Core Java Master Course

PART 2

OOPs

Data Structure

Modularity

Design patterns

Concurrency

JVM

Java New Features

Java Stream

Rajeev Gupta

rgupta.mtech@gmail.com

https://www.linkedin.com/in/rajeevguptajavatrainer

rgupta.mtech@gmail.com

Session 8:
Java Concurrency
Session 9:
Design Patterns GOF

rgupta.mtech@gmail.com

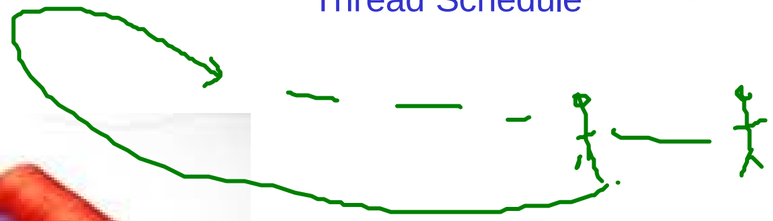
C :(

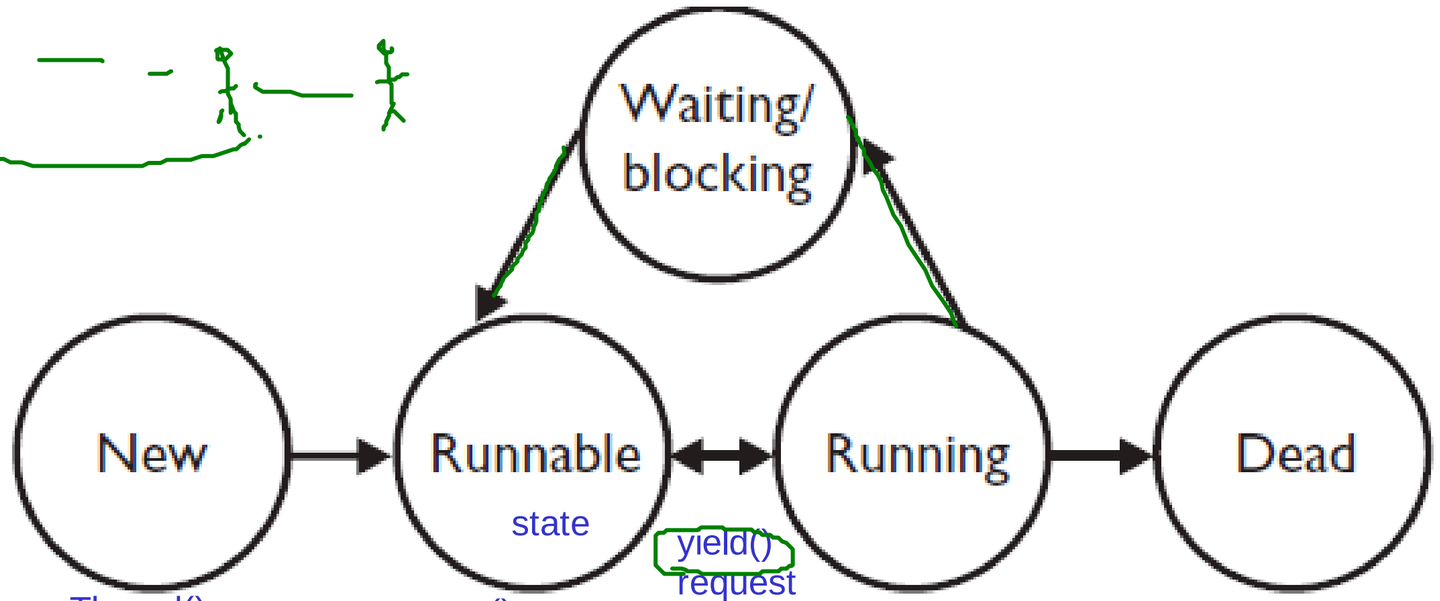
Session 8:
Java Concurrency

Threads depends on OS

OS

Thread Schedule





java

os level therad

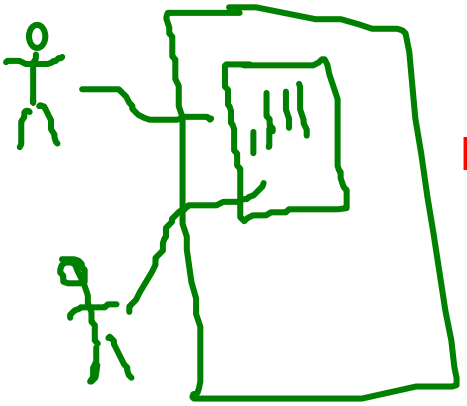
Thread t=new Thread();

t.start()

cpu

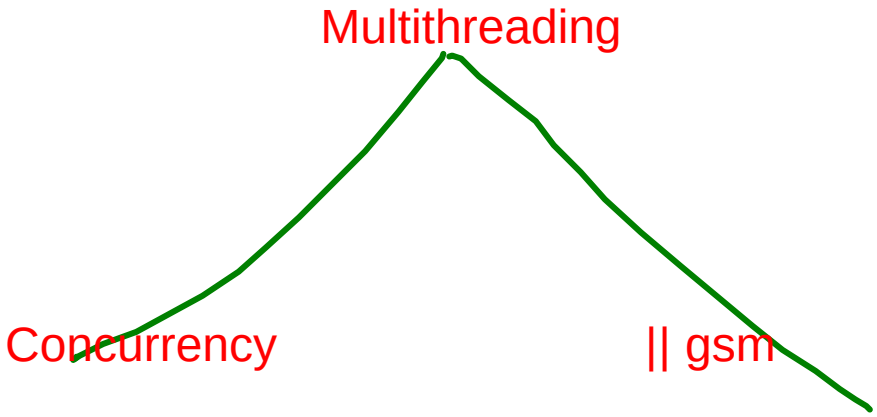
rgupta.mtech@gmail.com

Program vs Process vs Threads



Race condition

a=a+1;



Java Multithreading

* java 1.4 classical threads
  Thread, Runnable
    Synch, wait and notify, thread life cycle

* java 5 java util concurrent package
  ThreadPool ++++

* java 8
  declartive data || processing

java 19: vitual threads*

Truth about concurrency

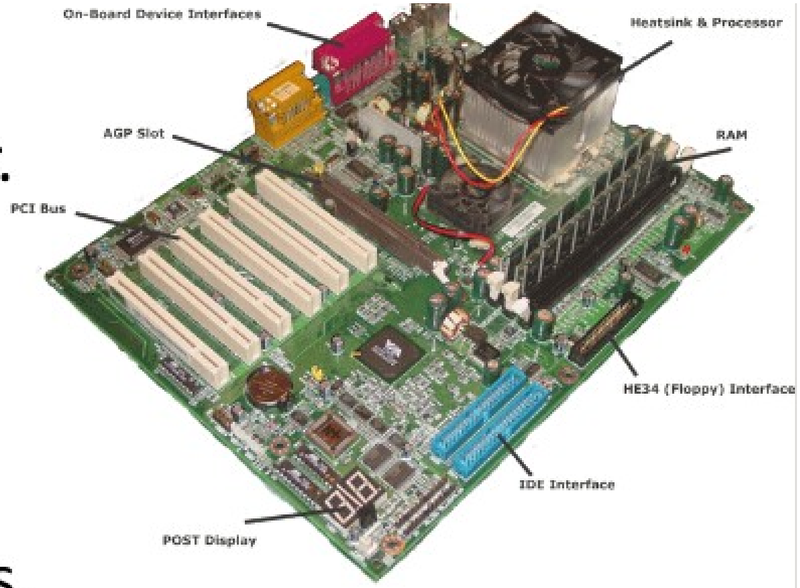
How you designed it

What happens in reality

rgupta.mtech@gmail.com

How concurrency is achived?

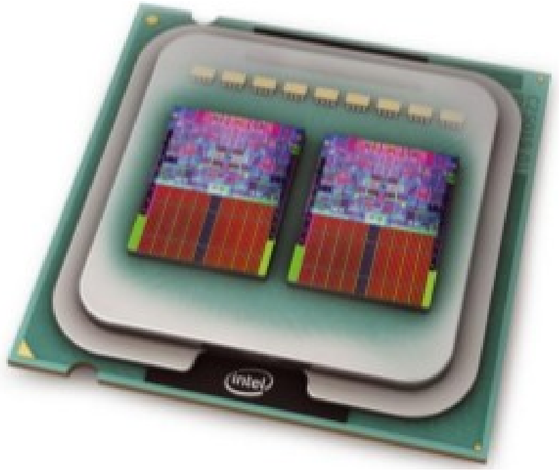
- Single Processor:
  - Has only one actual processor.
  - Concurrent tasks are often multiplexed or multi tasked.
- Multi Processor.
  - Has more than one processors.
  - Concurrent tasks are executed on different processors.

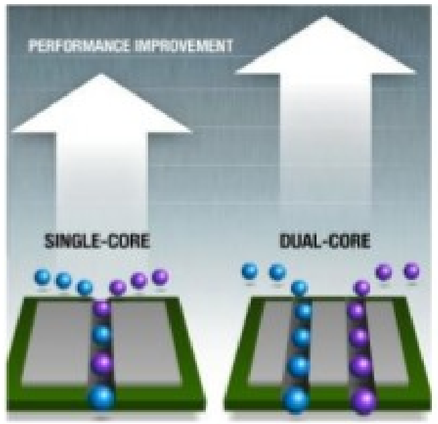


rgupta.mtech@gmail.com

How concurrency is achived?

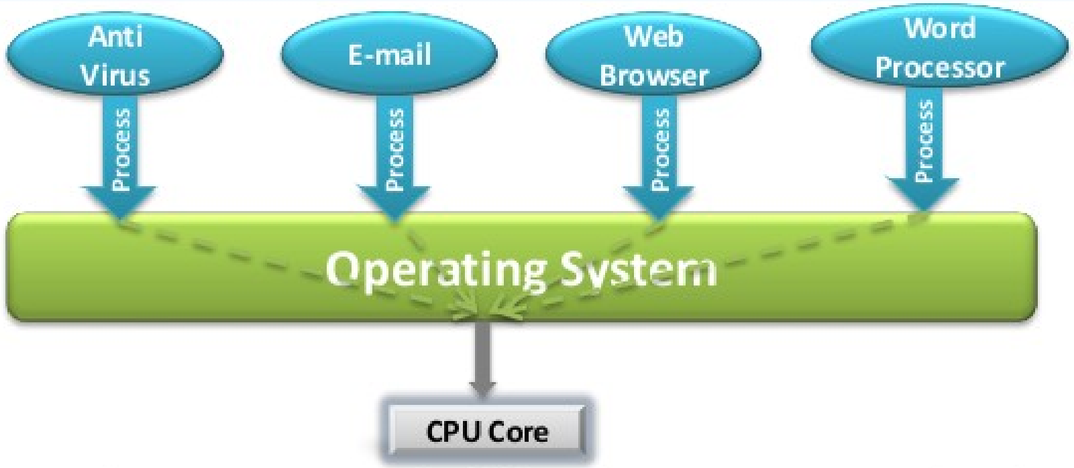
- Both types of systems can have more than one core per processor (Multicore).
- Multicore processors behave the same way as if you had multiple processors



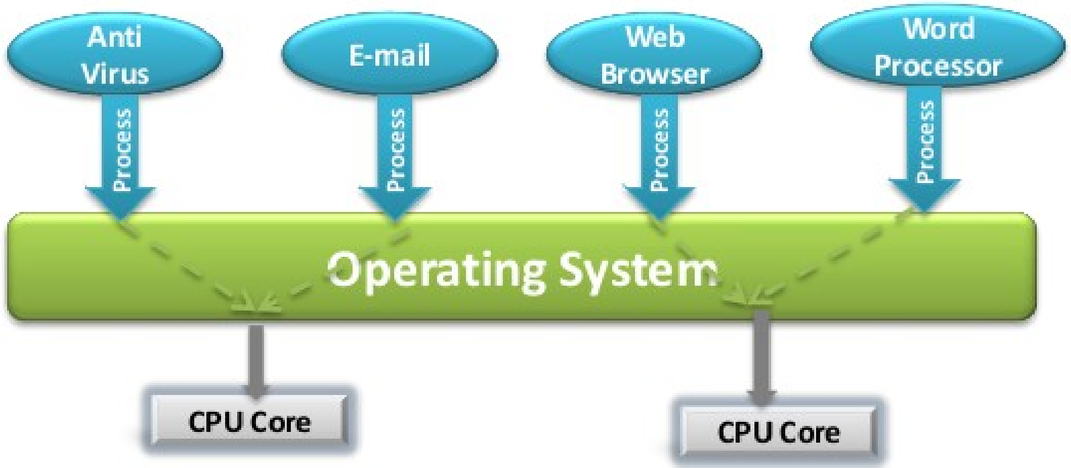


rgupta.mtech@gmail.com

CPU Cores



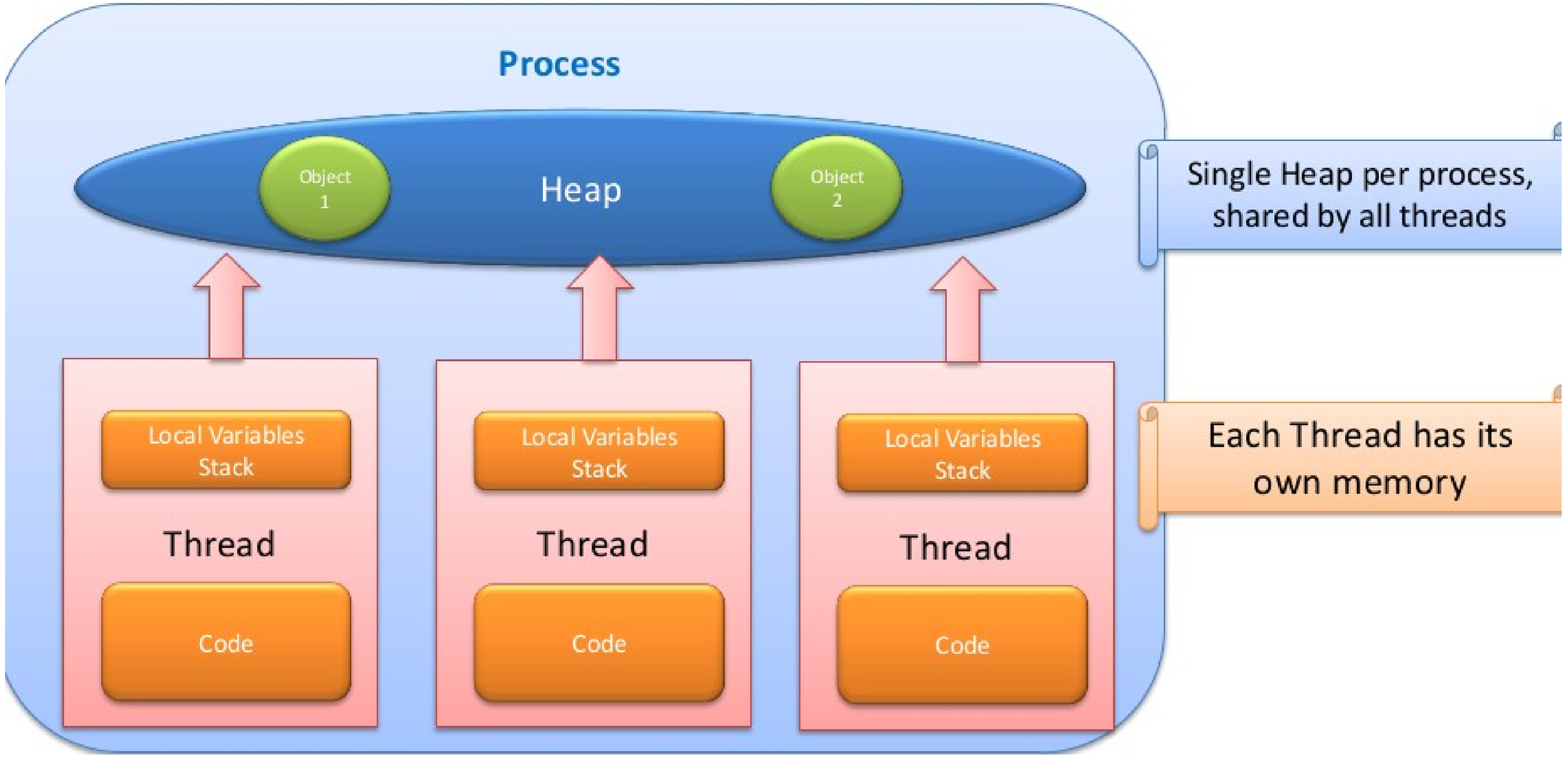
Single-core systems schedule tasks on 1 CPU to multitask



Dual-core systems enable multitasking operating systems to execute two tasks simultaneously

rgupta.mtech@gmail.com

Process and Threads



rgupta.mtech@gmail.com

Background Threads

Nothing little app

```
public class ThreadDumpSample {

    public static void main(String[] args)
        Thread.sleep(100000);
    }
}
```

```
C:\Users\nadeem\Desktop>jps -l
3796 sun.tools.jps.Jps
4756
4276 com.prokarma.app.gtp.thread.ThreadDumpSample

C:\Users\nadeem\Desktop>jstack 4276 > ThreadDump.txt
```

This is how, thread dump is created

Thread Dump of ThreadDumpSample

```
Full thread dump Java HotSpot(TM) 64-Bit Server VM (20.45-b01 mixed mode):
"Low Memory Detector" daemon prio=6 tid=0x000000000064b5800 nid=0x1238 runnable [0x0000000000000000]
   java.lang.Thread.State: RUNNABLE
"C2 CompilerThread1" daemon prio=10 tid=0x000000000064b2000 nid=0x11b0 waiting on condition [0x0000000000000000]
   java.lang.Thread.State: RUNNABLE
"C2 CompilerThread0" daemon prio=10 tid=0x0000000000529000 nid=0xe30 waiting on condition [0x0000000000000000]
   java.lang.Thread.State: RUNNABLE
"Attach Listener" daemon prio=10 tid=0x0000000000527800 nid=0xdbc waiting on condition [0x0000000000000000]
   java.lang.Thread.State: RUNNABLE
"Signal Dispatcher" daemon prio=10 tid=0x000000000064a0800 nid=0x81c runnable [0x0000000000000000]
   java.lang.Thread.State: RUNNABLE
"Finalizer" daemon prio=8 tid=0x0000000000512800 nid=0x804 in Object.wait() [0x000000000645f000]
   java.lang.Thread.State: WAITING (on object monitor)
    at java.lang.Object.wait(Native Method)
    - waiting on <0x00000007d5801300> (a java.lang.ref.ReferenceQueue$Lock)
    at java.lang.ref.ReferenceQueue.remove(ReferenceQueue.java:118)
    - locked <0x00000007d5801300> (a java.lang.ref.ReferenceQueue$Lock)
    at java.lang.ref.ReferenceQueue.remove(ReferenceQueue.java:134)
    at java.lang.ref.Finalizer$FinalizerThread.run(Finalizer.java:171)
"Reference Handler" daemon prio=10 tid=0x0000000000509800 nid=0xf30 in Object.wait() [0x000000000635f000]
   java.lang.Thread.State: WAITING (on object monitor)
    at java.lang.Object.wait(Native Method)
    - waiting on <0x00000007d5801fd8> (a java.lang.ref.Reference$Lock)
    at java.lang.Object.wait(Object.java:485)
    at java.lang.ref.Reference$ReferenceHandler.run(Reference.java:116)
    - locked <0x00000007d5801fd8> (a java.lang.ref.Reference$Lock)
"main" prio=6 tid=0x000000000060b000 nid=0x1368 waiting on condition [0x000000000254f000]
   java.lang.Thread.State: TIMED_WAITING (sleeping)
    at java.lang.Thread.sleep(Native Method)
    at com.prokarma.app.gtp.thread.ThreadDumpSample.main(ThreadDumpSample.java:6)
"VM Thread" prio=10 tid=0x0000000000501000 nid=0x13e4 runnable
"GC task thread#0 (ParallelGC)" prio=6 tid=0x000000000045f000 nid=0x1104 runnable
"GC task thread#1 (ParallelGC)" prio=6 tid=0x0000000000460800 nid=0x4a4 runnable
"VM Periodic Task Thread" prio=10 tid=0x000000000064cc000 nid=0x1398 wait
JNI global references: 883
```

rgupta.mtech@gmail.com

Basic Operations on Threads

```java
public class BasicThreadOperations {

    public static void main(String[] args) {

        // Get Control of the current thread
        Thread currentThread = Thread.currentThread();
        //Print Thread information
        System.out.println("Thread Name : " + currentThread.getName());
        System.out.println("Thread Group : " + currentThread.getThreadGroup());
        System.out.println("Thread Priority : " + currentThread.getPriority());
        System.out.println("Thread is Daemon : " + currentThread.isDaemon());
        System.out.println("Thread State : " + currentThread.getState());
        //Update current thread details
        currentThread.setName("The Main Thread");
        currentThread.setPriority(6);

        System.out.println("\nNew Thread Name : " + currentThread.getName());
        System.out.println("New Thread Priority : " + currentThread.getPriority());
        System.out.println();

        for (int i = 0; i < 6; i++) {
            System.out.println("Current value of i = " + i);
            System.out.println("Going to Sleep for 1 second");
            try {
                Thread.sleep(1000);
            } catch (InterruptedException e) {
                System.out.println("Somebody interrupted " + currentThread + " for un
            }
            System.out.println("Woke up");
        }
        System.out.println("Going to Die...don't stop me.");
    }
}
```

```
Thread Name : main
Thread Group : java.lang.ThreadGroup[name=main,maxpri=1
Thread Priority : 5
Thread is Daemon : false
Thread State : RUNNABLE

New Thread Name : The Main Thread
New Thread Priority : 6

Current value of i = 0
Going to Sleep for 1 second
Woke up
Current value of i = 1
Going to Sleep for 1 second
Woke up
Current value of i = 2
Going to Sleep for 1 second
Woke up
Current value of i = 3
Going to Sleep for 1 second
Woke up
Current value of i = 4
Going to Sleep for 1 second
Woke up
Current value of i = 5
Going to Sleep for 1 second
Woke up
Going to Die...don't stop me.
```

rgupta.mtech@gmail.com

Let’s clarify some terms

Concurrency ≠ Parallelism

Multithreading ≠Parallelism

rgupta.mtech@gmail.com

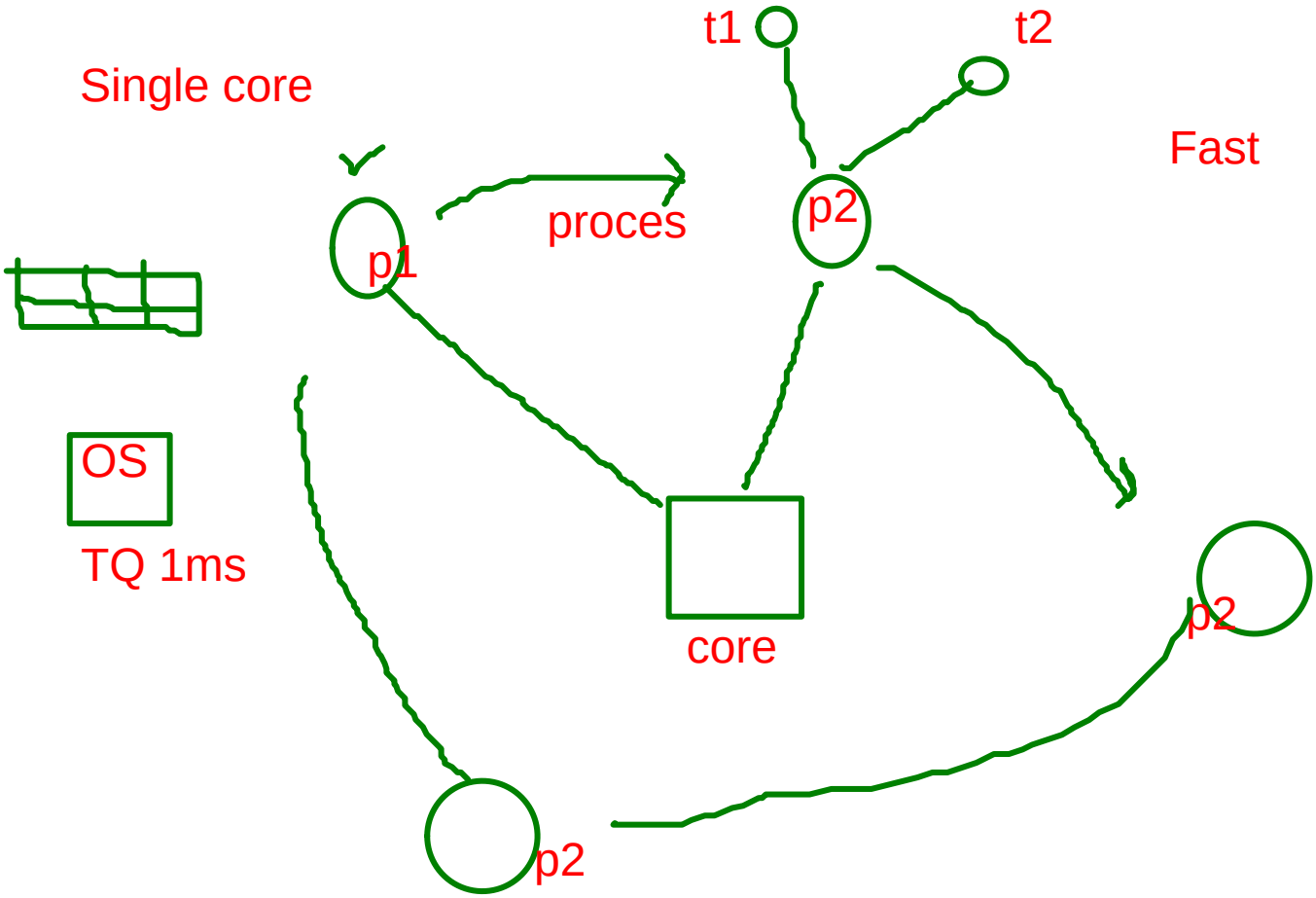
Concurrency vs Parallelism

Concurrency: A condition that exists when at least two threads are making progress

Parallelism: A condition that arises when at least two threads are executing simultaneously

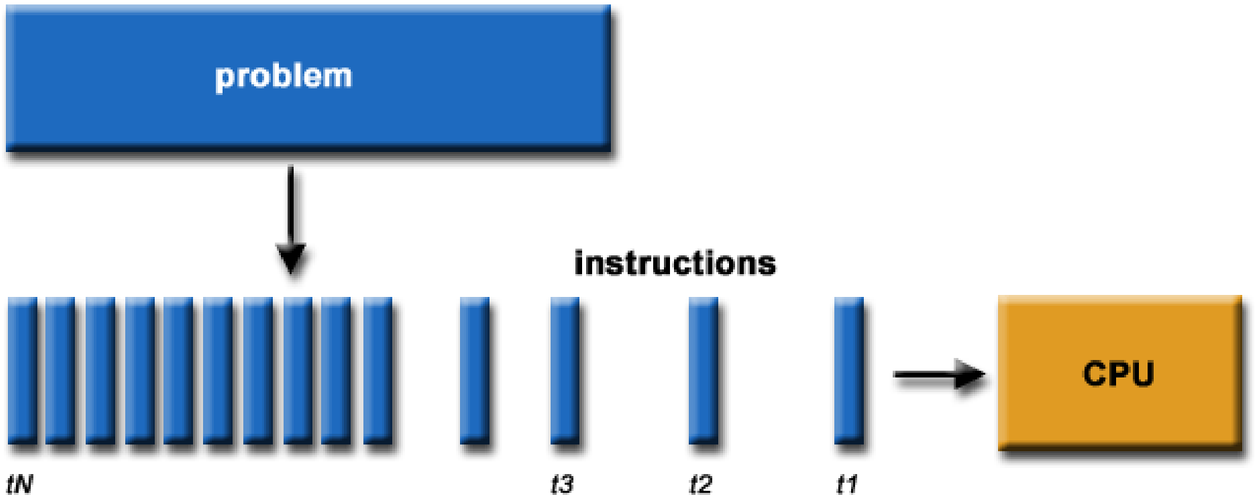
Multithreading: Allows access to two or more threads. Execution occurs in more than one thread of control, using parallel or concurrent processing

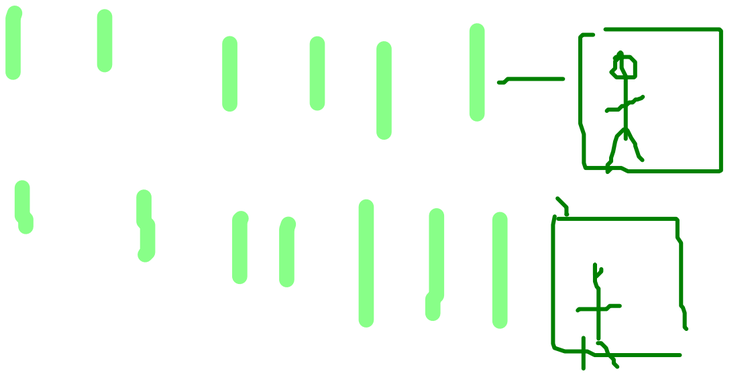
rgupta.mtech@gmail.com

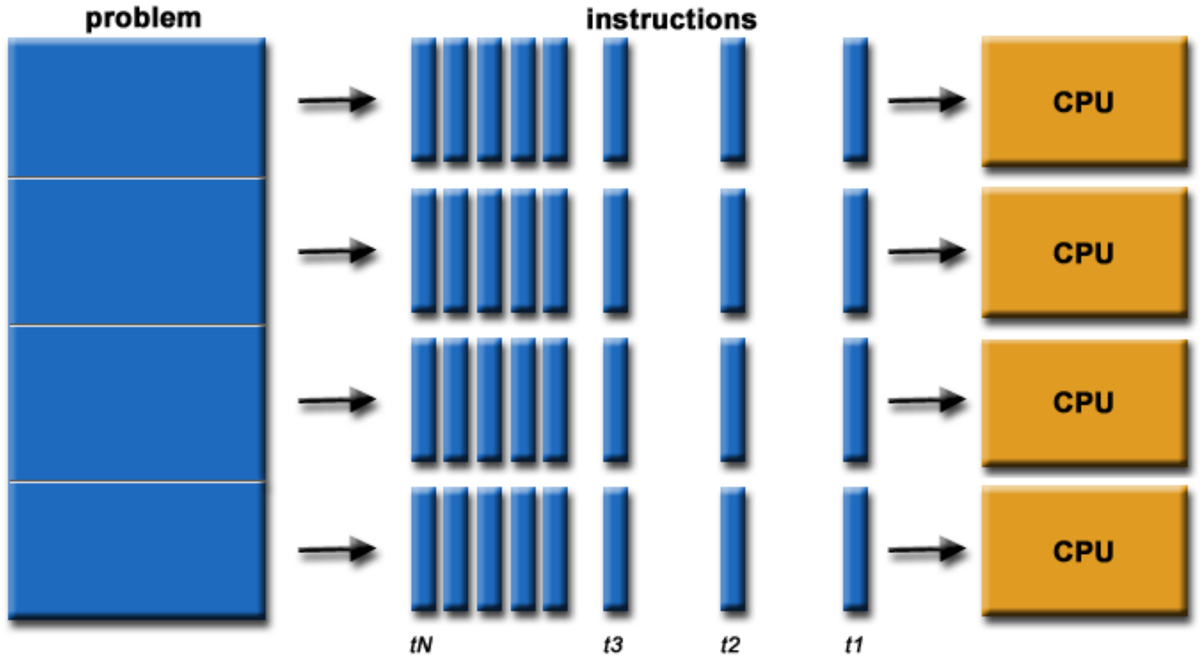

Single core
t1
t2
Fast
proces
p1
p2
OS
TQ 1ms
core
p2
p2

Process
IO
CPU

Concurrency vs Parallelism

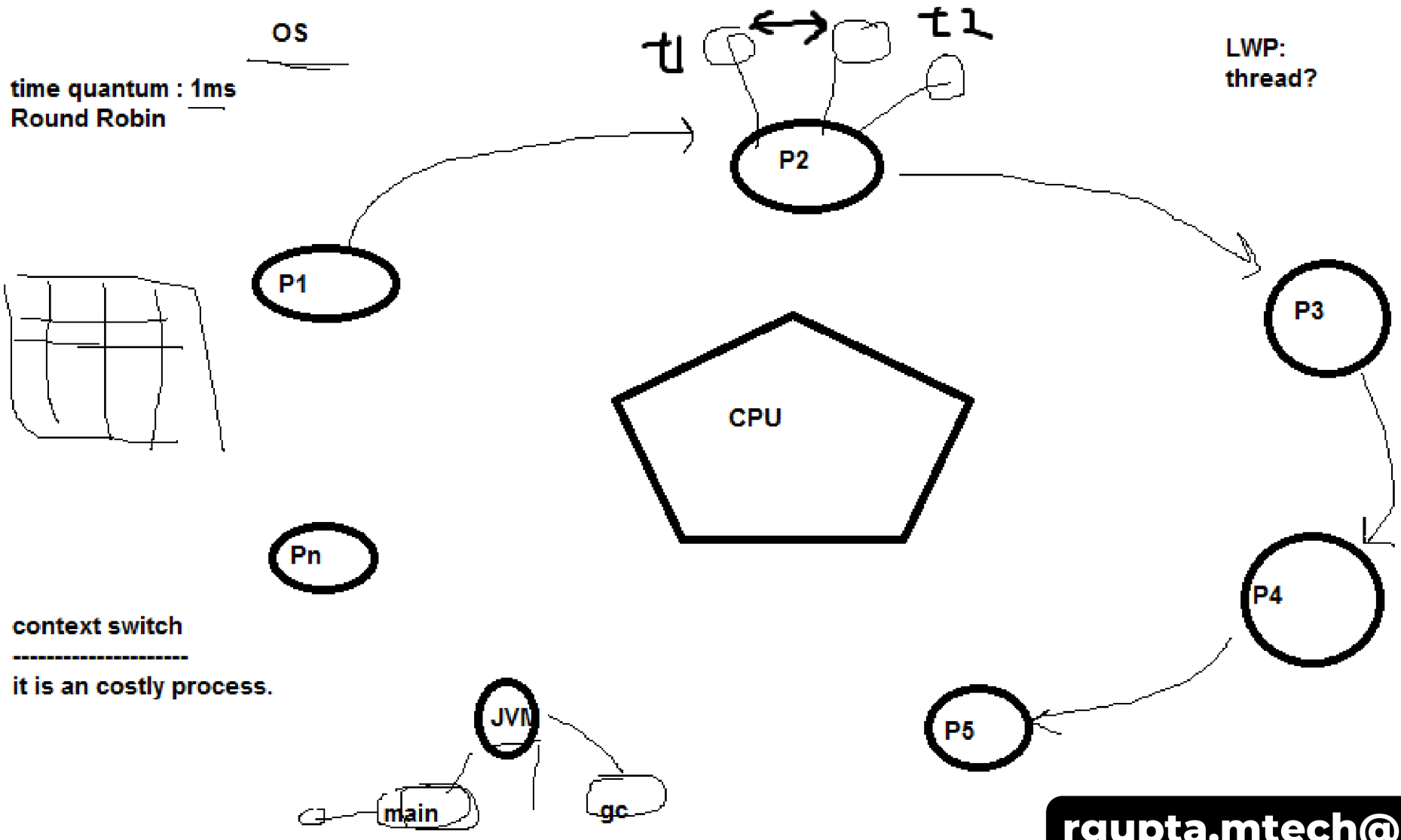






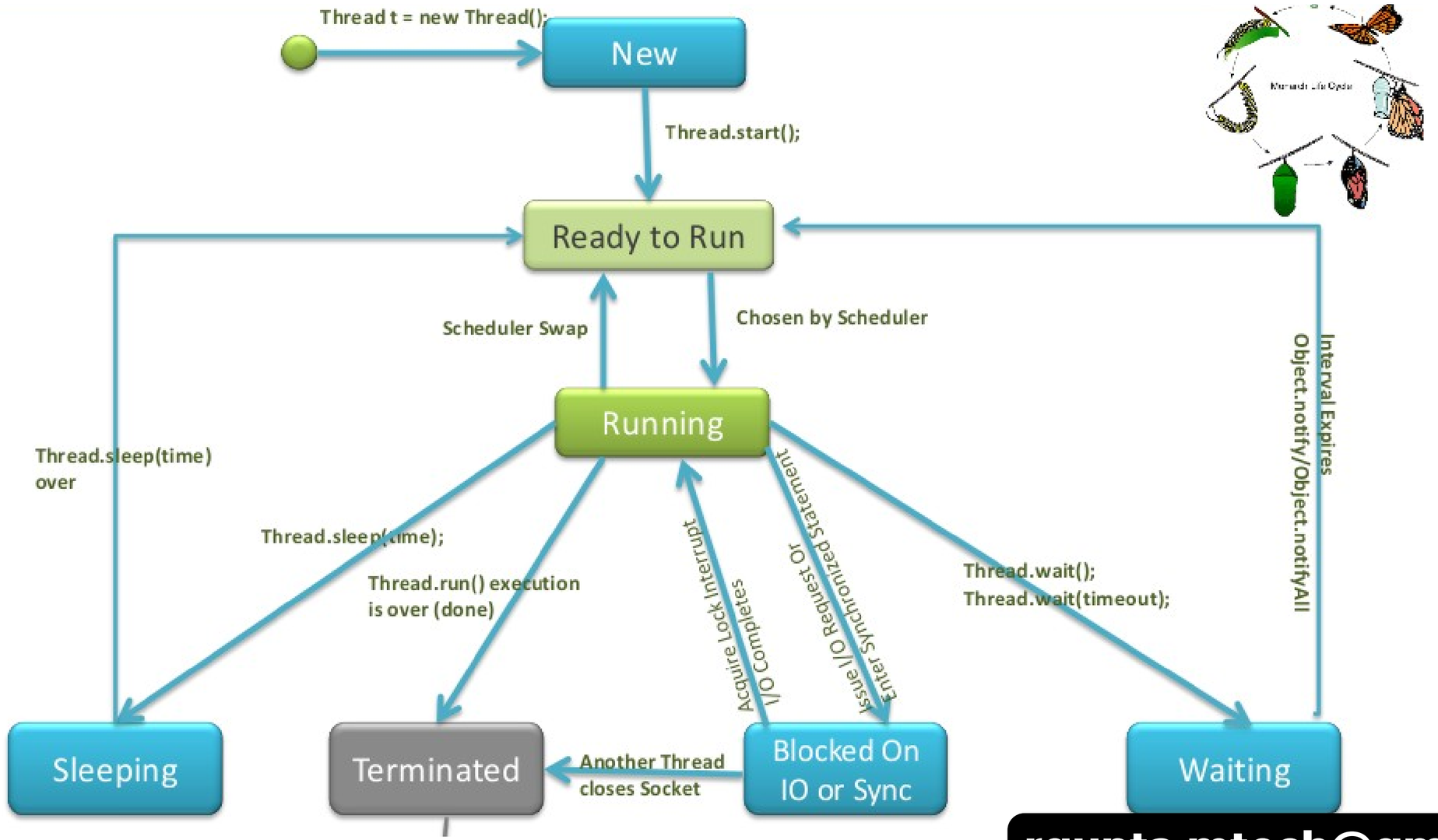
rgupta.mtech@gmail.com

What is threads? LWP



rgupta.mtech@gmail.com

Thread LifeCycle



rgupta.mtech@gmail.com

Important Methds of Thread class

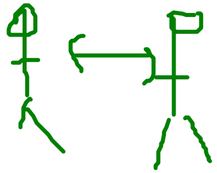
- **start()**
  - Makes the Thread Ready to run
- **isAlive()**
  - A Thread is alive if it has been started and not died.
- **sleep(milliseconds)**
  - Sleep for number of milliseconds.
- **isInterrupted()**
  - Tests whether this thread has been interrupted.
- **Interrupt()**
  - Indicate to a Thread that we want to finish. If the thread is blocked in a method that responds to interrupts, an InterruptedException will be thrown in the other thread, otherwise the interrupt status is set.
- **join()**
  - Wait for this thread to die.
- **yield()**
  - Causes the currently executing thread object to temporarily pause and allow other threads to execute.

rgupta.mtech@gmail.com

sleep() yield() and wait()

- sleep(n) says ***"I'm done with my time slice, and please don't give me another one for at least n milliseconds."*** The OS doesn't even try to schedule the sleeping thread until requested time has passed.
- yield() says ***"I'm done with my time slice, but I still have work to do."*** The OS is free to immediately give the thread another time slice, or to give some other thread or process the CPU the yielding thread just gave up.
- .wait() says ***"I'm done with my time slice. Don't give me another time slice until someone calls notify()."*** As with sleep(), the OS won't even try to schedule your task unless someone calls notify() (or one of a few other wakeup scenarios occurs).

rgupta.mtech@gmail.com



wait() vs sleep()

Called from Synchronized block	No Such requirement
Monitor is released	Monitor is not released
Awake when notify() and notifyAll() method is called on the monitor which is being waited on.	Not awake when notify() or notifyAll() method is called, it can be interrupted.
not a static method	static method
wait() is generally used on condition	sleep() method is simply used to put your thread on sleep.
Can get *spurious wakeups* from wait (i.e. the thread which is waiting resumes for no apparent reason). We Should always wait whilst spinning on some condition , ex : *synchronized {* *while (!condition) monitor.wait();* *}*	This is deterministic.
Releases the lock on the object that wait() is called on	Thread does *not* release the locks it holds

rgupta.mtech@gmail.com

volatile

- What is the expected output?
- What is the problem here?

