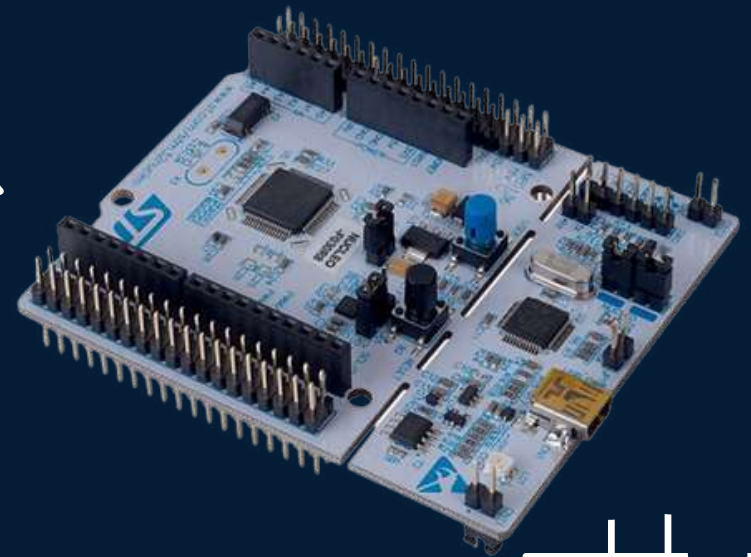
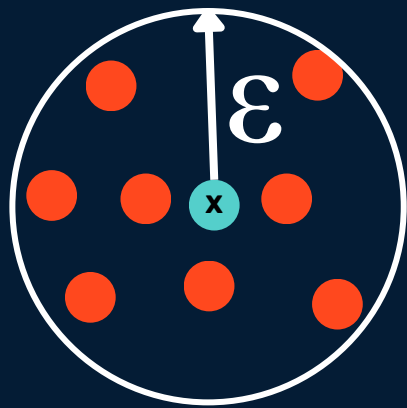
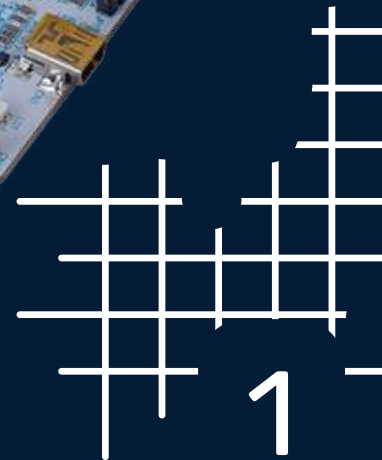


# Algoritmo DBSCAN

Portando o DBSCAN para outra plataforma com mais restrições



Vinícius Menezes Monte  
Paulo Diego De Menezes

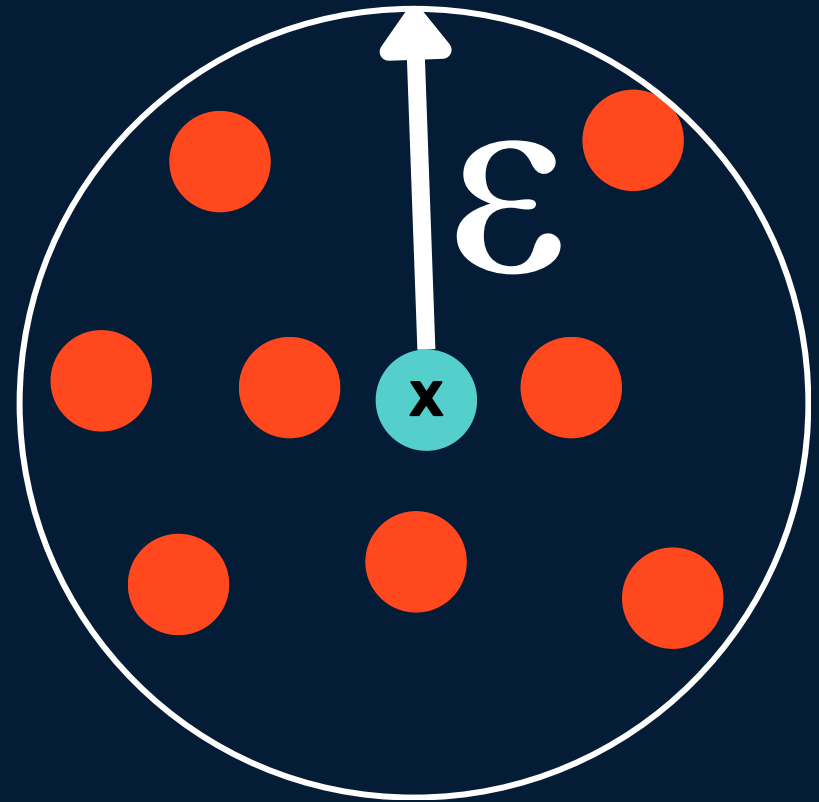
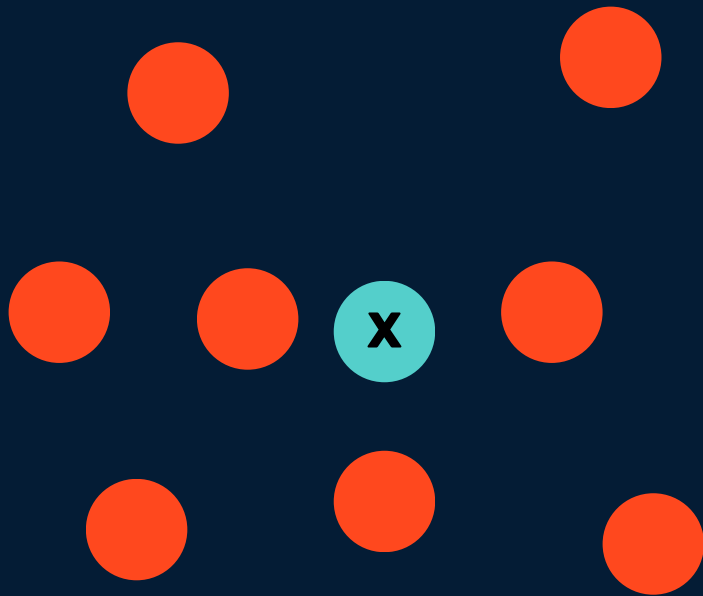




# 0 Algoritmo

# $\epsilon$ – Épsilon

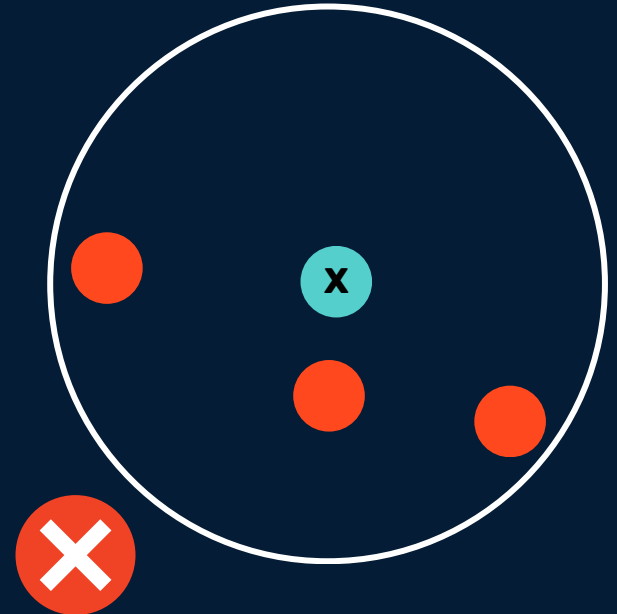
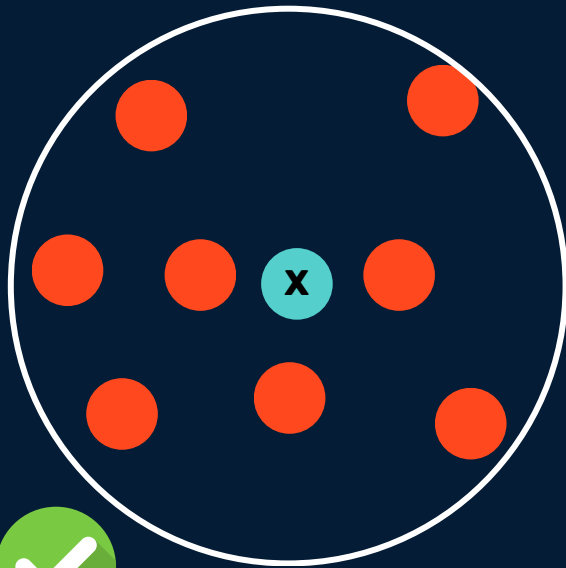
Épsilon é raio da vizinhança de um dado ponto.



# MinPts

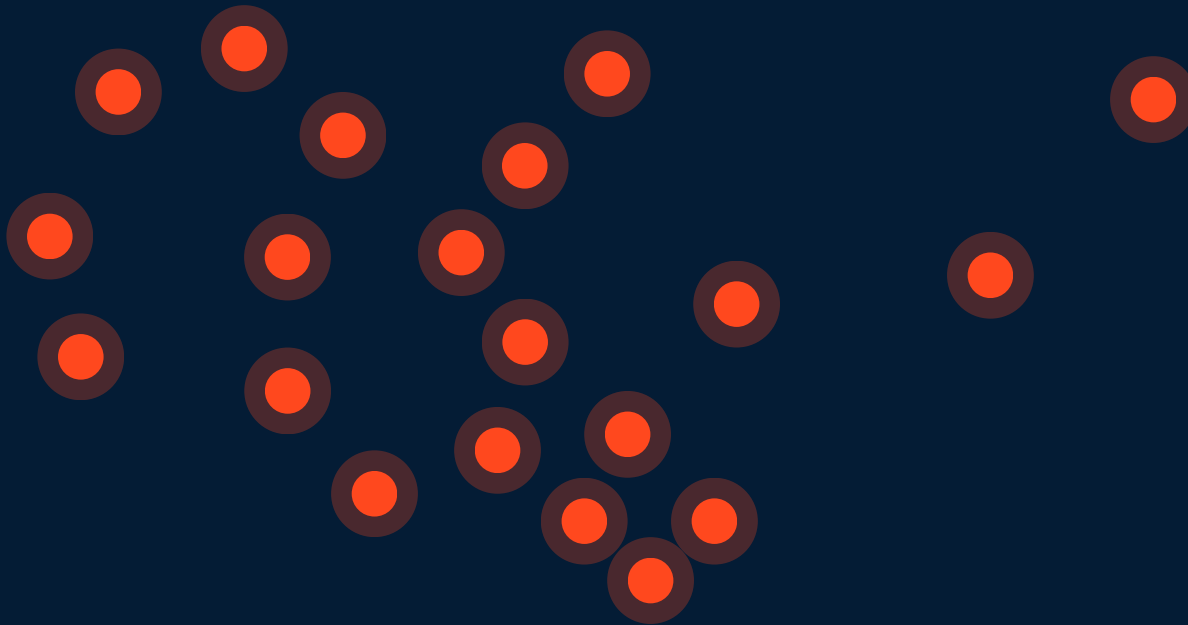
MinPts é o limite de densidade. Se uma vizinhança tiver pelo menos MinPts pontos, ele será considerada densa e poderá fazer parte de um cluster.

