main thread

```
for i = 1 .. 1000 {
    ti.start()
}
```

1 Java thread = 1 OS thread

t0   t1   t2   t999

expensive

Running 1000s of tasks asynchronously

thread-pool

main thread

```
for i = 1 .. 100 {
    service.execute(new Task());
}
```

blocking queue

| task1 | task2 | task3 | task4 | ..... |

Fetch next task
from queue

Execute it

t0   t1   t2   t9

How thread pool internally works

thread-pool

task1	task2	task3	task4	.....

*No threads available to fetch
and execute the task*

CPU

Core 1	Core 2
Core 3	Core 4

Max 4 threads can run at a time

Waiting threads

t3	t5	t6	t7	....

*Threads waiting for IO operation
response from the OS*

IO intensive operations

---

main-thread

thread-pool

Service.schedule

service.scheduleAtFixedRate

service.scheduleAtFixedDelay

Delay queue

task1	task2	task3	task4	.....

*Schedule the tasks to run based
on time delay (and retrigger for
fixedRate / fixedDelay)*

*Life Cycle: More threads are
created if required.*

t0   t1   t2   t9

Scheduled Thread Pool =

Constructor Parameters

Parameter	Type	Meaning
corePoolSize	int	Minimum/Base size of the pool
maxPoolSize	int	Maximum size of the pool
keepAliveTime + unit	long	Time to keep an idle thread alive (after which it is killed)
workQueue	BlockingQueue	Queue to store the tasks from which threads fetch them
threadFactory	ThreadFactory	The factory to use to create new threads
handler	RejectedExecutionHandler	Callback to use when tasks submitted are rejected

main-thread   thread-pool

Synchronous queue
(can hold only 1 task)

task

```
for i = 1 .. 100 {
    service.execute(new Task());
}
```

If all threads are busy, then
create a new thread for the task
and place it in the pool

Life cycle: If thread is idle for 60
seconds (no task to execute) then
kill the thread

t0   t1   t2   t9   new

Cached Thread Pool

Parameter	FixedThreadPool	CachedThreadPool	ScheduledThreadPool	SingleThreaded
corePoolSize	constructor-arg	0	constructor-arg	1
maxPoolSize	same as corePoolSize	Integer.MAX_VALUE	Integer.MAX_VALUE	1
keepAliveTime	0 seconds	60 seconds	60 seconds	0 seconds

Note: Core pool threads are never killed unless
allowCoreThreadTimeOut(boolean value) is set to true.

Pool Size Changes

Current size

| Core pool size | Current pool size | Max pool size |

Initial size

Threads added / deleted based on
pool type and keepAlive time.

