Concurrency with Gevent

Marconi Moreto

@marconimjr

Concurrency

From Wikipedia:

“... concurrency is a property of systems in which several computations are executing simultaneously, and potentially interacting with each other”

Gevent

From http://www.gevent.org

“gevent is a coroutine-based Python networking library that uses greenlet to provide a high-level synchronous API on top of the libevent event loop”

Concurrency is not Parallelism

“Concurrency is not parallelism, although it enables parallelism. If you have only one processor, your program can still be concurrent but it cannot be parallel.”

- Rob Pike

Lets see some code

Stand back, I know concurrency

Synchronous

```
import time
import random

def worker(multiplier):
    time.sleep(random.random())
    print '*' * multiplier

for i in range(1, 6):
    worker(i)
```

Synchronous

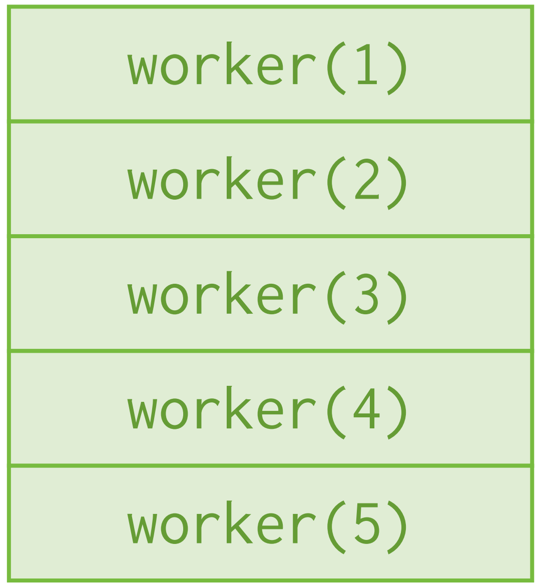
```
import time
import random

def worker(multiplier):
    time.sleep(random.random())
    print '*' * multiplier

for i in range(1, 6):
    worker(i)
```

Output:

```
*
**
***
****
*****
```

Synchronous

```
import time
import random

def worker(multiplier):
    time.sleep(random.random())
    print '*' * multiplier

for i in range(1, 6):
    worker(i)
```

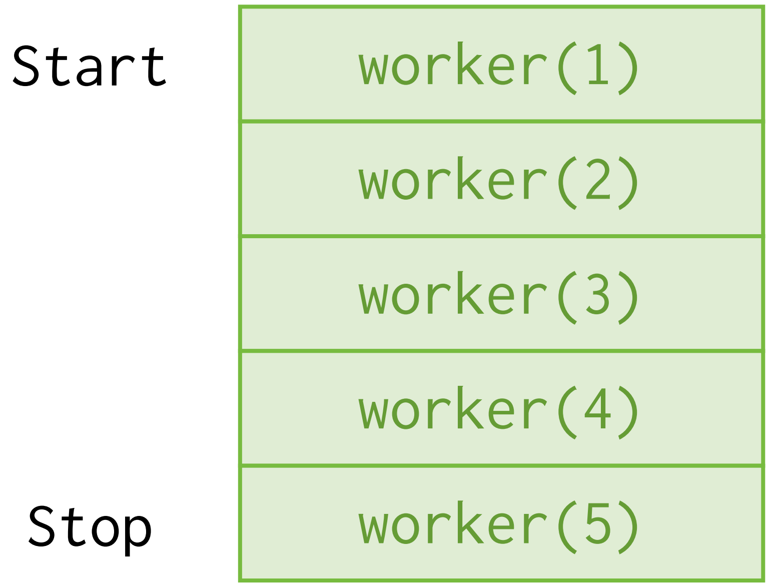


Output:

```
*
**
***
****
*****
```

Synchronous

```
import time
import random

def worker(multiplier):
    time.sleep(random.random())
    print '*' * multiplier

for i in range(1, 6):
    worker(i)
```

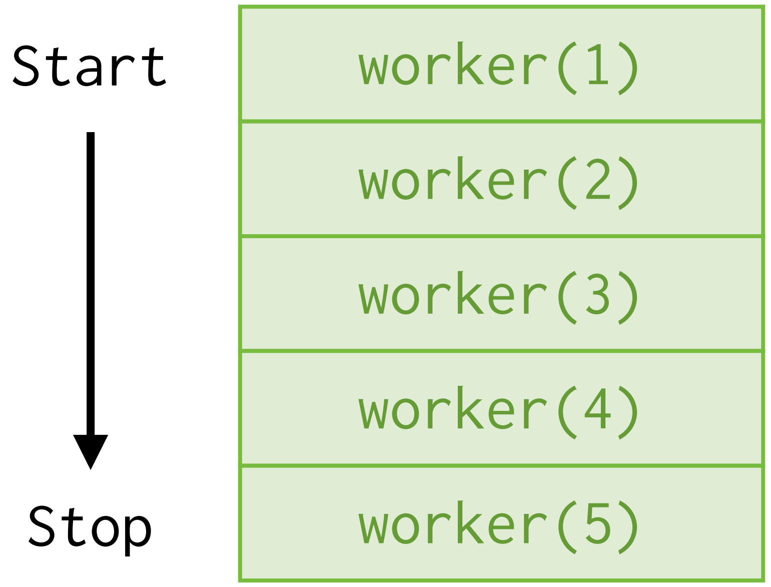


Output:

```
*
**
***
****
*****
```

Synchronous

```
import time
import random

def worker(multiplier):
    time.sleep(random.random())
    print '*' * multiplier

for i in range(1, 6):
    worker(i)
```



Output:

```
*
**
***
****
*****
```

Asynchronous printing workers

```
import gevent
import random

def worker(multiplier):
    gevent.sleep(random.random())
    print '*' * multiplier

greenlets = [gevent.spawn(worker, i)
             for i in range(1, 6)]
gevent.joinall(greenlets)
```

Asynchronous printing workers

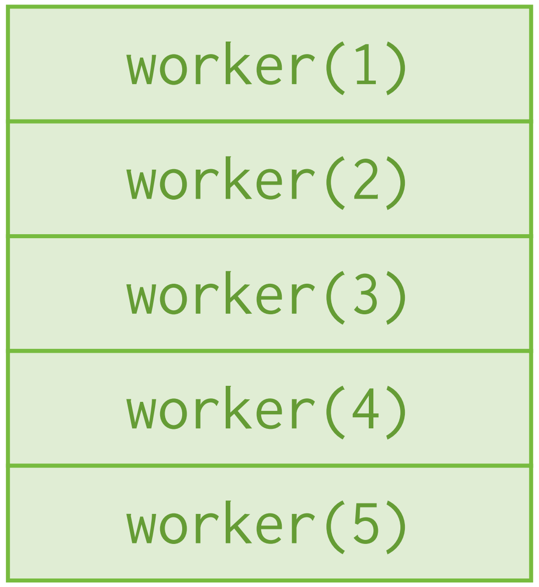
```
import gevent
import random

def worker(multiplier):
    gevent.sleep(random.random())
    print '*' * multiplier

greenlets = [gevent.spawn(worker, i)
             for i in range(1, 6)]
gevent.joinall(greenlets)
```

Output:

```
*****
****
*
**
***
```

Asynchronous printing workers

```
import gevent
import random

def worker(multiplier):
    gevent.sleep(random.random())
    print '*' * multiplier

greenlets = [gevent.spawn(worker, i)
             for i in range(1, 6)]
gevent.joinall(greenlets)
```

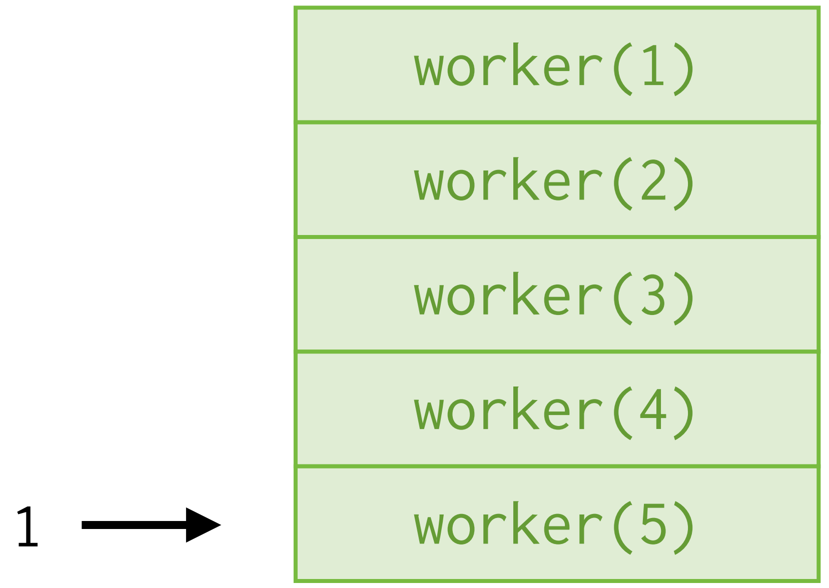


Output:

```
*****
****
*
**
***
```

Asynchronous printing workers

```
import gevent
import random

def worker(multiplier):
    gevent.sleep(random.random())
    print '*' * multiplier

greenlets = [gevent.spawn(worker, i)
             for i in range(1, 6)]
gevent.joinall(greenlets)
```

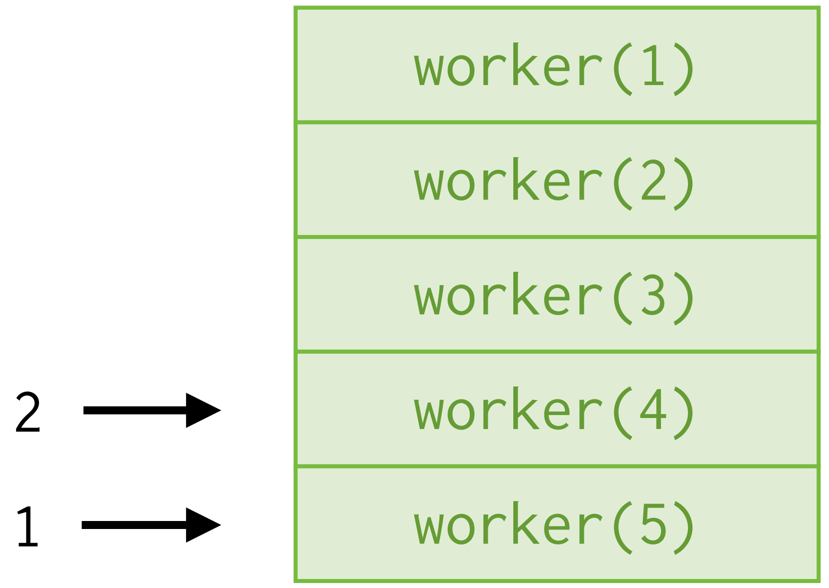


Output:

```
*****
****
*
**
***
```

Asynchronous printing workers

```
import gevent
import random

def worker(multiplier):
    gevent.sleep(random.random())
    print '*' * multiplier

greenlets = [gevent.spawn(worker, i)
             for i in range(1, 6)]
gevent.joinall(greenlets)
```

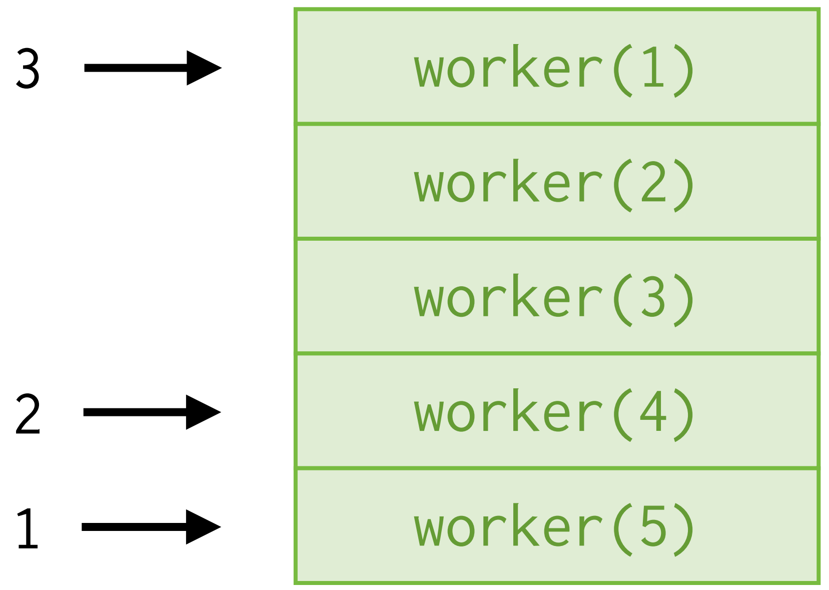


Output:

```
*****
****
*
**
***
```

Asynchronous printing workers

```python
import gevent
import random

def worker(multiplier):
    gevent.sleep(random.random())
    print '*' * multiplier

greenlets = [gevent.spawn(worker, i)
             for i in range(1, 6)]
gevent.joinall(greenlets)
```

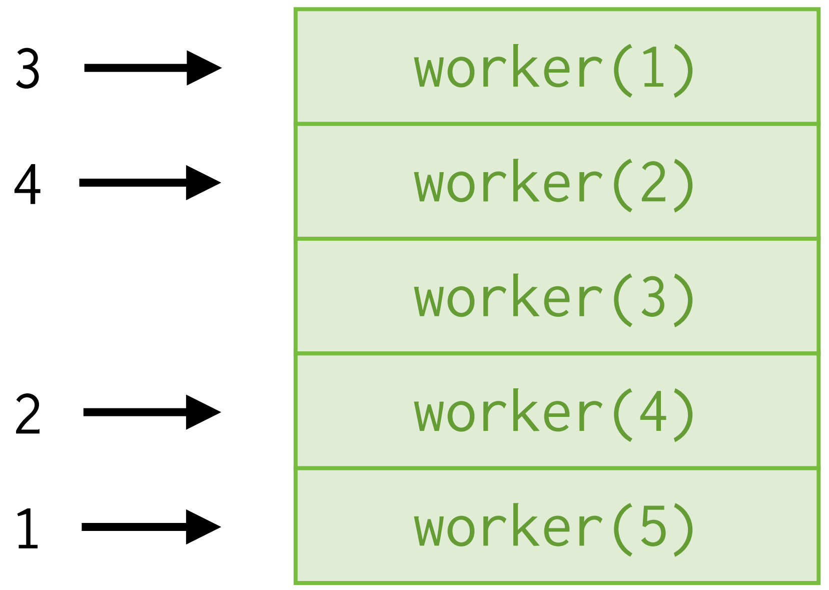


Output:

```
*****
****
*
**
***
```

Asynchronous printing workers

```
import gevent
import random

def worker(multiplier):
    gevent.sleep(random.random())
    print '*' * multiplier

greenlets = [gevent.spawn(worker, i)
                 for i in range(1, 6)]
gevent.joinall(greenlets)
```

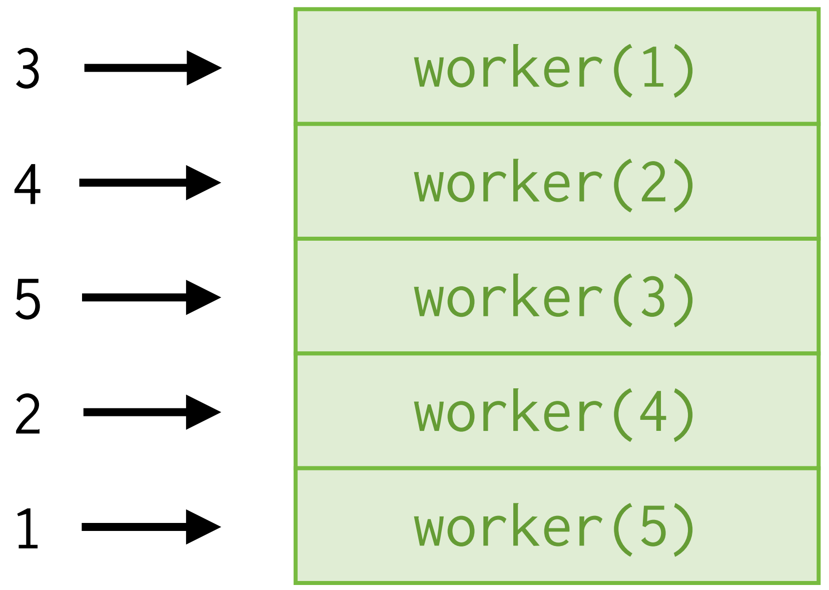


Output:

```
*****
****
*
**
***
```

Asynchronous printing workers

```python
import gevent
import random

def worker(multiplier):
    gevent.sleep(random.random())
    print '*' * multiplier

greenlets = [gevent.spawn(worker, i)
             for i in range(1, 6)]
gevent.joinall(greenlets)
```



Output:

```
*****
****
*
**
***
```

Workers with timeout

```
import gevent
import random

def worker(multiplier):
    gevent.sleep(random.random())
    print '*' * multiplier

greenlets = [gevent.spawn(worker, i)
             for i in range(1, 6)]
gevent.joinall(greenlets, timeout=0.5)
```

Workers with timeout

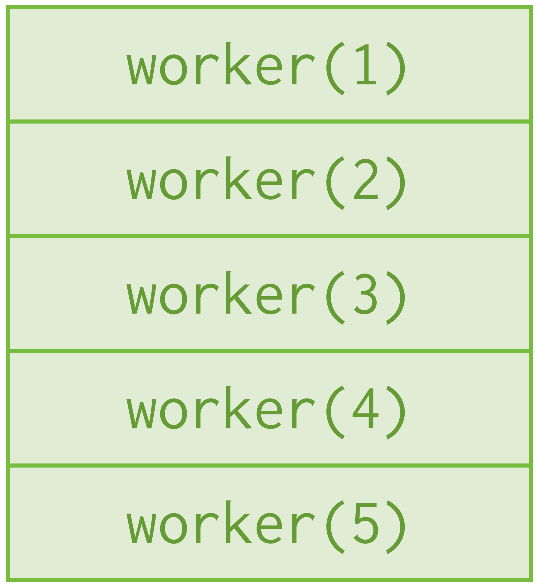
```
import gevent
import random

def worker(multiplier):
    gevent.sleep(random.random())
    print '*' * multiplier

greenlets = [gevent.spawn(worker, i)
             for i in range(1, 6)]
gevent.joinall(greenlets, timeout=0.5)
```

Output:

```
****
*
***
```

Workers with timeout

```
import gevent
import random

def worker(multiplier):
    gevent.sleep(random.random())
    print '*' * multiplier

greenlets = [gevent.spawn(worker, i)
             for i in range(1, 6)]
gevent.joinall(greenlets, timeout=0.5)
```

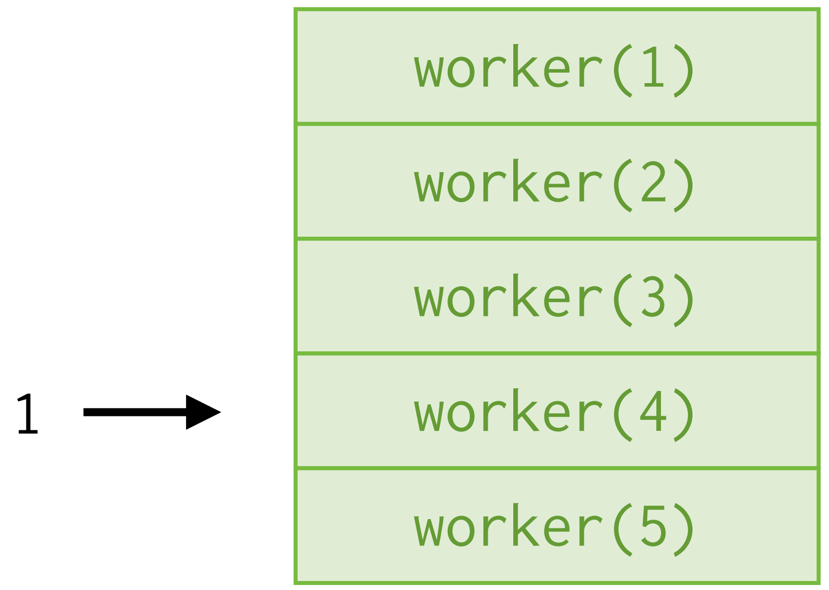


Output:

```
****
*
***
```

Workers with timeout

```
import gevent
import random

def worker(multiplier):
    gevent.sleep(random.random())
    print '*' * multiplier

greenlets = [gevent.spawn(worker, i)
             for i in range(1, 6)]
gevent.joinall(greenlets, timeout=0.5)
```

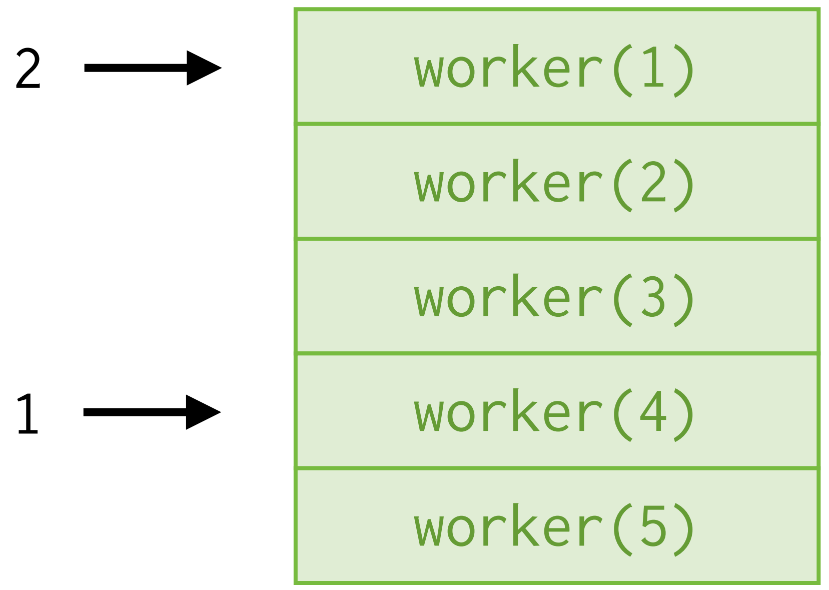


Output:

```
****
*
***
```

Workers with timeout

```
import gevent
import random

def worker(multiplier):
    gevent.sleep(random.random())
    print '*' * multiplier

greenlets = [gevent.spawn(worker, i)
             for i in range(1, 6)]
gevent.joinall(greenlets, timeout=0.5)
```

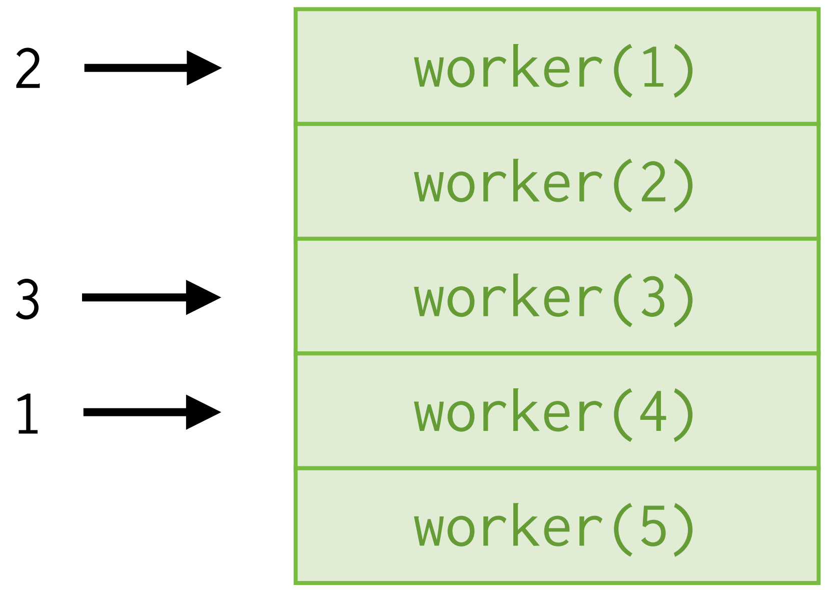


Output:

```
****
*
***
```

Workers with timeout

```
import gevent
import random

def worker(multiplier):
    gevent.sleep(random.random())
    print '*' * multiplier

greenlets = [gevent.spawn(worker, i)
             for i in range(1, 6)]
gevent.joinall(greenlets, timeout=0.5)
```