**MinPts = 5**



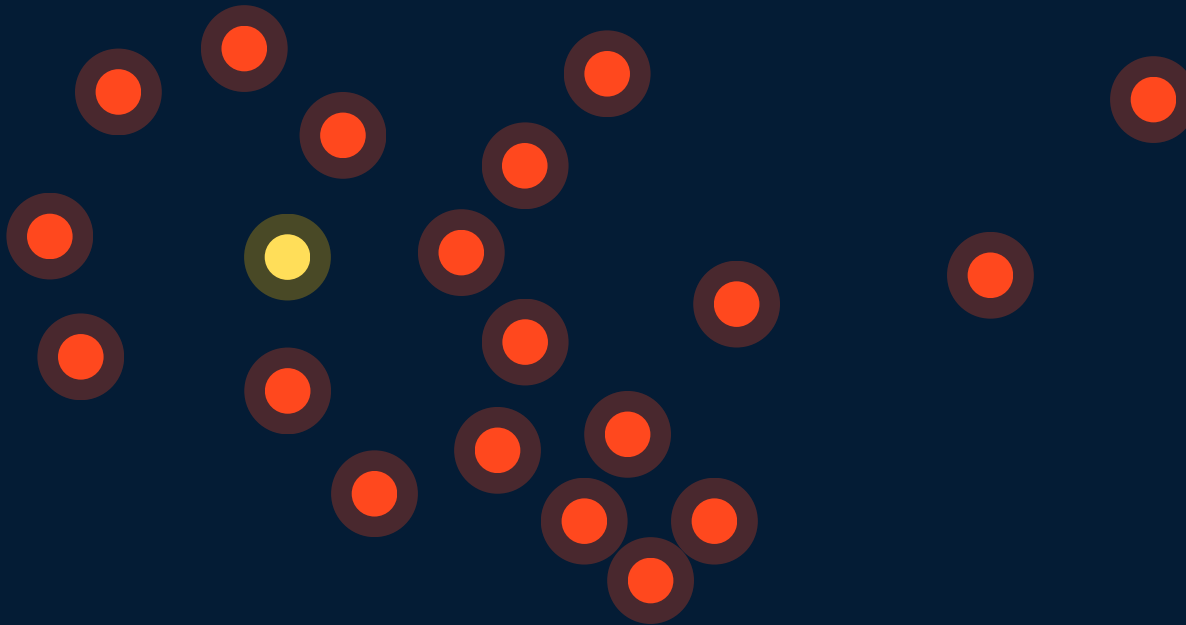
# Procedimento

Os pontos das vizinhanças densas dão origem a mais vizinhanças densas, e esses pontos são agrupados. Quando deixa de ser possível criar mais vizinhanças, um novo ponto de partida é tomado.



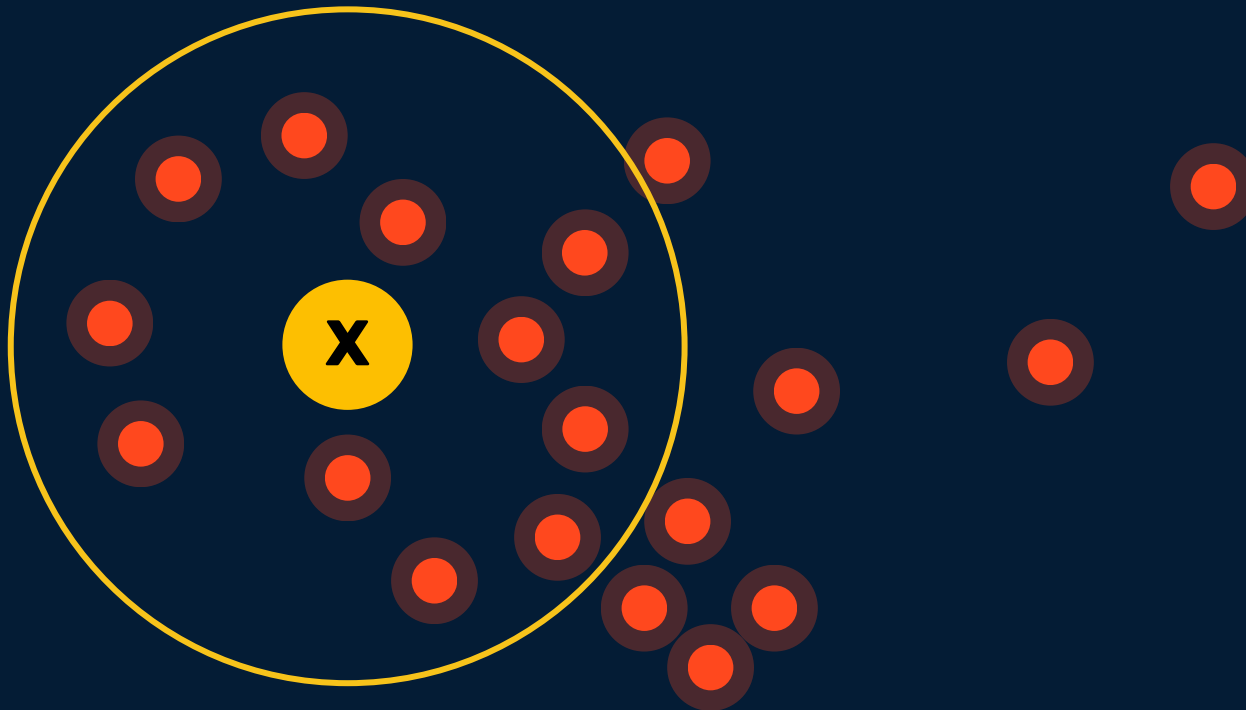
# Procedimento

Os pontos das vizinhanças densas dão origem a mais vizinhanças densas, e esses pontos são agrupados. Quando deixa de ser possível criar mais vizinhanças, um novo ponto de partida é tomado.



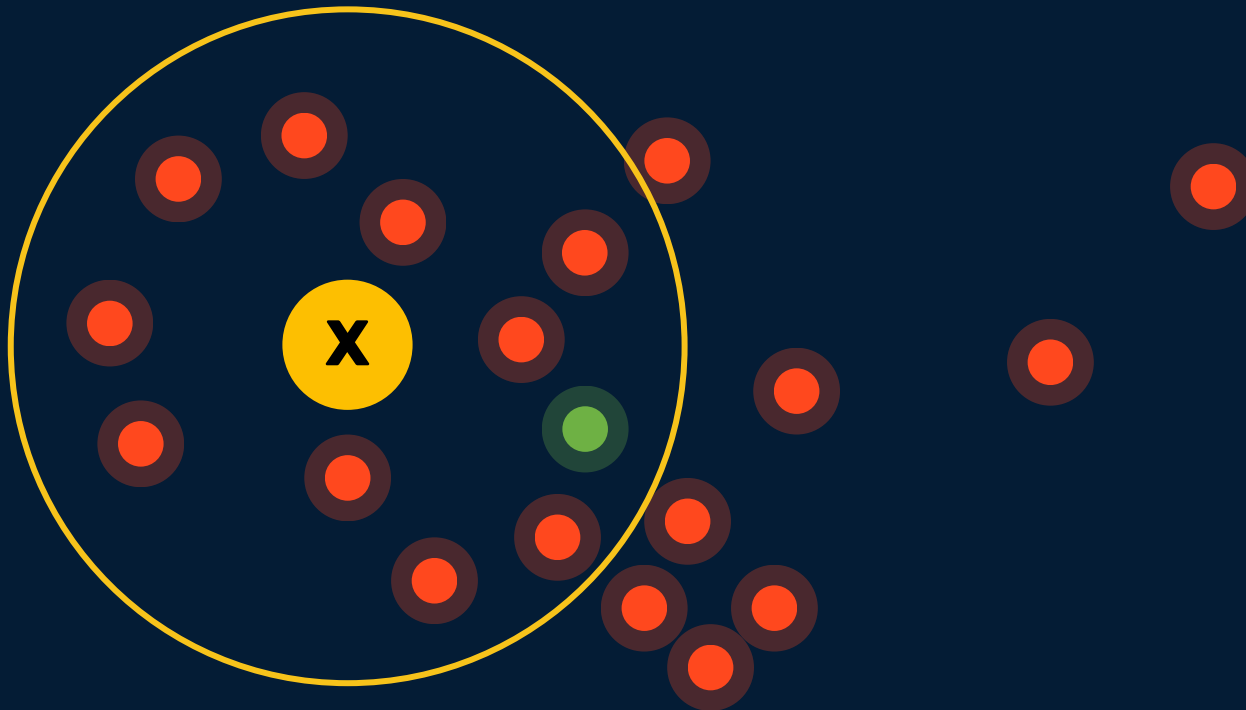
# Procedimento

Os pontos das vizinhanças densas dão origem a mais vizinhanças densas, e esses pontos são agrupados. Quando deixa de ser possível criar mais vizinhanças, um novo ponto de partida é tomado.



# Procedimento

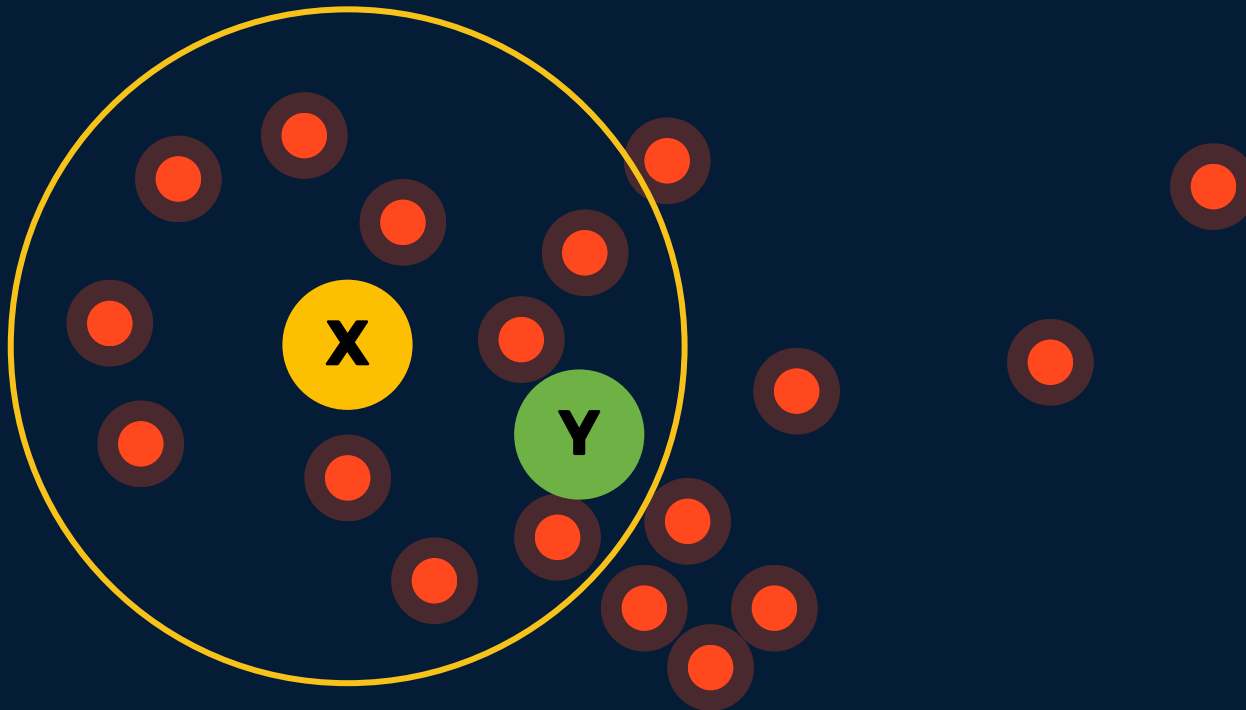
Os pontos das vizinhanças densas dão origem a mais vizinhanças densas, e esses pontos são agrupados. Quando deixa de ser possível criar mais vizinhanças, um novo ponto de partida é tomado.





