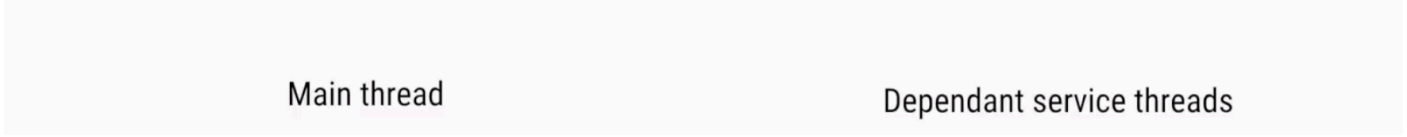
Multithreading 6 (Part1): Phaser vs CountDownLatch vs CyclicBarrier

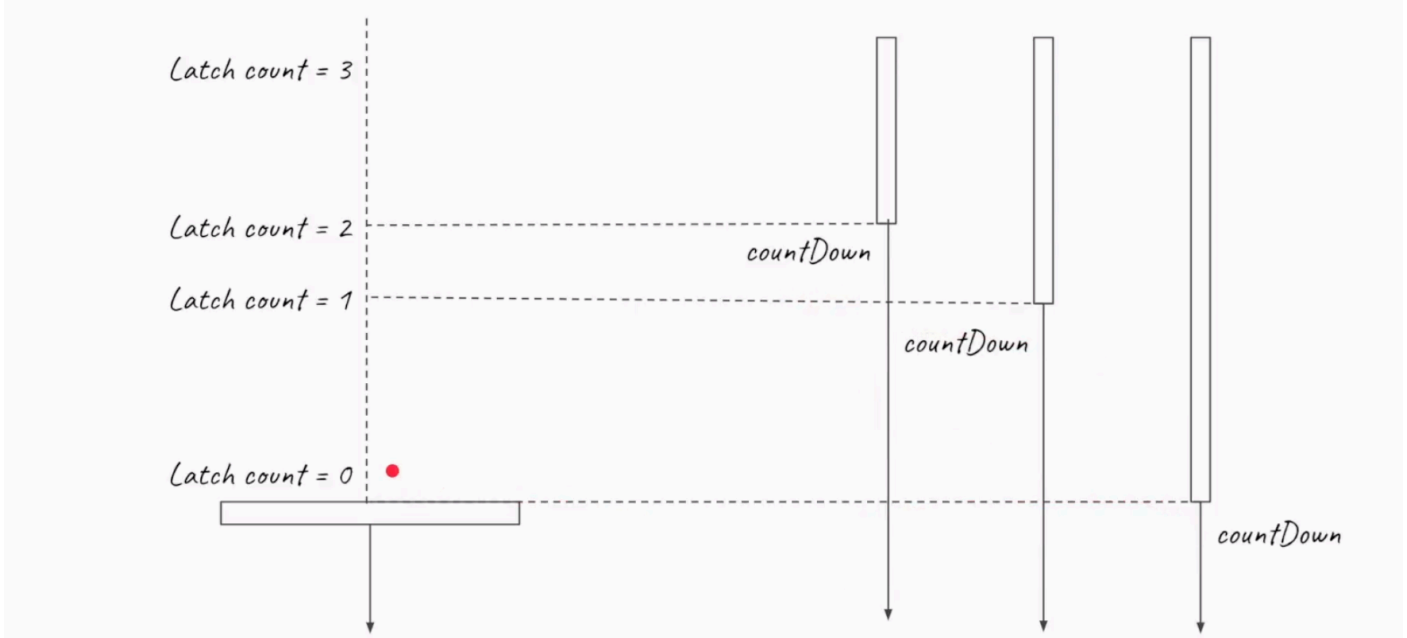
CountDownLatch:

```
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
    }
}
```

Here we want main thread to wait for 3 dependent service to initilize. Only when 3 dependent service are initilize then in main thread main thread complete **latch.await** and then only main thread could start further operation.