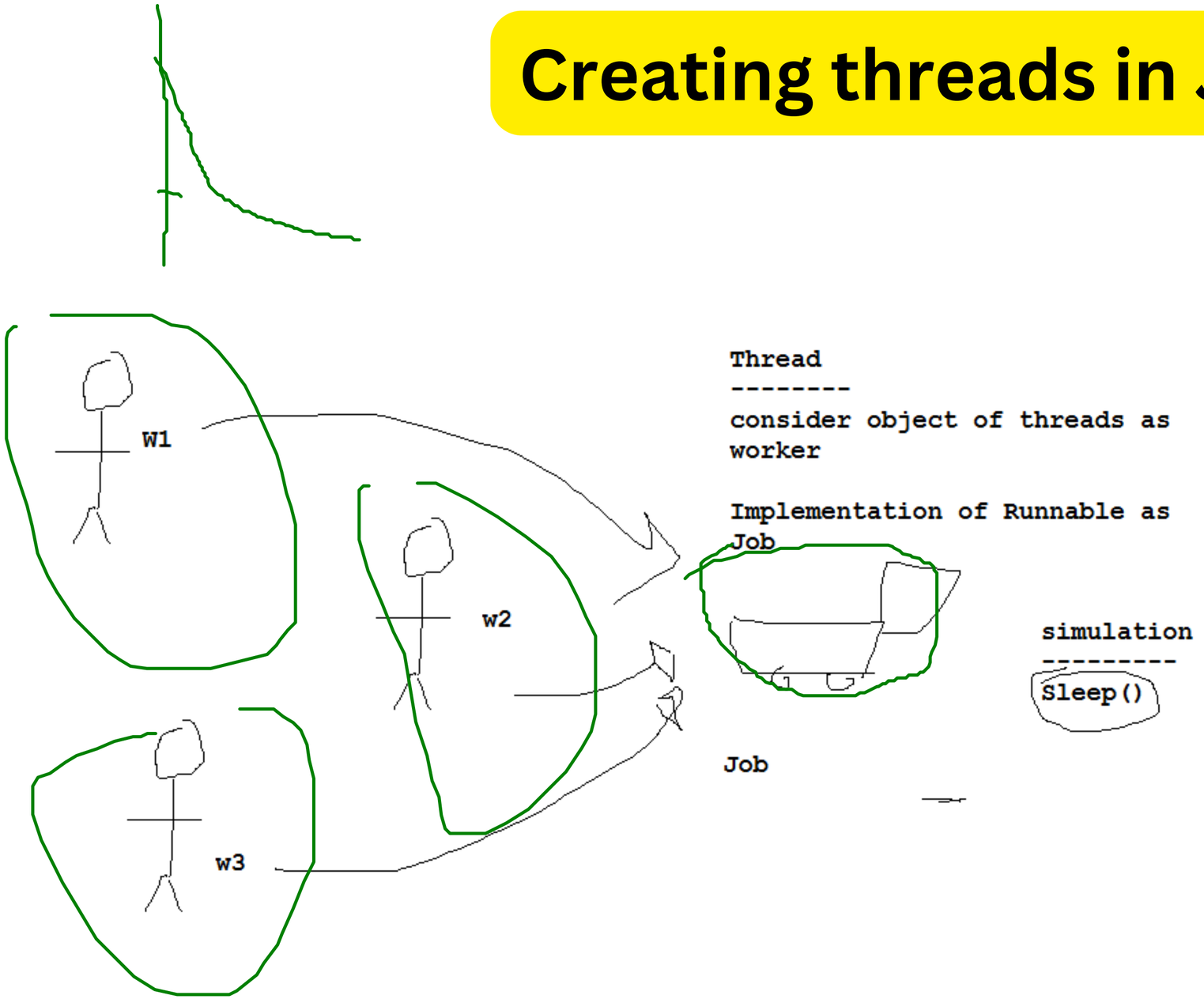
```
public class RaceCondition {

    private static boolean done;

    public static void main(String[] args) throws InterruptedException {

        new Thread(new Runnable() {
            public void run() {
                int i = 0;
                while(!done) { i++; }
                System.out.println("Done! [" + Thread.currentThread().getName() + "]");

            }
        }).start();

        TimeUnit.SECONDS.sleep(1);
        done = true;
        System.out.println("flag done set to true [" + Thread.currentThread().getName() + "]");
    }
}
```

- The program just prints (in windows 7 x64 bit)

  "flag done set to true [main]" and stuck for ever.

rgupta.mtech@gmail.com

Creating threads in Java?



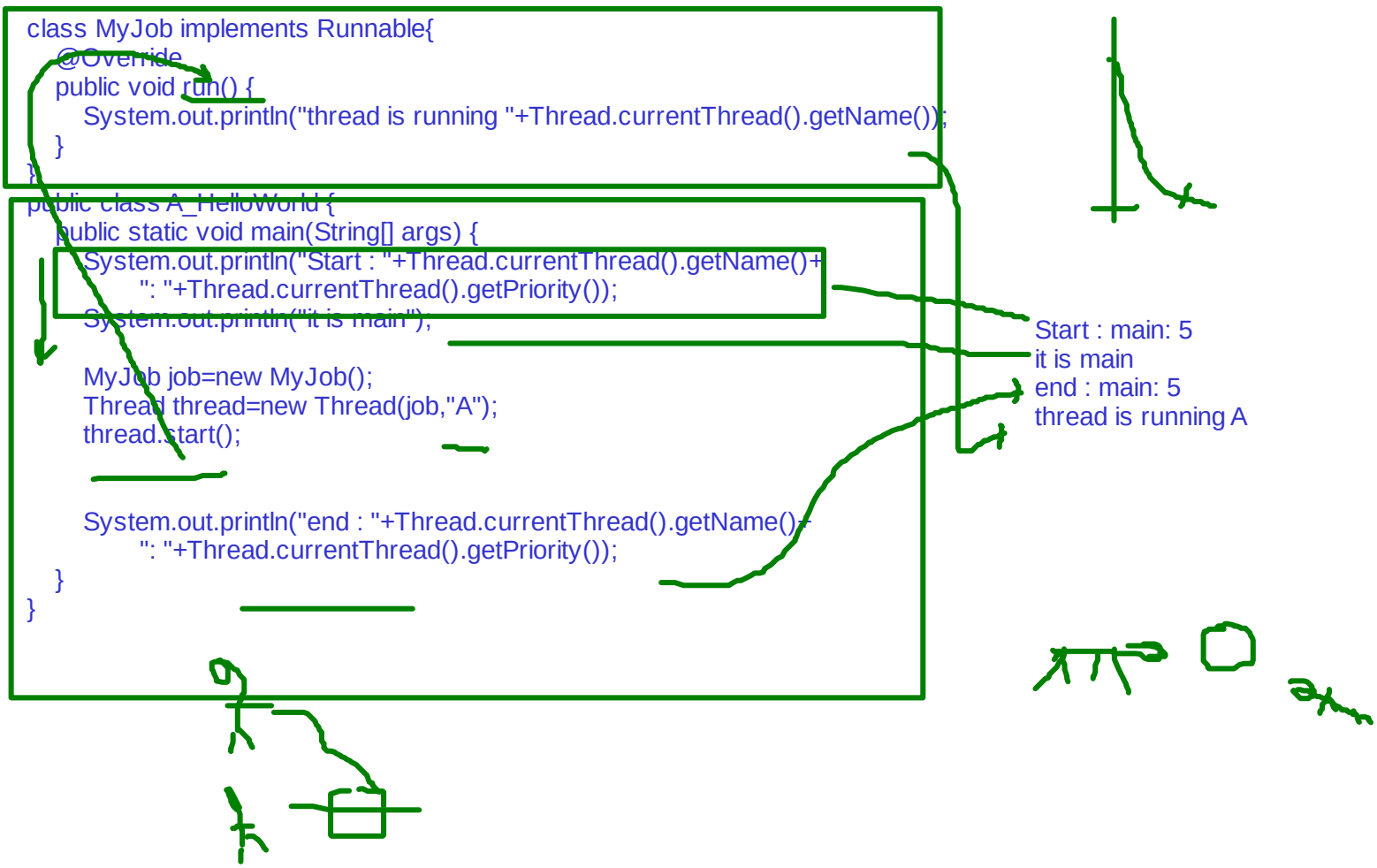

```java
class Job implements Runnable{

    public void run() {
        // TODO Auto-generated method stub
    }

}
```

```java
class MyThread extends Thread{
    public void run() {
        // TODO Auto-generated method stub
    }
}
```

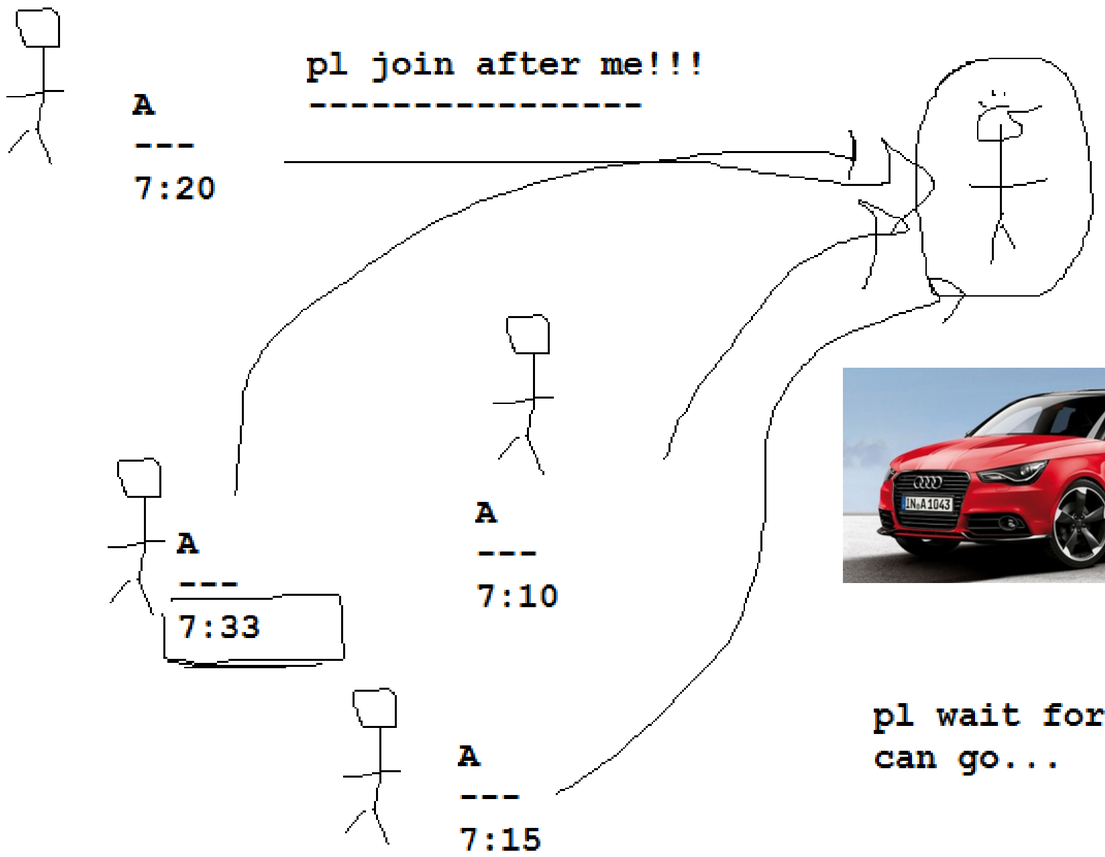
rgupta.mtech@gmail.com

```
class MyJob implements Runnable{
    @Override
    public void run() {
        System.out.println("thread is running "+Thread.currentThread().getName());
    }
}
public class A_HelloWorld {
    public static void main(String[] args) {
        System.out.println("Start : "+Thread.currentThread().getName()+
                ": "+Thread.currentThread().getPriority());
        System.out.println("it is main");

        MyJob job=new MyJob();
        Thread thread=new Thread(job,"A");
        thread.start();


        System.out.println("end : "+Thread.currentThread().getName()+
                ": "+Thread.currentThread().getPriority());
    }
}
```

```
Start : main: 5
it is main
end : main: 5
thread is running A
```

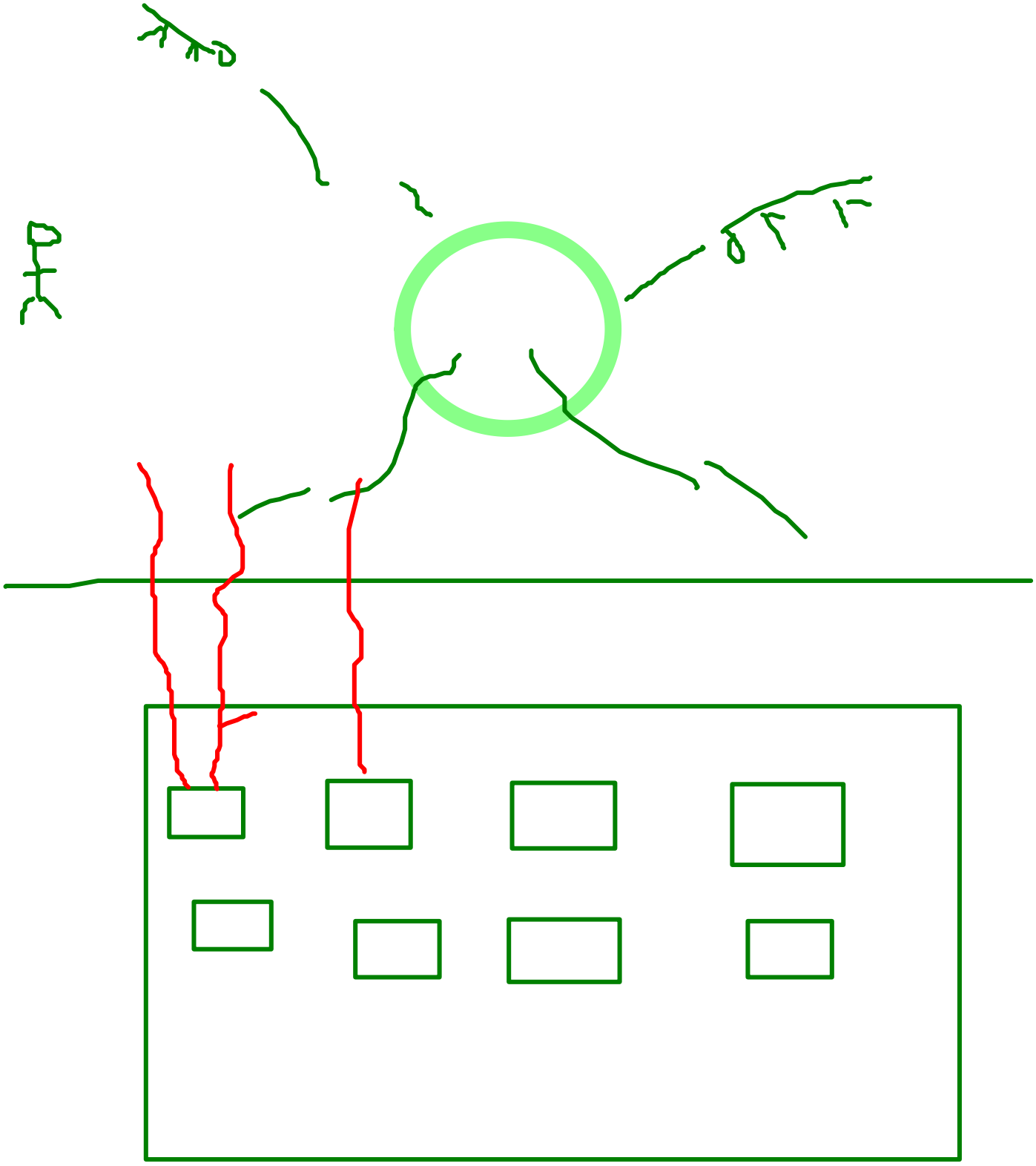


Understanding join() method



pl wait for me for 5 min else u can go...

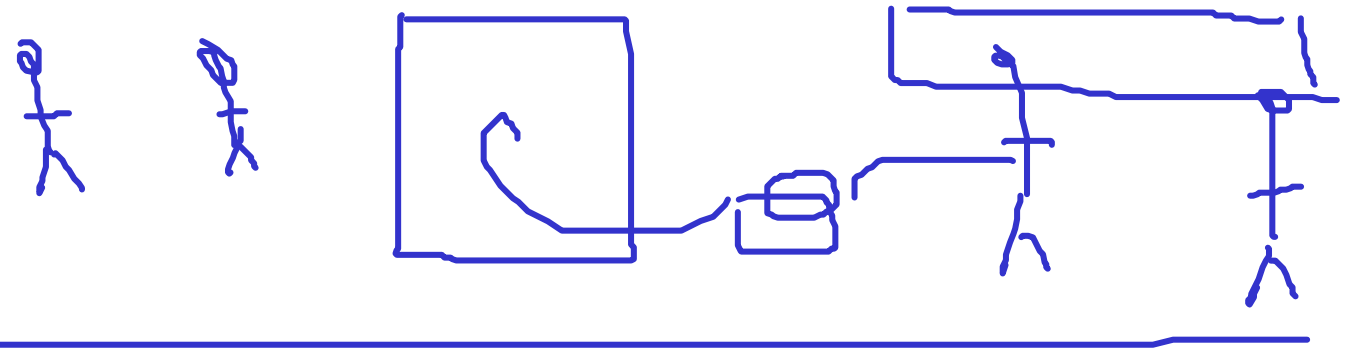
rgupta.mtech@gmail.com



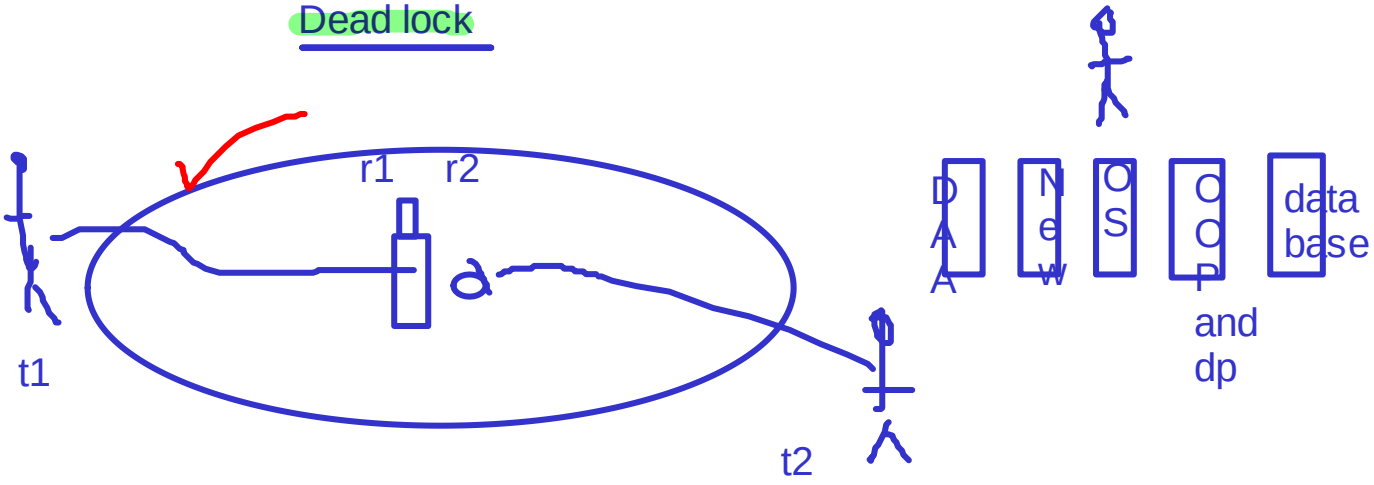
```
class Printer{
    public void printLetter(String letter){
        System.out.println("[");
        try{
            Thread.sleep(1000);
        }catch (InterruptedException e){}
        System.out.println( letter+ "]");
    }
}
class Client implements Runnable{
    private Printer printer;
    private String  letter;

    private Thread thread;

    public Client(Printer printer, String letter){
        this.printer=printer;
        this.letter=letter;
        this.thread=new Thread(this);
        thread.start();
    }
    @Override
    public void run() {
        printer.printLetter(letter);
    }
}
public class C_NeedOfSyn {
    public static void main(String[] args) {

        Printer printer=new Printer();
        Client c1=new Client(printer,"java is best");
        Client c2=new Client(printer,"java is $@###");
        Client c3=new Client(printer,"python data science that i want to learn");


    }
}
```

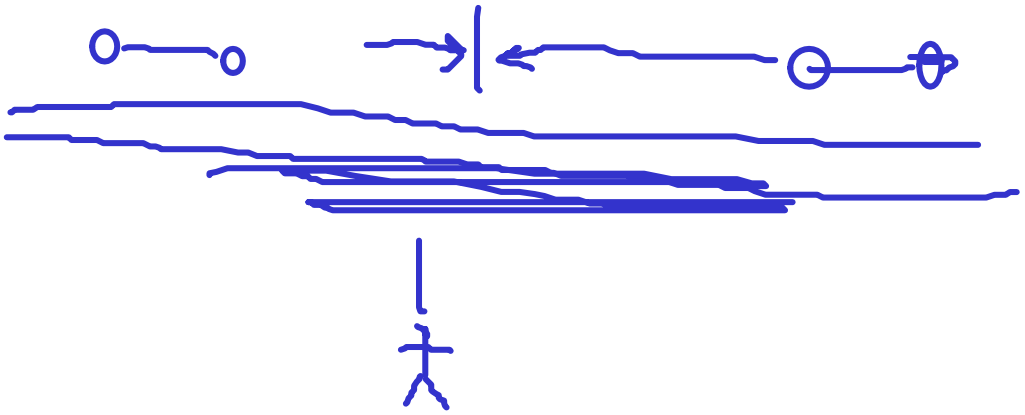
job
and
worker
compose

Race Condition /Critical section



Dead lock





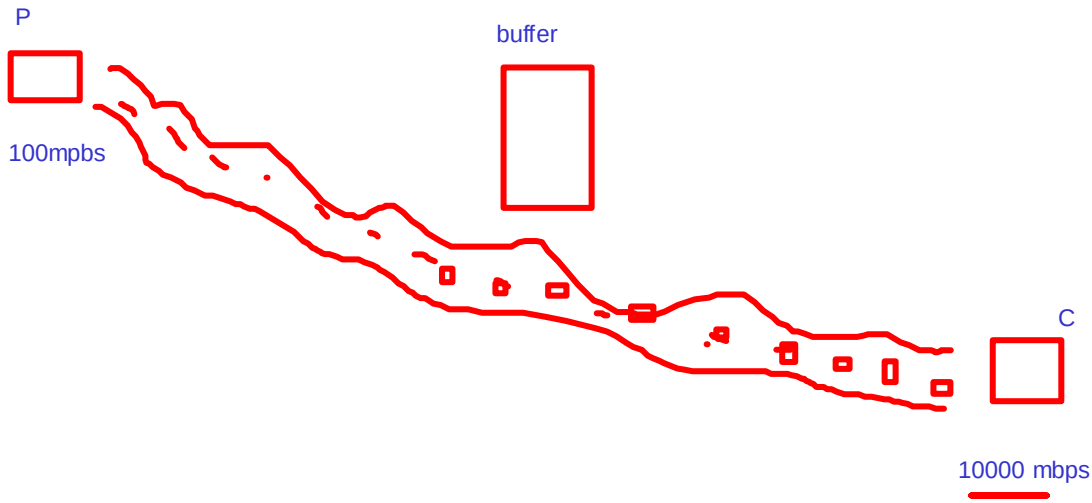
```
final Object r1="bat";
    final Object r2="ball";
```

```
Thread t1=new Thread(new Runnable() {
    @Override
    public void run() {
        synchronized (r1){
            System.out.println("t1 want to get lock on r1(bat)");

            try{
                Thread.sleep(1000);
            }catch (InterruptedException e){e.printStackTrace();}

            synchronized (r2){
                System.out.println("t1 want to get lock on r2(ball)");
            }
        }
    }
});
```

```
Thread t2=new Thread(new Runnable() {
        @Override
        public void run() {
            synchronized (r1){
                System.out.println("t2 want to get lock on r1(bat)");

                try{
                    Thread.sleep(1000);
                }catch (InterruptedException e){e.printStackTrace();}

                synchronized (r2){
                    System.out.println("t2 want to get lock on r2(ball)");
                }
            }
        }
    });
```

```
t1.start();
t2.start();
```

P and C



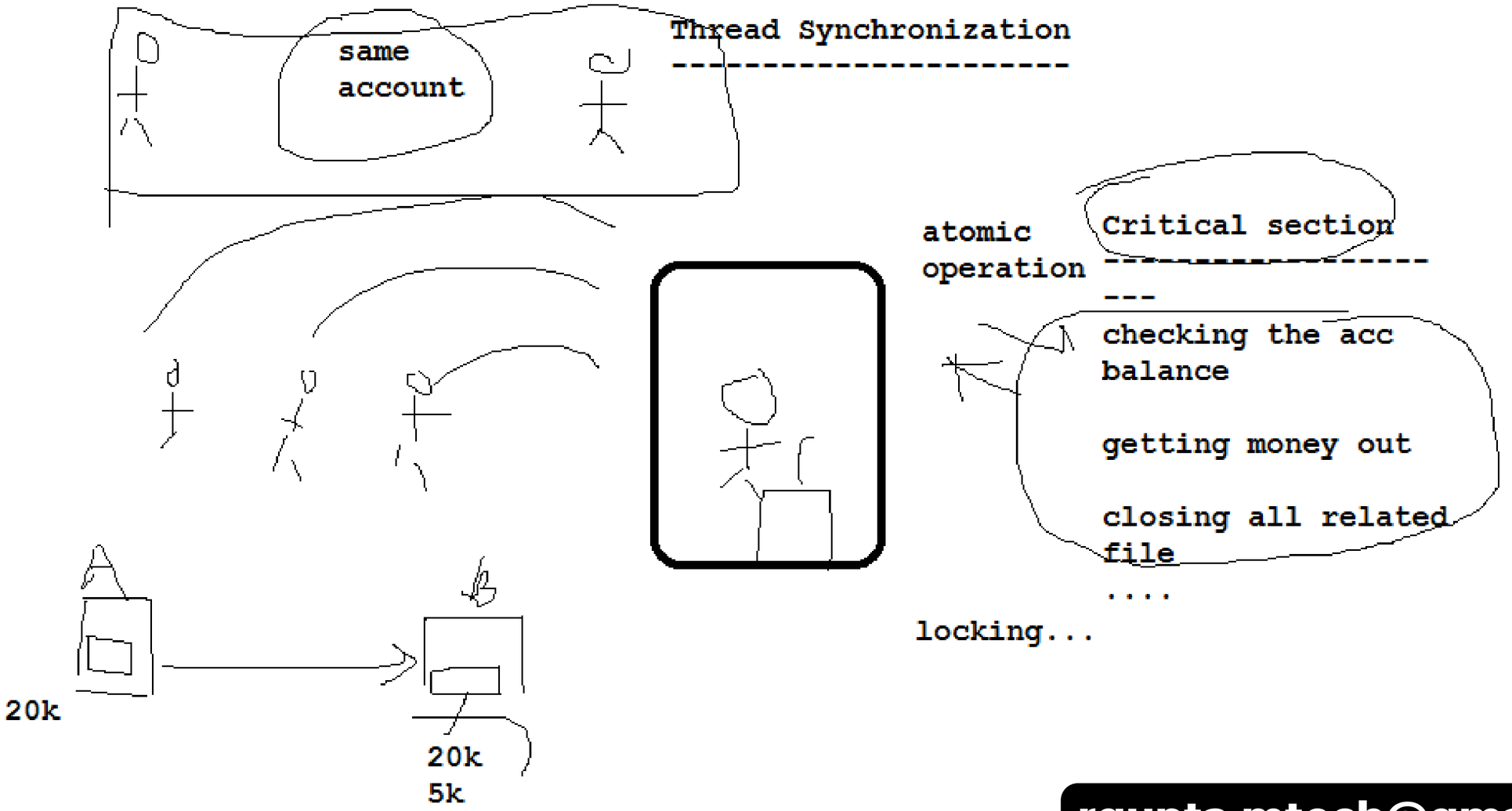
Checking thread priorities

```
class Clicker implements Runnable
{
     int click=0;
     Thread t;
     private volatile boolean running=true;
     public  Clicker(int p)
     {
          t=new Thread(this);
          t.setPriority(p);
     }
     public void run()
     {
          while(running)
               click++;
     }
     public void stop()
     {
          running=false;
     }
     public void start()
     {
          t.start();
     }
}
```

```
.....
Thread.currentThread().setPriority(Thread.MAX_PRIORITY);
Clicker hi=new Clicker(Thread.NORM_PRIORITY+2);
Clicker lo=new Clicker(Thread.NORM_PRIORITY-2);
lo.start();
hi.start();
try
{
     Thread.sleep(10000);
}
catch(InterruptedException ex){}
lo.stop();
hi.stop();
//wait for child to terminate
try
{
     hi.t.join();
     lo.t.join();
}
catch(InterruptedException ex)
{

}
System.out.println("Low priority thread:"+lo.click);
System.out.println("High priority thread:"+hi.click);
....
....
```

rgupta.mtech@gmail.com

Understanding thread synchronization



rgupta.mtech@gmail.com

Synchronization

Synchronization

- **Mechanism to controls the order in which threads execute**
- **Competition vs. cooperative synchronization**

Mutual exclusion of threads

- **Each synchronized method or statement is guarded**
- **by an object.**
- **When entering a synchronized method or statement,**
- **the object will be locked until the method is finished.**
- **When the object is locked by another thread, the**
- **current thread must wait.**

rgupta.mtech@gmail.com

Synchronized Keyword

- Synchronizing instance method

```
class SpeechSynthesizer {
    synchronized void say( String words ) {
        // speak
    }
}
```

- Synchronizing multiple methods.

```
class SpreadSheet {
    int cellA1, cellA2, cellA3;

    synchronized int sumRow() {
        return cellA1 + cellA2 + cellA3;
    }

    synchronized void setRow( int a1, int a2, int a3 ) {
        cellA1 = a1;
        cellA2 = a2;
        cellA3 = a3;
    }
    ...
}
```

- Synchronizing a block of code.

```
synchronized ( myObject ) {
    // Functionality that needs exclusive access to resources
}
```

```
synchronized void myMethod () {
    ...
}
```

is equivalent to:

```
void myMethod () {
    synchronized ( this ) {
        ...
    }
}
```

rgupta.mtech@gmail.com

Using thread synchronization

```
class CallMe
{
      synchronized void call(String msg)
      {
            System.out.print("["+msg);
            try
            {
                  Thread.sleep(500);
            }
            catch(InterruptedException ex){}
            System.out.println("]");
      }
}
```

```
class Caller implements Runnable
{
      String msg;
      CallMe target;
      Thread t;
      public Caller(CallMe targ,String s)
      {
            target=targ;
            msg=s;
            t=new Thread(this);
            t.start();
      }
      public void run()
      {
            target.call(msg);
      }
}
```

```
Caller ob1=new Caller(target, "Hello");
Caller ob2=new Caller(target,"Synchronized");
Caller ob3=new Caller(target,"Java");
```

rgupta.mtech@gmail.com

Inter thread communication

Java have elegant Interprocess

- **communication using wait() notify() and notifyAll() methods**
- **All these method defined final in the Object class Can be only called from a synchronized context**

wait()

- **Tells the calling thread to give up the monitor and go to the sleep until some other thread enter the same monitor and call**

notify()

- **Wakes up the first thread that called wait() on same object**

notifyAll()

- **Wakes up all the thread that called wait() on same object, highest priority thread is going to run first**

rgupta.mtech@gmail.com

Inter thread communication

Java have elegant Interprocess

- **communication using wait() notify() and notifyAll() methods**
- **All these method defined final in the Object class Can be only called from a synchronized context**

wait()

- **Tells the calling thread to give up the monitor and go to the sleep until some other thread enter the same monitor and call**

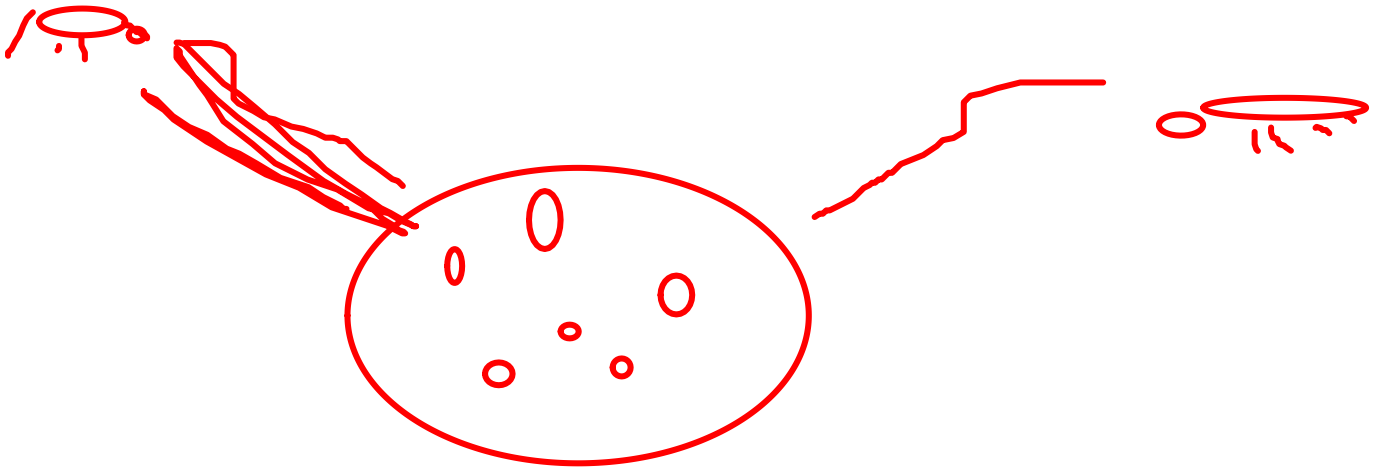
notify()

- **Wakes up the first thread that called wait() on same object**

notifyAll()

- **Wakes up all the thread that called wait() on same object, highest priority thread is going to run first**

rgupta.mtech@gmail.com



Incorrect implementation of produce consumer

```
class Q
{
        int n;
        synchronized int get()
        {
            System.out.println("got:"+n);
            return n;
        }
        synchronized void put(int n)
        {
            this.n=n;
            System.out.println("Put:"+n);
        }
}
```

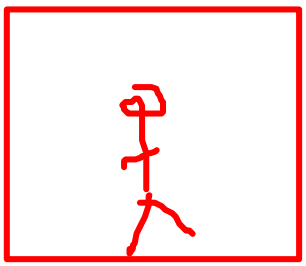
```
class Producer implements Runnable
{
    Q q;
    public Producer(Q q) {
        this.q=q;
        new Thread(this,"Producer").start();
    }
    public void run()
    {
        int i=0;
        while(true)
            q.put(i++);
    }
}
```

```
public class PandC {
public static void main(String[] args) {
    Q q=new Q();
    new Producer(q);
    new Consumer(q);
    System.out.println("ctrol C for exit");
}
}
```

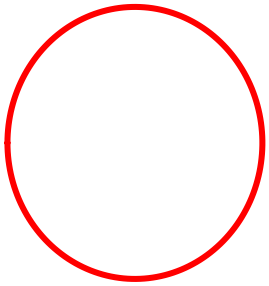
```
class Consumer implements Runnable
{
    Q q;
    Consumer(Q q)
    {
        this.q=q;
        new Thread(this,"consumer").start();
    }
    public void run()
    {
        while(true)
            q.get();
    }
```

rgupta.mtech@gmail.com

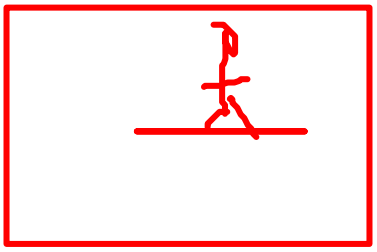
boolean valueSet=false;



p



q

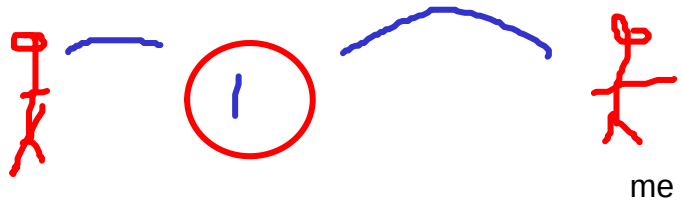


c

```
class Q {
   int n;
   boolean valueSet=false;

   //consumer t1
   synchronized   int get() {
     if(!valueSet){
       try{
         wait();
       }catch (InterruptedException e){}
     }

     System.out.println("get: " + n)
     valueSet=false;
     notifyAll();

     return n;
   }
```

```
synchronized void put(int n) {
     if(valueSet){
       try{
          wait();
       }catch (InterruptedException e){}
     }
     this.n = n;
     valueSet=true;
     System.out.println("put: " + n);
     notifyAll();
   }
```

vs =t



mom

me

Q

Queue
size
n

n

consumer

boolean valueSet=false;

producer

```
synchronized   int get() {
      while(!valueSet){
        try{
           wait();
        }catch (InterruptedException e){}
      }
      System.out.println("get: " + n);
      valueSet=false;
      notifyAll();

      return n;
   }
```

```
synchronized void put(int n) {
      while(valueSet){
        try{
           wait();
        }catch (InterruptedException e){}
      }
      this.n = n;
      valueSet=true;
      System.out.println("put: " + n);
      notifyAll();
   }
```

put :1

get:1

C

P

boolean valueSet=t

f

Avoid using low level

t1

t1

```
class Q {
  int n;

  int synchronized get() {
     System.out.println("get: " + n);
     return n;
  }

   void synchronized put(int n) {
     this.n = n;
     System.out.println("put: " + n);
  }
}
```

```
class Q {

     synchronized(this){
       int n;
        int getM() {
              System.out.println("get: " + n);
              return n;
        }
        void putM(int n) {
              this.n = n;
              System.out.println("put: " + n);
        }
     }
}
```

t1.start();
t2.start();

```
private static BlockingQueue<Integer> queue=new ArrayBlockingQueue<>(1);
```

```
Thread t1=new Thread(new Runnable() {
        @Override
        public void run() {
          try {
            produce();
          } catch (InterruptedException e) {
             throw new RuntimeException(e);
          }
        }
    });
```

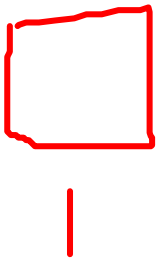
```
Thread t2=new Thread(new Runnable() {
        @Override
        public void run() {
          try {
             consumer();
          } catch (InterruptedException e) {
             throw new RuntimeException(e);
          }
        }
    });
```

t1.start()
t2.start()

```
private static void consumer()throws InterruptedException {
   //InterruptedException *
   while (true){
      System.out.println("got:"+queue.take()+" :"+queue.size());
   }
}
```

observer design

```
private static void produce() throws InterruptedException {
     Random random=new Random();
     while (true){
        Integer value=random.nextInt(100);
        System.out.println("put: "+value);
        queue.put(value);//Blocking method
     }
  }
```



```
public class DemoBQ
   private static BlockingQueue<Integer> queue=new ArrayBlockingQueue<>(1);
```