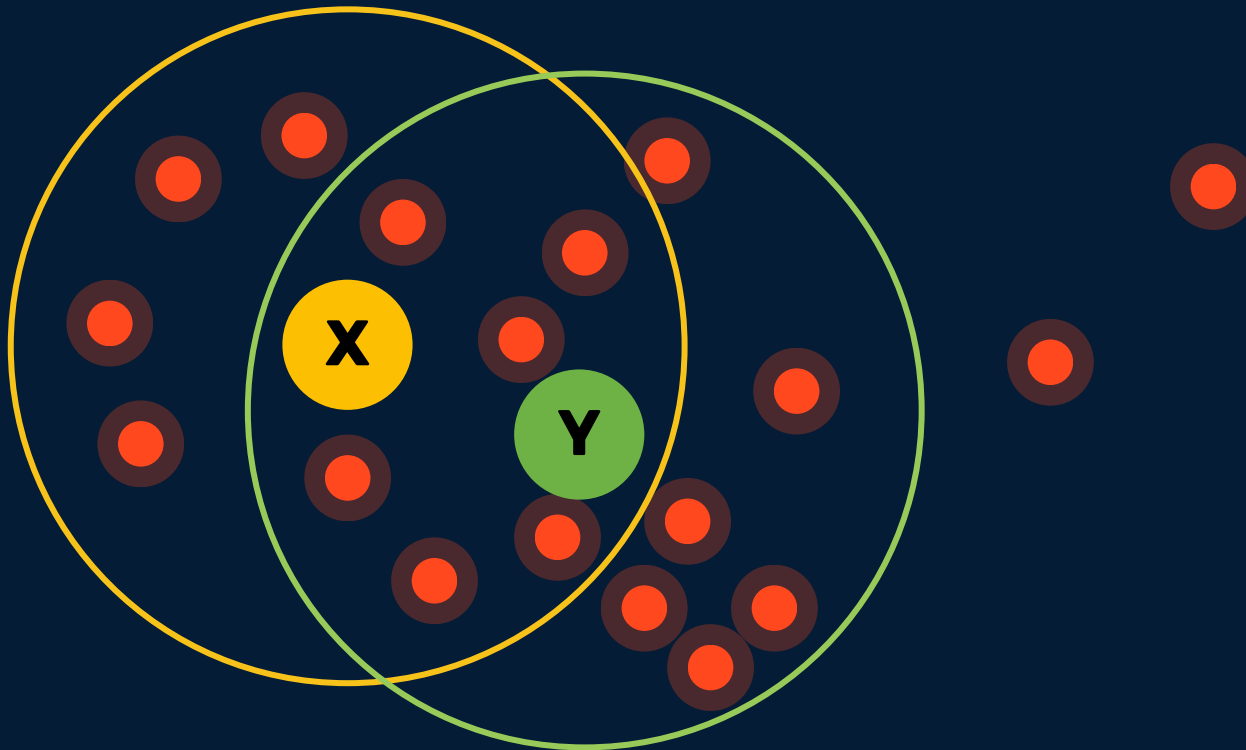
# Procedimento

Os pontos das vizinhanças densas dão origem a mais vizinhanças densas, e esses pontos são agrupados. Quando deixa de ser possível criar mais vizinhanças, um novo ponto de partida é tomado.



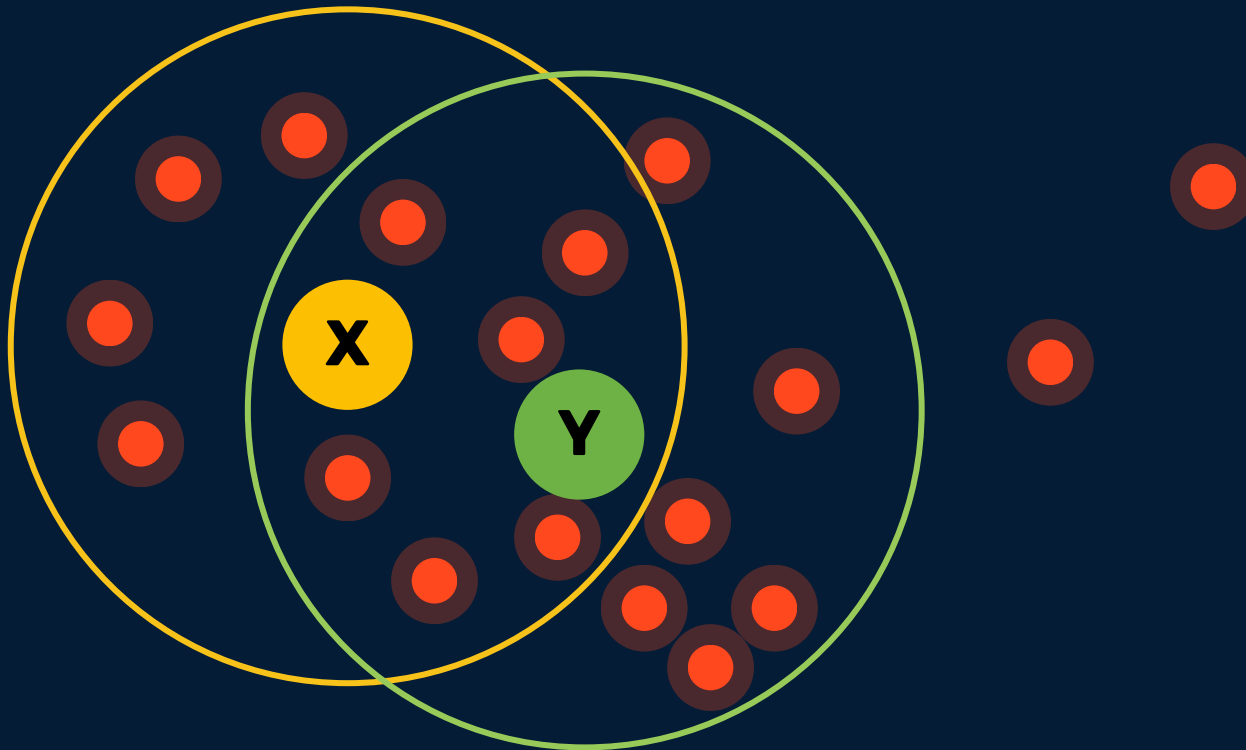
# Procedimento

Os pontos das vizinhanças densas dão origem a mais vizinhanças densas, e esses pontos são agrupados. Quando deixa de ser possível criar mais vizinhanças, um novo ponto de partida é tomado.



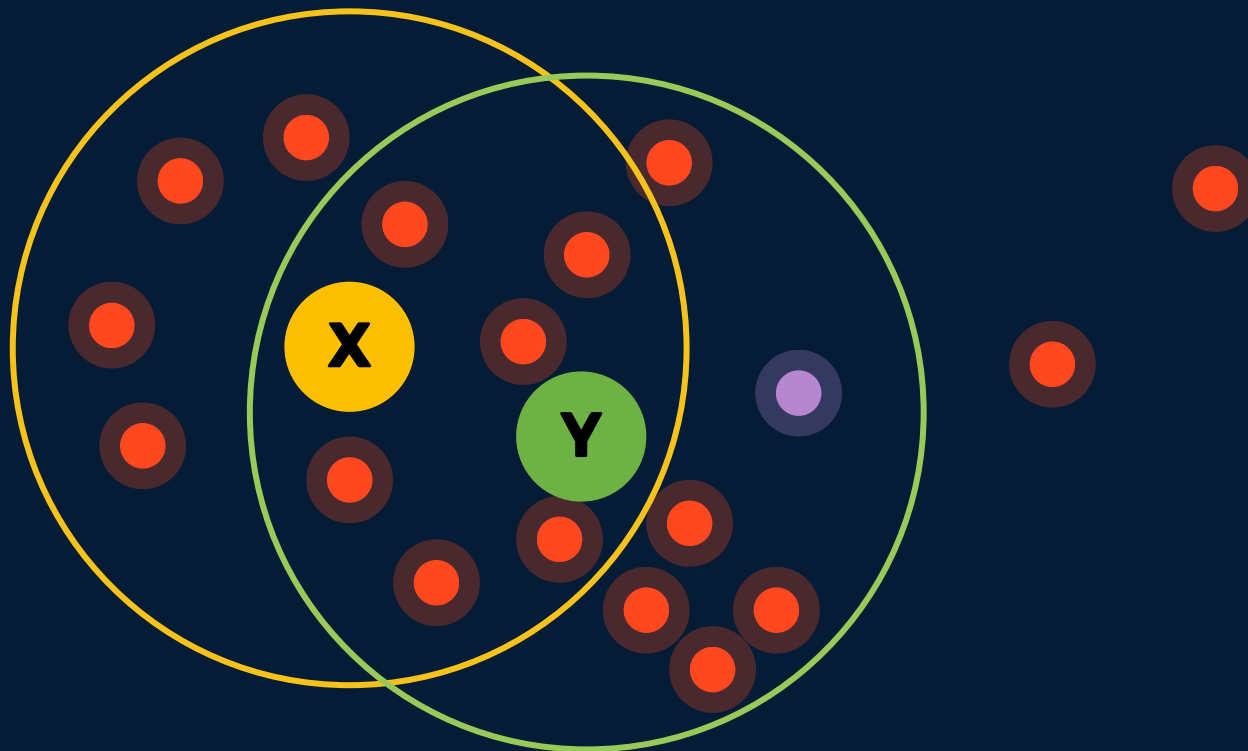
# Procedimento

Os pontos das vizinhanças densas dão origem a mais vizinhanças densas, e esses pontos são agrupados. Quando deixa de ser possível criar mais vizinhanças, um novo ponto de partida é tomado.