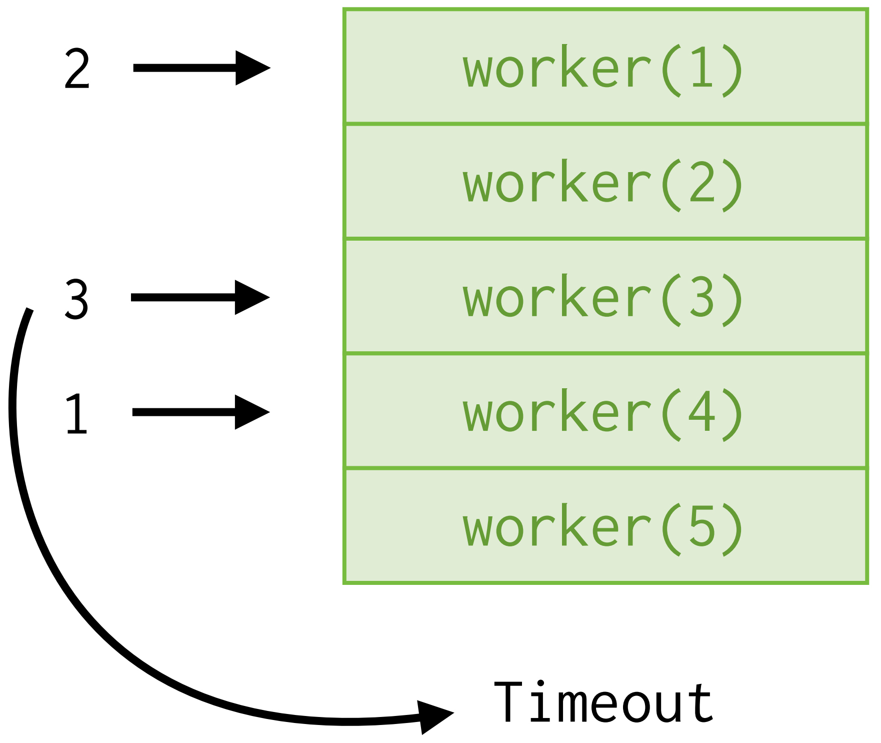


Output:

```
****
*
***
```

Workers with timeout

```
import gevent
import random

def worker(multiplier):
    gevent.sleep(random.random())
    print '*' * multiplier

greenlets = [gevent.spawn(worker, i)
             for i in range(1, 6)]
gevent.joinall(greenlets, timeout=0.5)
```



Output:

```
****
*
***
```

Collecting workers result

```
import random
import gevent

def worker(multiplier):
    gevent.sleep(random.random())
    return '*' * multiplier

def producers():
    greenlets = [gevent.spawn(worker, i)
                 for i in range(1, 6)]
    gevent.joinall(greenlets)
    return [g.value for g in greenlets]

print '\n'.join(producers())
```

Collecting workers result

```
import random
import gevent

def worker(multiplier):
    gevent.sleep(random.random())
    return '*' * multiplier

def producers():
    greenlets = [gevent.spawn(worker, i)
                 for i in range(1, 6)]
    gevent.joinall(greenlets)
    return [g.value for g in greenlets]

print '\n'.join(producers())
```

Output:

```
*
**
***
****
*****
```

Collecting workers result

```
import random
import gevent

def worker(multiplier):
    gevent.sleep(random.random())
    return '*' * multiplier

def producers():
    greenlets = [gevent.spawn(worker, i)
                 for i in range(1, 6)]
    gevent.joinall(greenlets)
    return [g.value for g in greenlets]

print '\n'.join(producers())
```

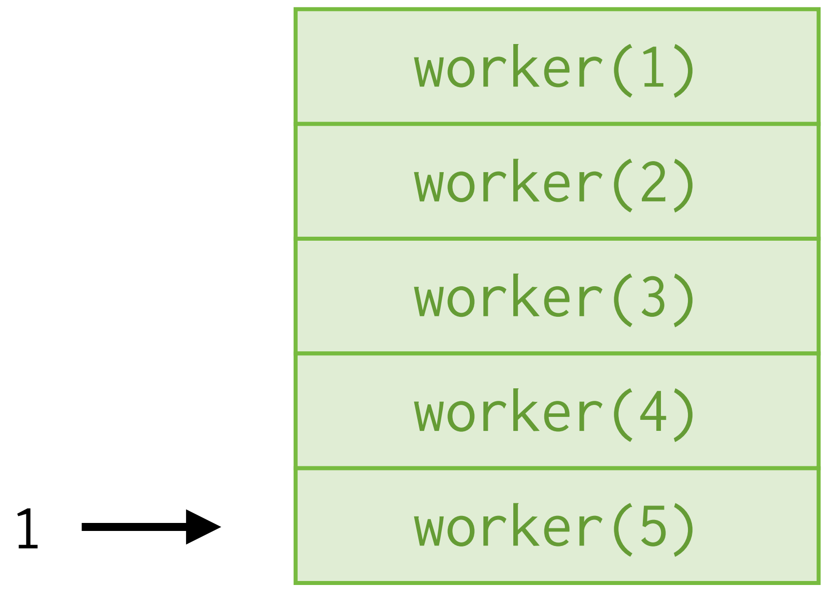
worker(1)
worker(2)
worker(3)
worker(4)
worker(5)

Output:

```
*
**
***
****
*****
```

Collecting workers result

```
import random
import gevent

def worker(multiplier):
    gevent.sleep(random.random())
    return '*' * multiplier

def producers():
    greenlets = [gevent.spawn(worker, i)
                 for i in range(1, 6)]
    gevent.joinall(greenlets)
    return [g.value for g in greenlets]

print '\n'.join(producers())
```

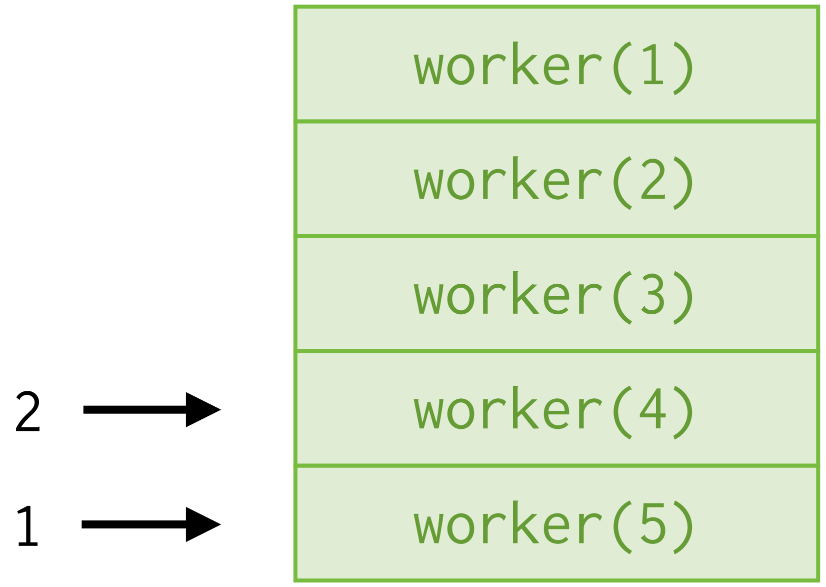


Output:

```
*
**
***
****
*****
```

Collecting workers result

```
import random
import gevent

def worker(multiplier):
    gevent.sleep(random.random())
    return '*' * multiplier

def producers():
    greenlets = [gevent.spawn(worker, i)
                 for i in range(1, 6)]
    gevent.joinall(greenlets)
    return [g.value for g in greenlets]

print '\n'.join(producers())
```

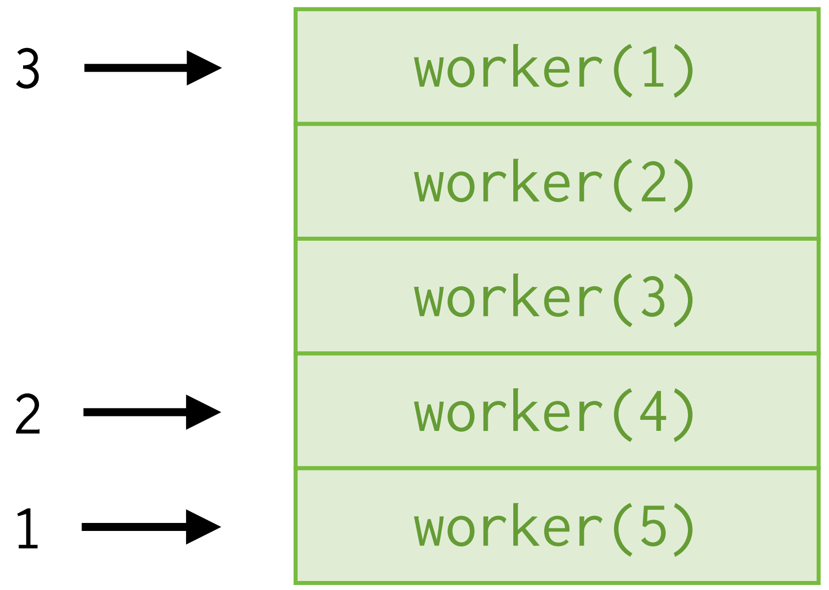


Output:

```
*
**
***
****
*****
```

Collecting workers result

```
import random
import gevent

def worker(multiplier):
    gevent.sleep(random.random())
    return '*' * multiplier

def producers():
    greenlets = [gevent.spawn(worker, i)
                     for i in range(1, 6)]
    gevent.joinall(greenlets)
    return [g.value for g in greenlets]

print '\n'.join(producers())
```

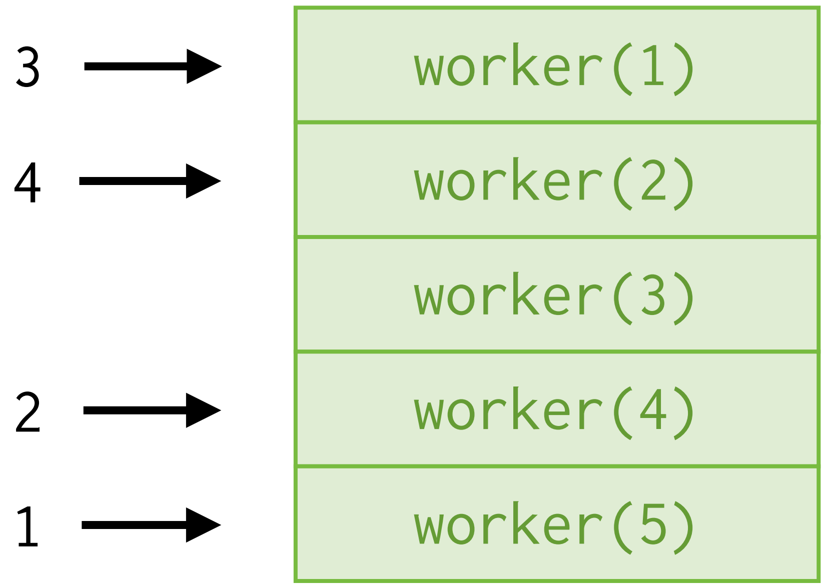


Output:

```
*
**
***
****
*****
```

Collecting workers result

```
import random
import gevent

def worker(multiplier):
    gevent.sleep(random.random())
    return '*' * multiplier

def producers():
    greenlets = [gevent.spawn(worker, i)
                         for i in range(1, 6)]
    gevent.joinall(greenlets)
    return [g.value for g in greenlets]

print '\n'.join(producers())
```

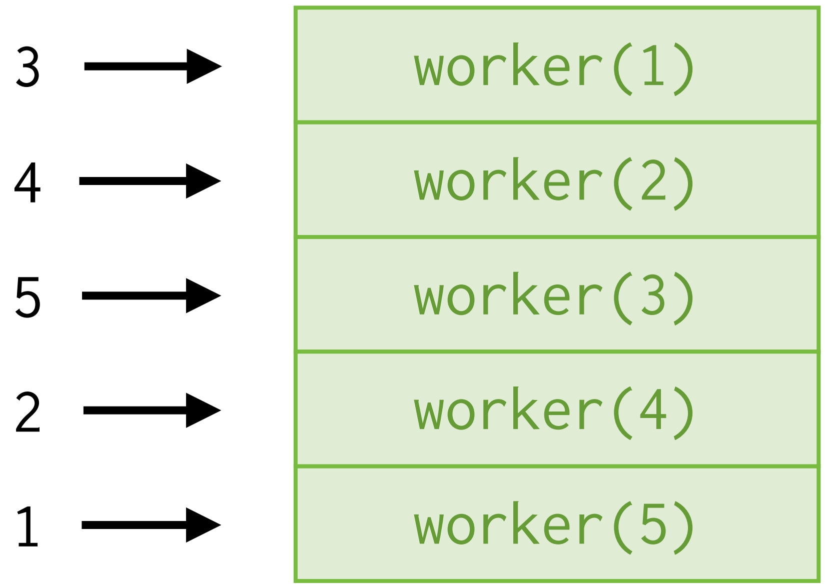


Output:

```
*
**
***
****
*****
```

Collecting workers result

```
import random
import gevent

def worker(multiplier):
    gevent.sleep(random.random())
    return '*' * multiplier

def producers():
    greenlets = [gevent.spawn(worker, i)
                 for i in range(1, 6)]
    gevent.joinall(greenlets)
    return [g.value for g in greenlets]

print '\n'.join(producers())
```



Output:

```
*
**
***
****
*****
```

Producers and consumer (1/2)

```
import random
import gevent
from gevent.queue import Queue

tasks = Queue()

def worker(multiplier):
    gevent.sleep(random.random())
    tasks.put('*' * multiplier)

def producers():
    greenlets = [gevent.spawn(worker, i)
                 for i in range(1, 6)]
    gevent.joinall(greenlets)

def consumer():
    while True:
        print tasks.get()

gevent.spawn(consumer)
producers()
```

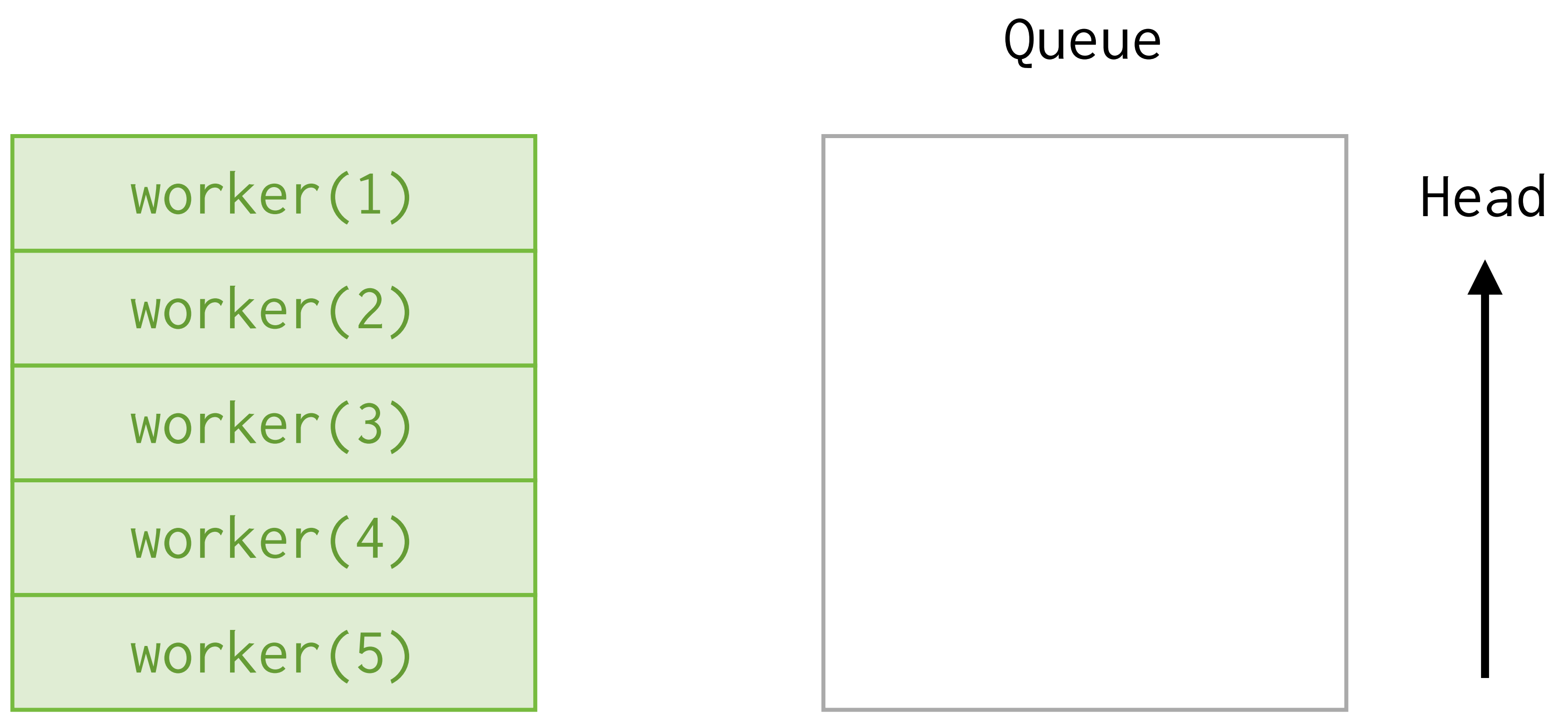
Producers and consumer (1/2)

```
import random
import gevent
from gevent.queue import Queue

tasks = Queue()

def worker(multiplier):
    gevent.sleep(random.random())
    tasks.put('*' * multiplier)

def producers():
    greenlets = [gevent.spawn(worker, i)
                 for i in range(1, 6)]
    gevent.joinall(greenlets)

def consumer():
    while True:
        print tasks.get()

gevent.spawn(consumer)
producers()
```

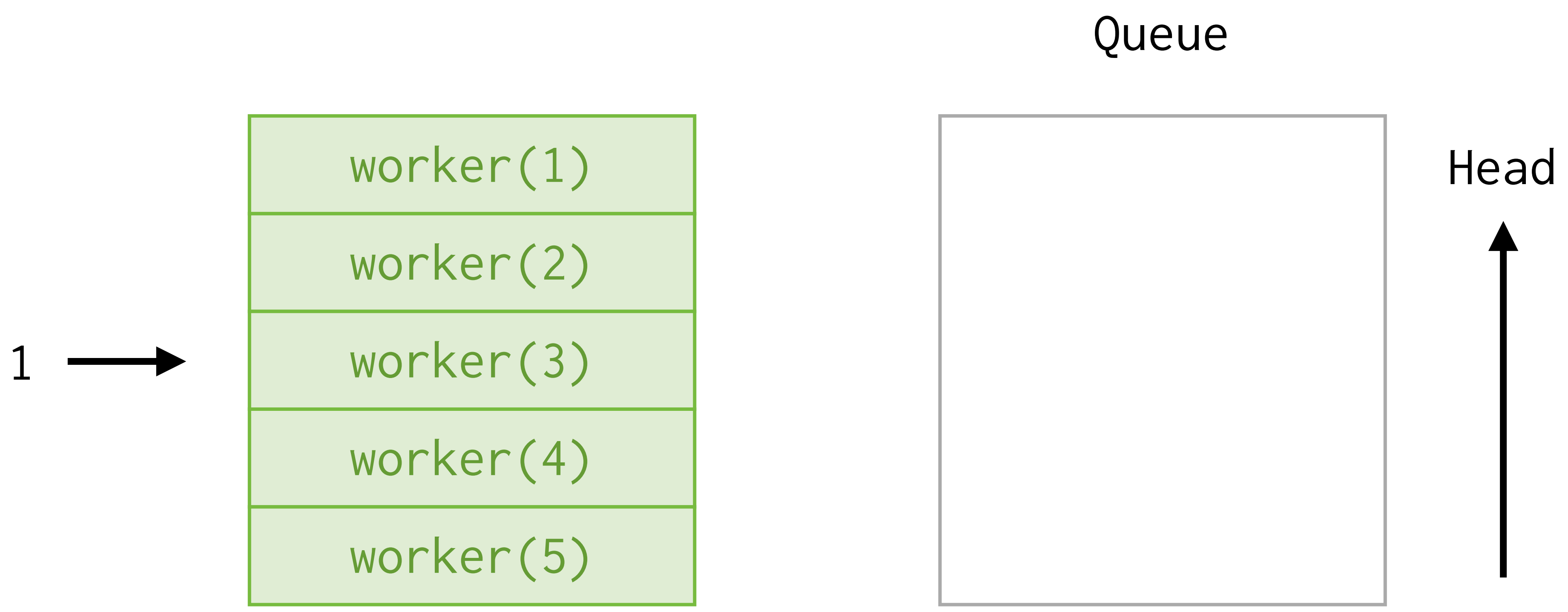
Output:

```
***
*
**
****
*****
```

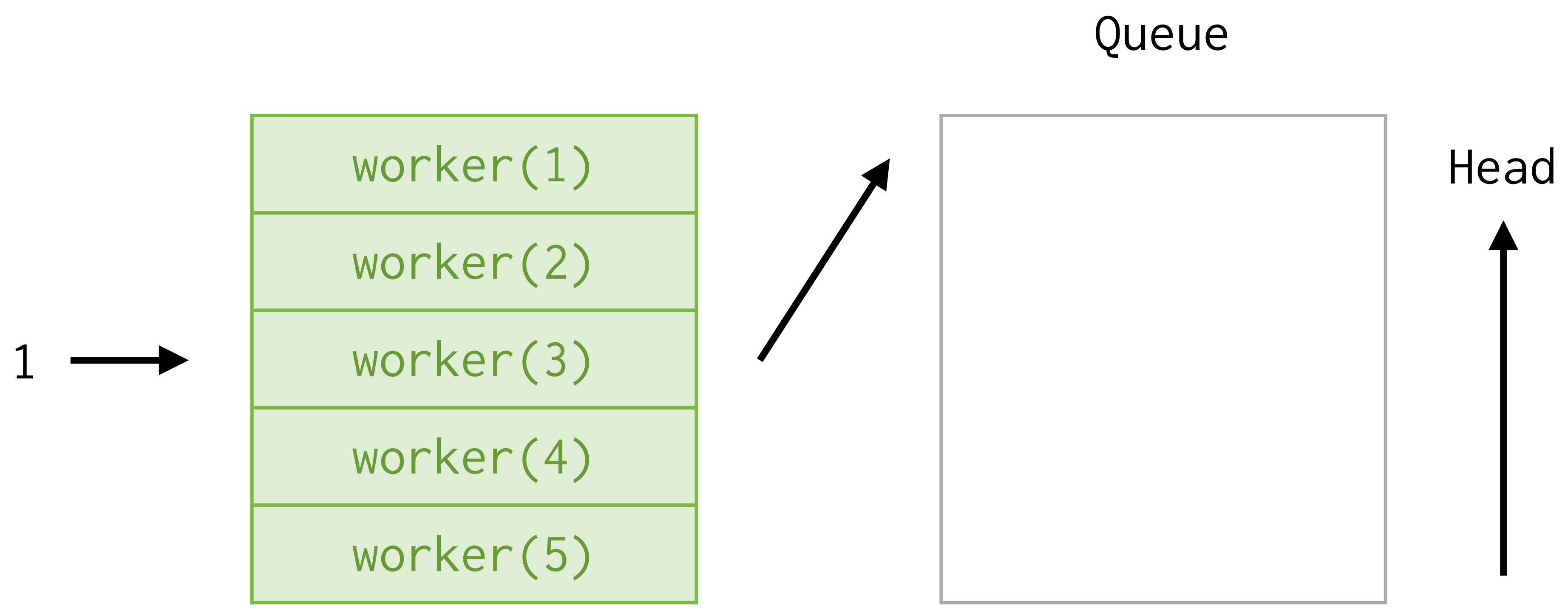
Producers and consumer (2/2)



