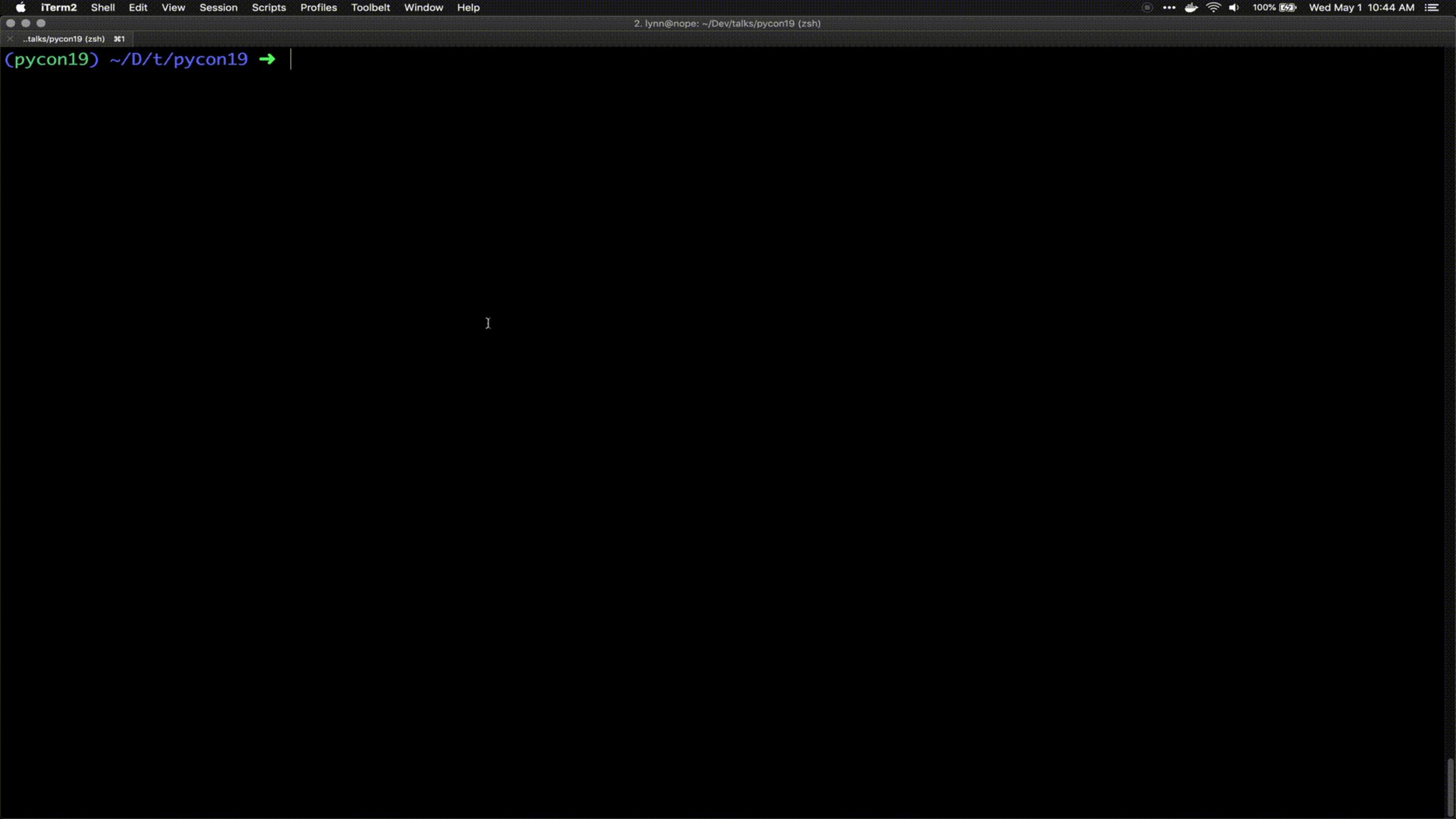
```
$ timeout -s INT 5s kernprof -o mayhem.prof --line-by-line mayhem.py
$ python -m line_profiler mayhem.prof

Timer unit: 1e-06 s

Total time: 0.0011 s
File: mayhem.py
Function: save at line 69

Line #    Hits         Time  Per Hit   % Time  Line Contents
==============================================================
    69                                           @profile
    70                                           async def save(msg):
    71         7    269.0     38.4     24.5          await asyncio.sleep(random.r
    72         5     23.0      4.6      2.1          msg.saved = True
    73         5    808.0    161.6     73.5          await logger.info(f"Saved {m
```

profiling asyncio code

live profiling

live profiling

```
$ profiling live-profile --mono mayhem.py
```


iTerm2 Shell Edit View Session Scripts Profiles Toolbelt Window Help
100% Wed May 1 10:44 AM
2. lynn@nope: ~/Dev/talks/pycon19 (zsh)
..talks/pycon19 (zsh) ⌘1
(pycon19) ~/D/t/pycon19

profiling asyncio code

- No idea? Visualize with **`cProfile`** with **`pyprof2calltree`** + KCacheGrind
- Some idea? **`line_profiler`**
- f-it, we'll do it live: **`profiling live-profiler`**

Thank you!

rogue.ly/adv-aio

Lynn Root | SRE | @roguelynn

Spotify