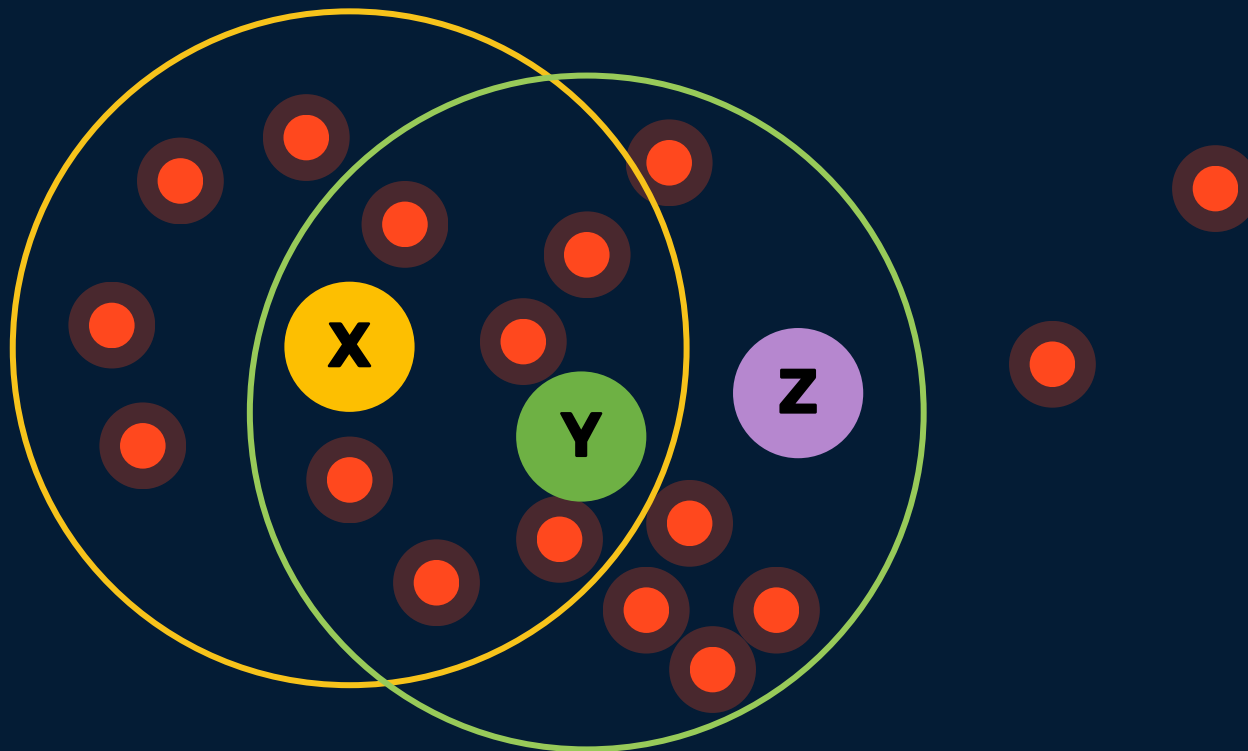
# Procedimento

Os pontos das vizinhanças densas dão origem a mais vizinhanças densas, e esses pontos são agrupados. Quando deixa de ser possível criar mais vizinhanças, um novo ponto de partida é tomado.



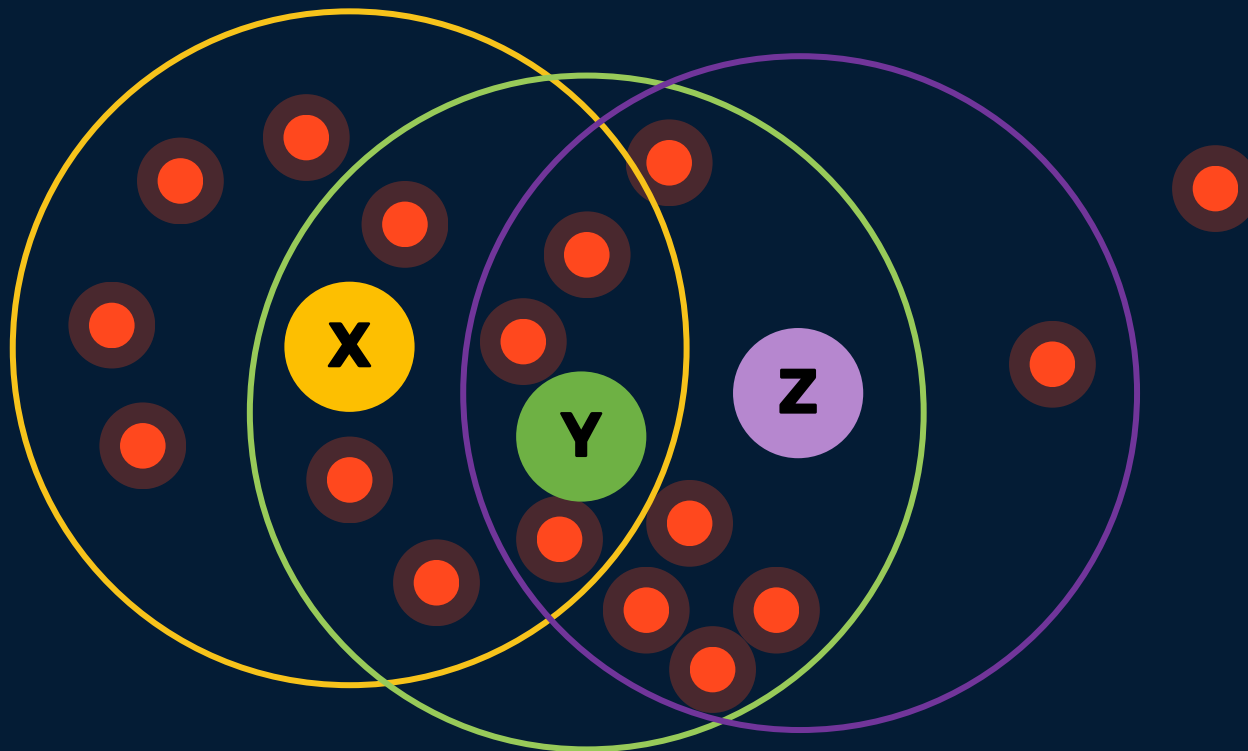
# Procedimento

Os pontos das vizinhanças densas dão origem a mais vizinhanças densas, e esses pontos são agrupados. Quando deixa de ser possível criar mais vizinhanças, um novo ponto de partida é tomado.



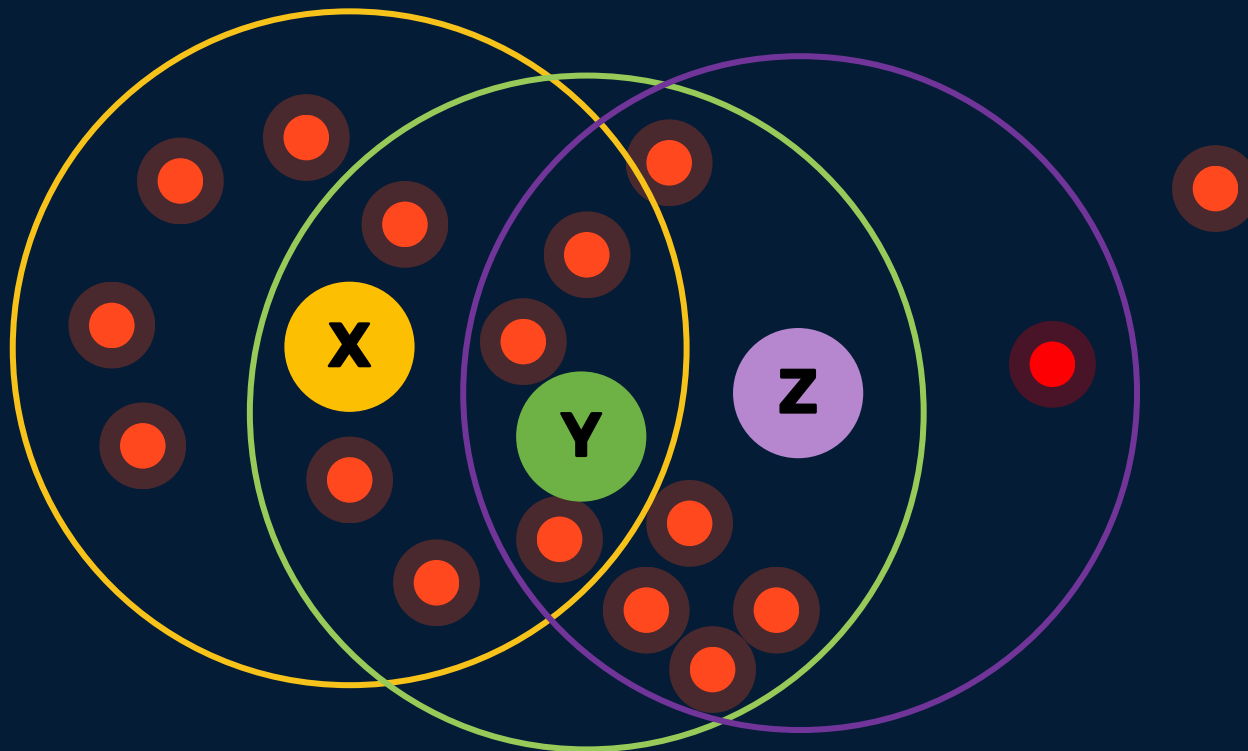
# Procedimento

Os pontos das vizinhanças densas dão origem a mais vizinhanças densas, e esses pontos são agrupados. Quando deixa de ser possível criar mais vizinhanças, um novo ponto de partida é tomado.



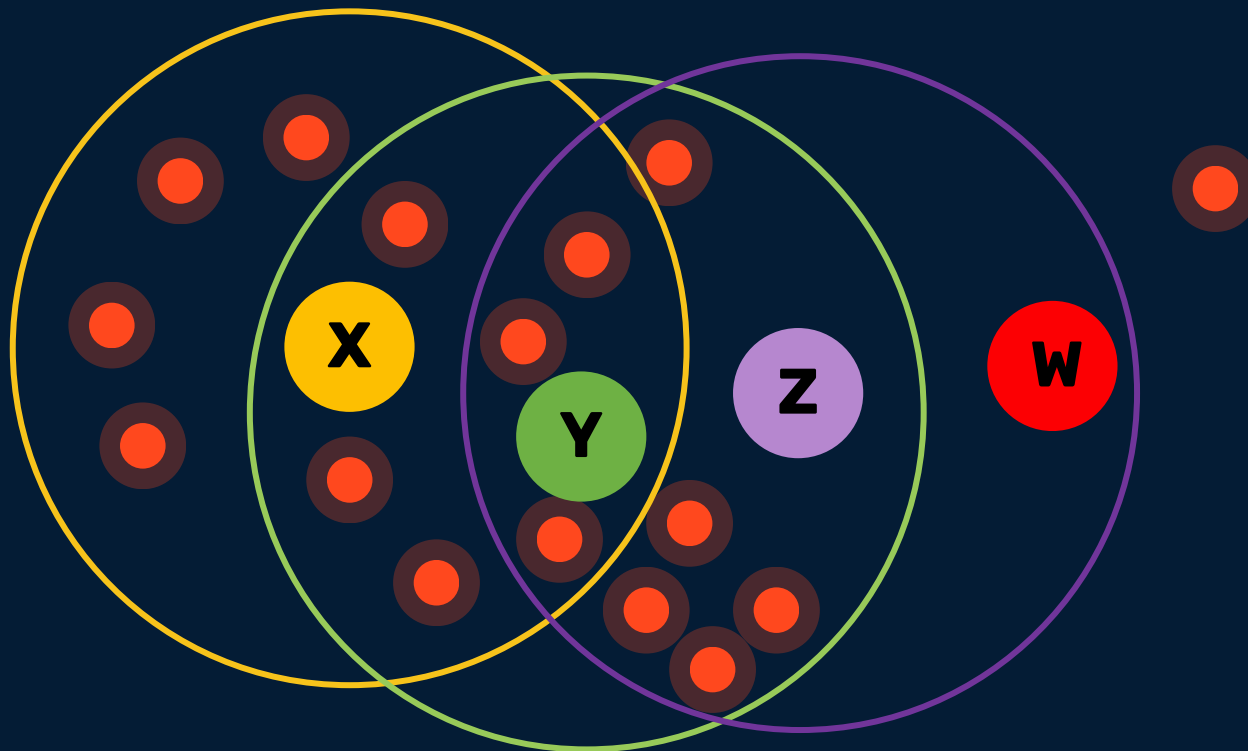
# Procedimento

Os pontos das vizinhanças densas dão origem a mais vizinhanças densas, e esses pontos são agrupados. Quando deixa de ser possível criar mais vizinhanças, um novo ponto de partida é tomado.



