Concurrency:

Concurrency is the ability of a database to allow multiple users to affect multiple transactions. This is one of the main properties that separates a database from other forms of data storage like spreadsheets. The ability to offer **concurrency** is unique to databases .

Concurrency JAVA API:

Java 5 added a new Java package to the Java platform, the **java.util.concurrent** package. This package contains a set of classes that makes it easier to develop concurrent (multithreaded) **applications** in Java. Before this package was added, you would have to program your utility classes yourself.

What is concurrency and parallelism?

Concurrency means multiple tasks which start, run, and complete in overlapping time periods, in no specific order. **Parallelism** is when multiple tasks OR several part of a unique task literally run at the same time, e.g. on a multi-core processor. Remember that **Concurrency and parallelism** are NOT the same thing.

It 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**.. **parallel programming** happens when code is being executed at the same time and each execution is independent of the other.

What is a concurrent system?

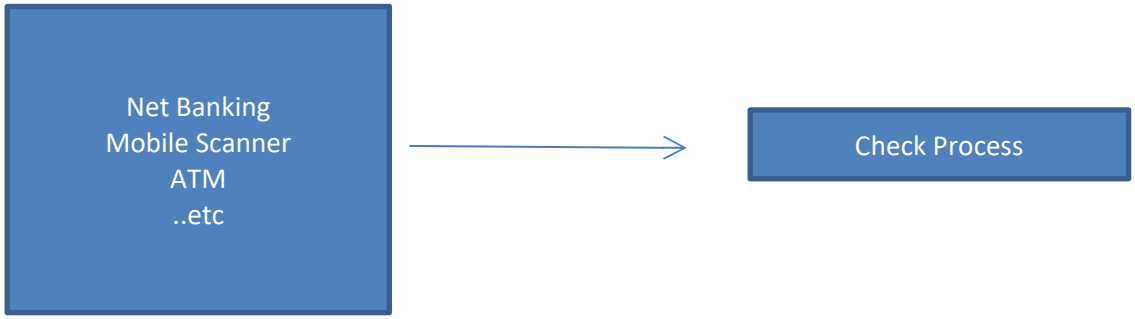
Concurrent processing is a computing model in which multiple processors execute instructions simultaneously for better performance. **Concurrent** means something that happens at the same time as something else.

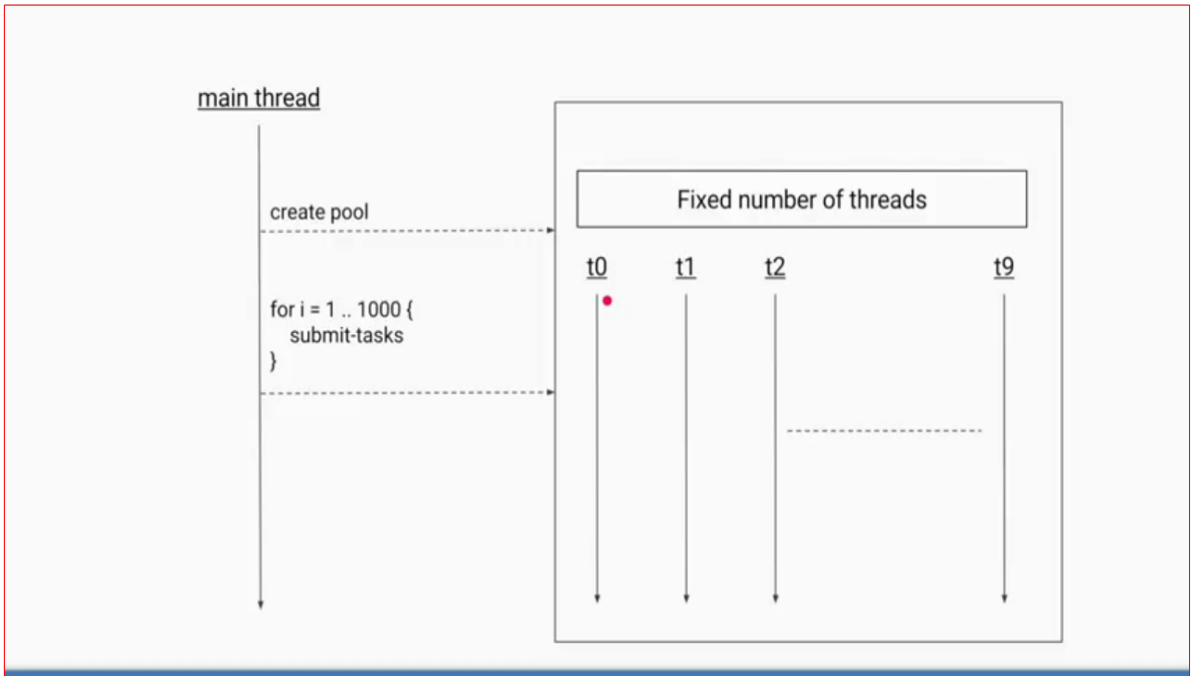
Thread Pool OR Executor Framework :