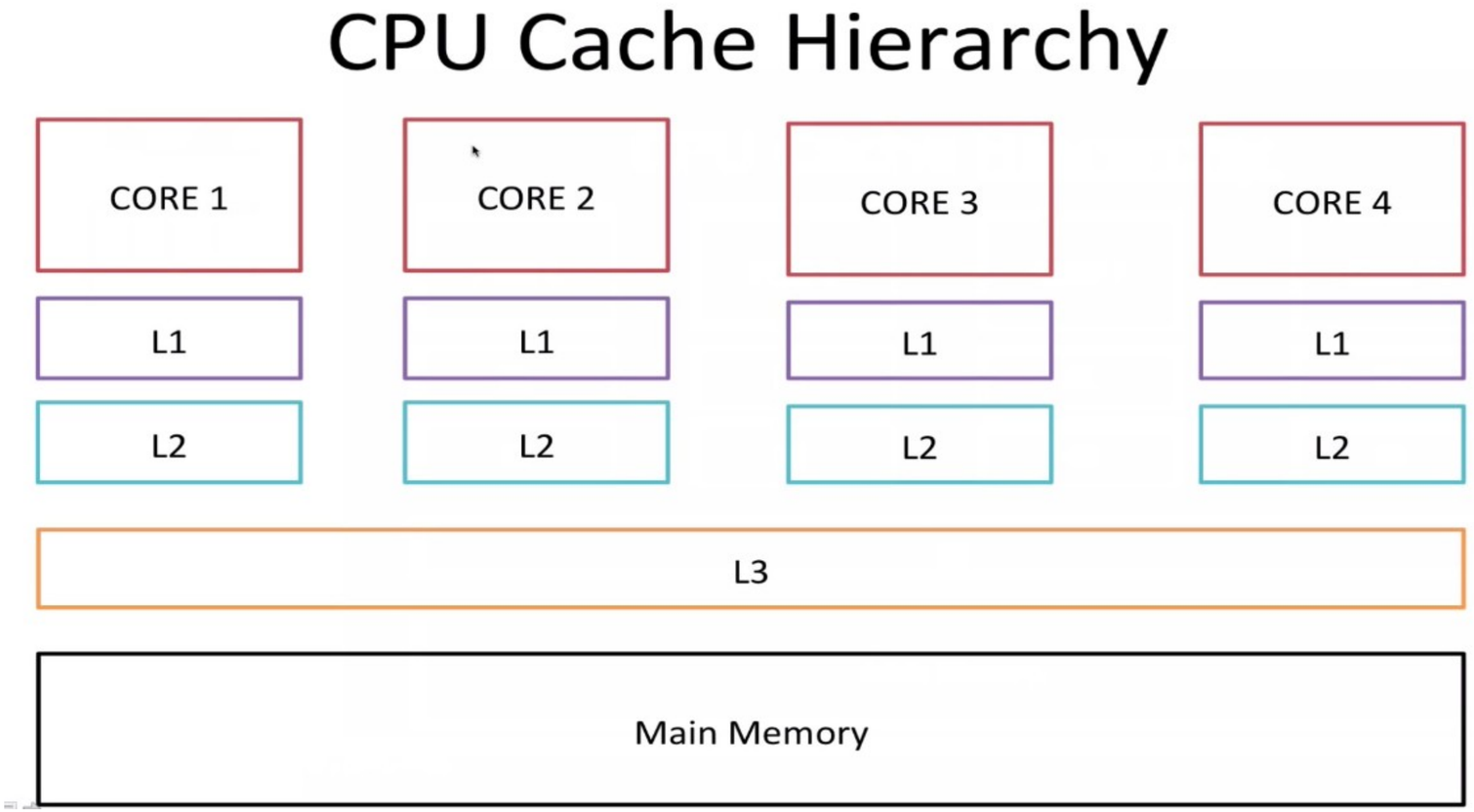
stop() deprecated : should not be use

Correct implementation of produce consumer

```
class Q
{   int n;
        boolean valueSet=false;
        synchronized int get()
        {
            if(!valueSet)
                try
                {
                    wait();
                }
            catch(InterruptedException ex){}
            System.out.println("got:"+n);
            valueSet=false;
            notify();
            return n;
        }
        synchronized void put(int n)
        {
            if(valueSet)
                try
                {
                    wait();
                }
            catch(InterruptedException ex){}
            this.n=n;
            valueSet=true;
            System.out.println("Put:"+n);
            notify();
        }
}
```

rgupta.mtech@gmail.com

volatile



rgupta.mtech@gmail.com

Java Memory Model JMM

JMM is a specification which guarantees visibility of fields (aka happens before) amidst reordering of instructions.

rgupta.mtech@gmail.com

Out of Order executions

Performance driven changes done by Compiler, JVM or CPU

Consider the code

```
a = 3;
b = 2;
a = a + 1;
```

Instructions

→ Load a
→ Set to 3
→ Store a

→ Load b
→ Set to 2
→ Store b

→ Load a
→ Set to 4
→ Store a

Instruction Re-ordering

```
a = 3;
a = a + 1;

b = 2;
```

Instructions

→ Load a
→ Set to 3
→ Set to 4
→ Store a

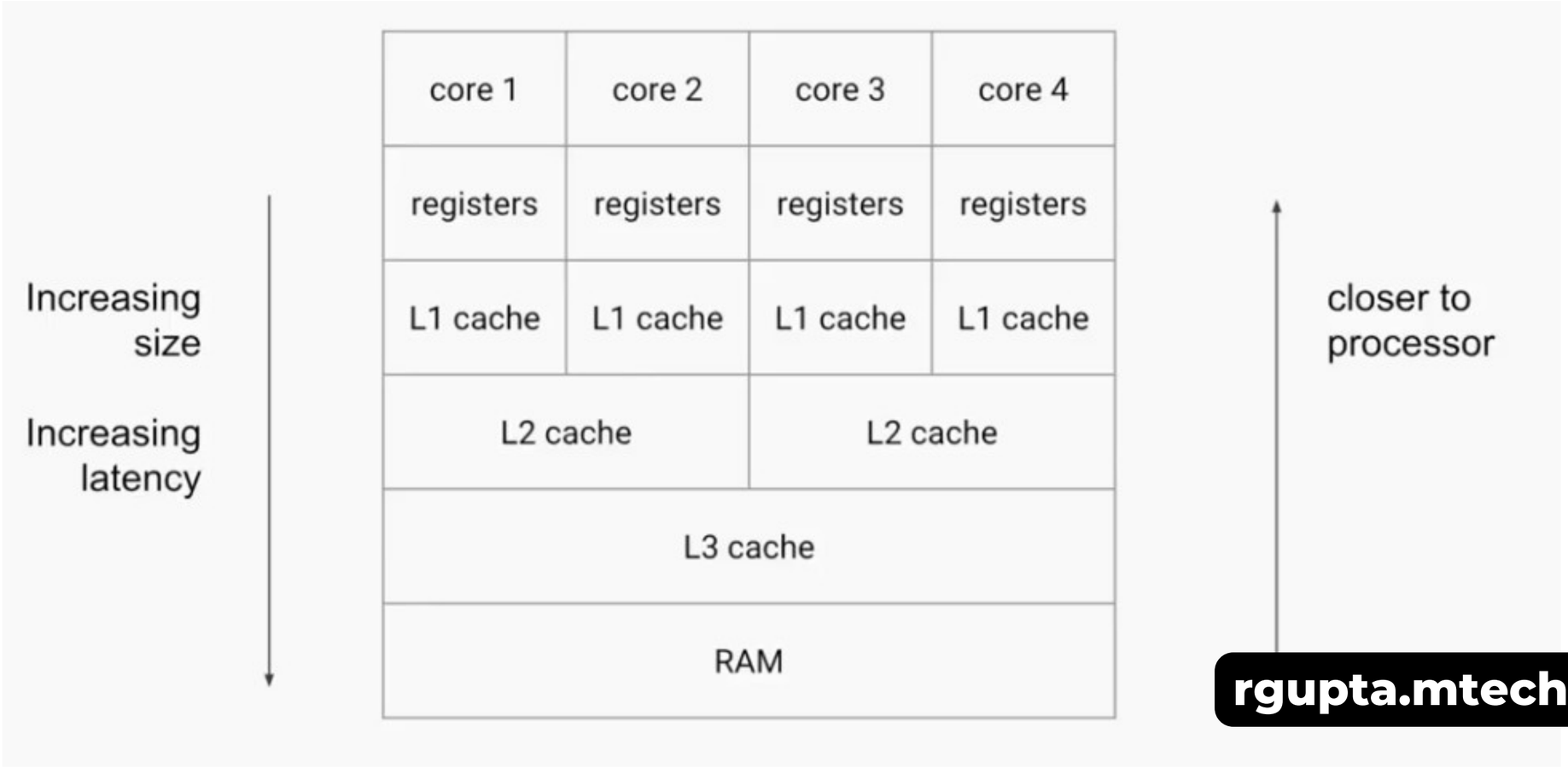
→ Load b
→ Set to 2
→ Store b

rgupta.mtech@gmail.com

Field Visibility

In persence of multiple threads aka Concurrency

CPU Cache (Multicore Processor)

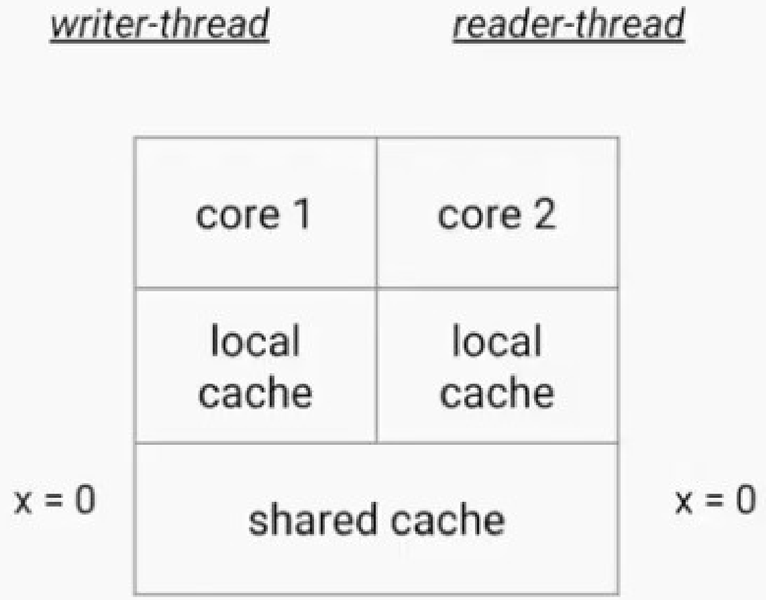


rgupta.mtech@gmail.com

Field Visibility Problem

Consider Two Threads : Writer and reader threads

```
public class FieldVisibility {

    int x = 0;

    public void writerThread() {
        x = 1;
    }

    public void readerThread() {
        int r2 = x;
    }
}
```

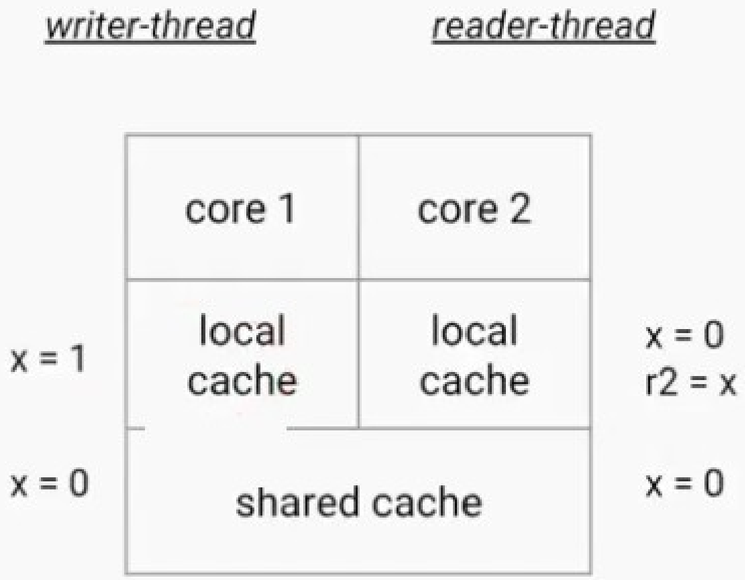


rgupta.mtech@gmail.com

Field Visibility Problem

Consider Two Threads : Writer and reader threads

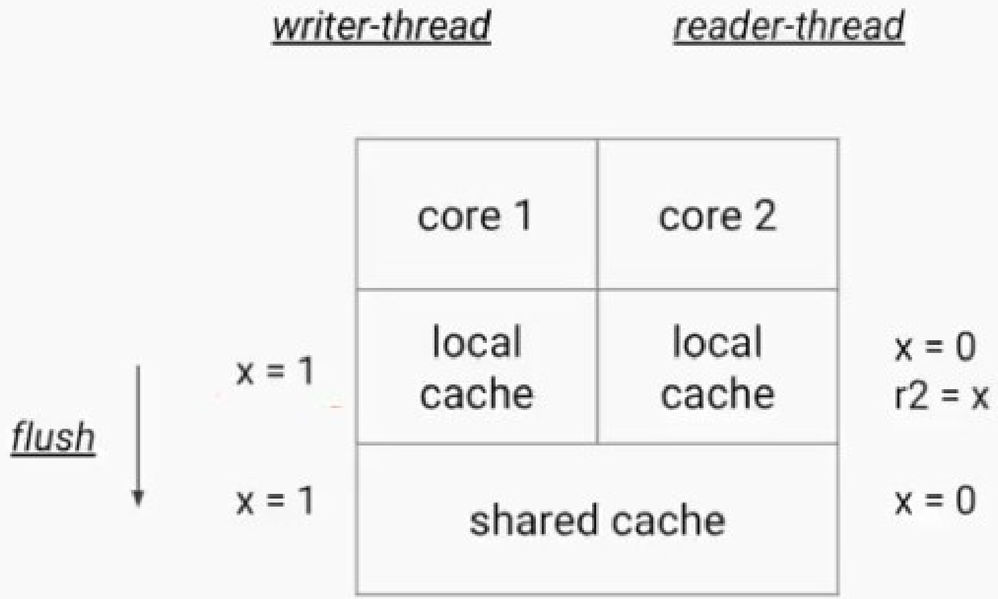
```
public class FieldVisibility {

    int x = 0;

    public void writerThread() {
        x = 1;
    }

    public void readerThread() {
        int r2 = x;
    }
}
```



rgupta.mtech@gmail.com

Field Visibility With volatile

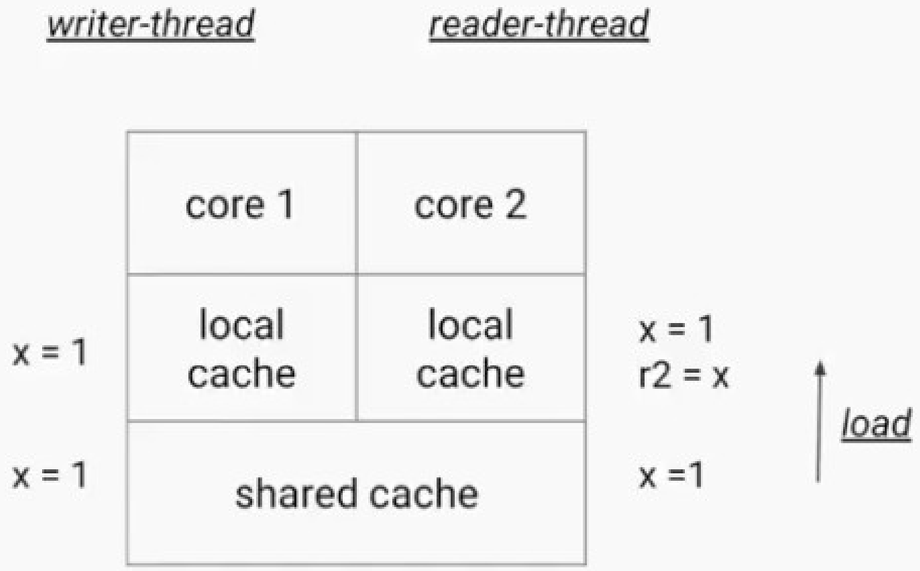
```java
public class VolatileVisibility {

    volatile int x = 0;

    public void writerThread() {
        x = 1;
    }

    public void readerThread() {
        int r2 = x;
    }
}
```



rgupta.mtech@gmail.com

Field Visibility With volatile

```
public class VolatileVisibility {

    volatile int x = 0;

    public void writerThread() {
        x = 1;
    }

    public void readerThread() {
        int r2 = x;
    }
}
```



rgupta.mtech@gmail.com

Happen Before Relationship for volatile

```
public class VolatileFieldsVisibility {

    int a = 0, b = 0, c = 0;
    volatile int x = 0;

    public void writerThread() {

        a = 1;
        b = 1;
        c = 1;

        x = 1;  // write of x
    }

    public void readerThread() {

        int r2 = x;   // read of x

        int d1 = a;
        int d2 = b;
        int d3 = c;
    }
}
```

JMM rule:
Whatever happens before x=1
in writer thread
must be visible in reader thread
after int r2=x;

rgupta.mtech@gmail.com

Happen Before Relationship for volatile

Happen Before is also application to :

- **Synchronized block**
- **Locks**
- **Concurrent collections**
- **Thread Operations (join, start)**
- **final fields (Special behavior)**

rgupta.mtech@gmail.com

Same Behaviour with syn block

```
public class SynchronizedFieldsVisibility {

    int a = 0, b = 0, c = 0;
    volatile int x = 0;

    public void writerThread() {
        a = 1;
        b = 1;
        c = 1;

        synchronized (this) {
            x = 1;
        }
    }

    public void readerThread() {

        synchronized (this) {
            int r2 = x;
        }

        int d1 = a;
        int d2 = b;
        int d3 = c;
    }
}
```

Very Important:
must synchronized on same object

rgupta.mtech@gmail.com

Same Behaviour with syn block

```
public class SynchronizedFieldsVisibility {

    int a = 0, b = 0, c = 0;
    volatile int x = 0;

    public void writerThread() {
        a = 1;
        b = 1;
        c = 1;

        synchronized (this) {
            x = 1;
        }
    }

    public void readerThread() {

        synchronized (this) {
            int r2 = x;
        }

        int d1 = a;
        int d2 = b;
        int d3 = c;
    }
}
```

Very Important:
must synchronized on same object

rgupta.mtech@gmail.com

Better to be more clear though!

```
public class SynchronizedFieldsVisibility {

    int a = 0, b = 0, c = 0;
    volatile int x = 0;

    public void writerThread() {

        synchronized (this) {
            a = 1;
            b = 1;
            c = 1;
            x = 1;
        }
    }

    public void readerThread() {

        synchronized (this) {
            int r2 = x;
            int d1 = a;
            int d2 = b;
            int d3 = c;
        }
    }
}
```

rgupta.mtech@gmail.com

Same behaviour with Locks

```
public class LockVisibility {

    int a = 0, b = 0, c = 0, x = 0;
    Lock lock = new ReentrantLock();

    public void writerThread() {

        lock.lock();
        a = 1;
        b = 1;
        c = 1;
        x = 1;
        lock.unlock();
    }

    public void readerThread() {

        lock.lock();
        int r2 = x;
        int d1 = a;
        int d2 = b;
        int d3 = c;
        lock.unlock();
    }
}
```

rgupta.mtech@gmail.com

Example

```java
public class VolatileVisibility {

    boolean flag = true;

    public void writerThread() {
        flag = false;
    }

    public void readerThread() {
        while (flag) {
            // do some operations
        }
    }
}
```

wrong

```java
public class VolatileVisibility {

    volatile boolean flag = true;

    public void writerThread() {
        flag = false;
    }

    public void readerThread() {
        while (flag) {
            // do some operations
        }
    }
}
```

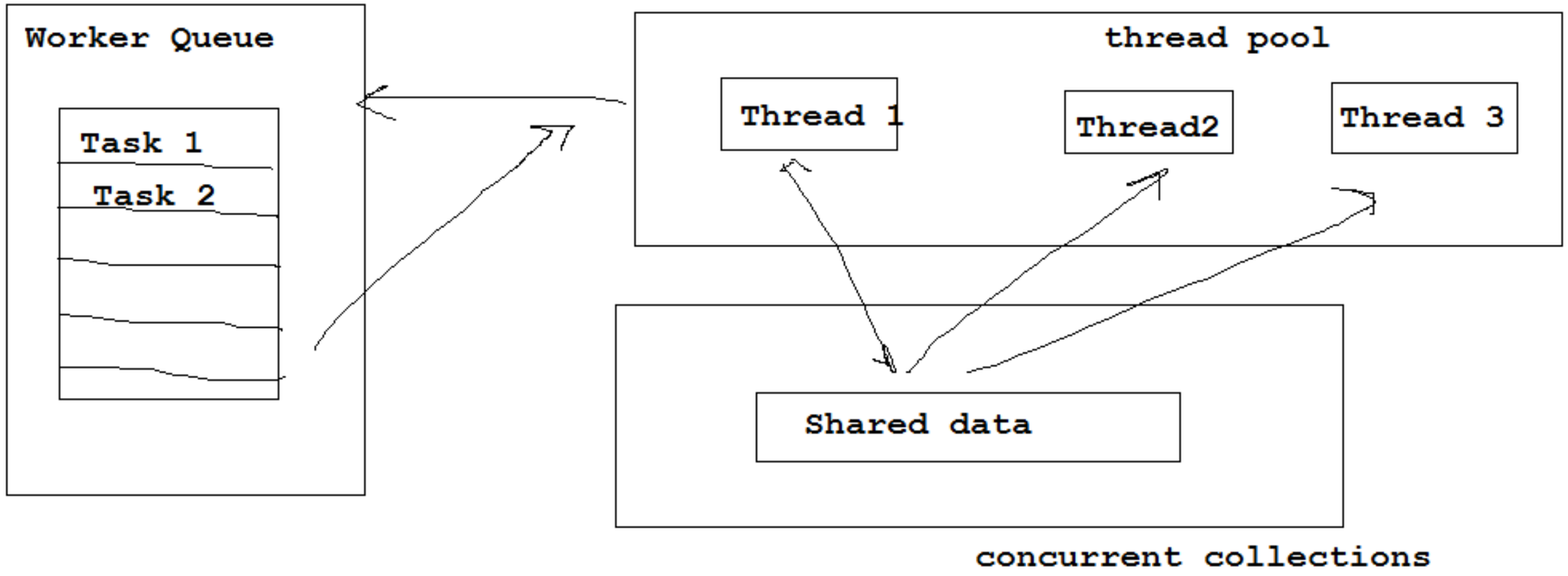
right

rgupta.mtech@gmail.com

java.util.concurrency j.u.c

Building block of concurrent application

- Thread pool
- Shared data
- Worker queues



rgupta.mtech@gmail.com

Java concurrency –Pre Java 5

- Task Executor
  - No such framework
  - Runnable/Thread
  - No of task= no of threads
  - Use wait() and notify() for thread intercommunication
- Collection
  - Vector, Hashtable
- Worker queue
  - No framework for task dispatching
  - Create an thread when a task to run
- Too primitive low level
  - High cost of thread management
  - Synchronization=poor performance
  - <<Runnable>> can not return values

rgupta.mtech@gmail.com

Java concurrency –Java 5

- Task Executor
  - Executor Service Framework
  - Support scheduling (Timed execution)
  - Thread pools (lifecycle management)
  - Specify order of execution
  - Task
    - Object of <<Callable>> or <<Runnable>>
    - <<Callable>> support for asynchronous communication call() and return values
    - Future: hold result of processing
- Collections
  - ConcurrentHashMap, CopyOnWriteList
- Worker queue
  - Blocking Queue, Deque.....
- Issues
  - No of task can't altered at run time
  - Very difficult and performance intensive when applied to recursive style of problems.....

rgupta.mtech@gmail.com

Java concurrency –Java 7

- Fork join framework
  - For supporting a style of programming in which problem are solved recursively splitting them into subtask that are solved in parallel, waiting for them to complete and then composing results !
- JSR 166 Java 7
  - Extend Executor framework for recursive style of problems
- Implements Work stealing algorithm
  - Idle worker “steal” the woks from worker who are busy..

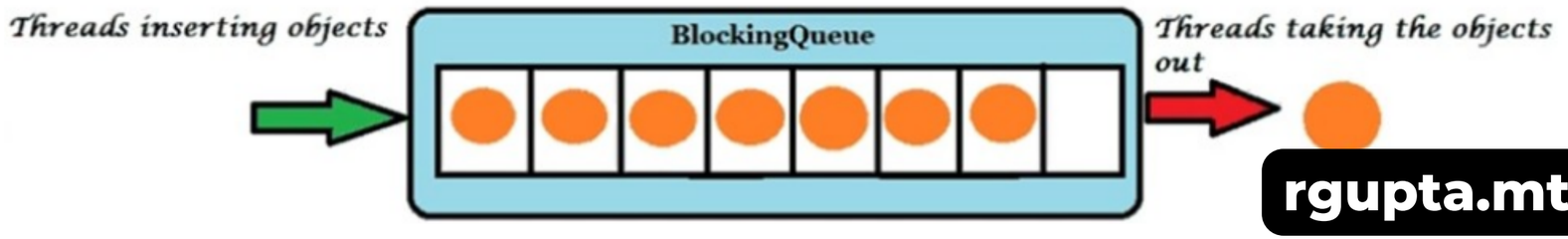
rgupta.mtech@gmail.com

Blocking Queue: introduction

- BlockingQueue is an interface under juc package. It help to contain objects when few threads is inserting object and other threads are taking the object out of it
-
- The threads will keep on inserting the objects unitill the max capacity of BlockingQueue is reached,after whick the threads will be blocked and they will not be able to insert further objects, threads will be in blocked state until few objects are taken out from BlockingQueue by other threads and there is space available in the blockingqueue
-
- The threads will keep on taking the objects out from the blockingqueue until there is no object left in blockingqueue, after which the threads will be blocked and they will remain in blocked state until few objects are insertred in the blockingqueue by other threads
-
- Use put() and take() to implement P and C
-

The **poll()** doesn't wait if the messageQueue is empty. It doesn't care. It just gets the value even if the messageQueue is empty. Finally the application ended.

The **take()** method waits at that particular line if the messageQueue is empty. It only goes to next line if the messageQueue got something inside to process. So **take()** method block the thread and keeps it, which makes the application runs endlessly.



rgupta.mtech@gmail.com

Blocking Queue: Thread safe DS

BlockingQueue<E> is an interface with the following methods

	Throws exception	*Special value*	*Blocks*	*Times out*
Insert	`add(e)`	`offer(e)`	`put(e)`	`offer(e,time,unit)`
Remove	`remove()`	`poll()`	`take()`	`poll(time,unit)`
Examine	`element()`	`peek()`	*N/A*	*N/A*

rgupta.mtech@gmail.com

Synchronization with Locks: ReentrantLock

```
class PrintQueue
{
    private final Lock queLock=new ReentrantLock();
    public void printJob()
    {
        queLock.lock();
        try{
            System.out.println("Thread "+Thread.currentThread().getName()+" Done the job");
            Thread.sleep(100);
        }
        catch(InterruptedException ex){}

        finally{
            queLock.unlock();
        }
    }

}
```

creating an lock

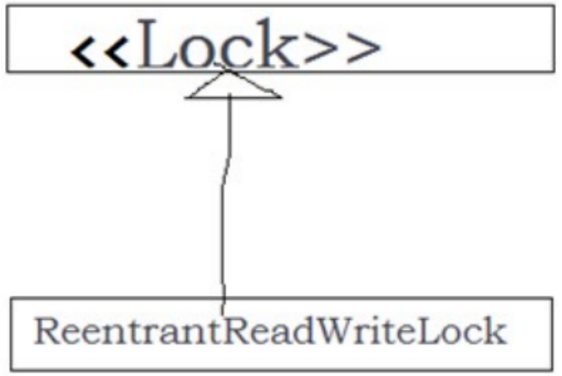
gettting control of lock

must be unlock in finally block

rgupta.mtech@gmail.com

Synchronization with Locks: ReentrantReadWriteLock

- ReentrantReadWriteLock has two locks
  - One for read
    - More then one thread can read
  - One for write
    - Only one thread can write



    - Ex: Lets consider case when one thread writing Price Information that is read by more then one reader

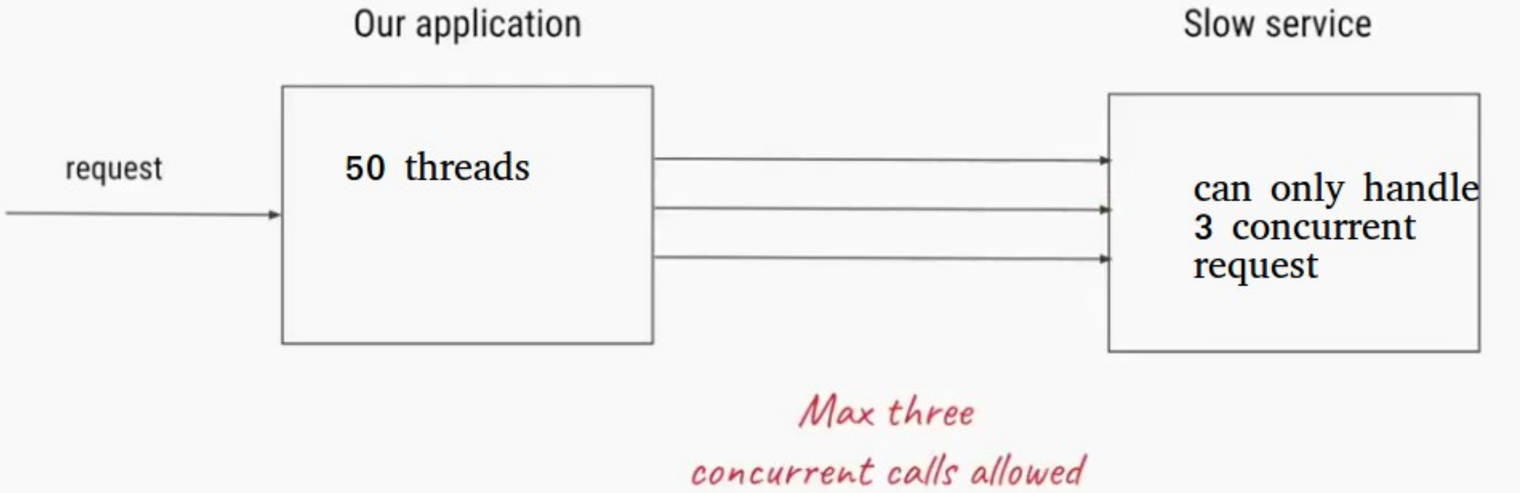
Synchronization with Locks: ReentrantReadWriteLock

```
class PriceInfo
{
    private double price1,price2;
    private ReadWriteLock lock=new ReentrantReadWriteLock();
    PriceInfo(){
        price1=1.0;
        price2=2.0;
    }
    public double getPrice1(){
        lock.readLock().lock();
        double value=price1;
        lock.readLock().unlock();
        return value;
    }
    public double getPrice2(){
        lock.readLock().lock();
        double value=price2;
        lock.readLock().unlock();
        return value;
    }
    public void setPrice(double price1, double price2){
        lock.writeLock().lock();
        this.price1=price1;
        this.price2=price2;
        lock.writeLock().unlock();
    }

}
```

rgupta.mtech@gmail.com

Semaphores: use cases

Use Case: We have a slow API to class from our application,
it can only handle 3 concurrent class
while we have 50 thread to call that service? How to handle?



rgupta.mtech@gmail.com

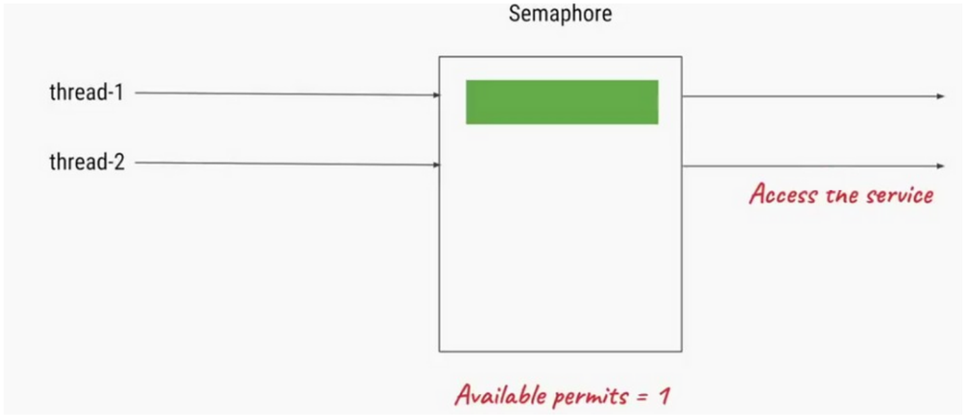
Semaphores: use cases

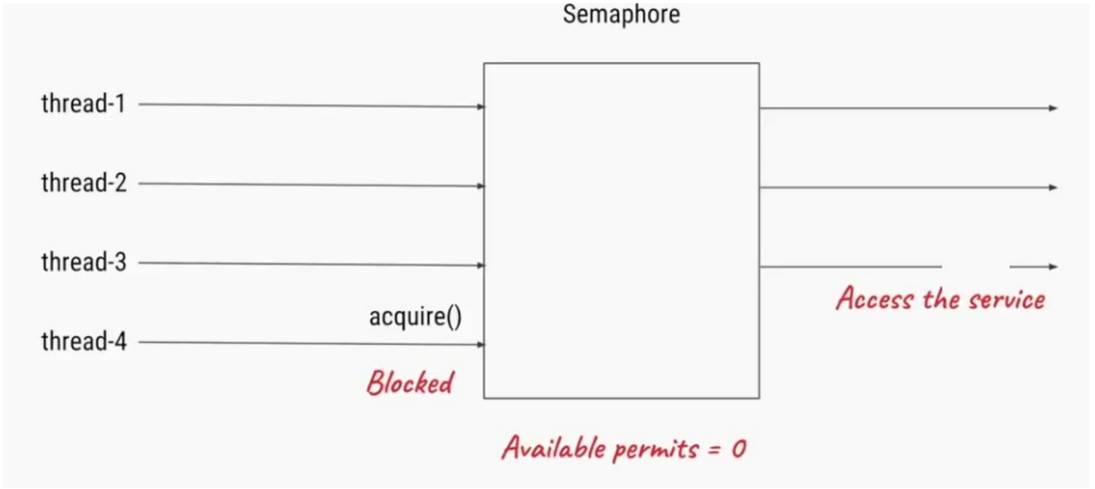
```java
public static void main(String[] args) throws InterruptedException {

    ExecutorService service = Executors.newFixedThreadPool( nThreads: 50);
    IntStream.of(1000).forEach(i -> service.execute(new Task()));

    service.shutdown();
    service.awaitTermination( timeout: 1, TimeUnit.MINUTES);
}

static class Task implements Runnable {

    @Override
    public void run() {
        // some processing
        // IO call to the slow service
        // rest of processing
    }
}
```

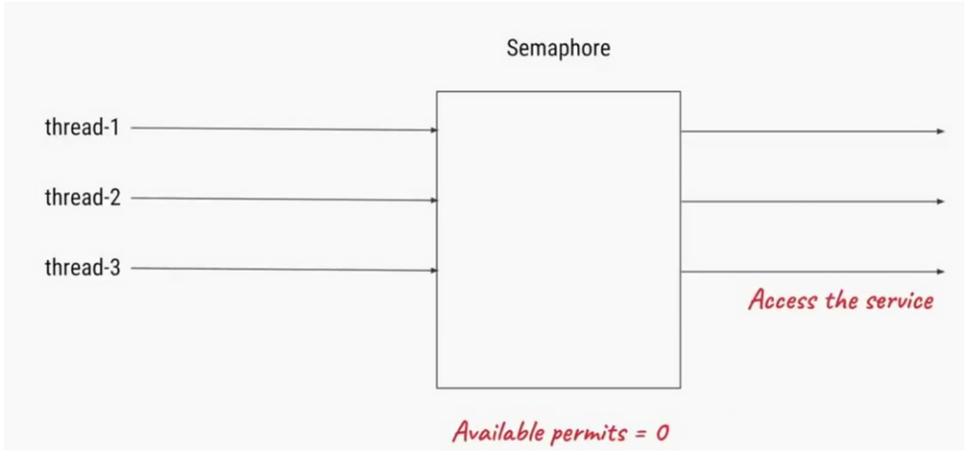
This might be called 50 times concurrently!!