# Procedimento

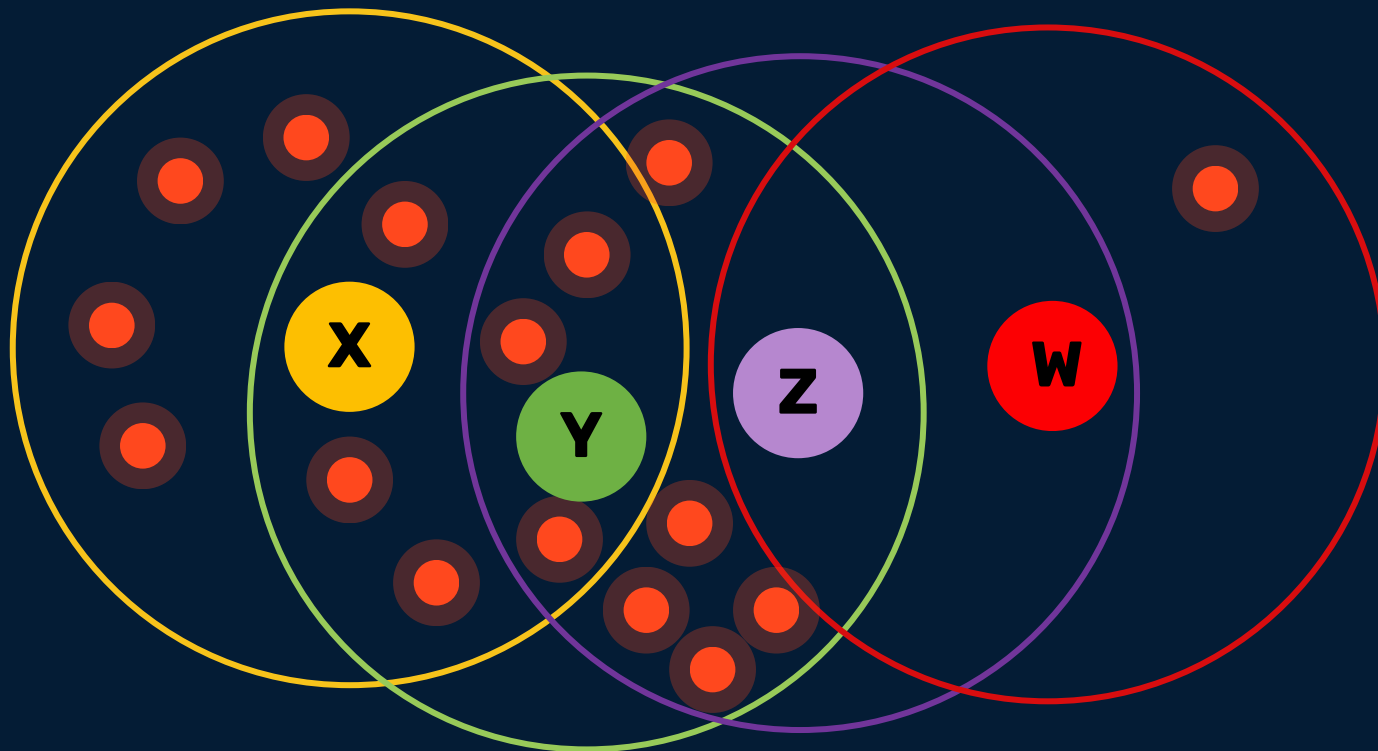
Os pontos das vizinhanças densas dão origem a mais vizinhanças densas, e esses pontos são agrupados. Quando deixa de ser possível criar mais vizinhanças, um novo ponto de partida é tomado.





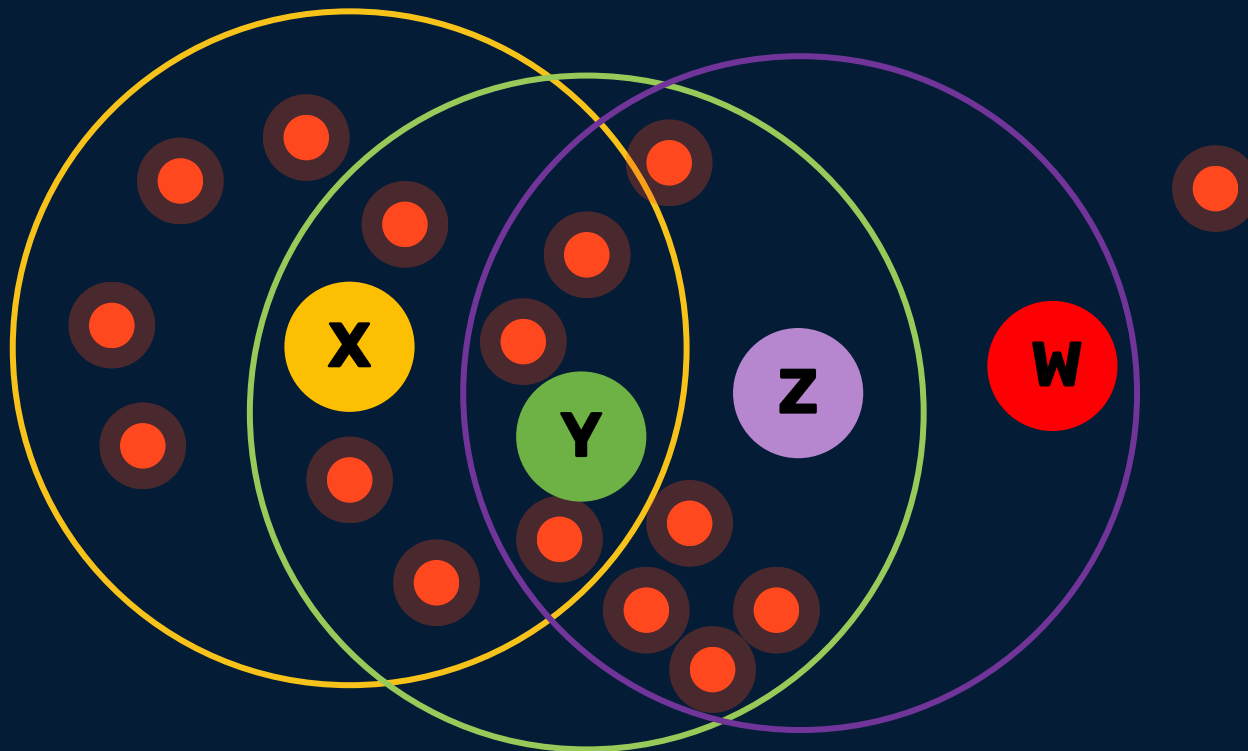
# Procedimento

Os pontos das vizinhanças densas dão origem a mais vizinhanças densas, e esses pontos são agrupados. Quando deixa de ser possível criar mais vizinhanças, um novo ponto de partida é tomado.



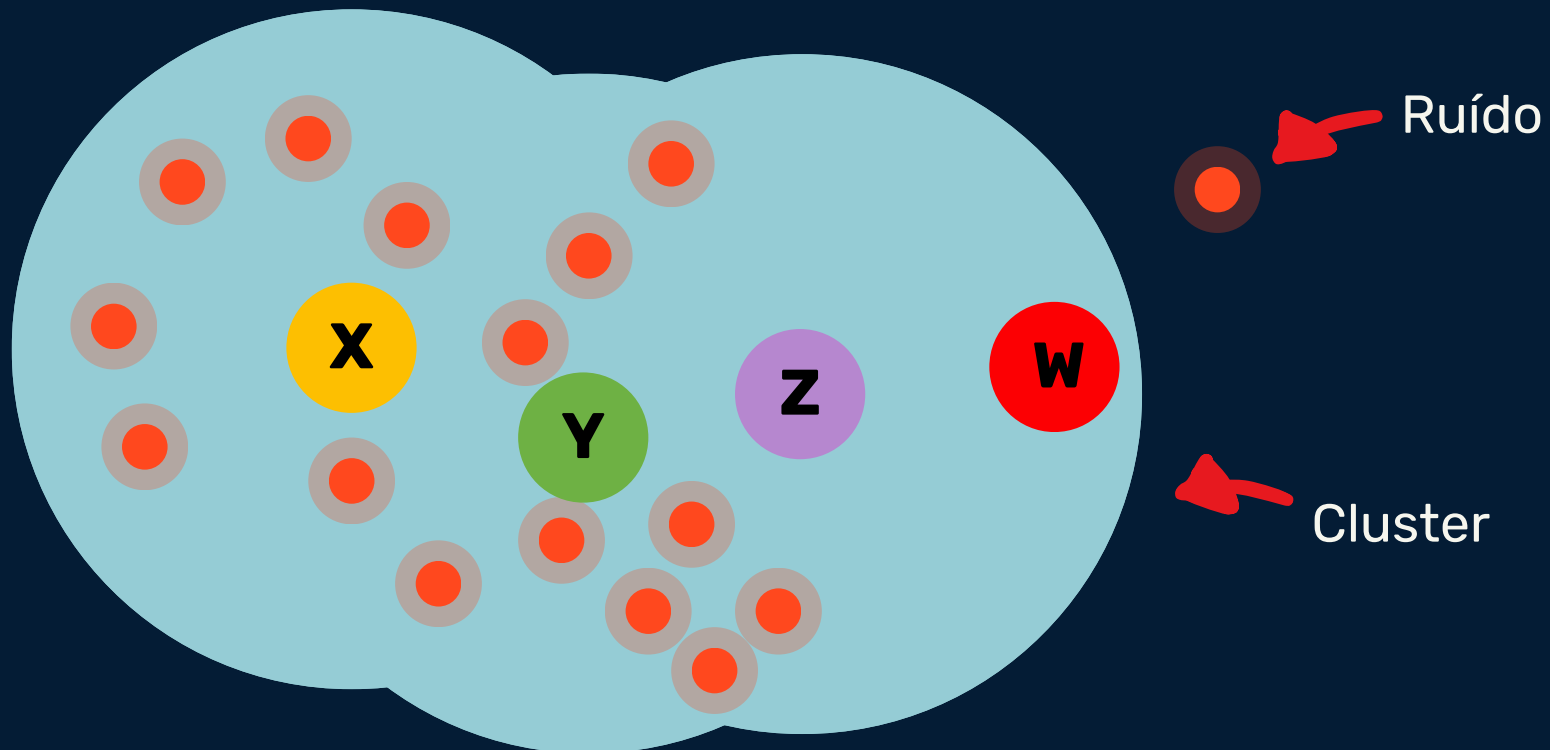
# Procedimento

Os pontos das vizinhanças densas dão origem a mais vizinhanças densas, e esses pontos são agrupados. Quando deixa de ser possível criar mais vizinhanças, um novo ponto de partida é tomado.



