

UNIVERSITÀ DEGLI STUDI DI FIRENZE

CORSO DI LAUREA MAGISTRALE IN INGEGNERIA INFORMATICA

PROGETTO MID-TERM DEL CORSO DI PARALLEL COMPUTING:

## K-Means

Studente: Kevin Maggi

---

Anno Accademico 2021/2022

# Overview

**Contesto:** K-Means clustering

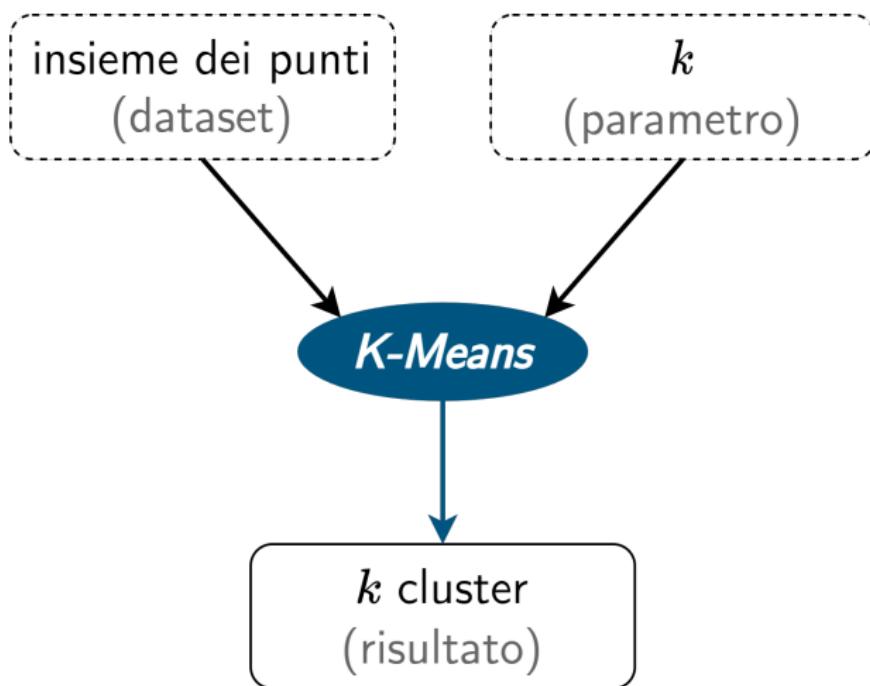
**Applicazione:** Image Color Quantization