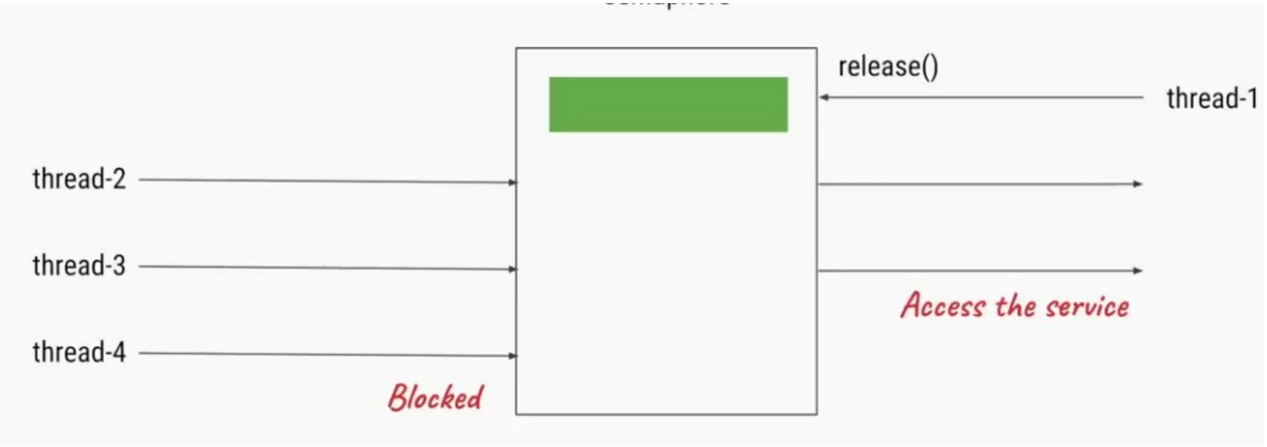
rgupta.mtech@gmail.com

Semaphores: how permit works?









rgupta.mtech@gmail.com

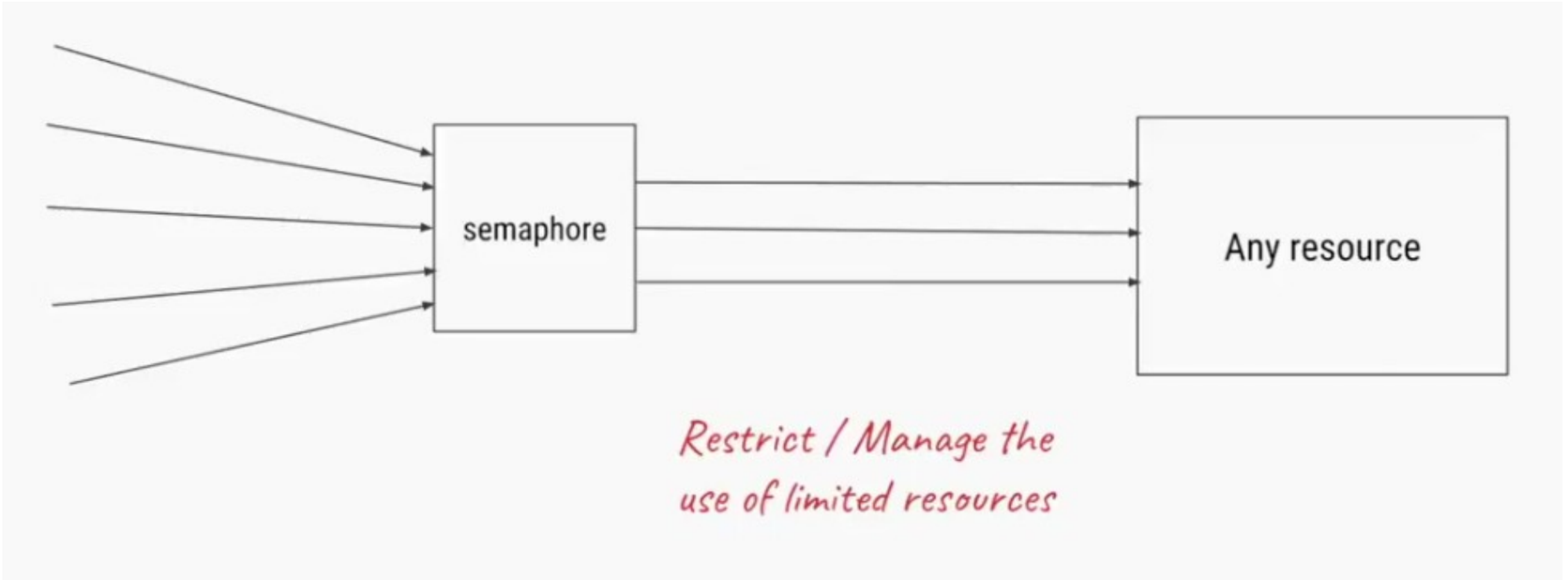
Semaphores: use cases

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

    Semaphore semaphore = new Semaphore( permits: 3);

    ExecutorService service = Executors.newFixedThreadPool( nThreads: 50);
    IntStream.of(1000).forEach(i -> service.execute(new Task(semaphore)));

    service.shutdown();
    service.awaitTermination( timeout: 1, TimeUnit.MINUTES);
}

static class Task implements Runnable {

    @Override
    public void run() {
        // some processing

        semaphore.acquireUninterruptibly();
        // IO call to the slow service
        semaphore.release();

        // rest of processing
    }
```

Only 3 threads can acquire at a time

rgupta.mtech@gmail.com

Semaphores: use cases



Method	Meaning
tryAcquire	Try to acquire, if no permit available, do not block. Continue doing something else.
tryAcquire (timeout)	Same as above but with timeout
availablePermits	Returns count of permits available
new Semaphore (count, fairness)	FIFO. Fairness guarantee for threads waiting the longest.

rgupta.mtech@gmail.com

Thread synchronization utilities:CountDownLatch

- Waiting for multiple concurrent events
- Allow one/more thread to wait until a set of operations are completed..
- CountDownLatch class initialized with an integer number i.e. no of operations that thread must wait to complete before they finished
- When a thread want to wait for executions of those operations, it uses await() method, it put thread to sleep until operation are complete
- When one of these operation complete, it use CountDownLatchs, countDown() method to decrease the count
- When it (counter) become zero, the class wake up all the threads that sleeping in await() method

- Ex: CountDownLatch class implementing an video conference system. The system wait for all participant before it begin

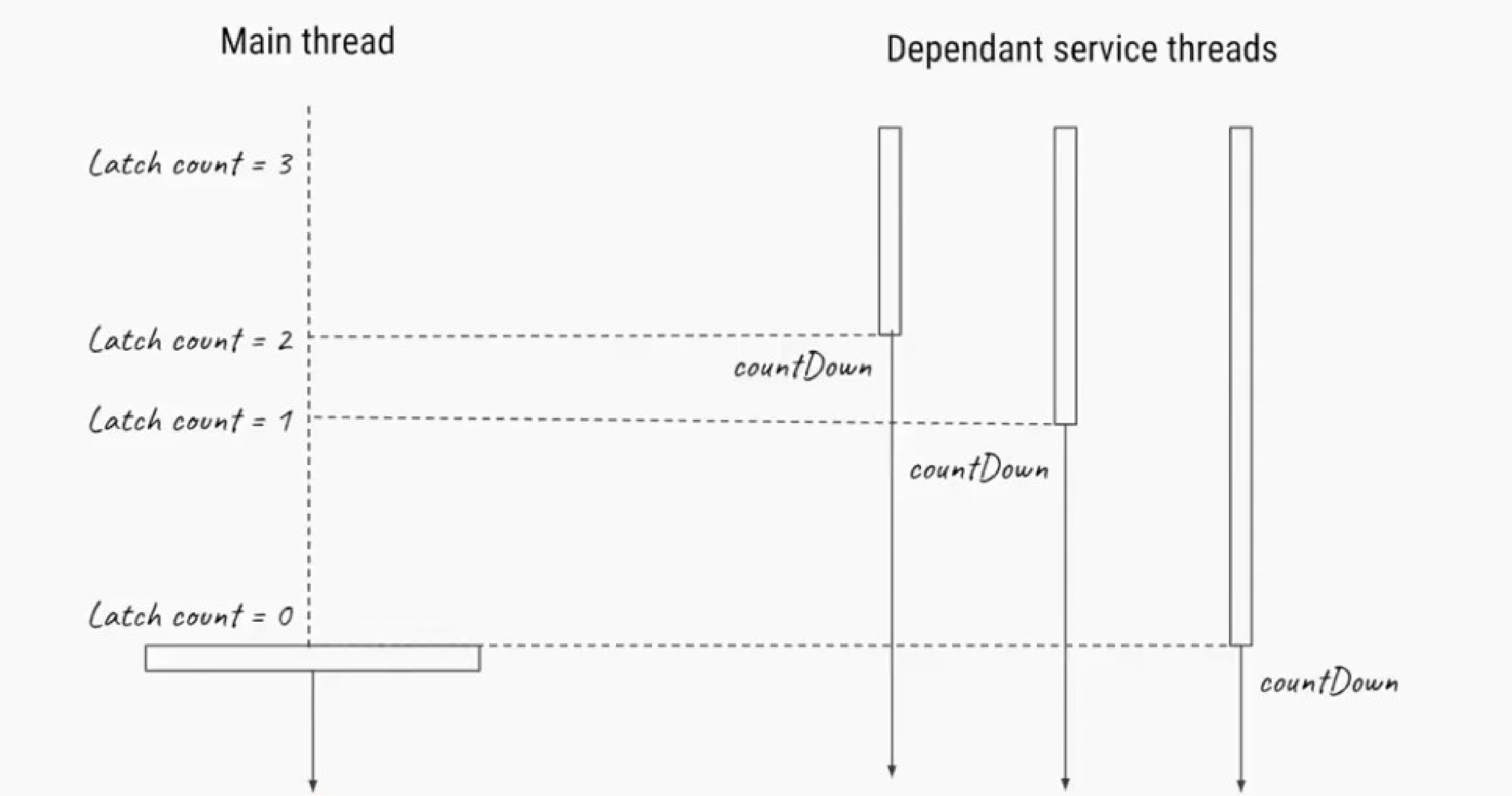
rgupta.mtech@gmail.com

CountDownLatch

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

    ExecutorService executor = Executors.newFixedThreadPool( nThreads: 4);

    CountDownLatch latch = new CountDownLatch(3);
    executor.submit(new DependentService(latch));
    executor.submit(new DependentService(latch));
    executor.submit(new DependentService(latch));

    latch.await();

    System.out.println("All dependant services initialized");
    // program initialized, perform other operations
}

public static class DependentService implements Runnable {

    private CountDownLatch latch;
    public DependentService(CountDownLatch latch) { this.latch = latch; }

    @Override
    public void run() {
        // startup task
        latch.countDown();
        // continue w/ other operations
    }
}
```

rgupta.mtech@gmail.com

CountDownLatch



rgupta.mtech@gmail.com

CountDownLatch Example

```
class VideoConference implements Runnable
{
    private CountDownLatch controller;

    VideoConference(int n)
    {
        controller=new CountDownLatch(n);
    }

    public void arrive(String name)
    {
        System.out.println("Guest name:"+name+" arrived");
        controller.countDown();//decrease.
        System.out.println("Controller wating for more:"+controller.getCount());
    }
    @Override
    public void run() {

        System.out.println("init video conference"+controller.getCount());
        try
        {
            controller.await();
            System.out.println("All comes....lets start now.....");
        }
        catch(InterruptedException ex){}
    }

}
```

rgupta.mtech@gmail.com

CountDownLatch Example

```java
class Participant implements Runnable
{
    private VideoConference conference;
    private String participantsName;

    Participant(VideoConference conference,String participantsName)
    {
        this.conference=conference;
        this.participantsName=participantsName;
    }
    @Override
    public void run() {
        try
        {
            Thread.sleep(100);
        }
        catch(InterruptedException ex){}
        conference.arrive(participantsName);
    }

}
```

```java
public class VideoConferenceDemo {

    public static void main(String[] args) {
        VideoConference vc=new VideoConference(10);

        //running conference thread.....
        Thread t=new Thread(vc);
        t.start();

        //now create 10 participants for above conference

        for(int i=0;i<10;i++)
        {
            Participant p=new Participant(vc,"Participant "+i);
            Thread t1=new Thread(p);
            t1.start();
        }
    }
}
```

rgupta.mtech@gmail.com

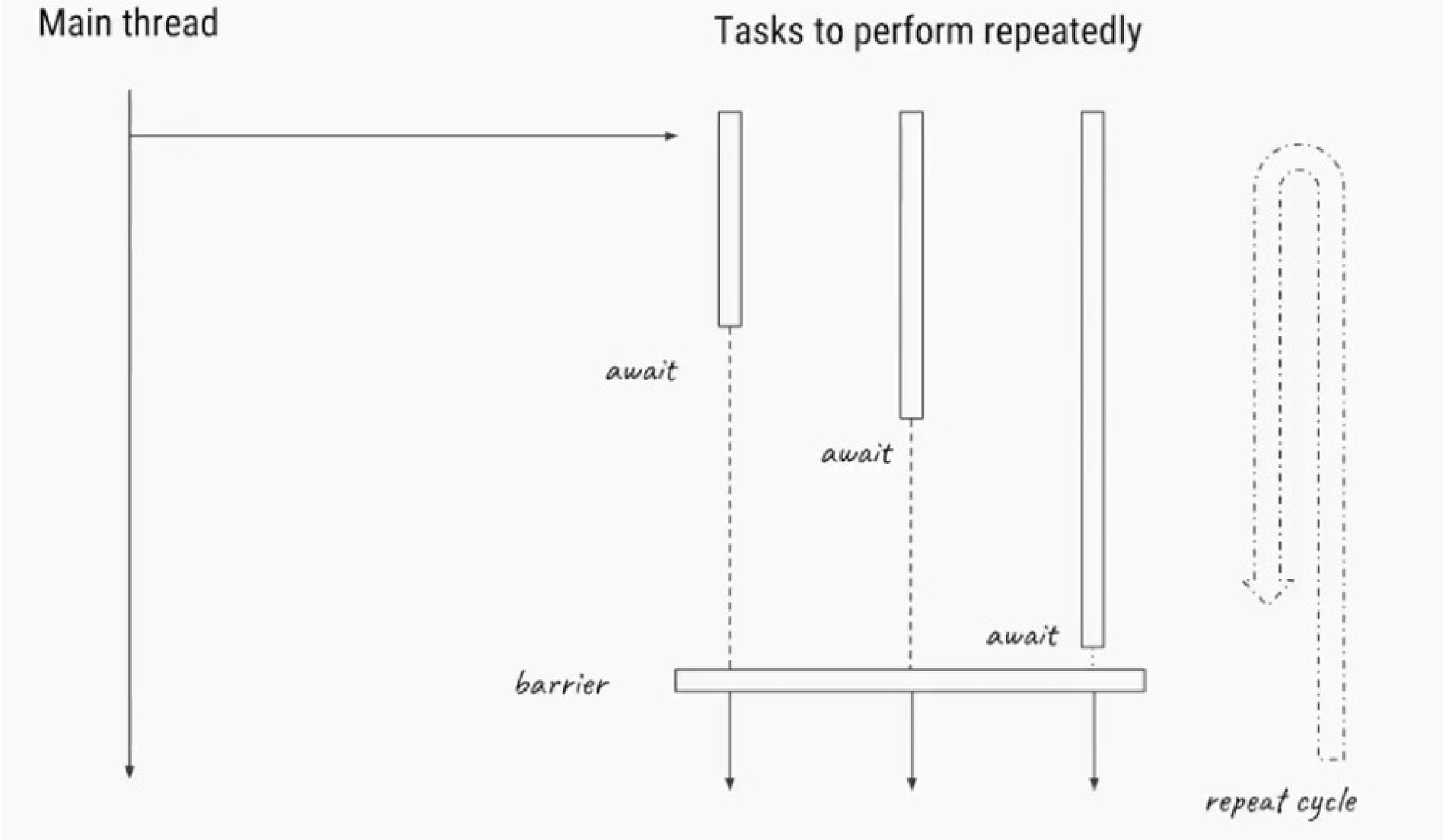
CyclicBarrier: Use cases

- Let we have a game of 3 player, we wnat to repeatedly send message to all 3 player without discrimination, the task run in infine loop (repedatly), all thread threds say barrier.await() (all can do in different time) , all three thread need to wait then when all arived then they continue processing

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

    ExecutorService executor = Executors.newFixedThreadPool( nThreads: 4);

    CyclicBarrier barrier = new CyclicBarrier( parties: 3);
    executor.submit(new Task(barrier));
    executor.submit(new Task(barrier));
    executor.submit(new Task(barrier));

    Thread.sleep( millis: 2000);
}

public static class Task implements Runnable {

    private CyclicBarrier barrier;
    public Task(CyclicBarrier barrier) { this.barrier = barrier; }

    @Override
    public void run() {

        while (true) {
            try {
                barrier.await();
            } catch (InterruptedException | BrokenBarrierException e) {
                e.printStackTrace();
            }
            // send message to corresponding system
        }
    }
```

rgupta.mtech@gmail.com

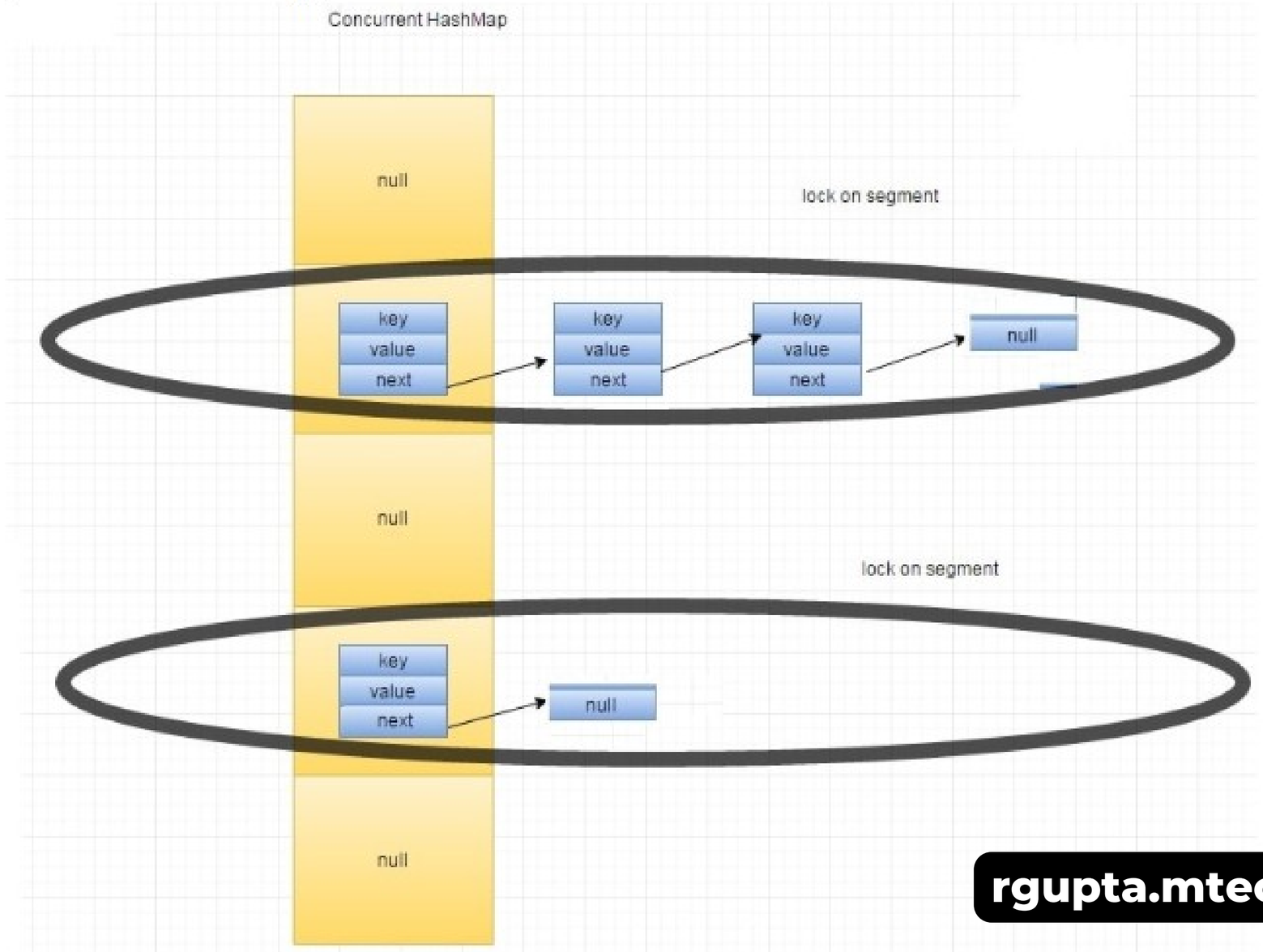
CyclicBarrier: Use cases



rgupta.mtech@gmail.com

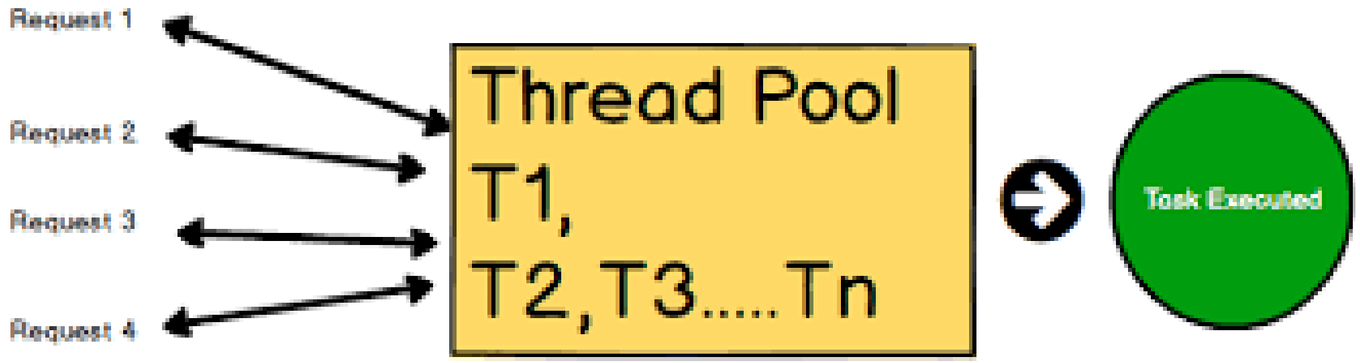
Thread safe DS java

ConcurrentHashMap-



rgupta.mtech@gmail.com

ExecutorService
Java Thread Pool framewrok



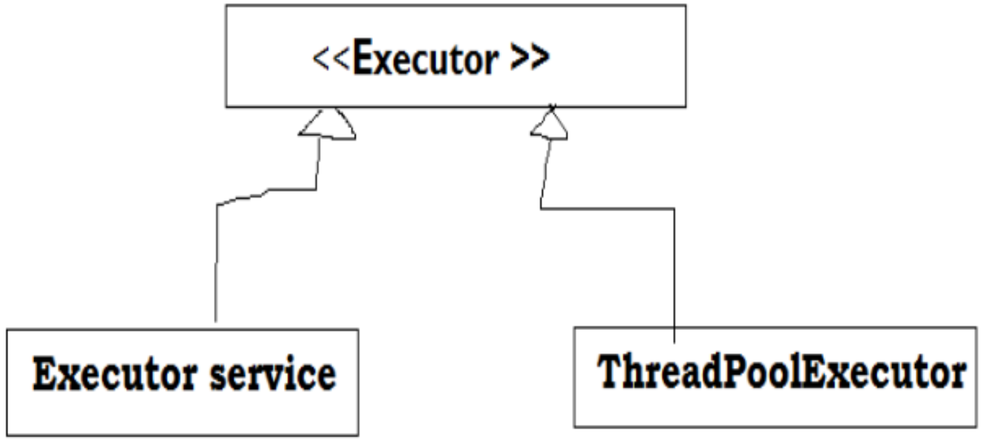
Thread Pooling

Thread Pools are useful when you **need** to limit the number of **threads** running in your application at the same time. ... Instead of starting a new **thread** for every task to execute concurrently, the task can be passed to a **thread pool**. As soon as the **pool** has any idle **threads** the task is assigned to one of them and executed. Oct 15, 2015

rgupta.mtech@gmail.com

Executor framework

- It separate task of creation and execution of thread
- We only have to implements Runnable (Make a job) and send it to executor
- Then executor is responsible for execution management
- Executor maintain an thread pool
  - avoid continuously spawning of threads



rgupta.mtech@gmail.com

Executor framework

Executors, A framework for creating and managing threads. Executors framework helps you with –

- **Thread Creation**: It provides various methods for creating threads, more specifically a pool of threads, that your application can use to run tasks concurrently.
- **Thread Management**: It manages the life cycle of the threads in the thread pool. You don't need to worry about whether the threads in the thread pool are active or busy or dead before submitting a task for execution.
- **Task submission and execution**: Executors framework provides methods for submitting tasks for execution in the thread pool, and also gives you the power to decide when the tasks will be executed. For example, You can submit a task to be executed now or schedule them to be executed later or make them execute periodically.

rgupta.mtech@gmail.com

Executor framework

Java Concurrency API defines the following three executor interfaces that covers everything that is needed for creating and managing threads –

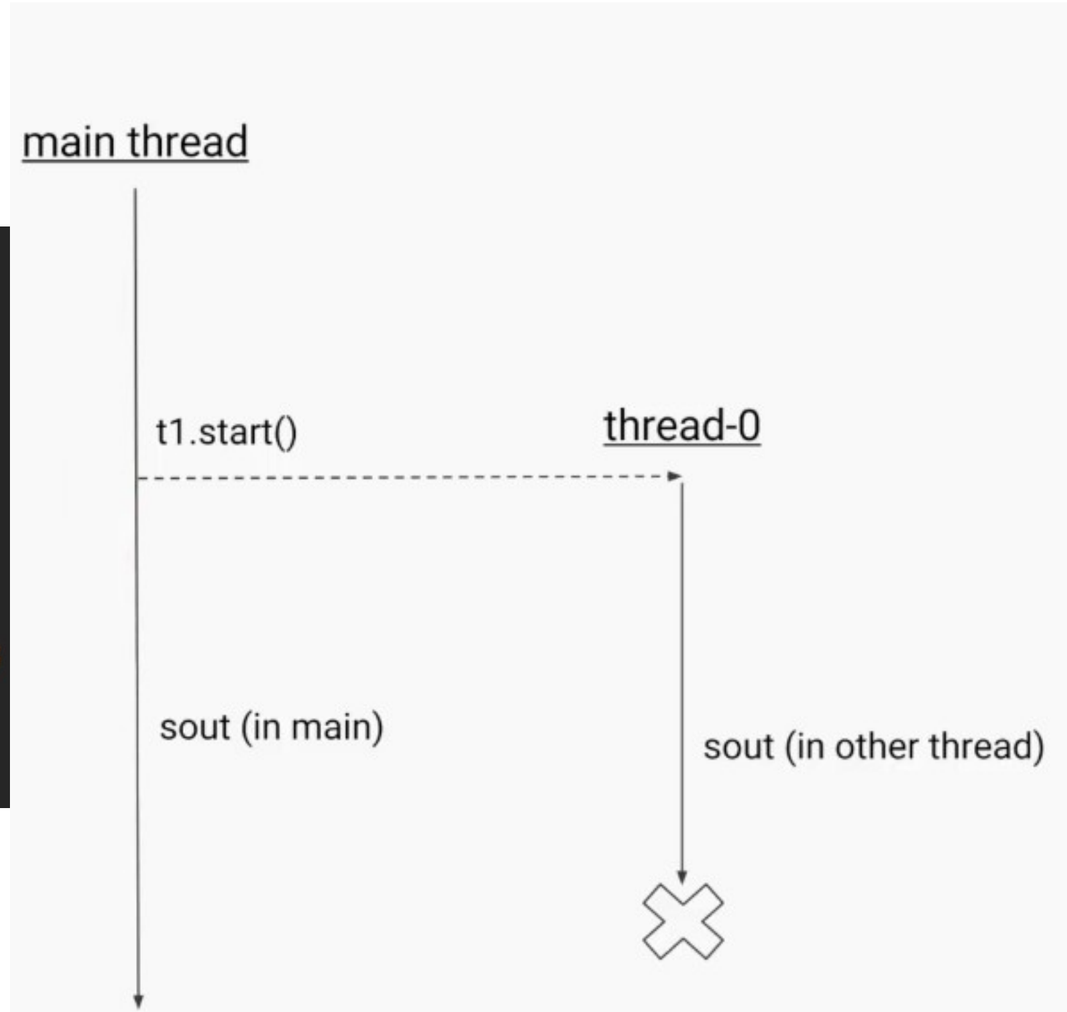
- **Executor** – A simple interface that contains a method called execute(Runnable command) to launch a task specified by a Runnable object.
- **ExecutorService** – A sub-interface of Executor that adds functionality to manage the lifecycle of the tasks. It also provides a submit() method whose overloaded versions can accept a Runnable as well as a Callable object. Callable objects are similar to Runnable except that the task specified by a Callable object can also return a value.
- **ScheduledExecutorService** – A sub-interface of ExecutorService. It adds functionality to schedule the execution of the tasks.

rgupta.mtech@gmail.com

Need of thread pool

Consider running a task asynchronously

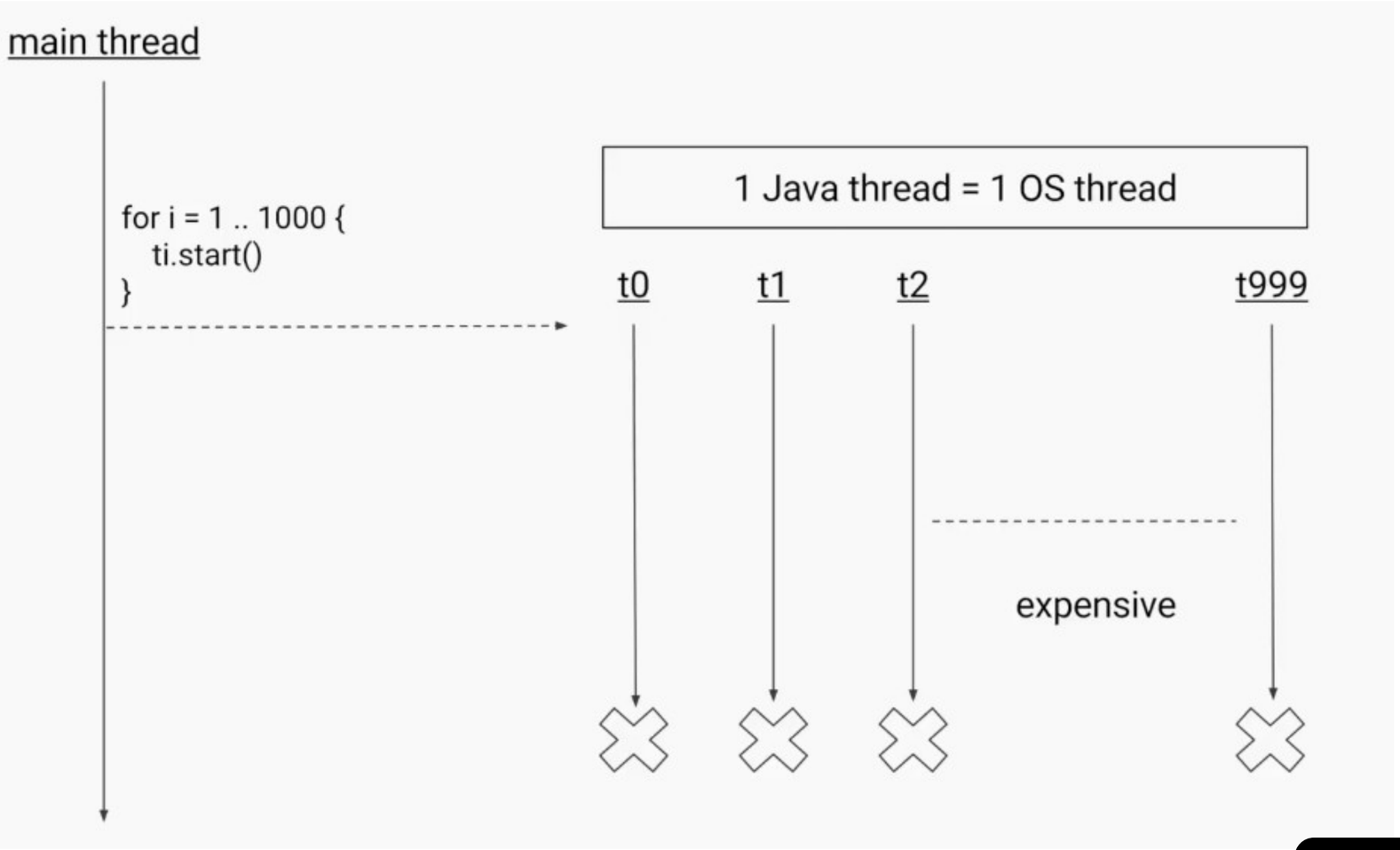
```
public static void main(String[] args) {

    Thread thread1 = new Thread(new Task());
    thread1.start();
    System.out.println("Thread Name: " + Thread.currentThread().getName());
}

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



rgupta.mtech@gmail.com

Need of thread pool

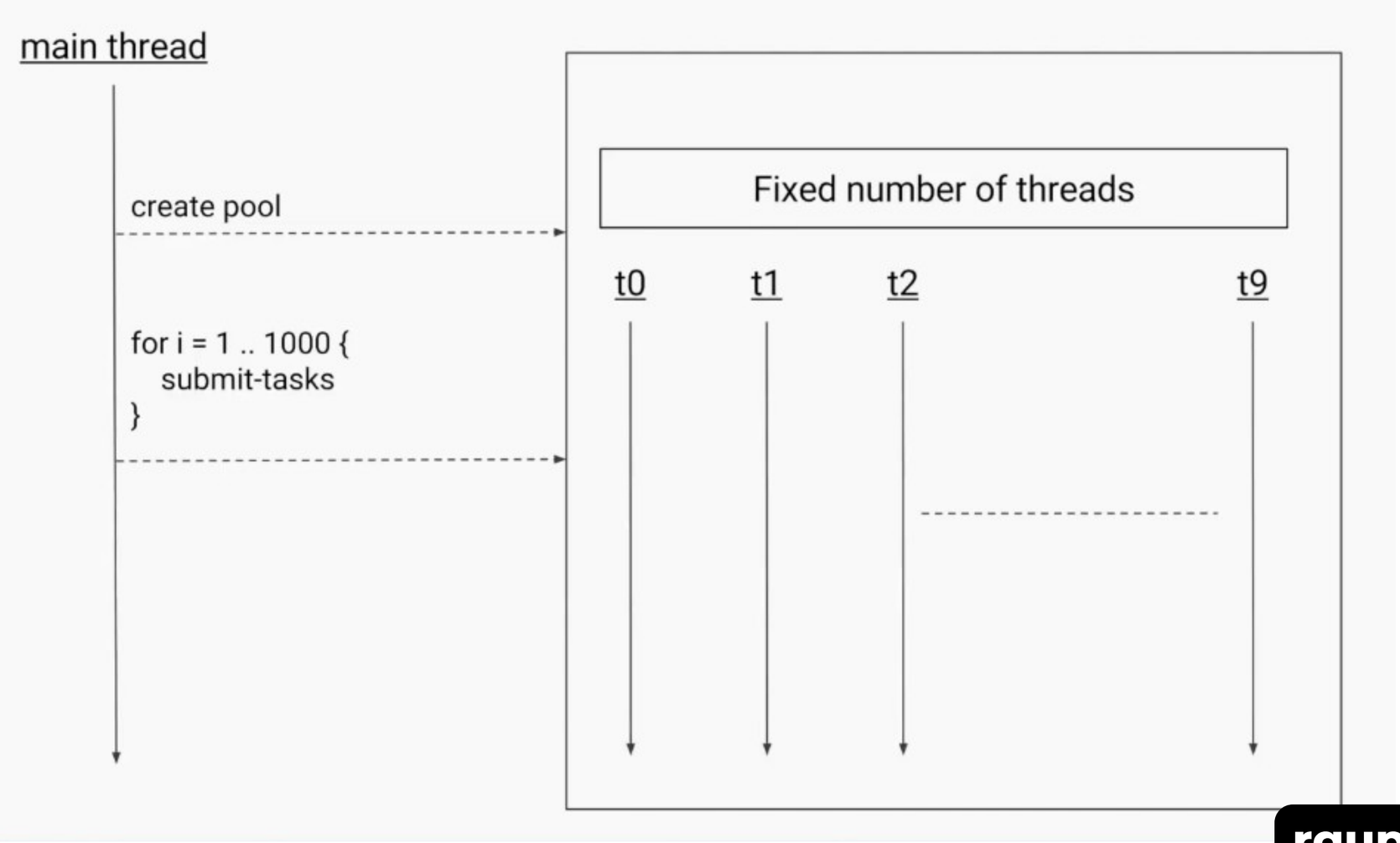
Consider running 1000 task asynchronously



rgupta.mtech@gmail.com

Need of thread pool

With Thread Pool we dont need to manually manage the thread life cycle and we need less number of threads



rgupta.mtech@gmail.com

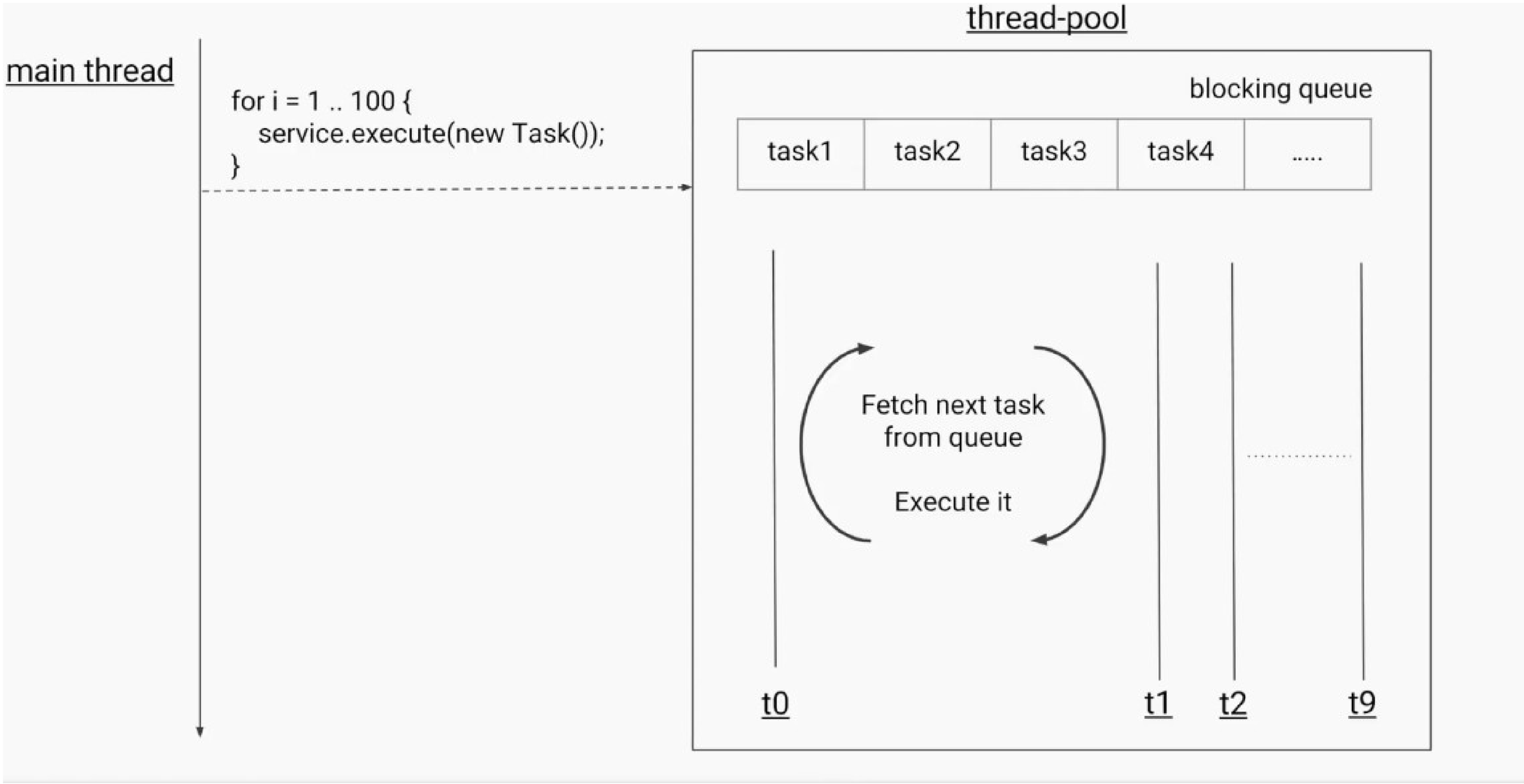
Submitting task using thread pool

Simplify code and better performance

```
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());
    }
}
```

rgupta.mtech@gmail.com

How Thread Pool works internally?



rgupta.mtech@gmail.com

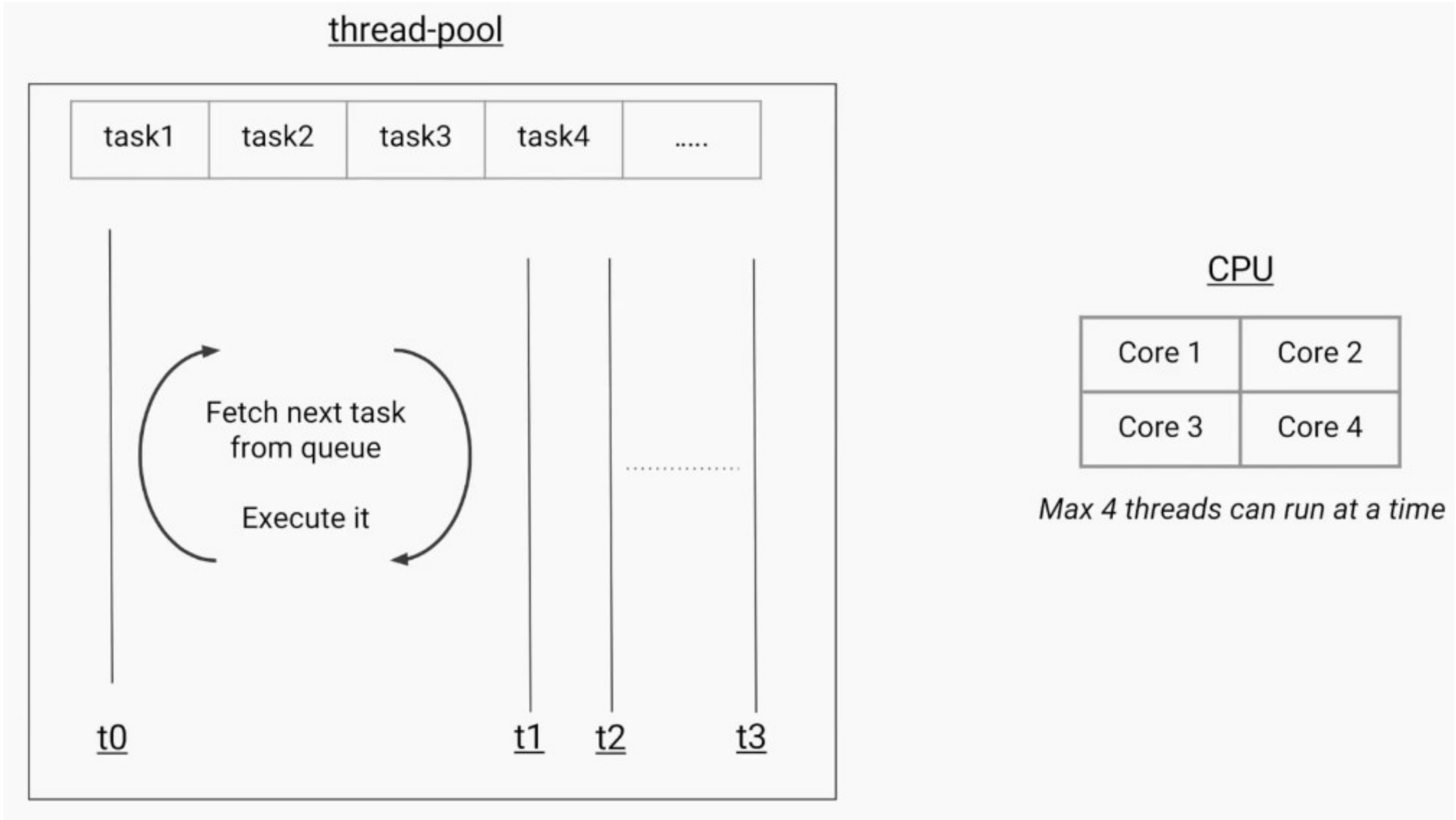
Type of Thread Pool

1. FixedThreadPool
2. CachedThreadPool
3. ScheduledThreadPool
4. SingleThreadedExecutor

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.

