

# ANNOY-GPU: Approximate Nearest Neighbor Oh Yeah – GPU

Final Project Presentation

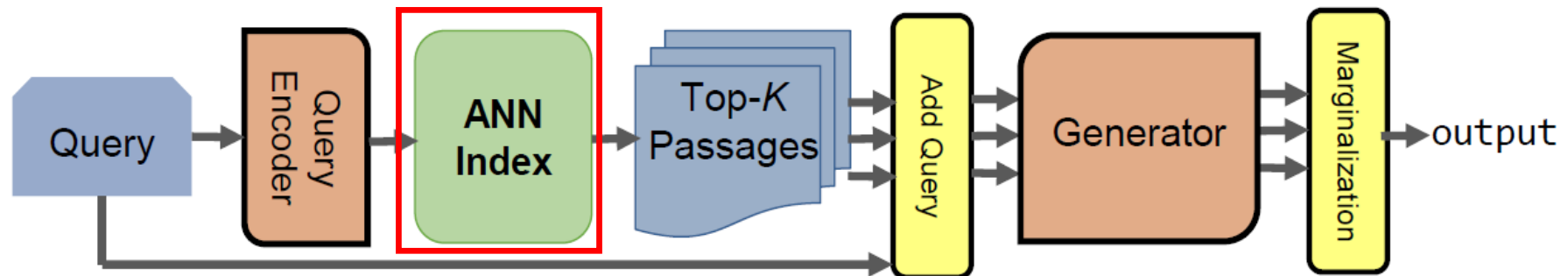
吳永玓

- **Goal**
- **Motivation**
- **ANN semantic retrieval steps**
- **Build Index Tree (CPU) – Illustration**
- **Build Index Tree (GPU) – Illustration**
- **Performance**
- **Demo**

- **Use GPU to speed up the Approximate Nearest Neighbor (ANN) algorithm.**

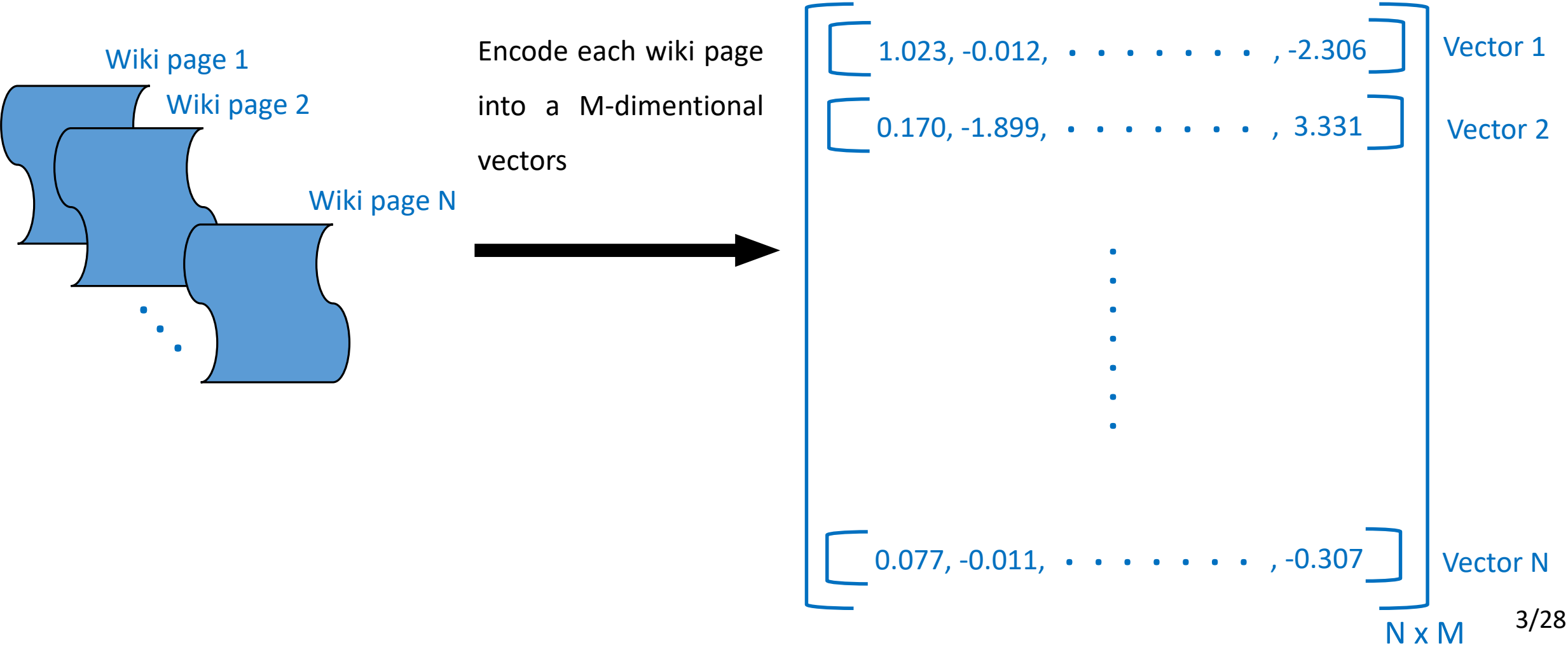
- In knowledge intensive NLP tasks (e.g. GPT-3), ANN is used to aid in semantic retrieval.