**Implementazioni:**  
Java sequenziale  
Java parallelo (esplicito)  
Java parallelo (implicito)

**Obiettivo:** Speed up

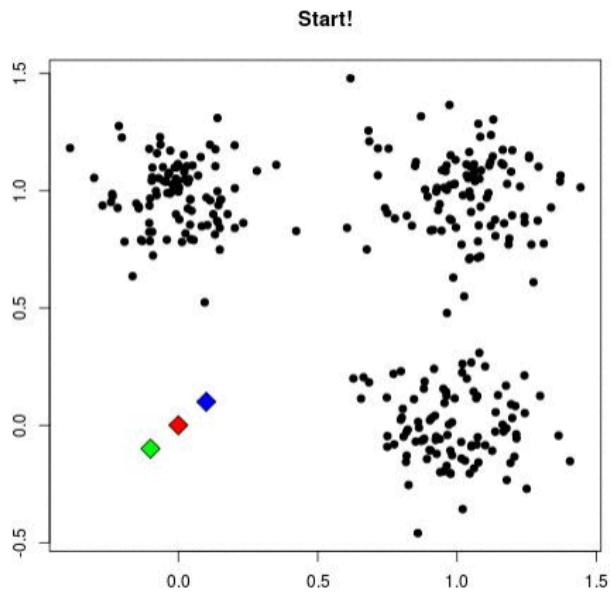
# K-Means

## Principio



# K-Means

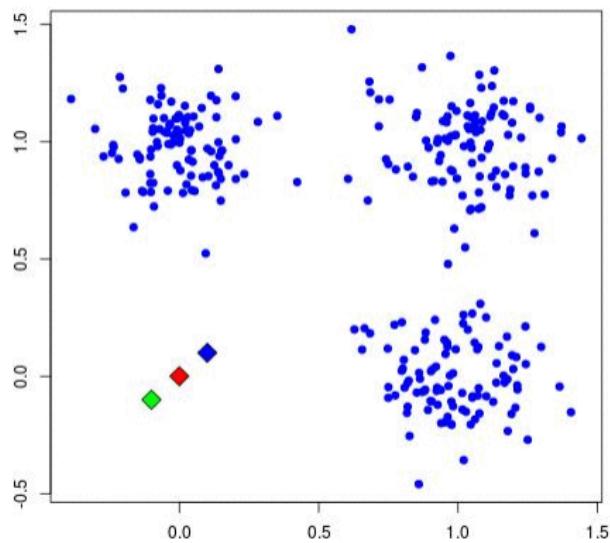
## Funzionamento



# K-Means

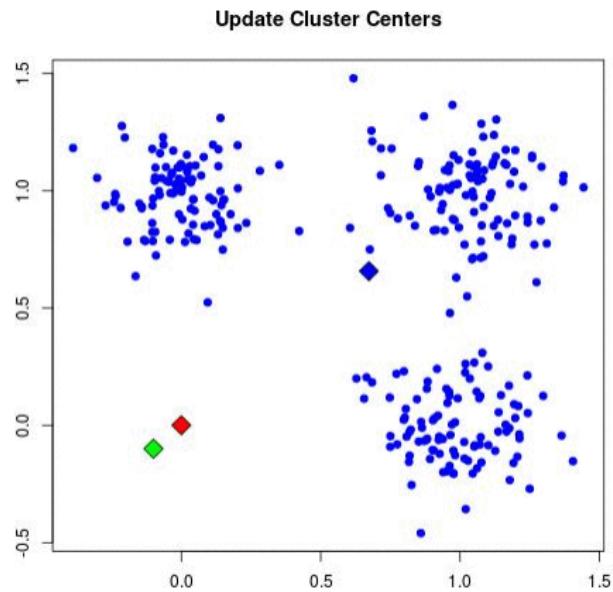
## Funzionamento

Update Cluster Assignments



# K-Means

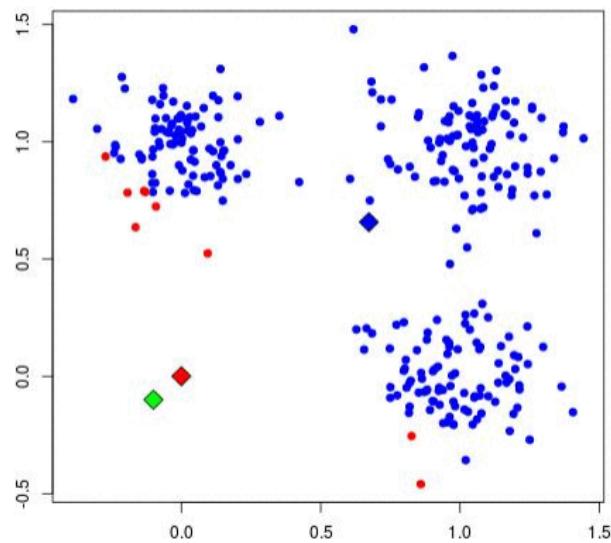
## Funzionamento



# K-Means

## Funzionamento

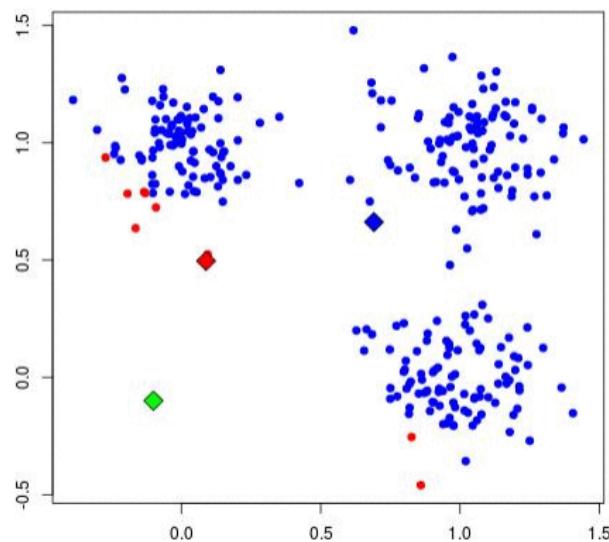
Update Cluster Assignments