Traditional Framework :

1- Not Robust
2- Expensive
3- Not proper resource utilization

Assume a Producer Consumer Environment :

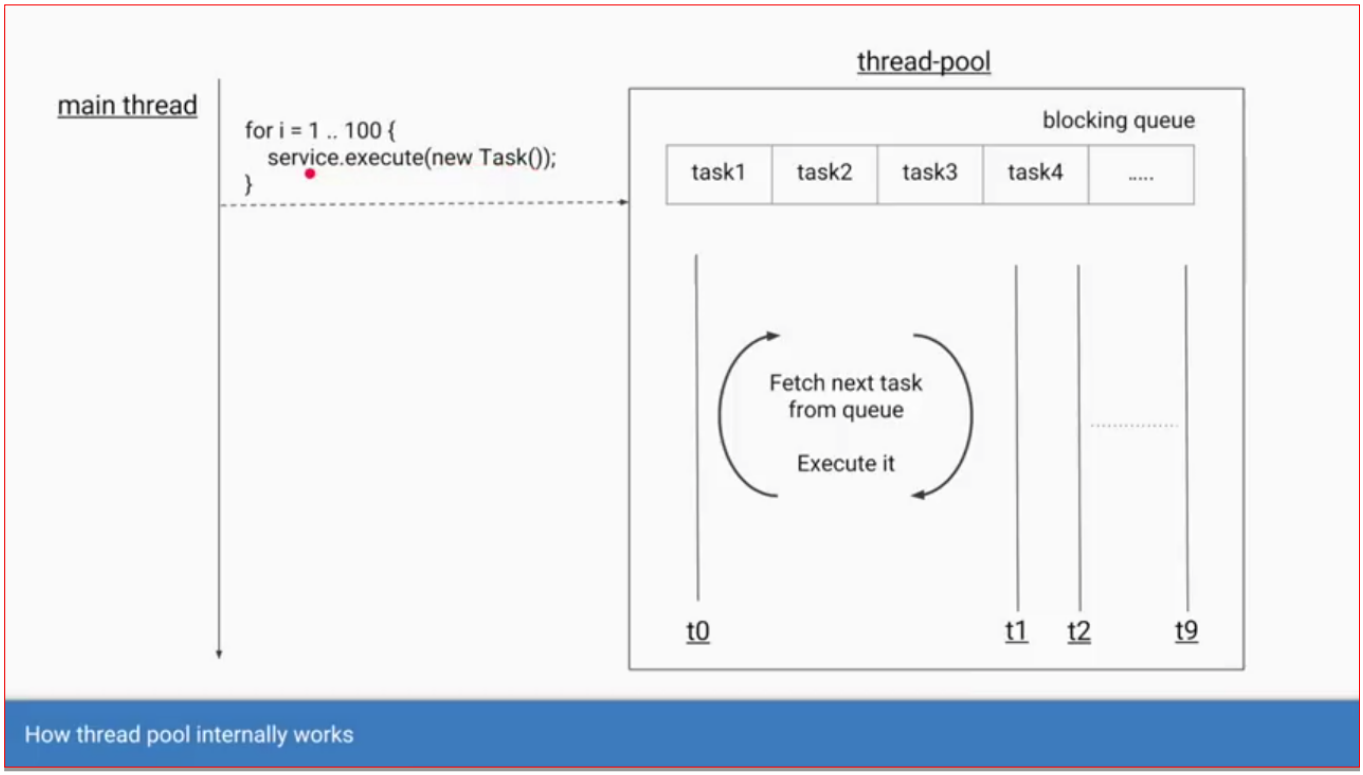




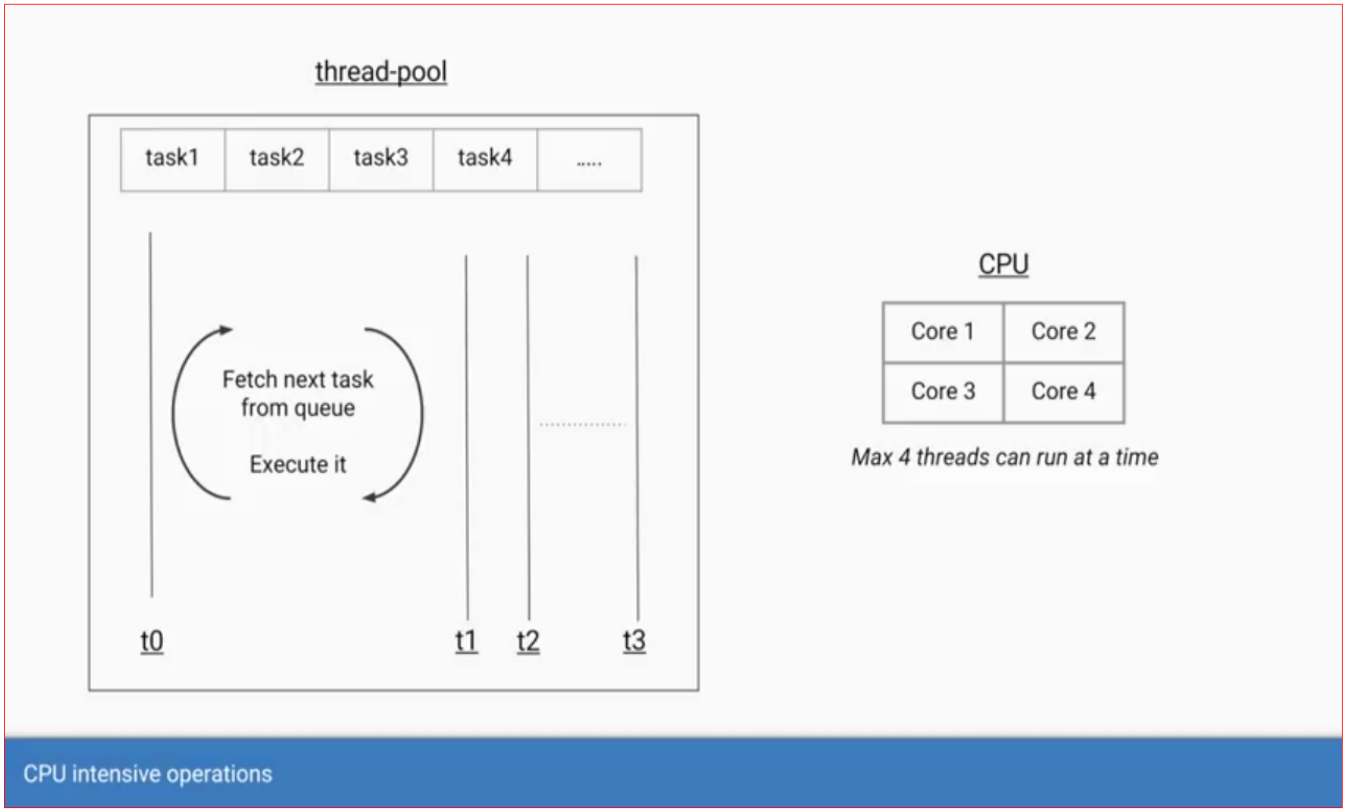
Thread pool

```java
public static void main(String[] args) {

    // create the pool
    ExecutorService service = Executors.newFixedThreadPool( nThreads: 10);

    // submit the tasks for execution
    for (int i = 0; i < 100; i++) {
        service.execute(new Task());
    }
    System.out.println("Thread Name: " + Thread.currentThread().getName());
}

static class Task implements Runnable {
    public void run() {
        System.out.println("Thread Name: " + Thread.currentThread().getName());
    }
}
```