ANN is part of the processing pipeline of an knowledge intensive NLP model



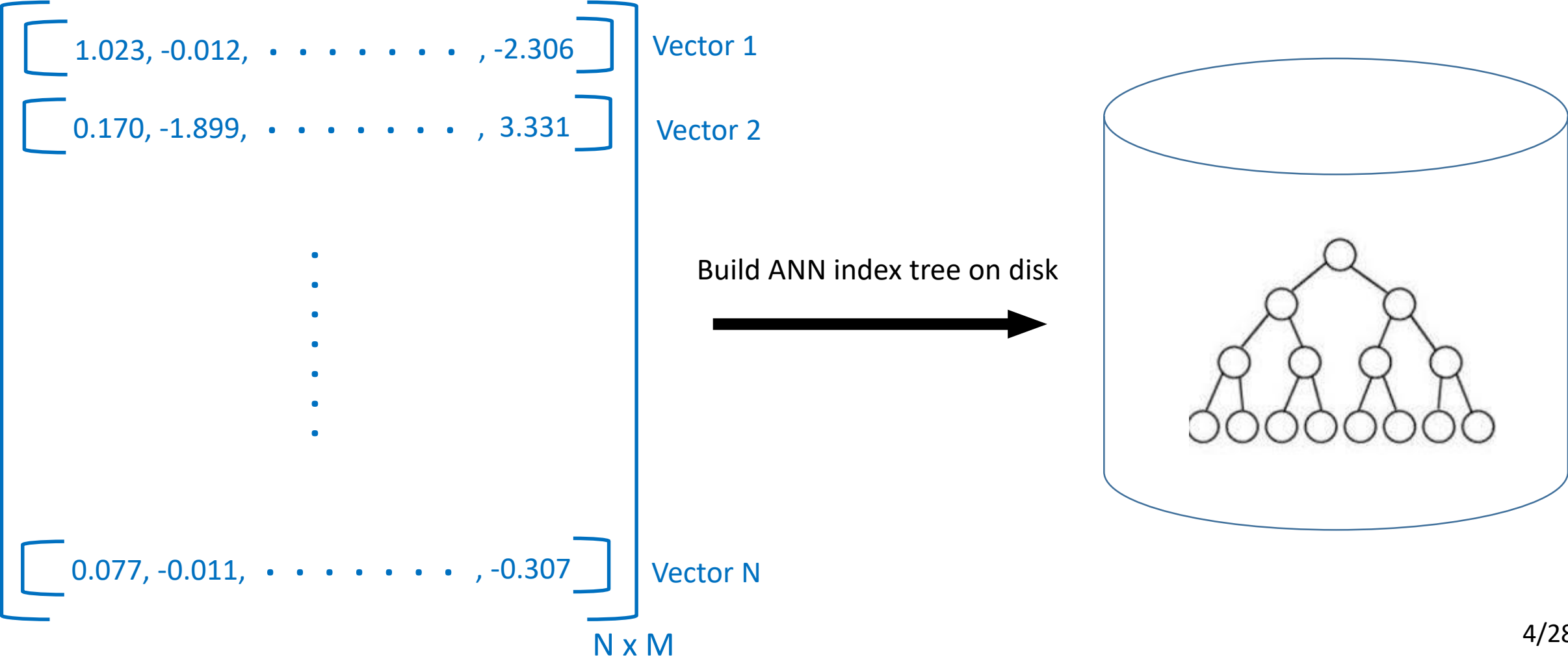
# ANN semantic retrival steps

## Step 1



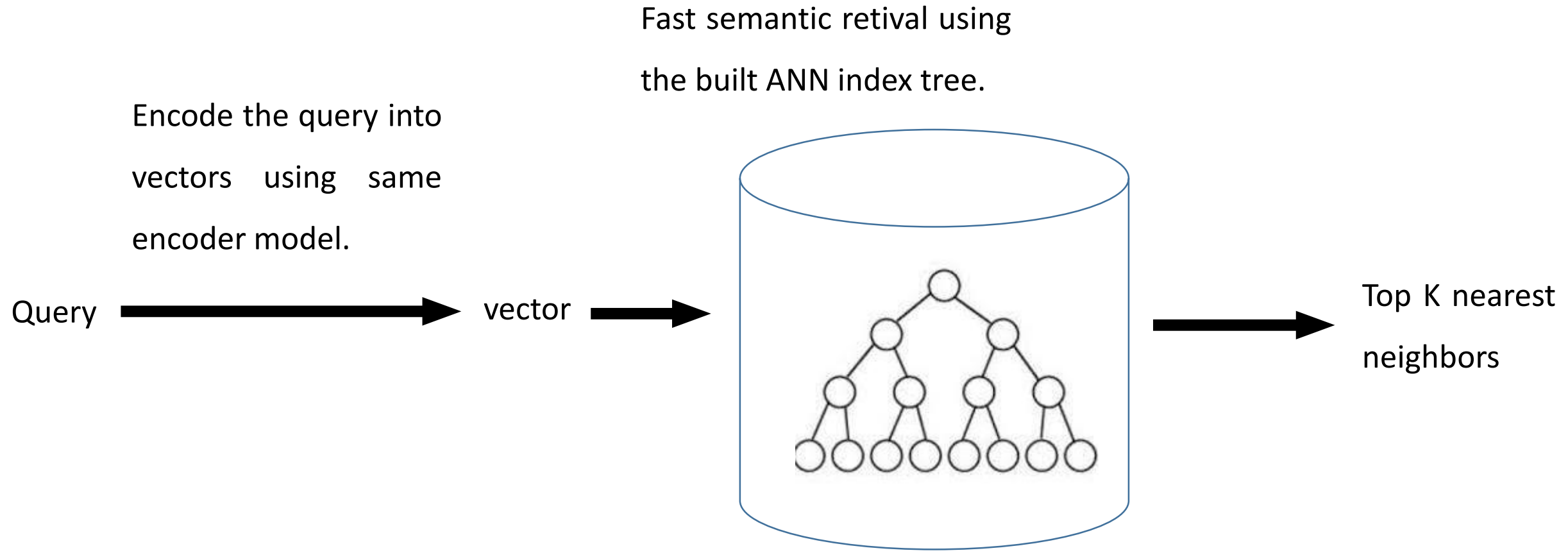