# Procedimento

Os pontos das vizinhanças densas dão origem a mais vizinhanças densas, e esses pontos são agrupados. Quando deixa de ser possível criar mais vizinhanças, um novo ponto de partida é tomado.



# Estrutura de dados usado

Para o conjunto de dados foi usado um Array Nx2, descrito como:

- pontos[i][0] -> Valor x do ponto i.
- pontos[i][1] -> Valor y do ponto i

```
static const float points[NUM_POINTS][NUM_FEATURES] = {  
    { 472.431845, 133.637138 },  
    // .....  
}
```

Para o ids clusters foi usado um Struct, "SetOfPoints" descrito como:

- clusterIds[i] -> Id do cluster onde o ponto "i" está.
- size -> Numero de pontos que já tem cluster.

```
typedef struct {  
    int clusterIds[MAX_POINTS];  
    int size;  
} SetOfPoints;
```

# Complexidade Temporal

Por passar por  $n$  pontos  $n$  vezes.

$$O(n^2)$$

# Complexidade Espacial

Array necessário para armazenar os clusters IDs.

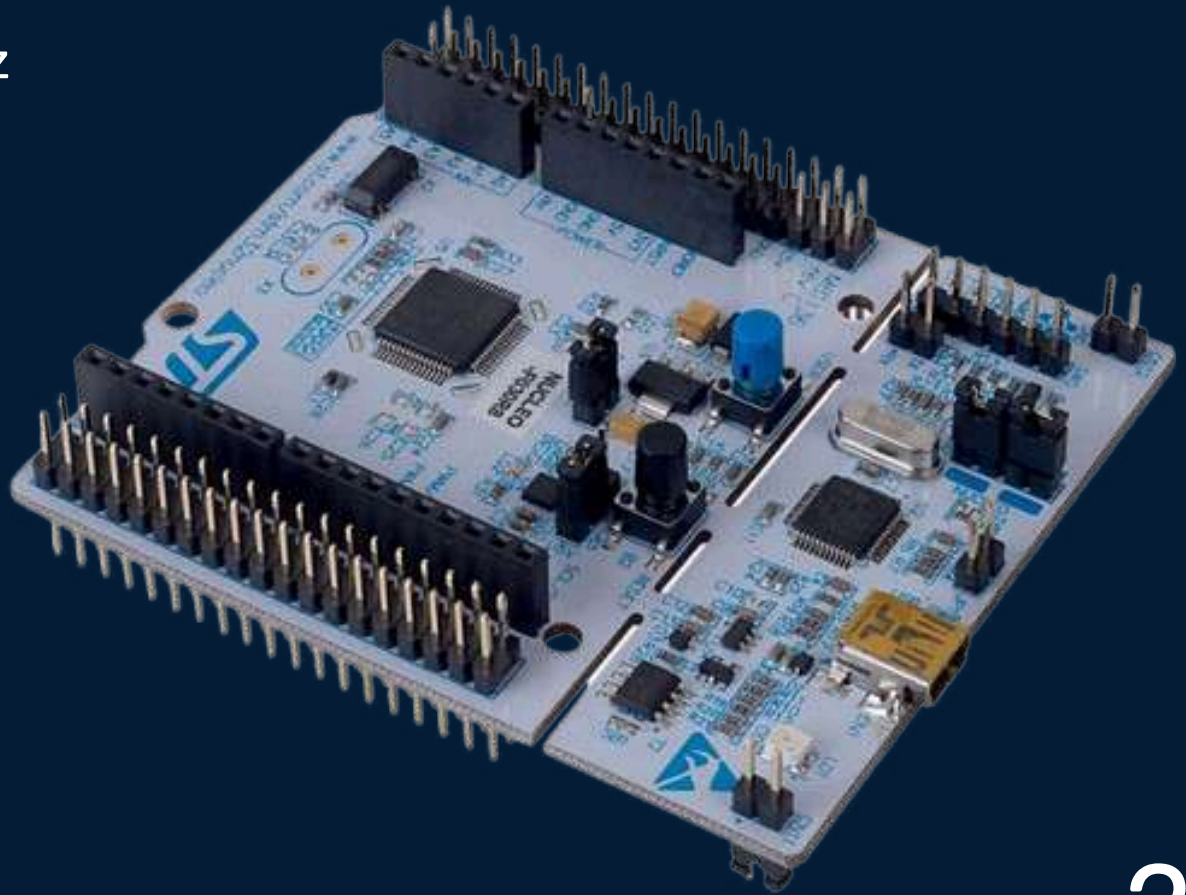
$$O(n)$$




# A Plataforma

# STM32F030R8

- **Familia:** ARM Cortex-M0
- **RAM:** 8KiB
- **Flash:** 64KiB
- **Frequência:** até 48MHz





# **As Dificuldades: Conhecimento da plataforma e do RTOS**



# Compilação do código

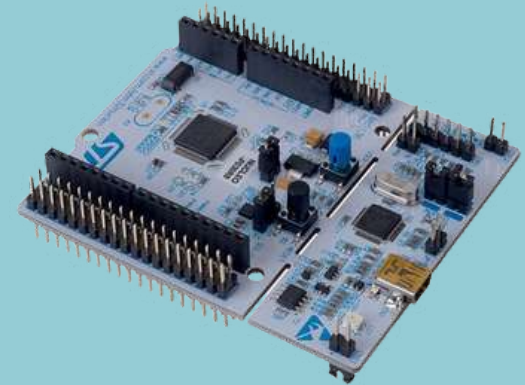
No início, não sabíamos como compilar no Keil Studio.




Depois de um tempo, o processo de compilação e execução se tornou natural.

# Saída padrão

Como ver os resultados do código usando o microcontrolador e o MBED OS?



Usando a classe Serial, instanciamos um objeto e chamamos o método printf a partir dele.



# **As Dificuldades: Gerenciamento de Memória**

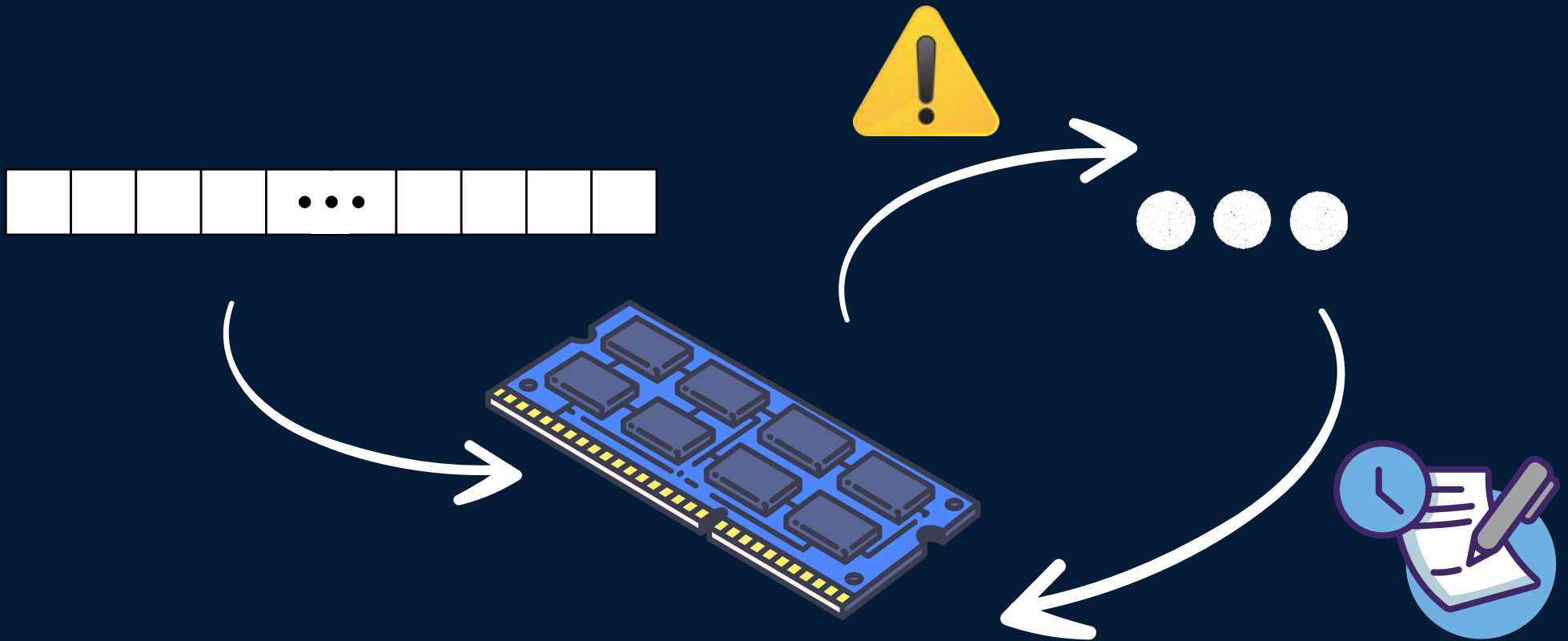
# Estouro de memória

8 KB de memória RAM não era o suficiente para comportar as estruturas de dados do nosso algoritmo.



# O que estava acontecendo?

O algoritmo parava a execução, caso fosse passado para ele mais de 100 pontos. Nesse cenário, manter os requisitos de projeto, também se tornou um desafio!



# Qual a causa?

Nossa primeira implementação do algoritmo usava estruturas de dados com tipos de dados desnecessariamente grandes.

- int = 4 bytes
  - clusterIds = 4 \* 1000 bytes

```
typedef struct {  
    int clusterIds[MAX_POINTS];  
    int size;  
} SetOfPoints;
```

# Qual a causa?

Usávamos a Struct SetOfPoints, tanto na main, quanto em dois dos métodos.

```
static int expandCluster(const float points[][NUM_FEATURES], SetOfPoints *set, int index, int clusterId, float eps, int minPts)
// Encontra todos os pontos vizinhos do ponto atual
SetOfPoints seeds = regionQuery(points, set->size, points[index], eps);
```

```
// Retorna uma estrutura SetOfPoints contendo os índices dos pontos vizinhos
static SetOfPoints regionQuery(const float points[][NUM_FEATURES], int size, const float *point, float eps) {
    SetOfPoints result;
```

```
int main() {
    SetOfPoints set;
```



Isso dá BEM MAIS que 8 KB.