# K-Means

## Funzionamento

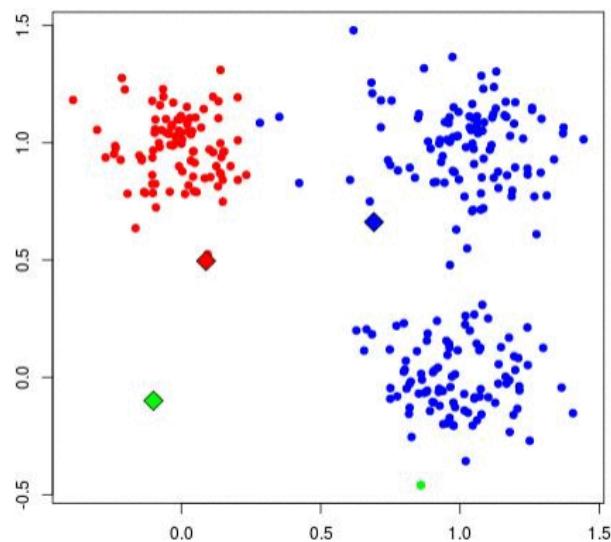
Update Cluster Centers



# K-Means

## Funzionamento

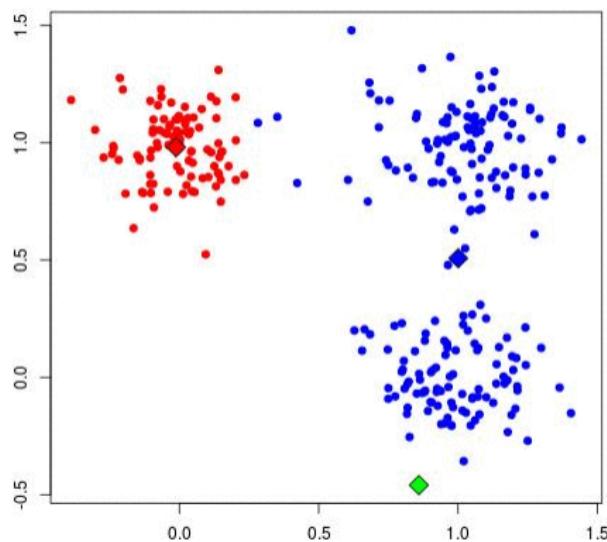
Update Cluster Assignments



# K-Means

## Funzionamento

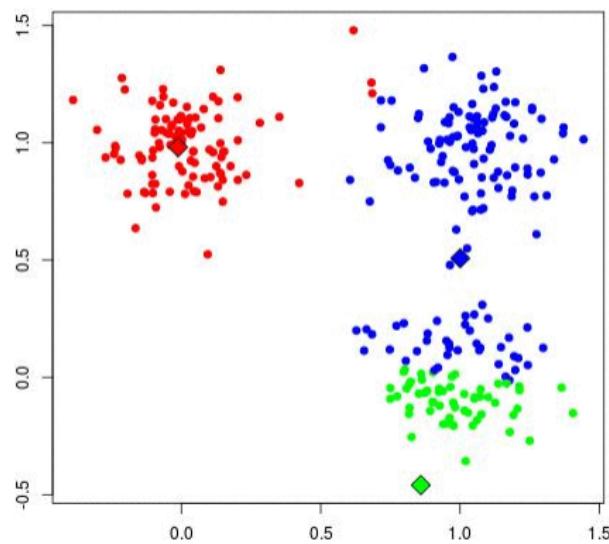
Update Cluster Centers



# K-Means

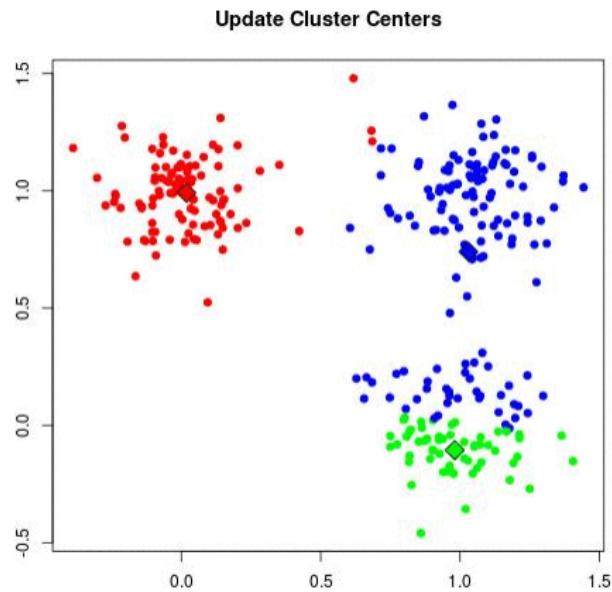
## Funzionamento

Update Cluster Assignments



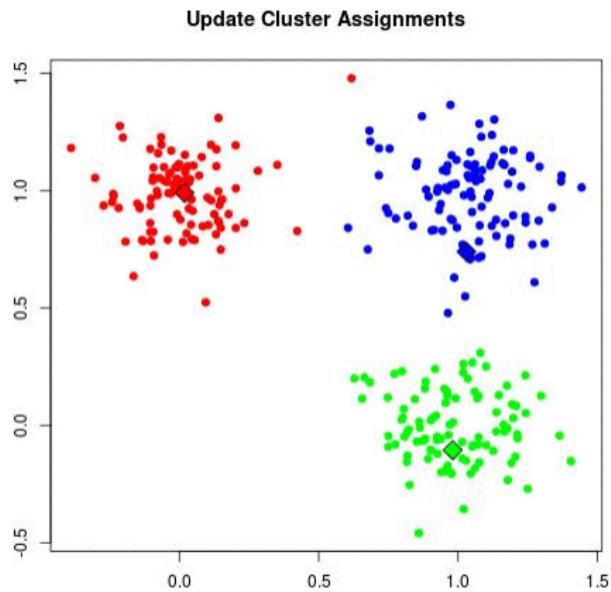
# K-Means

## Funzionamento



# K-Means

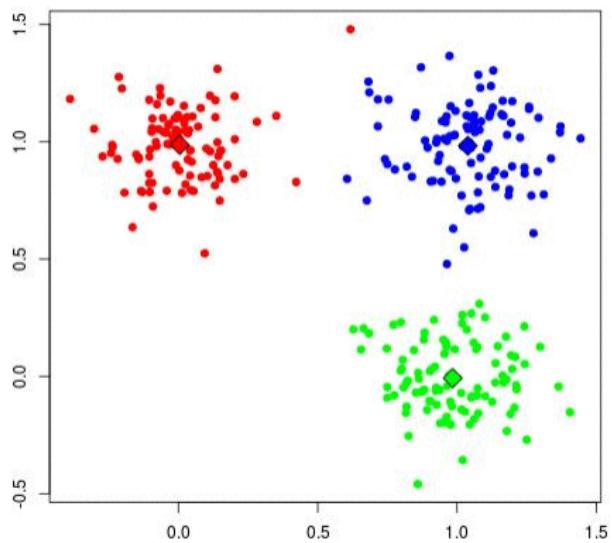
