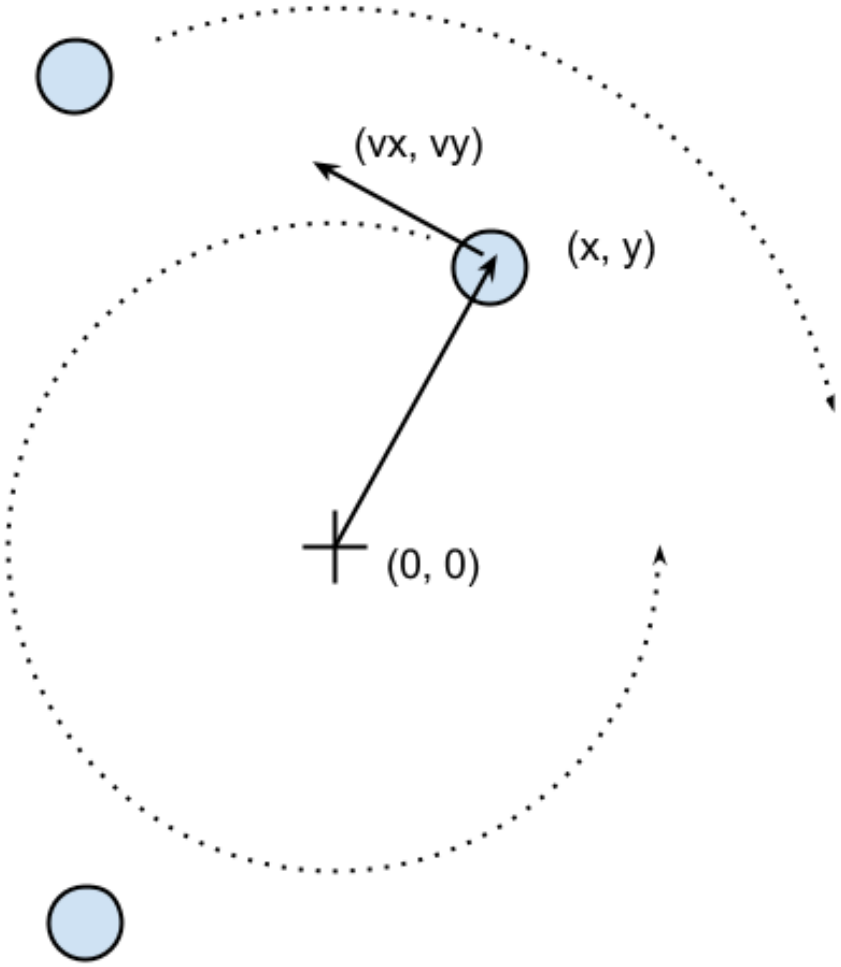
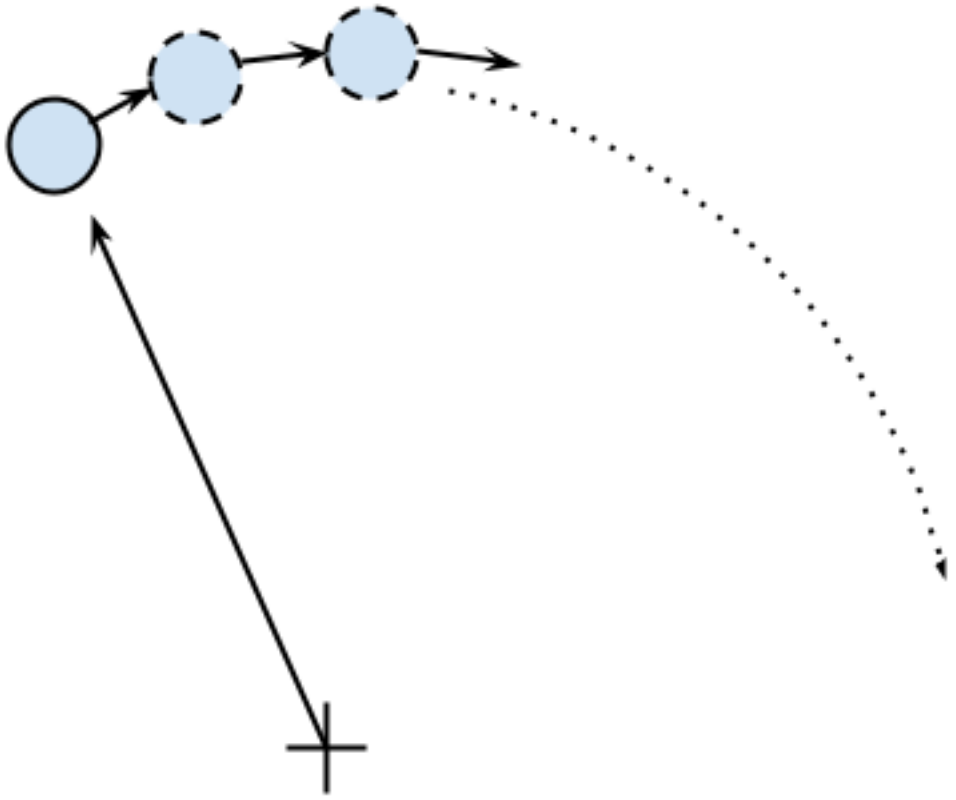
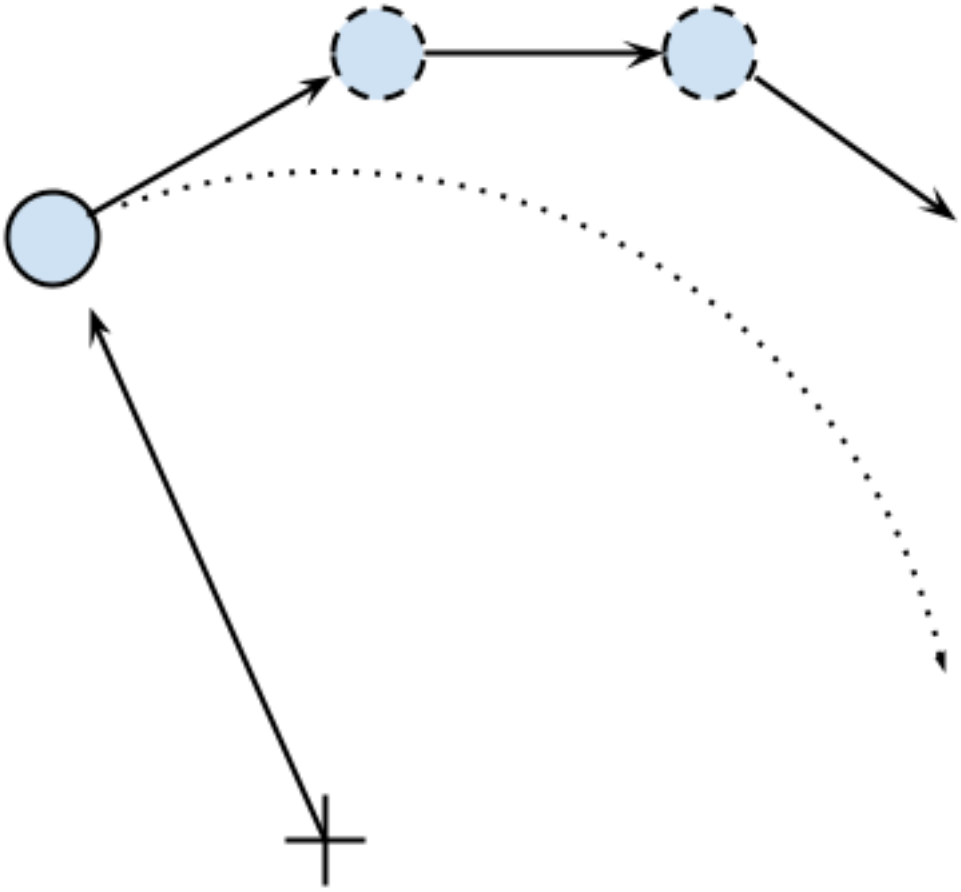
Chapter 1: Benchmarking and Profiling

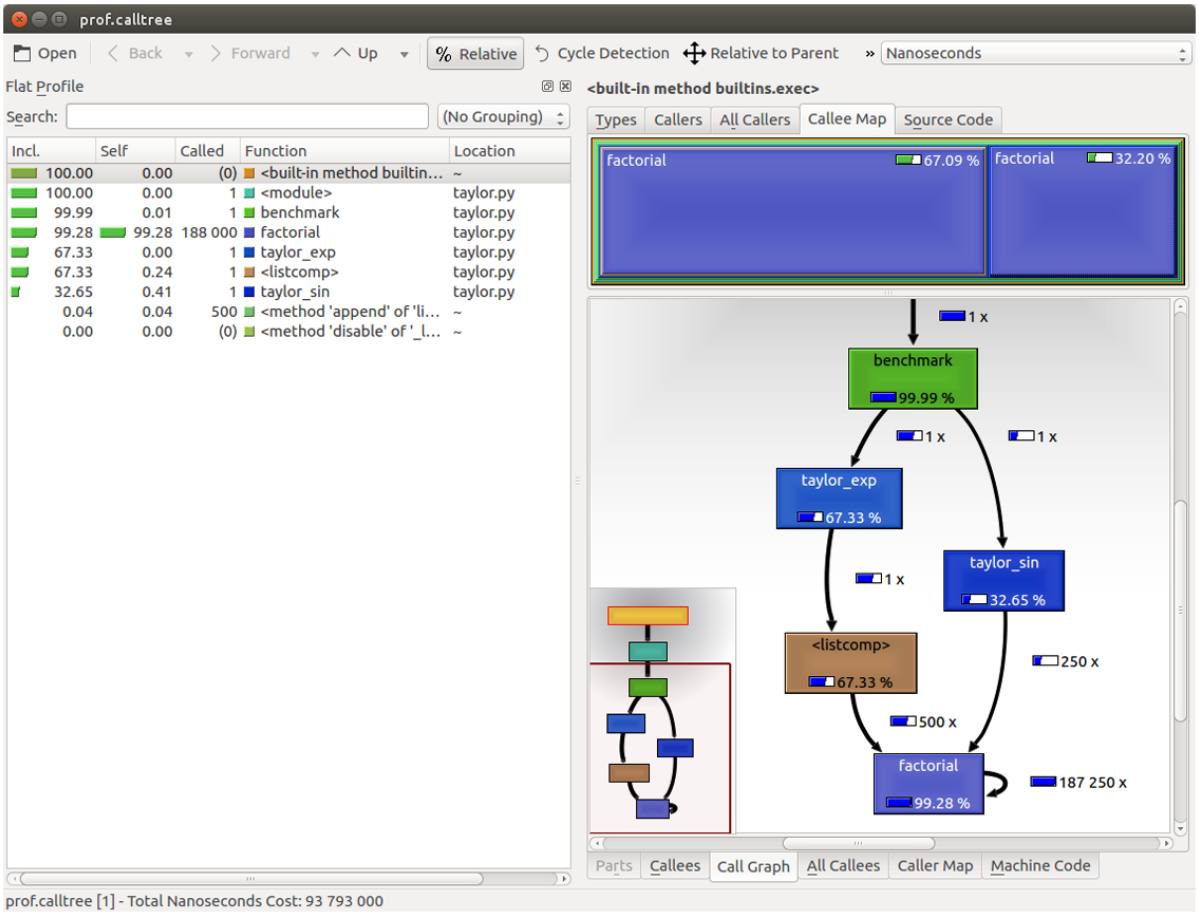






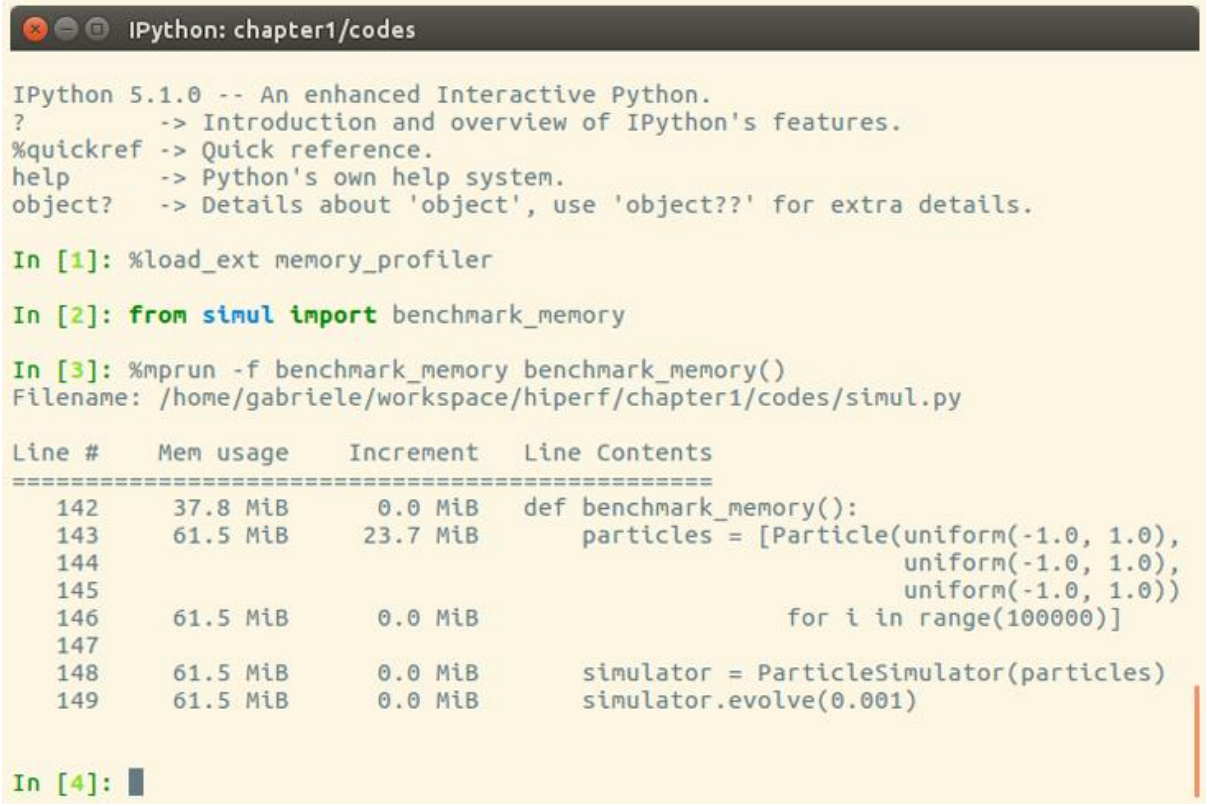
```
============================================= test session starts ==============================================
platform linux -- Python 3.5.2, pytest-3.0.5, py-1.4.32, pluggy-0.4.0
benchmark: 3.0.0 (defaults: timer=time.perf_counter disable_gc=False min_rounds=5 min_time=5.00us max_time=1.00s cal
ibration_precision=10 warmup=False warmup_iterations=100000)
rootdir: /home/gabriele/workspace/hiperf/chapter1, inifile:
plugins: benchmark-3.0.0
collected 2 items

test_simul.py .


------------------------------------------ benchmark: 1 tests ------------------------------------------
Name (time in ms)         Min      Max     Mean  StdDev   Median     IQR  Outliers(*)  Rounds  Iterations
--------------------------------------------------------------------------------------------------------
test_evolve           29.4716  41.1791  30.4622  2.0234  29.9630  0.7376          2;2      34           1
--------------------------------------------------------------------------------------------------------

(*) Outliers: 1 Standard Deviation from Mean; 1.5 IQR (InterQuartile Range) from 1st Quartile and 3rd Quartile.
========================================== 1 passed in 2.52 seconds ==========================================
```