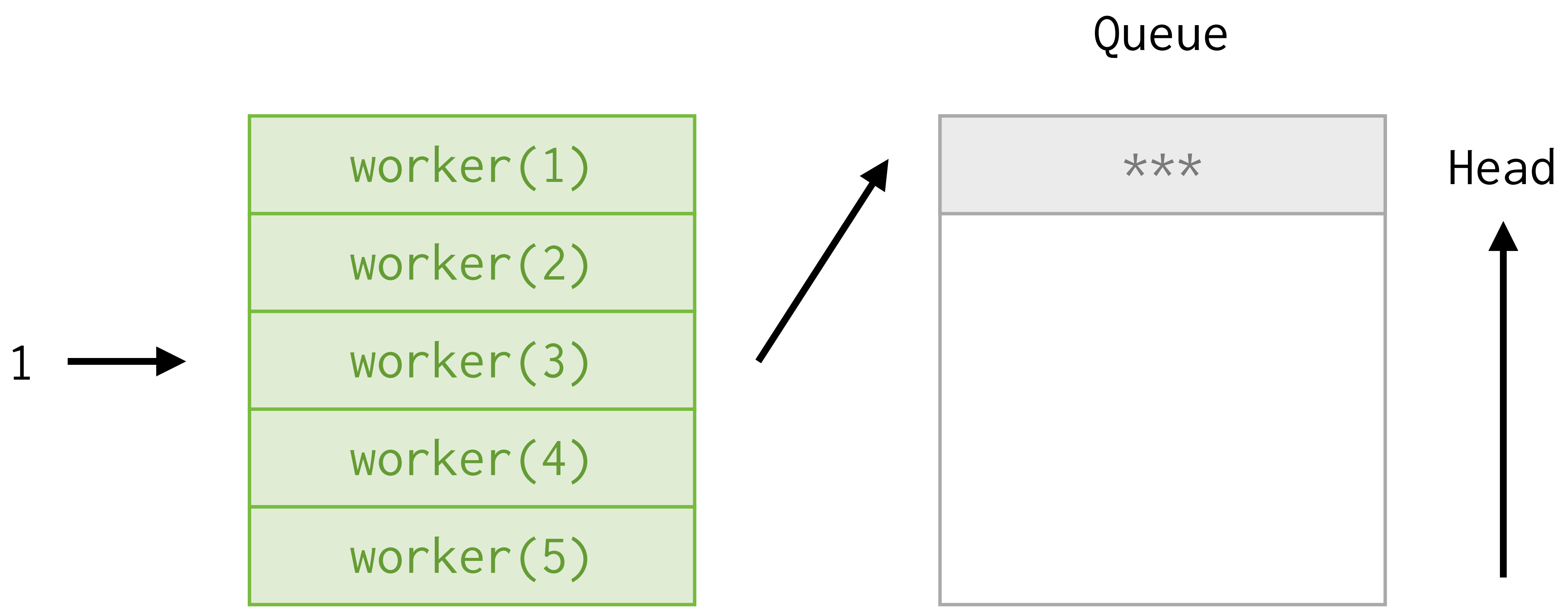
Producers and consumer (2/2)



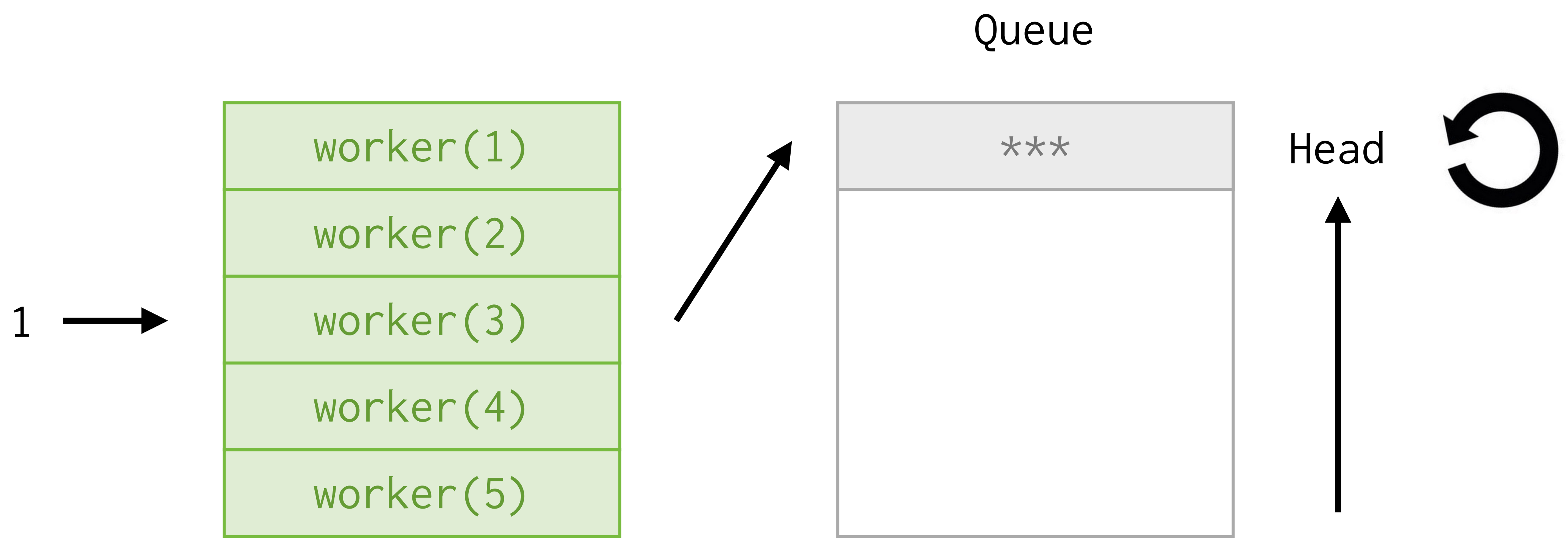
Producers and consumer (2/2)



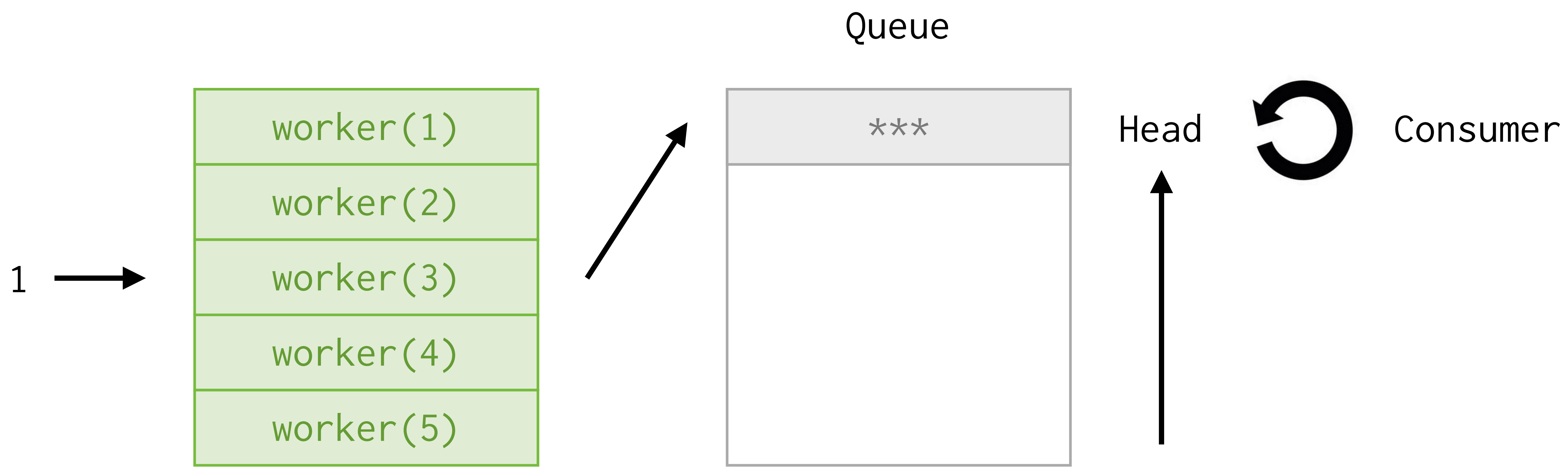
Producers and consumer (2/2)



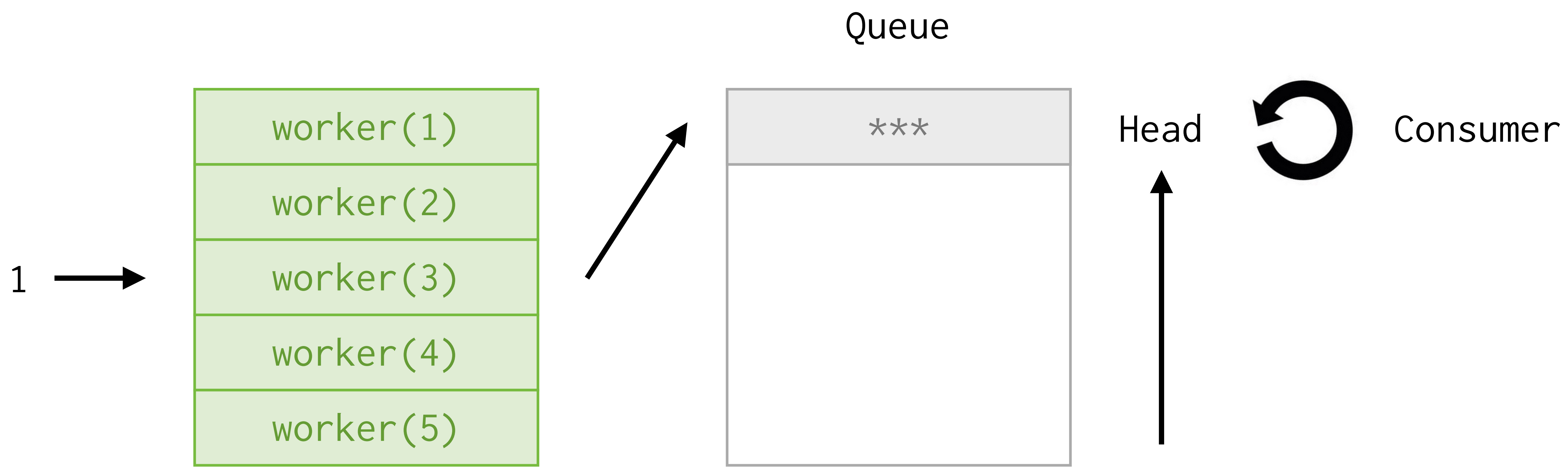
Producers and consumer (2/2)



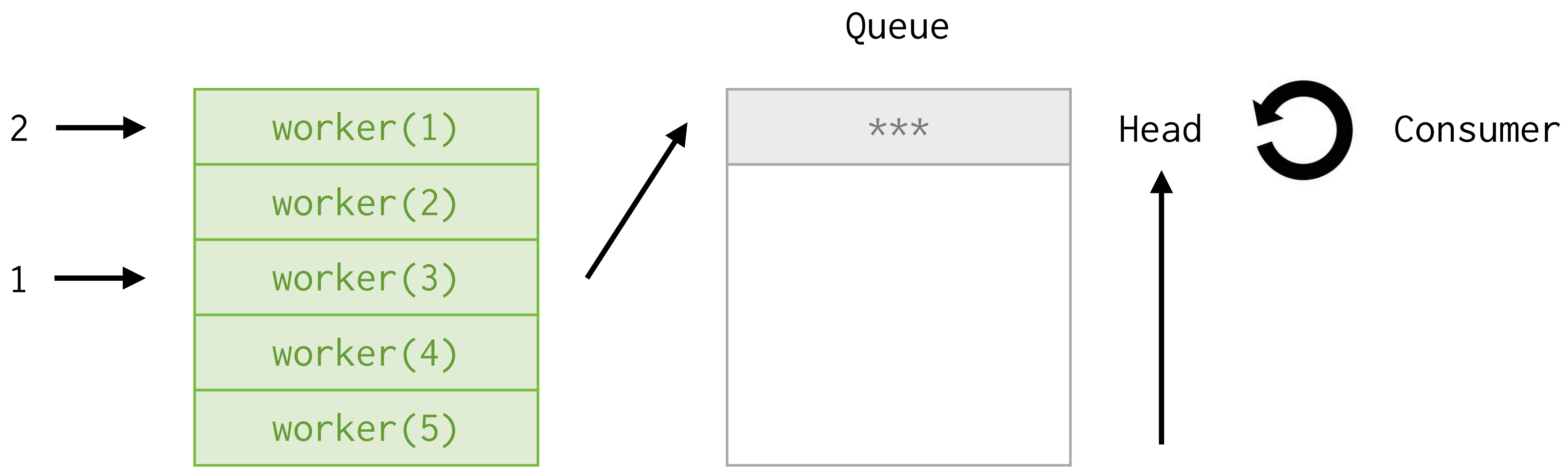
Producers and consumer (2/2)



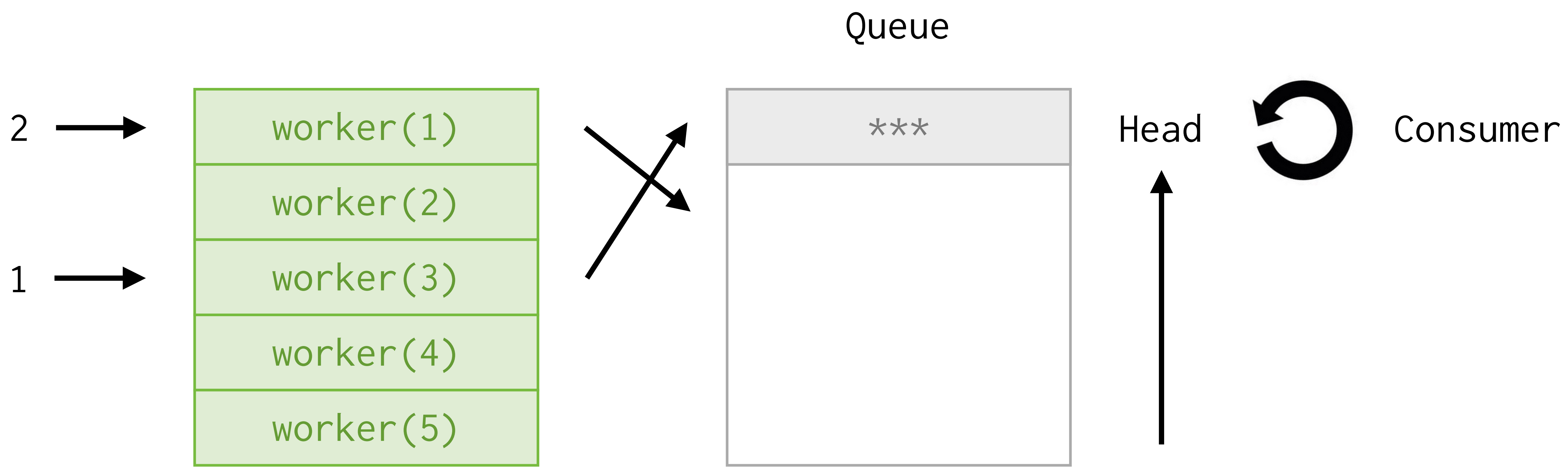
Producers and consumer (2/2)



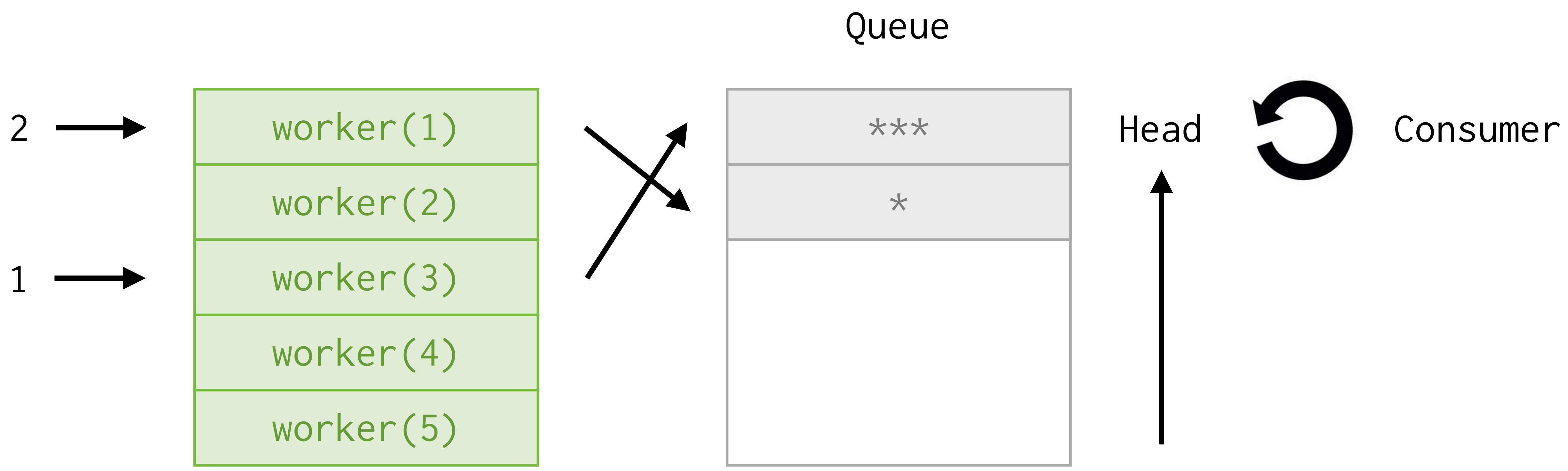
Producers and consumer (2/2)



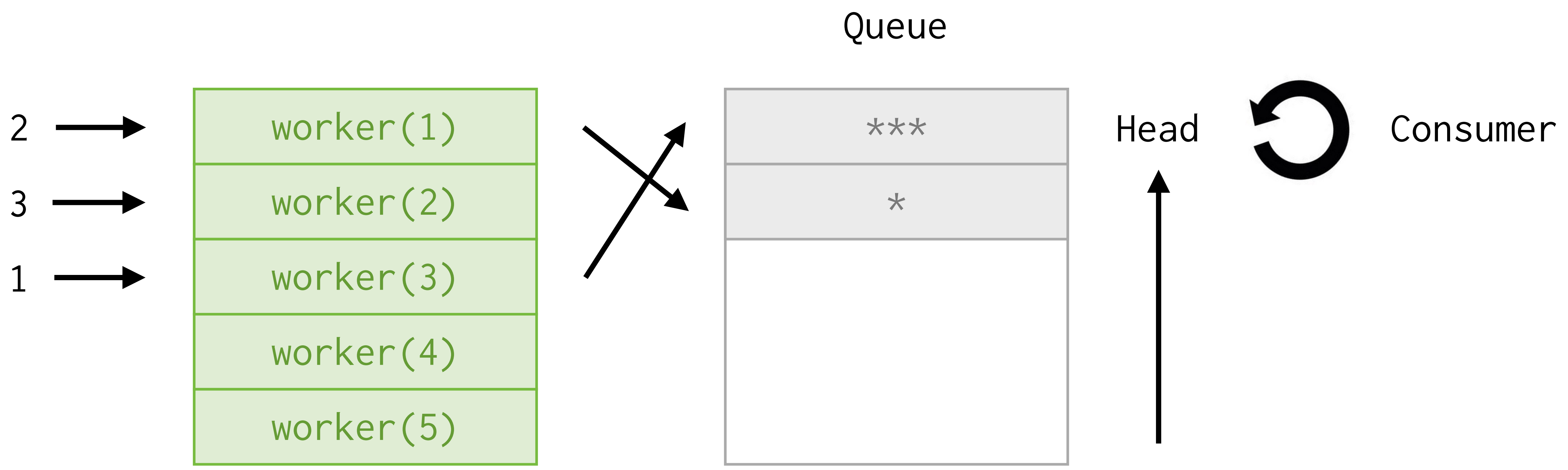
Producers and consumer (2/2)



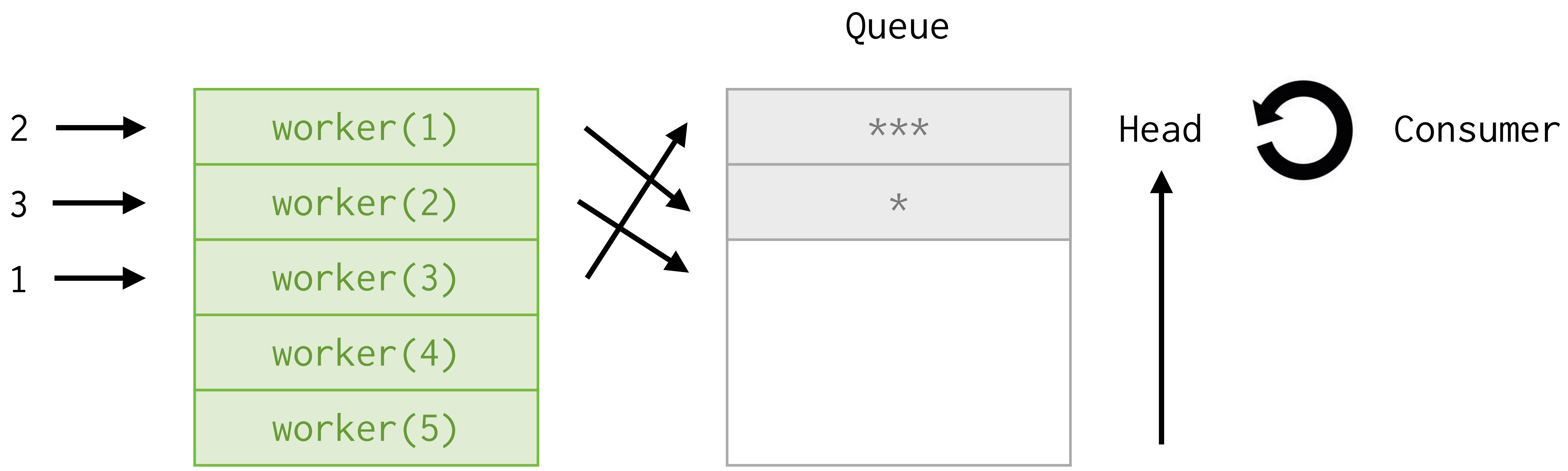
Producers and consumer (2/2)



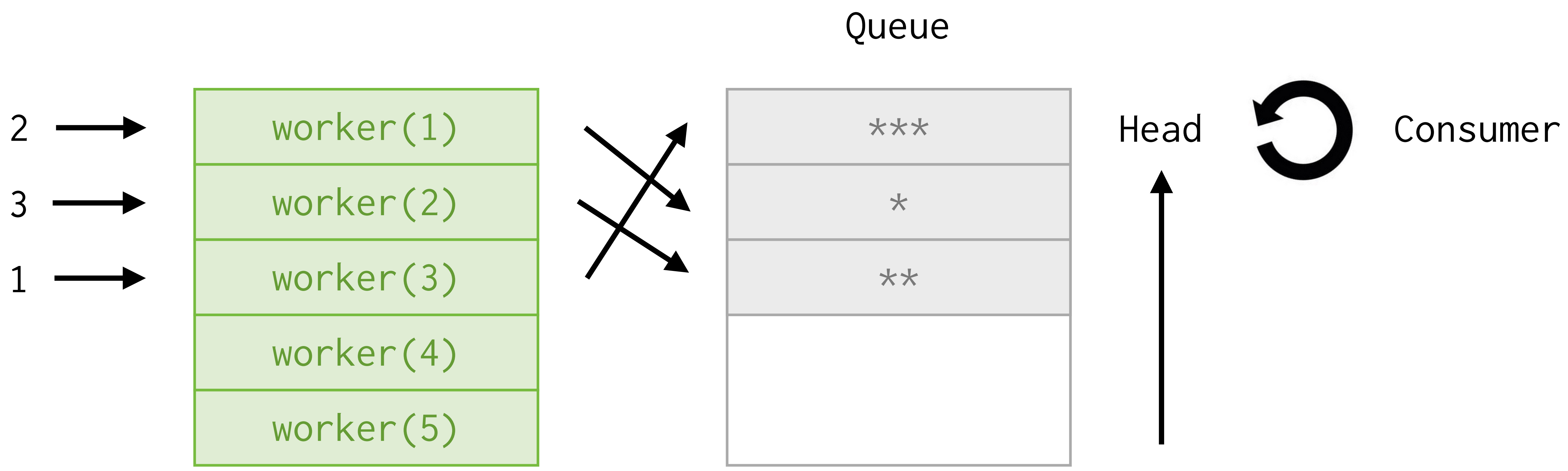
Producers and consumer (2/2)