Slide 31   Q&A   Notes   Pointer   EXIT
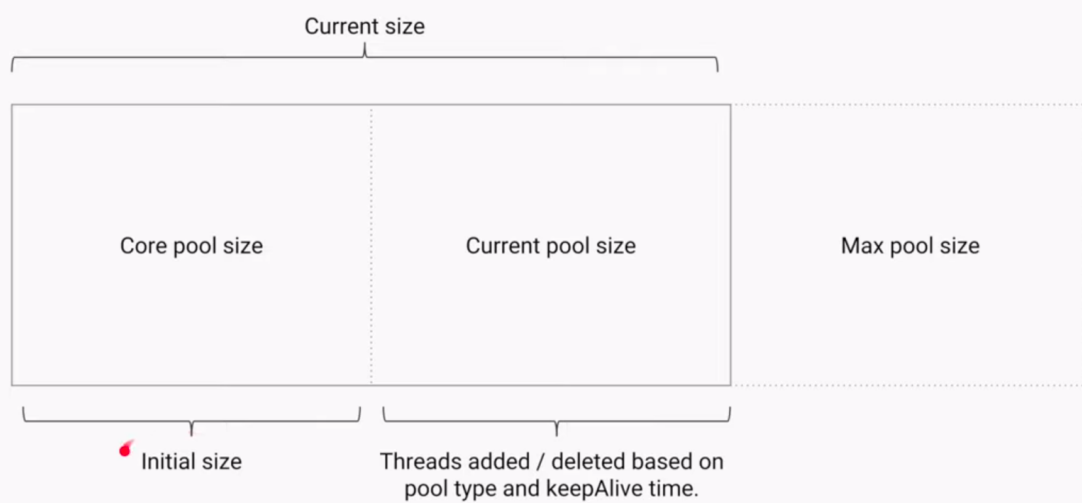
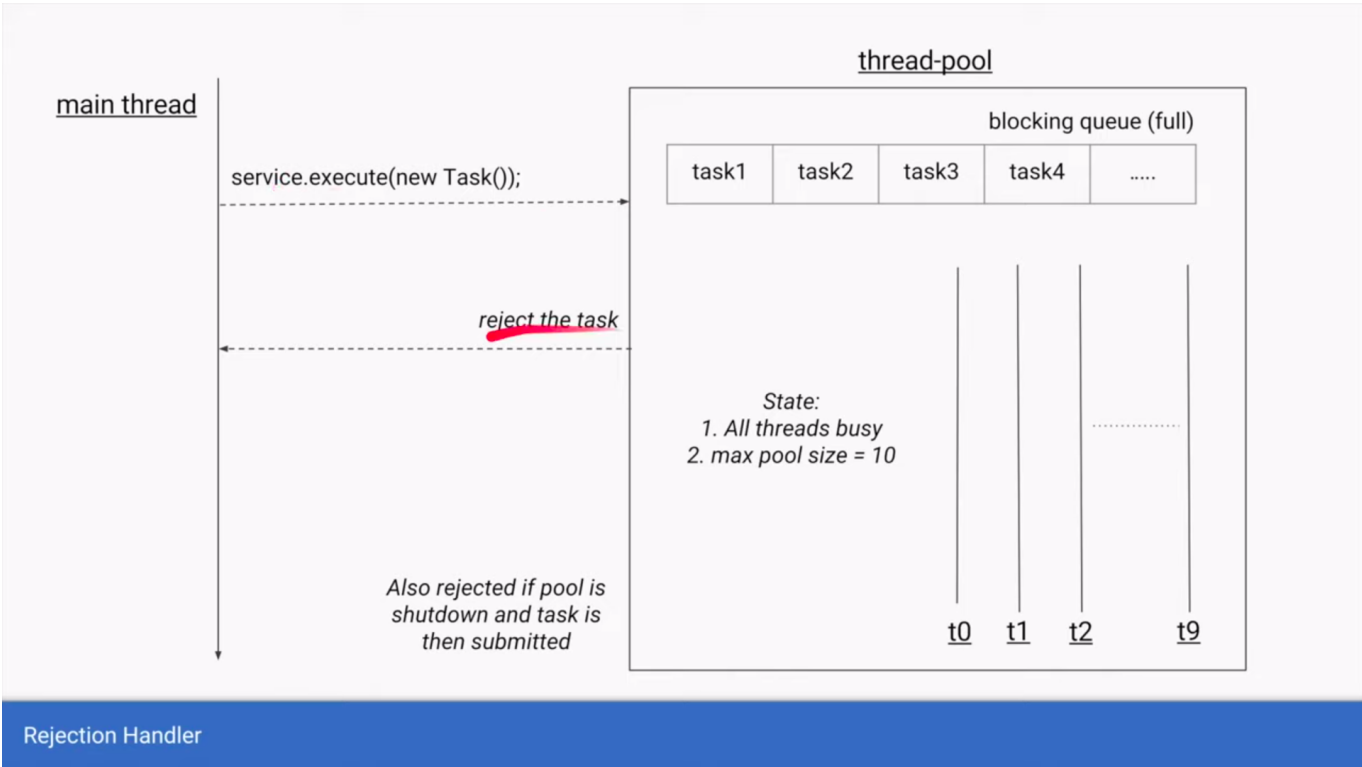| Pool	Queue Type	Why?
FixedThreadPool	LinkedBlockingQueue	Threads are limited, thus unbounded queue to store all tasks.
SingleThreadExecutor	LinkedBlockingQueue	Note: Since queue can never become full, new threads are never created.
CachedThreadPool	SynchronousQueue	Threads are unbounded, thus no need to store the tasks. Synchronous queue is a queue with single slot
ScheduledThreadPool	DelayedWorkQueue	Special queue that deals with schedules/time-delays
Custom	ArrayBlockingQueue	Bounded queue to store the tasks. If queue gets full, new thread is created (as long as count is less than maxPoolSize).