We have intilize countdown latch with 3 as this is having 3 dependent service.



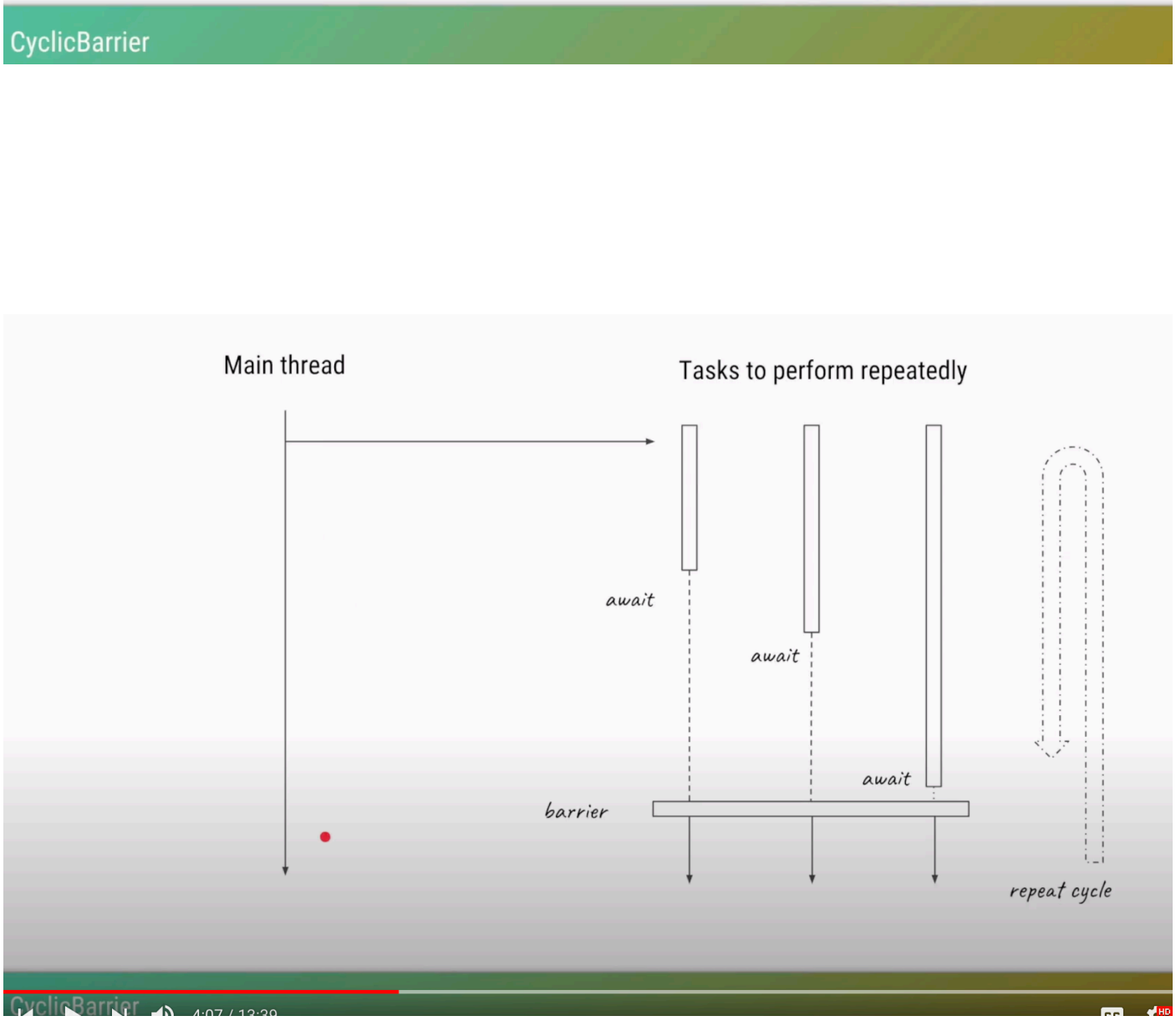


CountDownLatch

CyclicBarrier

```
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
            }
        }
    }
```



Once all 3 three thread could complete some task and would say **barrier. await** then only they move further.