## Funzionamento



# K-Means

## Funzionamento

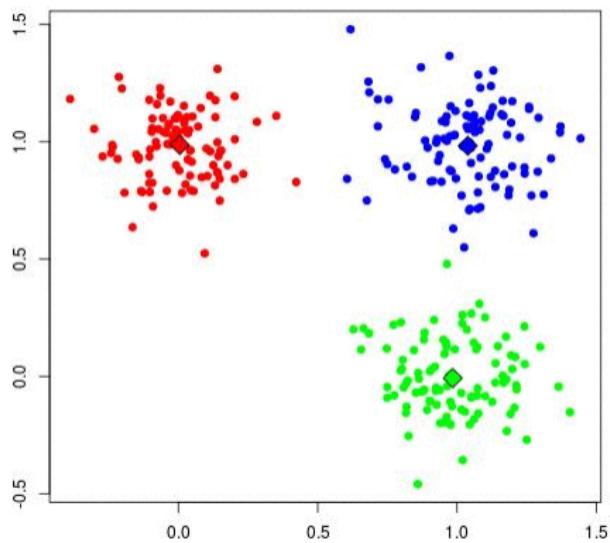
Update Cluster Centers



# K-Means

## Funzionamento

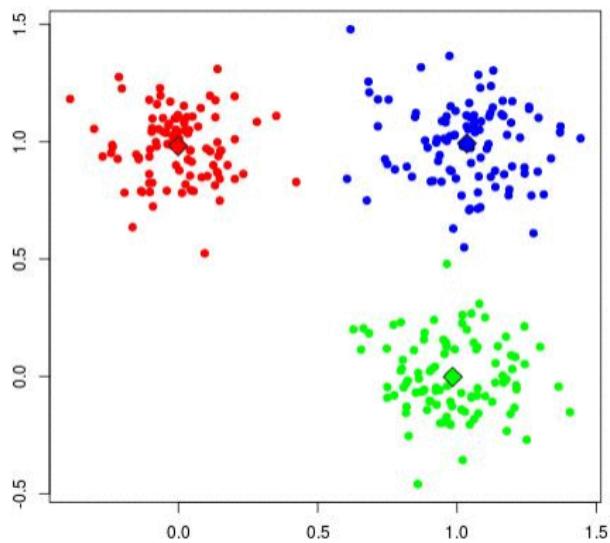
Update Cluster Assignments



# K-Means

## Funzionamento

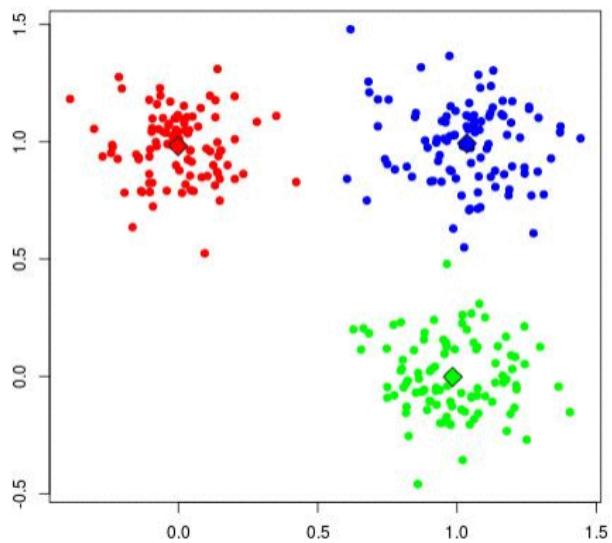
Update Cluster Centers



# K-Means

## Funzionamento

Update Cluster Assignments



# K-Means

## Algoritmo

Select  $k$  points  $m_1^{(0)}, \dots, m_k^{(0)}$  as centroids

**while** not converged **do**

$$S_i^{(t)} = \left\{ x_p \mid \|x_p - m_i^{(t)}\| \leq \|x_p - m_j^{(t)}\| \right\}$$