Submit tasks using ThreadPool



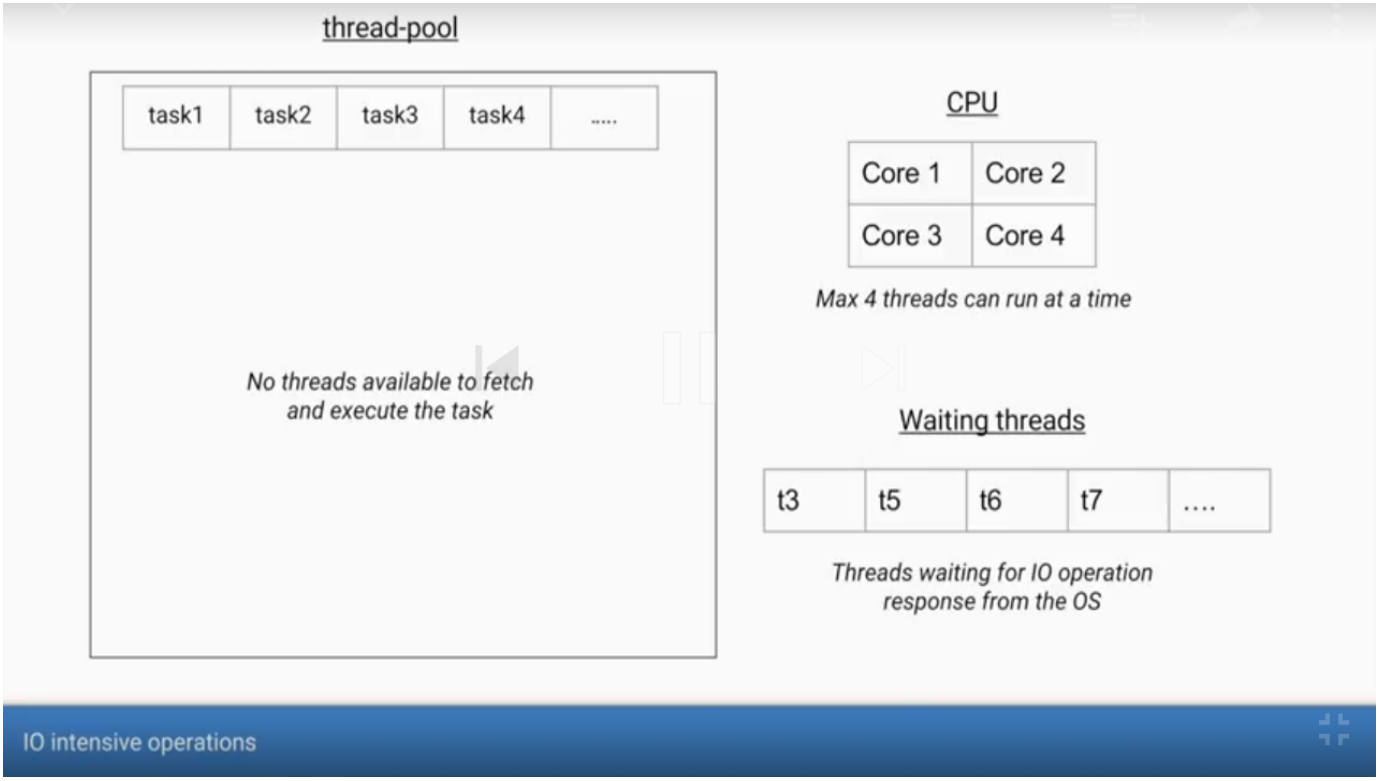
How thread pool internally works



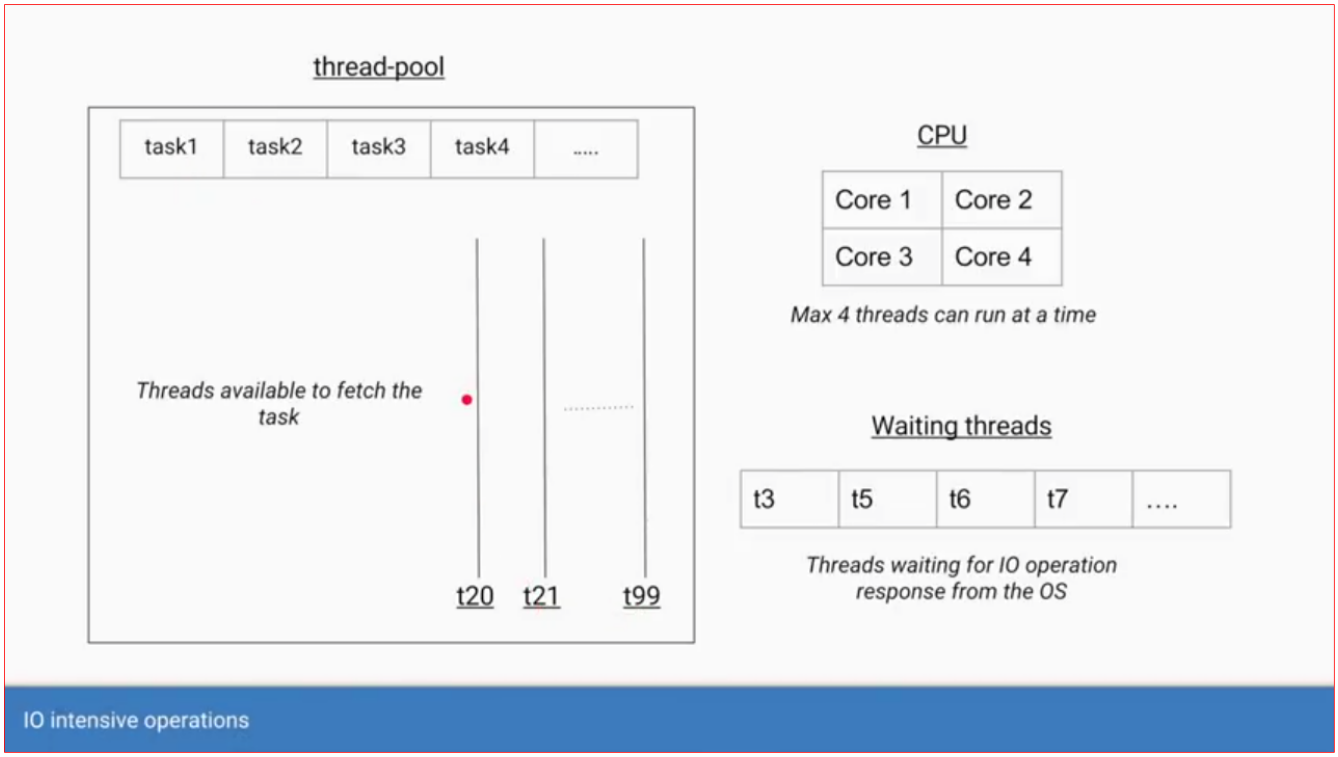
CPU intensive operations

```java
public static void main(String[] args) {

    // get count of available cores
    int coreCount = Runtime.getRuntime().availableProcessors();
    ExecutorService service = Executors.newFixedThreadPool(coreCount);

    // submit the tasks for execution
    for (int i = 0; i < 100; i++) {
        service.execute(new CpuIntensiveTask());
    }
}

static class CpuIntensiveTask implements Runnable {
    public void run() {
        // some CPU intensive operations
    }
}
```

Pool size for CPU intensive Tasks



IO intensive operations



IO intensive operations

```java
public static void main(String[] args) {

    // much higher count for IO tasks
    ExecutorService service = Executors.newFixedThreadPool( nThreads: 100);

    // submit the tasks for execution
    for (int i = 0; i < 100; i++) {
        service.execute(new IOTask());
    }
}

static class IOTask implements Runnable {
    public void run() {
        // some IO operations which will cause thread to block/wait
    }
}
```

Pool size for IO intensive Tasks

Task Type	Ideal pool size	Considerations
CPU intensive	CPU Core count	How many other applications (or other executors/threads) are running on the same CPU.
IO intensive	High	Exact number will depend on rate of task submissions and average task wait time. Too many threads will increase memory consumption too.

Pool size depends on the task type