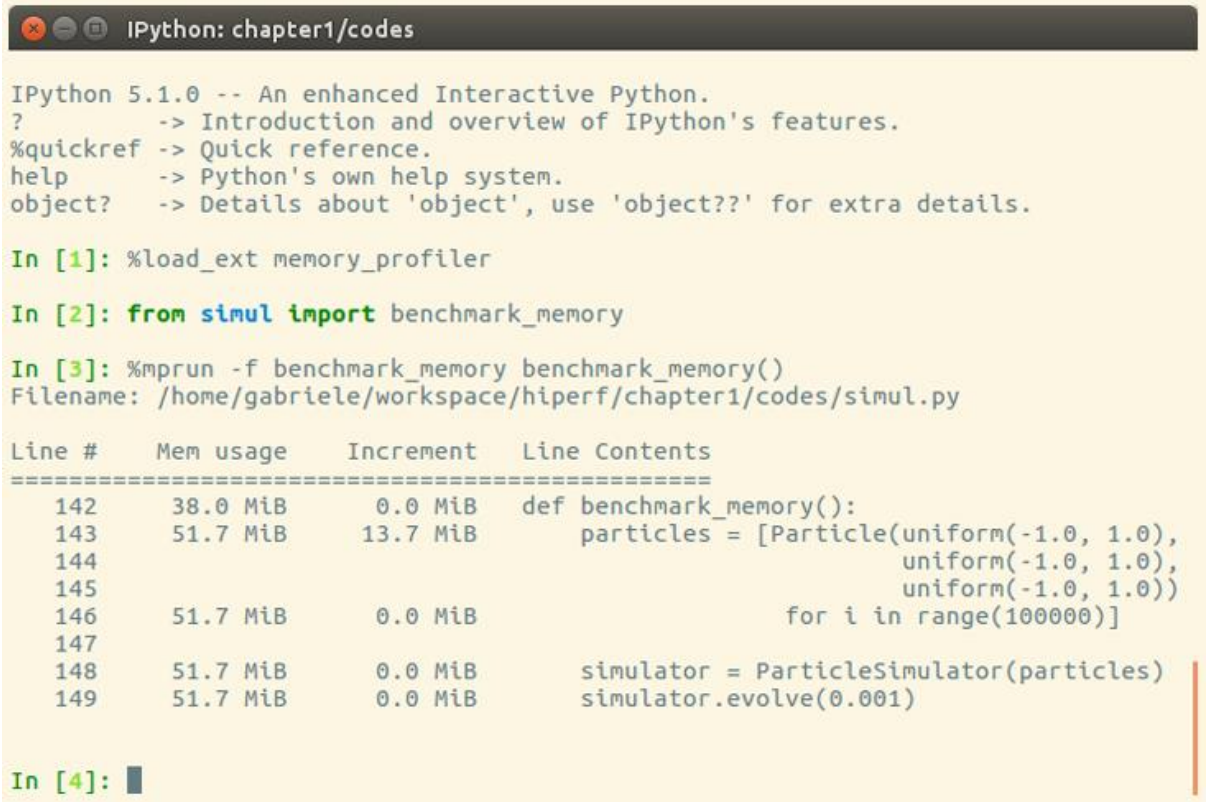
```
IPython: chapter1/codes
(hperf) → codes ipython
Python 3.5.2 |Continuum Analytics, Inc.| (default, Jul  2 2016, 17:53:06)
Type "copyright", "credits" or "license" for more information.

IPython 5.1.0 -- An enhanced Interactive Python.
?         -> Introduction and overview of IPython's features.
%quickref -> Quick reference.
help      -> Python's own help system.
object?   -> Details about 'object', use 'object??' for extra details.

In [1]: from simul import benchmark

In [2]: %prun benchmark()
         707 function calls in 1.231 seconds

   Ordered by: internal time

   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
        1    1.230    1.230    1.230    1.230 simul.py:21(evolve)
        1    0.000    0.000    0.001    0.001 simul.py:118(<listcomp>)
      300    0.000    0.000    0.000    0.000 random.py:342(uniform)
      100    0.000    0.000    0.000    0.000 simul.py:10(__init__)
      300    0.000    0.000    0.000    0.000 {method 'random' of '_random.Random' objects}
        1    0.000    0.000    1.231    1.231 {built-in method builtins.exec}
        1    0.000    0.000    1.231    1.231 <string>:1(<module>)
        1    0.000    0.000    1.231    1.231 simul.py:117(benchmark)
        1    0.000    0.000    0.000    0.000 simul.py:18(__init__)
        1    0.000    0.000    0.000    0.000 {method 'disable' of '_lsprof.Profiler' objects}

In [3]:
```



IPython: chapter1/codes

```
In [1]: %load_ext line_profiler

In [2]: from simul import benchmark, ParticleSimulator

In [3]: %lprun -f ParticleSimulator.evolve benchmark()
Timer unit: 1e-06 s

Total time: 8.66675 s
File: /home/gabriele/workspace/hiperf/chapter1/codes/simul.py
Function: evolve at line 21

Line #      Hits         Time  Per Hit   % Time  Line Contents
==============================================================
    21                                           def evolve(self, dt):
    22         1            2      2.0      0.0          timestep = 0.00001
    23         1            4      4.0      0.0          nsteps = int(dt/timestep)
    24
    25     10001        12561      1.3      0.1          for i in range(nsteps):
    26   1010000       867457      0.9     10.0              for p in self.particles:
    27
    28   1000000      1859312      1.9     21.5                  norm = (p.x**2 + p.y**2)**0.5
    29   1000000       972028      1.0     11.2                  v_x = (-p.y)/norm
    30   1000000       921008      0.9     10.6                  v_y = p.x/norm
    31
    32   1000000       982441      1.0     11.3                  d_x = timestep * p.ang_vel * v_x
    33   1000000       974838      1.0     11.2                  d_y = timestep * p.ang_vel * v_y
    34
    35   1000000      1058183      1.1     12.2                  p.x += d_x
    36   1000000      1018915      1.0     11.8                  p.y += d_y

In [4]:
```


IPython: chapter1/codes
IPython 5.1.0 -- An enhanced Interactive Python.
? -> Introduction and overview of IPython's features.
%quickref -> Quick reference.
help -> Python's own help system.
object? -> Details about 'object', use 'object??' for extra details.
In [1]: %load_ext memory_profiler
In [2]: from simul import benchmark_memory
In [3]: %mprun -f benchmark_memory benchmark_memory()
Filename: /home/gabriele/workspace/hiperf/chapter1/codes/simul.py
Line # Mem usage Increment Line Contents
142 37.8 MiB 0.0 MiB def benchmark_memory():
143 61.5 MiB 23.7 MiB particles = [Particle(uniform(-1.0, 1.0),
144 uniform(-1.0, 1.0),
145 uniform(-1.0, 1.0))
146 61.5 MiB 0.0 MiB for i in range(100000)]
147
148 61.5 MiB 0.0 MiB simulator = ParticleSimulator(particles)
149 61.5 MiB 0.0 MiB simulator.evolve(0.001)
In [4]:


IPython: chapter1/codes
IPython 5.1.0 -- An enhanced Interactive Python.
? -> Introduction and overview of IPython's features.
%quickref -> Quick reference.
help -> Python's own help system.
object? -> Details about 'object', use 'object??' for extra details.
In [1]: %load_ext memory_profiler
In [2]: from simul import benchmark_memory
In [3]: %mprun -f benchmark_memory benchmark_memory()
Filename: /home/gabriele/workspace/hiperf/chapter1/codes/simul.py
Line # Mem usage Increment Line Contents
142 38.0 MiB 0.0 MiB def benchmark_memory():
143 51.7 MiB 13.7 MiB particles = [Particle(uniform(-1.0, 1.0),
144 uniform(-1.0, 1.0),
145 uniform(-1.0, 1.0))
146 51.7 MiB 0.0 MiB for i in range(100000)]
147
148 51.7 MiB 0.0 MiB simulator = ParticleSimulator(particles)
149 51.7 MiB 0.0 MiB simulator.evolve(0.001)
In [4]:

Chapter 2: Pure Python Optimizations

Code	N=10000 (µs)	N=20000 (µs)	N=30000 (µs)	Time
`list.pop()`	0.50	0.59	0.58	*O*(1)
`list.pop(0)`	4.20	8.36	12.09	*O*(*N*)
`list.append(1)`	0.43	0.45	0.46	*O*(1)
`list.insert(0, 1)`	6.20	11.97	17.41	*O*(*N*)

Code	N=10000 (µs)	N=20000 (µs)	N=30000 (µs)	Time
`deque.pop()`	0.41	0.47	0.51	*O*(1)
`deque.popleft()`	0.39	0.51	0.47	*O*(1)
`deque.append(1)`	0.42	0.48	0.50	*O*(1)
`deque.appendleft(1)`	0.38	0.47	0.51	*O*(1)

Code	N=10000 (µs)	N=20000 (µs)	N=30000 (µs)	Time
`deque[0]`	0.37	0.41	0.45	O(1)
`deque[N - 1]`	0.37	0.42	0.43	O(1)
`deque[int(N / 2)]`	1.14	1.71	2.48	O(N)

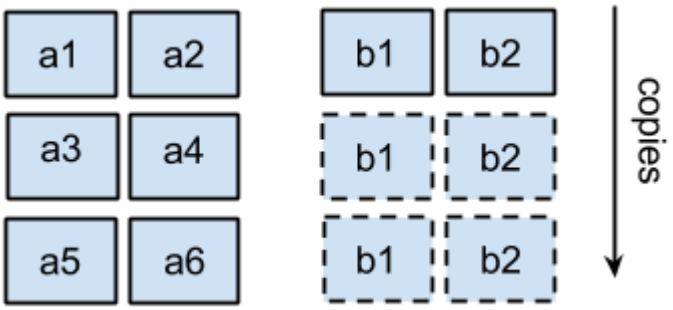
Code	N=10000 (µs)	N=20000 (µs)	N=30000 (µs)	Time
`list.index(a)`	87.55	171.06	263.17	*O*(N)
`index_bisect(list, a)`	3.16	3.20	4.71	*O*(log(N))

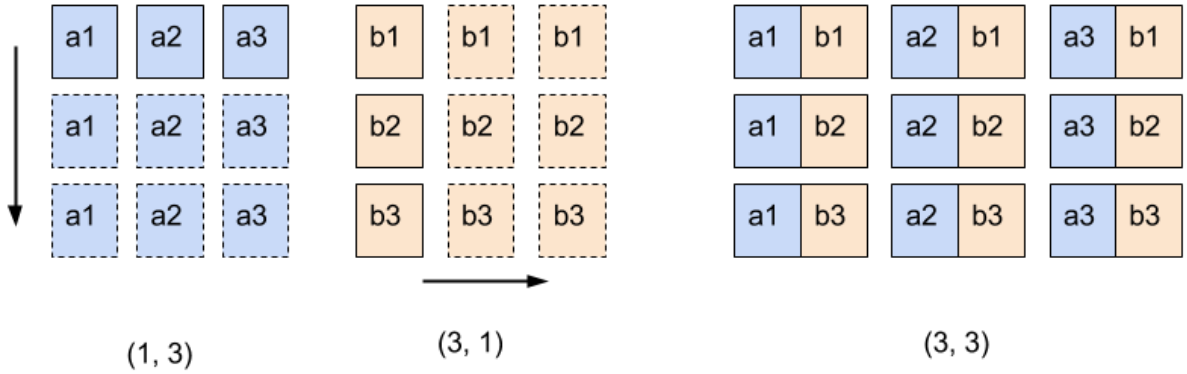
Code	N=1000 (µs)	N=2000 (µs)	N=3000 (µs)	Time
`Counter(items)`	51.48	96.63	140.26	O(N)
`counter_dict(items)`	111.96	197.13	282.79	O(N)
`counter_defaultdict(items)`	120.90	238.27	359.60	O(N)

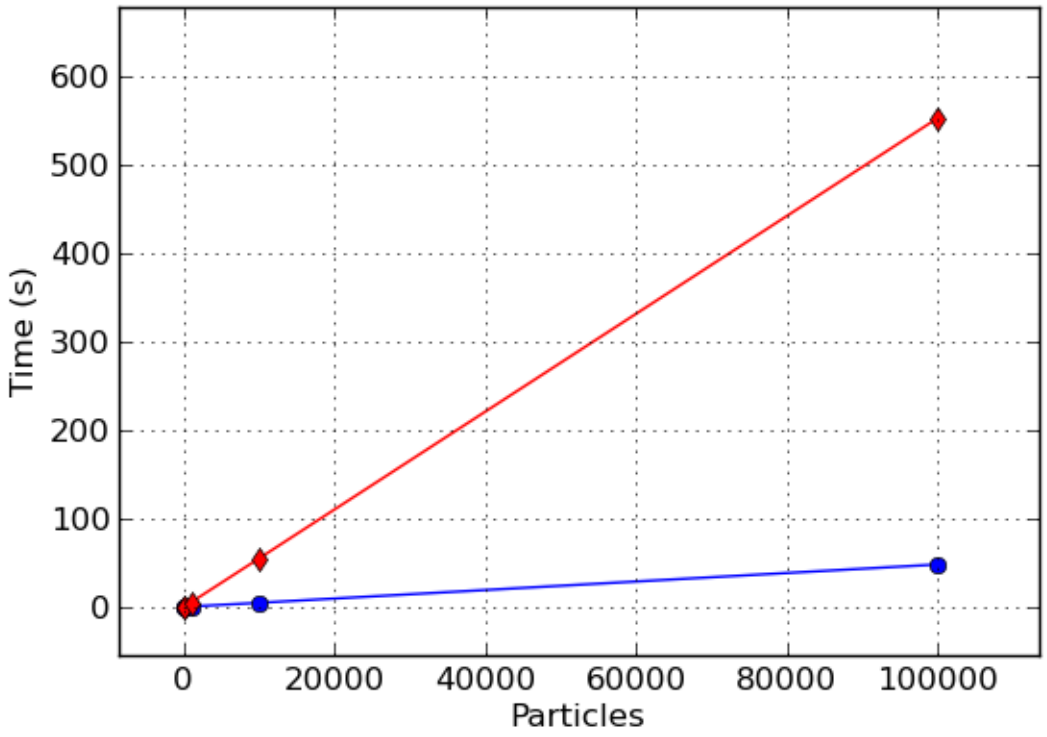
Code	Time
`s.union(t)`	*O*(*S* + *T*)
`s.intersection(t)`	*O*(*min*(*S*, *T*))
`s.difference(t)`	*O*(*S*)

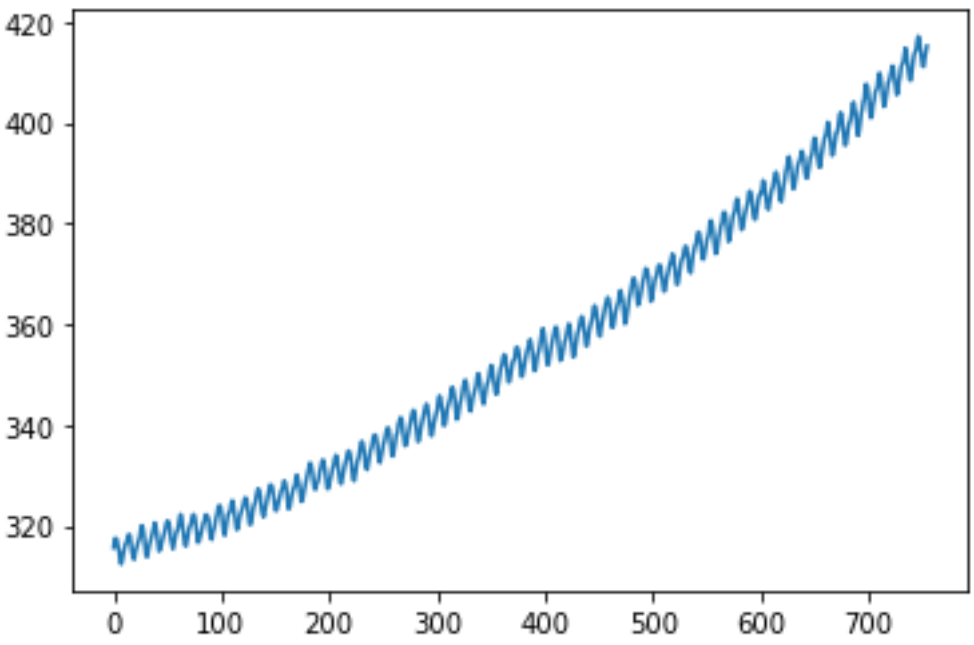
Algorithm	N=10000 (μs)	N=20000 (μs)	N=30000 (μs)	Time
Trie	17.12	17.27	17.47	O(S)
Linear scan	1978.44	4075.72	6398.06	O(N)