# Como foi contornada?

Usamos tipagem de dados otimizada.


```
typedef int8_t cluster_id_t;  
  
typedef int16_t point_index_t;  
  
typedef float coord_t;
```

Tipo	Tamanho (bytes)	Intervalo de Valores
int8_t	1	-128 a 127
int16_t	2	-32.768 a 32.767
int	4	-2.147.483.648 a 2.147.483.647

# Como foi contornada?

Usamos tipagem de dados otimizada.

```
typedef struct {  
    int clusterIds[MAX_POINTS];  
    int size;  
} SetOfPoints;
```




```
typedef struct  
{  
    cluster_id_t clusterIds[NUM_POINTS];  
    point_index_t size;  
} SetOfPoints;
```




# Como foi contornada?

Removemos a instância extra do SetOfPoints nos dois métodos, e passamos a chamá-lo em apenas um e com os dados otimizados.

```
static SetOfPoints regionQuery  
    SetOfPoints result;
```



```
static int expandCluster  
    // Encontra todos os  
    SetOfPoints seeds = I
```



```
point_index_t neighbors[MAX_NEIGHBORS];  
point_index_t seeds[MAX_NEIGHBORS];
```

# Resultado

Otimização do uso da memória feita, e código rodando com sucesso!

```
elf2bin DBSCAN-MBED-OS.NUCLEO_F030R8  
Build succeeded
```

```
Flash: 33 KB / 64 KB
```

```
Memory: 4 KB / 8 KB
```

```
ROM: 33 KB
```





# Os Testes

# Escolha do conjunto de dados

Para teste, foi escolhido o conjunto de dados de sequência de RNA (Hi-Seq) PANCAN, que contém características genéticas de pacientes com tumores. O mesmo usado anteriormente.

	P1	P2
0	-0.056920	0.082045
1	-0.001607	-0.082631
2	-0.066280	0.010366
3	-0.077986	0.066873
4	-0.063931	0.011263
...	...	...
999	-0.056405	0.024994
1000	-0.012920	-0.050042
1001	0.008534	-0.024661
1002	-0.050052	0.080348
1003	-0.044771	0.043976

1004 rows x 2 columns



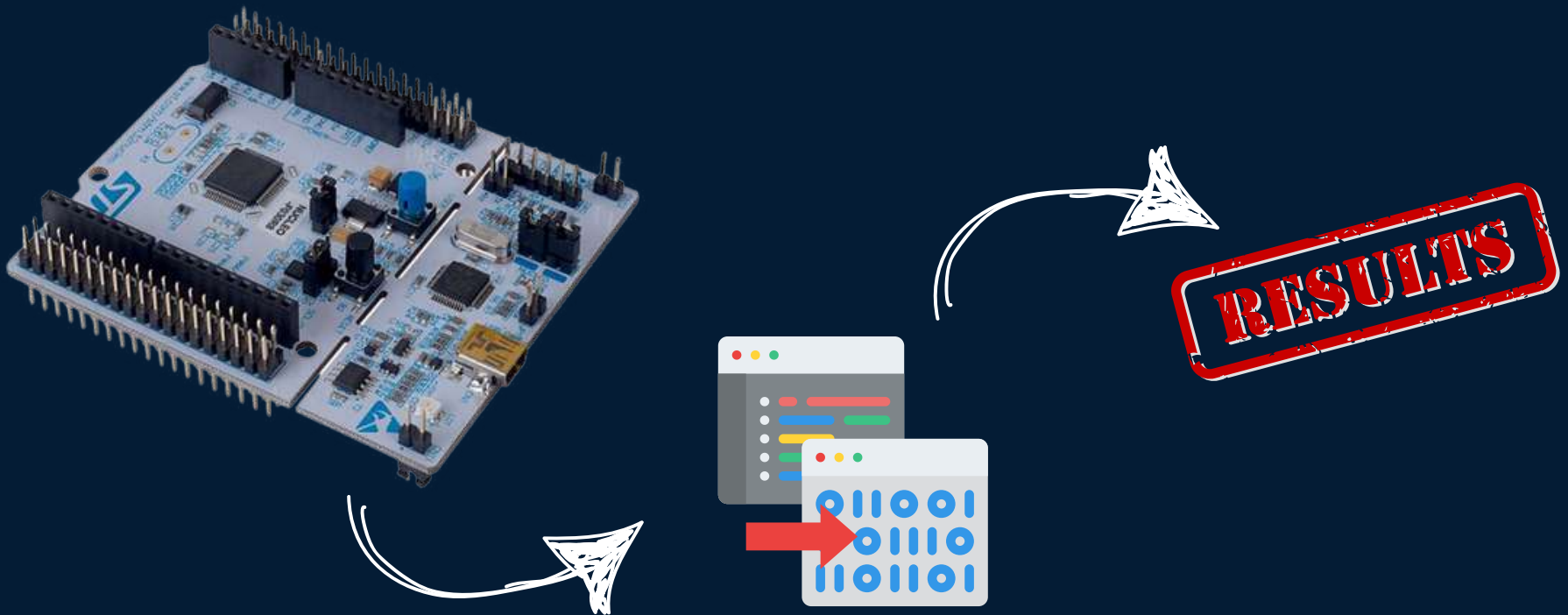
# Como fizemos para testar?

Fizemos os testes necessários no computador, usando nossa implementação antes de levar para a plataforma embarcada, a única diferença foi que no computador não usamos MBED OS.



# Como fizemos para testar?

Também testamos no STM32 em sala de aula. Copiamos os valores da saída padrão.



# Saída do código no STM32

Algumas das saídas são descritas abaixo:

```
Using TUMOR dataset with 1004 points
Parameters: eps=0.008, minPts=10
Starting clustering...
DBSCAN: Starting with 1004 points
Processing point 0 of 1004
Processing point 100 of 1004
Processing point 200 of 1004
Processing point 300 of 1004
Processing point 400 of 1004
Processing point 500 of 1004
Processing point 600 of 1004
Processing point 700 of 1004
Processing point 800 of 1004
Processing point 900 of 1004
Processing point 1000 of 1004
```

```
START_CLUSTERING_DATA
```

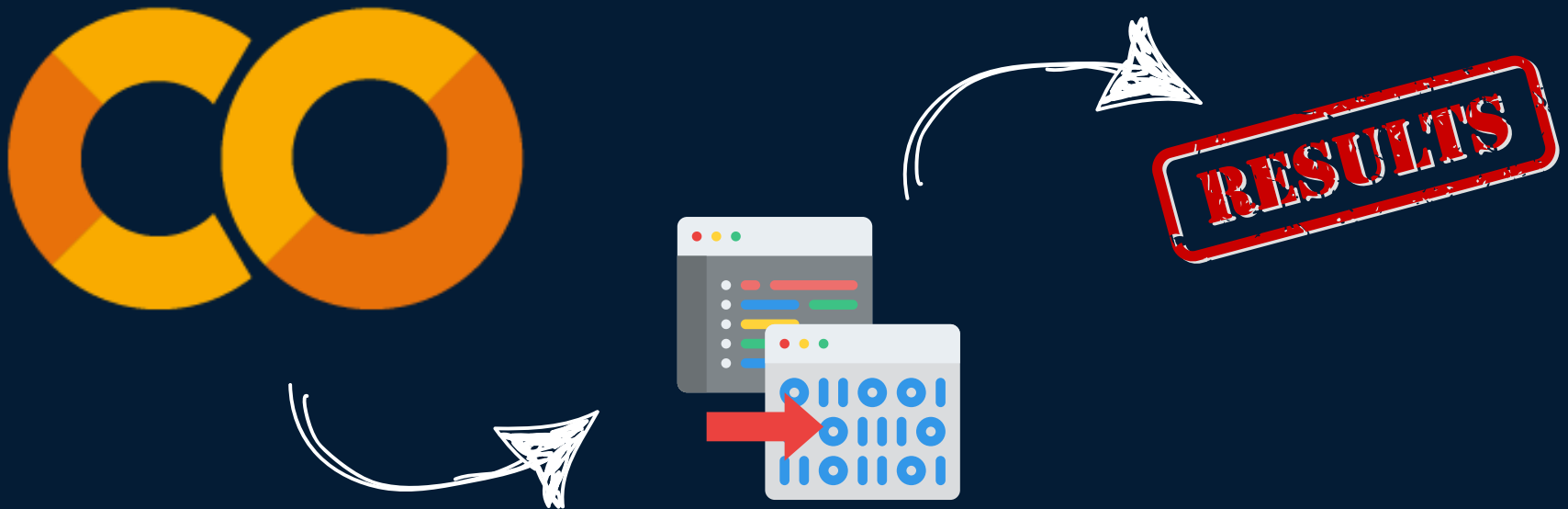
```
1
4
2
0
2
1
0
0
```

```
4
2
4
4
1
2
```

```
END_CLUSTERING_DATA
```

# Como fizemos para testar?

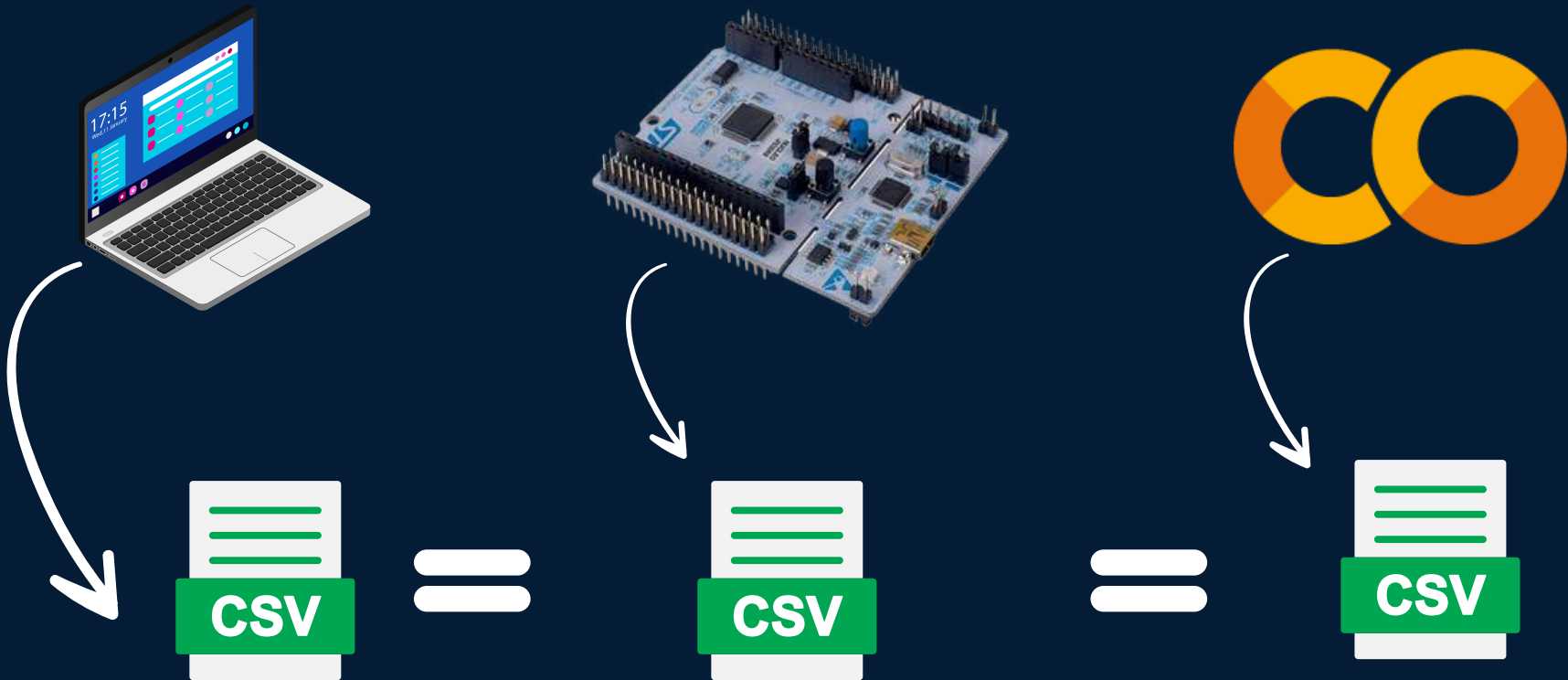
Testamos em ambiente cloud, usando a linguagem python e a biblioteca scikitlearn.