rgupta.mtech@gmail.com

CPU Intensive Operations



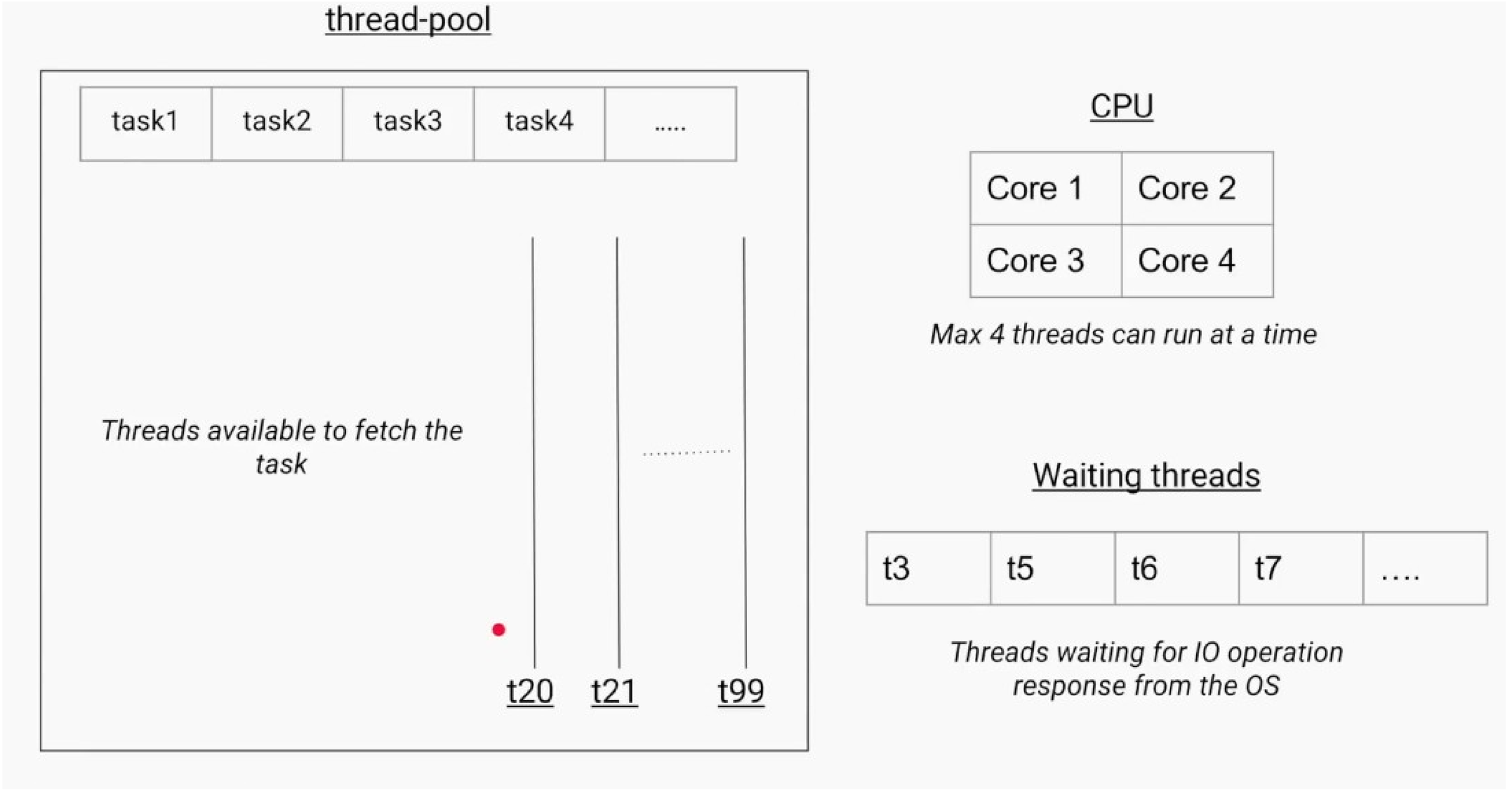
rgupta.mtech@gmail.com

CPU Intensive Operations

```
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
    }
}
```

rgupta.mtech@gmail.com

IO Intensive Operations



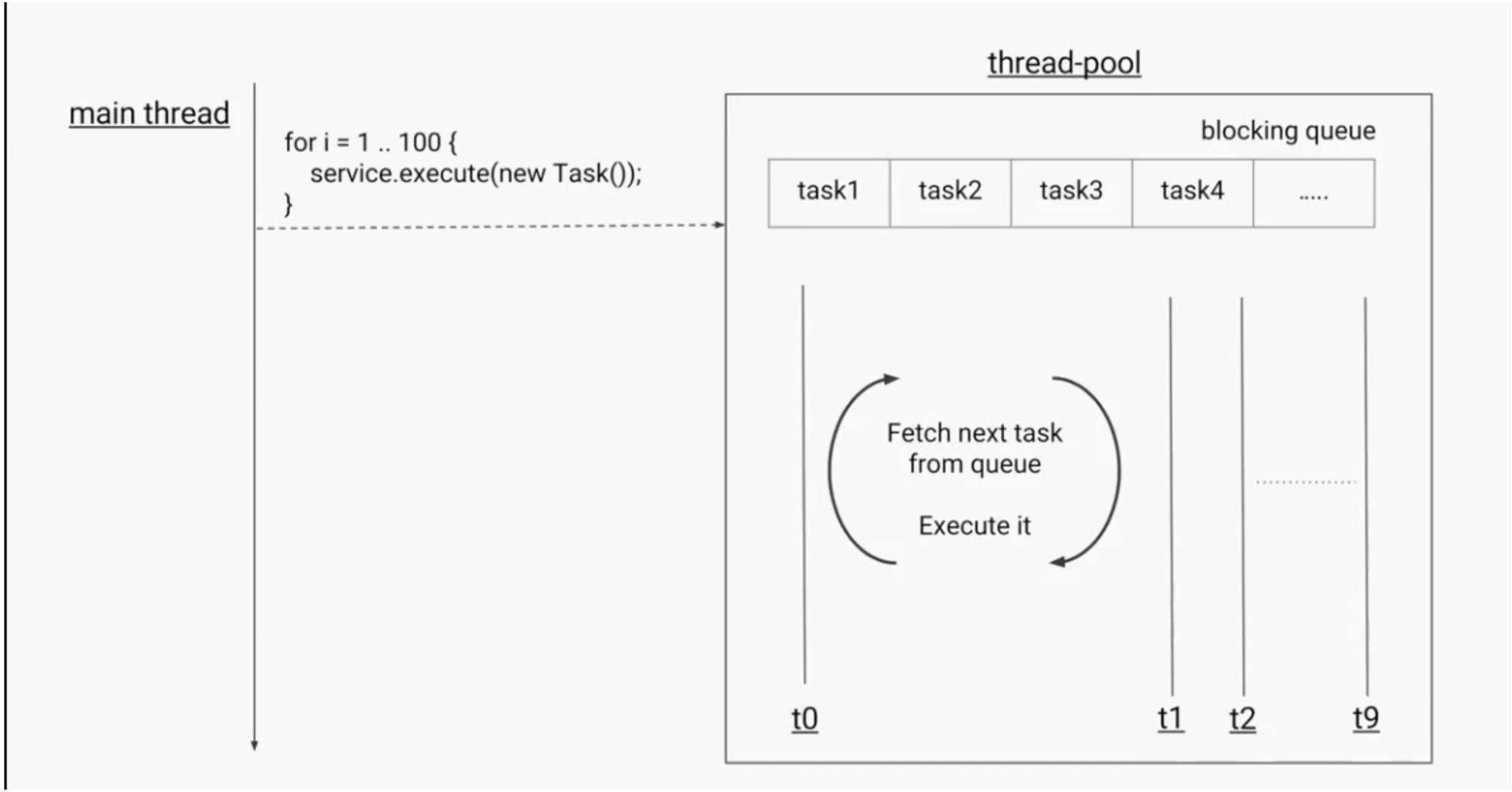
rgupta.mtech@gmail.com

IO Intensive Operations

```
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
    }
}
```

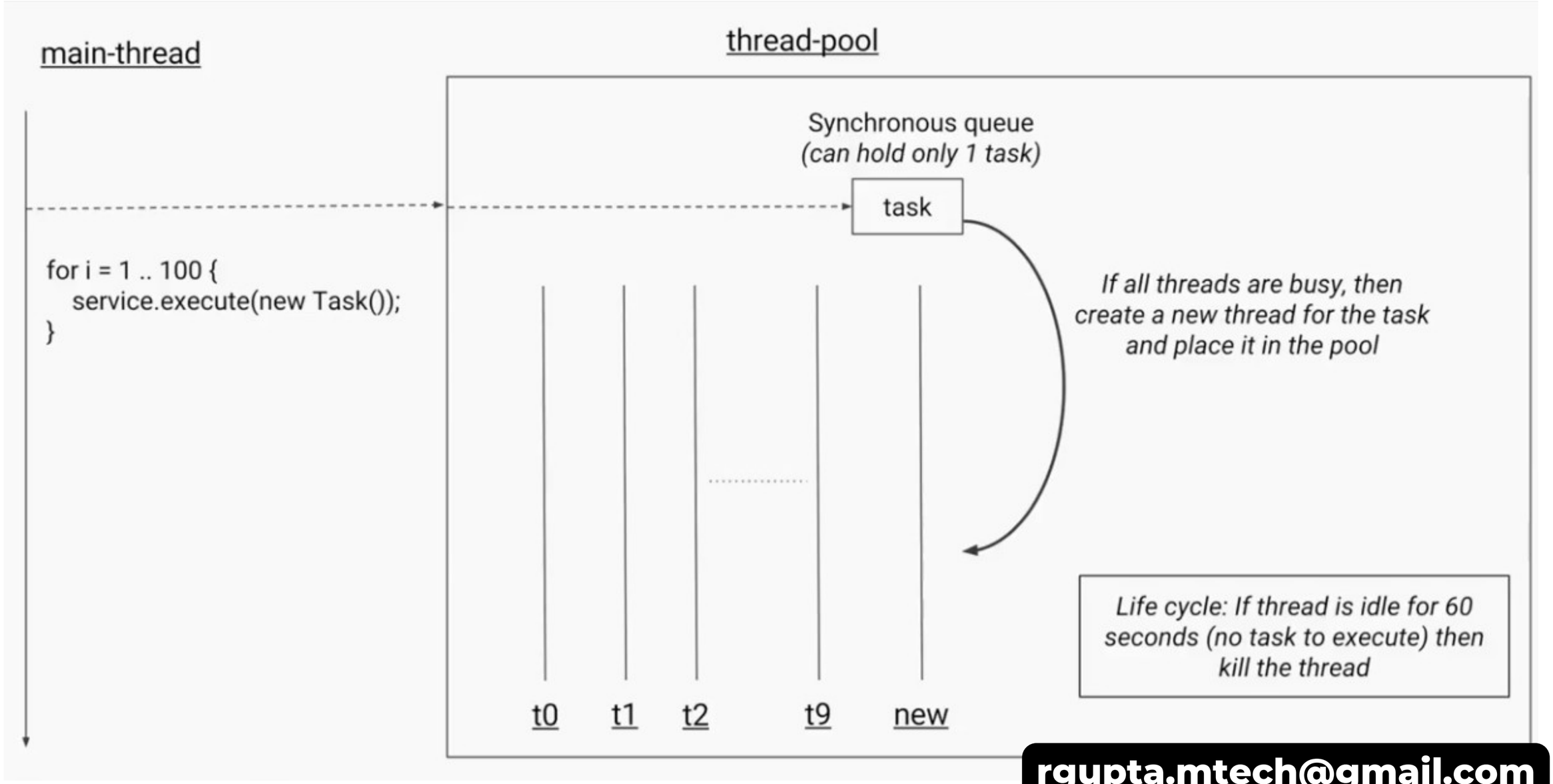
rgupta.mtech@gmail.com

How Thread Pool works Internally?



rgupta.mtech@gmail.com

Cached Thread Pool



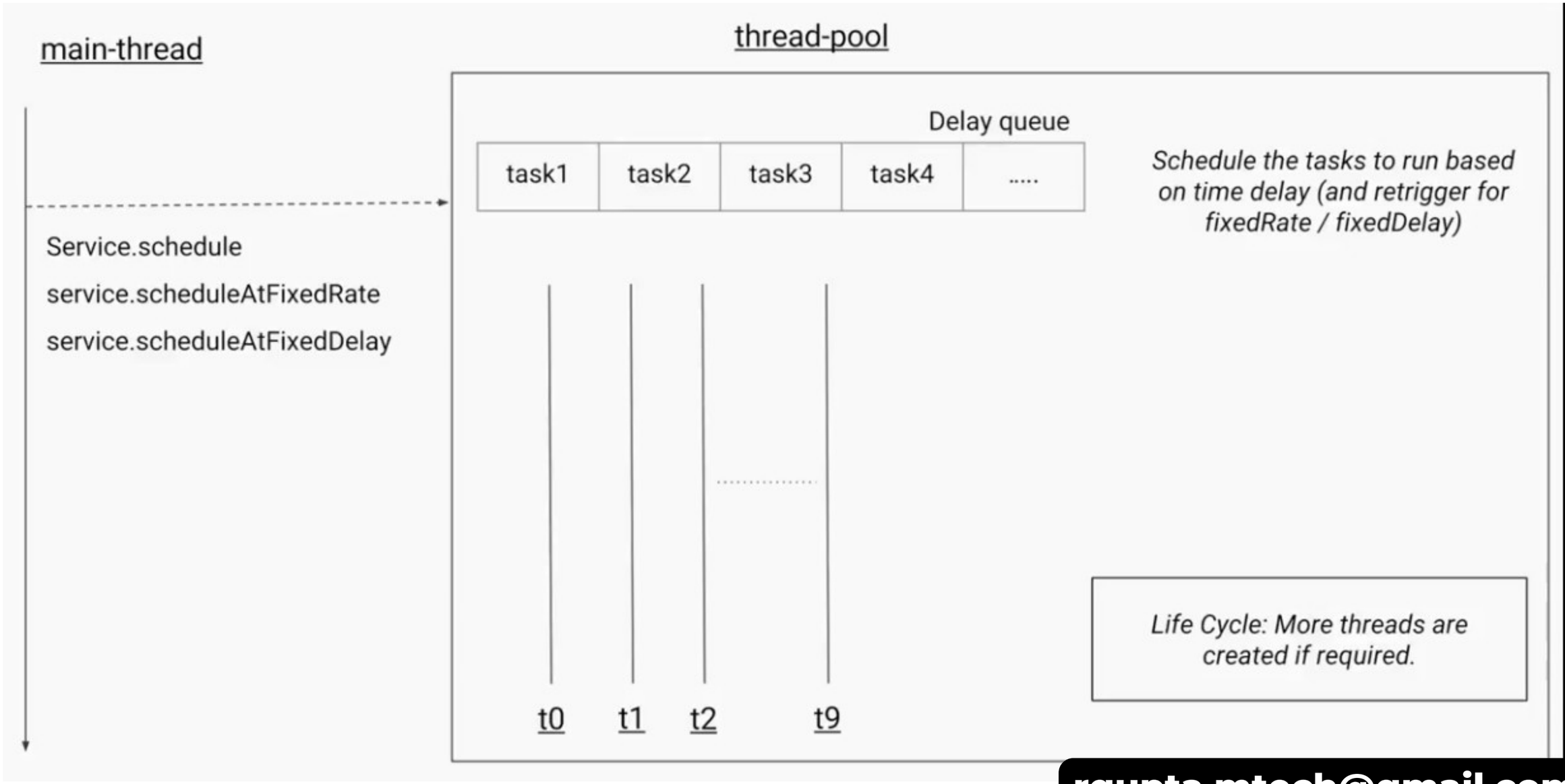
rgupta.mtech@gmail.com

Cached Thread Pool

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

    // for lot of short lived tasks
    ExecutorService service = Executors.newCachedThreadPool();

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

static class Task implements Runnable {
    public void run() {
        // short lived task
    }
}
```

rgupta.mtech@gmail.com

Scheduled Thread Pool



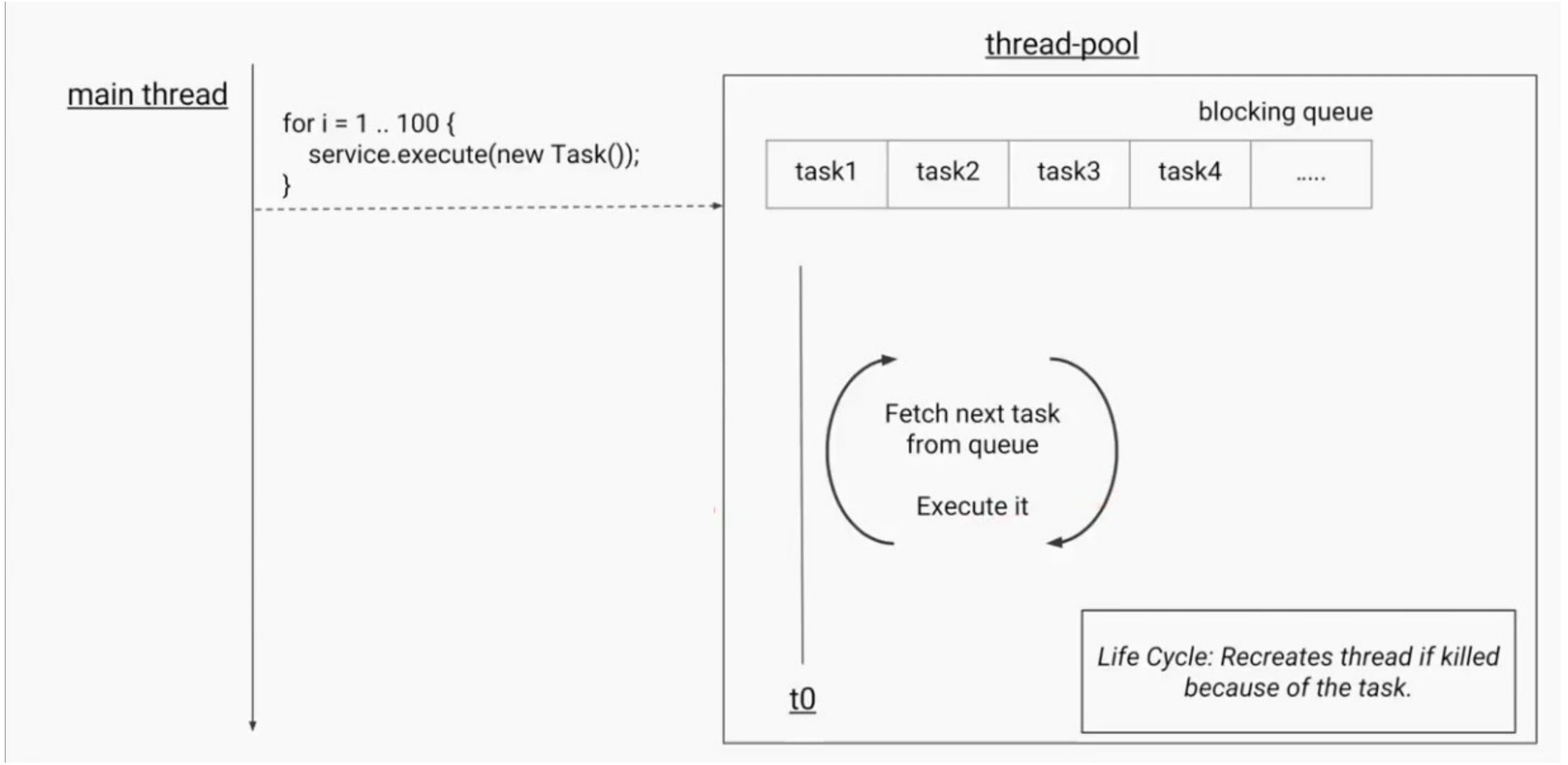
rgupta.mtech@gmail.com

Scheduled Thread Pool

```
blic static void main(String[] args) {

  // for scheduling of tasks
  ScheduledExecutorService service = Executors.newScheduledThreadPool( corePoolSize: 10);

  // task to run after 10 second delay
  service.schedule(new Task(), delay: 10, SECONDS);

  // task to run repeatedly every 10 seconds
  service.scheduleAtFixedRate(new Task(), initialDelay: 15, period: 10, SECONDS);

  // task to run repeatedly 10 seconds after previous task completes
  service.scheduleWithFixedDelay(new Task(), initialDelay: 15, delay: 10, TimeUnit.SECONDS);


atic class Task implements Runnable {
  public void run() {
      // task that needs to run
      // based on schedule
  }
```

rgupta.mtech@gmail.com

Single Thread Executor



rgupta.mtech@gmail.com

Thread Pool: Life Cycle Methods

```java
ExecutorService service = Executors.newFixedThreadPool( nThreads: 10);
for (int i = 0; i < 100; i++) {
    service.execute(new Task());
}

// initiate shutdown
service.shutdown();

// will throw RejectionExecutionException
// service.execute(new Task());

// will return true, since shutdown has begun
service.isShutdown();

// will return true if all tasks are completed
// including queued ones
service.isTerminated();

// block until all tasks are completed or if timeout occurs
service.awaitTermination( timeout: 10, TimeUnit.SECONDS);

// will initiate shutdown and return all queued tasks
List<Runnable> runnables = service.shutdownNow();
```

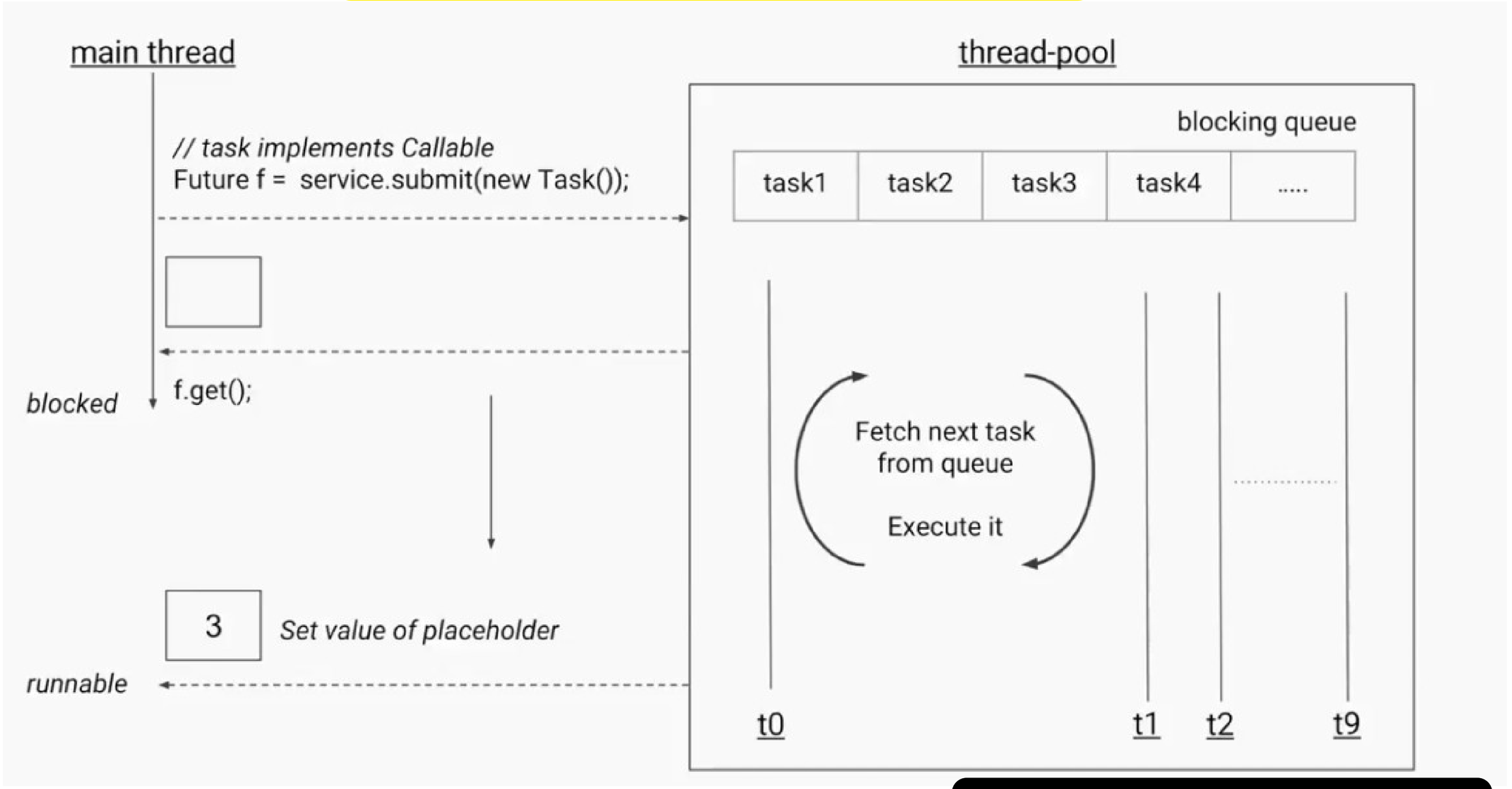
rgupta.mtech@gmail.com

Thread Pool: Examples

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

// submit the tasks for execution
List<Future> allFutures = new ArrayList<>();
for (int i = 0; i < 100; i++) {
    Future<Integer> future = service.submit(new Task());
    allFutures.add(future);
}

// 100 futures, with 100 placeholders.

// perform some unrelated operations

// 1 sec
try {
    Integer result = future.get(); // blocking
} catch (InterruptedException e) {
    e.printStackTrace();
} catch (ExecutionException e) {
    e.printStackTrace();
}
```

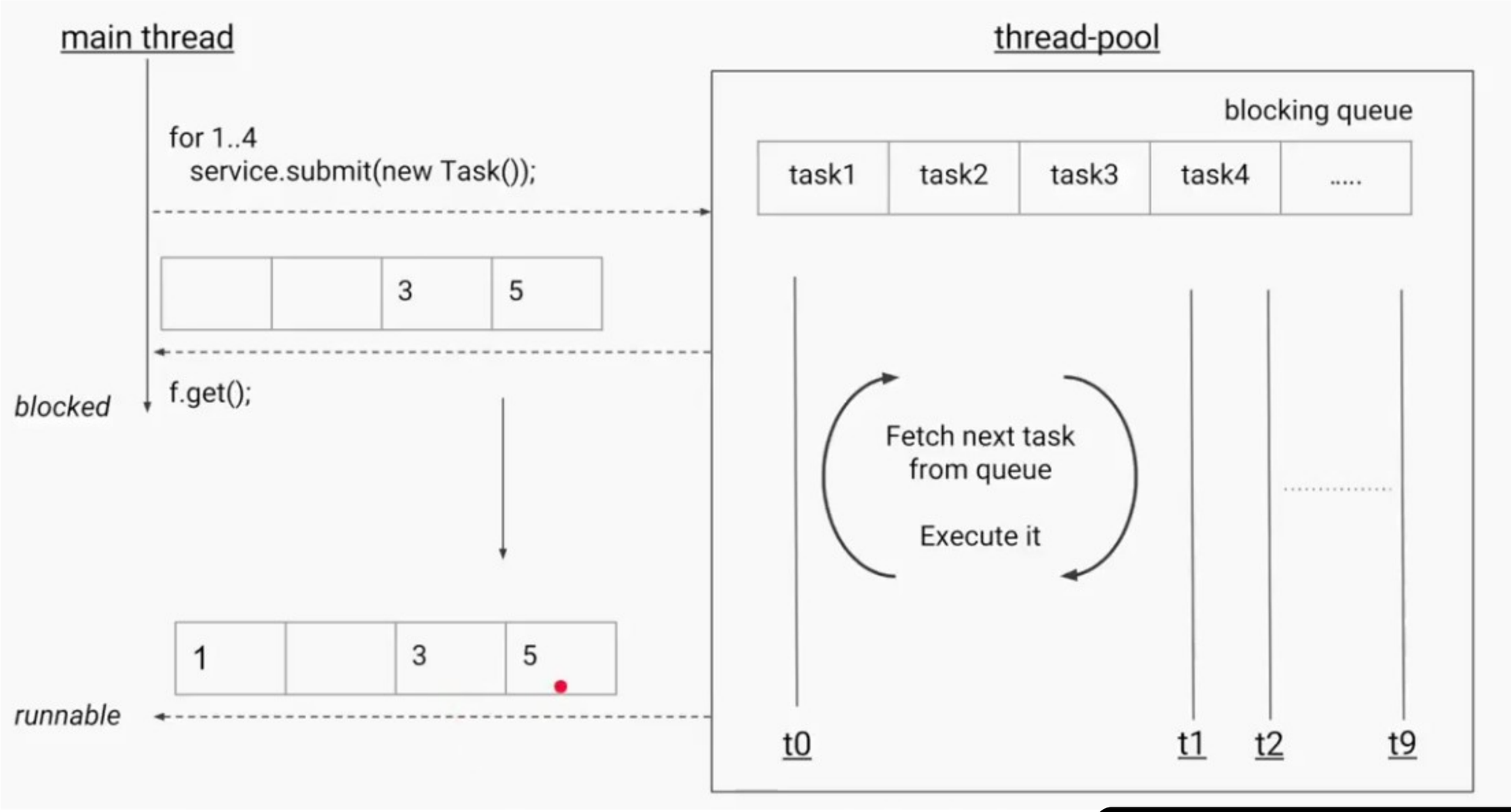
rgupta.mtech@gmail.com

Thread Pool Future How it works?



rgupta.mtech@gmail.com

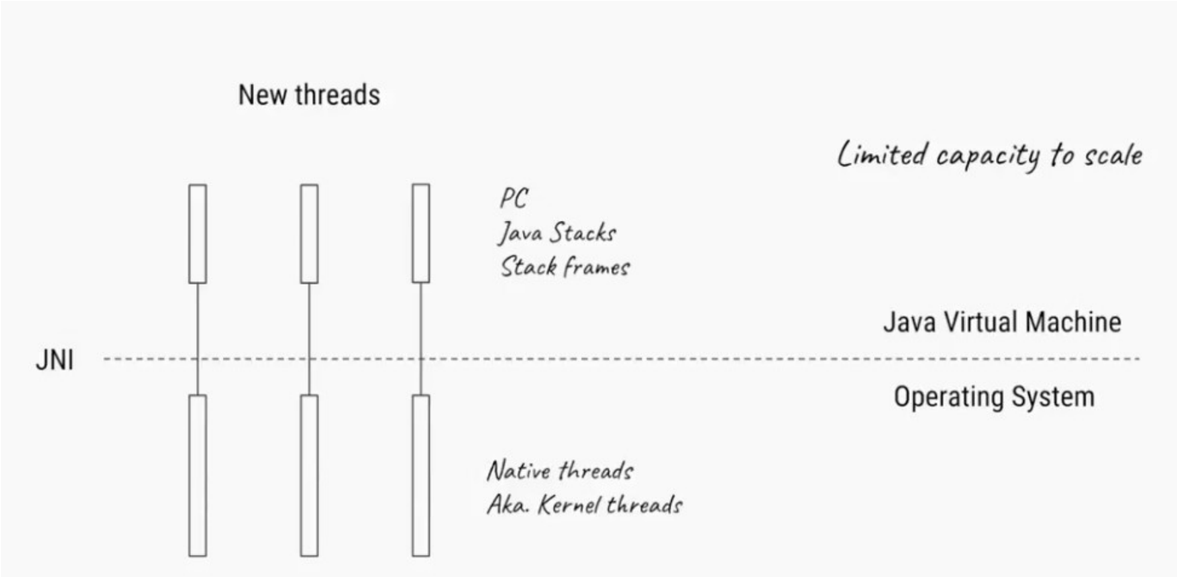
Thread pool: Blocked till all work is not done!



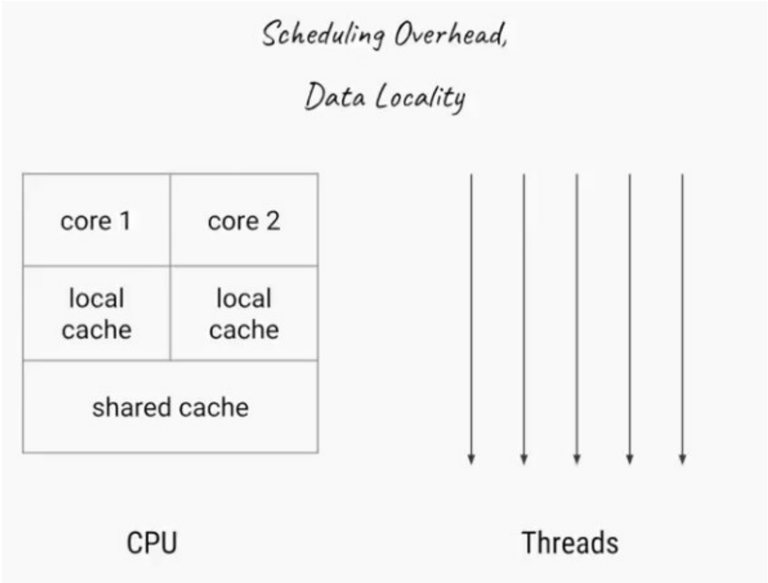
rgupta.mtech@gmail.com

CompletableFuture

Java Thread= Native Thread



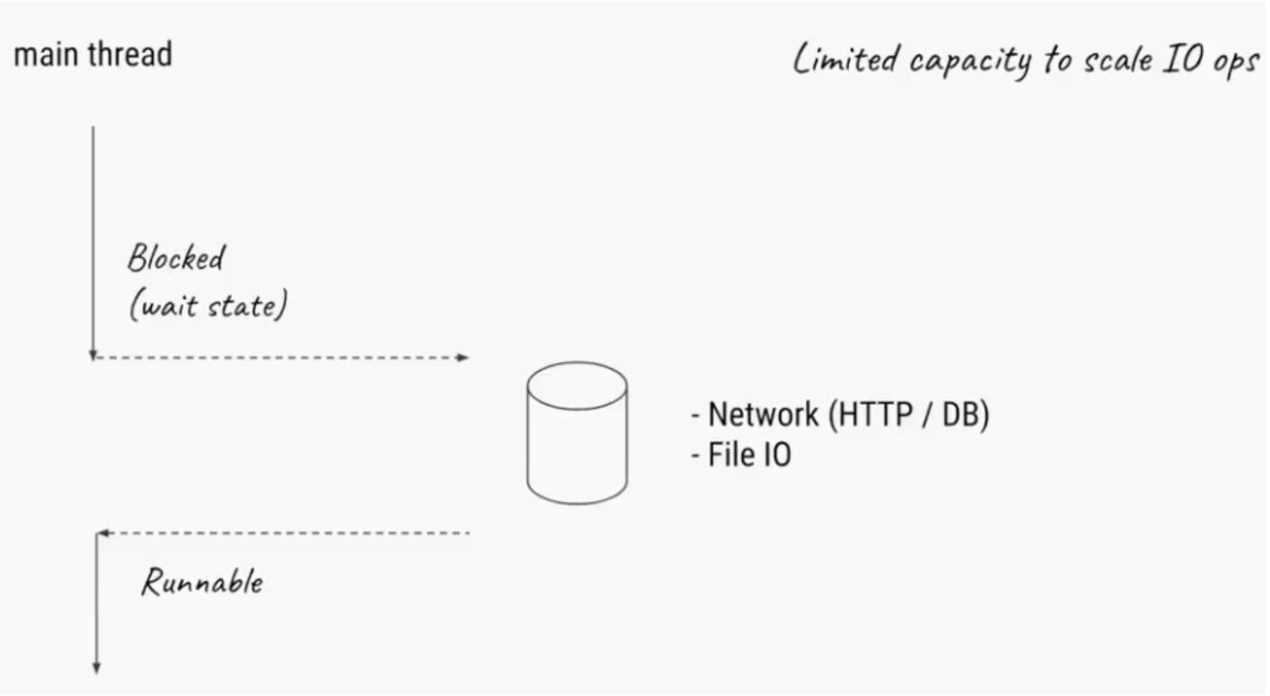
Other Issues



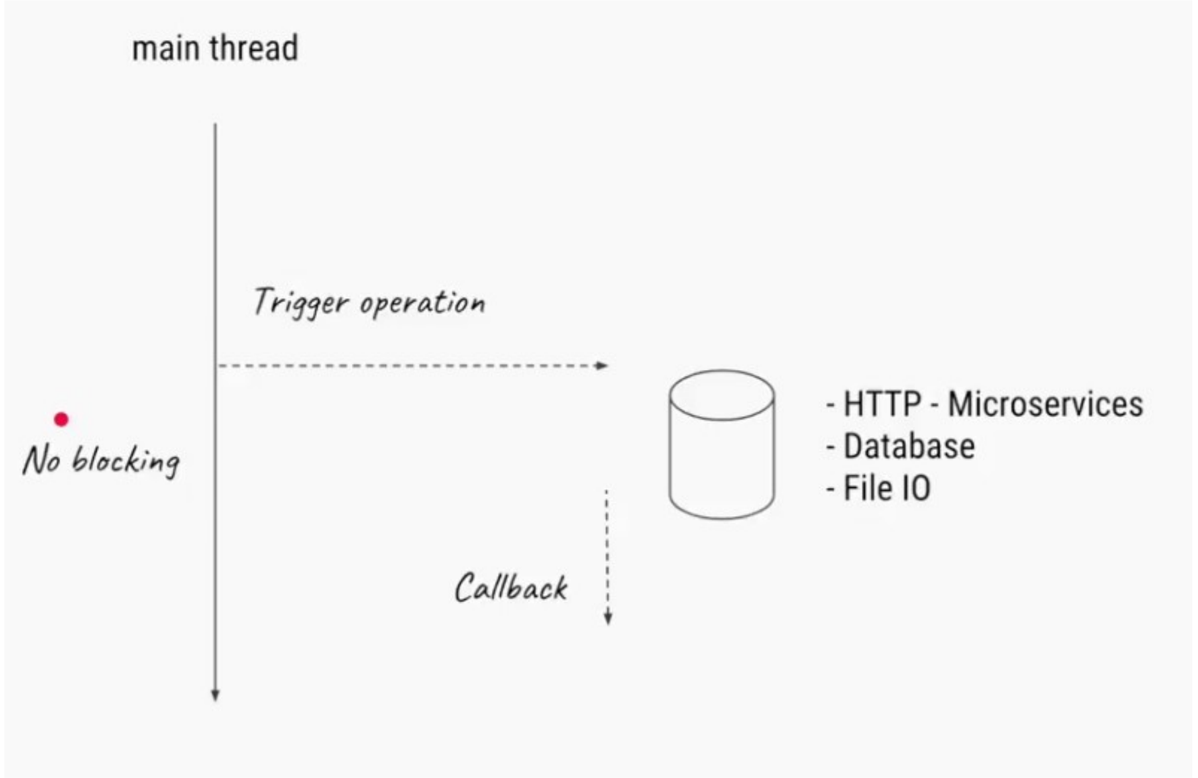
rgupta.mtech@gmail.com

CompletableFuture

IO Operation need more threads



None Blocking IO



rgupta.mtech@gmail.com

Synchronous vs Asyn API

```java
for (Integer id : employeeIds) {

    // Step 1: Fetch Employee details from DB
    Future<Employee> future = service.submit(new EmployeeFetcher(id));
    Employee emp = future.get(); // blocking

    // Step 2: Fetch Employee tax rate from REST service
    Future<TaxRate> rateFuture = service.submit(new TaxRateFetcher(emp));
    TaxRate taxRate = rateFuture.get(); // blocking

    // Step 3: Calculate current year tax
    BigDecimal tax = calculateTax(emp, taxRate);

    // Step 4: Send email to employee using REST service
    service.submit(new SendEmail(emp, tax));
}
```

```java
for (Integer id : employeeIds) {
    CompletableFuture.supplyAsync(() -> fetchEmployee(id))
            .thenApplyAsync(employee -> fetchTaxRate(employee))
            .thenApplyAsync(taxRate -> calculateTax(taxRate))
            .thenAcceptAsync(taxValue -> sendEmail(taxValue));
}
```

Callback chaining (similar to JS)

rgupta.mtech@gmail.com

Servlet 3.0 Allow more concurrent requests

```java
@WebServlet(urlPatterns={"/user"}, asyncSupported=true)
public class UserAsyncServlet extends HttpServlet {

    @Override
    public void doGet(HttpServletRequest request,
                      HttpServletResponse response) {

        final AsyncContext context = request.startAsync();
        context.start(new Runnable() {
            public void run() {
                // make the network call
                ServletResponse response = context.getResponse();
                // print to the response
                context.complete();
            }
        });
    }
}
```