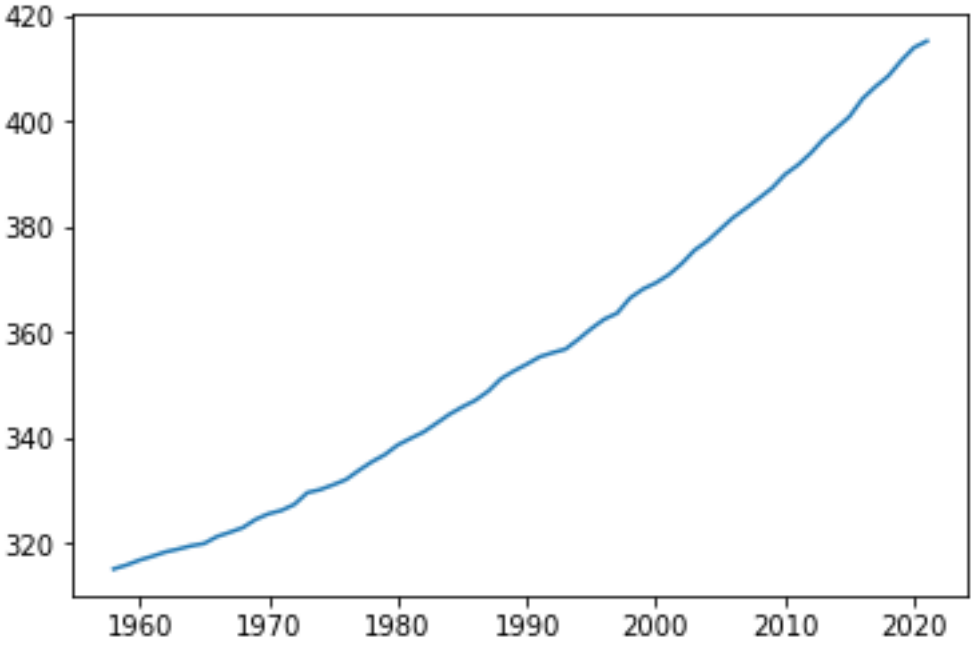
Chapter 3: Fast Array Operations with NumPy and Pandas

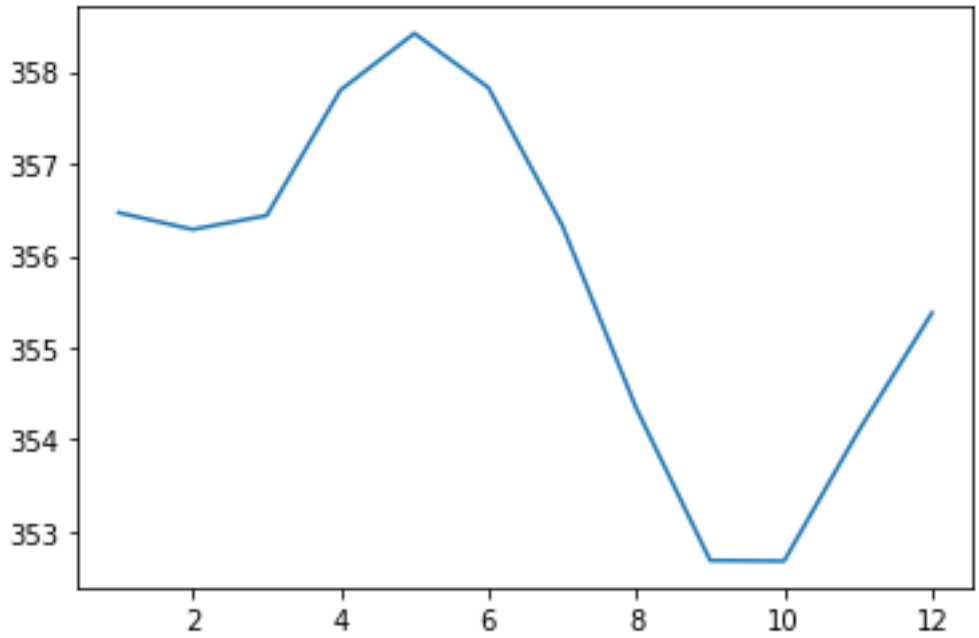







420
400
380
360
340
320
0
100
200
300
400
500
600
700


420
400
380
360
340
320
1960
1970
1980
1990
2000
2010
2020



[7]:
```
ds = xr.Dataset.from_dataframe(df)
ds
```

[7]: xarray.Dataset

▸ Dimensions: (**month**: 12, **year**: 64)

▾ Coordinates:

year	(year)	int64	1958 1959 1960 ... 2019 2020 2021
month	(month)	int64	1 2 3 4 5 6 7 8 9 10 11 12

▾ Data variables:

co2	(year, month)	float64	nan nan 315.7 317.4 ... nan nan nan

▸ Attributes: (0)

[9]:
```
ds['co2'][:10, :5]
```

[9]: xarray.DataArray 'co2' (**year**: 10, **month**: 5)

```
array([[   nan,    nan, 315.7 , 317.45, 317.51],
       [315.58, 316.49, 316.65, 317.72, 318.29],
       [316.43, 316.98, 317.58, 319.03, 320.04],
       [316.89, 317.7 , 318.54, 319.48, 320.58],
       [317.94, 318.55, 319.68, 320.57, 321.02],
       [318.74, 319.07, 319.86, 321.38, 322.25],
       [319.57, 320.01, 320.74, 321.84, 322.26],
       [319.44, 320.44, 320.89, 322.14, 322.17],
       [320.62, 321.6 , 322.39, 323.7 , 324.08],
       [322.33, 322.5 , 323.04, 324.42, 325.  ]])
```

▼ Coordinates:

year	(year)	int64	1958 1959 1960 ... 1965 1966 1967
month	(month)	int64	1 2 3 4 5

▶ Attributes: (0)

[11]:
```
ds.mean(dim='month')
```

[11]: xarray.Dataset

▶ Dimensions: (**year**: 64)

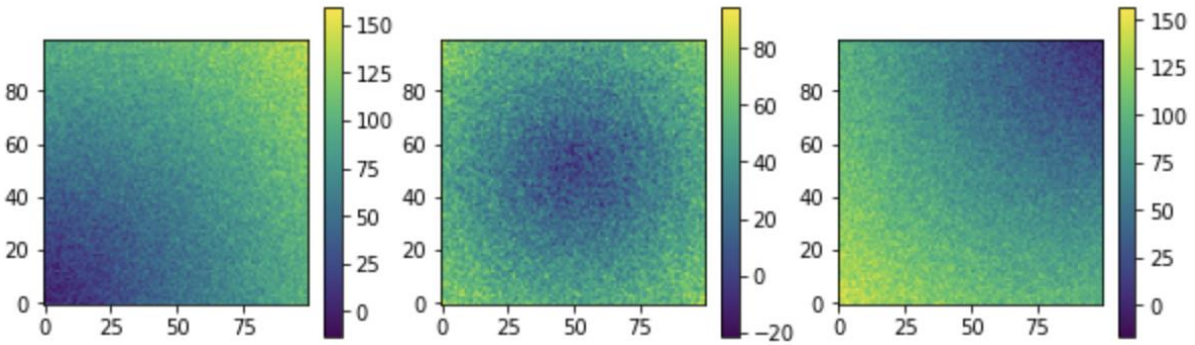
▼ Coordinates:

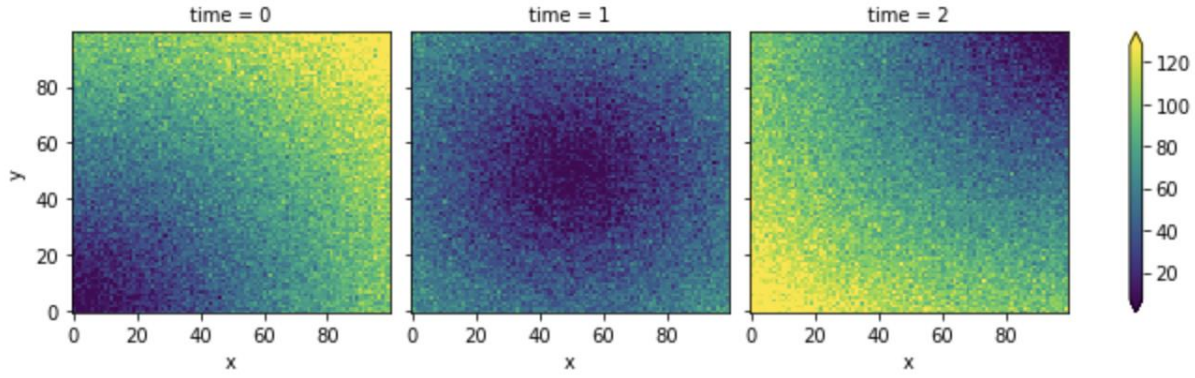
year	(year)	int64	1958 1959 1960 ... 2019 2020 2021

▼ Data variables:

co2	(year)	float64	315.2 316.0 316.9 ... 413.9 415.2

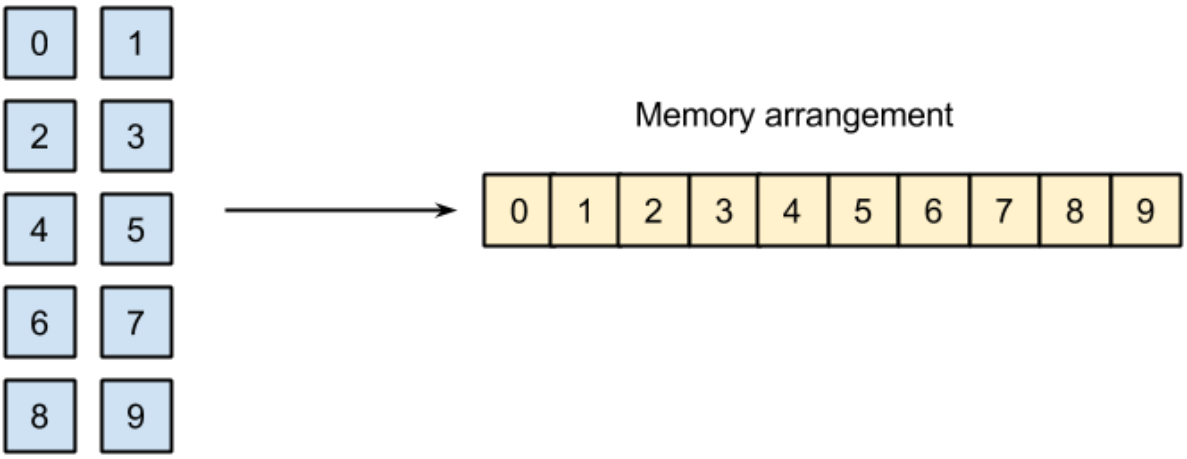
▶ Attributes: (0)





Index type	**N=10000**	**N=20000**	**N=30000**	**Time**
Unique	12.30	12.58	13.30	$O(1)$
Non-unique	494.95	814.10	1129.95	$O(N)$
Non-unique (sorted)	145.93	145.81	145.66	$O(log(N))$

Chapter 4: C Performance with Cython



generated for it.

Raw output: cevolve.c

```
+01: import numpy as np
 02: cimport cython
 03: from libc.math cimport sqrt
 04:
+05: def c_evolve(double[:, :] r_i,double[:] ang_speed_i,
 06:               double timestep,int nsteps):
 07:     cdef int i
 08:     cdef int j
+09:     cdef int nparticles = r_i.shape[0]
 10:     cdef double norm, x, y, vx, vy, dx, dy, ang_speed
 11:
 12:
+13:     for i in range(nsteps):
+14:         for j in range(nparticles):
+15:             x = r_i[j, 0]
+16:             y = r_i[j, 1]
+17:             ang_speed = ang_speed_i[j]
 18:
+19:             norm = sqrt(x ** 2 + y ** 2)
 20:
+21:             vx = (-y)/norm
+22:             vy = x/norm
       if (unlikely(__pyx_v_norm == 0)) {
         #ifdef WITH_THREAD
         PyGILState_STATE __pyx_gilstate_save = PyGILState_Ensure();
         #endif
         PyErr_SetString(PyExc_ZeroDivisionError, "float division");
         #ifdef WITH_THREAD
         PyGILState_Release(__pyx_gilstate_save);
         #endif
         {__pyx_filename = __pyx_f[0]; __pyx_lineno = 22; __pyx_clineno = __LINE__; goto __pyx_L1_error;}
       }
       __pyx_v_vy = (__pyx_v_x / __pyx_v_norm);
 23:
+24:             dx = timestep * ang_speed * vx
+25:             dy = timestep * ang_speed * vy
 26:
+27:             r_i[j, 0] += dx
+28:             r_i[j, 1] += dy
 29:
```

In [15]:
```
%%cython -a
import numpy as np

cdef int max(int a, int b):
    return a if a > b else b

cdef int chebyshev(int x1, int y1, int x2, int y2):
    return max(abs(x1 - x2), abs(y1 - y2))

def c_benchmark():
    a = np.random.rand(1000, 2)
    b = np.random.rand(1000, 2)

    for x1, y1 in a:
        for x2, y2 in b:
            chebyshev(x1, x2, y1, y2)
```

Out[15]:

Generated by Cython 0.25.2

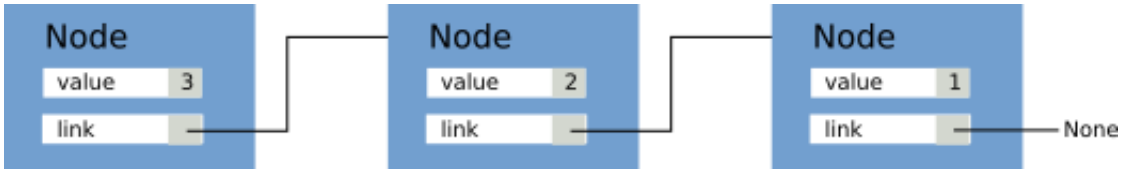
Yellow lines hint at Python interaction.
Click on a line that starts with a "+" to see the C code that Cython generated for it.

```
 01: # cython: profile=True
+02: import numpy as np
 03:
+04: cdef int max(int a, int b):
+05:     return a if a > b else b
 06:
+07: cdef int chebyshev(int x1, int y1, int x2, int y2):
+08:     return max(abs(x1 - x2), abs(y1 - y2))
 09:
+10: def c_benchmark():
+11:     a = np.random.rand(1000, 2)
+12:     b = np.random.rand(1000, 2)
 13:
+14:     for x1, y1 in a:
+15:         for x2, y2 in b:
+16:             chebyshev(x1, x2, y1, y2)
```