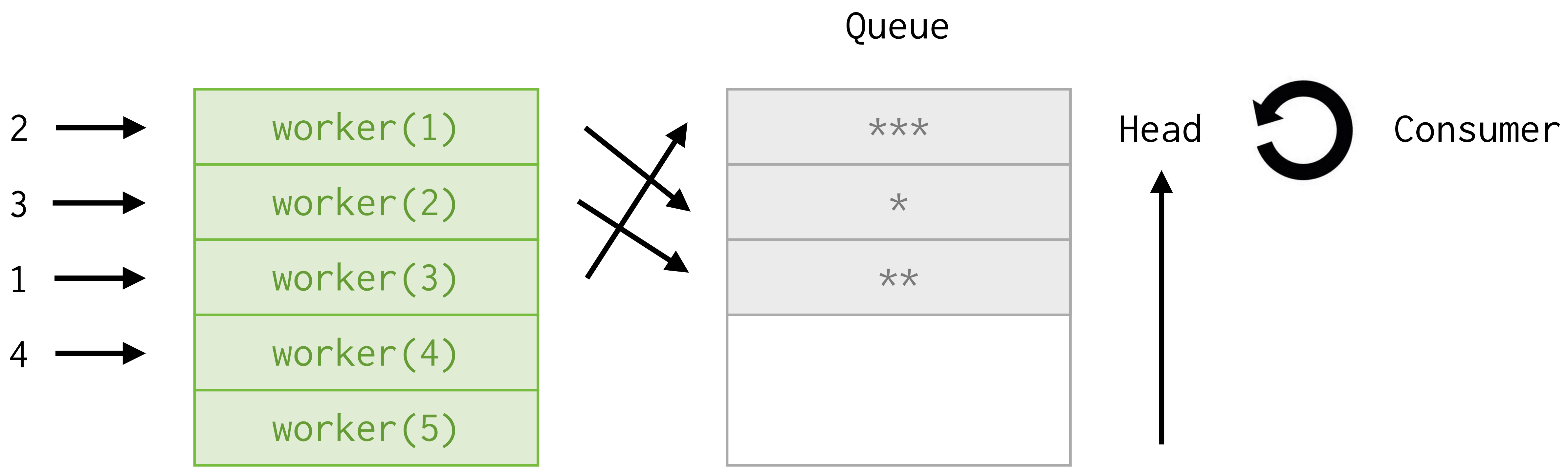
Producers and consumer (2/2)



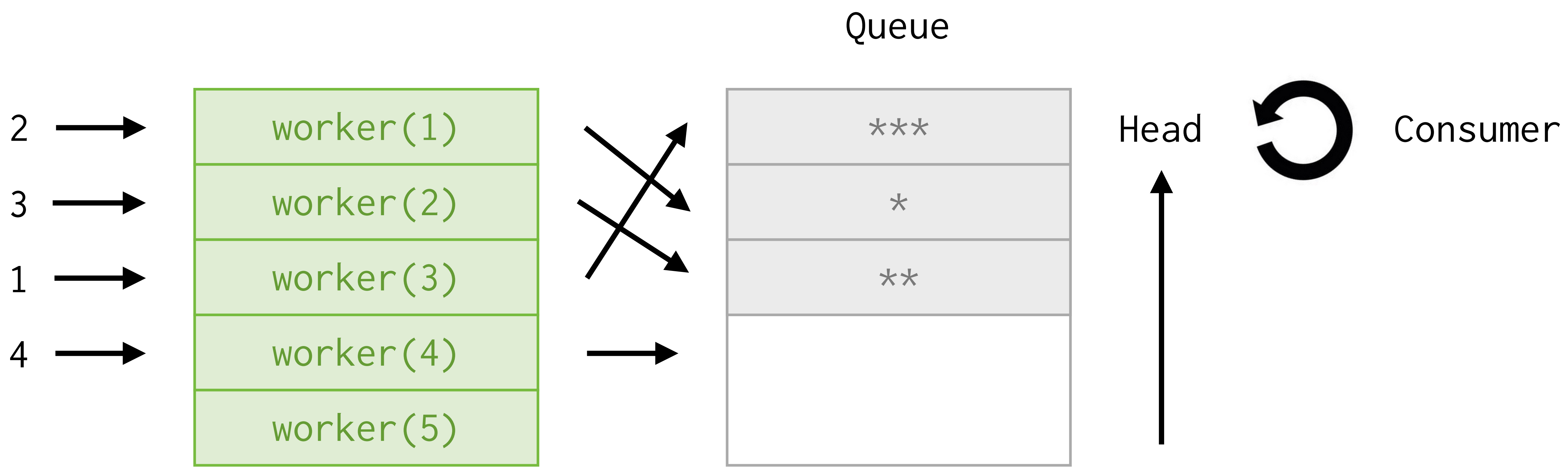
Producers and consumer (2/2)



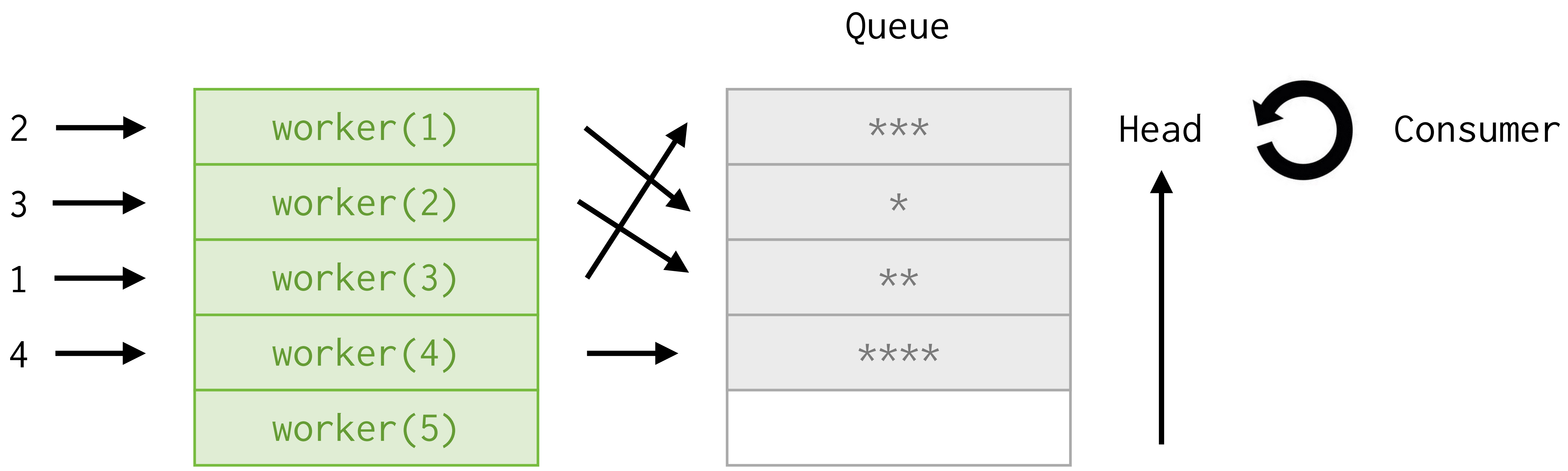
Producers and consumer (2/2)



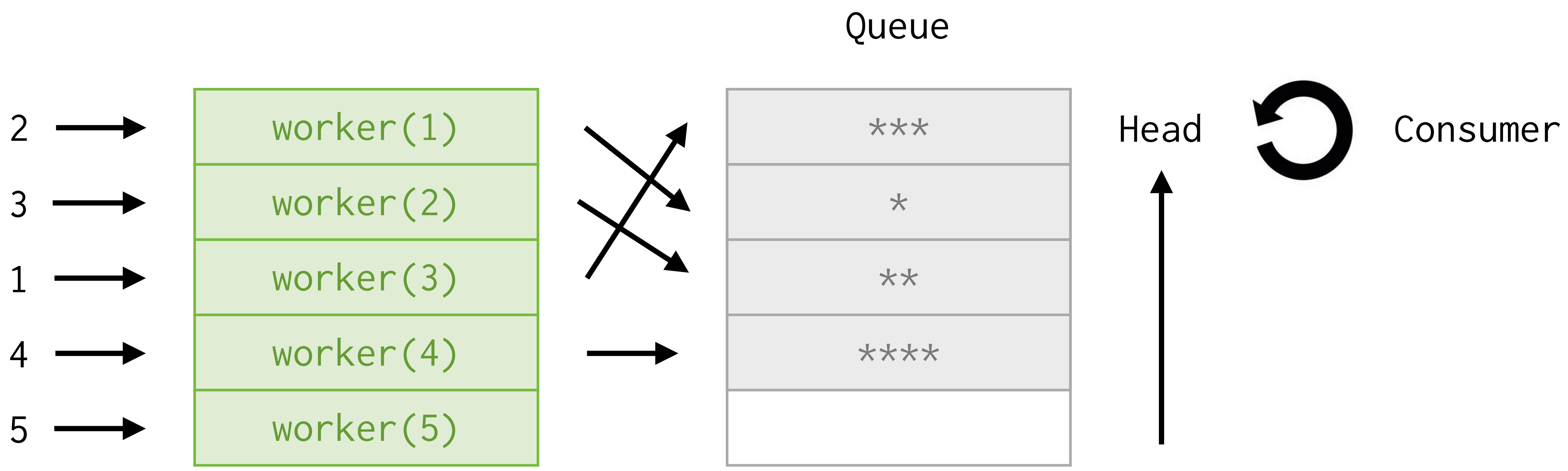
Producers and consumer (2/2)



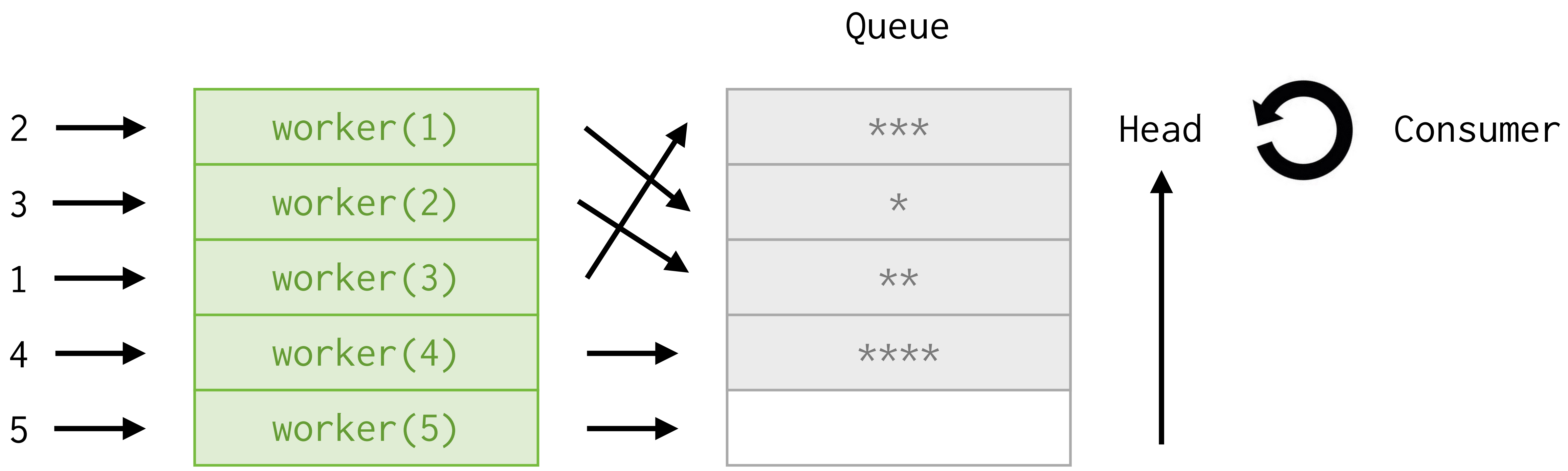
Producers and consumer (2/2)



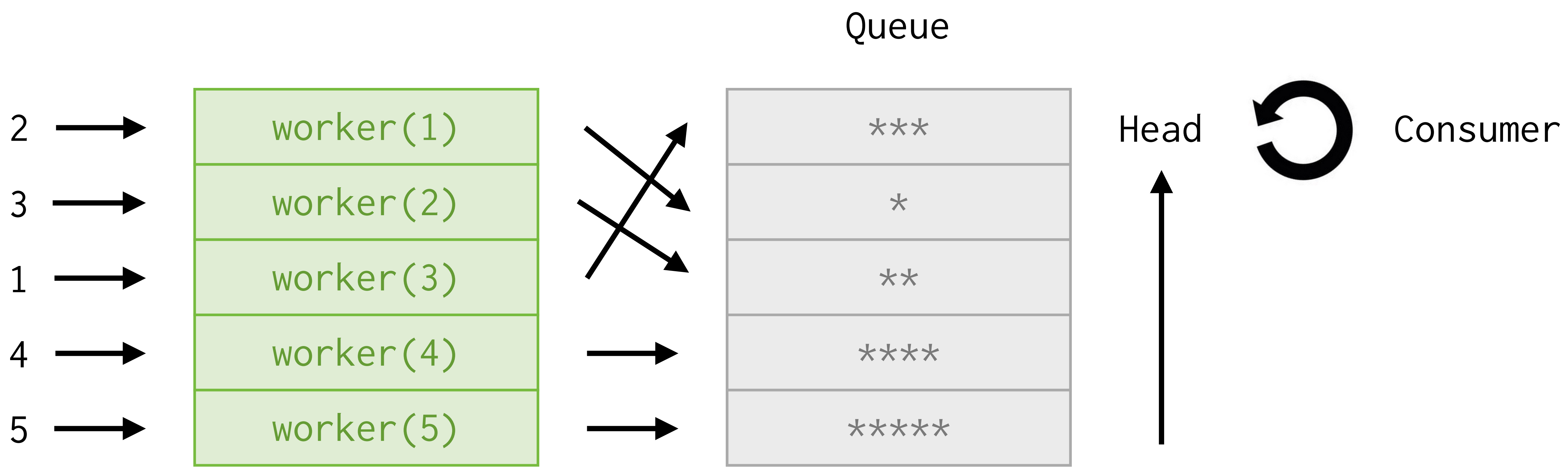
Producers and consumer (2/2)



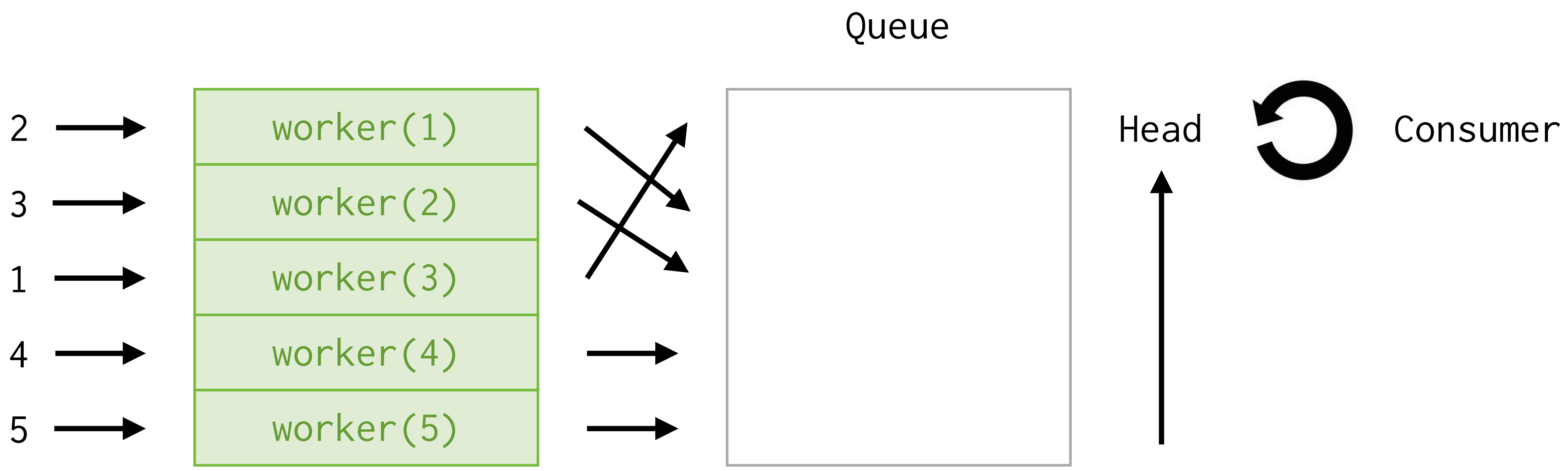
Producers and consumer (2/2)



Producers and consumer (2/2)



Producers and consumer (2/2)



Worker pool (1/2)

```
import random
import gevent
from gevent.queue import Queue
from gevent.pool import Pool

pool = Pool(2)
tasks = Queue()

def worker(multiplier):
    gevent.sleep(random.random())
    tasks.put('*' * multiplier)

def producers():
    pool.map(worker, range(1, 6))

def consumer():
    while True:
        print tasks.get()

gevent.spawn(consumer)
producers()
```

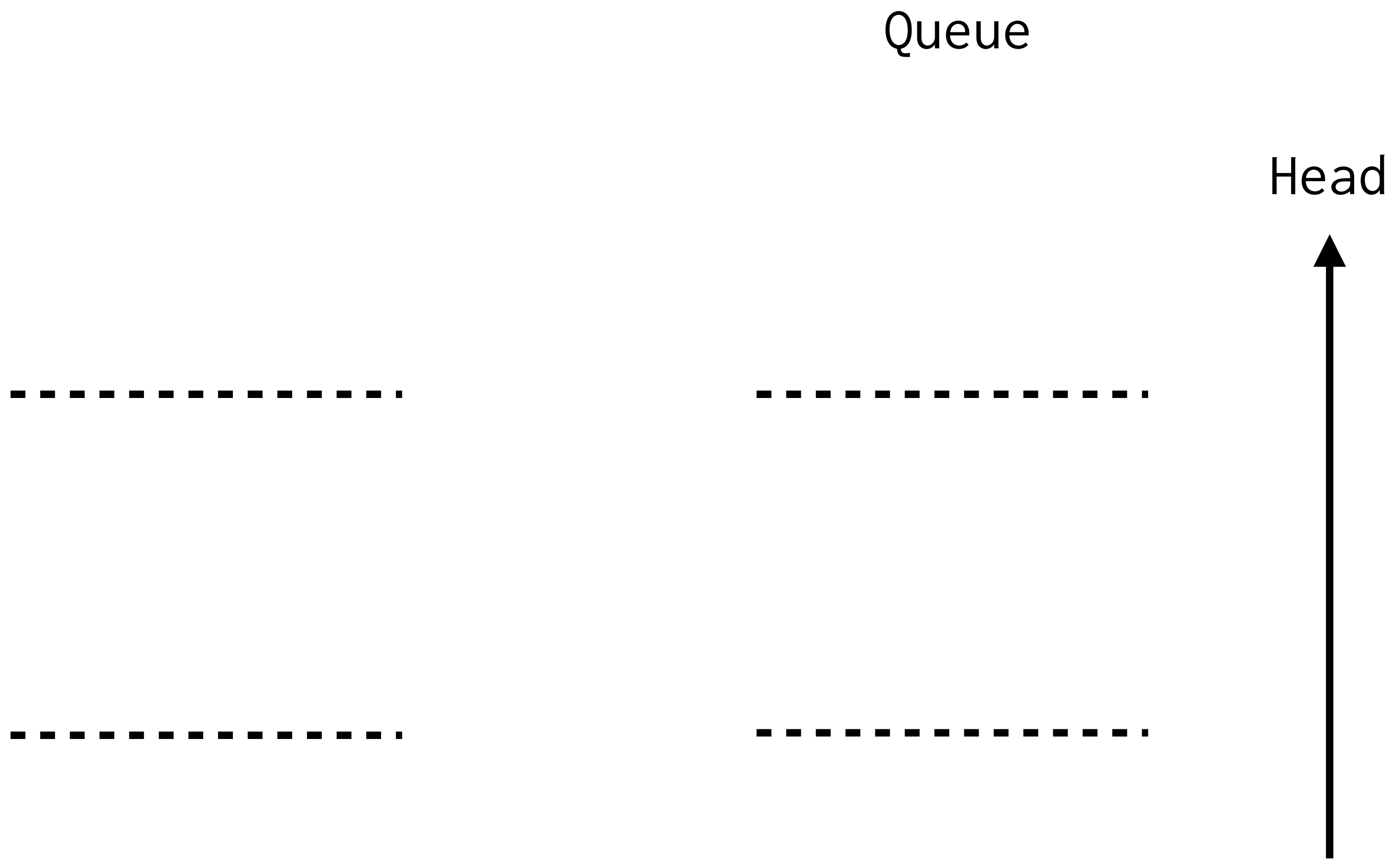
Worker pool (1/2)

```
import random
import gevent
from gevent.queue import Queue
from gevent.pool import Pool

pool = Pool(2)
tasks = Queue()

def worker(multiplier):
    gevent.sleep(random.random())
    tasks.put('*' * multiplier)

def producers():
    pool.map(worker, range(1, 6))

def consumer():
    while True:
        print tasks.get()

gevent.spawn(consumer)
producers()
```

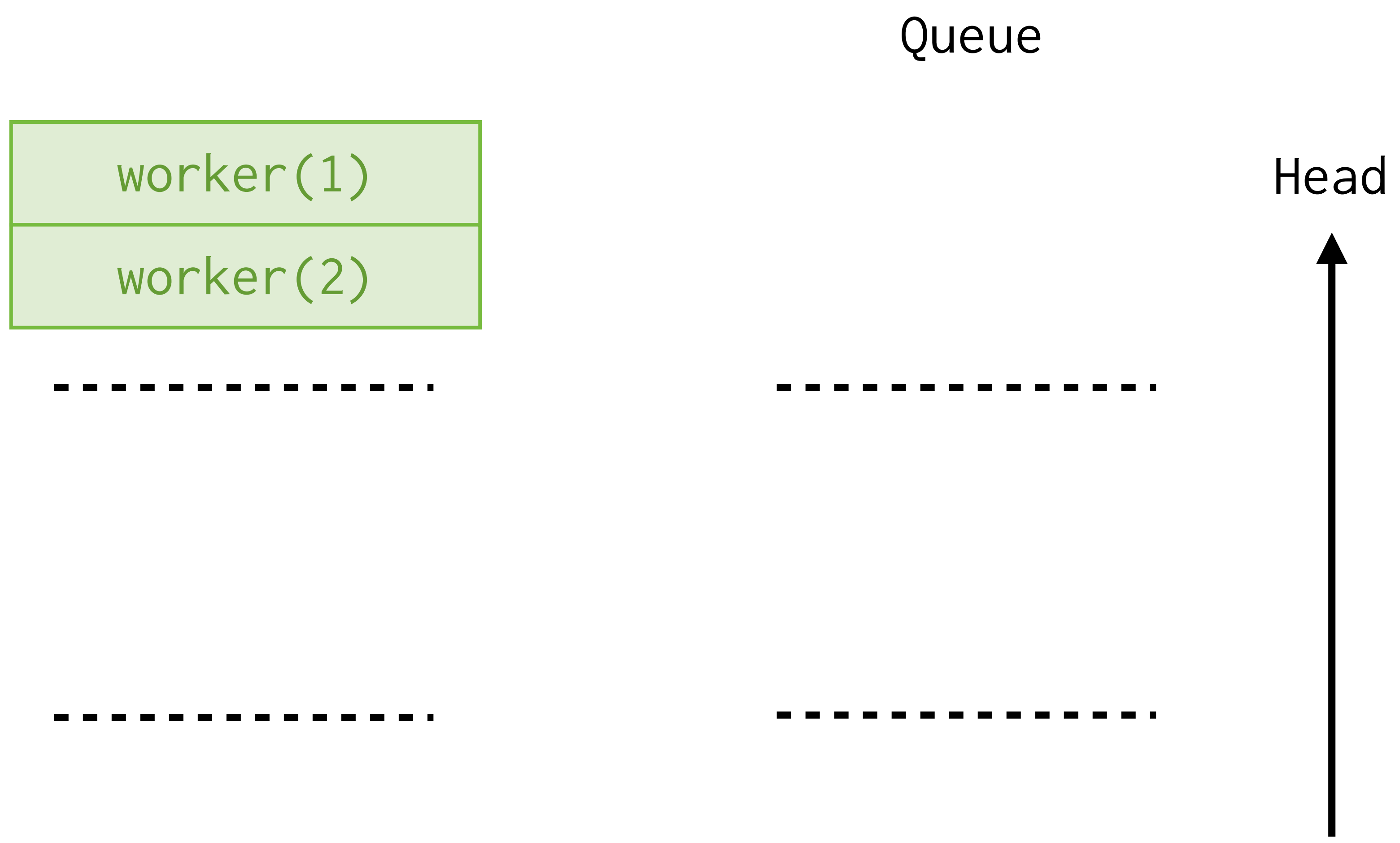
Output:

```
**
*
****
*****
***
```

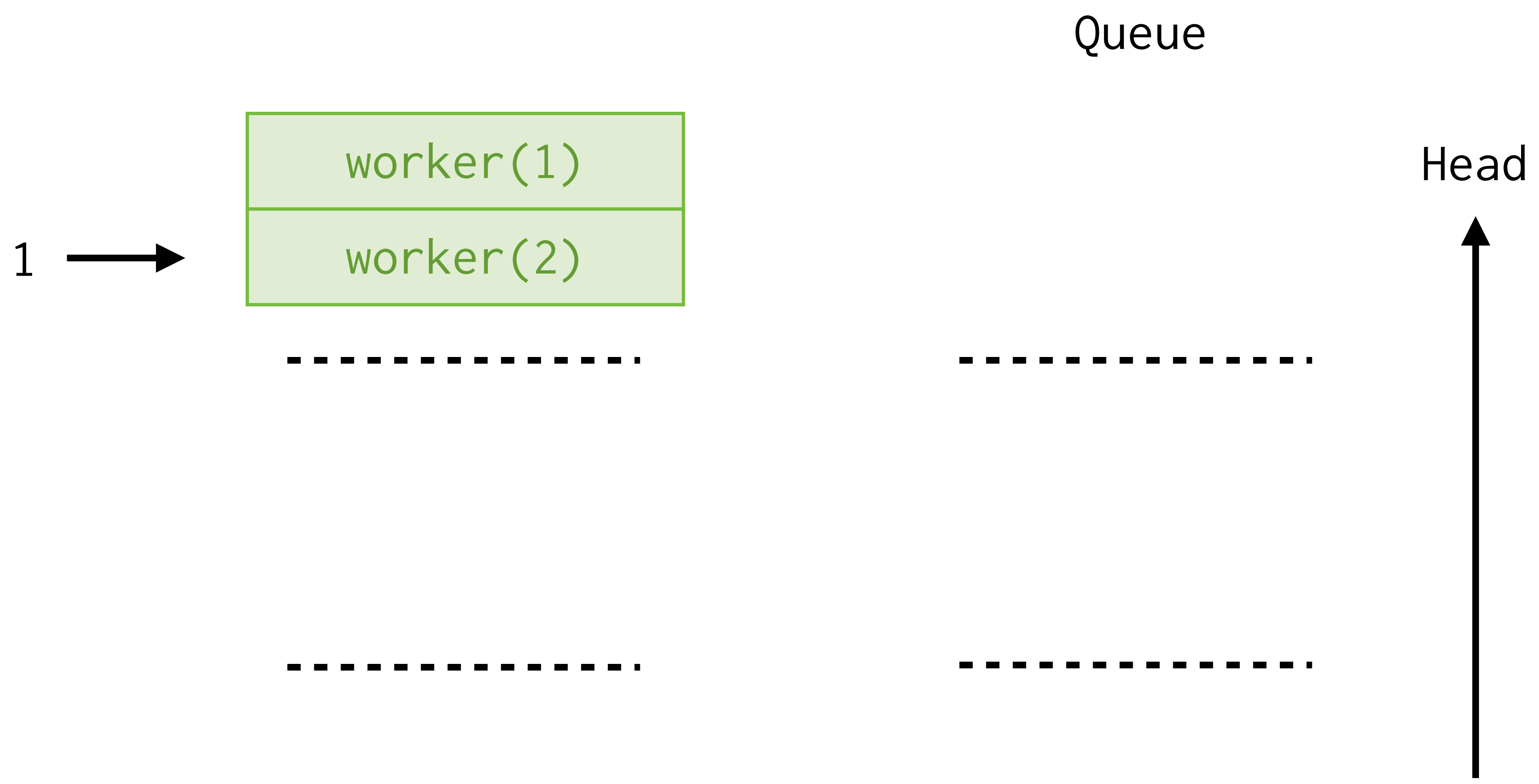
Worker pool (2/2)