Run operations in separate thread

Immediately return to serve other requests

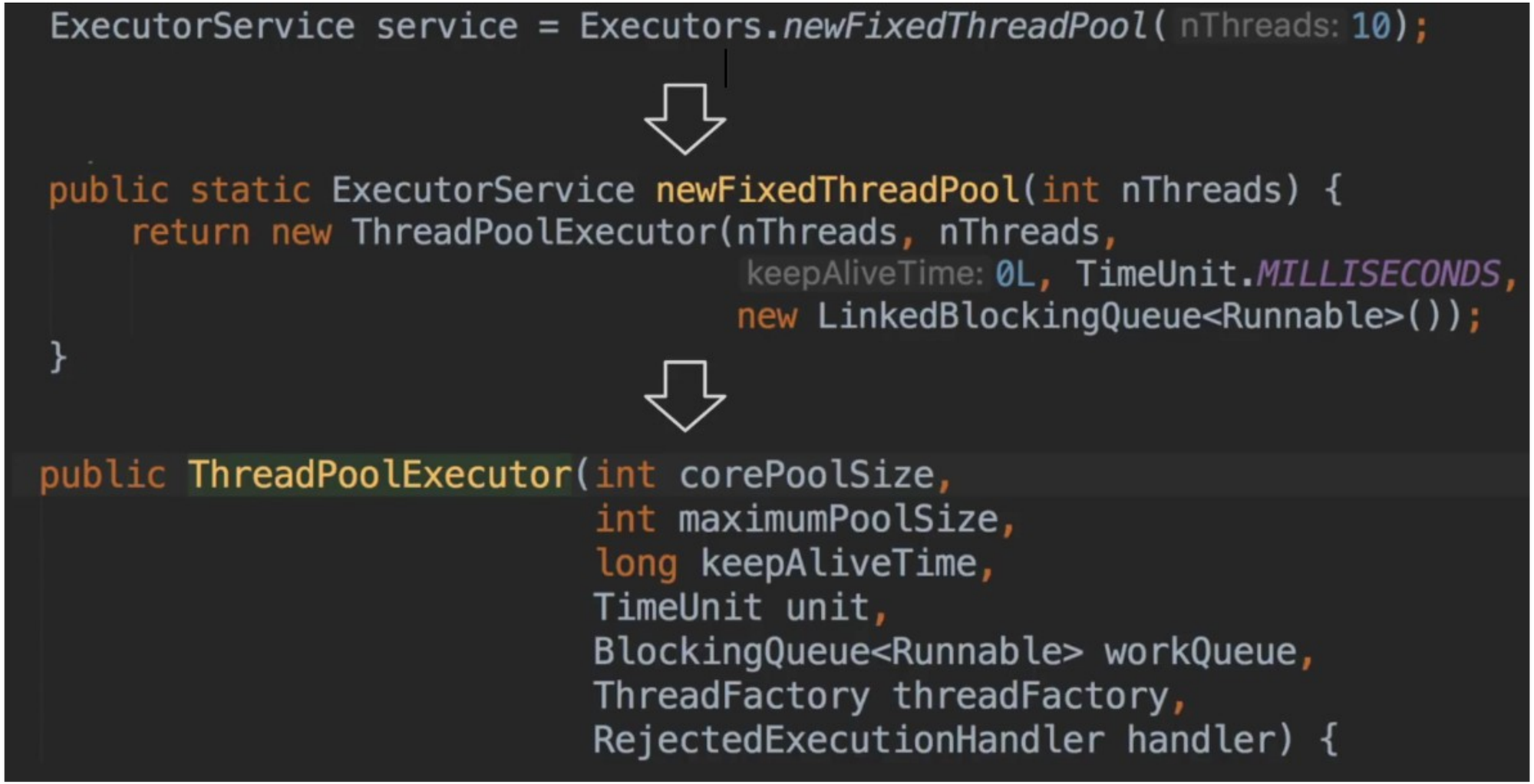
rgupta.mtech@gmail.com

Spring Reactive Programming

```java
@RestController
public class MyRestController {

    @Autowired
    private UserRepository userRepository;

    @GetMapping("/get/user/{id}")
    public Mono<User> getUser(@PathVariable String id) {
        return userRepository.findUser(id);
    }
```

rgupta.mtech@gmail.com

ThreadPoolExecutor Constructor:customization



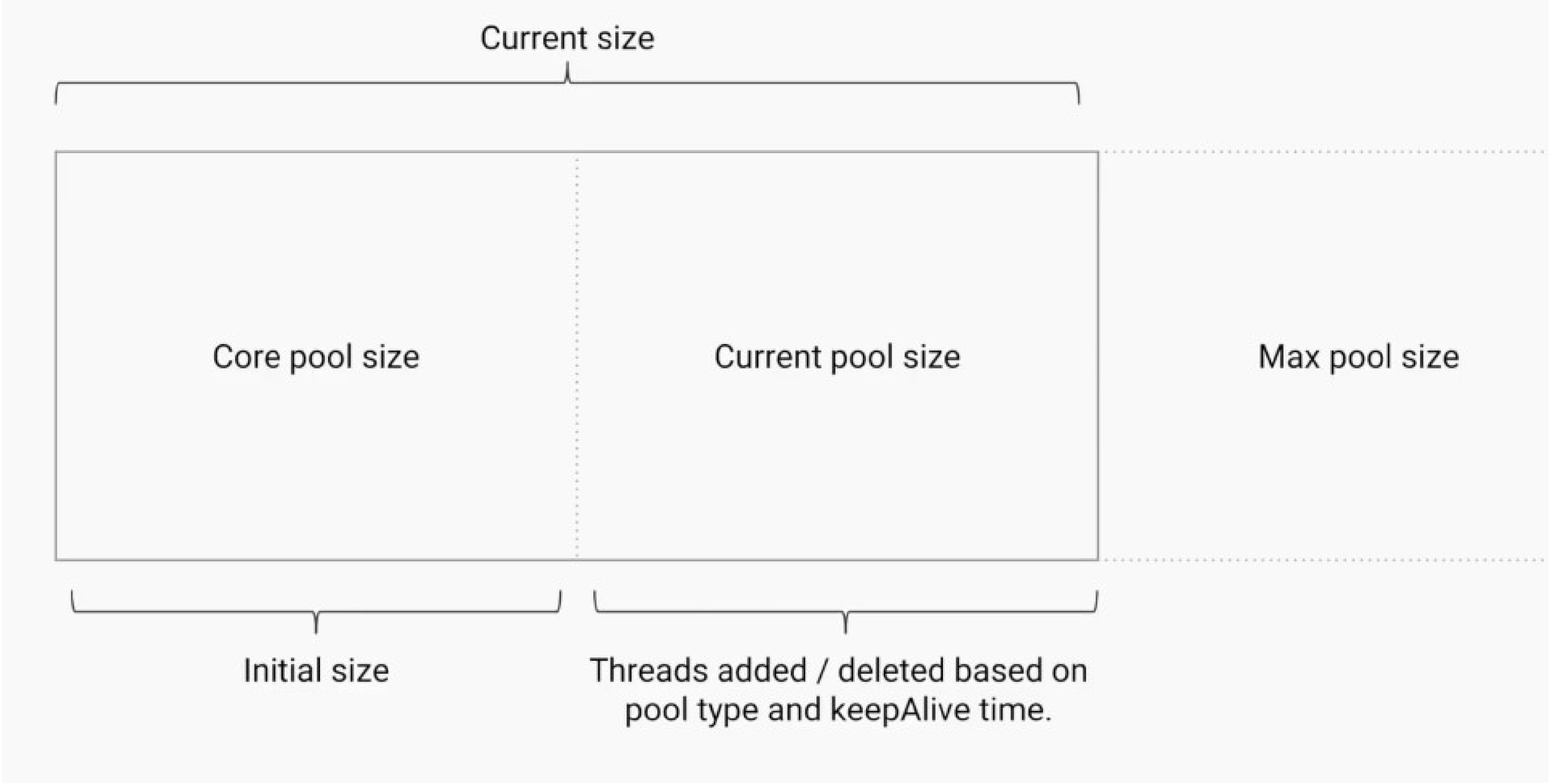
rgupta.mtech@gmail.com

ThreadPoolExecutor 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

rgupta.mtech@gmail.com

ThreadPoolExecutor :understanding pool size



rgupta.mtech@gmail.com

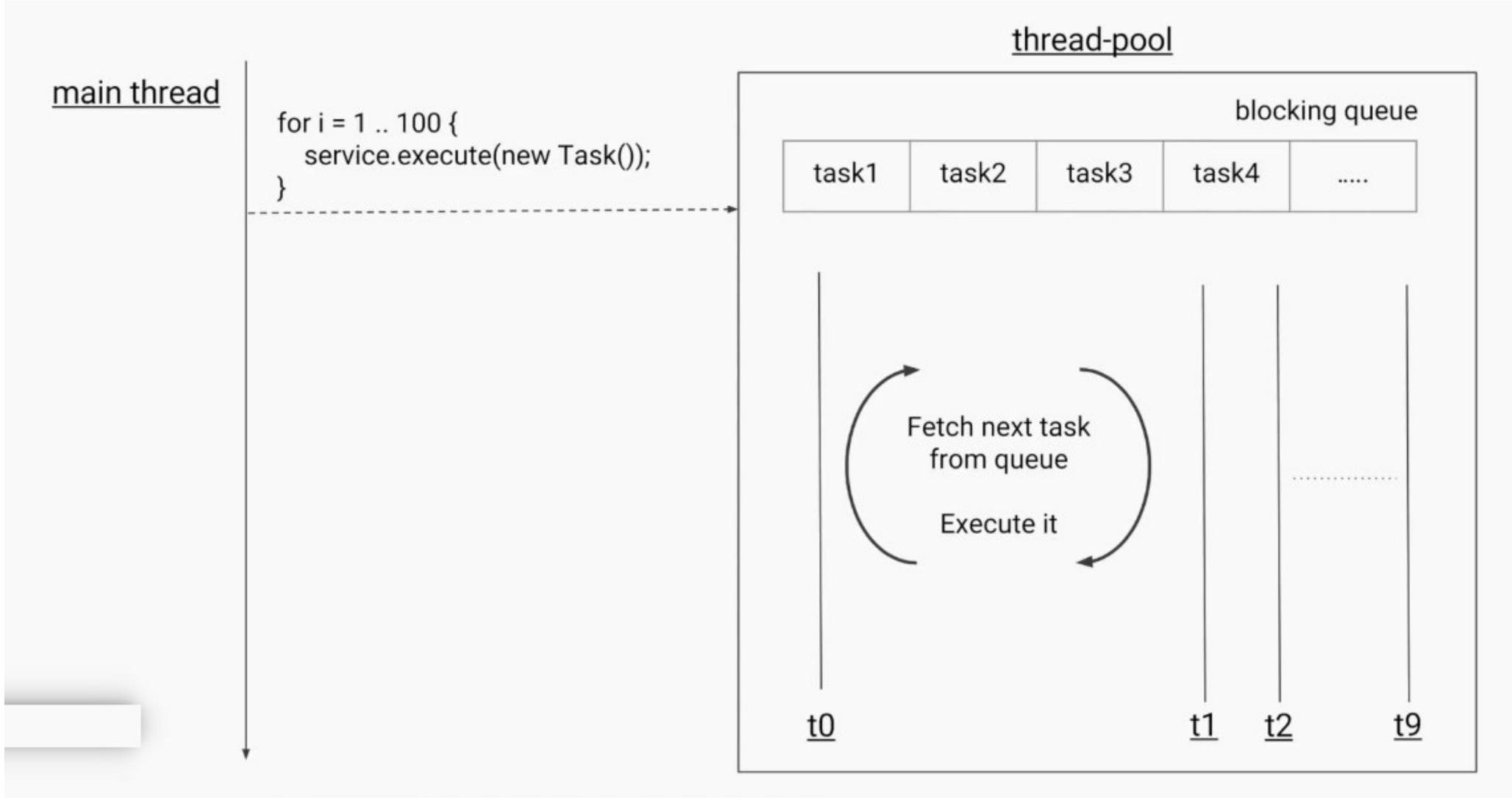
ThreadPoolExecutor :understanding pool size

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.

rgupta.mtech@gmail.com

Type of Queues used in thread pool



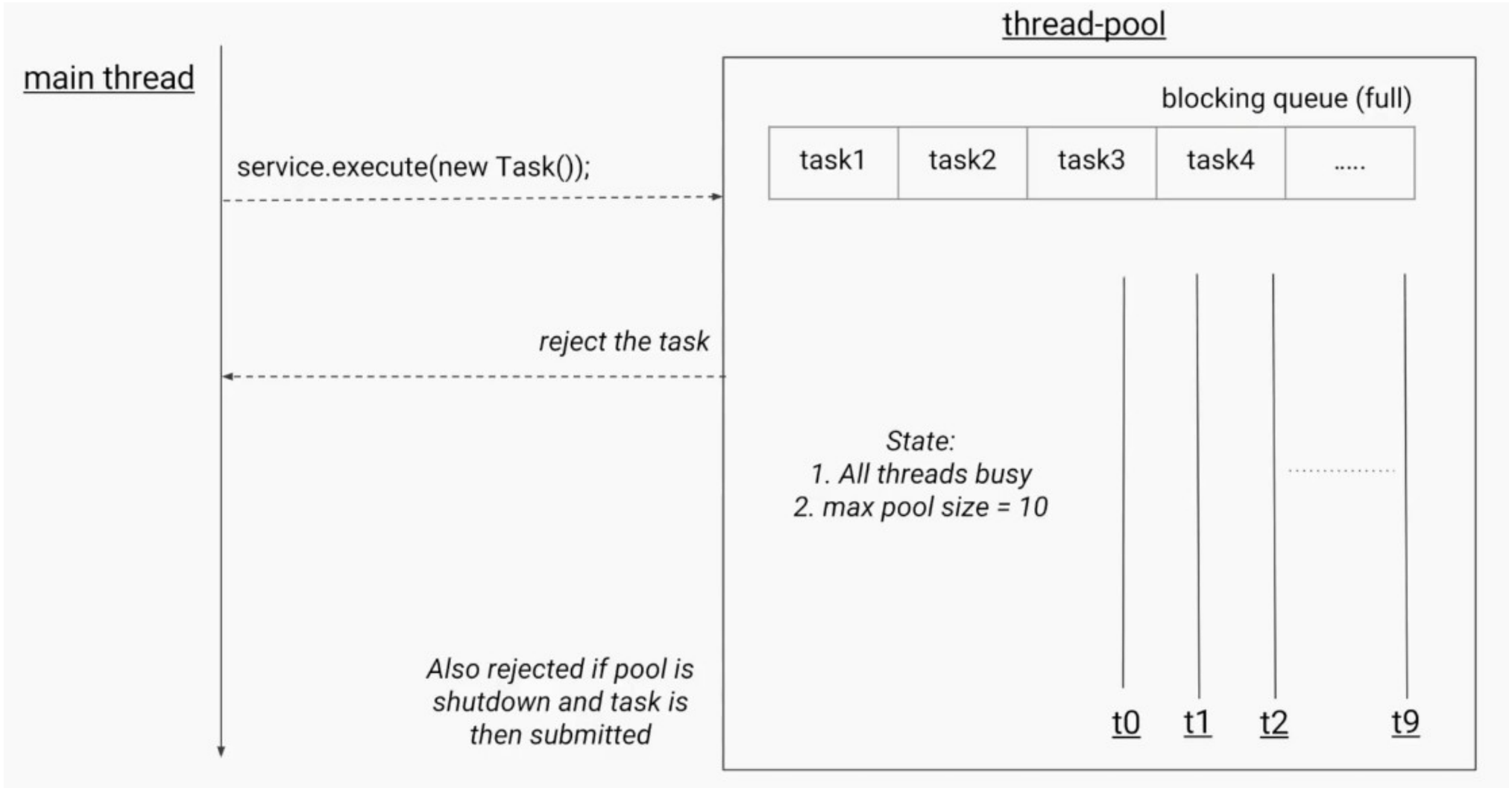
rgupta.mtech@gmail.com

ThreadPoolExecutor :understanding Queue types

Pool	Queue Type	Why?
FixedThreadPool	LinkedBlockingQueue	Threads are limited, thus unbounded queue to store all tasks. *Note: Since queue can never become full, new threads are never created.*
SingleThreadExecutor	LinkedBlockingQueue	
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).

rgupta.mtech@gmail.com

Thread pool: handling rejections



rgupta.mtech@gmail.com

thread pool: rejection policies

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.

rgupta.mtech@gmail.com

thread pool: RejectedExecutionException

```
ExecutorService service
        = new ThreadPoolExecutor(
                corePoolSize: 10,
                maximumPoolSize: 100,
                keepAliveTime: 120, TimeUnit.SECONDS,
                new ArrayBlockingQueue<>( capacity: 300));

try {
    service.execute(new Task());
} catch (RejectedExecutionException e) {
    System.err.println("task rejected " + e.getMessage());
}
```

rgupta.mtech@gmail.com

thread pool: RejectedExecutionException

```
ExecutorService service
        = new ThreadPoolExecutor(
                corePoolSize: 10,
                maximumPoolSize: 100,
                keepAliveTime: 120, TimeUnit.SECONDS,
                new ArrayBlockingQueue<>( capacity: 300),
                new CustomRejectionHandler());


private static class CustomRejectionHandler implements RejectedExecutionHandler{
    @Override
    public void rejectedExecution(Runnable r, ThreadPoolExecutor executor) {
        // logging / operations to perform on rejection
    }
}
```

rgupta.mtech@gmail.com

Fork-Join framework

Started with JSR 166 efforts

JDK 7 includes it in java.util.concurrent
Can be used with JDK 6.

Download jsr166y package maintained by Doug Lea
http://gee.cs.oswego.edu/dl/concurrency-interest/

rgupta.mtech@gmail.com

Fork Join Framework

Parallel version of Divide and Conquer

1.Recursively break down the problem into sub problems

2.Solve the sub problems in parallel

3.Combine the solutions to sub-problems to arrive at final result

```
// PSEUDOCODE
Result solve(Problem problem) {
    if (problem.size < SEQUENTIAL_THRESHOLD)
        return solveSequentially(problem);
    else {
        Result left, right;
        INVOKE-IN-PARALLEL {
            left = solve(extractLeftHalf(problem));
            right = solve(extractRightHalf(problem));
        }
        return combine(left, right);
    }
}
```

rgupta.mtech@gmail.com

Selecting a max number

```
public class SelectMaxProblem {
    private final int[] numbers;
    private final int start;
    private final int end;
    public final int size;

    public SelectMaxProblem(int[] numbers2, int i, int j) {
        this.numbers = numbers2;
        this.start = i;
        this.end = j;
        this.size = j - i;
        System.out.println("start:" + start + ",end:" + end + ",size:" + size);
    }

    public int solveSequentially() {
        int max = Integer.MIN_VALUE;
        for (int i=start; i<end; i++) {
            int n = numbers[i];
            if (n > max)
                max = n;
        }
        System.out.println("returning max:"  + max);
        return max;
    }

    public SelectMaxProblem subproblem(int subStart, int subEnd) {
        return new SelectMaxProblem(numbers, start + subStart,
                                    start + subEnd);
    }
}
```

Sequential algorithm

rgupta.mtech@gmail.com

Selecting max with Fork-Join framework

```
public class MaxWithForkJoin extends RecursiveAction {
    private final int threshold;
    private final SelectMaxProblem problem;
    public long result;

    public MaxWithForkJoin(SelectMaxProblem problem, int threshold) {
        this.problem = problem;
        this.threshold = threshold;
    }

    @Override
    protected void compute() {

        if (problem.size < threshold) {
            result = problem.solveSequentially();
        }
        else {
            int midpoint = problem.size / 2;

            MaxWithForkJoin left = new MaxWithForkJoin(problem.subproblem(0, midpoint), threshold);
            MaxWithForkJoin right = new MaxWithForkJoin(problem.subproblem(midpoint + 1, problem.size), threshold);


            invokeAll(left, right);


            result = Math.max(left.result, right.result);
        }
    }
```

A Fork-Join Task

Solve sequentially if problem is small

Otherwise Create sub tasks & Fork

Join the results

rgupta.mtech@gmail.com

Fork-Join framework implementation

Basic threads (Thread.start() , Thread.join())

- **May require more threads than VM can support**

Conventional thread pools

- **Could run into thread starvation deadlock as fork jointasks spend much of the time waiting for other tasks**

ForkJoinPool is an optimized thread pool executor
(for fork-join tasks)

ForkJoinTask as is could improve performance...
However some boilerplate work is still needed. Is there a better way?
i.e. declarative specification of parallelization

rgupta.mtech@gmail.com

Lambda's make life easier.

```
Predicate<SLAData> june2013 = new Predicate<SLAData>() {
    public boolean op(SLAData s) {
        return s.year == 2013 && s.month == 6;
    }
};

Predicate<SLAData> peakHour = new Predicate<SLAData>() {
    public boolean op(SLAData s) {
        return s.hour == 14;
    }
};
Predicate<SLAData> deleteCommand = new Predicate<SLAData>() {
    public boolean op(SLAData s) {
        return s.command.equalsIgnoreCase("DEL-COMMAND");
    }
};
Ops.ObjectToDouble<SLAData> selectResponseTime = new Ops.ObjectToDouble<SLAData>() {
    public double op(SLAData s) {
        return s.responseTime;
    }
};

ParallelDoubleArray slaDataProcessed = slaData.withFilter(june2013)
                                              .withFilter(peakHour )
                                              .withFilter(deleteCommand)
                                              .withMapping(selectResponseTime )
                                              .all();
double average = slaDataProcessed.sum() / slaDataProcessed.size();
```

With Lambda's

```
slaData.parallelStream().filter(s -> s.getYear() == 2013)
                        .filter(s -> s.getMonth() == JUNE)
                        .filter(s -> s.hour() == 14)
                        .filter(s -> s.getCommand() == DELETE)
                        .mapToInt(s -> s.getResponseTime())
                        .sum();
```

rgupta.mtech@gmail.com

Fork-Join and Map-Reduce

	Fork-Join	Map-Reduce
Processing on	Cores on a single compute node	Independent compute nodes
Division of tasks	Dynamic	Decided at start-up
Inter-task communication	Allowed	No
Task redistribution	Work-stealing	Speculative execution
When to use	Data size that can be handled in a node	Big Data
Speed-up	Good speed-up according to #cores, works with decent size data	Scales incredibly well for huge data sets

rgupta.mtech@gmail.com

Issues Parallel Stream

rgupta.mtech@gmail.com

Stream and Processing Pipeline

- What is a stream?
  - A stream is a ~~collection~~ ~~sequence~~ ~~stream~~ of ~~objects~~.
  - A stream is an ***abstraction*** that represents zero or more ***values***.
- Not (necessarily) a collection: values might not be stored anywhere
- Not (necessarily) a sequence: order might not matter
- Values, not objects: avoid mutation and side effects

Pipelines

- A pipeline consists of:
  - a stream ***source***
  - zero or more ***intermediate*** operations
  - a ***terminal*** operation

```
collection.stream()          // source
          .filter(...)       // intermediate operation
          .map(...)          // intermediate operation
          .collect(...);     // terminal operation
```

rgupta.mtech@gmail.com

Parallel Streams

- Source starts with stream(), parallelStream(), or other stream factory
- Can be switched using parallel() or sequential() calls
- Parallel vs sequential is a property of the entire pipeline
  - can't switch between parallel and sequential in the middle
  - "last one wins"
- Parallel makes it auto-magically go faster, right?

```
collection.stream()
          .filter(...)
          .parallel()
          .map(...)
          .sequential()
          .collect(...);

// entire stream runs sequentially
```

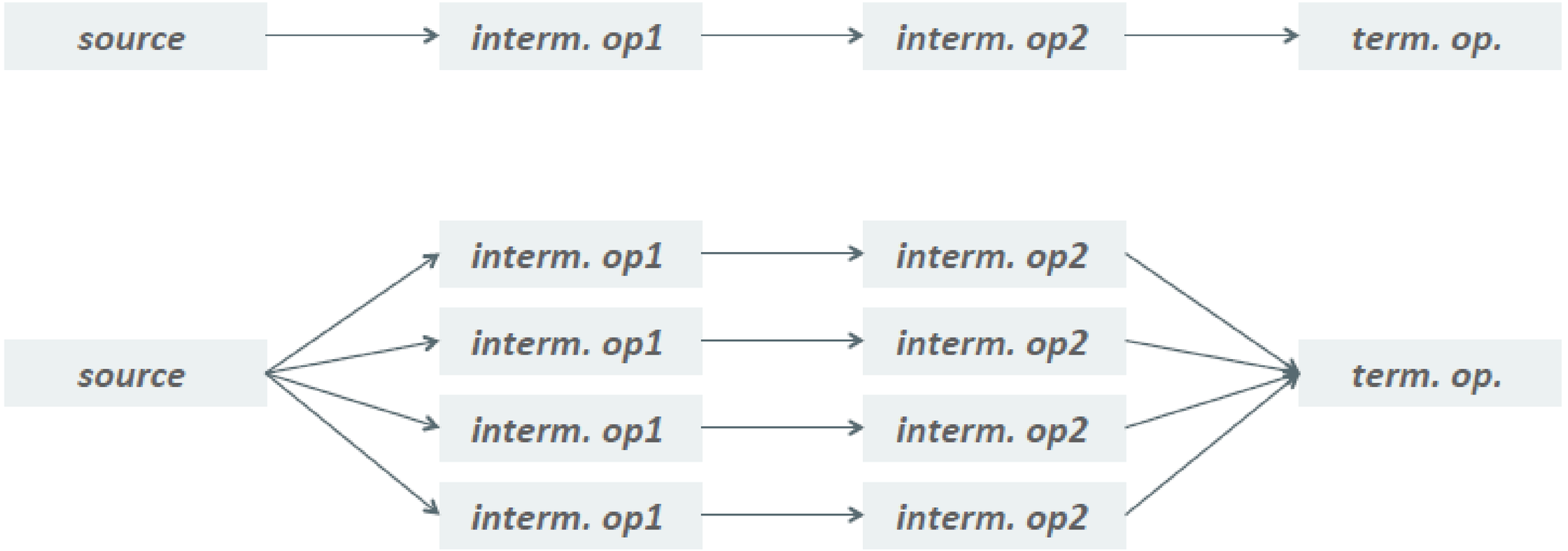
rgupta.mtech@gmail.com

Parallel Streams Considerations

- Parallel and sequential streams should give the same result
  - parallelism leads to nondeterminism, usually bad! Need to control it.
- Encounter order vs. processing order
- Stateless vs. stateful: managing side effects
- Accumulation vs. Reduction
- Reduction: identity and associativity
- Explicit nondeterminism can speed things up
- Parallelism has overhead, might slow things down

rgupta.mtech@gmail.com

Sequential vs Parallel Streams

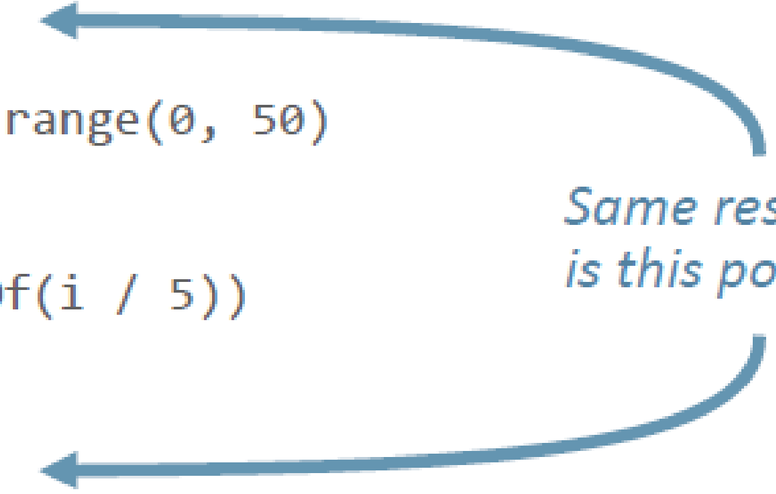


rgupta.mtech@gmail.com

Sequential vs Parallel Streams

```
List<String> output = IntStream.range(0, 50)
    .filter(i -> i % 5 == 0)
    .mapToObj(i -> String.valueOf(i / 5))
    .collect(toList());

[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

```
List<String> output = IntStream.range(0, 50)
    .parallel()
    .filter(i -> i % 5 == 0)
    .mapToObj(i -> String.valueOf(i / 5))
    .collect(toList());

[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```



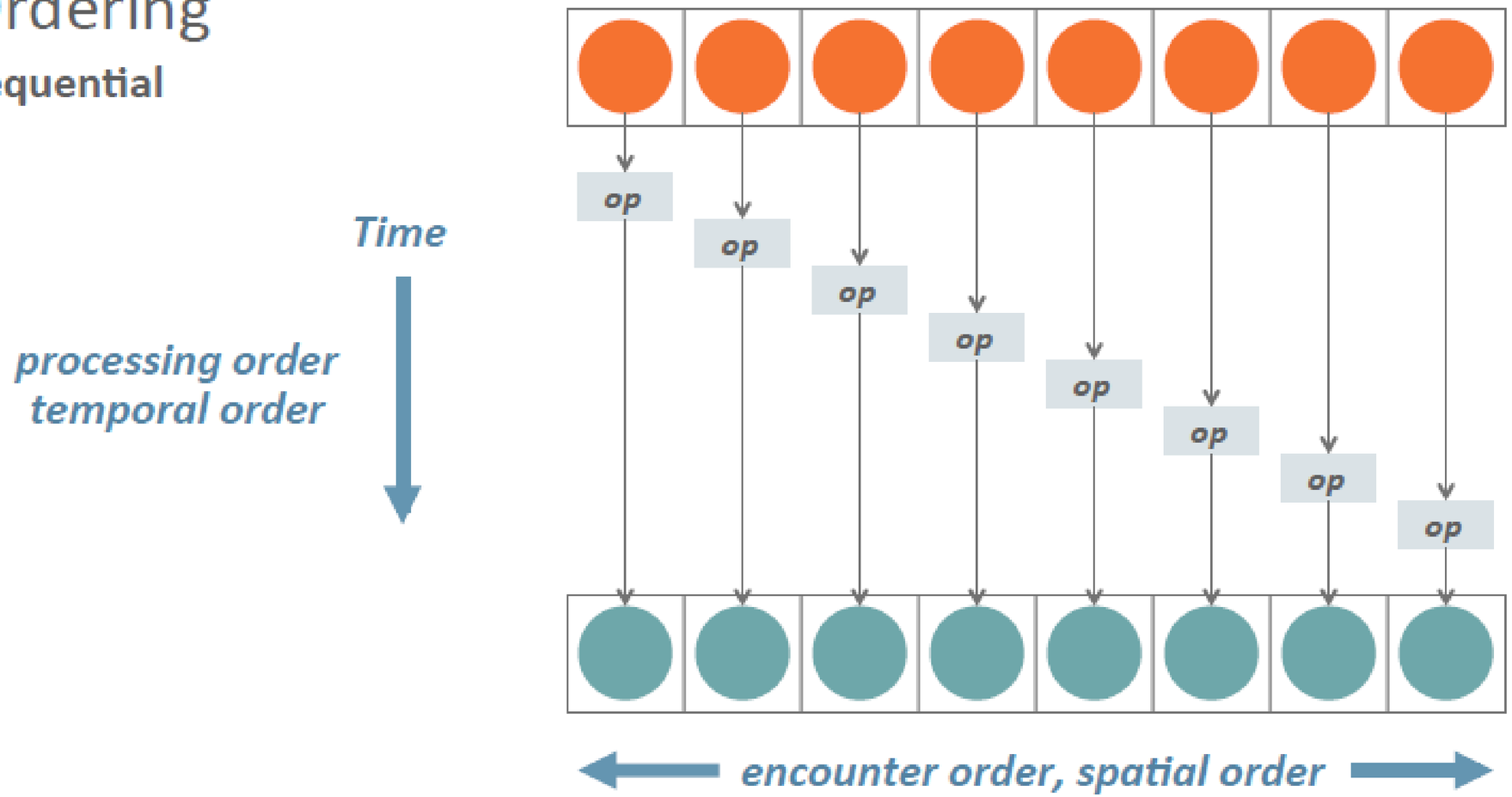
Same result; how is this possible?

rgupta.mtech@gmail.com

Sequential vs Parallel Streams

Ordering

Sequential

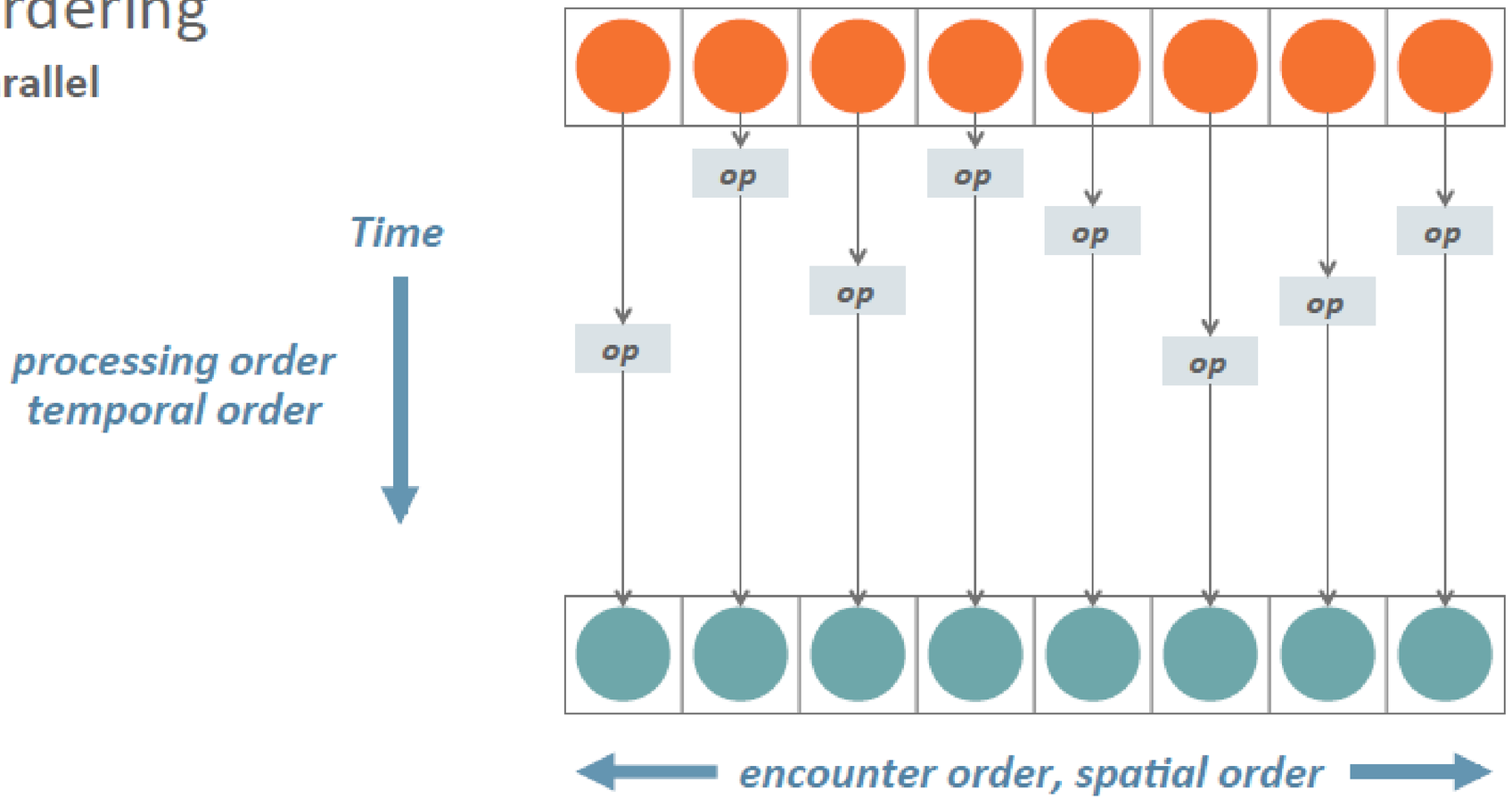


rgupta.mtech@gmail.com

Sequential vs Parallel Streams

Ordering

Parallel

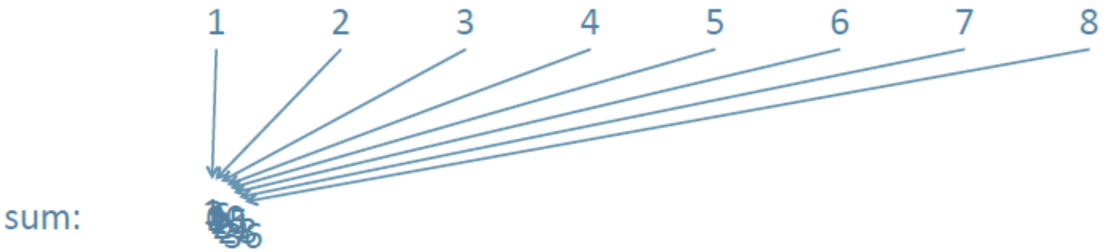


rgupta.mtech@gmail.com

Accumulation vs Reduction

```
long sum = 0L;

for (long i = 1L; i <= 1_000_000L; i++)
    sum += i;
}

System.out.println(sum);

500000500000
```

Summation by Accumulation



Contention!

rgupta.mtech@gmail.com

Accumulation vs Reduction

A Better Way: Reduction

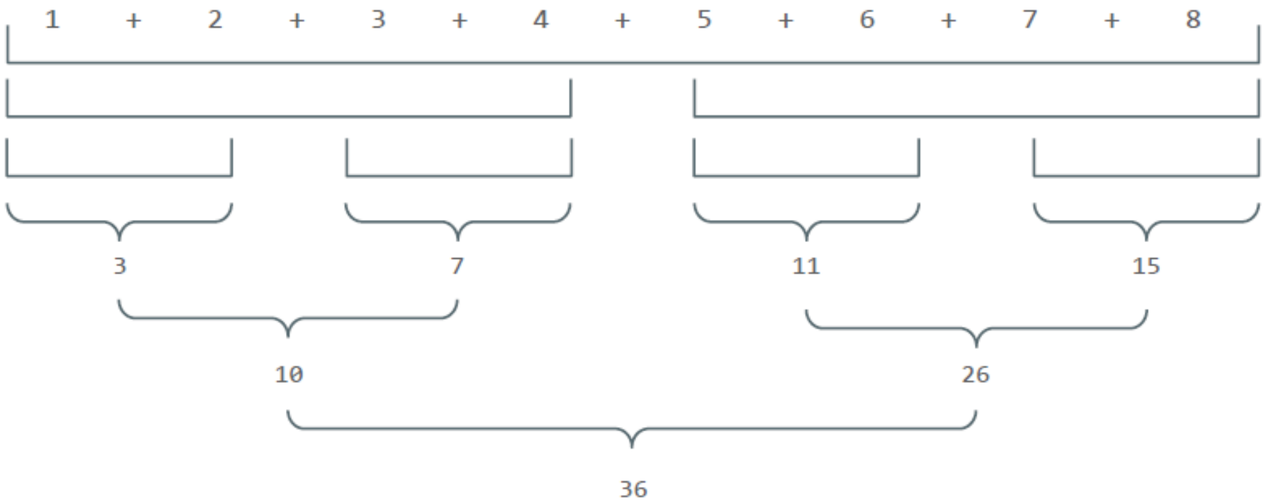
1 2 3 4 5 6 7 8

Reduction over addition:
Just put a plus between each value.