$$m_i^{(t+1)} = \frac{1}{|S_i^{(t)}|} \sum_{x_j \in S_i^{(t)}} x_j$$

# K-Means

## Centroidi iniziali

Scelta dei centroidi iniziali:

- casuale
- centroidi di una clusterizzazione più grezza
- massimizzando la loro distanza relativa

# K-Means

## Centroidi iniziali

Scelta dei centroidi iniziali:

- casuale
- centroidi di una clusterizzazione più grezza
- massimizzando la loro distanza relativa



# K-Means

## Convergenza

Convergenza quando da un'iterazione alla successiva:

- Formalmente:
  - i punti non cambiano cluster
- Rilassamenti:
  - il diametro dei cluster non cambia
  - i centroidi non cambiano posizione

# K-Means

## Convergenza

Convergenza quando da un'iterazione alla successiva:

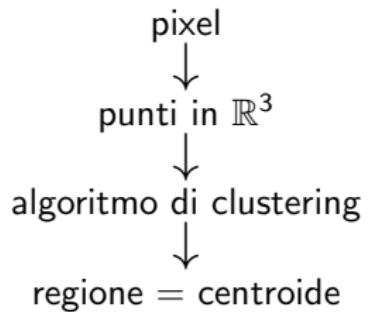
- Formalmente:
  - i punti non cambiano cluster
- Rilassamenti:
  - il diametro dei cluster non cambia
  - i centroidi non cambiano posizione



# Image Color Quantization

**Perché:** Image Segmentation

**Come:**



# Pseudocodice

```
 $m_1^{(0)} \leftarrow \text{first point}$                                 ▷ così è riproducibile
for  $i = 2, \dots, k$ 
     $m_i^{(0)} \leftarrow \text{the point whose min distance from } m_1^{(0)}, \dots, m_{i-1}^{(0)} \text{ is max}$ 
 $\text{clusterization} = [ ]$ 
 $it \leftarrow 0$ 
do
    for each point in dataset do
         $\text{clusterization}[point] \leftarrow i : \|point - m_i^{(it)}\| \leq \|point - m_j^{(it)}\|$ 
    for  $j = 1, \dots, k$ 
         $S_j^{(it)} = \{point | \text{clusterization}[point] = j\}$ 
         $m_j^{(it+1)} = \frac{1}{|S_j^{(it)}|} \sum_{x_w \in S_j^{(it)}} x_w$ 
     $it \leftarrow it + 1$ 
while  $\exists j \in \{1, \dots, k\} : \|m_j^{(it-1)} - m_j^{(it)}\| > toll$ 
```

# Java

```
centroids ← initialCentroids( $k$ ,  $points$ )
clusterization = [ ]
do
    oldCentroids ← centroids
    updateClusters(clusterization, centroids, points)
    centroids ← newCentroids(clusterization, points)
while checkStop(oldCentroids, centroids)
```