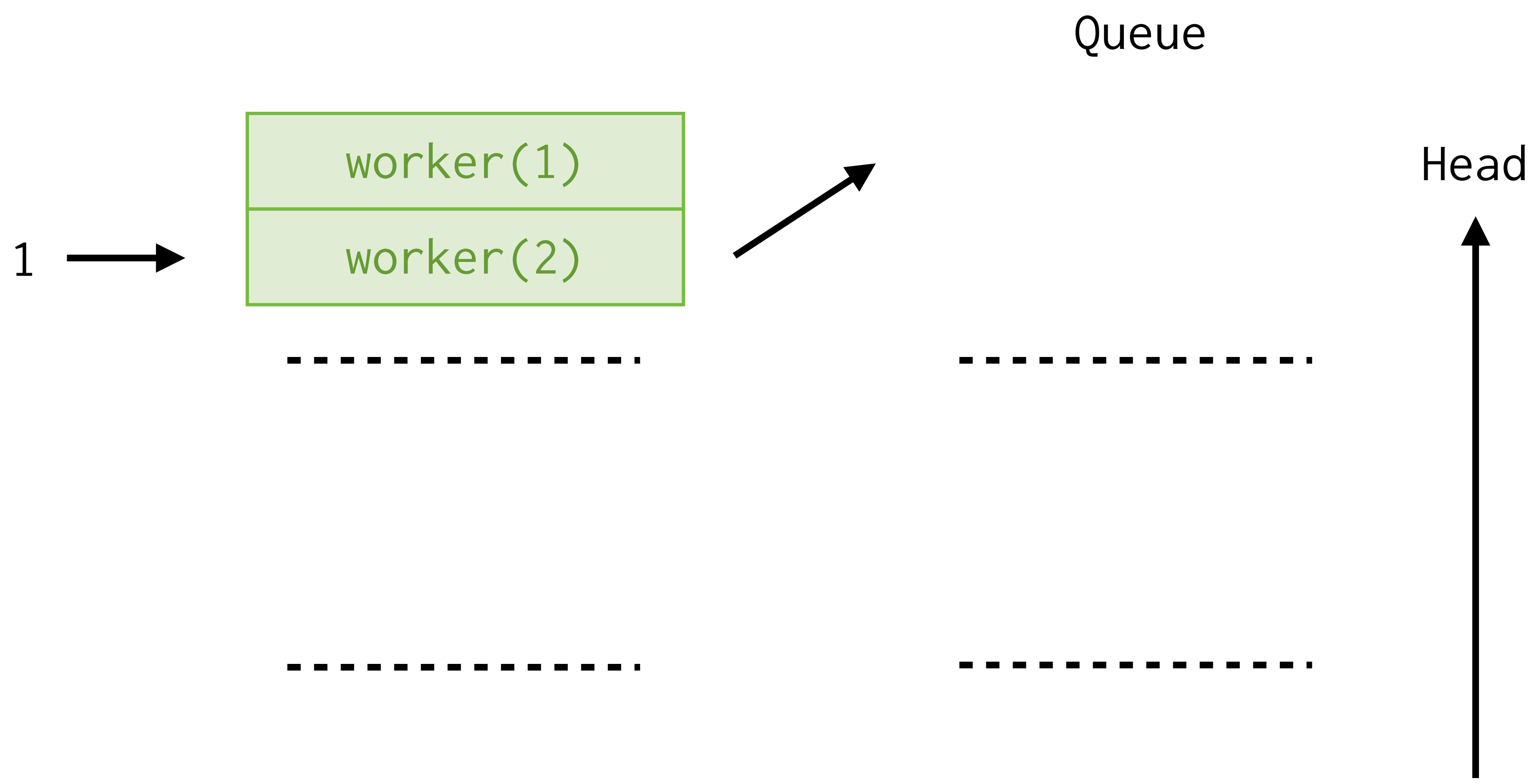
Worker pool (2/2)



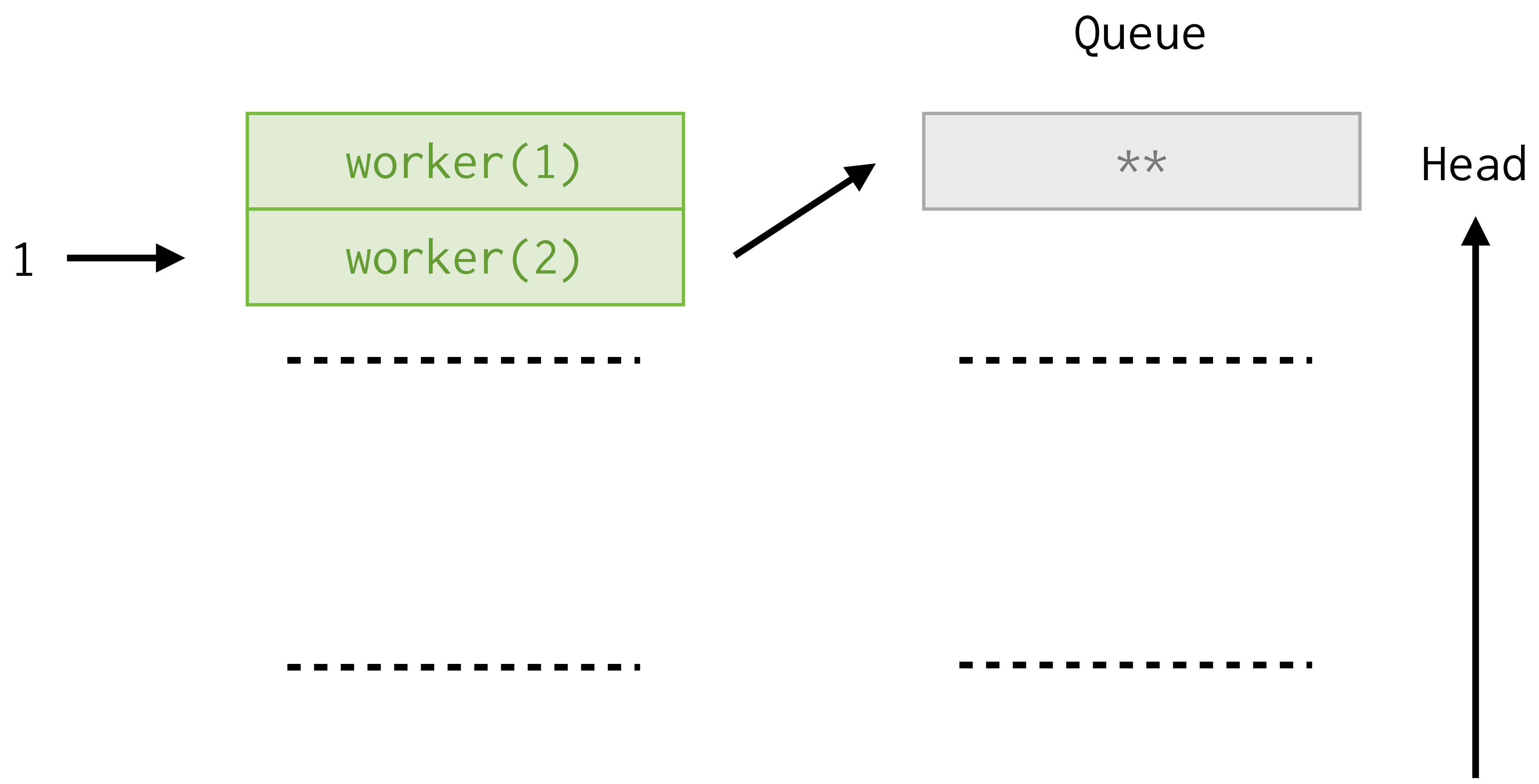
Worker pool (2/2)



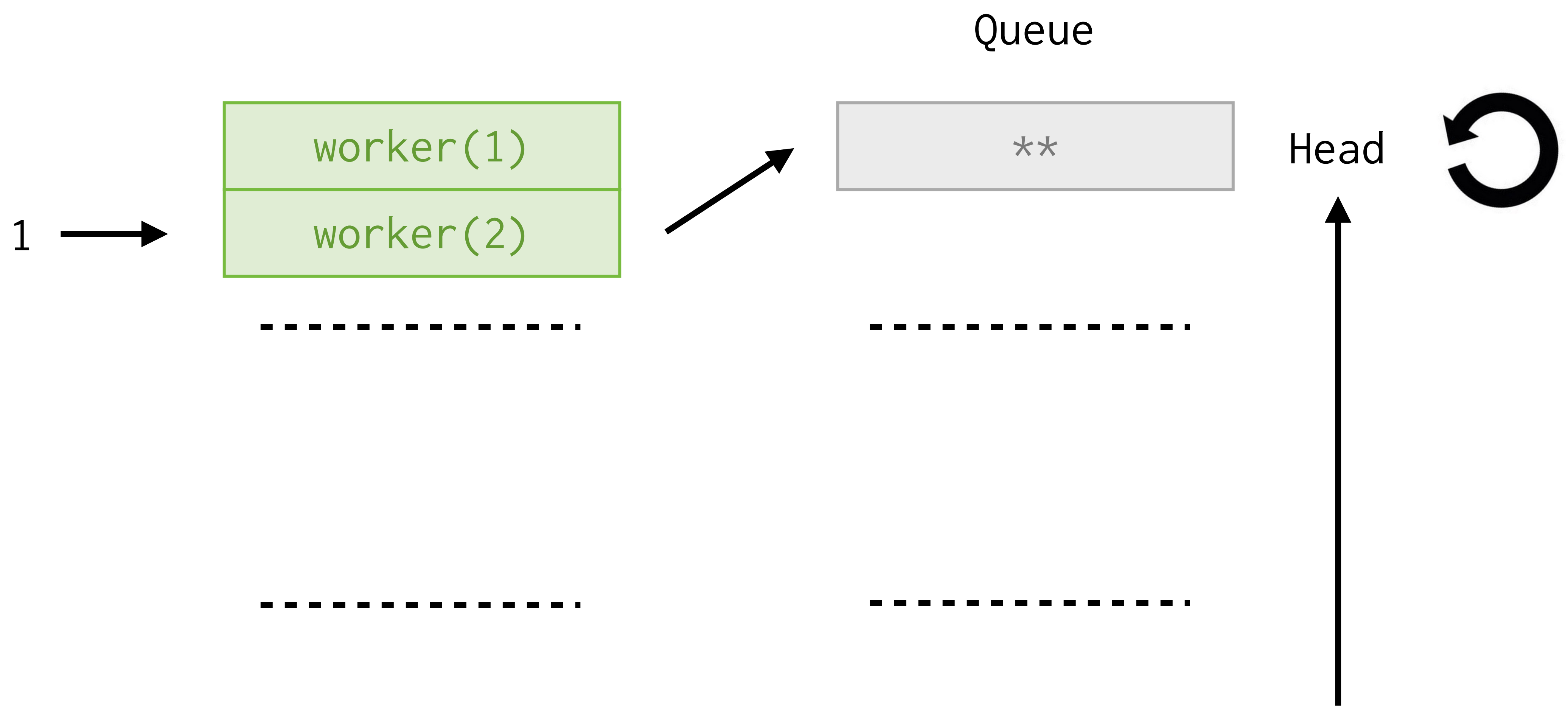
Worker pool (2/2)



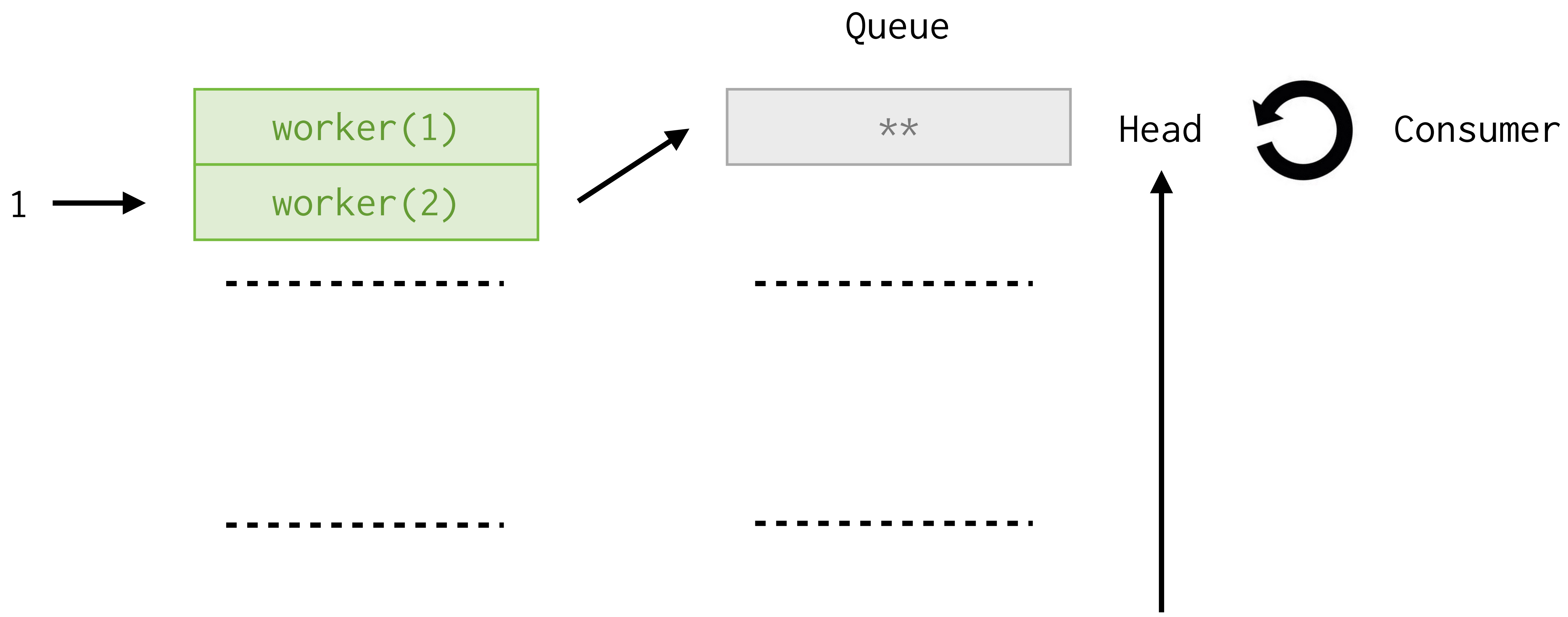
Worker pool (2/2)



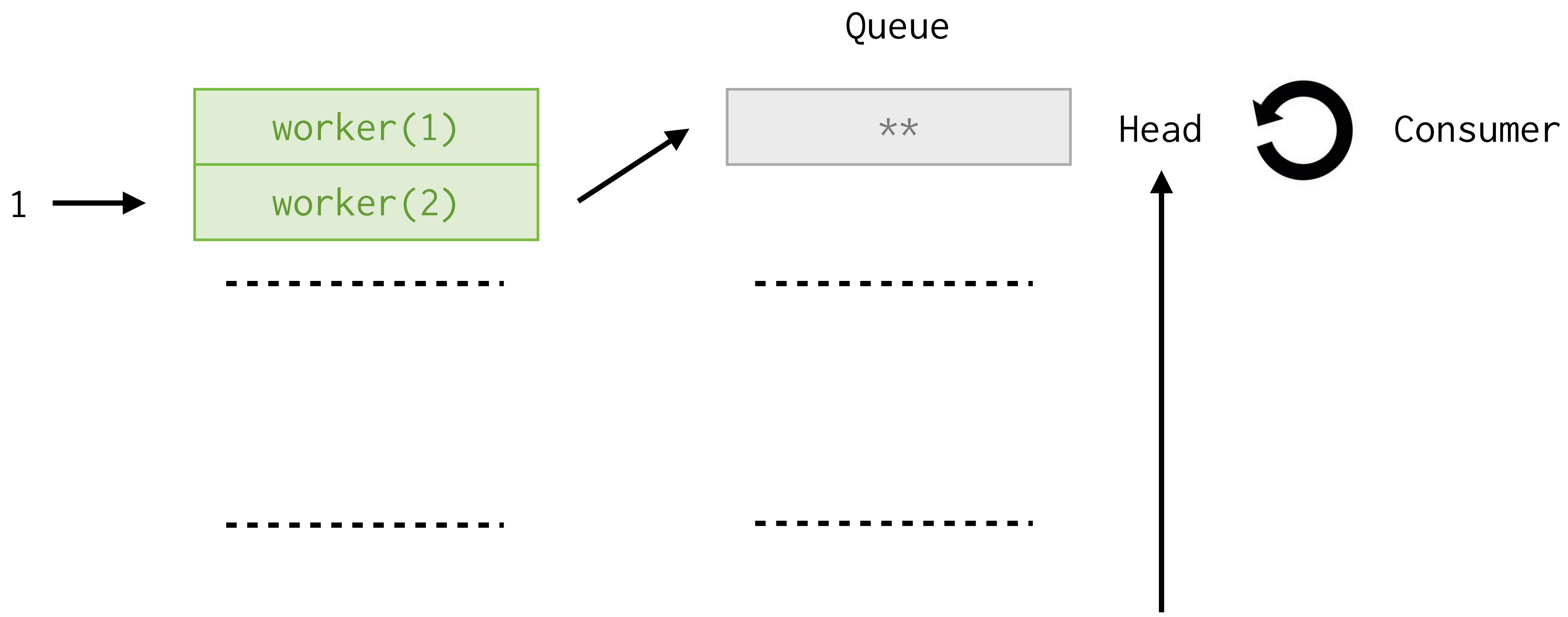
Worker pool (2/2)



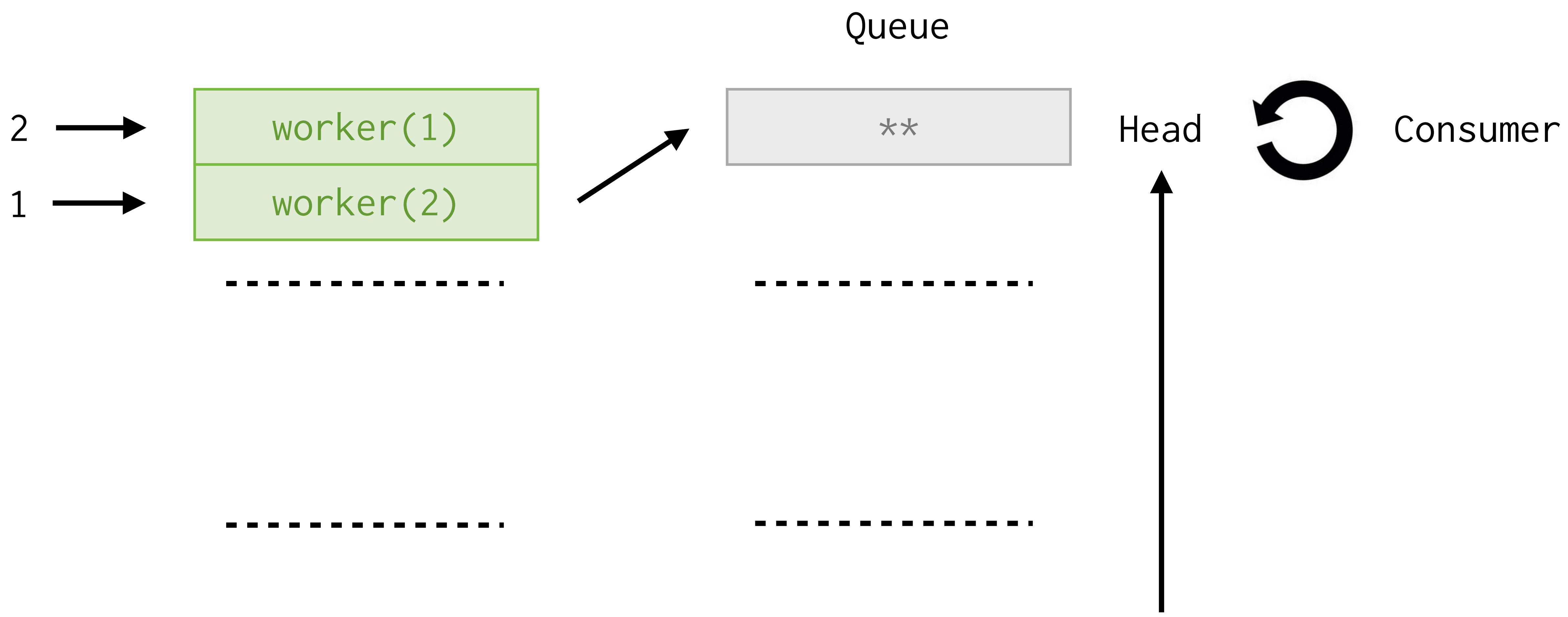
Worker pool (2/2)