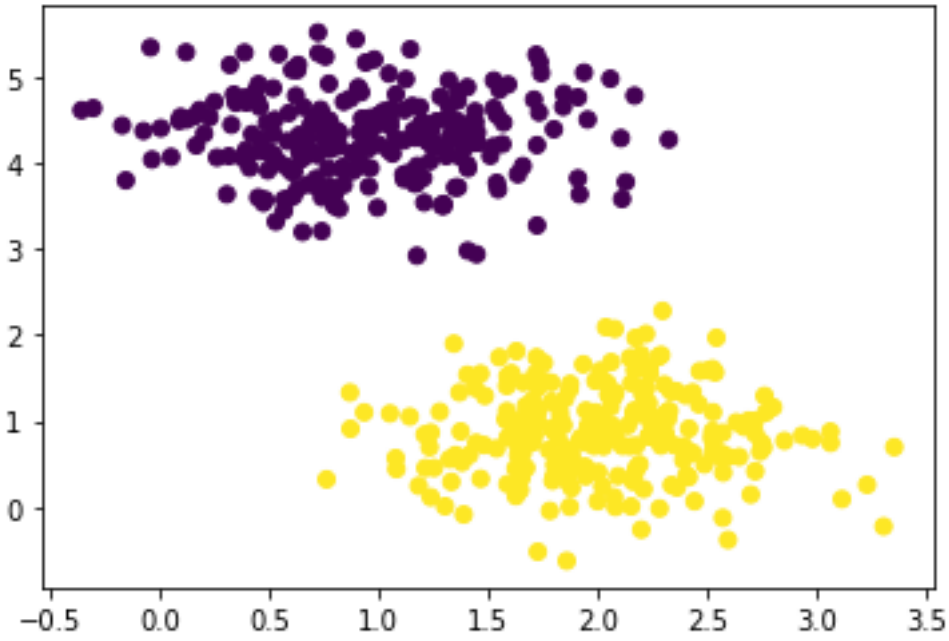
In [22]:
```
%prun c_benchmark()
```

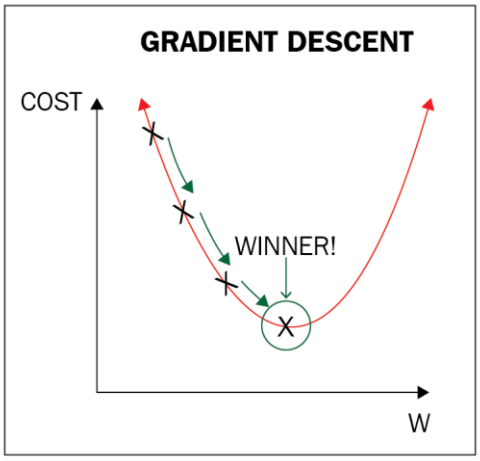
```
         2000005 function calls in 1.370 seconds

   Ordered by: internal time

   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
        1    1.127    1.127    1.370    1.370 _cython_magic_c7d6eab16ab5658137c9af8534d5cafb.pyx:10(c_benchma
rk)
  1000000    0.191    0.000    0.243    0.000 _cython_magic_c7d6eab16ab5658137c9af8534d5cafb.pyx:7(chebyshev)
  1000000    0.052    0.000    0.052    0.000 _cython_magic_c7d6eab16ab5658137c9af8534d5cafb.pyx:4(max)
        1    0.000    0.000    1.370    1.370 <string>:1(<module>)
        1    0.000    0.000    1.370    1.370 {built-in method builtins.exec}
        1    0.000    0.000    1.370    1.370 {_cython_magic_c7d6eab16ab5658137c9af8534d5cafb.c_benchmark}
        1    0.000    0.000    0.000    0.000 {method 'disable' of '_lsprof.Profiler' objects}
```

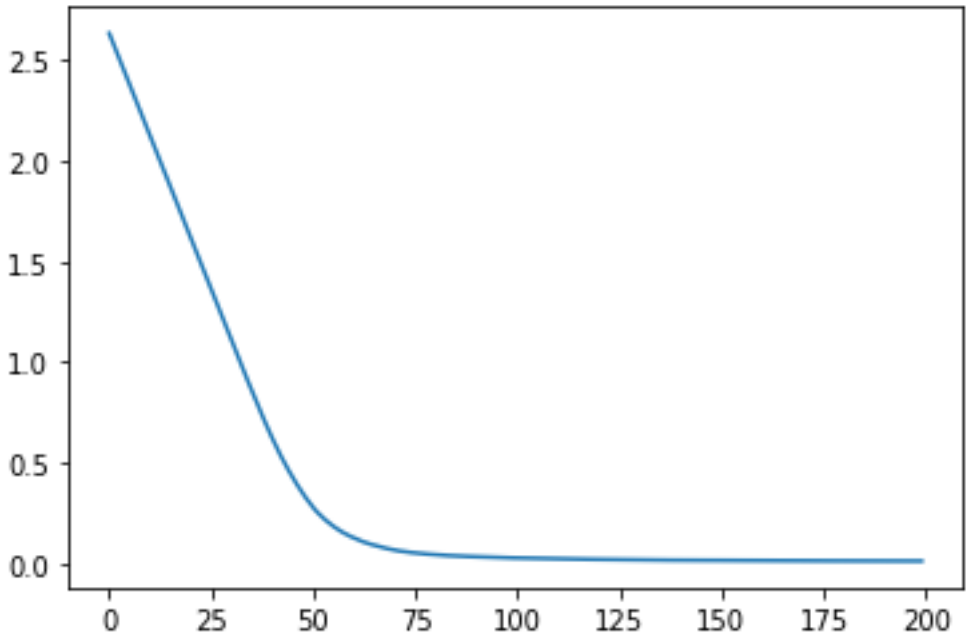
Chapter 5: Exploring Compilers

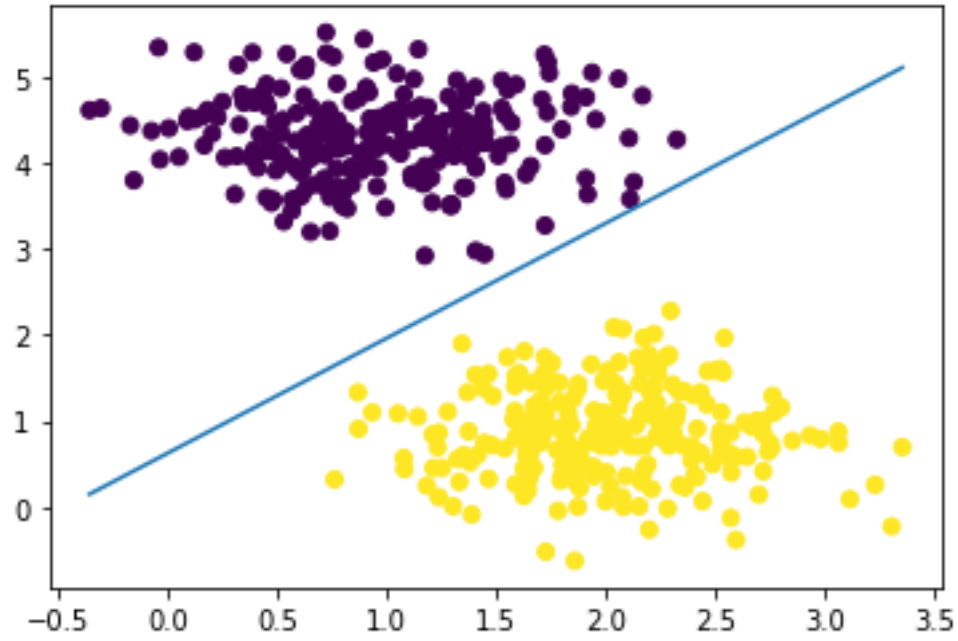


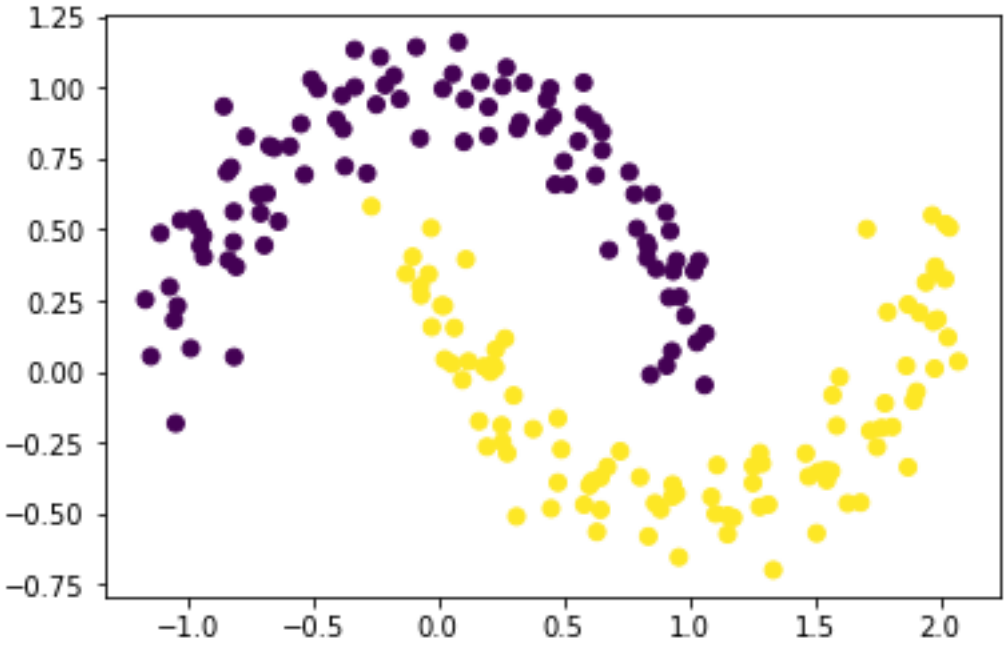
Chapter 6: Automatic Differentiation and Accelerated Linear Algebra for Machine Learning