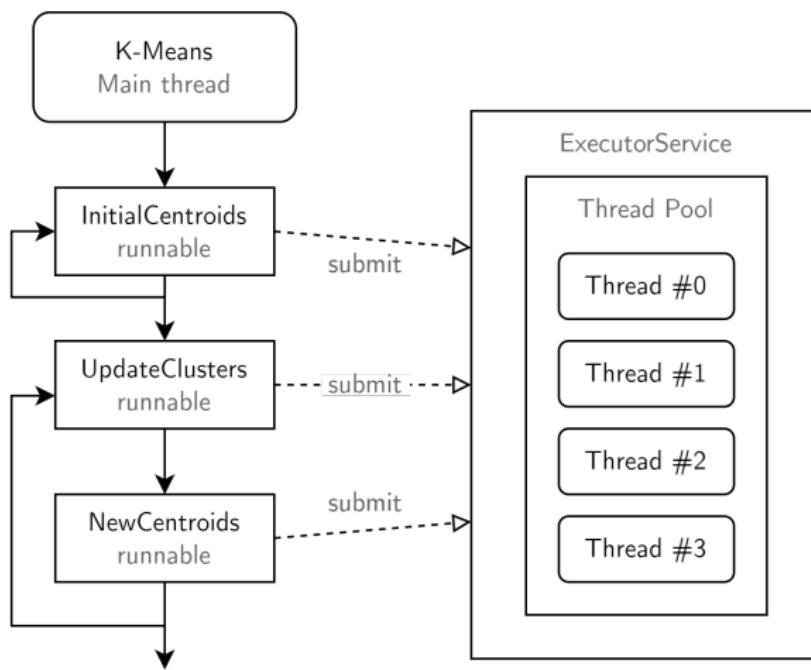
# Profiling

- Hotspot primario (85-88% del tempo di esecuzione):
  - `updateCluster`
    - ↓
    - imbarazzantemente parallelo**
- Hotspot minori (12-14% del tempo di esecuzione):
  - `initialCentroids`
  - `newCentroids`

# Decomposizione dei dati

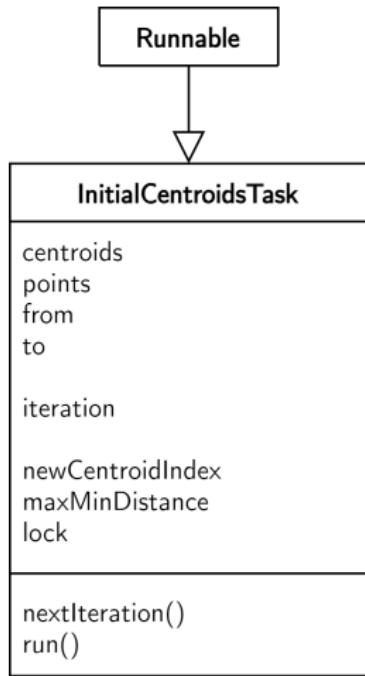


# Java (esplicito)



# Java (esplicito)

## Task InitialCentroids

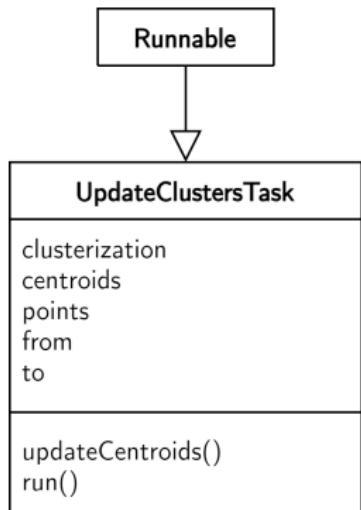


- *centroids/points*: condivisi, ma solo lettura
- *from/to*: decomposizione dei dati
- singola creazione e multipla invocazione:
  - *iteration* tiene conto dell'iterazione
  - *nextIteration()* aggiorna l'iterazione
- selezione candidato privata → selezione del migliore in sezione critica con *lock*

↓  
aggiornamento “atomico” di due variabili  
condivise  
*(newCentroidIndex/maxMinDistance)*

# Java (esplicito)

## Task updateClusters



- *centroids/points*: condivisi, ma solo lettura
- *from/to*: decomposizione dei dati
- singola creazione e multipla invocazione:  
*updateCentroids()*
- *clusterization*: condiviso e usato in scrittura