Queue Types
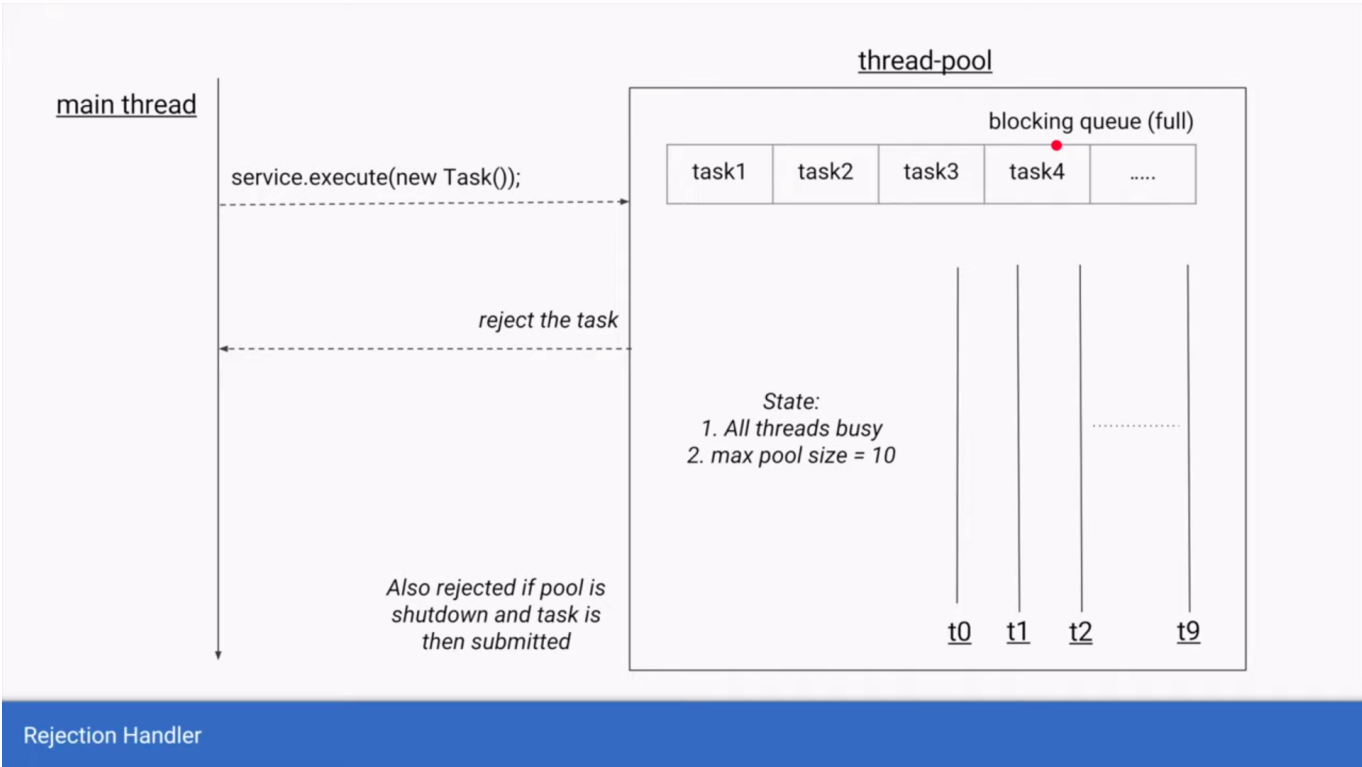
Rejection Handler

Policy	What it means?
AbortPolicy	Submitting new tasks throws RejectedExecutionException (Runtime exception)
DiscardPolicy	Submitting new tasks silently discards it.
DiscardOldestPolicy	Submitting new tasks drops existing oldest task, and new task is added to the queue.
CallerRunsPolicy	Submitting new tasks will execute the task on the caller thread itself. This can create feedback loop where caller thread is busy executing the task and cannot submit new tasks at fast pace.

Rejection Policies

thread-pool

main thread

service.execute(new Task());

reject the task

blocking queue (full)

| task1 | task2 | task3 | task4 | ..... |

State:
1. All threads busy
2. max pool size = 10

Also rejected if pool is
shutdown and task is
then submitted

t0   t1   t2   t9

Rejection Handler

Callable