# ANN semantic retrival steps

## Step 2



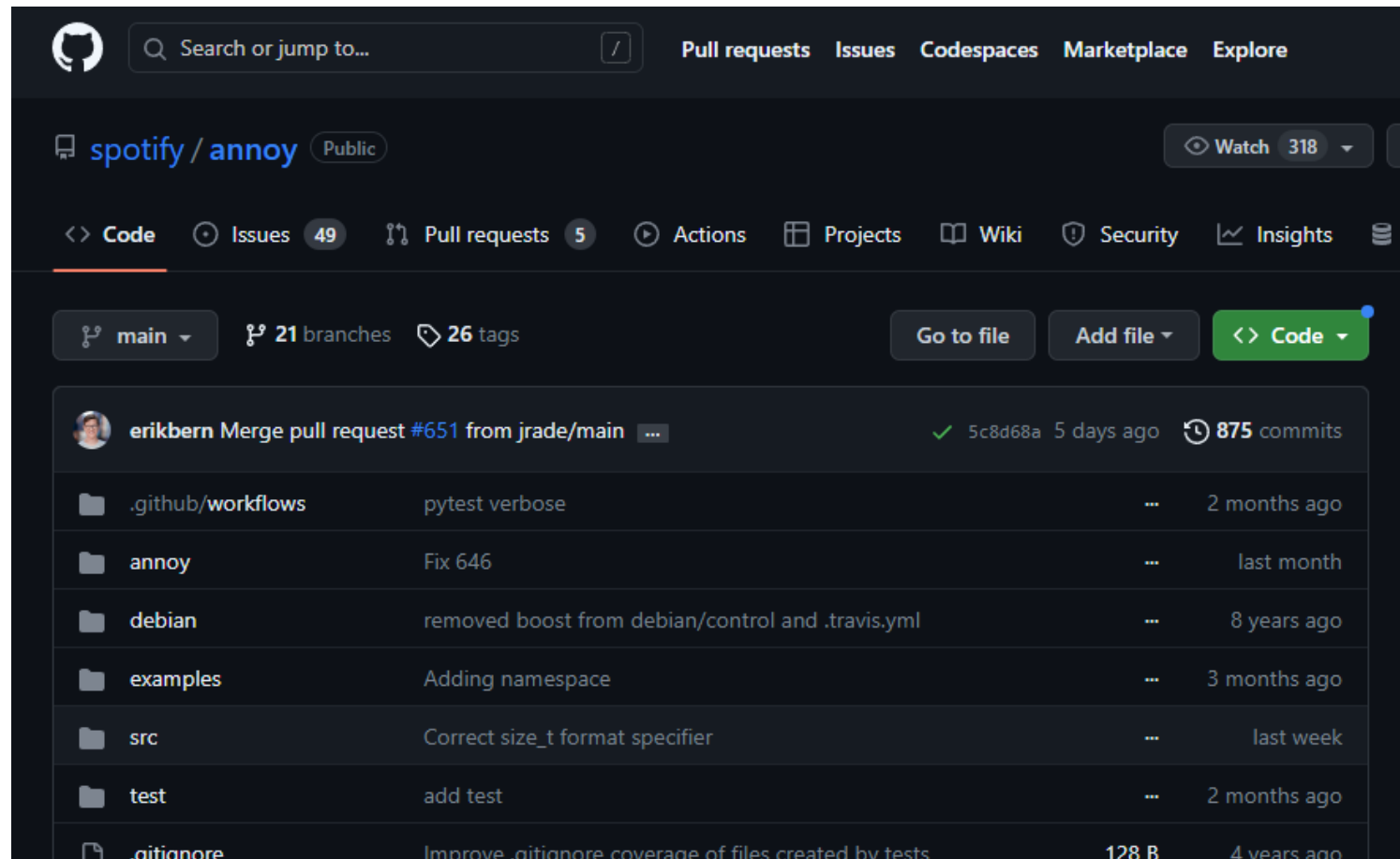
# ANN semantic retrieval steps

## Step 3



# ANNOY – Approximation Nearest Neighbor Oh Yeah

- Start from an existing project: ANNOY, which is built by an employee from the company Spotify.
- ANNOY can only use multi-thread (CPU) to speed up build process.
- I will use GPU to speed up build process.