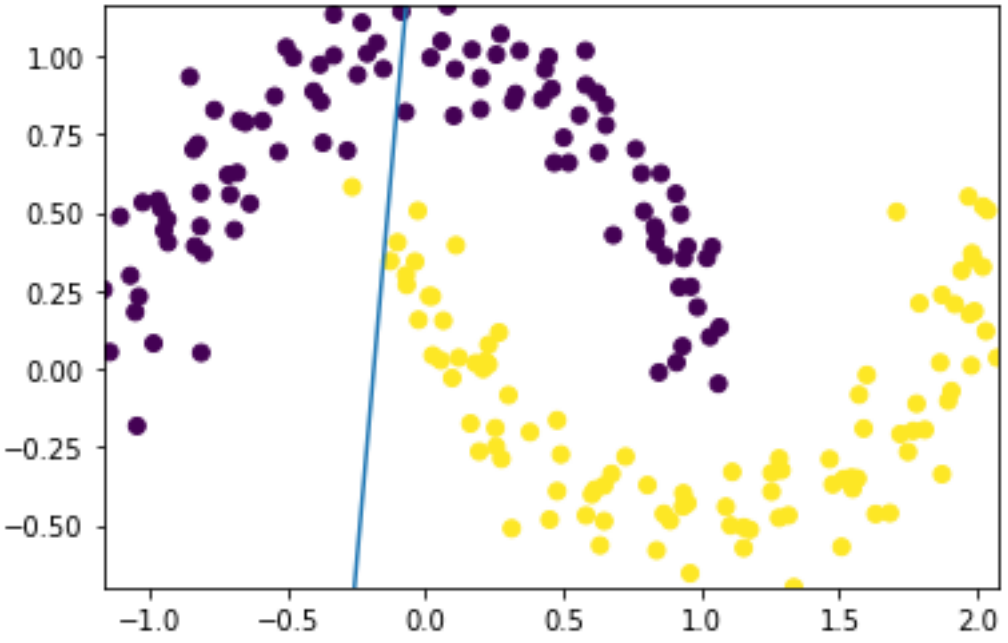


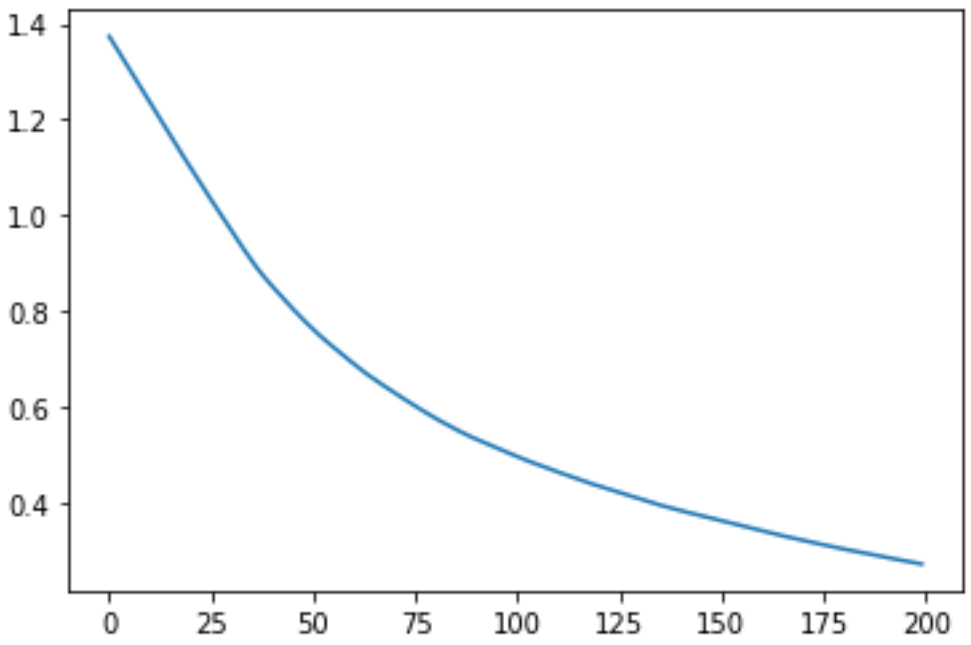


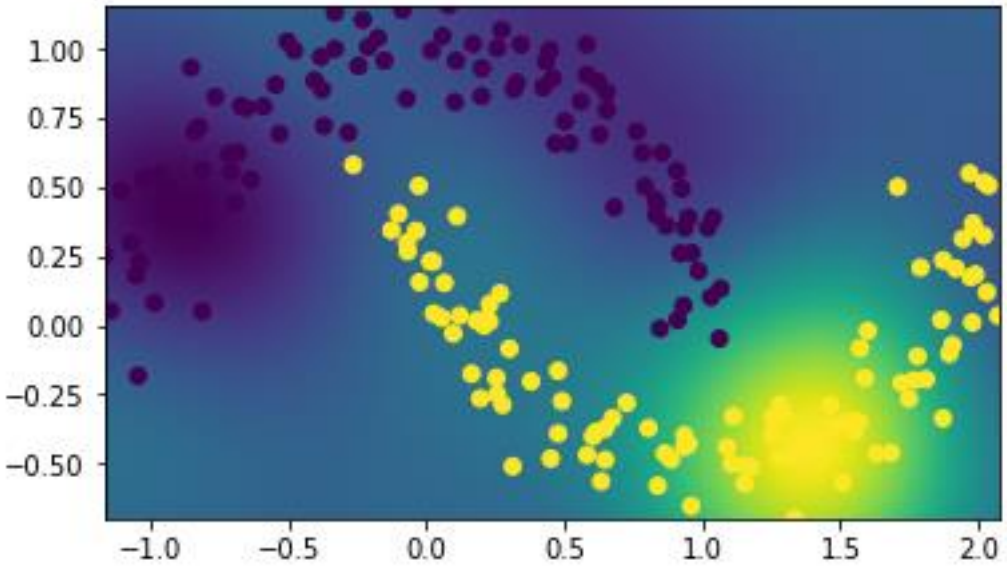

2.5
2.0
1.5
1.0
0.5
0.0
0
25
50
75
100
125
150
175
200


5
4
3
2
1
0
−0.5
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5

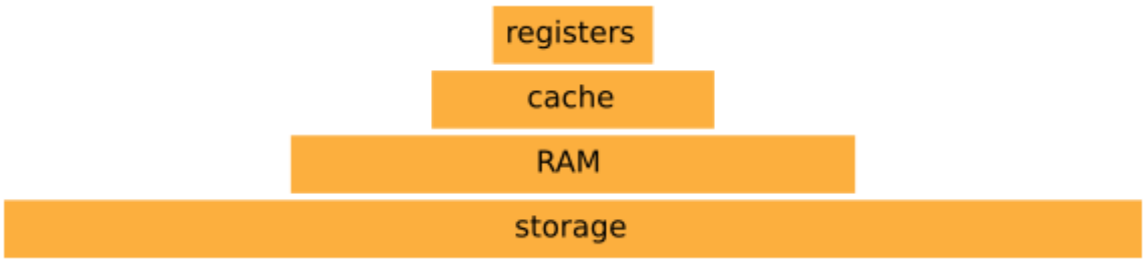

1.25
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0

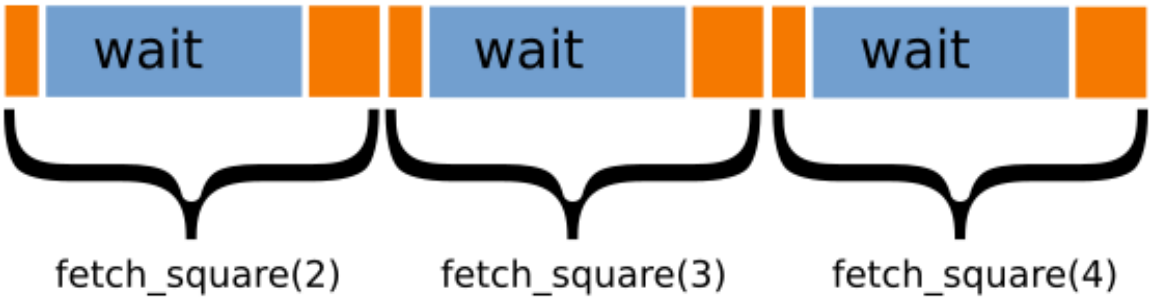

1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0

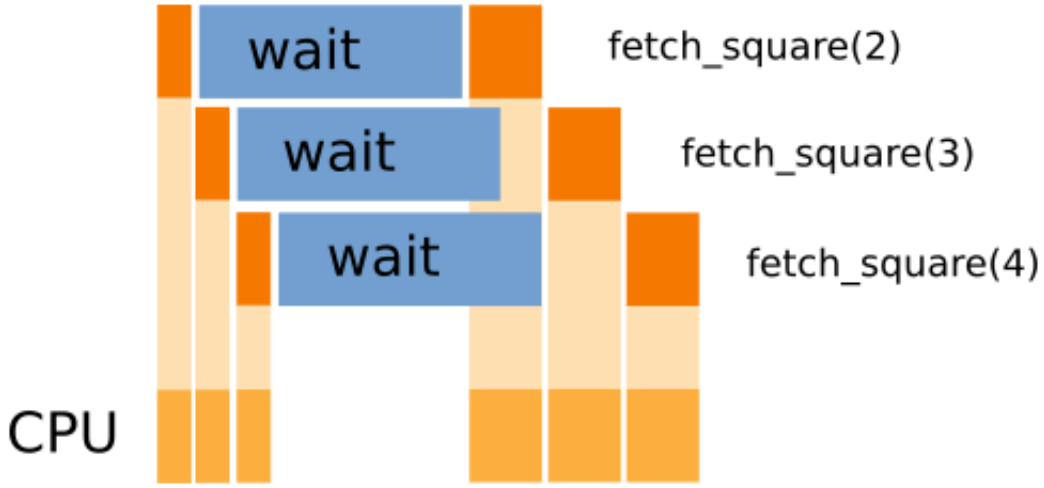

1.4
1.2
1.0
0.8
0.6
0.4
0
25
50
75
100
125
150
175
200


1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0

Chapter 7: Implementing Concurrency

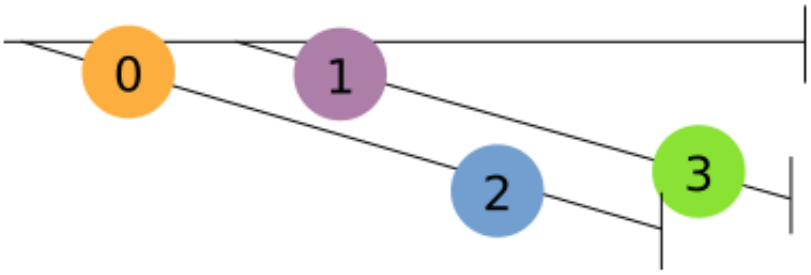


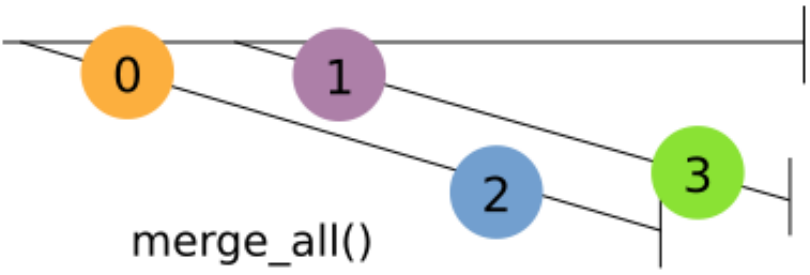




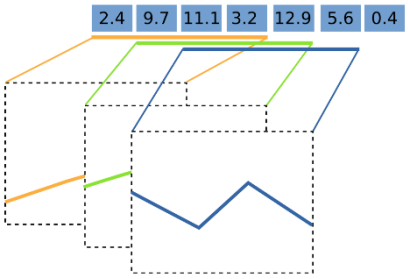
map(lambda x: x**2)

groupby(lambda x: x % 2)





merge_all()


2.4
9.7
11.1
3.2
12.9
5.6
0.4

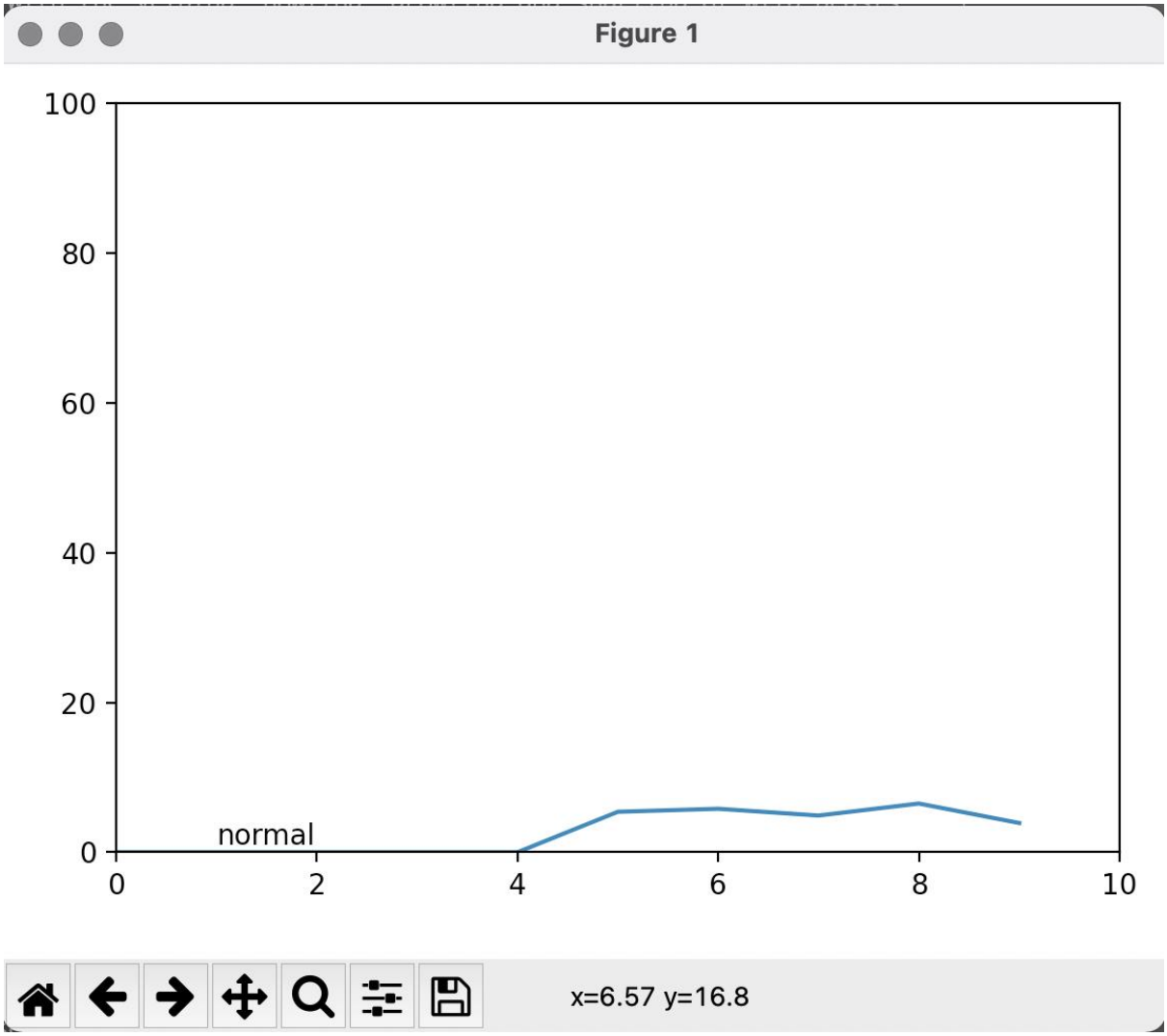
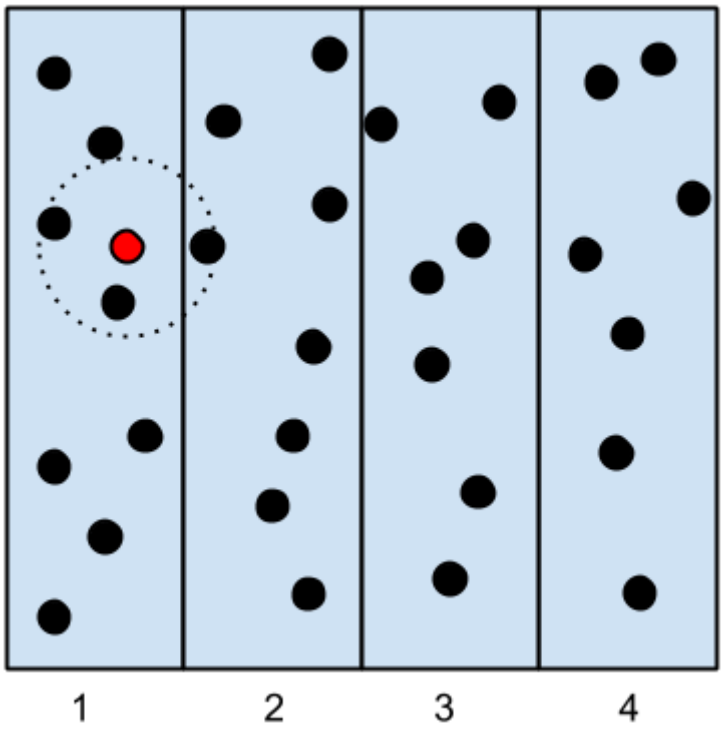
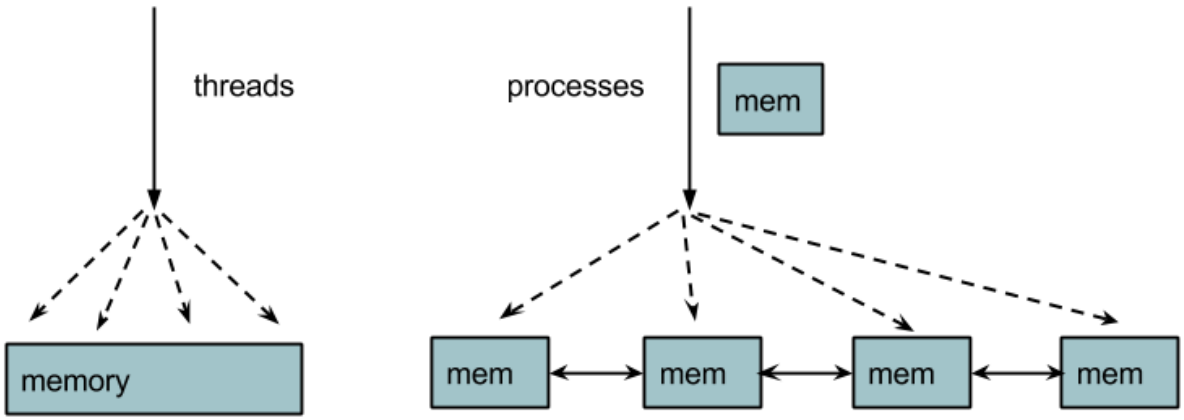
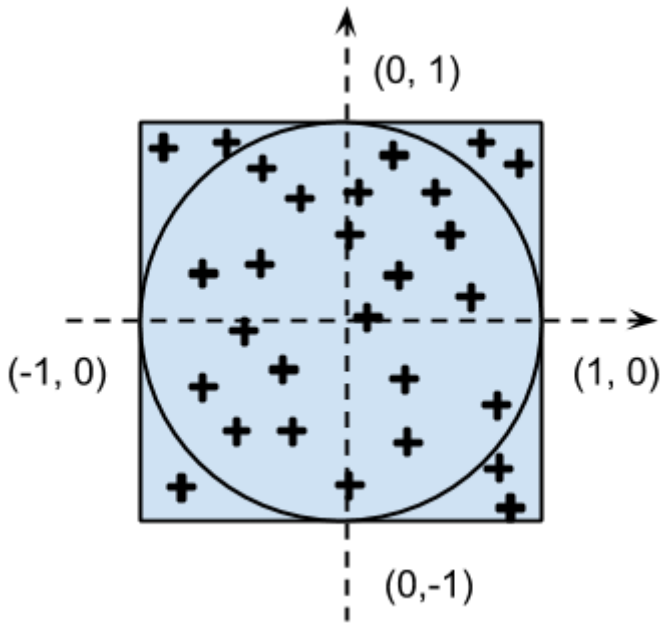


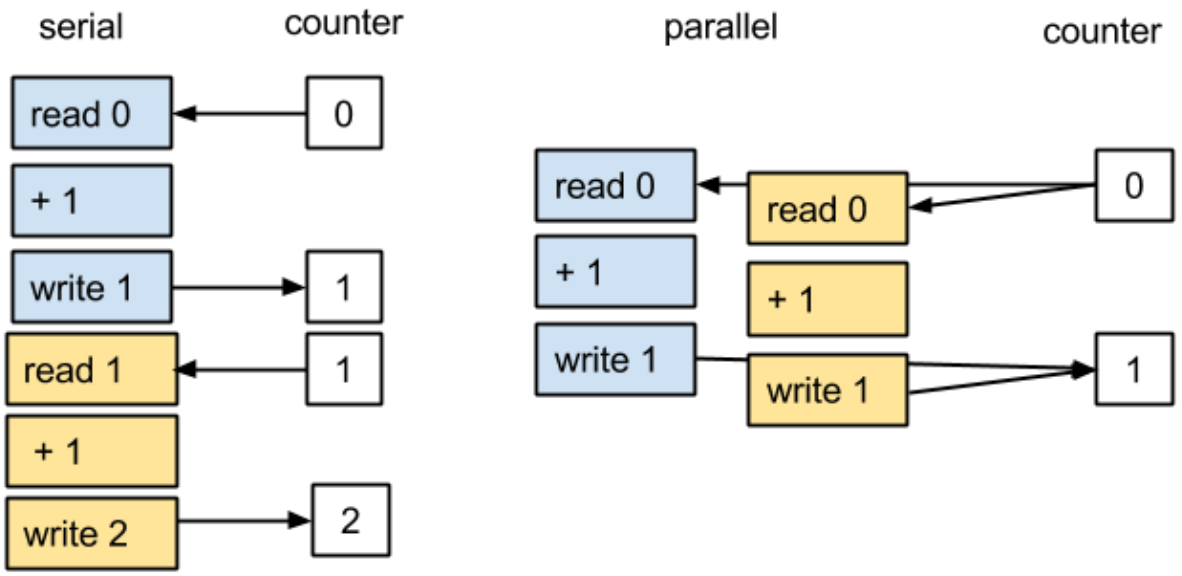
Figure 1
100
80
60
40
20
0
normal
0
2
4
6
8
10
x=6.57 y=16.8

