ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

# First-Class Tasks

Parallel Programming in Scala

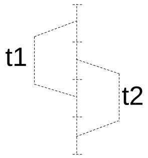
Viktor Kuncak

## More flexible construct for parallel computation

```
val (v1, v2) = parallel(e1, e2)
```

we can write alternatively using the `task` construct:

```
val t1 = task(e1)
val t2 = task(e2)
val v1 = t1.join
val v2 = t2.join
```



`t = task(e)` starts computation `e` "in the background"

- `t` is a *task*, which performs computation of `e`
- current computation proceeds in parallel with `t`
- to obtain the result of `e`, use `t.join`
- `t.join` blocks and waits until the result is computed
- subsequent `t.join` calls quickly return the same result

## Task interface

Here is a minimal interface for tasks:

```
def task(c: => A) : Task[A]

trait Task[A] {
  def join: A
}
```

`task` and `join` establish maps between computations and tasks

In terms of the value computed the equation `task(e).join==e` holds

We can omit writing `.join` if we also define an implicit conversion:

```
implicit def getJoin[T](x:Task[T]): T = x.join
```

## Example: Starting Four Tasks

We have seen four-way parallel *p*-norm:

```
val ((part1, part2),(part3,part4)) =
  parallel(parallel(sumSegment(a, p, 0, mid1),
                    sumSegment(a, p, mid1, mid2)),
           parallel(sumSegment(a, p, mid2, mid3),
                    sumSegment(a, p, mid3, a.length)))
power(part1 + part2 + part3 + part4, 1/p)
```

Here is essentially the same computation expressed using `task`:

```
val t1 = task {sumSegment(a, p, 0, mid1)}
val t2 = task {sumSegment(a, p, mid1, mid2)}
val t3 = task {sumSegment(a, p, mid2, mid3)}
val t4 = task {sumSegment(a, p, mid3, a.length)}
power(t1 + t2 + t3 + t4, 1/p)
```

## Can we define parallel using task?

Suppose you are allowed to use `task`

Implement `parallel` construct as a method using `task`

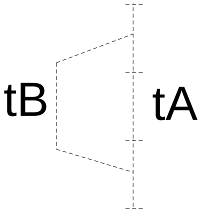
```
def parallel[A, B](cA: => A, cB: => B): (A, B) = {
  ...
}
```

## Can we define parallel using task?

Suppose you are allowed to use `task`

Implement `parallel` construct as a method using `task`

```
def parallel[A, B](cA: => A, cB: => B): (A, B) = {
  val tB: Task[B] = task { cB }
  val tA: A = cA
  (tA, tB.join)
}
```



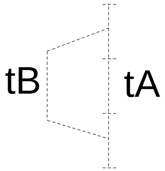
## What is wrong with parallelWrong definition?

```
// CORRECT
def parallel[A, B](cA: => A, cB: => B): (A, B) = {
  val tB: Task[B] = task { cB }
  val tA: A = cA
  (tA, tB.join)
}

// WRONG
def parallelWrong[A, B](cA: => A, cB: => B): (A, B) = {
  val tB: B = (task { cB }).join
  val tA: A = cA
  (tA, tB.join)
}
```

## What is wrong with parallelWrong definition?

```
// CORRECT
def parallel[A, B](cA: => A, cB: => B): (A, B) = {
  val tB: Task[B] = task { cB }
  val tA: A = cA
  (tA, tB.join)
}
```



```
// WRONG
def parallelWrong[A, B](cA: => A, cB: => B): (A, B) = {
  val tB: B = (task { cB }).join
  val tA: A = cA
  (tA, tB.join)
}
```