A Better Way: Reduction

1 + 2 + 3 + 4 + 5 + 6 + 7 + 8

Reduction Implementation

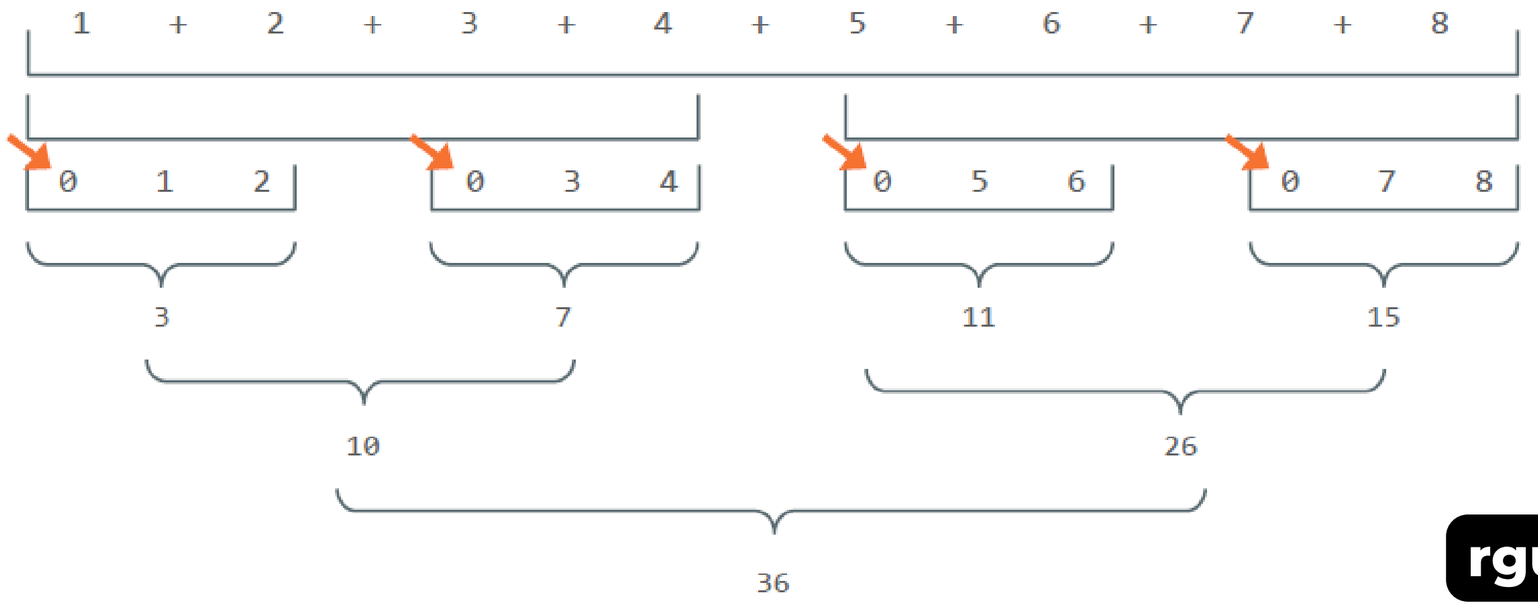


rgupta.mtech@gmail.com

Reduction Identity

- Identity Value
  - the starting value of each partition of a parallel reduction
  - becomes the result if there are no values in the stream
  - it must be the ***right*** identity value
  - must really be an identity ***value*** (immutable, not mutable)

Reduction Implementation

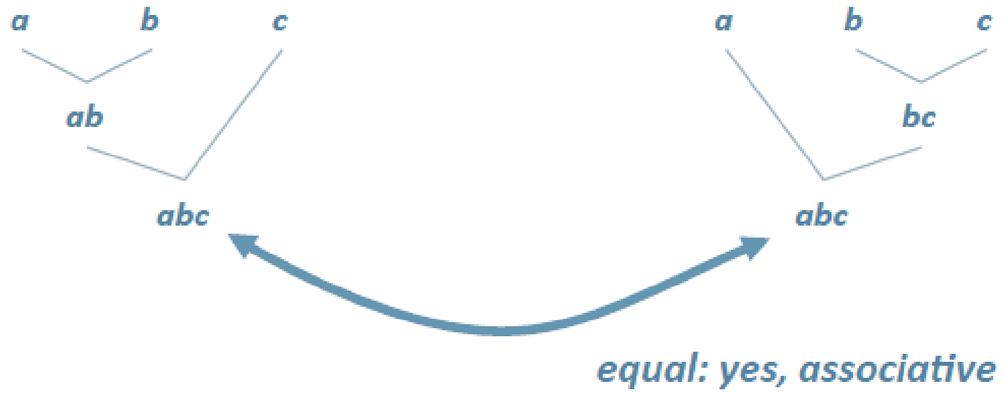


rgupta.mtech@gmail.com

Associativity

- Remember elementary arithmetic?
  - (a + b) + c = a + (b + c) ?
- Turns out it's quite important
- A function is ***associative*** if different groupings of operands don't affect the result
- Reduction functions must be associative

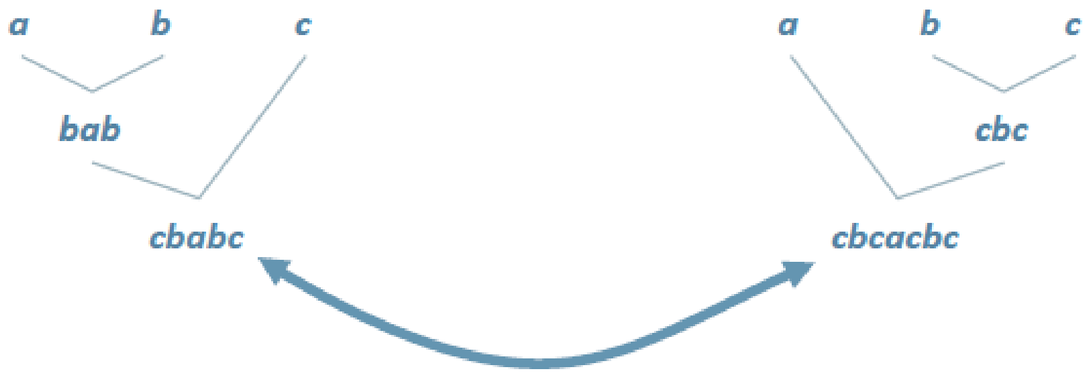
```
(String x, String y) -> x + y
```



Associativity

- Remember elementary arithmetic?
  - (a + b) + c = a + (b + c) ?
- Turns out it's quite important
- A function is ***associative*** if different groupings of operands don't affect the result
- Reduction functions must be associative

```
(String x, String y) -> y + x + y
```



not equal, not associative!

Non Determinism

- Usually we want to get the same result for parallel as sequential
- Consider findFirst()
  - "first" means first in encounter (spatial) order
  - parallel can find *a* matching element quickly
  - but still has to search space to the left to ensure it's first
- Consider findAny()
  - parallel can find a matching element quickly
  - and it's done!

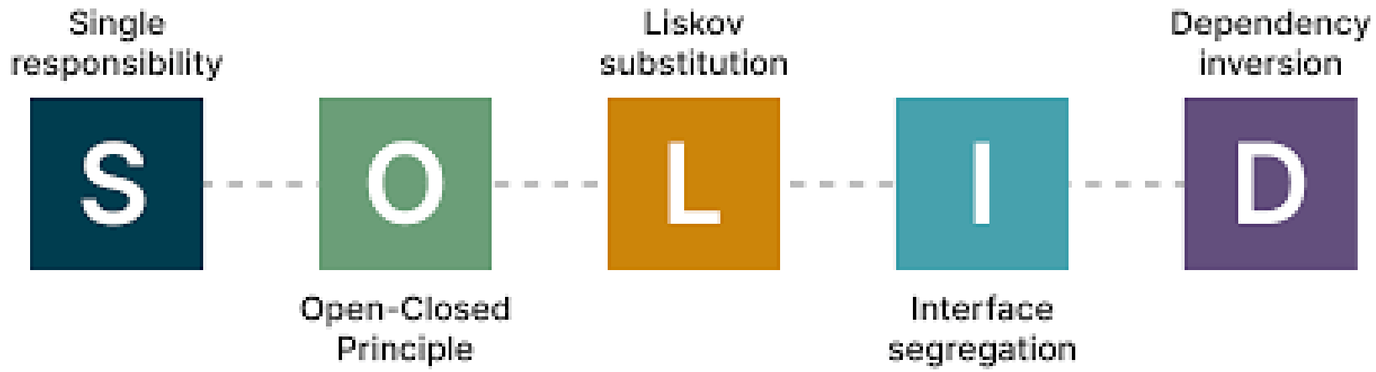
rgupta.mtech@gmail.com

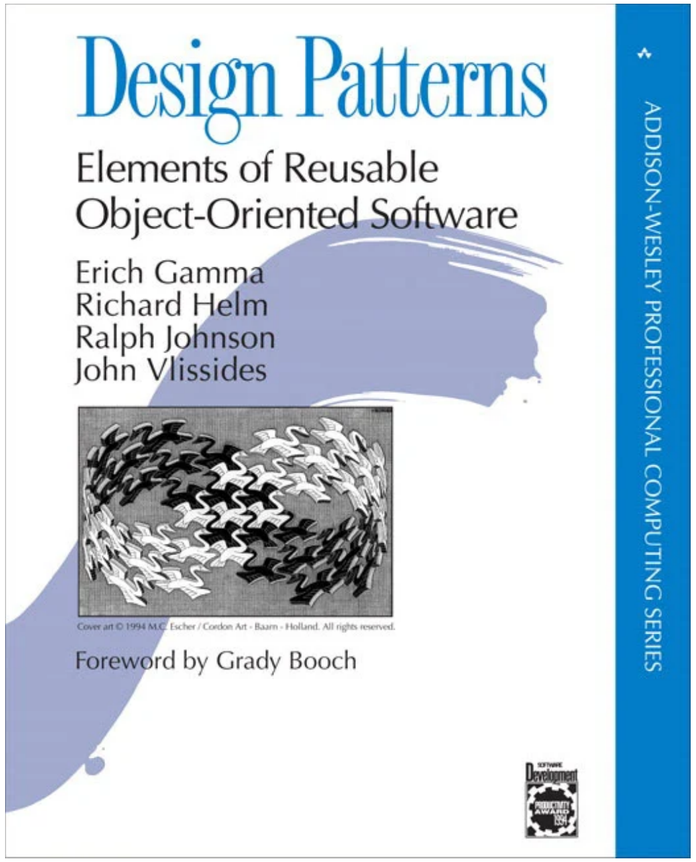
Under the hood?

- Parallel stream initiated by parallelStream() or parallel() call
  - who starts the threads? where do they come from?
- Stream workload split and dispatched to the ***common fork-join pool***
- Control over concurrency explicitly opaque in the API
  - allocation of resources should be by administrator/deployer, not programmer
  - common FP pool controlled by system properties; needs to be enhanced
- Policy APIs need development
  - split policy, degree of parallelism, handling blocking tasks

rgupta.mtech@gmail.com

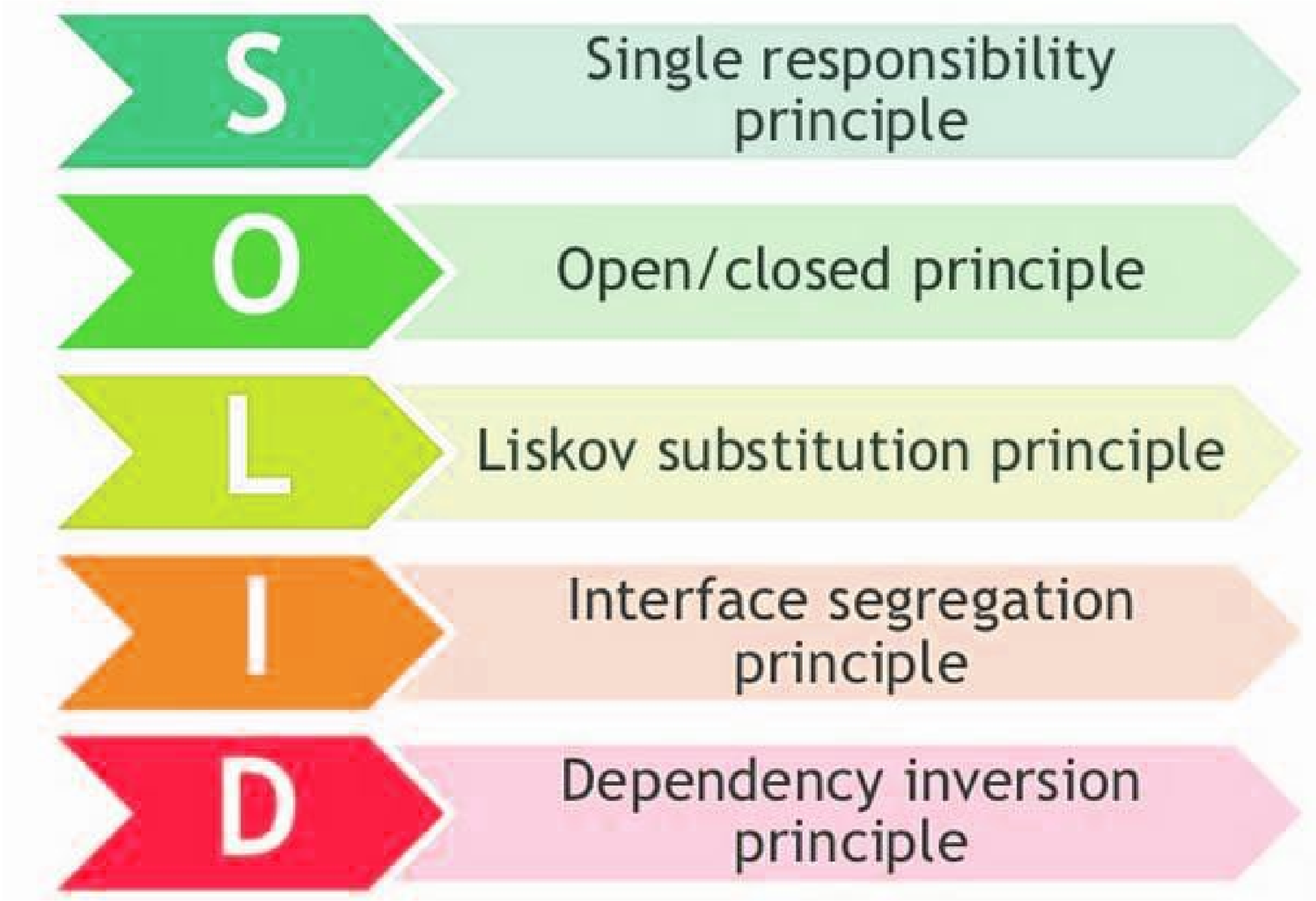
Session 9:
SOLID & GOF Patterns





rgupta.mtech@gmail.com

SOLID Principles



rgupta.mtech@gmail.com

Without Good design our application would be...

Rigid

Fragile

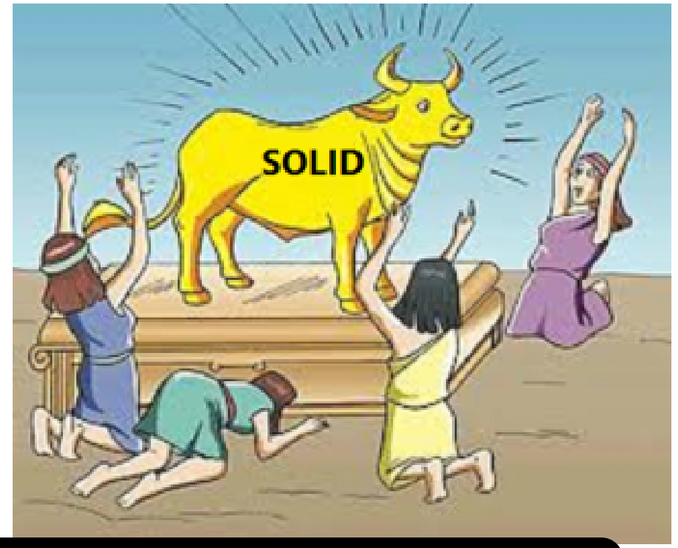
Immobile

Making everyone's life miserable

rgupta.mtech@gmail.com

SOLID Principles

- In software engineering **SOLID** is an acronym for **5** design principles
- These principles are mainly promoted by **Robert C. Martin** in his book back in **2000**
- **It helps us to write code that is …**
  - **Loosely coupled**
  - **Highly cohesive**
  - **Easily composable**
  - **Reusable**



rgupta.mtech@gmail.com

S - Single Responsibility Principle

- Every single software entity (**class** or **method**) should have only a **single reason** to change
- If a given class (or method) does **multiple operations** then it is advisable to separate into distinct classes (or methods)
- If there are **2** reasons to change a given class then it is a sign of violating the single responsibility principle

”There should be **NEVER** be more than **ONE** reason for a class to change”

rgupta.mtech@gmail.com

O – Open/Closed Principle

- Software entities should be **open for extension** and closed for modification
- We have to design every new module such that if we add a new behavior then we **do not have to change the existing modules**
- CLOSELY RELATED TO SINGLE RESONSIBILITY PRINCIPLE
- a class should not extend an other class explicitly – we should define a **common interface** instead
- we can change the classes at **runtime** due to the common interface

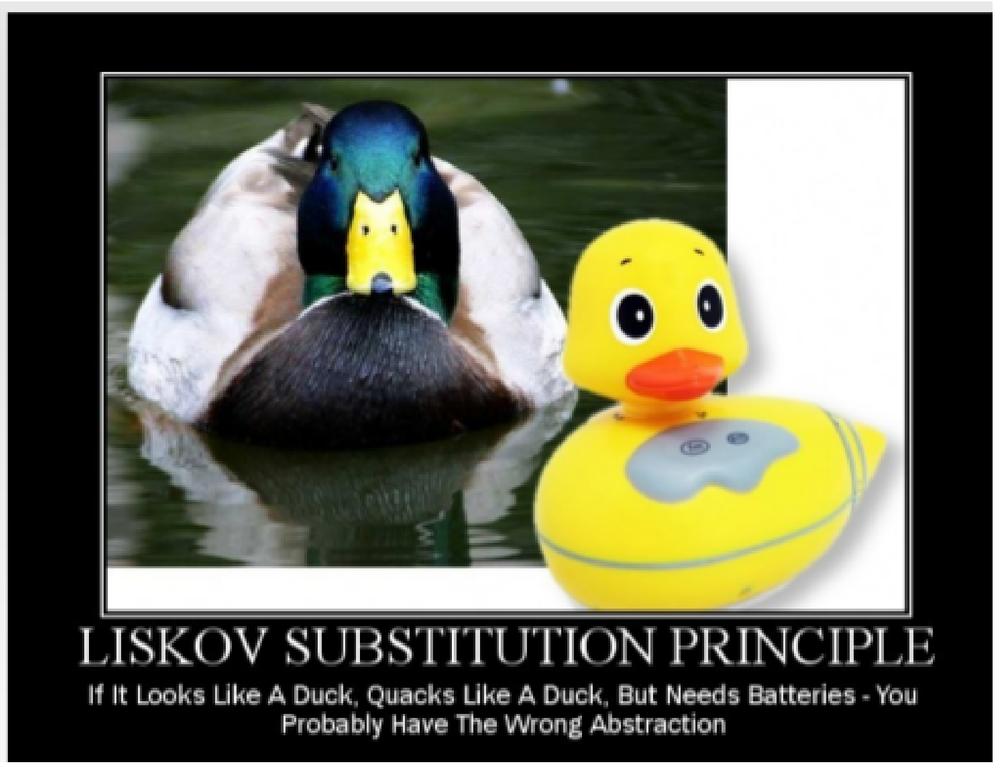
”Modules must be **OPEN** for extension, **CLOSED** for modification”

rgupta.mtech@gmail.com

L – Liskov Substitution Principle

We extend some classes and creating some **derived classes**

It would be great if the new derived classed would work as well **without replacing the functionality** of the classes otherwise the new classes can produce undesired effects when they are used in **existing program modules**



*"Objects of a superclass shall be replaceable with objects of its subclasses **without breaking** the application"*

rgupta.mtech@gmail.com

I – Interface Segregation Principle

- What is the motivation behind **interface segregation principle**?

WE USE SEVERAL INTERFACES OR ABSTRACT CLASSES
IN ORDER TO ACHIEVE ABSTRACTION

- Sometimes we want to implement that interface but just for the sake of some methods defined in by that interface we can end up with **fat interfaces** – containing more methods than the actual class needs

”Clients should not depend upon the interface they do not use”

How can we pollute the interfaces?
OR
How do we end up creating Fat interfaces?

rgupta.mtech@gmail.com

D – Dependency Inversion Principle

- What is the motivation behind **dependency inversion principle**?
- USUALLY THE LOW LEVELS MODULES RELY HEAVILY ON HIGH LEVEL MODULES (BOTTOM UP SOFTWARE DEVELOPMENT)
  - When implementing an application usually we start with the **low level** software components then we implement the **high level** modules that rely on these low level modules

”High level modules should not depend on the low level details modules, instead both should depend on abstractions”

rgupta.mtech@gmail.com

Design patterns
GOF

rgupta.mtech@gmail.com

Design Patterns

design patterns *are more about how to design your code*

+ concrete implementations of the design principles

design principles *(SOLID principles) allow scalable and maintainable software architectures*

rgupta.mtech@gmail.com

Top 10 Object Oriented Design Principles

1. DRY (Don't repeat yourself) - avoids duplication in code.
2. Encapsulate what changes - hides implementation detail, helps in maintenance
3. Open Closed design principle - open for extension, closed for modification
4. SRP (Single Responsibility Principle) - one class should do one thing and do it well
5. DIP (Dependency Inversion Principle) - don't ask, let framework give to you
6. Favor Composition over Inheritance - Code reuse without cost of inflexibility
7. LSP (Liskov Substitution Principle) - Sub type must be substitutable for super type
8. ISP (Interface Segregation Principle) - Avoid monolithic interface, reduce pain on client side
9. Programming for Interface - Helps in maintenance, improves flexibility
10. Delegation principle - Don't do all things by yourself, delegate it

rgupta.mtech@gmail.com

Design Patterns

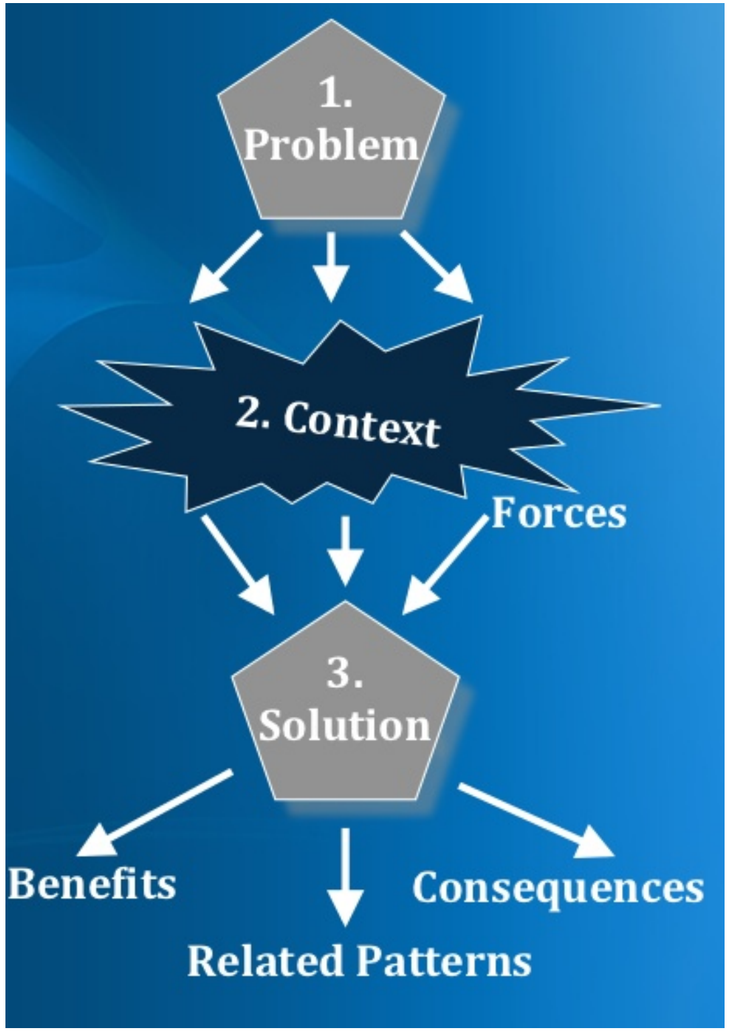
Proven way of doing Things, Gang of 4 design patterns

- Total 23 patterns

- Classification patterns
- Creational
- Structural
- Behavioral

Design patterns are design level solutions for recurring problems that we software engineers come across often.

It's not code - I repeat, ✕CODE. It is like a description on how to tackle these problems and design a solution

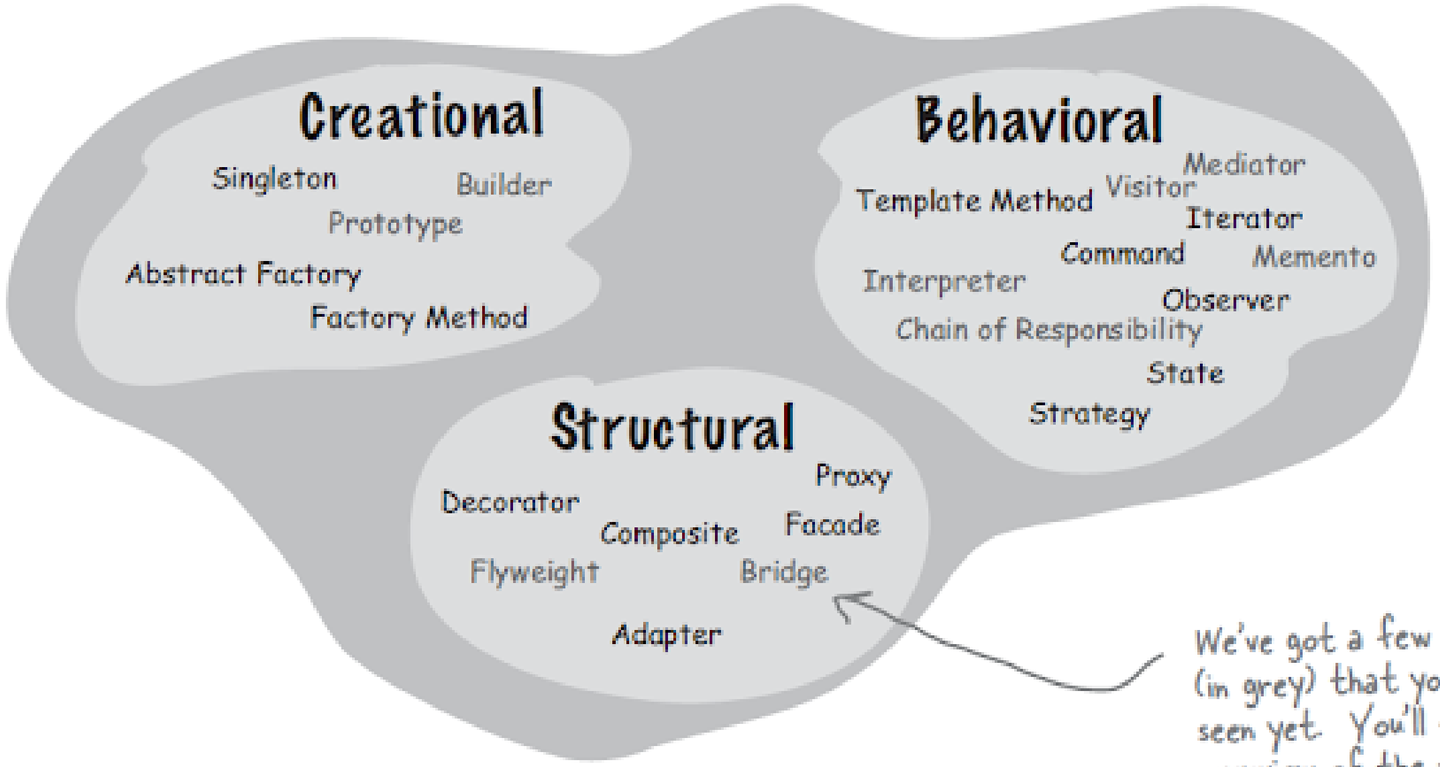


rgupta.mtech@gmail.com

Design Patterns

Creational patterns involve object instantiation and all provide a way to decouple a client from the objects it needs to instantiate.

Any pattern that is a **Behavioral Pattern** is concerned with how classes and objects interact and distribute responsibility.

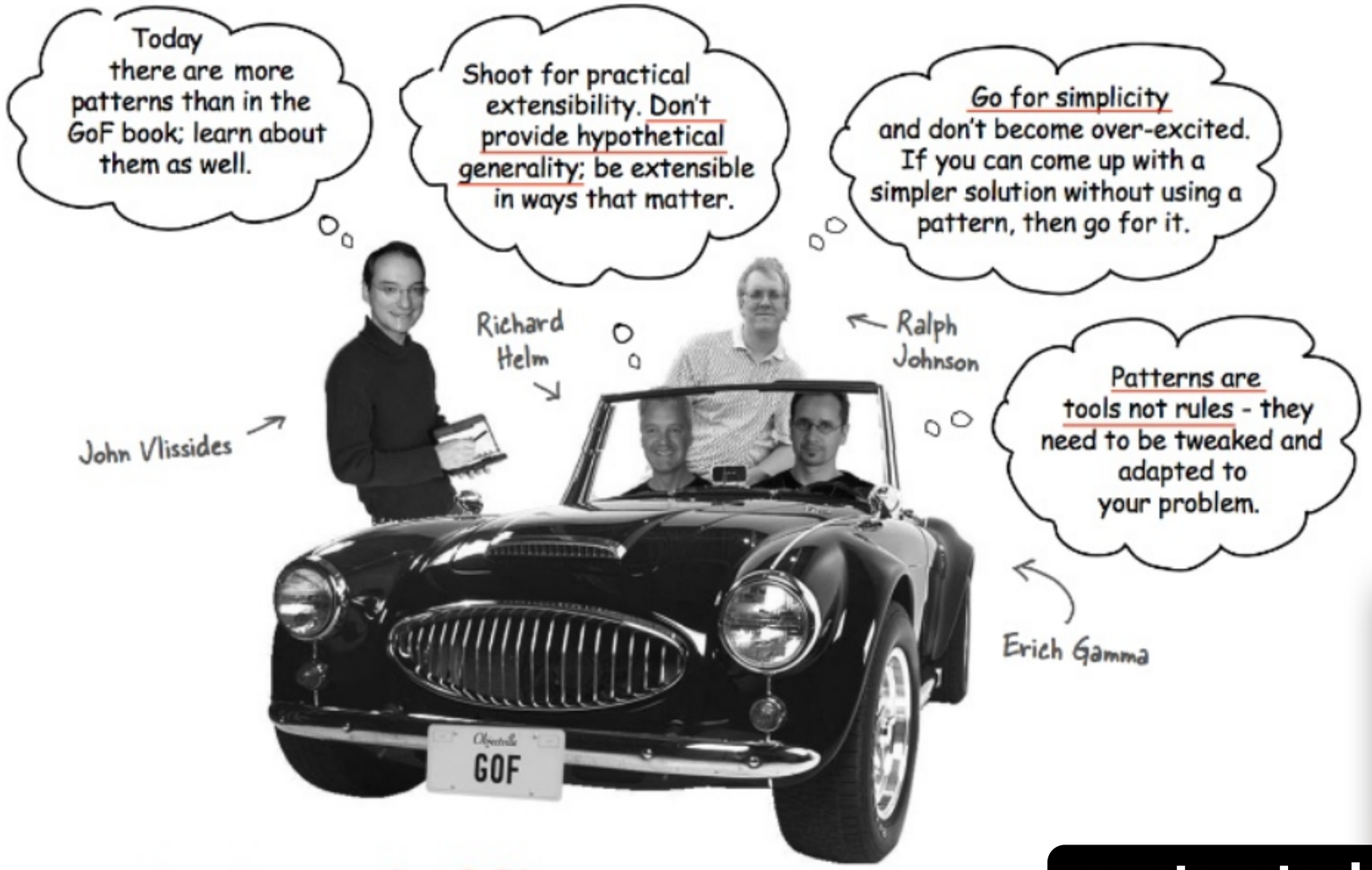


We've got a few patterns (in grey) that you haven't seen yet. You'll find an overview of the these patterns in the appendix.

Structural patterns let you compose classes or objects into larger structures.

rgupta.mtech@gmail.com

Design Patterns



Keep it simple (KISS)

rgupta.mtech@gmail.com

Design Patterns- Classification

Structural Patterns

- 1. Decorator
- 2. Proxy
- 3. Bridge
- 4. Composite
- 5. Flyweight
- 6. Adapter
- 7. Facade

Creational Patterns

- 1. Prototype
- 2. Factory Method
- 3. Singleton
- 4. Abstract Factory
- 5. Builder

Behavioral Patterns

- 1. Strategy
- 2. State
- 3. TemplateMethod
- 4. Chain of Responsibility
- 5. Command
- 6. Iterator
- 7. Mediator
- 8. Observer
- 9. Visitor
- 10. Interpreter
- 11. Memento

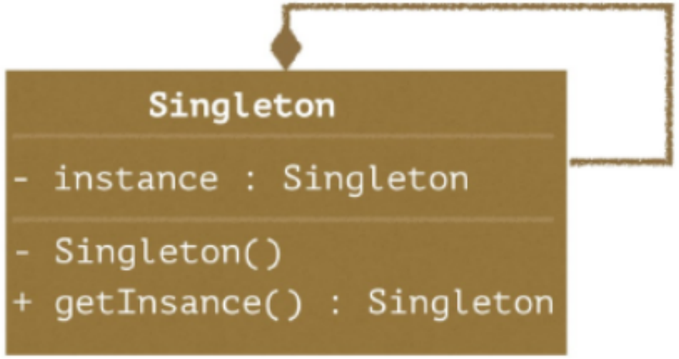
rgupta.mtech@gmail.com

rgupta.mtech@gmail.com

Singleton Pattern

- **Singleton** pattern is a creational design pattern
- It lets you ensure that a **class has only one instance** while providing a global access point to this instance
- It ensures that a given class has just a single instance
- The singleton pattern provides a **global access point** to that given instance

"Ensure that a class has only one instance and provide a global point of access to it."



- `java.lang.Runtime#getRuntime()`
- `java.awt.Desktop#getDesktop()`
- `java.lang.System#getSecurityManager()`

❖ **Java Runtime** class is used *to interact with java runtime environment*.
❖ Java Runtime class provides methods to execute a process, invoke GC, get total and free memory etc.
❖ There is only one instance of java.lang.Runtime class is available for one java application.
❖ The **Runtime.getRuntime()** method returns the singleton instance of Runtime class.

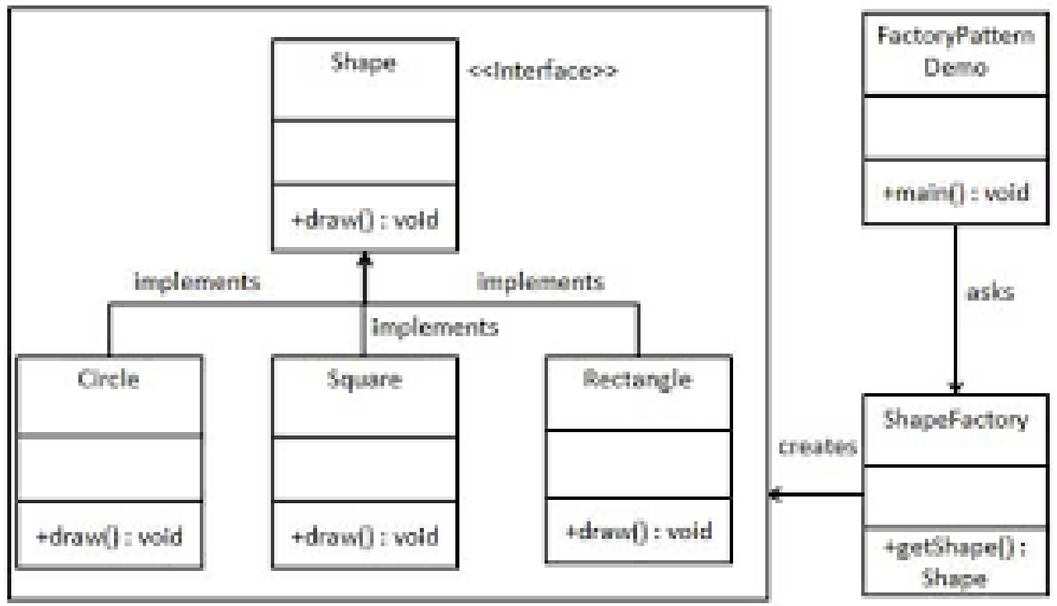
rgupta.mtech@gmail.com

Singleton Design Consideration

- Eager initialization
- Static block initialization
- Lazy Initialization
- Thread Safe Singleton
- Serialization issue
- Cloning issue
- Using Reflection to destroy Singleton Pattern
- Enum Singleton
- Best programming practices

rgupta.mtech@gmail.com

Factory design pattern



Factory(Simplified version of Factory Method) - **Creates objects without exposing the instantiation logic to the client** and **Refers to the newly created object through a common interface.**

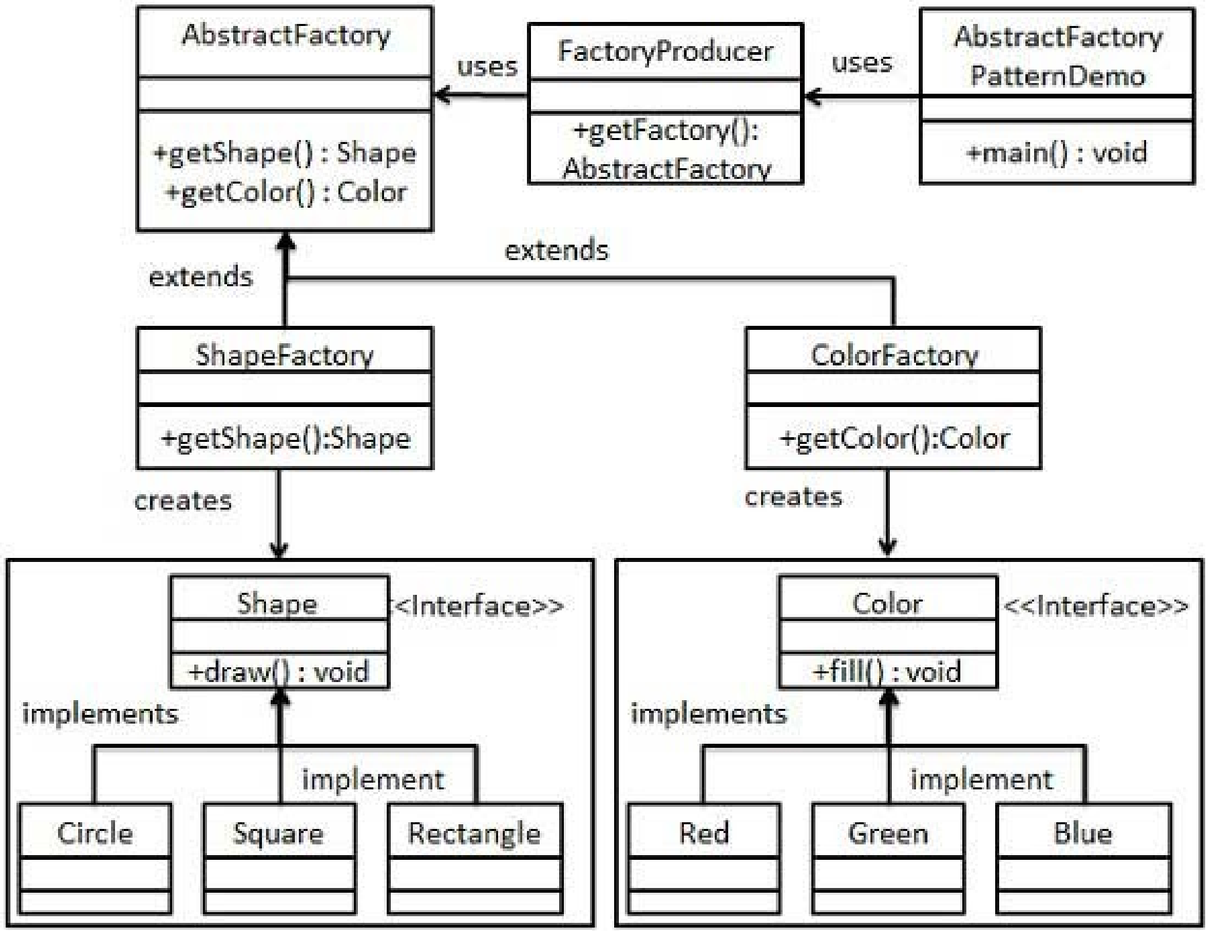
- `java.util.Calendar#getInstance()`
- `java.util.ResourceBundle#getBundle()`
- `java.text.NumberFormat#getInstance()`

Factory Method - **Defines an interface for creating objects, but let subclasses to decide which class to instantiate** and **Refers to the newly created object through a common interface.**

rgupta.mtech@gmail.com

Abstract Factory

Java: **Abstract Factory**. **Abstract Factory** is a creational **design pattern**, which solves the problem of creating entire product families without specifying their concrete classes. **Abstract Factory** defines an interface for creating all distinct products, but leaves the actual product creation to concrete **factory** classes.



- javax.xml.parsers.DocumentBuilderFactory#newInstance()
- javax.xml.transform.TransformerFactory#newInstance()
- javax.xml.xpath.XPathFactory#newInstance()

rgupta.mtech@gmail.com

Builder Pattern

telescoping constructors

```
public class Person {

    private int age;
    private Gender gender;
    private String dateOfBirth;
    private String firstName;
    private String lastName;
    private String nameOfMother;
    private Address address;
    private PhoneNumber phoneNumber;

    public Person(int age, Gender gender, String dateO
            String lastName, String nameOfMother,
            Address address, PhoneNumber phoneNumber)
        super();
        this.age = age;
        this.gender = gender;
        this.dateOfBirth = dateOfBirth;
        this.firstName = firstName;
        this.lastName = lastName;
        this.nameOfMother = nameOfMother;
```

what is the **motivation** behind builder pattern?