Chapter 8: Parallel Processing






(0, 1)
(-1, 0)
(1, 0)
(0,-1)

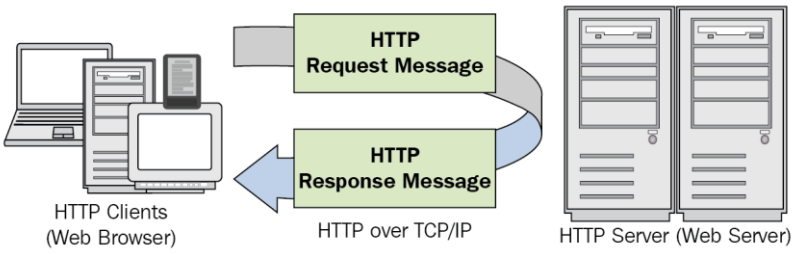

serial
counter
read 0
0
+ 1
write 1
1
read 1
1
+ 1
write 2
2
parallel
counter
read 0
read 0
0
+ 1
+ 1
write 1
write 1
1

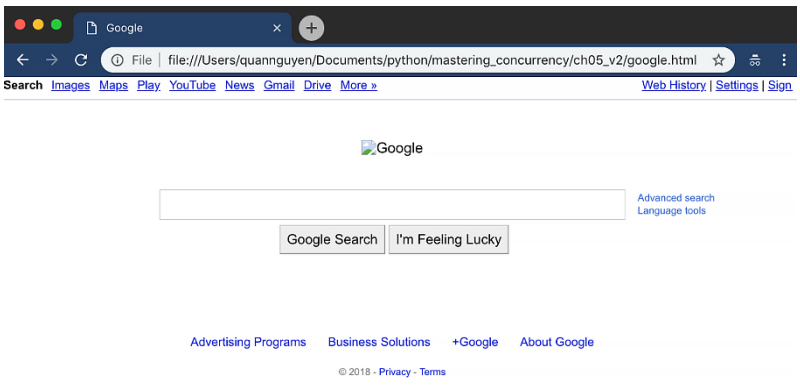
Chapter 9: Concurrent Web Requests

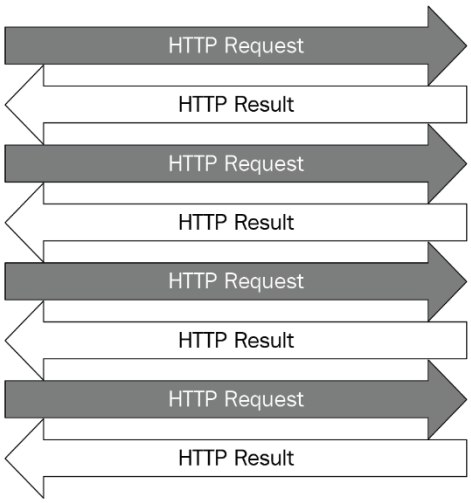
index.html | reuseheader.html | reusefooter.html | menu.html | events.html | styles.css

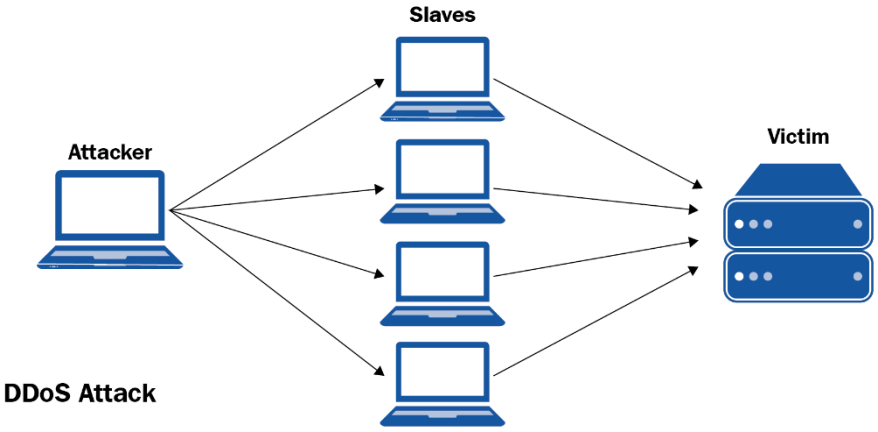
```
1
2  <div class="topNavTop">
3      <p>Welcome to Chilli restaurant</p>
4      <div class="topNavRight">
5          <img src="assets/top-nav/icon-phone.png">
6          <p>416-455-3221</p>
7          <img src="assets/top-nav/icon-email.png">
8          <p>info@company.com</p>
9          <img src="assets/top-nav/icon-magnifying-glass.png">
10     </div>
11 </div>
12 <div class="topNavBottom">
13     <img src="assets/chilli-logo.png">
14     <div class="topNavRightBottom">
15         <a href="index.html">HOME</a>
16         <a href="menu.html">MENU</a>
17         <a href="events.html">EVENTS</a>
18         <a href="#contact">CONTACT</a>
19     </div>
20 </div>
```

Line 1, Column 1 — 1 misspelled word — Tab Size: 4

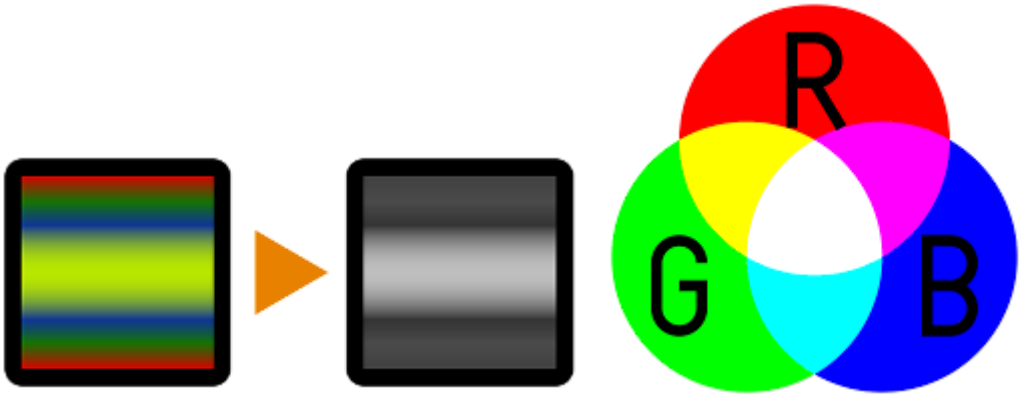


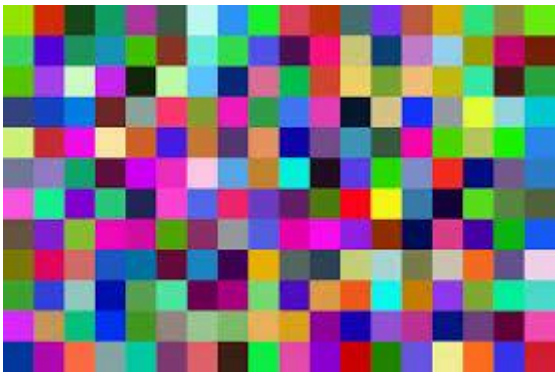



HTTP Request
HTTP Result
HTTP Request
HTTP Result
HTTP Request
HTTP Result
HTTP Request
HTTP Result