# Validação dos resultados

Por fim, comparamos os 3 resultados, através de um csv criado para cada um, e usando a plataforma google Colab.



# Comparação numérica dos 3 resultados

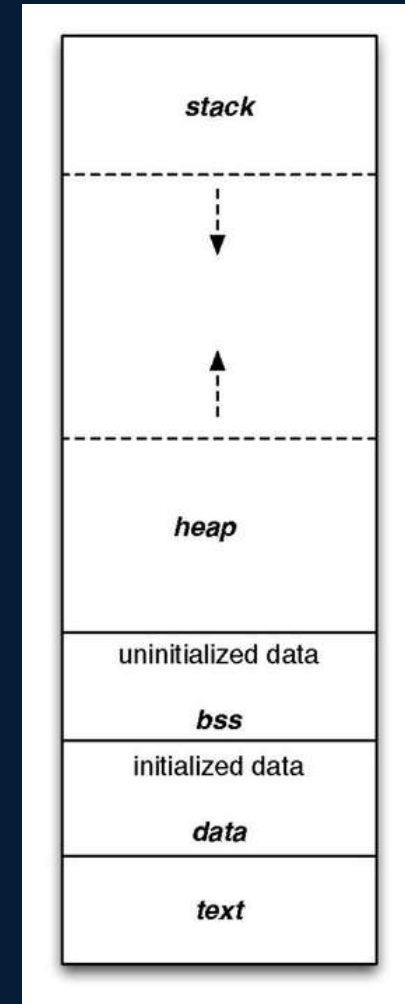
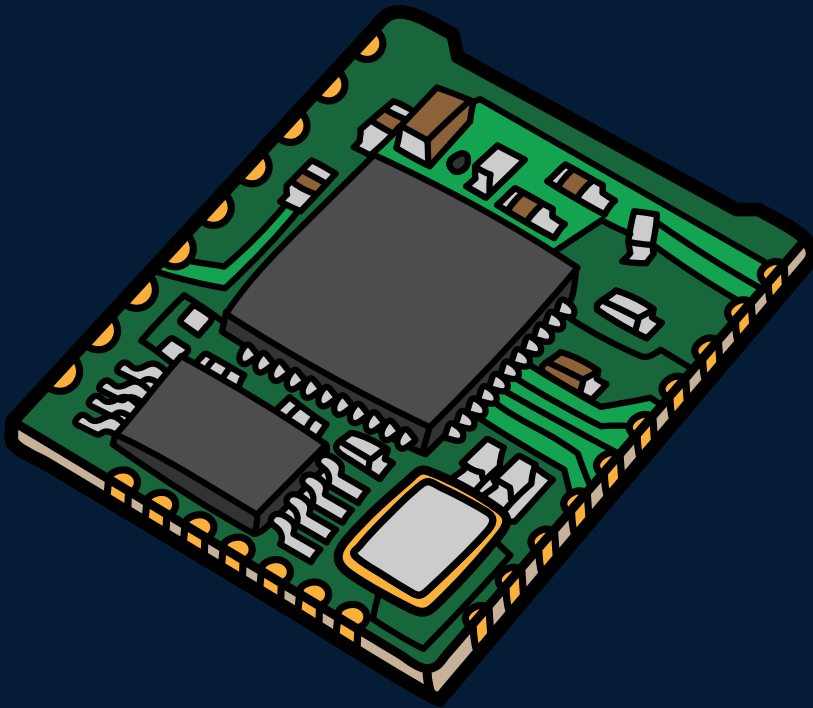
	python	stm32	desktop	Iguais
0	1	1	1	True
1	4	4	4	True
2	2	2	2	True
3	0	0	0	True
4	2	2	2	True
...	...	...	...	...
999	2	2	2	True
1000	4	4	4	True
1001	4	4	4	True
1002	1	1	1	True
1003	2	2	2	True

[1004 rows x 4 columns]

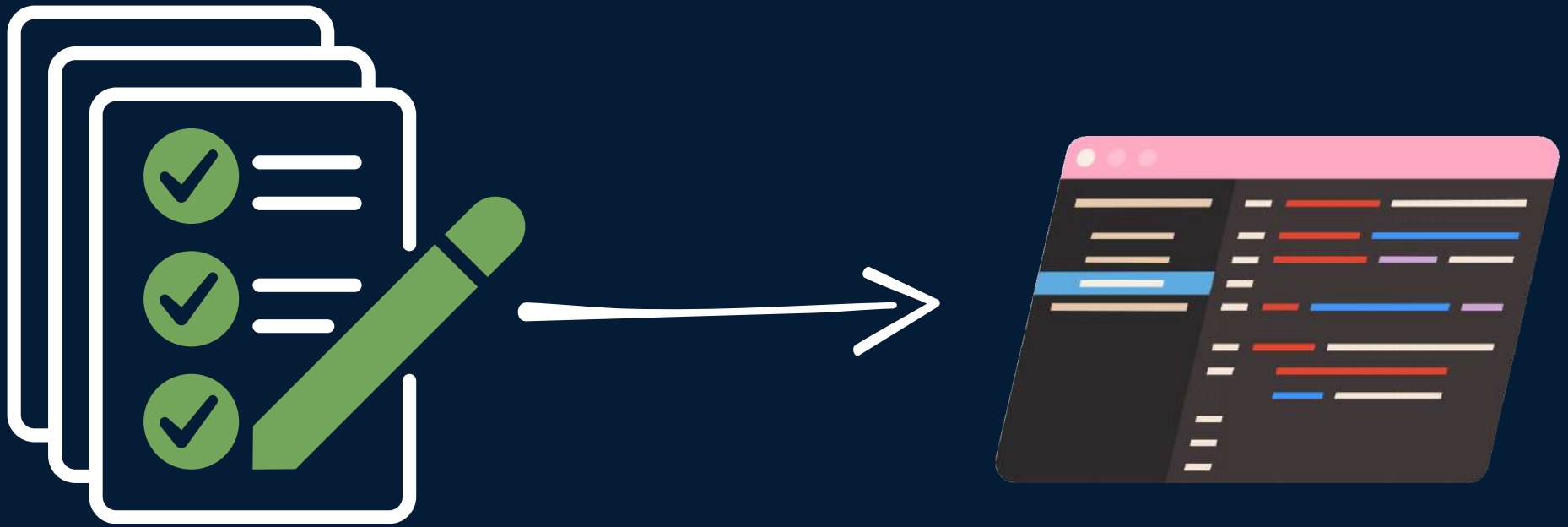


# Considerações

# Limitações do embarcado



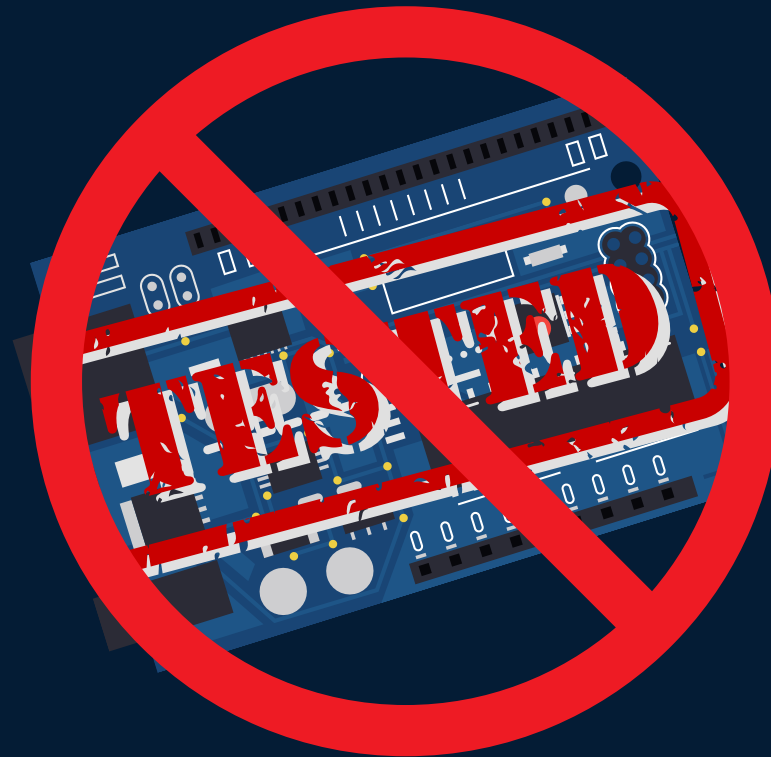
# Os requisitos importam MUITO



# Testes de software embarcado

## NÃO SE FAZ TESTE DE SOFTWARE NO EMBARCADO

Não tem problema em copiar o resultado e testar em outro lugar, pra ver se o valor bate.



# Bibliografia

AARM Keil Studio:

- <https://studio.keil.arm.com/>

Mbed OS

- <https://os.mbed.com/docs/mbed-os/v6.16/introduction/index.html>
- <https://os.mbed.com/docs/mbed-os/v6.16/reference/index.html>
- <https://os.mbed.com/docs/mbed-os/v6.16/quick-start/index.html>
- <https://os.mbed.com/docs/mbed-os/v6.16/apis/index.html>
- <https://os.mbed.com/handbook/C-Data-Types>

STM32F030R8

- <https://www.st.com/en/microcontrollers-microprocessors/stm32f030r8.html>

Nossa Implementação anterior

- <https://github.com/pauloDiego-sudo/DBSCAN-SEMB.git>

Outros

- <https://pt.stackoverflow.com/questions/575039/onde-os-dados-s%C3%A3o-colocados-na-mem%C3%B3ria-em-c>



**PERGUNTAS?**





**OBRIGADO!**