Phaser:

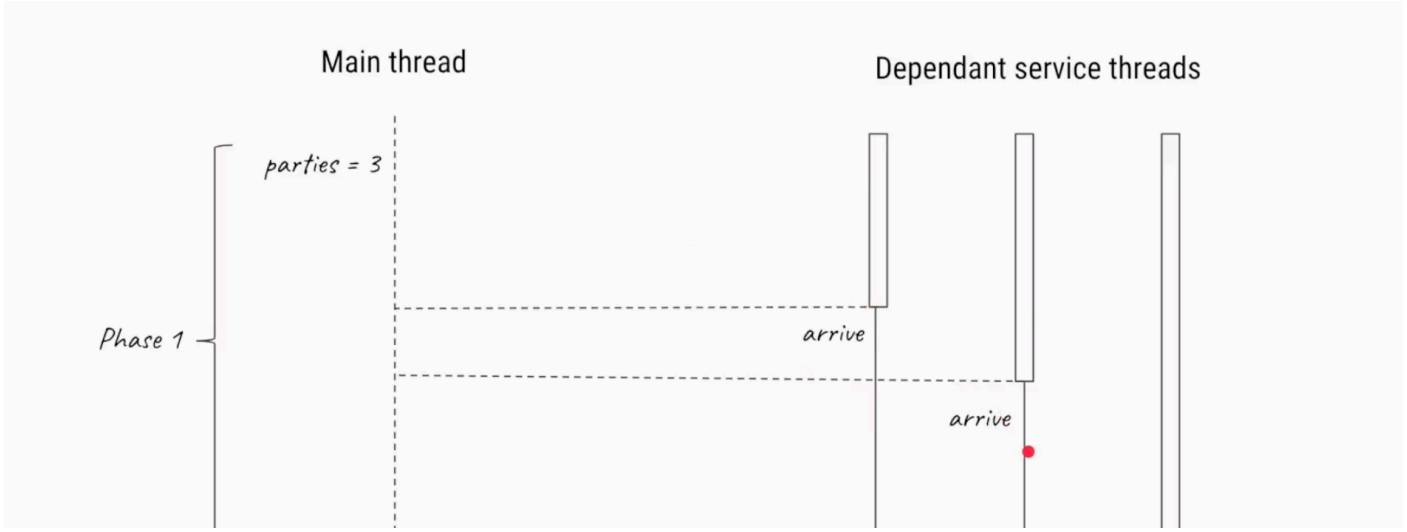
It consists of

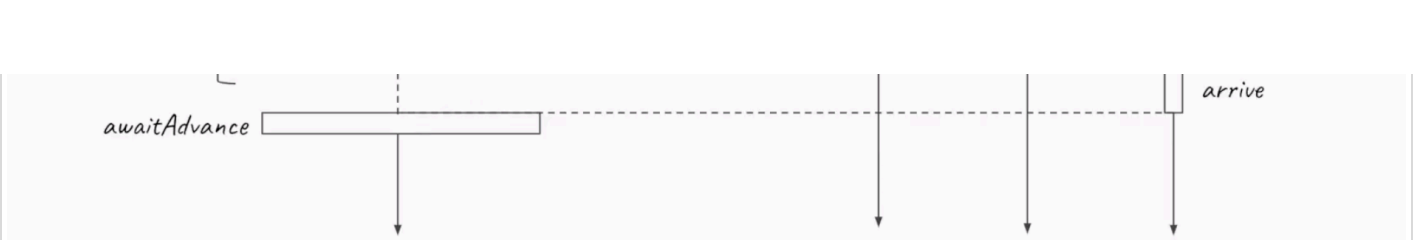
1. **Countdown latch**
2. **Cyclic Barrier**
3. **Some more functionality and flexibility**