Slaves
Attacker
Victim
DDoS Attack

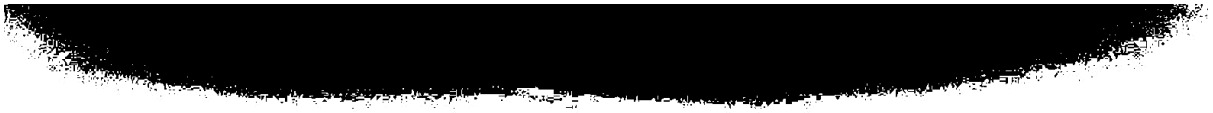
Chapter 10: Concurrent Image Processing

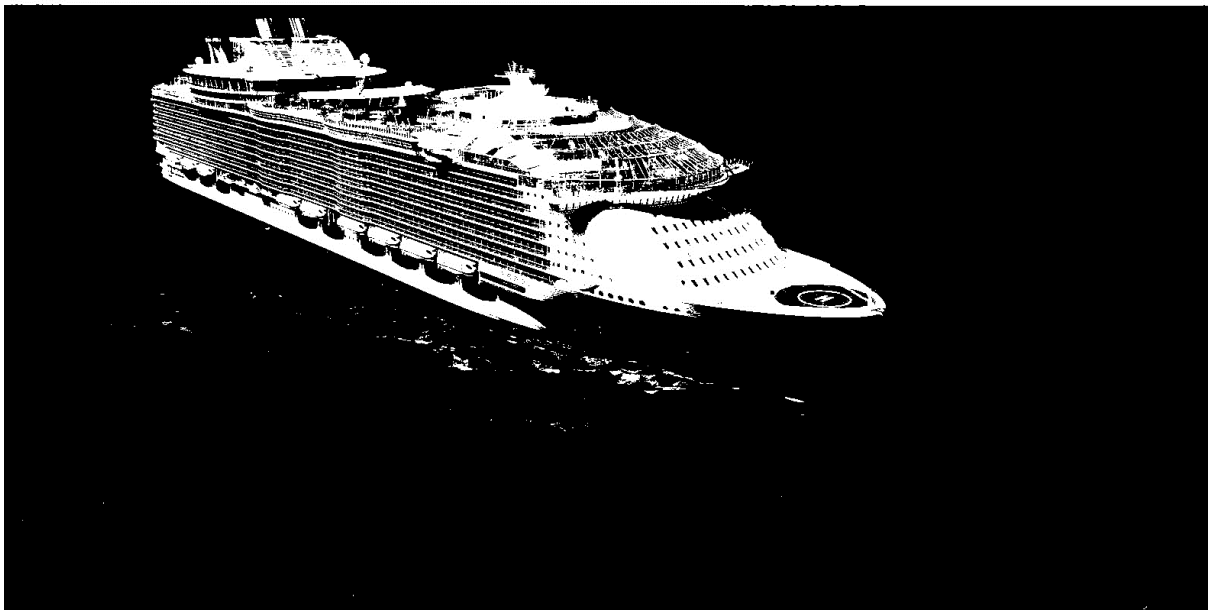


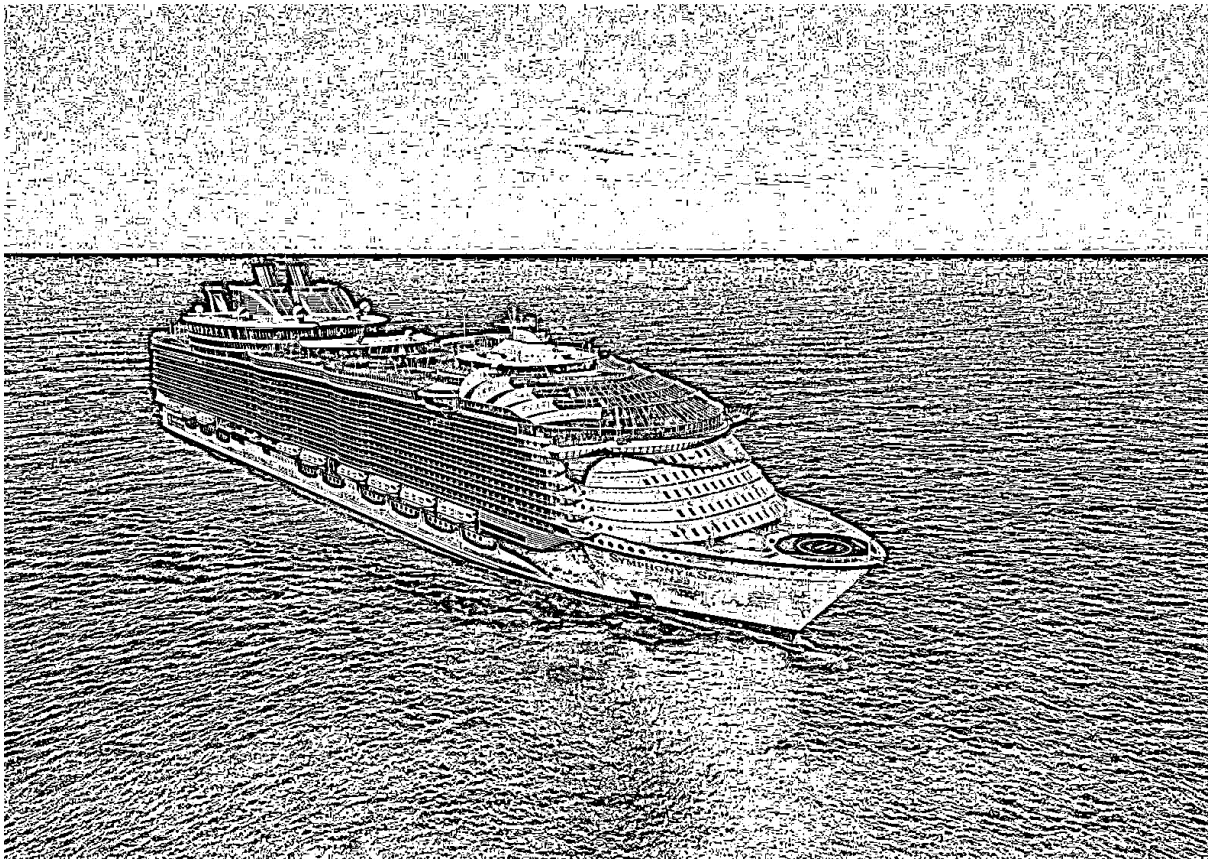


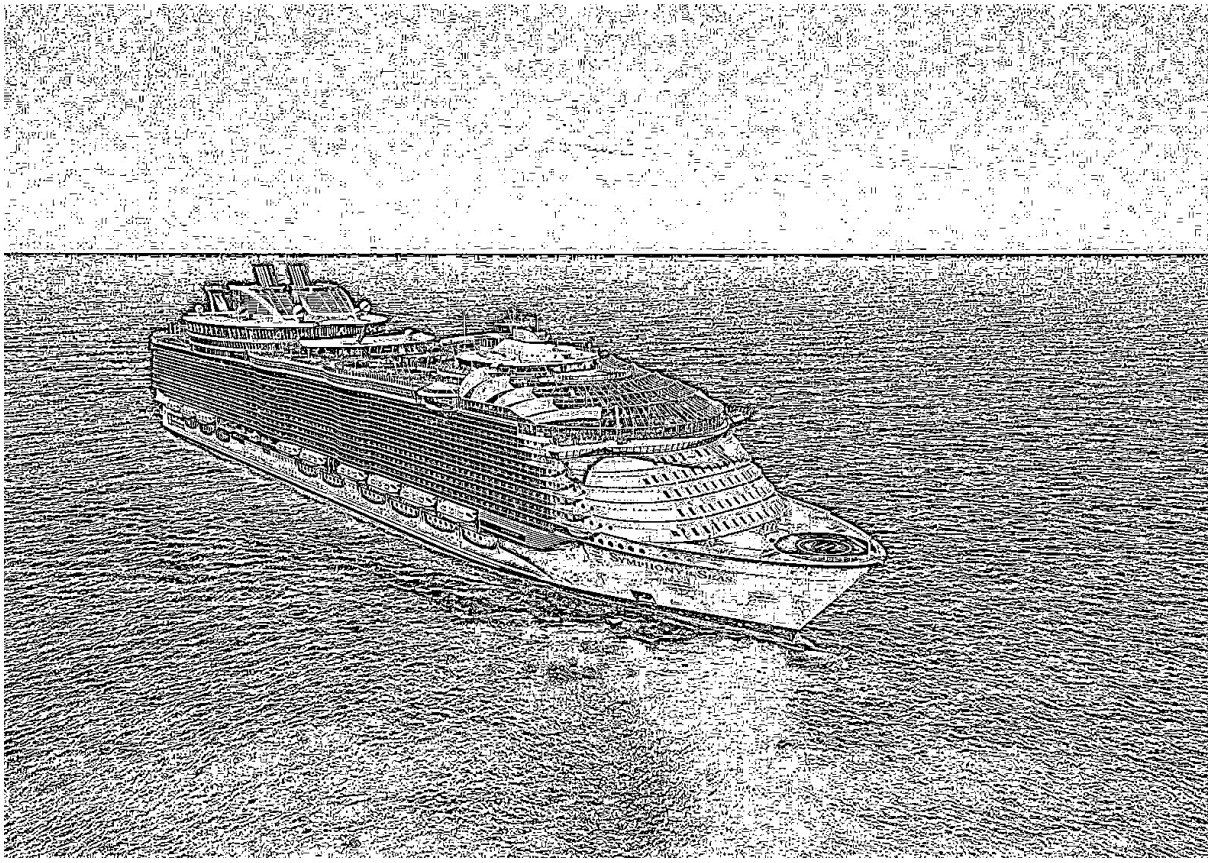
SYMPHONY of the SEAS

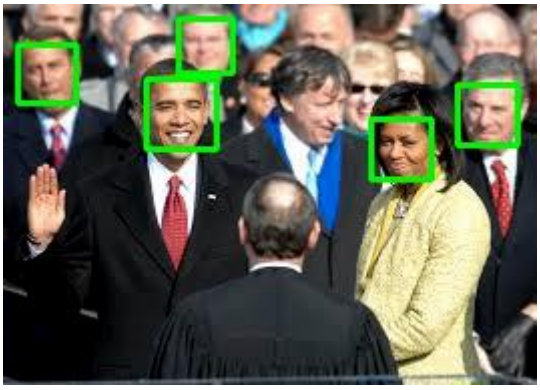
SYMPHONY of the SEAS

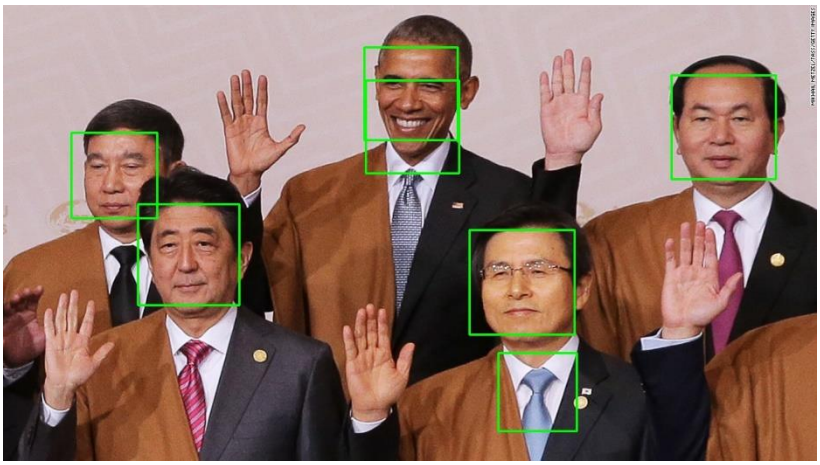




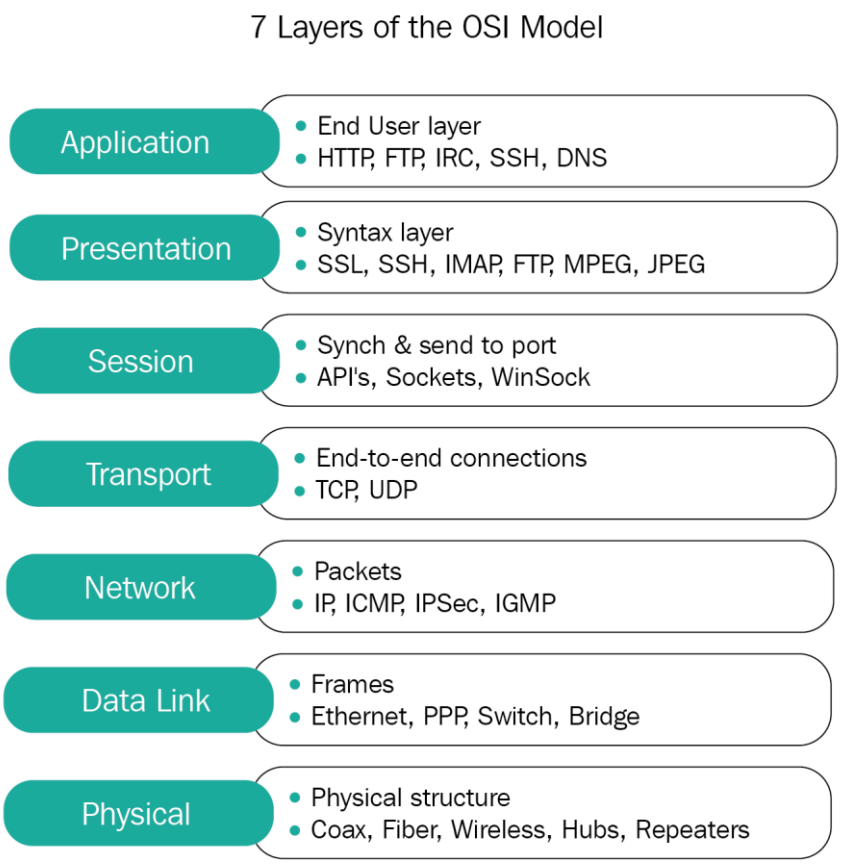


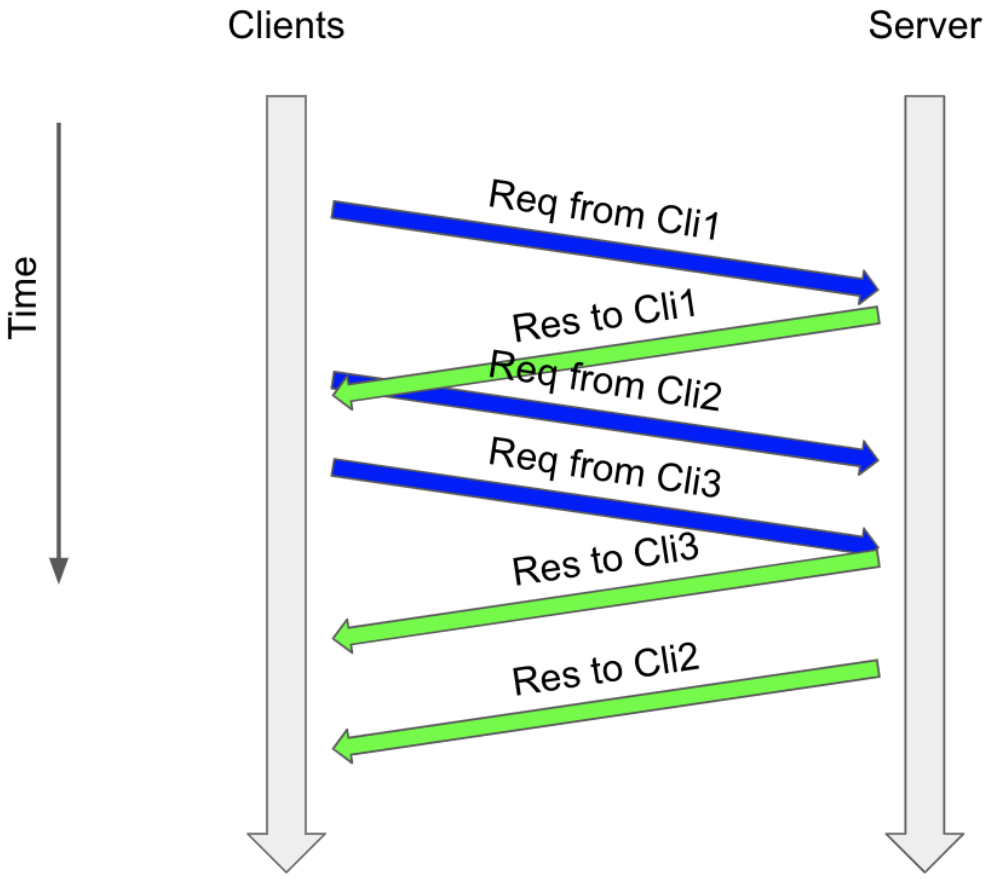


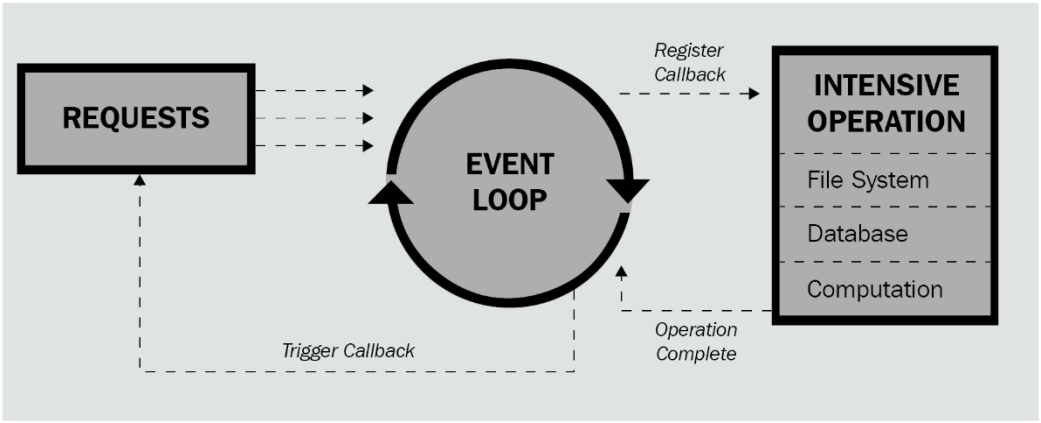




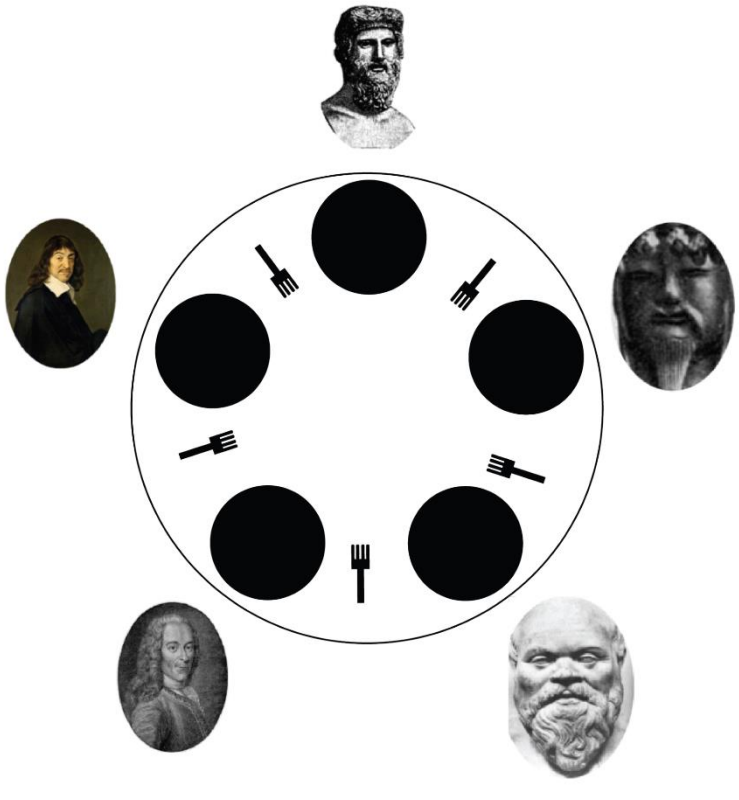
Chapter 11: Building Communication Channels with asyncio

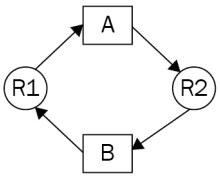


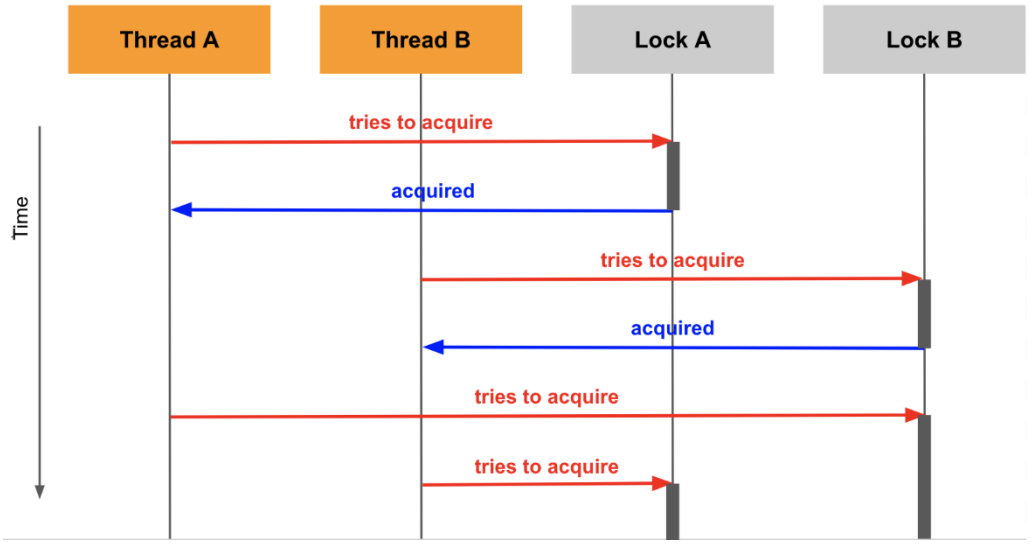

Clients
Server
Time
Req from Cli1
Res to Cli1
Req from Cli2
Req from Cli3
Res to Cli3
Res to Cli2


REQUESTS
EVENT LOOP
Register Callback
INTENSIVE OPERATION
File System
Database
Computation
Operation Complete
Trigger Callback

