Python Concurrency Ecosystem

1. Parallelism

- Consists of performing multiple operations at the same time.
- Concurrency does not imply parallelism.
- The standard multiprocessing module implements parallelism in Python.

2. Threading

- Concurrent execution model whereby multiple threads take turns executing tasks. Multiple threads run inside a single process and share the same memory space.
- **Threads do not run in parallel**, execution does not occur simultaneously on multiple physical cores.
- Good for IO-bound tasks (it implies waiting).
- The primary downsides to Python threading are *memory safety* and *race conditions*. All child threads of a parent process operate in the same shared memory space.
- The standard threading module implements parallelism in Python.

Python Concurrency Ecosystem

3. Async IO

- Single threaded, single process design that uses cooperative multitasking.

  Cooperative = no OS intervention, each task (coroutine) decides when to give up control.
- Asynchronous functions are able to “pause” while waiting on their ultimate result and let other routines run in the meantime (it gives the look and feel of concurrency).

Python Concurrency Ecosystem

Synchronous vs. Asynchronous (PyTalk 2017):

- Chess tournament: master and 24 players, 30 pair-moves on average per game.
- Chess master 5 sec. per move, opponent 50 sec. per move.
- Synchronous: one game at a time: (50 + 5) * 30 = 1800 sec. (30 min. per game). 30 min. x 24 players = 720 min. (12 hours)
- Asynchronous: The chess master moves from table to table, making one move at each table. One move on each table: 24 x 5 = 120 sec., 30 x 120 sec = 3600 sec. (1 hour in total).