The Builder pattern can be used to ease the construction of a complex object from simple objects.

java.lang.StringBuilder#append() (unsynchronized)
java.lang.StringBuffer#append() (synchronized)

LARGE NUMBER OF VARIABLES

*there may be a **large amount** of parameters in a constructor*

*because there may be several **instance variables** in a given class*

EASY TO CONFUSE THE PARAMETERS

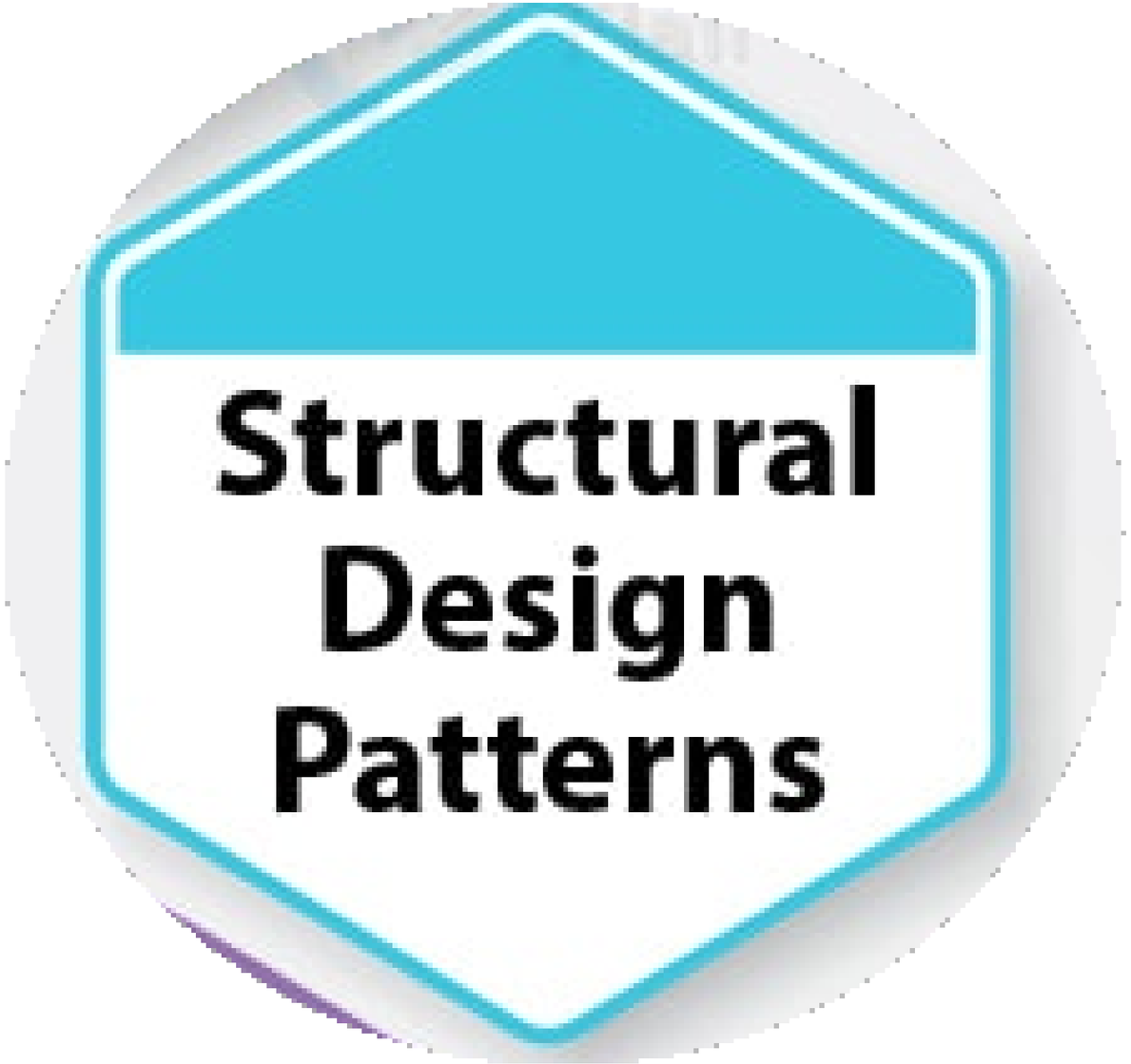
rgupta.mtech@gmail.com

Prototype Pattern

Cloning of an object to avoid creation. If the cost of creating
a new object is large and creation is resource intensive, we clone the object.

- `java.lang.Object#clone()` (the class has to implement `java.lang.Cloneable`)

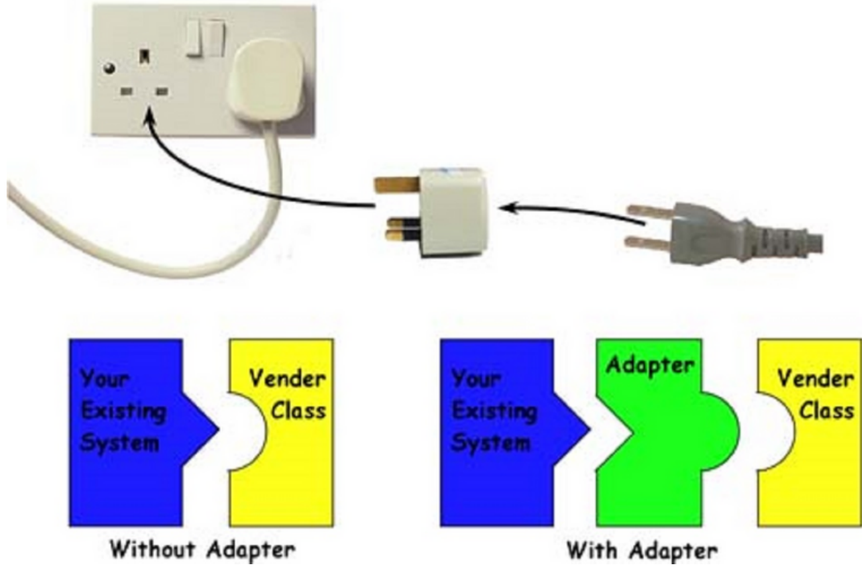
rgupta.mtech@gmail.com



rgupta.mtech@gmail.com

Adapter Pattern

The Adapter pattern is used so that two unrelated interfaces can work together.

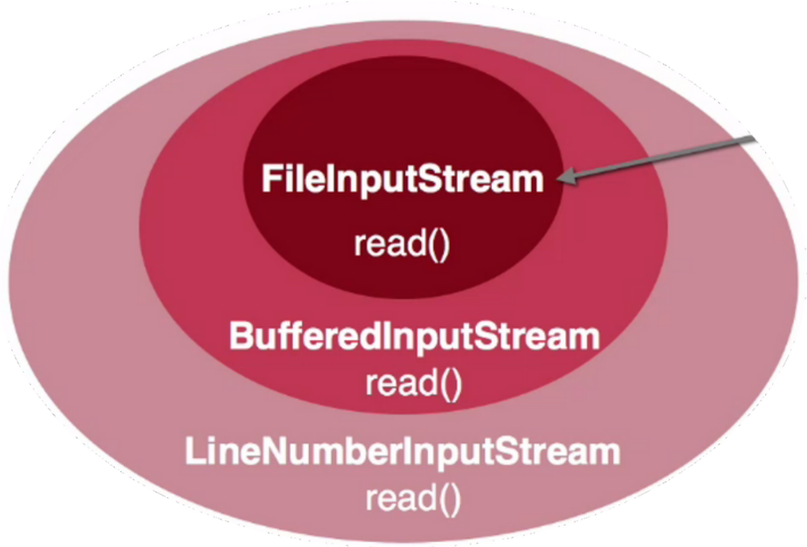


- `java.util.Arrays#asList()`
- `java.util.Collections#list()`
- `java.util.Collections#enumeration()`
- `java.io.InputStreamReader(InputStream)` (returns a `Reader`)
- `java.io.OutputStreamWriter(OutputStream)` (returns a `Writer`)

rgupta.mtech@gmail.com

Decorator design pattern

Series of wrapper class that define functionality, In the Decorator pattern, a decorator object is wrapped around the original object.



Adding behaviour statically or dynamically

Extending functionality without effecting the behaviour of other objects.

Adhering to Open for extension, closed for modification.

All subclasses of `java.io.InputStream`, `OutputStream`, `Reader` and `Writer` have a constructor taking an instance of same type.

`java.util.Collections`, the `checkedXXX()`, `synchronizedXXX()` and `unmodifiableXXX()` methods.

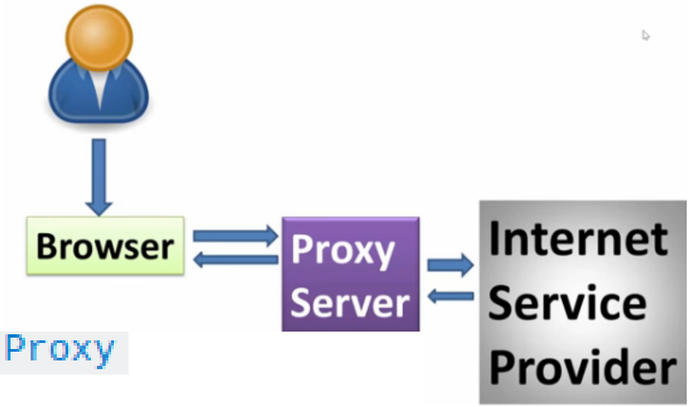
`javax.servlet.http.HttpServletRequestWrapper` and `HttpServletResponseWrapper`

`javax.swing.JScrollPane`

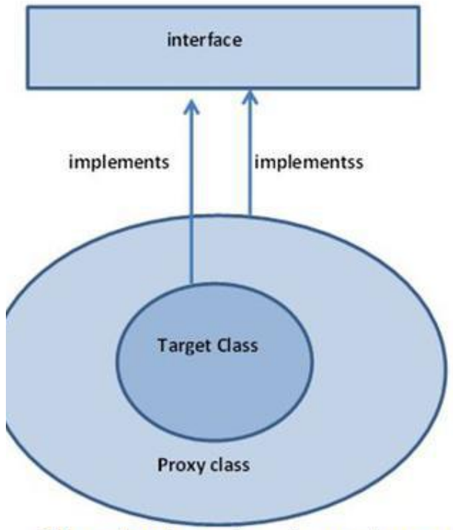
rgupta.mtech@gmail.com

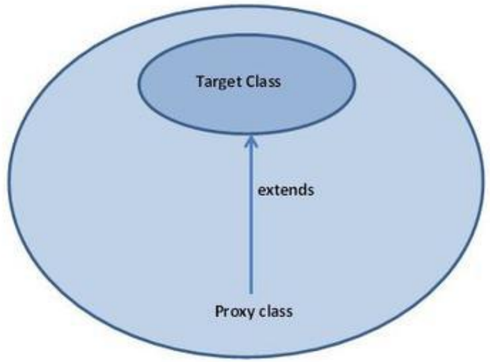
Proxy design Pattern

Proxy pattern, a class represents functionality of another class. This type of design pattern comes under structural pattern. In short, a proxy is a wrapper or agent object that is being called by the client to access the real serving object behind the scenes.



In Spring Framework AOP is implemented by creating proxy object for your service.





CGLIB proxy

`java.lang.reflect.Proxy`

`java.rmi.*`

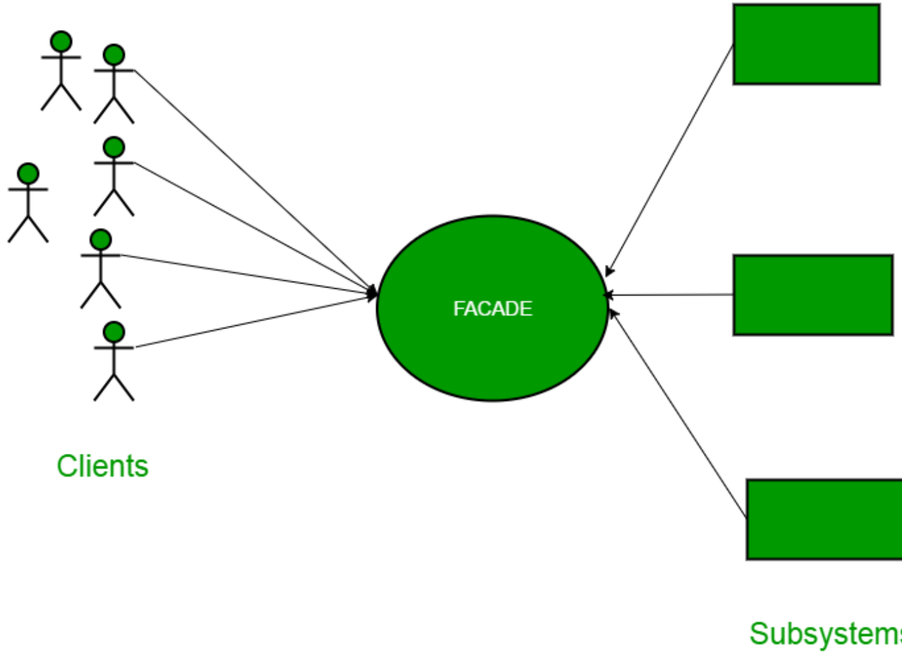
`javax.ejb.EJB` (explanation here)

`javax.inject.Inject` (explanation here)

`javax.persistence.PersistenceContext`

rgupta.mtech@gmail.com

Facade Pattern



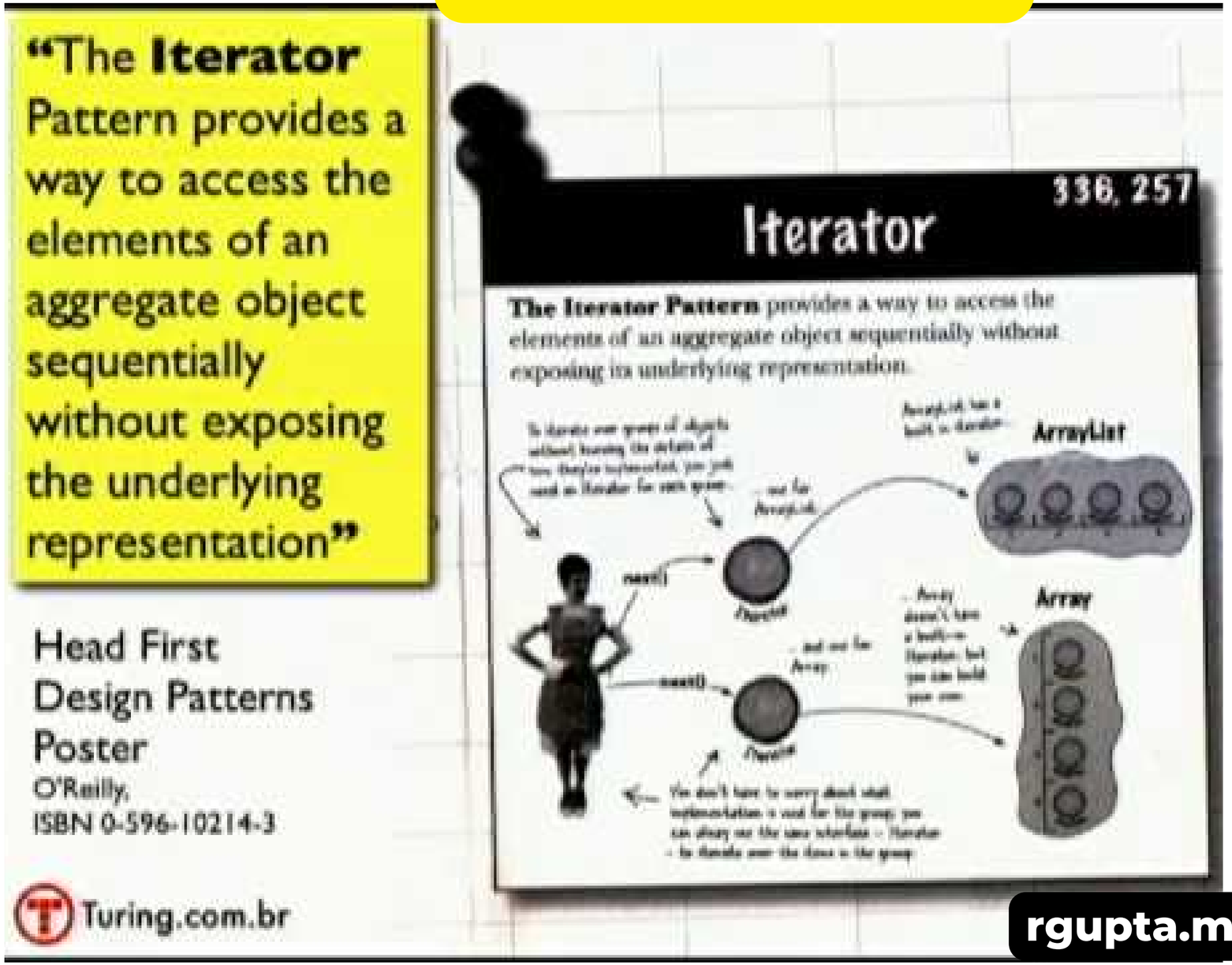
The **facade pattern** (also spelled as **façade**) is a software-**design pattern** commonly used with object-oriented programming. The name is an analogy to an architectural **façade**. A **facade** is an object that provides a simplified interface to a larger body of code, such as a class library.

- `javax.faces.context.FacesContext`, it internally uses among others the abstract/interface types `LifeCycle`, `ViewHandler`, `NavigationHandler` and many more without that the enduser has to worry about it (which are however overrideable by injection).
- `javax.faces.context.ExternalContext`, which internally uses `ServletContext`, `HttpSession`, `HttpServletRequest`, `HttpServletResponse`, etc.

rgupta.mtech@gmail.com

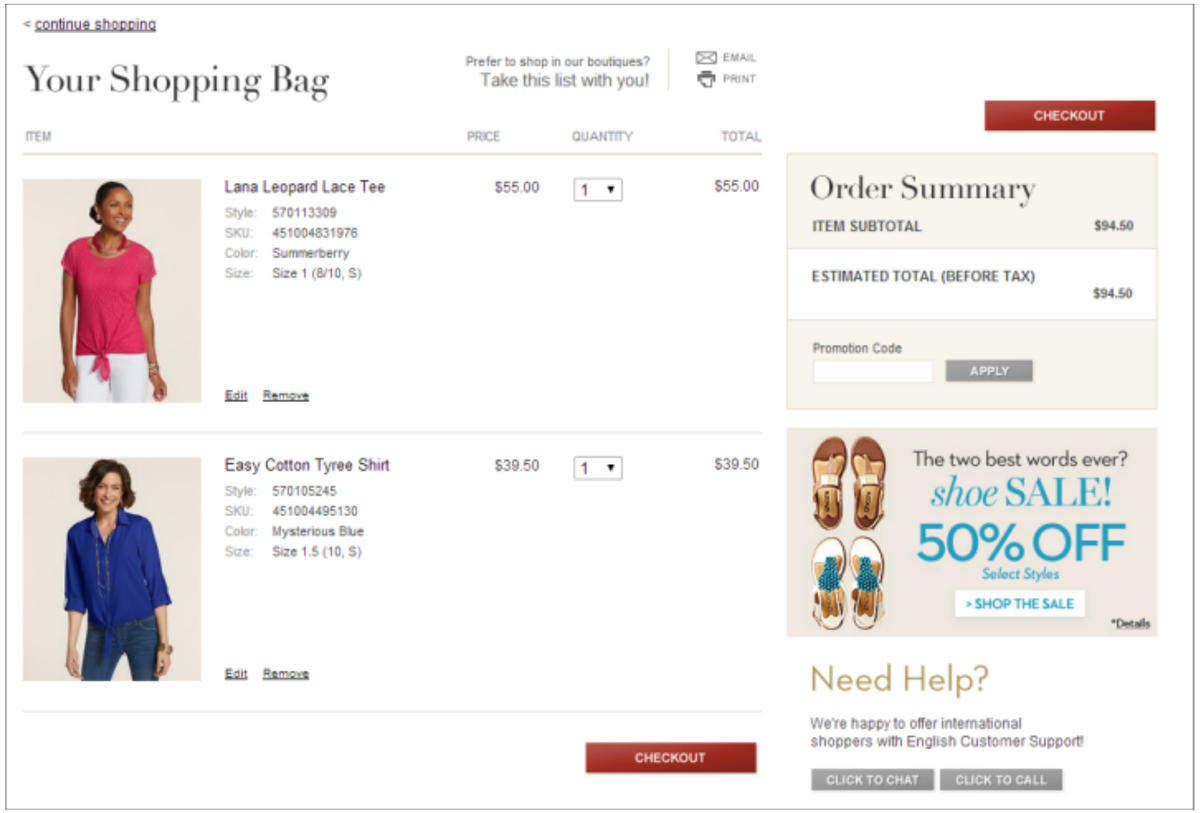
rgupta.mtech@gmail.com

Iterator Pattern



rgupta.mtech@gmail.com

Strategy pattern /policyPattern



Strategy - Define a family of algorithms, encapsulate each one, and make them interchangeable. Strategy lets the algorithm vary independently from clients that use it.

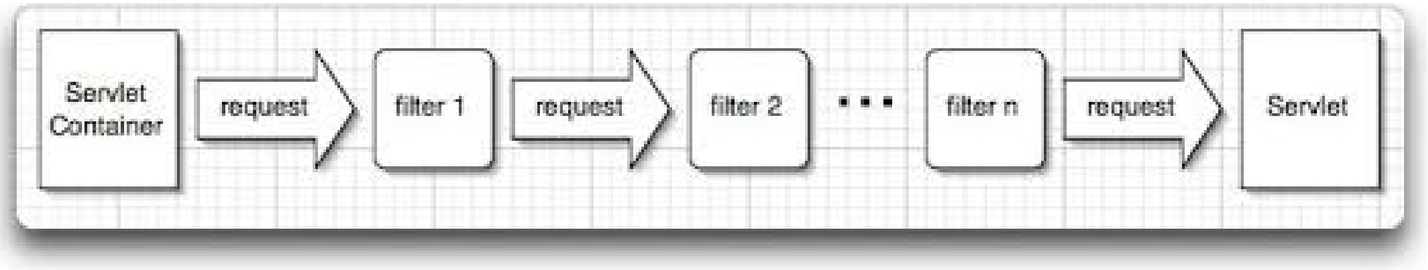
- `java.util.Comparator#compare()`, executed by among others `Collections#sort()`.
- `javax.servlet.http.HttpServlet`, the `service()` and all `doXXX()` methods take `HttpServletRequest` and `HttpServletResponse` and the implementor has to process them (and not to get hold of them as instance variables!).

rgupta.mtech@gmail.com

Chain of Responsibility Pattern

Chain of responsibility pattern is used to achieve loose coupling in software design where a request from client is passed to a chain of objects to process them.

Servlet Filter, Spring Security FilterChain

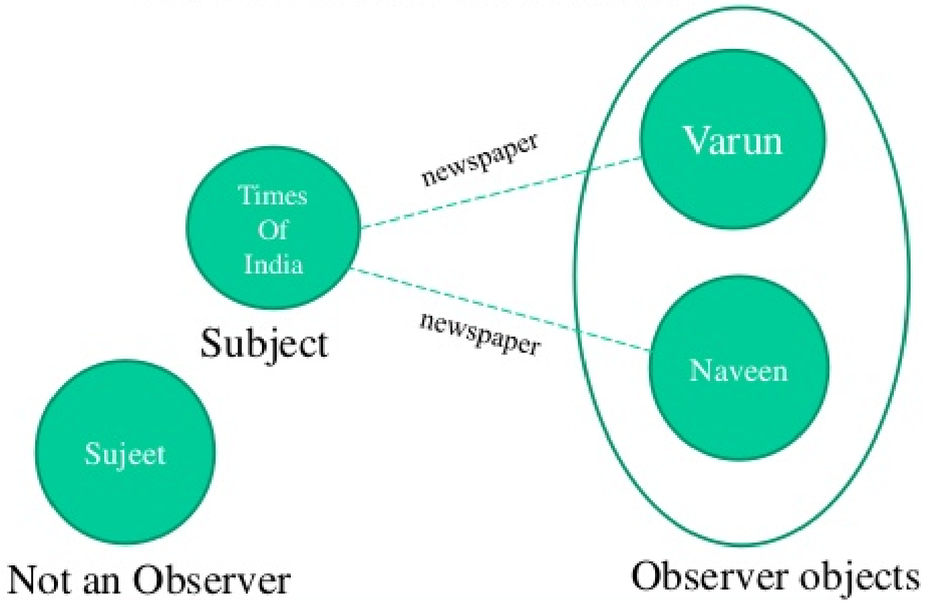


rgupta.mtech@gmail.com

Observer Design Pattern

Observer design pattern is useful when you are interested in the state of an object and want to get notified whenever there is any change.

- Publisher + Subscribers = observer pattern
- In observer pattern publisher is called the subject and subscriber is called the observer



- `java.util.Observer` / `java.util.Observable` (rarely used in real world though)
- All implementations of `java.util.EventListener` (practically all over Swing thus)
- `javax.servlet.http.HttpSessionBindingListener`
- `javax.servlet.http.HttpSessionAttributeListener`
- `javax.faces.event.PhaseListener`

rgupta.mtech@gmail.com

Template Design Pattern

Template Method - Define the skeleton of an algorithm in an operation, deferring some steps to subclasses / Template Method lets subclasses redefine certain steps of an algorithm without letting them to change the algorithm's structure.

Foundation

Pillars

Walls

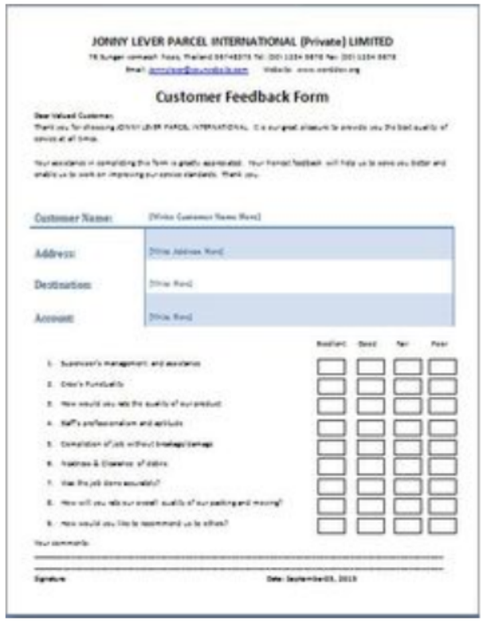
Windows

Template Method [Build a House]

1) Building foundation
2) Building pillars
3) Building walls
4) Building windows

Concrete House

Wooden House



JONNY LEVER PARCEL INTERNATIONAL (Private) LIMITED

Customer Feedback Form

spring JDBC

rgupta.mtech@gmail.com

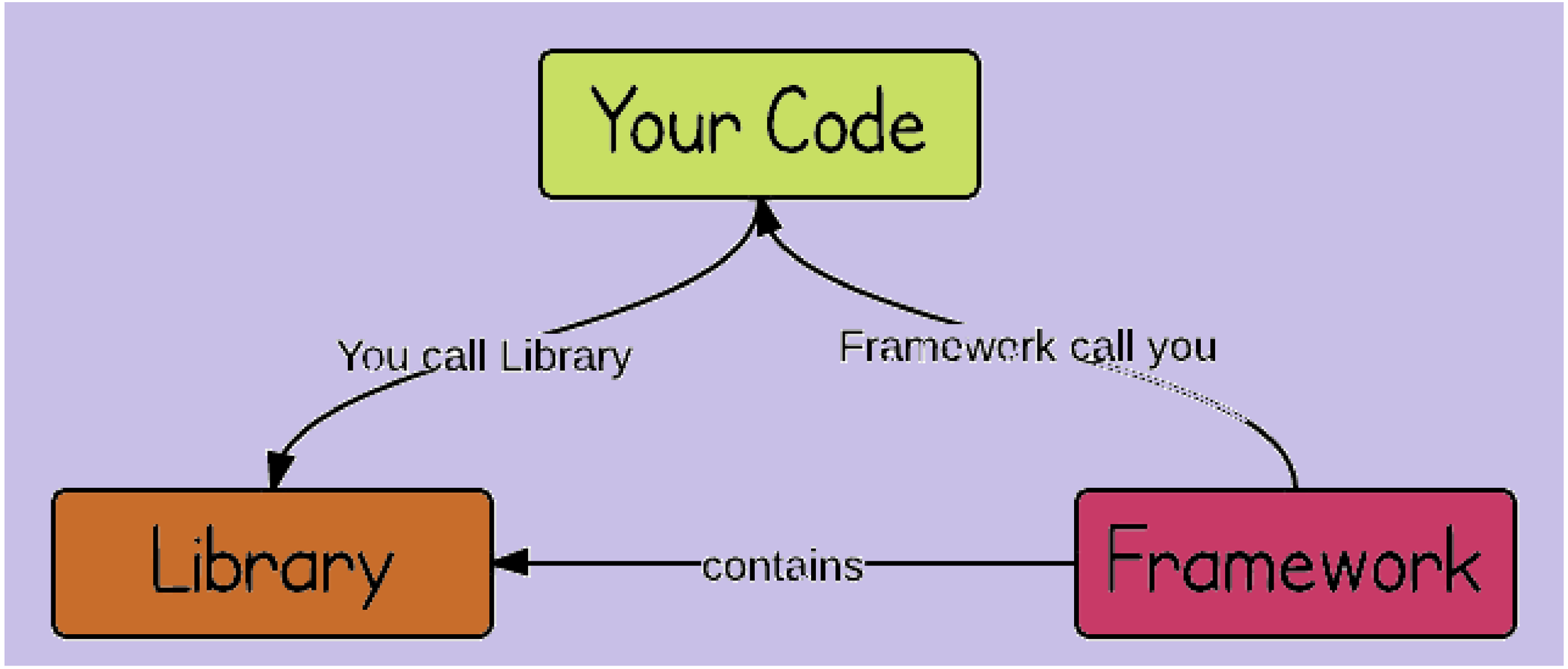
Pattern vs Framework

- Pattern is a set of guidelines on how to architect the application
- When we implement a pattern we need to have some classes and libraries
- **Thus, pattern is the way you can architect your application.**

- Framework helps us to follow a particular pattern when we are building a web application
- These prebuilt classes and libraries are provided by the MVC framework.
- **Framework provides foundation classes and libraries.**

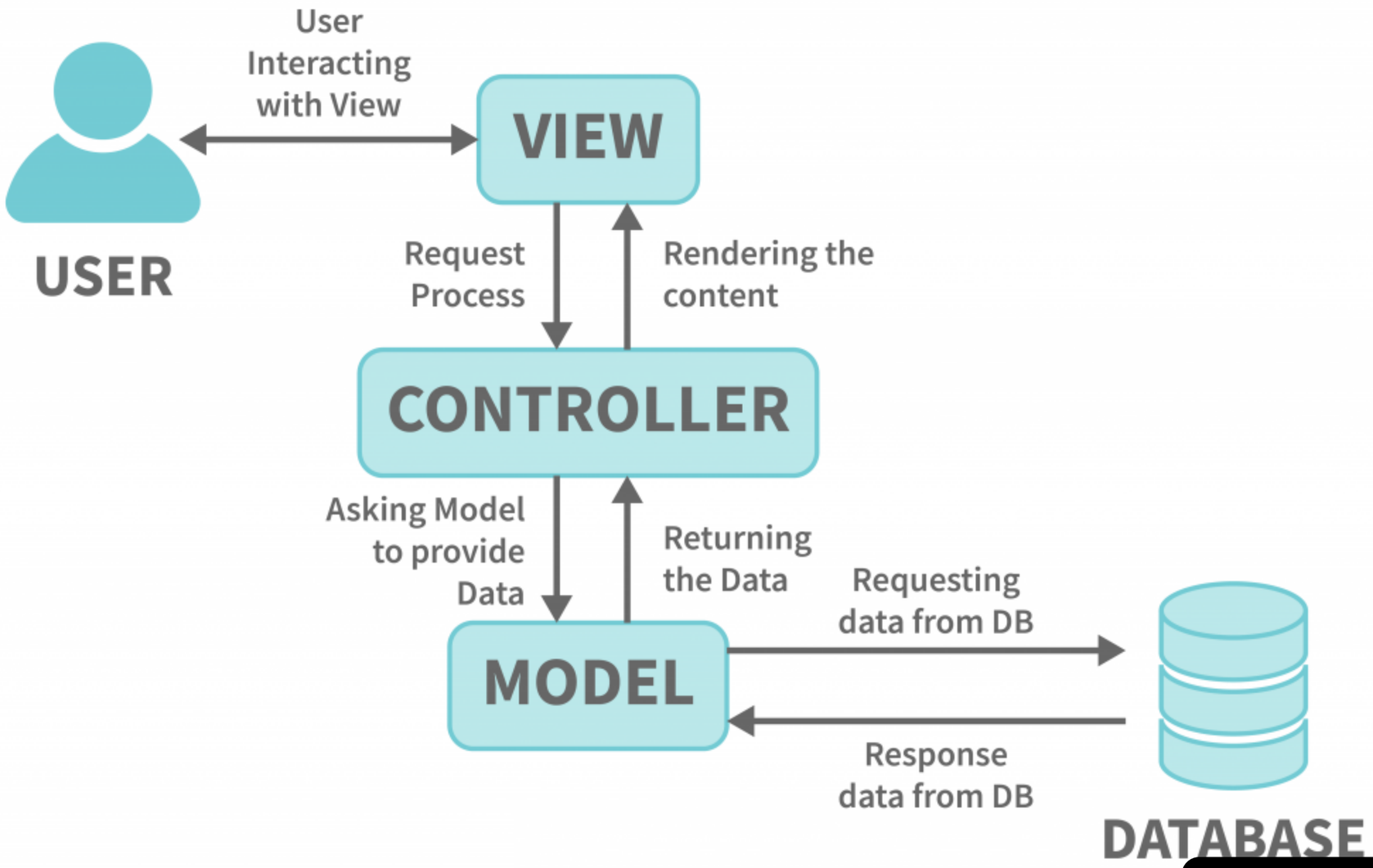
rgupta.mtech@gmail.com

Library vs Framework



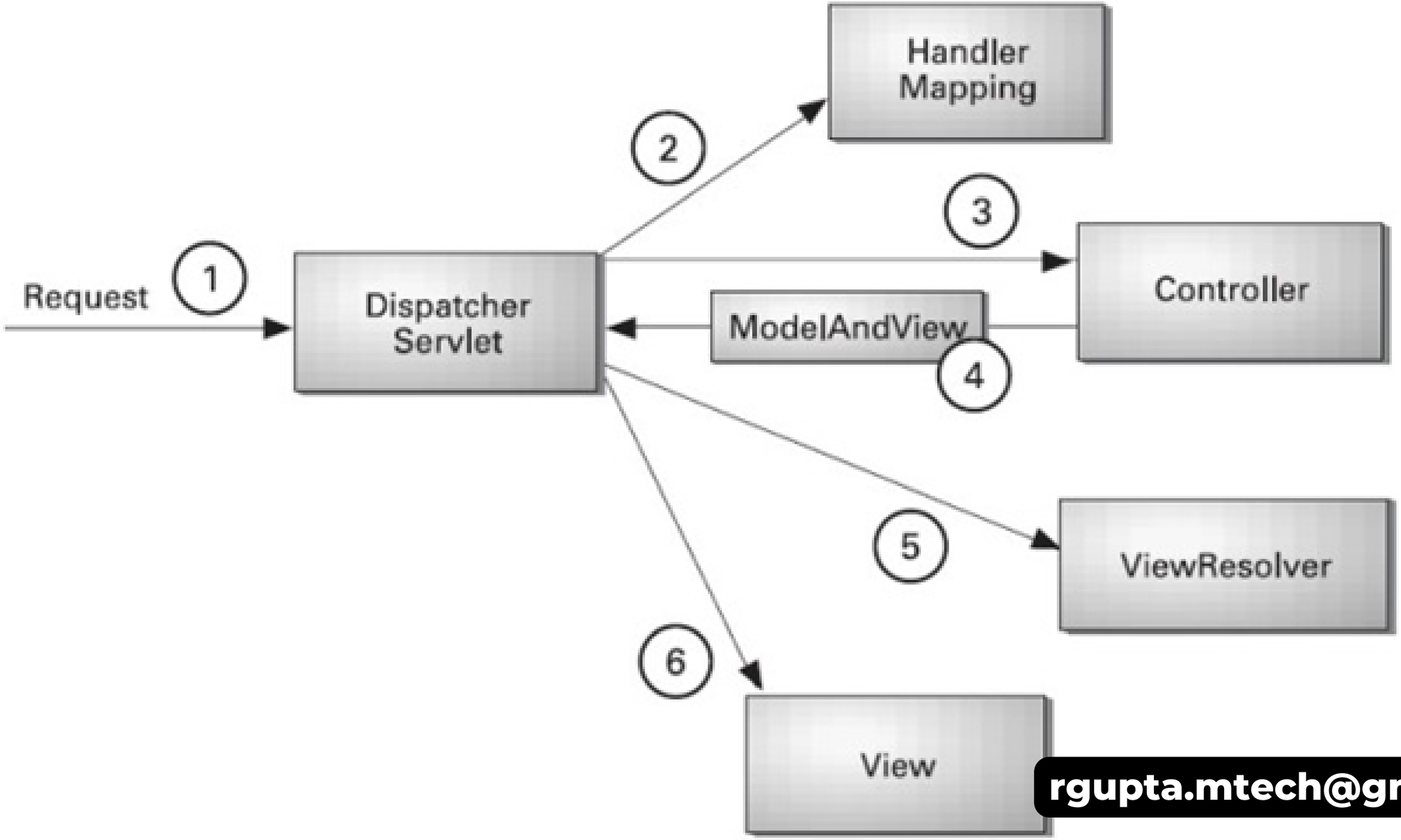
rgupta.mtech@gmail.com

MVC Design Pattern



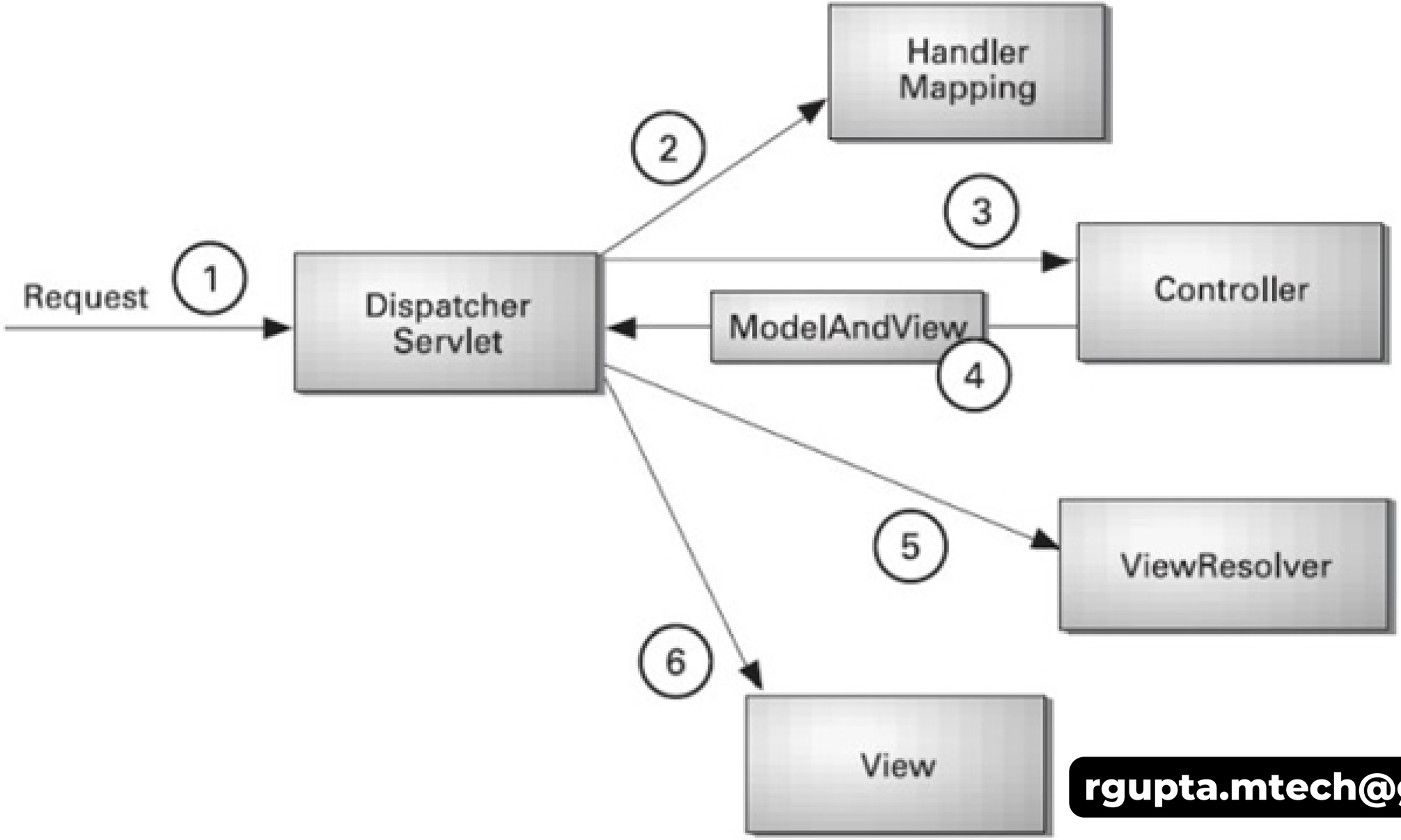
rgupta.mtech@gmail.com

MVC Design Pattern



rgupta.mtech@gmail.com

MVC Design Pattern



rgupta.mtech@gmail.com