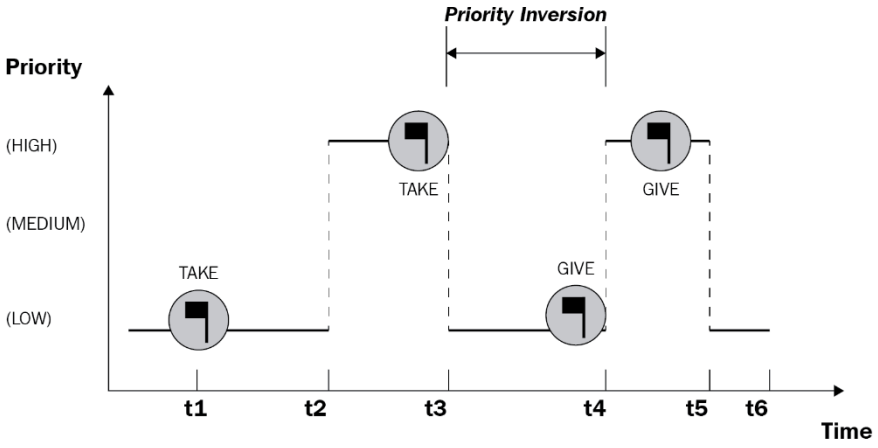
Chapter 12: Deadlocks

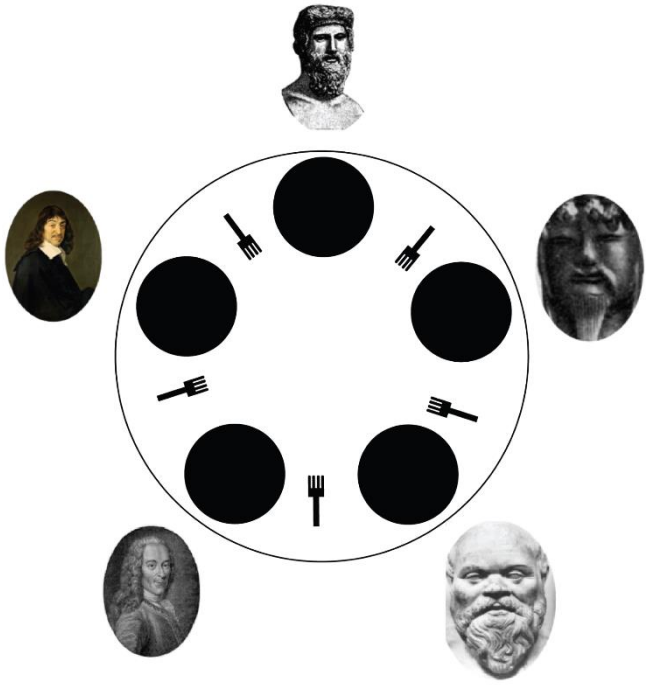




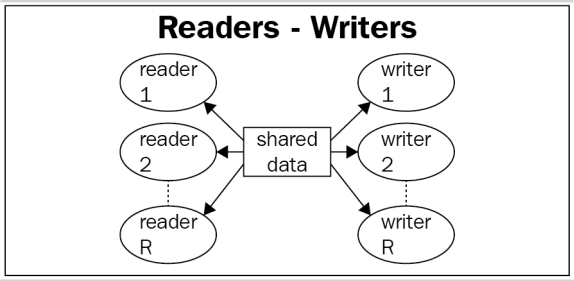


Chapter 13: Starvation



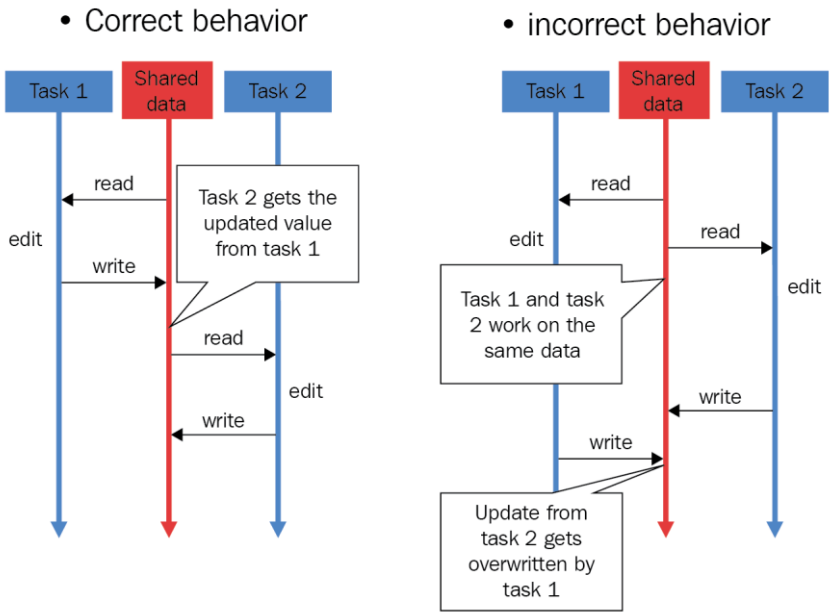


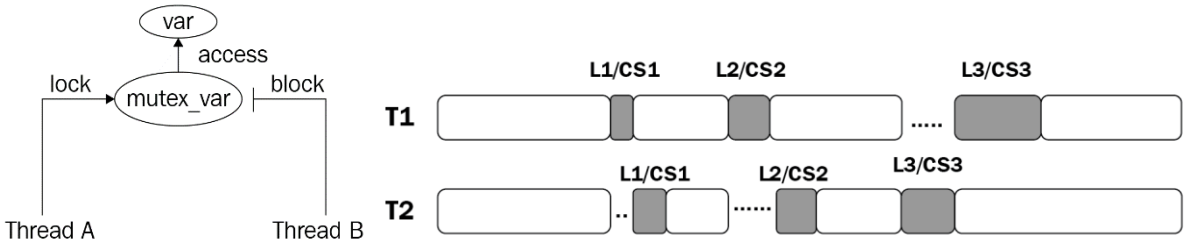
Activity of readers and writers on shared data:

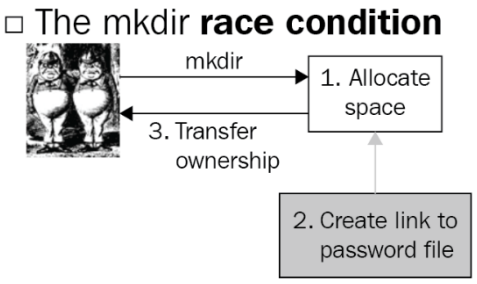


- **Readers - only read the dataset and do *not* perform any updates**
- **Writers - can both read and write**

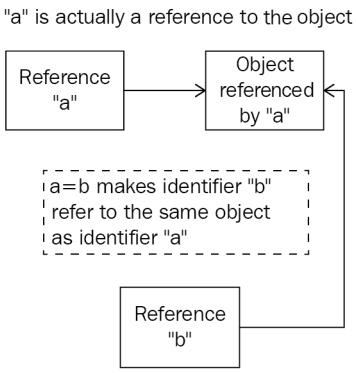
Chapter 14: Race Conditions







Chapter 15: The Global Interpreter Lock



Chapter 16: The Factory Pattern

no images*

Chapter 17: The Builder Pattern

no images

Chapter 18: Other Creational Patterns

no images

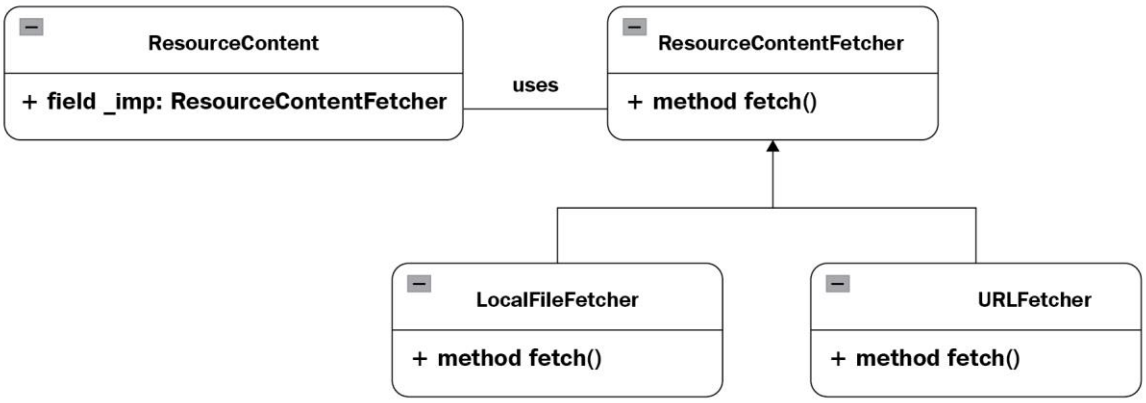
Chapter 19: The Adapter Pattern

no images*

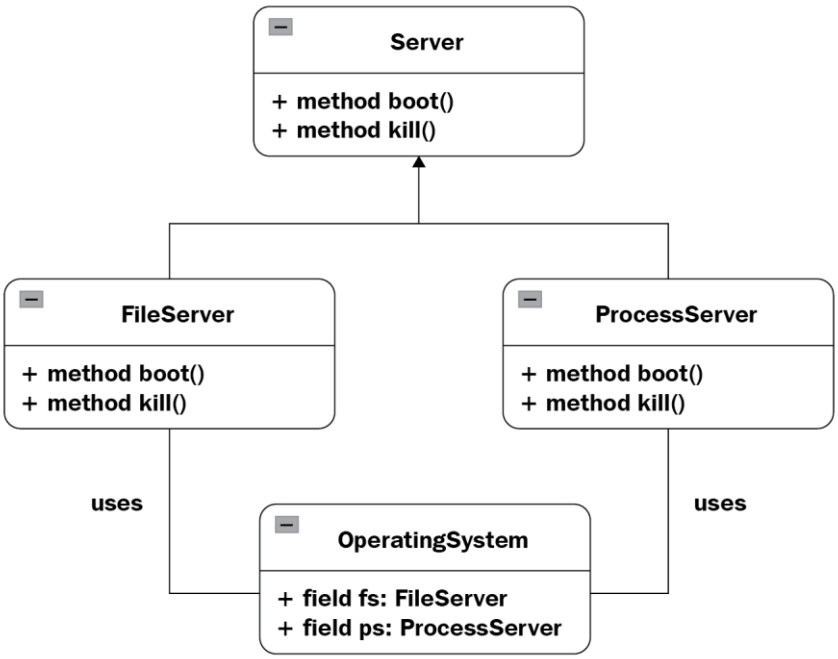
Chapter 20: The Decorator Pattern

no images

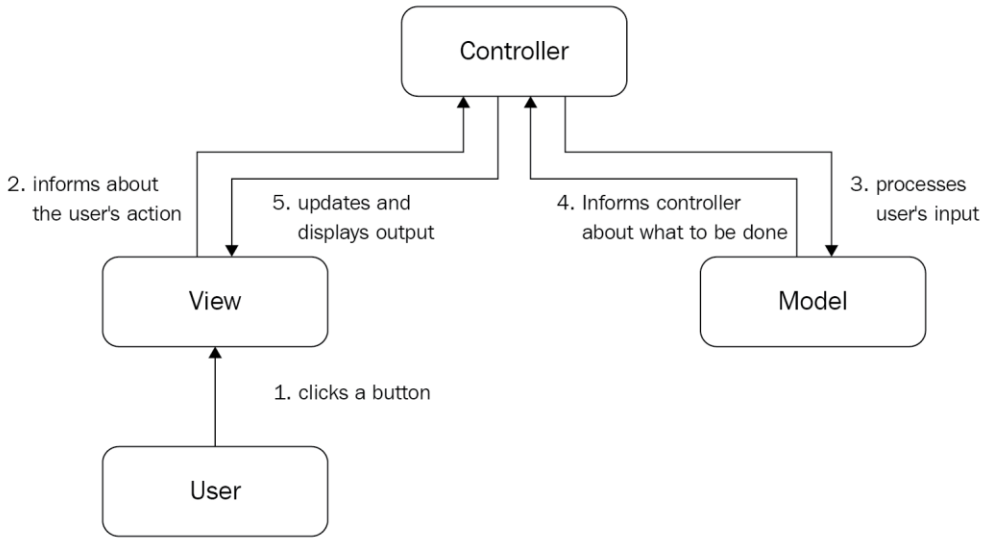
Chapter 21: The Bridge Pattern



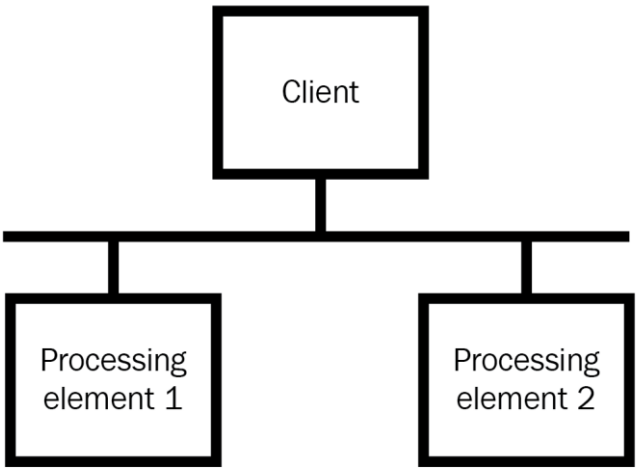
Chapter 22: The Façade Pattern

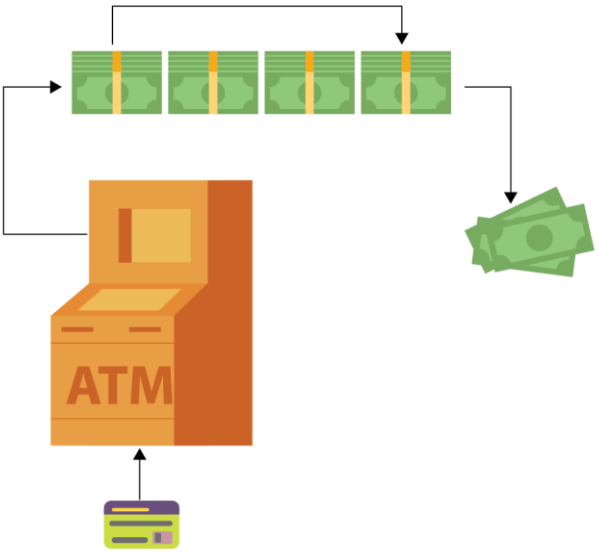


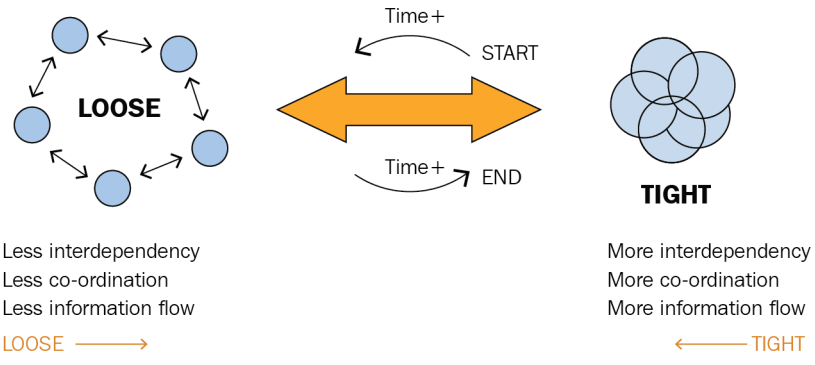
Chapter 23: Other Structural Patterns

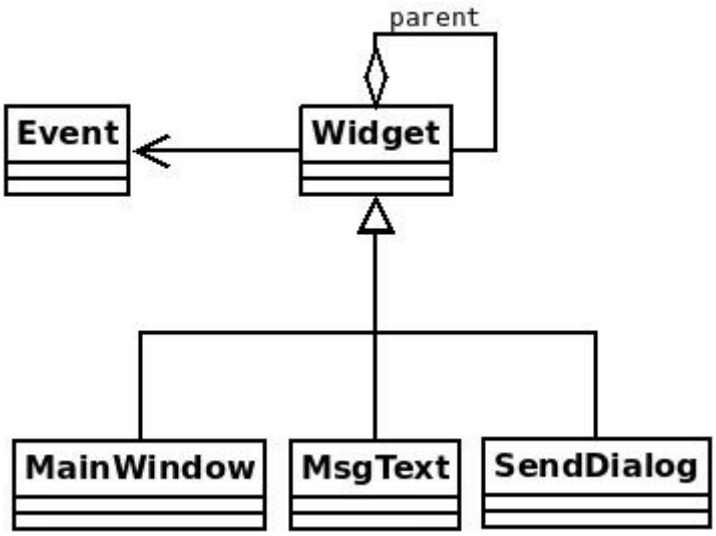


Chapter 24: The Chain of Responsibility Pattern








parent
Event
Widget
MainWindow
MsgText
SendDialog

Chapter 25: The Command Pattern

no images

Chapter 26: The Observer Pattern

no images*