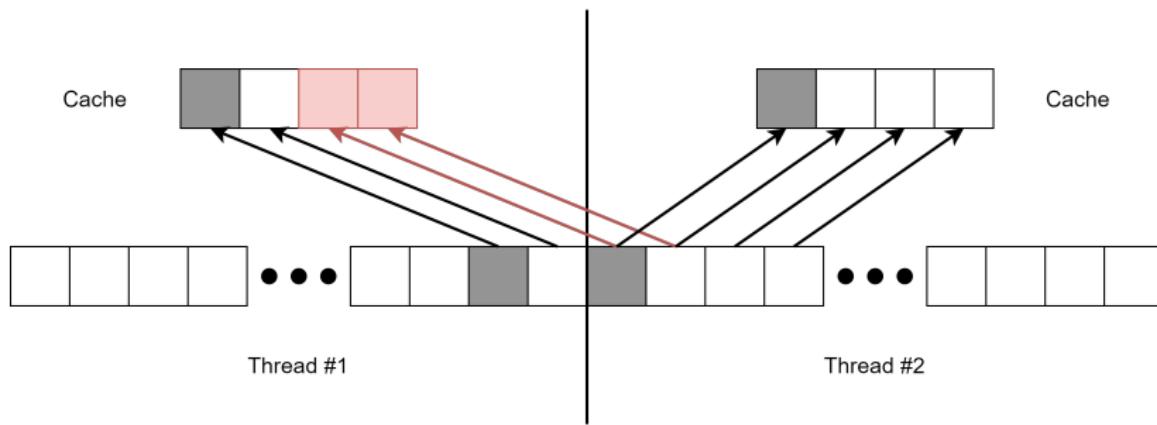


**probabilità di false sharing infinitesimale**

# Java (esplicito)

Task updateClusters

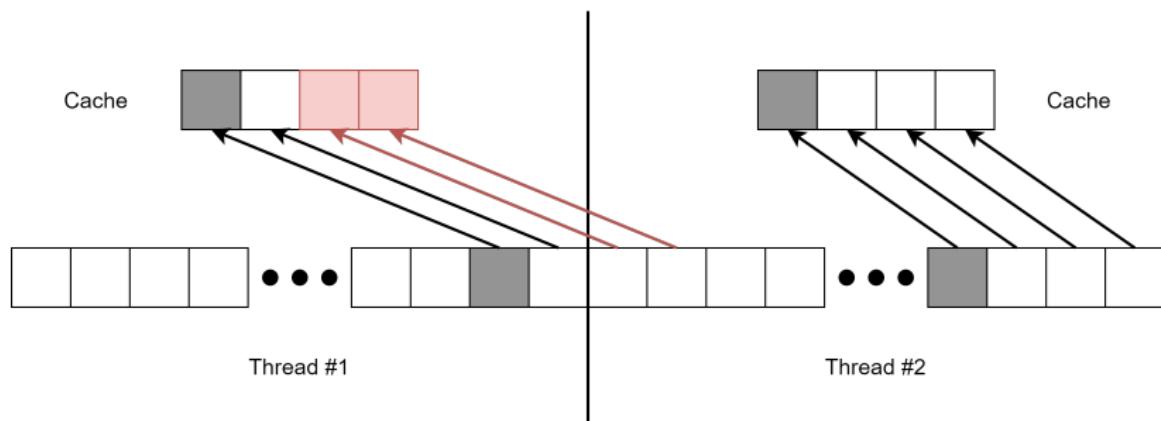
**Probabilità di false sharing infinitesimale:** ogni thread accede a moltissime posizioni contigue:



# Java (esplicito)

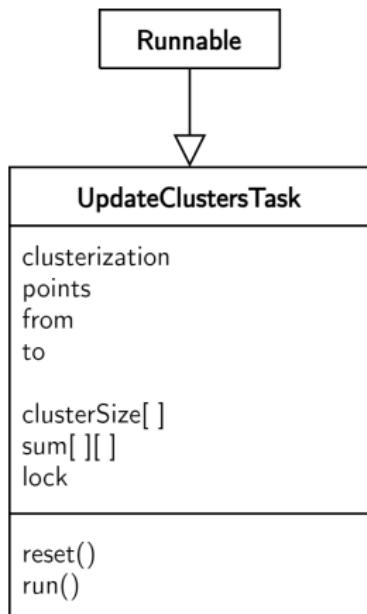
Task updateClusters

**Probabilità di false sharing infinitesimale:** ogni thread accede a moltissime posizioni contigue e tutti i thread procedono “di pari passo”:



# Java (esplicito)

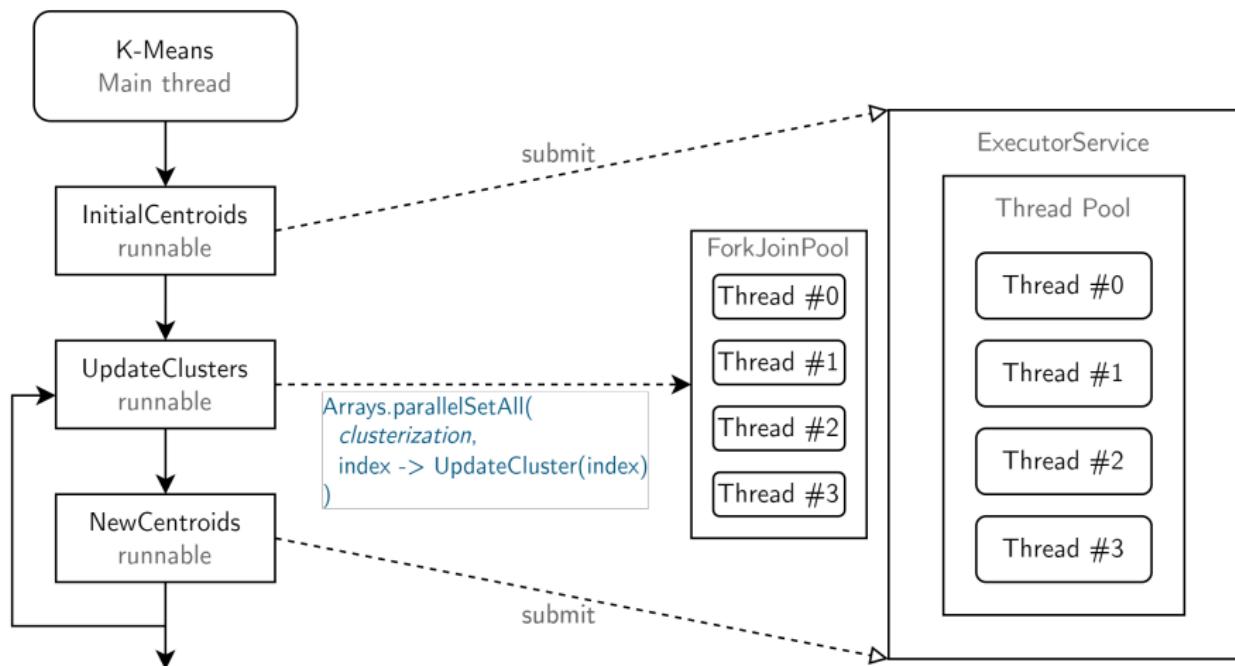
Task newCentroids



- *clusterization/points*: condivisi, ma solo lettura
- *from/to*: decomposizione dei dati
- singola creazione e multipla invocazione: *reset()*
- somme private e poi aggiornamento di *clusterSize/sum* (condivisi) in sezione critica con *lock*

↓  
**con strutture di atomic fenomeno di  
false sharing terrificante**

# Java (implicito)



`java.util.concurrent.ForkJoinPool.common.parallelism`

# Test

## Macchina:

- CPU: Intel Core i7-6500U con **Hyperthreading**:  
2.5GHz up to 3.1GHz, 2 core/4 thread
- RAM: 8GB

## Input:

- immagini: 4K, 5K, 6K
- $k$ : 2, 8, 14, 20

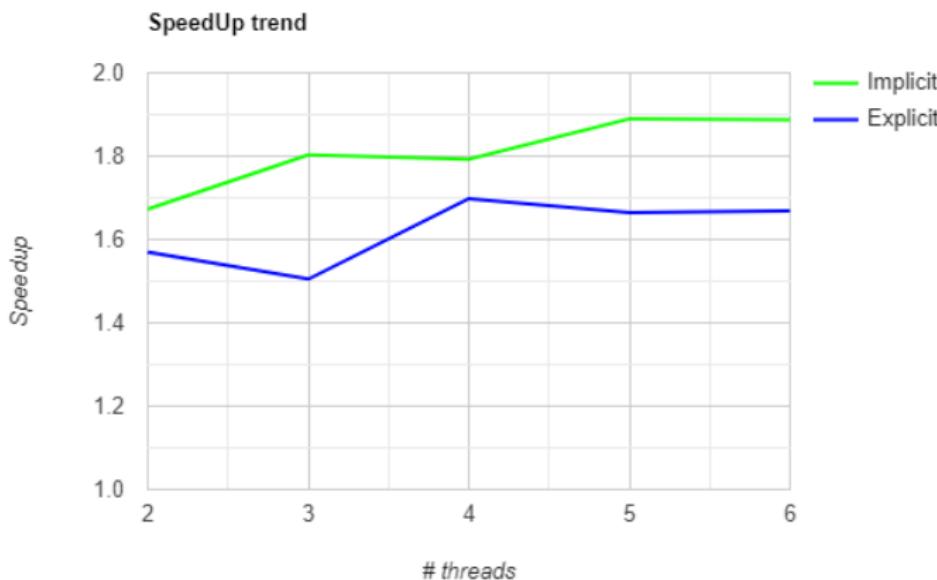
# Risultati

4 thread

Input	Speed up		
	esplicito	implicito	
4K	$k = 2$	1.76	1.71
	$k = 8$	1.65	1.78
	$k = 14$	1.68	1.85
	$k = 20$	1.81	1.94
5K	$k = 2$	1.50	1.50
	$k = 8$	1.63	1.74
	$k = 14$	1.86	<b>2.06</b>
	$k = 20$	1.84	<b>2.02</b>
6K	$k = 2$	1.43	1.43
	$k = 8$	1.74	1.82
	$k = 14$	1.78	1.89
	$k = 20$	1.68	1.80
Media		<b>1.7</b>	
		<b>1.8</b>	

## # thread ottimale

↑ Speed up medio: **1.9** (implicito)



## Extra test

**Domanda:** la funzione nativa implicita davvero sfrutta meglio l'Hyperthreading?

**Risposta:** sì

**Dimostrazione:** su una CPU senza Hyperthreading la versione implicita è più lenta della versione esplicita

Input	Tempo		
	esplicito	implicito	
4K	$k = 8$	11.9s	12.0s
	$k = 20$	37.7s	39.1s
5K	$k = 8$	20.1s	21.7s
	$k = 20$	72.1s	76.2s
6K	$k = 8$	25.5s	26.6s
	$k = 20$	133.2s	146.1s



Grazie per l'attenzione