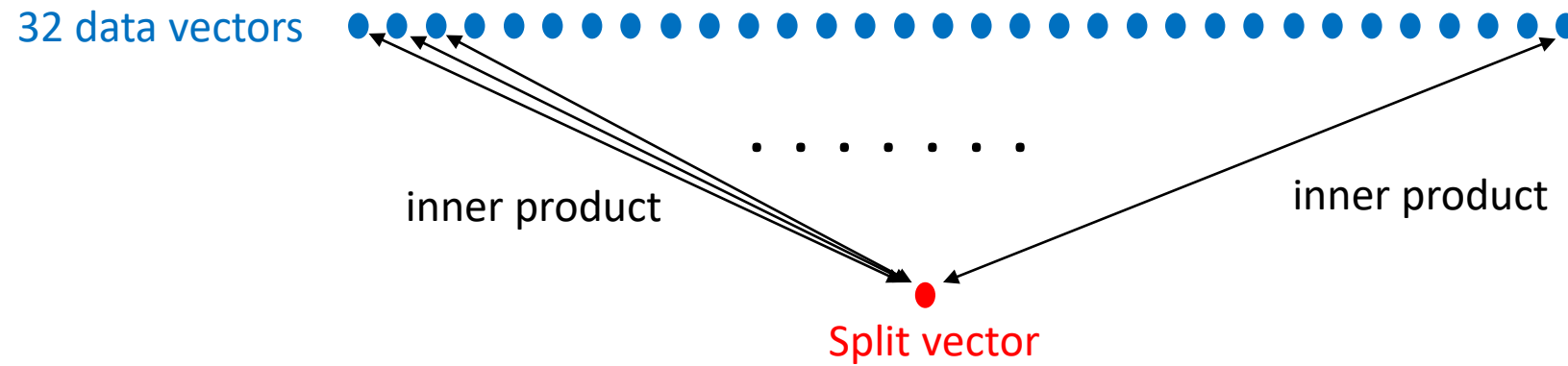


## Build Index Tree (CPU) - Illustration

32 data vectors

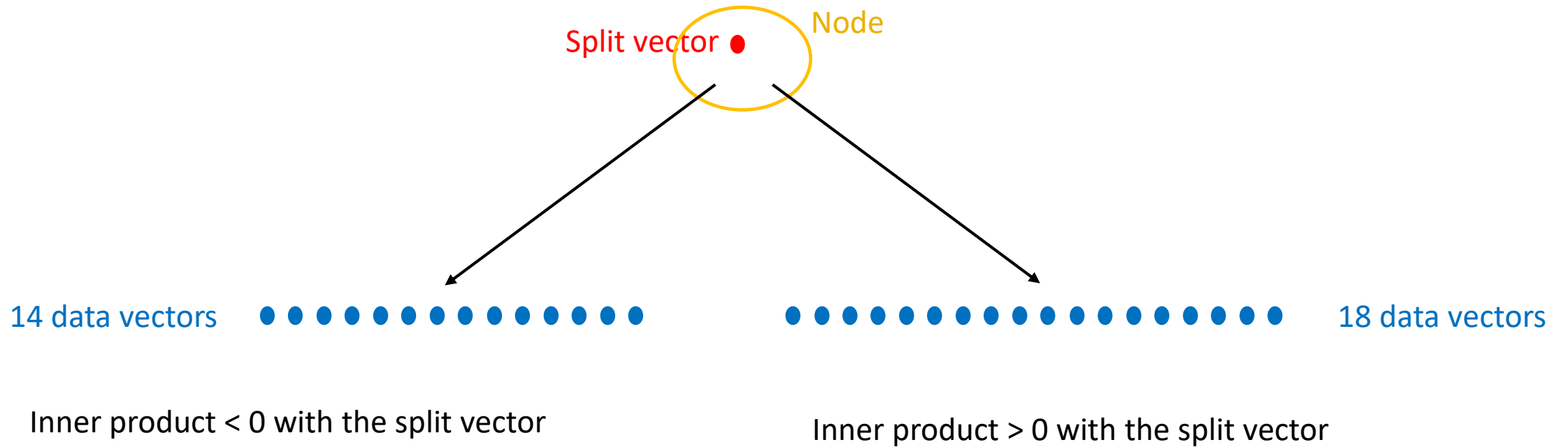
- We want to build index tree for a dataset of 32 vectors.