Phaser as Countdown Latch:

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

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

    Phaser phaser = new Phaser( parties: 3);
    executor.submit(new DependentService(phaser));
    executor.submit(new DependentService(phaser));
    executor.submit(new DependentService(phaser));

    phaser.awaitAdvance( phase: 1); Similar to await()

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

public static class DependentService implements Runnable {

    private Phaser phaser;
    public DependentService(Phaser phaser) { this.phaser = phaser; }

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

Phaser as CountDownLatch



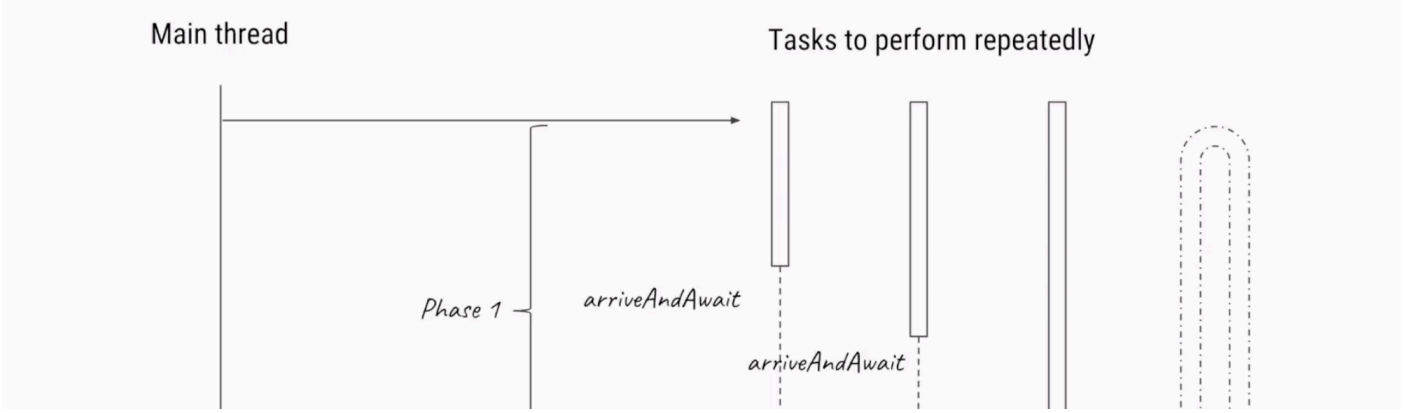


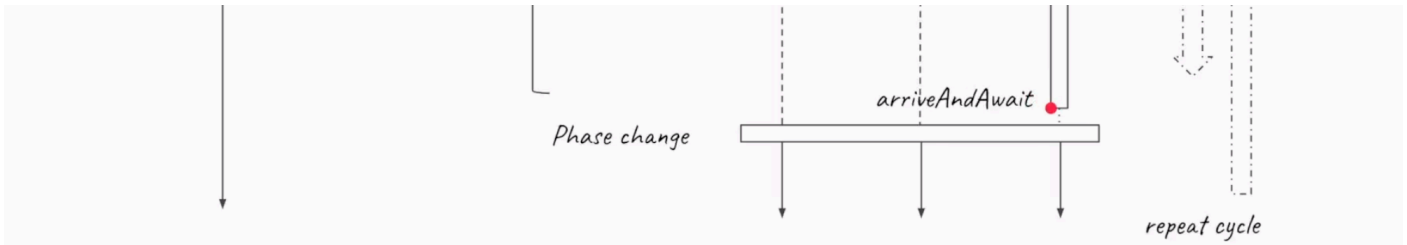
Phaser as CountDownLatch

Phaser at Cyclic Barrier:

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

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

    Phaser phaser = new Phaser( parties: 3);
    executor.submit(new Task(phaser));
    executor.submit(new Task(phaser));
    executor.submit(new Task(phaser));

    Thread.sleep( millis: 3000);
}

public static class Task implements Runnable {

    private Phaser phaser;
    public Task(Phaser phaser) { this.phaser = phaser; }

    @Override
    public void run() {

        while (true) {
            phaser.arriveAndAwaitAdvance(); Similar to barrier.await()
            // send message to corresponding system
        }
    }
}
```

Phaser as CyclicBarrier





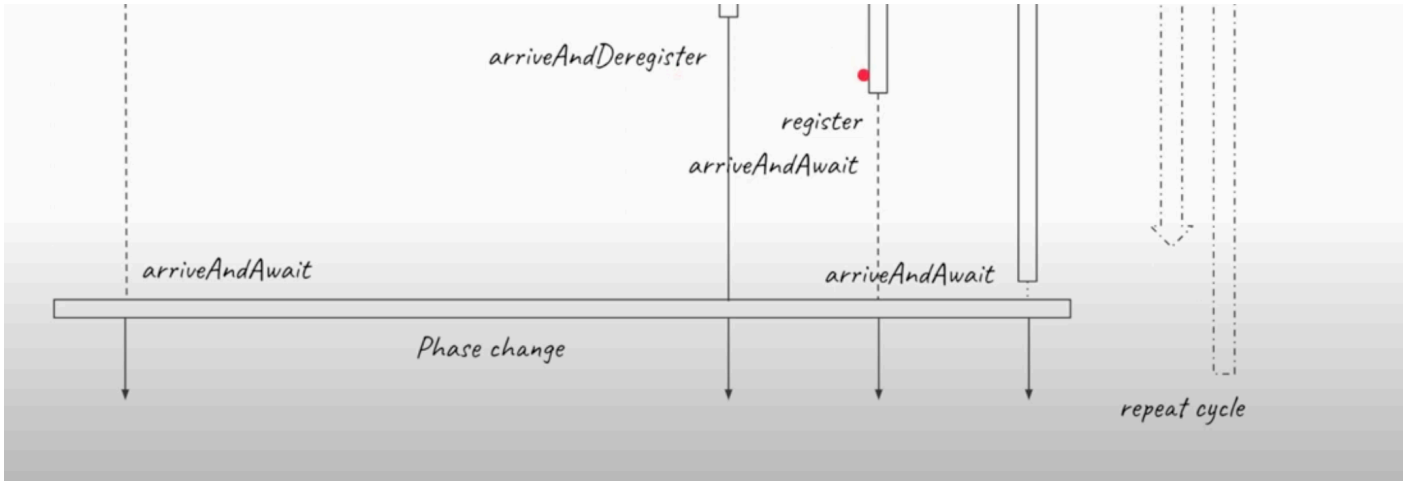
CyclicBarrier

Dynamic Registry using Phaser:

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

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

    Phaser phaser = new Phaser( parties: 1);  self-register

    executor.submit(new Service(phaser));
    executor.submit(new Service(phaser));

    phaser.arriveAndAwaitAdvance();

    phaser.bulkRegister( parties: 4);    bulk register later
}

public static class Service implements Runnable {

    private Phaser phaser;
    public Service(Phaser phaser) { this.phaser = phaser; }

    @Override
    public void run() {
        phaser.register();
        // some operations
        phaser.arrive();
        // other operations
    }
}
```

Allow threads to register themselves

final count 7 parties register



Phaser methods

new Phaser(partyCount)	*Set count of participating parties*
register	*Allow party to register itself*
bulkRegister(partyCount)	*Bulk register extra parties post constructor*
arrive	*Parties can arrive (and continue)*
arriveAndAwaitAdvance	*Parties can arrive and await for all parties*
arriveAndDeregister	*Parties can arrive, deregister (and continue)*