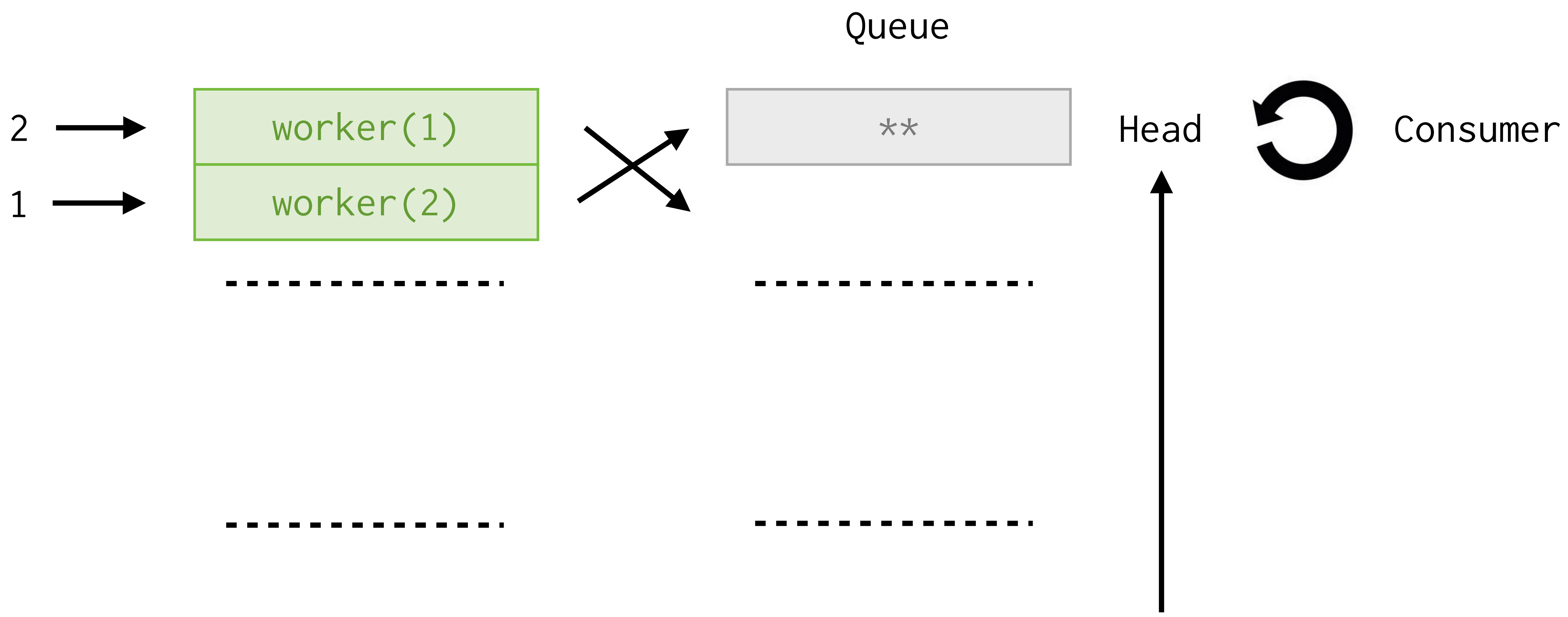
Worker pool (2/2)



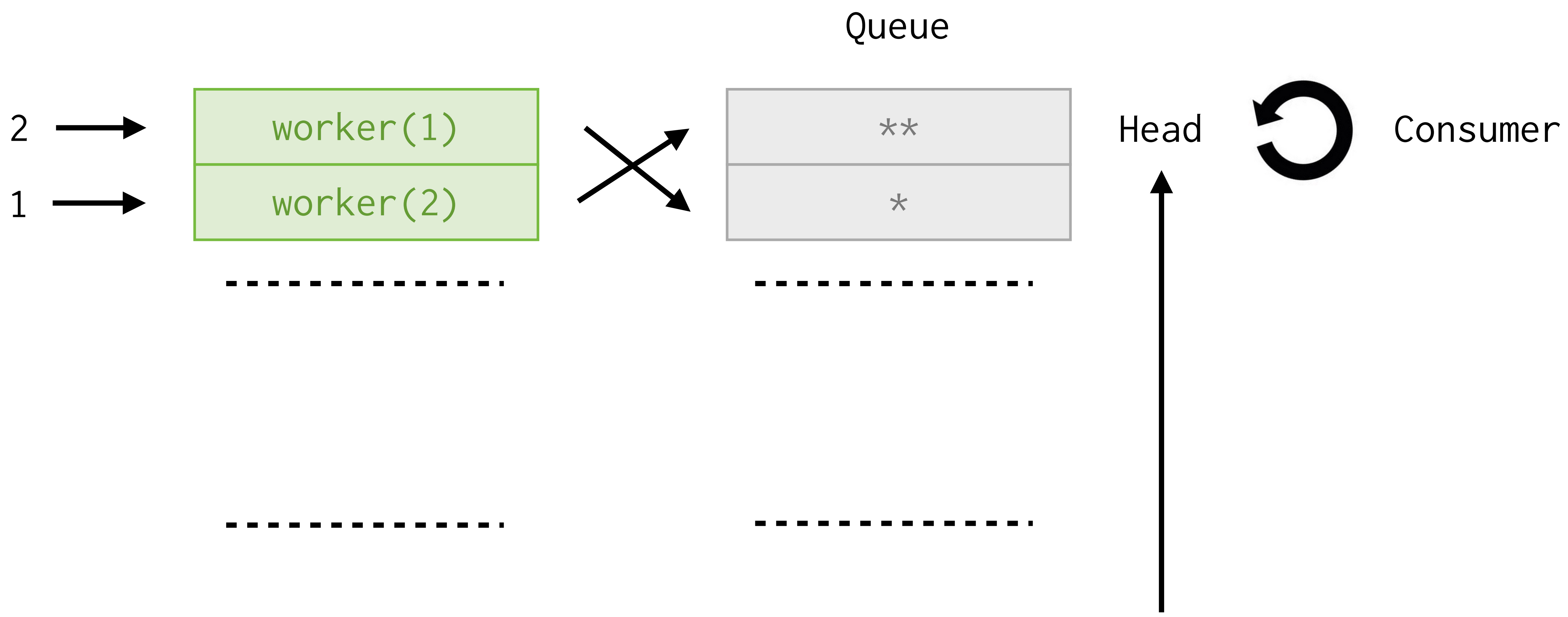
Worker pool (2/2)



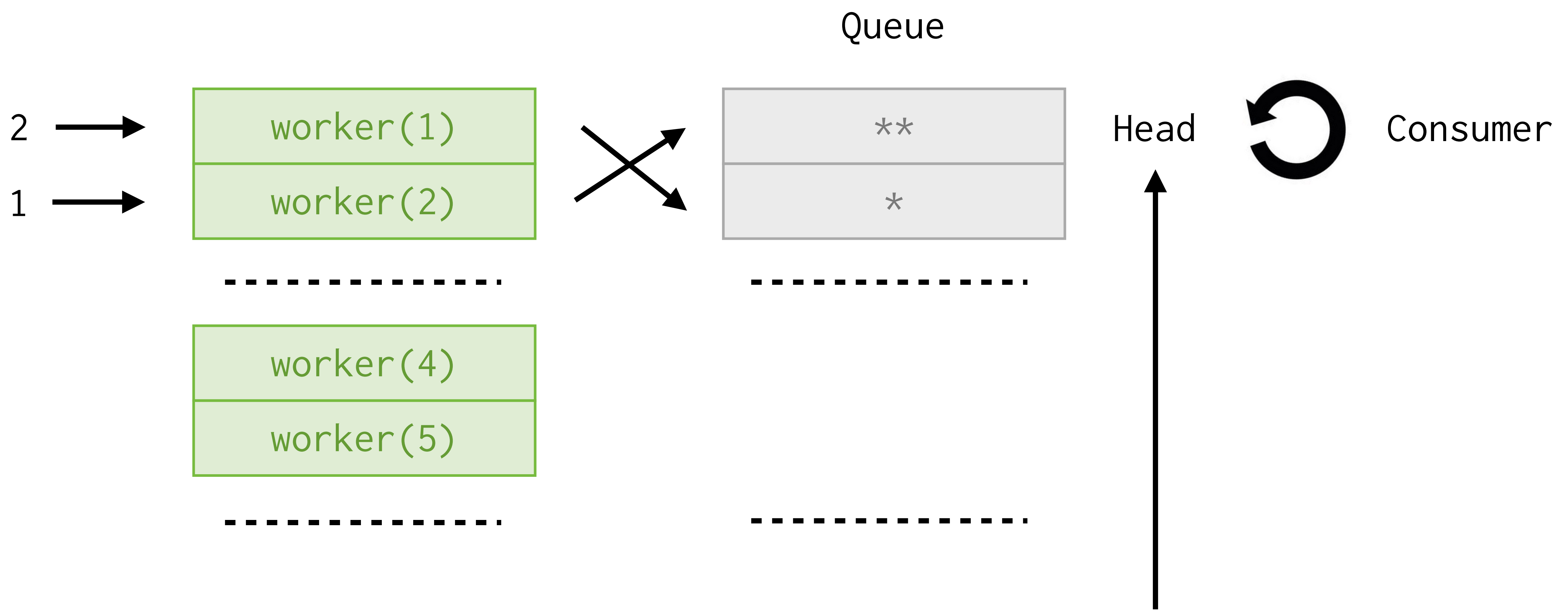
Worker pool (2/2)



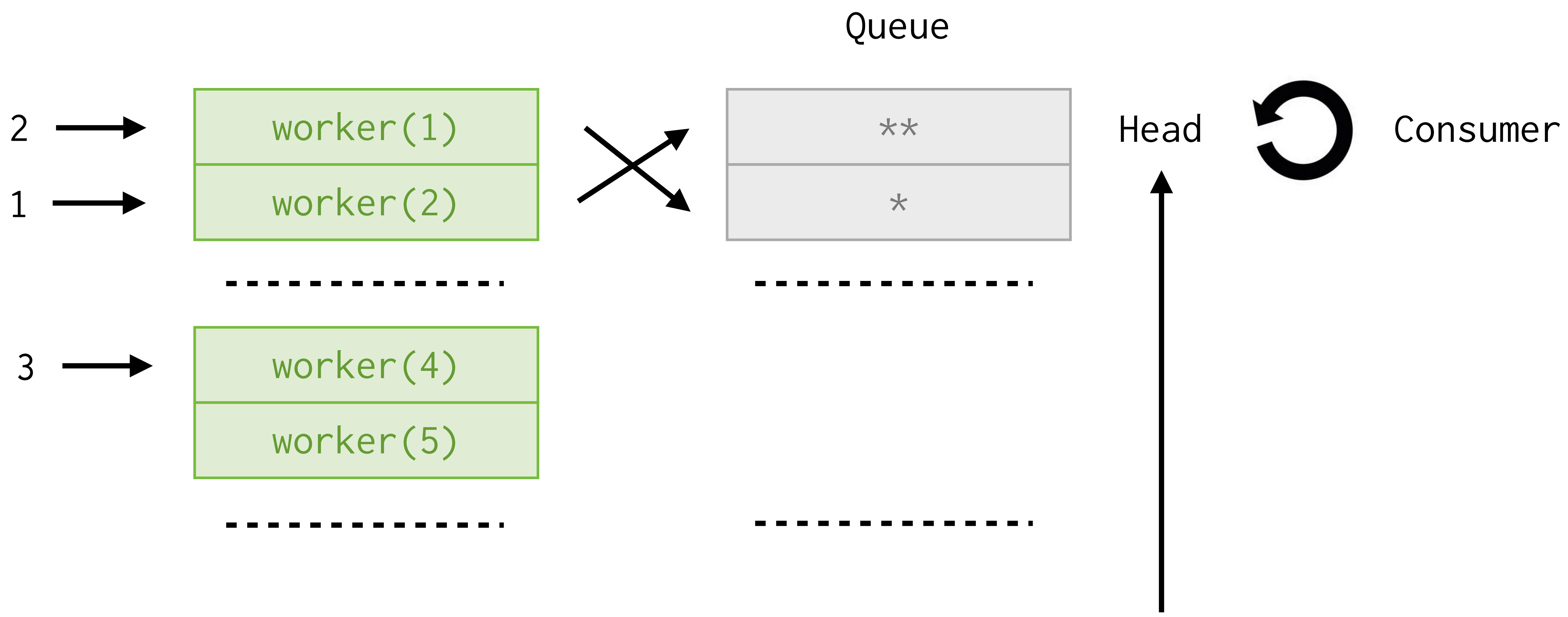
Worker pool (2/2)



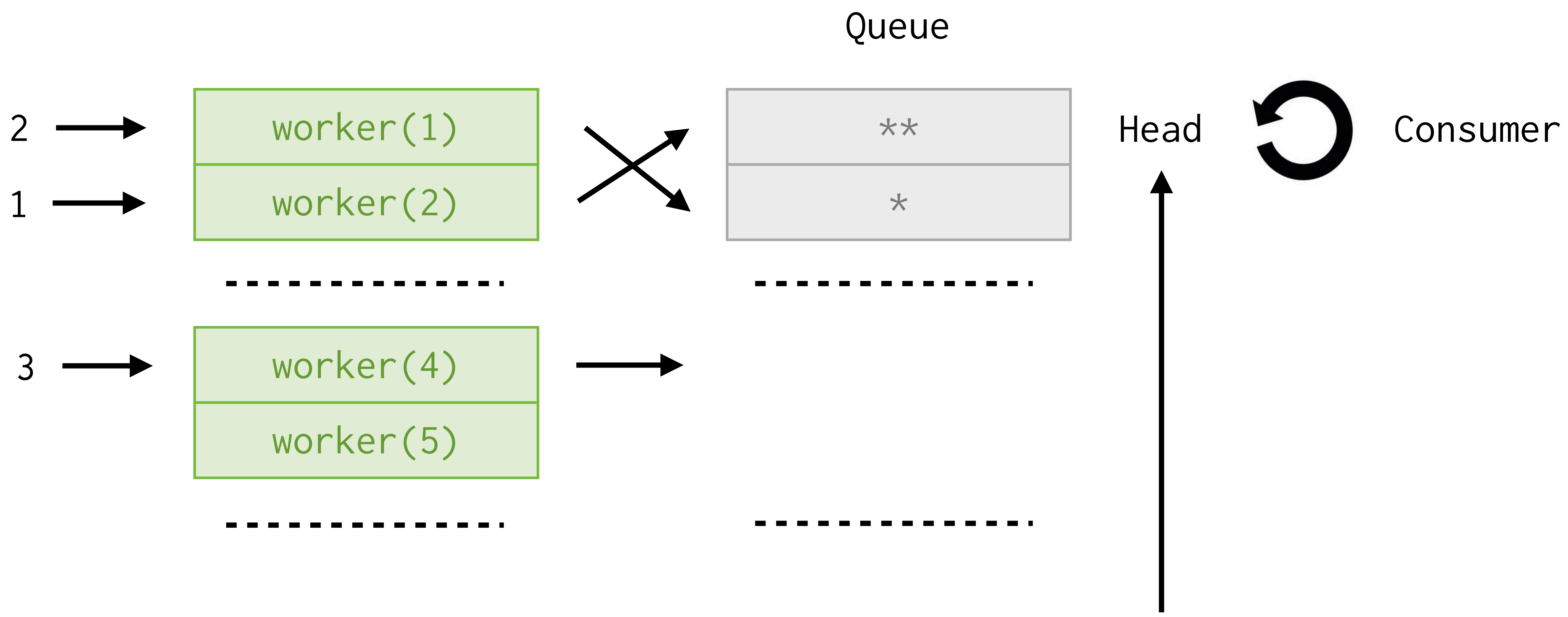
Worker pool (2/2)



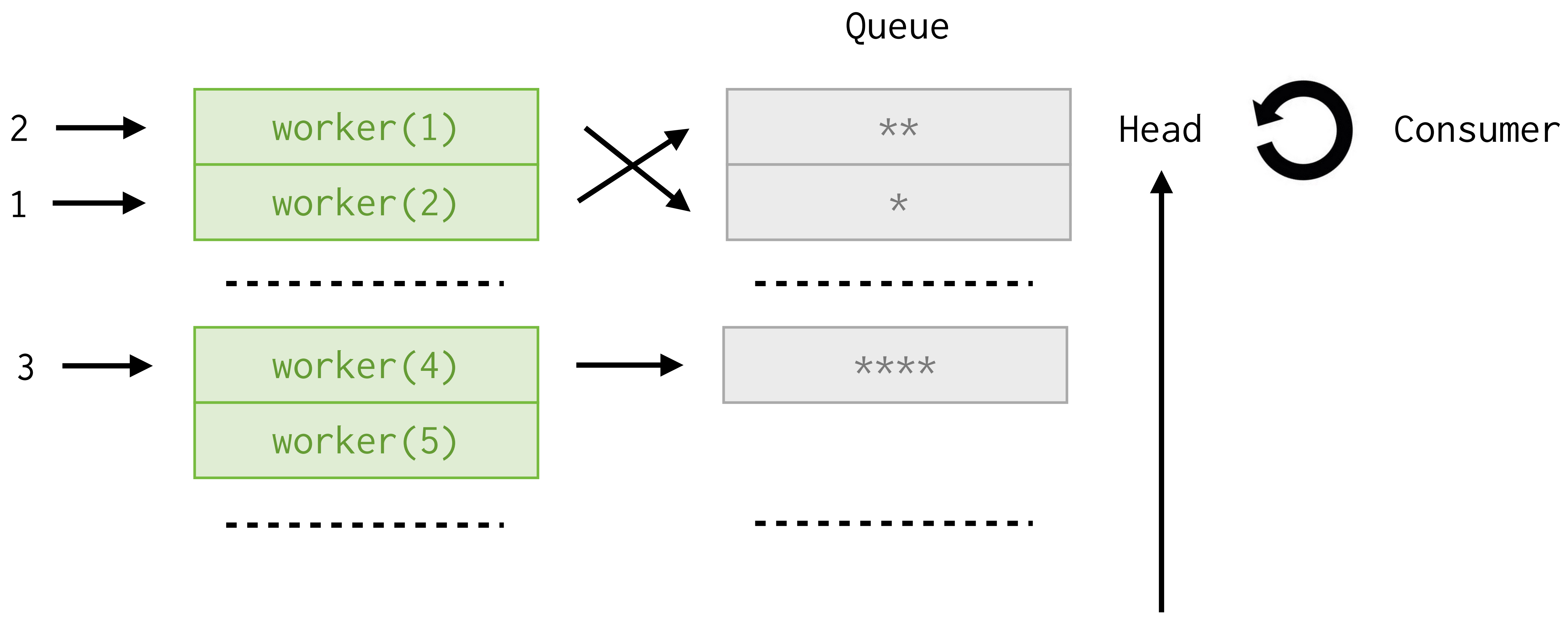
Worker pool (2/2)



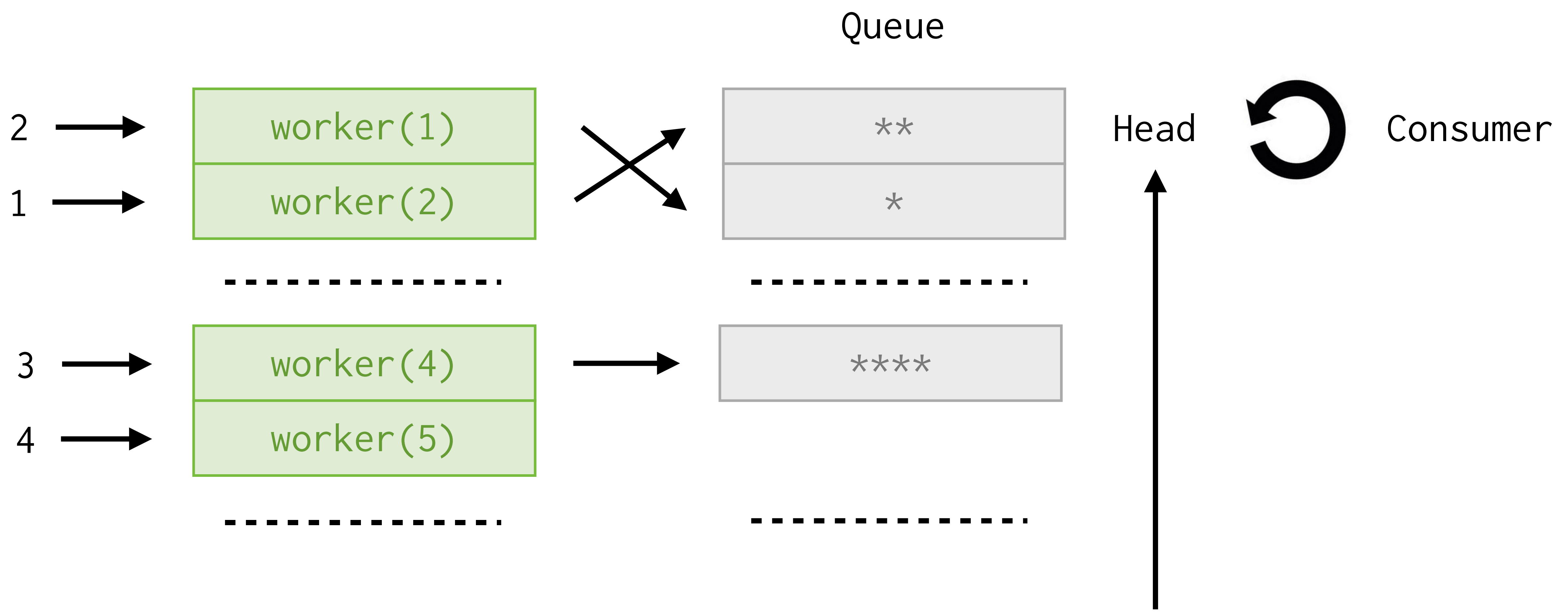
Worker pool (2/2)



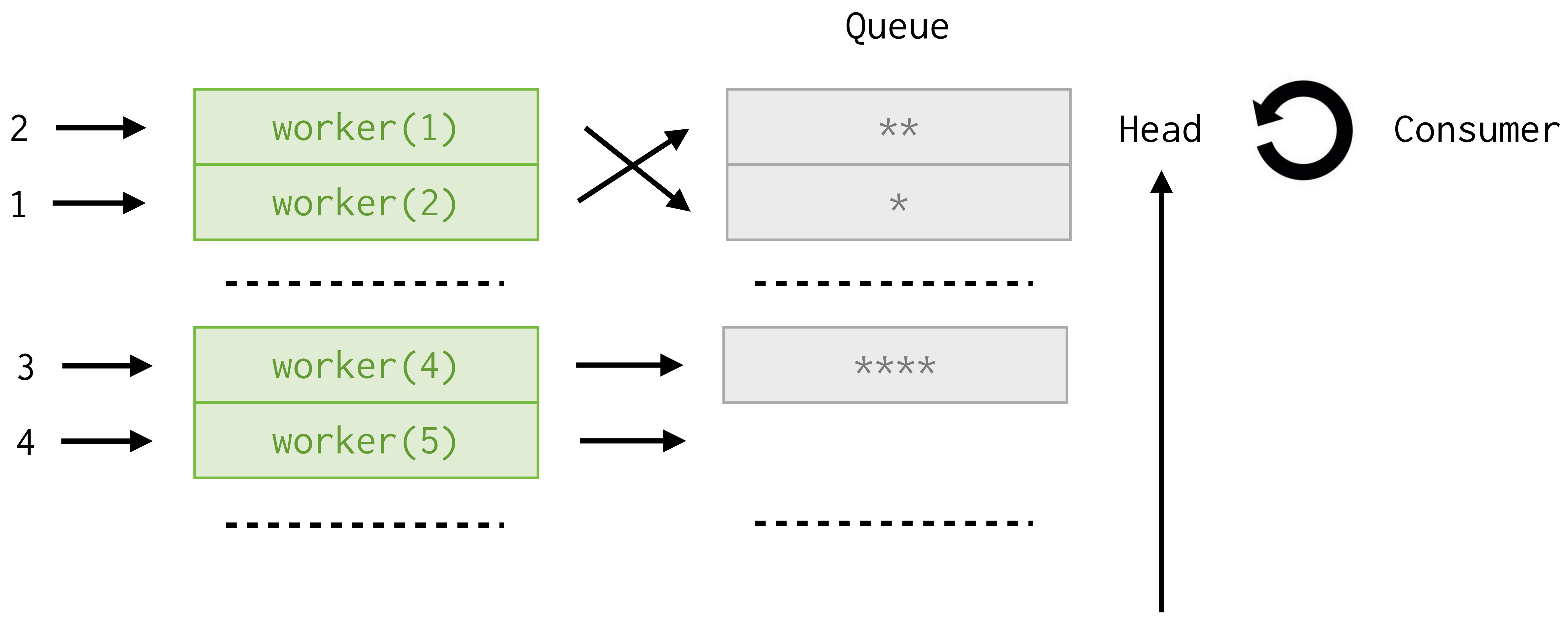
Worker pool (2/2)