# Build Index Tree (CPU) - Illustration



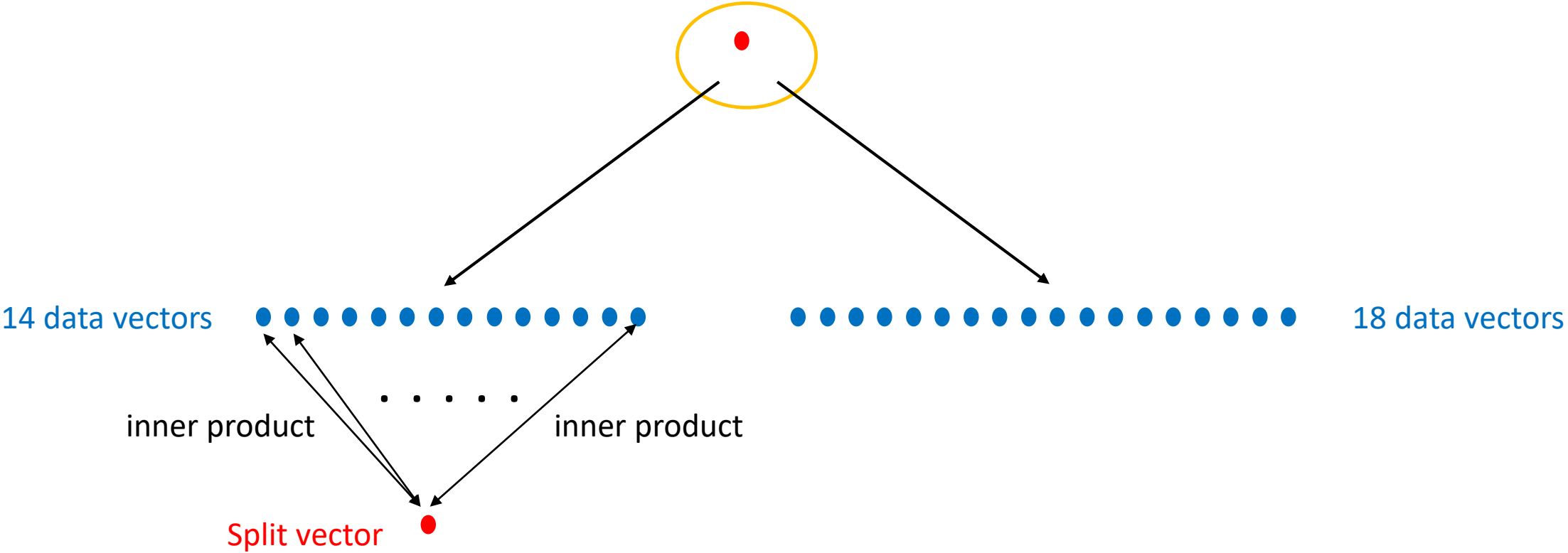
- Find a split vector that evenly divides the group of vectors (by dot product value  $>$  or  $<$  0).