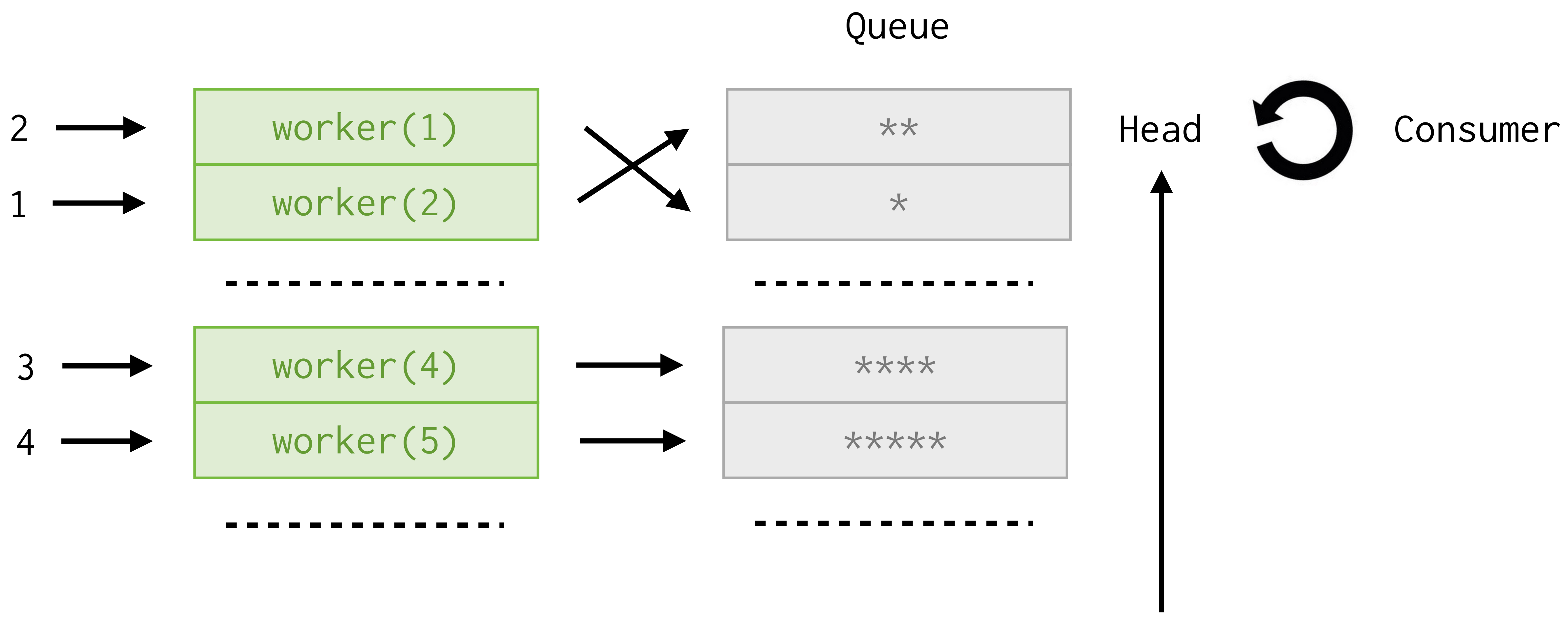
Worker pool (2/2)



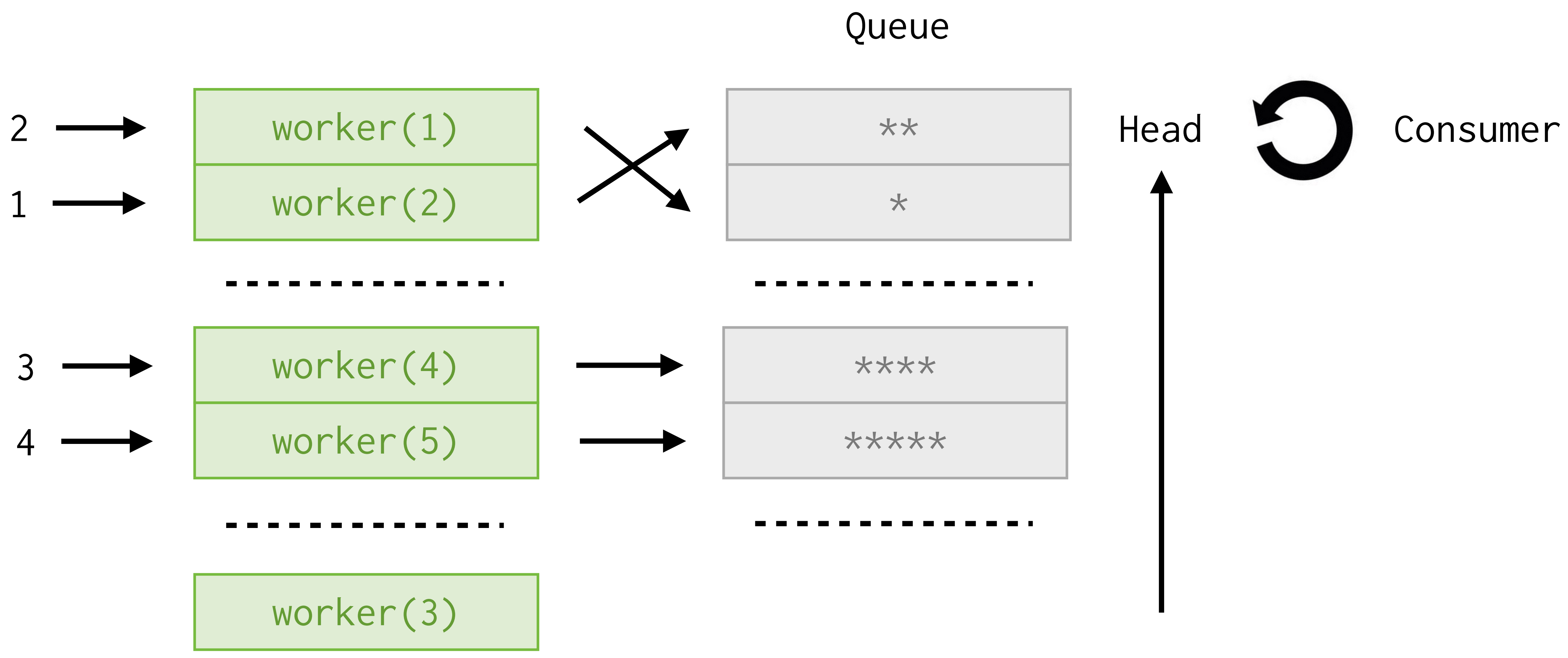
Worker pool (2/2)



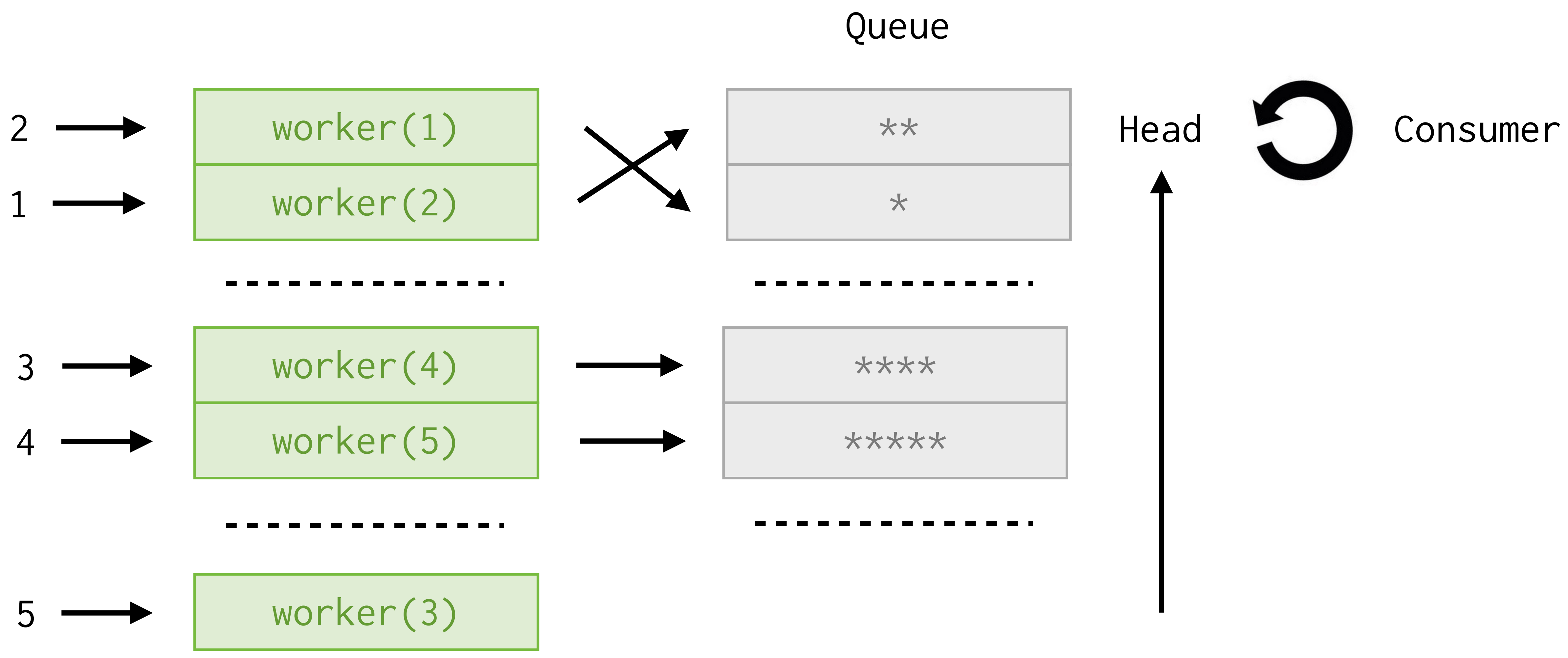
Worker pool (2/2)



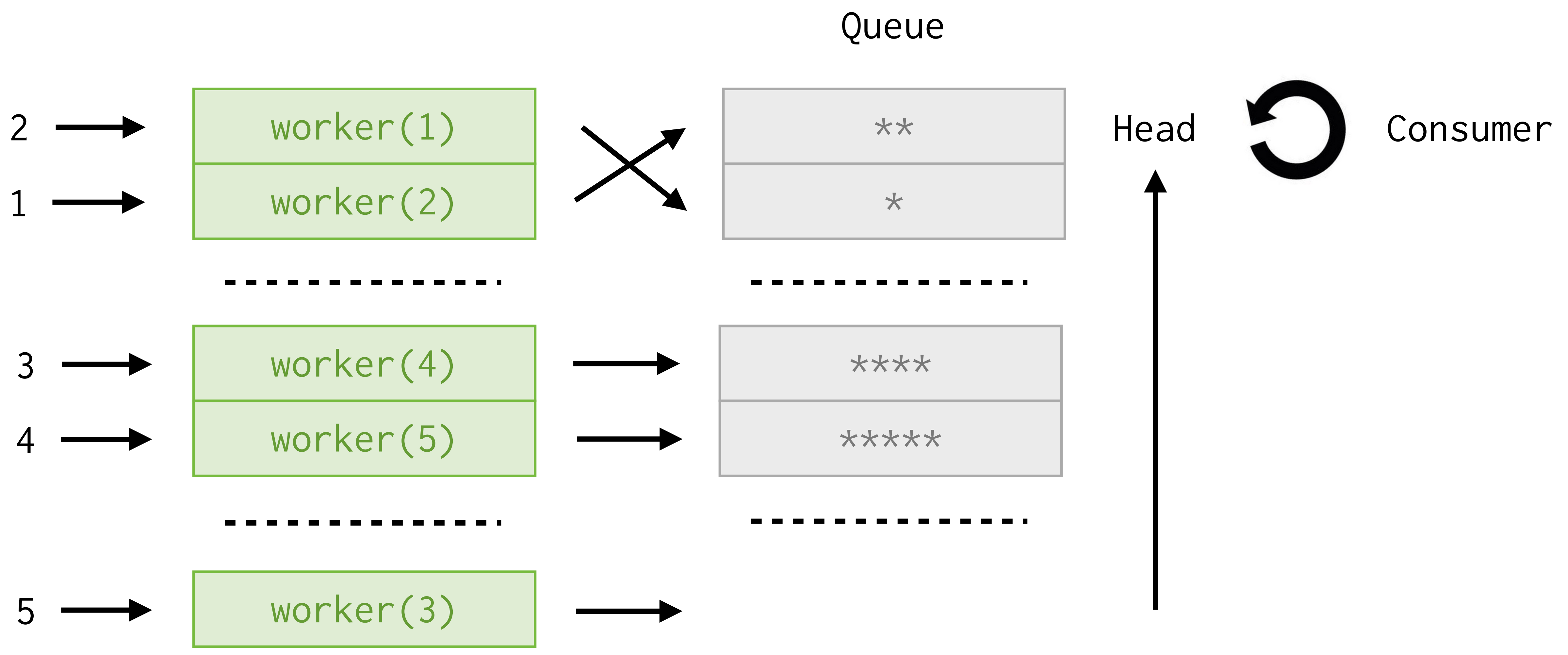
Worker pool (2/2)



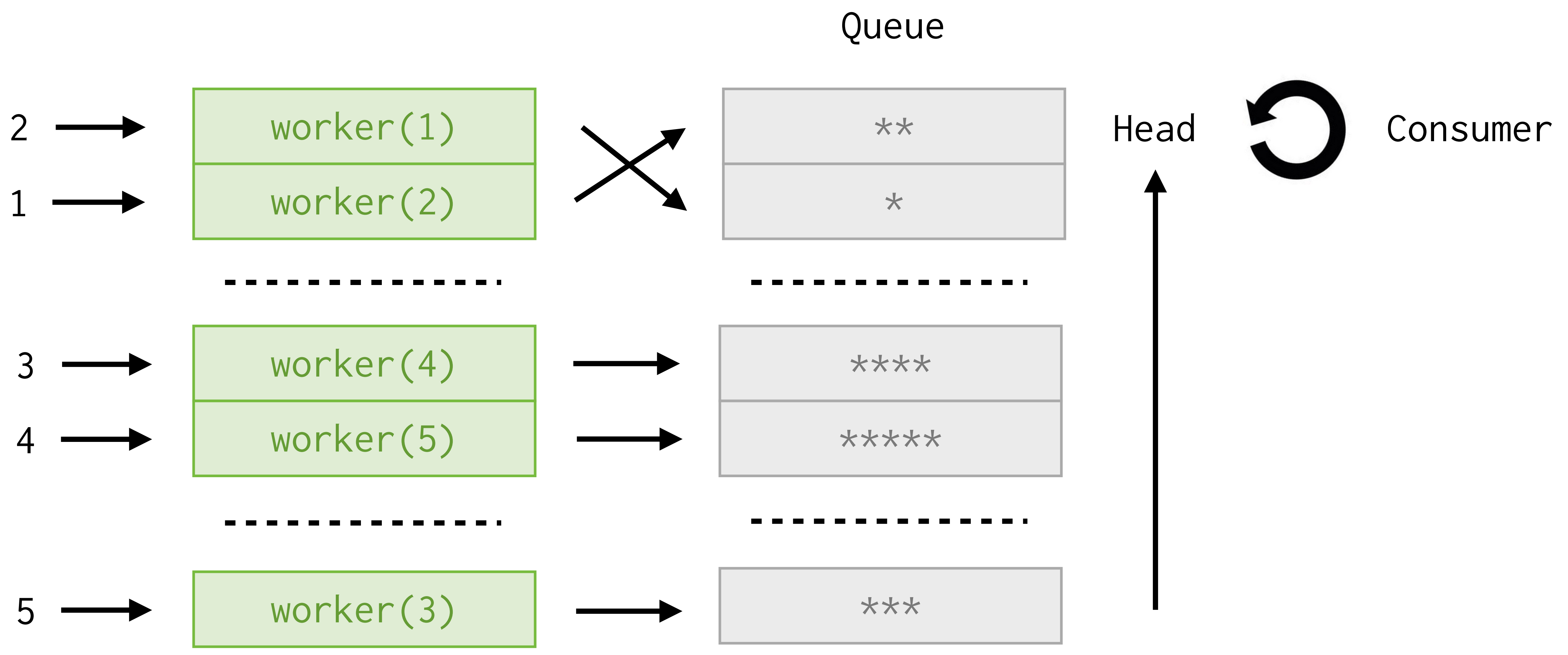
Worker pool (2/2)



Worker pool (2/2)



Worker pool (2/2)



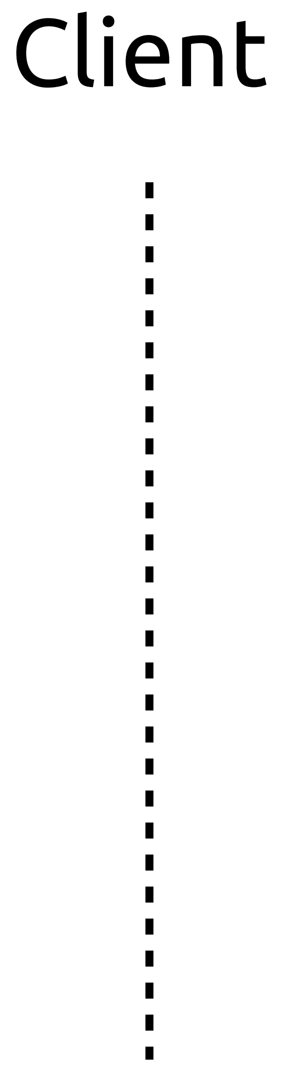
But, but, what about real examples?

Shorty: URL shortener

- TCP
- Message framing

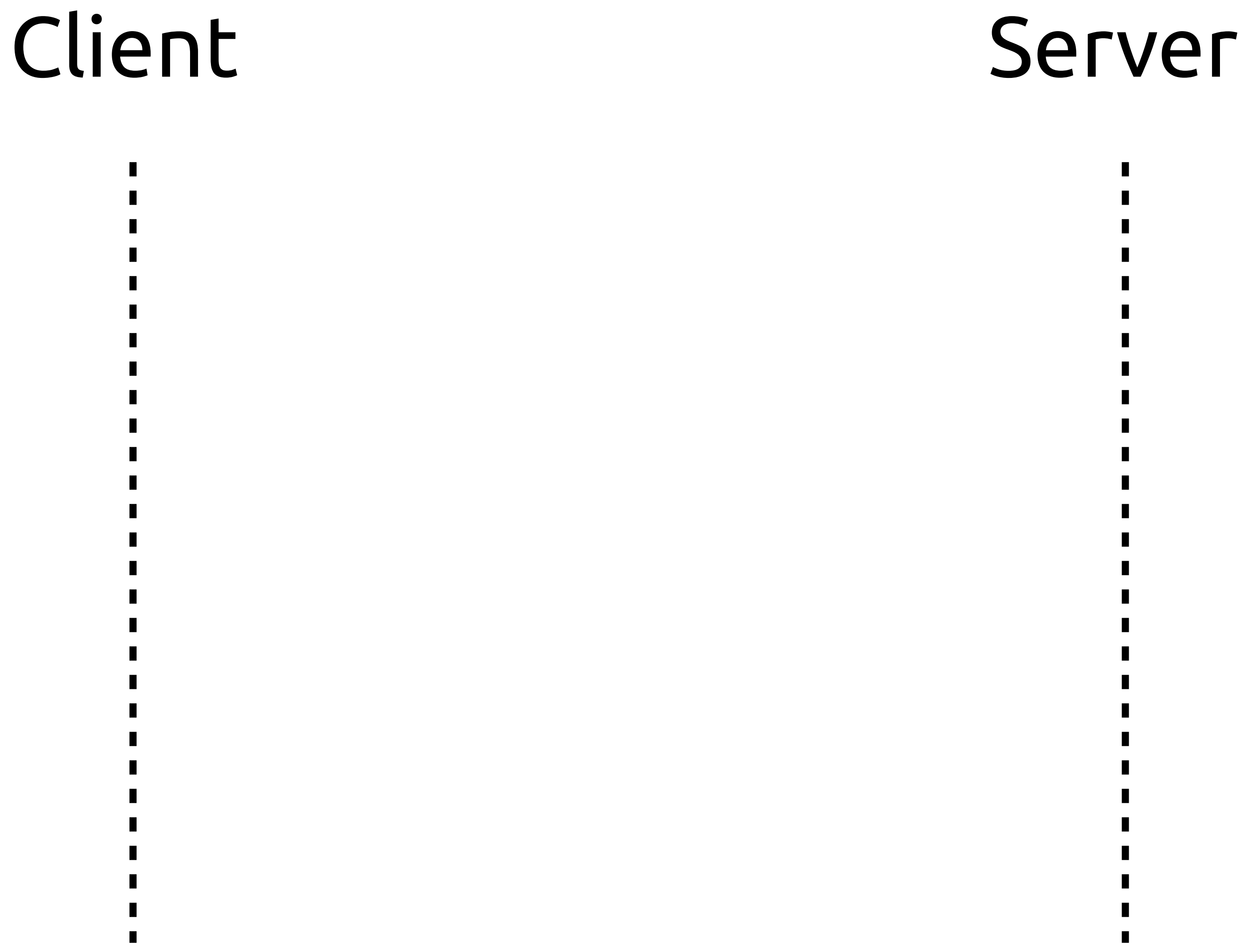
Shorty: URL shortener

- TCP
- Message framing



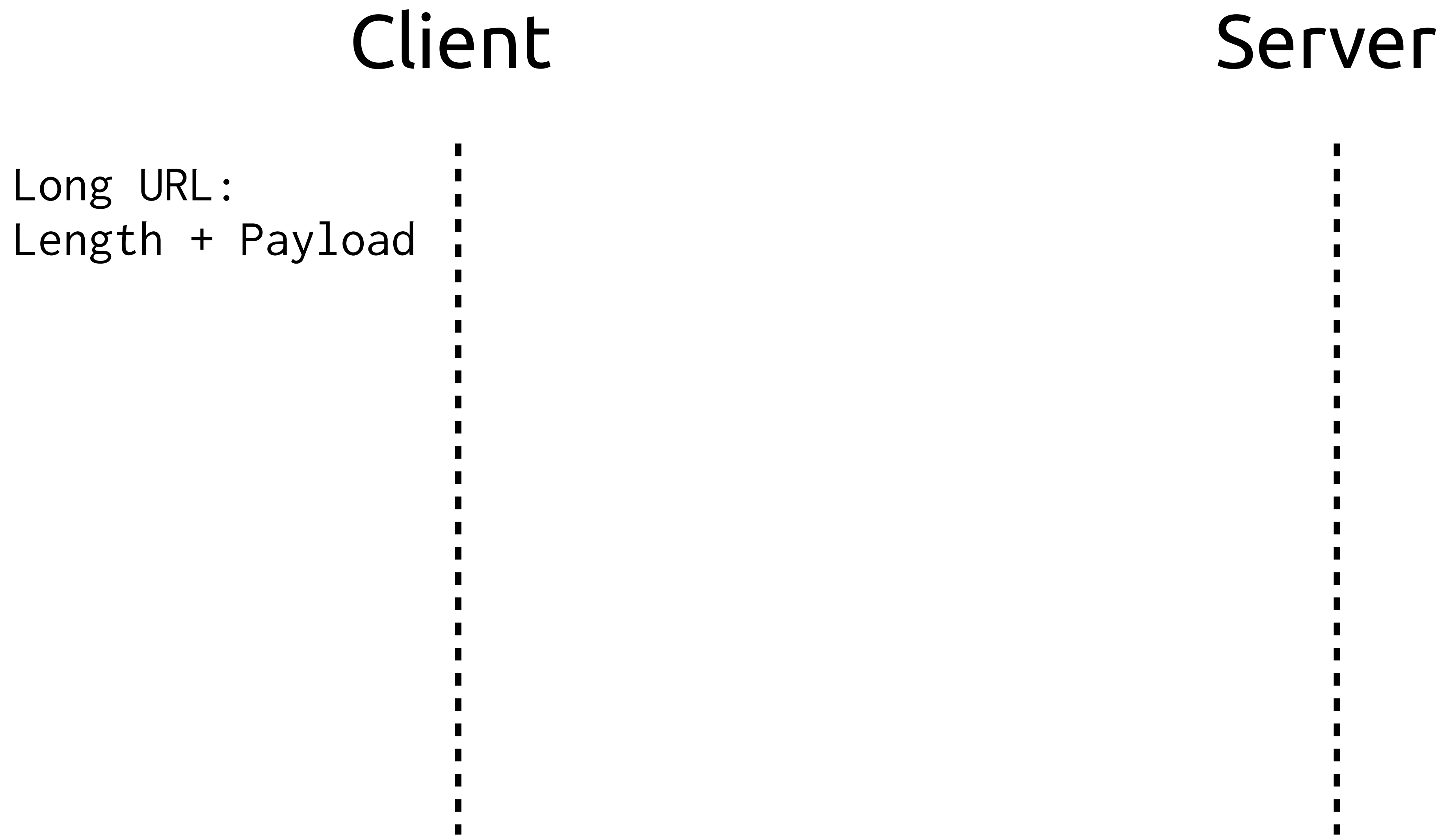
Shorty: URL shortener

- TCP
- Message framing



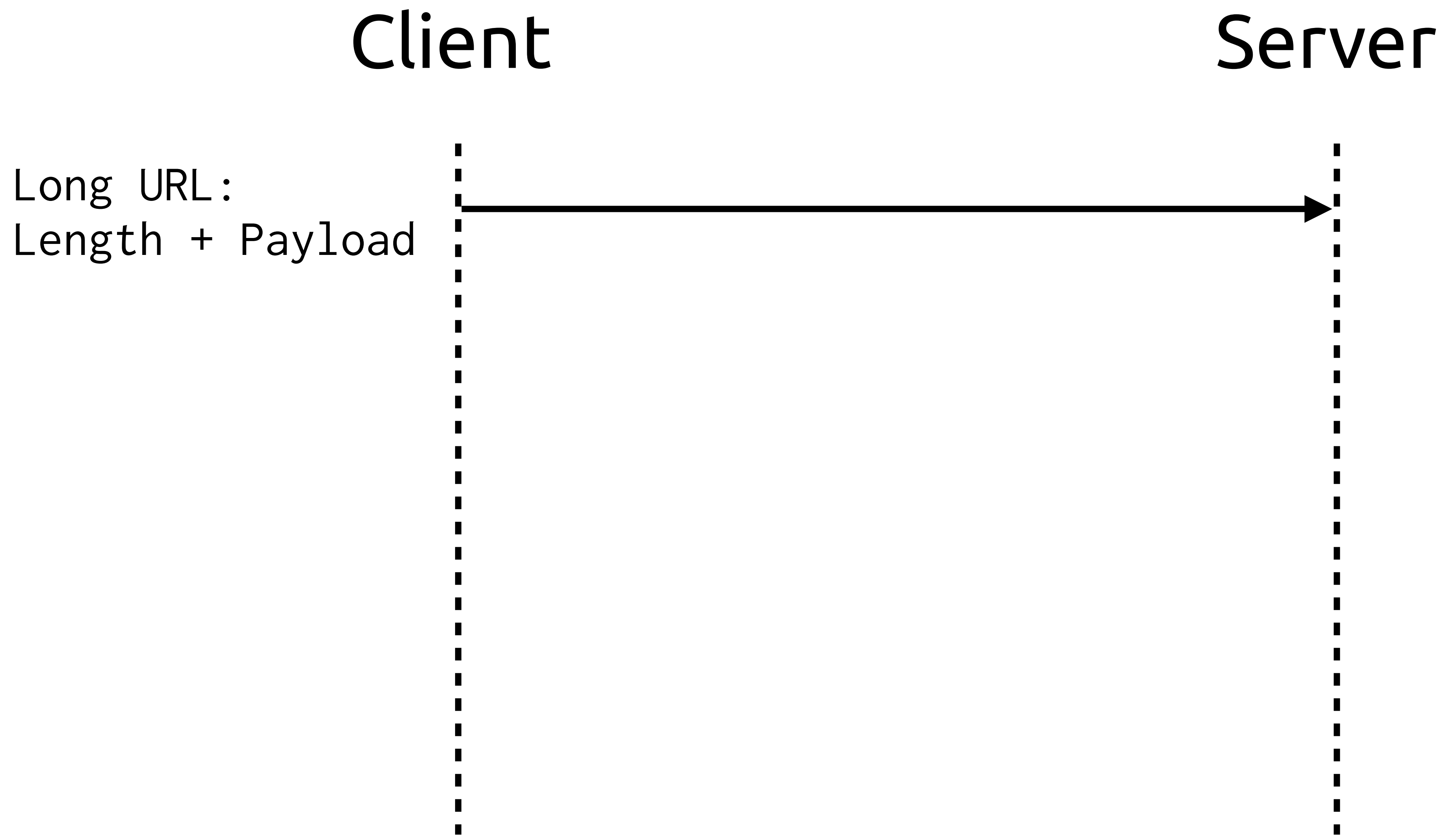
Shorty: URL shortener

- TCP
- Message framing



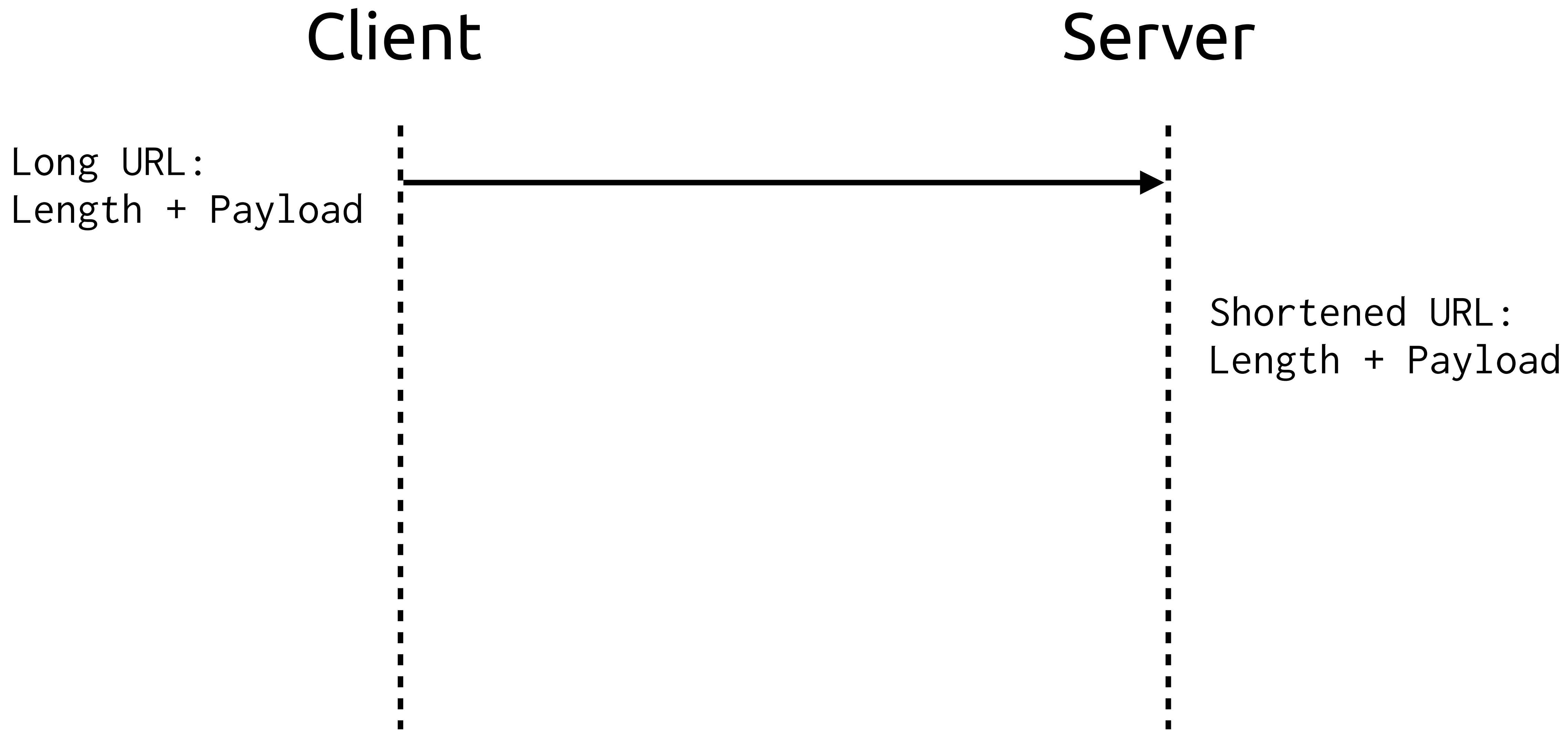
Shorty: URL shortener

- TCP
- Message framing



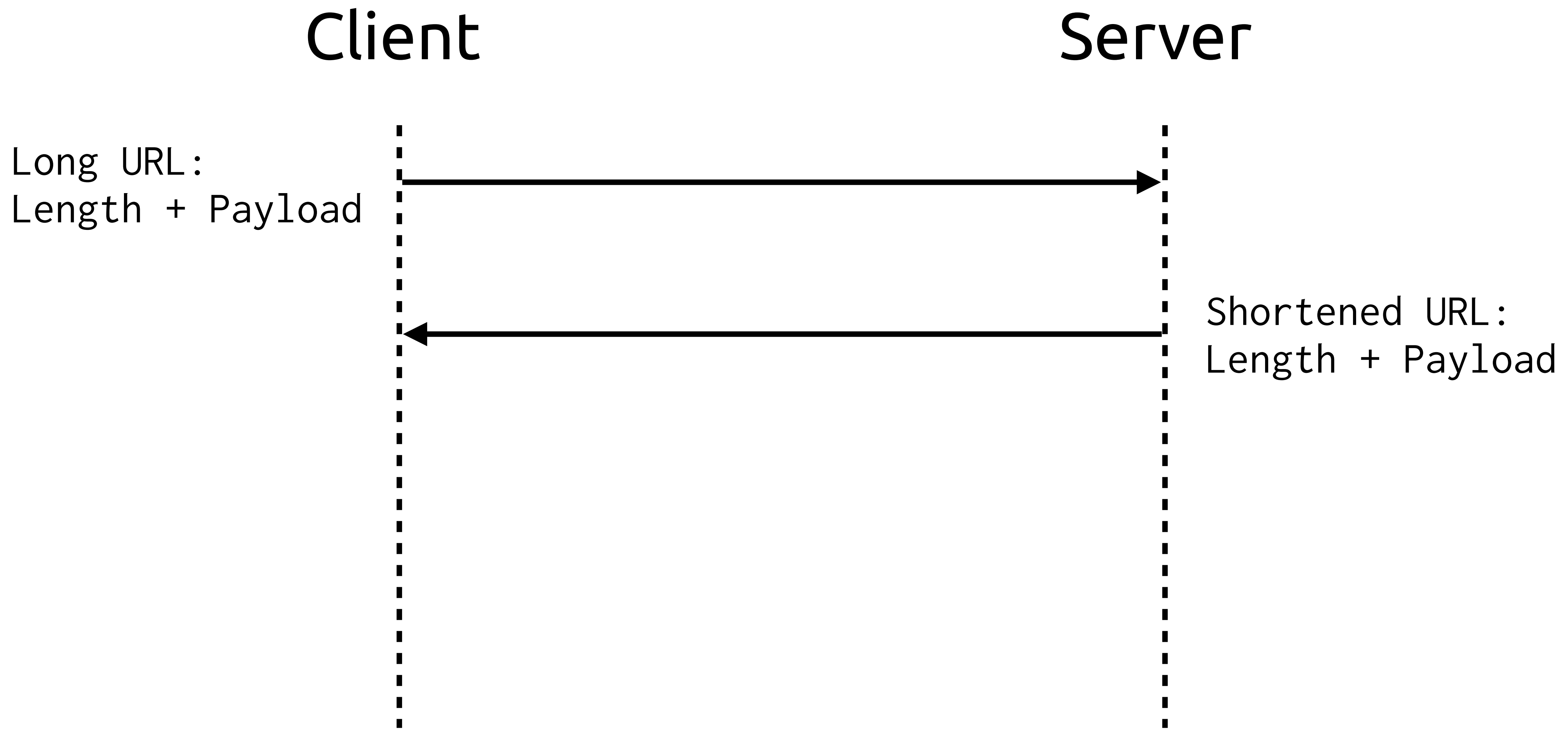
Shorty: URL shortener

- TCP
- Message framing



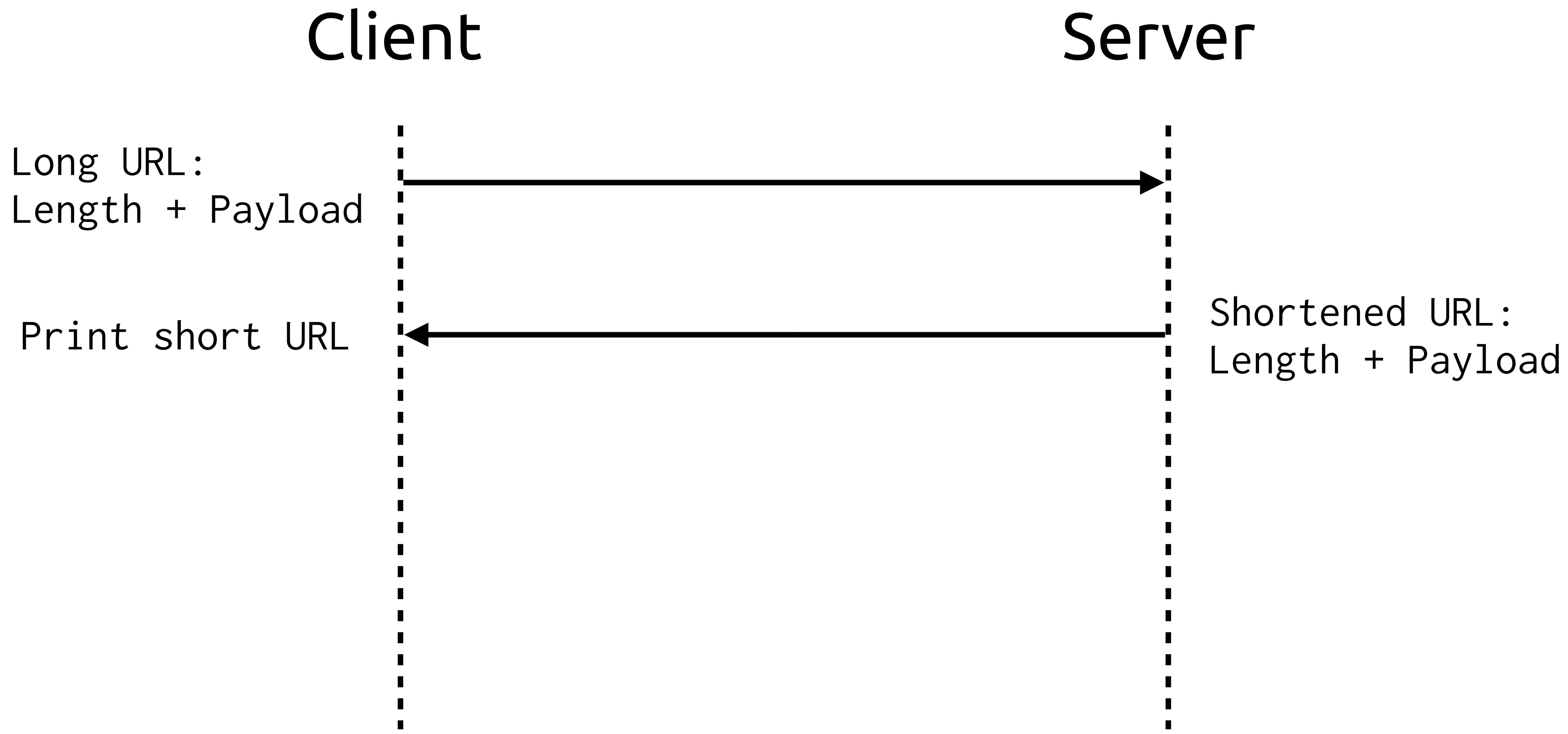
Shorty: URL shortener

- TCP
- Message framing



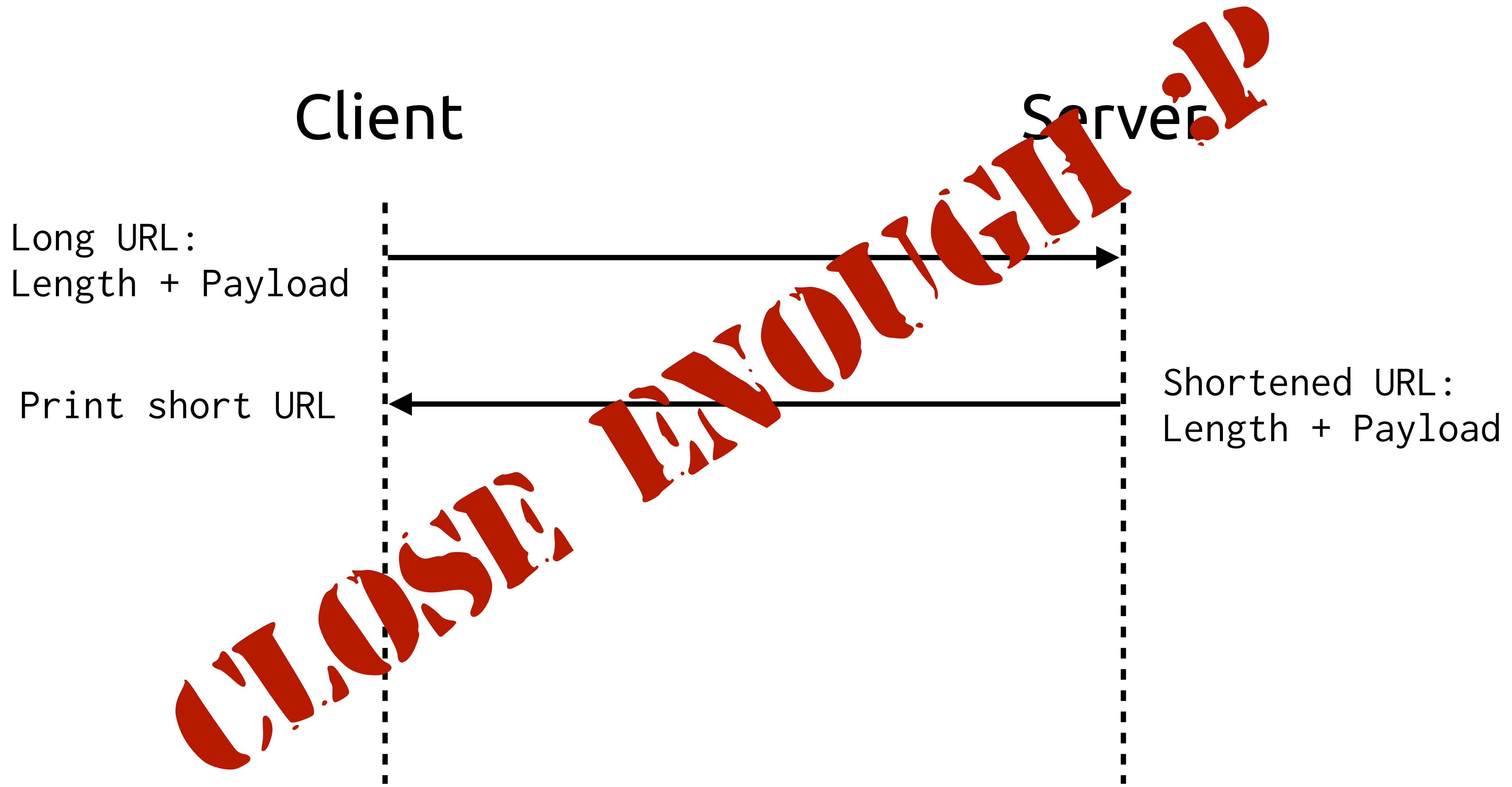
Shorty: URL shortener

- TCP
- Message framing



Shorty: URL shortener

- TCP
- Message framing



Blocking server

```
...
server = socket(AF_INET, SOCK_STREAM)
server.setsockopt(SOL_SOCKET, SO_REUSEADDR, 1)
server.bind(('', 9000))
server.listen(500)

def handle(conn, addr):
    logger.info('connected to {0}'.format(addr))
    time.sleep(1)  # delay
    # read payload
    payload_len_buf = read_bytes(conn, PLEN_BUF_SIZE)
    payload_len = struct.unpack('<L', payload_len_buf)[0]
    payload_buf = read_bytes(conn, payload_len)

    # shorten url and send it back
    short_url = shorten(payload_buf)
    payload_len = struct.pack('<L', len(short_url))
    conn.sendall(payload_len + short_url)
    conn.close()

# accept and handle incoming client connections
while True:
    conn, addr = server.accept()
    handle(conn, addr)
```

Client

```
...
def run():
    # connect to server
    client = socket()
    client.connect(('', 9000))

    # send payload
    payload = 'http://127.0.0.1:5000/{0}'.format(uuid.uuid4())
    payload_len = struct.pack('<L', len(payload))
    client.sendall(payload_len + payload)

    # read payload
    payload_len_buf = read_bytes(client, PLEN_BUF_SIZE)
    payload_len = struct.unpack('<L', payload_len_buf)[0]
    payload_buf = read_bytes(client, payload_len)

    client.close()
    return payload_buf

if __name__ == '__main__':
    print run()
```

Client

```
...
def run():
    # connect to server
    client = socket()
    client.connect(('', 9000))

    # send payload
    payload = 'http://127.0.0.1:5000/{0}'.format(uuid.uuid4())
    payload_len = struct.pack('<L', len(payload))
    client.sendall(payload_len + payload)

    # read payload
    payload_len_buf = read_bytes(client, PLEN_BUF_SIZE)
    payload_len = struct.unpack('<L', payload_len_buf)[0]
    payload_buf = read_bytes(client, payload_len)

    client.close()
    return payload_buf

if __name__ == '__main__':
    print run()
```

Output:

```
http://127.0.0.1:5000/867nv
```

So how does it perform?

Benchmark against blocking server

```
...
CONCURRENCY = 10
...
class Agent(Thread):
    ...

if __name__ == '__main__':
    runner = Runner()
    start = time.time()

    for _ in range(CONCURRENCY):
        agent = Agent(runner)
        agent.setDaemon(True)
        agent.start()

    print 'spawned {0} agents'.format(runner.spawned_agents)
    while runner.running_agents > 0:
        time.sleep(1)
        print 'connections/second: {0}'.format(runner.conn_per_sec)
        runner.reset_conn_per_sec()

    end = time.time()
    elapsed = end - start
    print 'took: {0}'.format(elapsed)
```

Benchmark against blocking server

```
...
CONCURRENCY = 10
...
class Agent(Thread):
    ...

if __name__ == '__main__':
    runner = Runner()
    start = time.time()

    for _ in range(CONCURRENCY):
        agent = Agent(runner)
        agent.setDaemon(True)
        agent.start()

    print 'spawned {0} agents'.format(runner.spawned_agents)
    while runner.running_agents > 0:
        time.sleep(1)
        print 'connections/second: {0}'.format(runner.conn_per_sec)
        runner.reset_conn_per_sec()

    end = time.time()
    elapsed = end - start
    print 'took: {0}'.format(elapsed)
```

Output:

```
...
took: 11.0153269768
```

What did we discover?

- Blocks on each request
- Very slow
- Poor performance
- Isn’t web scale

Non-blocking server

```
import gevent
from gevent import monkey; monkey.patch_socket()
from gevent.pool import Pool
from gevent.socket import socket
...

...
pool = Pool(100)
server = socket(AF_INET, SOCK_STREAM)
server.setsockopt(SOL_SOCKET, SO_REUSEADDR, 1)
server.bind(('', 9000))
server.listen(50)

def handle(conn, addr):
    logger.info('connected to {0}'.format(addr))
    gevent.sleep(1)  # delay
    ...

# accept and handle incoming client connections
while True:
    conn, addr = server.accept()
    pool.spawn(handle, conn, addr)
```

Non-blocking server

```
import gevent
from gevent import monkey; monkey.patch_socket()
from gevent.pool import Pool
from gevent.socket import socket
...

...
pool = Pool(100)
server = socket(AF_INET, SOCK_STREAM)
server.setsockopt(SOL_SOCKET, SO_REUSEADDR, 1)
server.bind(('', 9000))
server.listen(50)

def handle(conn, addr):
    logger.info('connected to {0}'.format(addr))
    gevent.sleep(1)  # delay
    ...

# accept and handle incoming client connections
while True:
    conn, addr = server.accept()
    pool.spawn(handle, conn, addr)
```

Output:

```
...
took: 2.00628781319
```

How about now?

- Doesn’t block on each request
- Significantly faster
- Better performance

Links

Gevent home page:
http://www.gevent.org/

Gevent for working python developer:
http://sdiehl.github.io/gevent-tutorial/

Multi-part file downloader using gevent:
https://github.com/marconi/pullite/tree/experiment

Slides and source files:
https://github.com/pizzapy/oct2013-meetup

Thank you

Marconi Moreto

@marconimjr

http://marconijr.com

https://github.com/marconi