# Build Index Tree (CPU) - Illustration

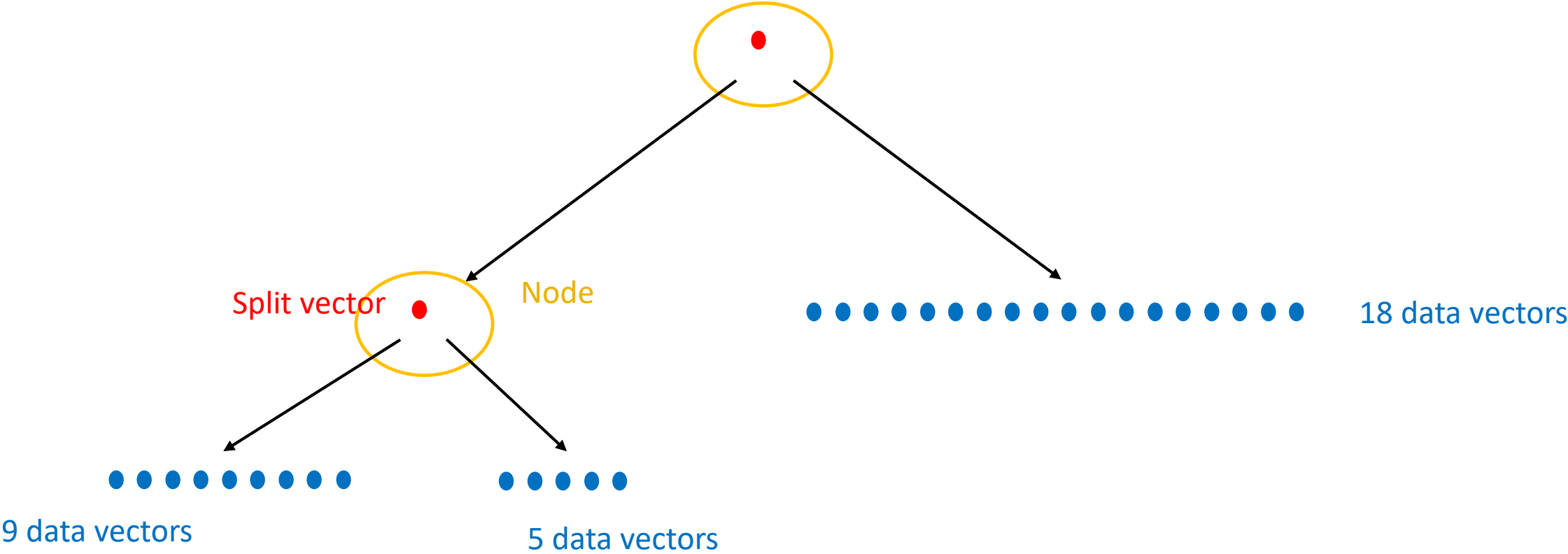


- Split the group of 32 data vector into 2 groups based on whether their inner product value with the split vector is larger than or smaller than 0.
- In case inner product is 0 → random.
- Compute imbalance: if  $\max(\frac{n_{left}}{n_{left}+n_{right}}, \frac{n_{right}}{n_{left}+n_{right}}) < \text{threshold} \rightarrow \text{balanced}$ .
- If not balanced → re-try.

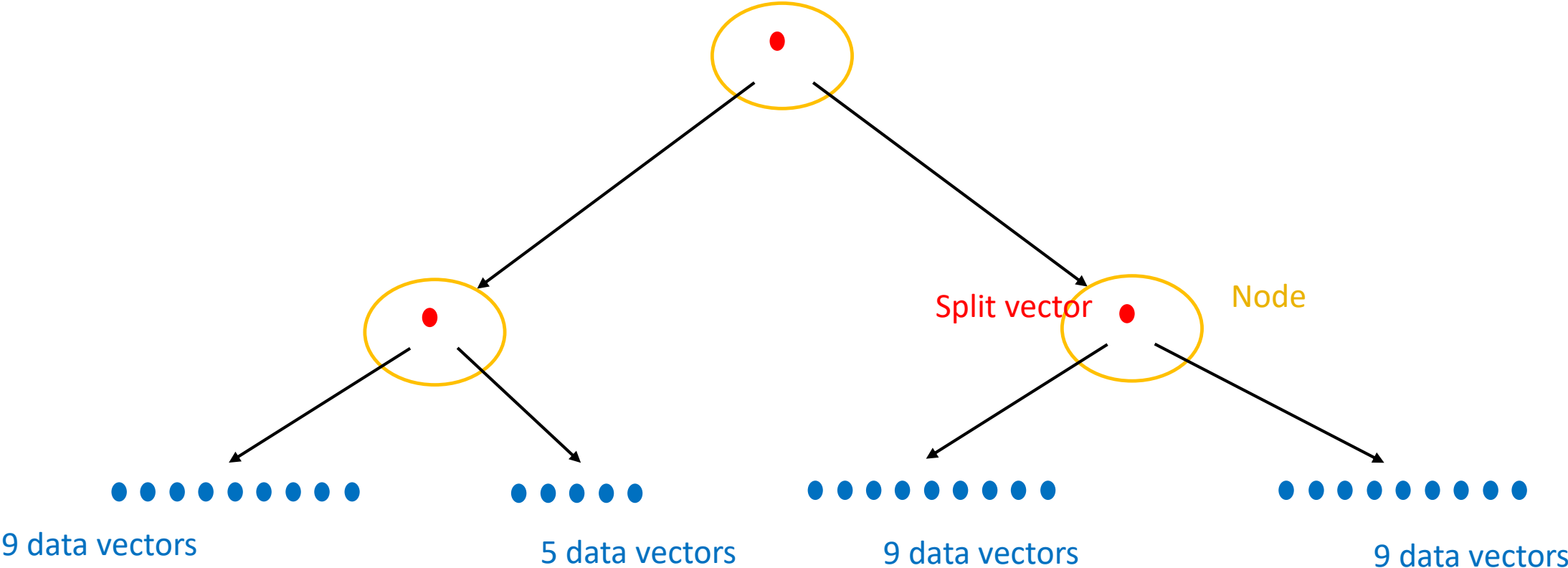
# Build Index Tree (CPU) - Illustration



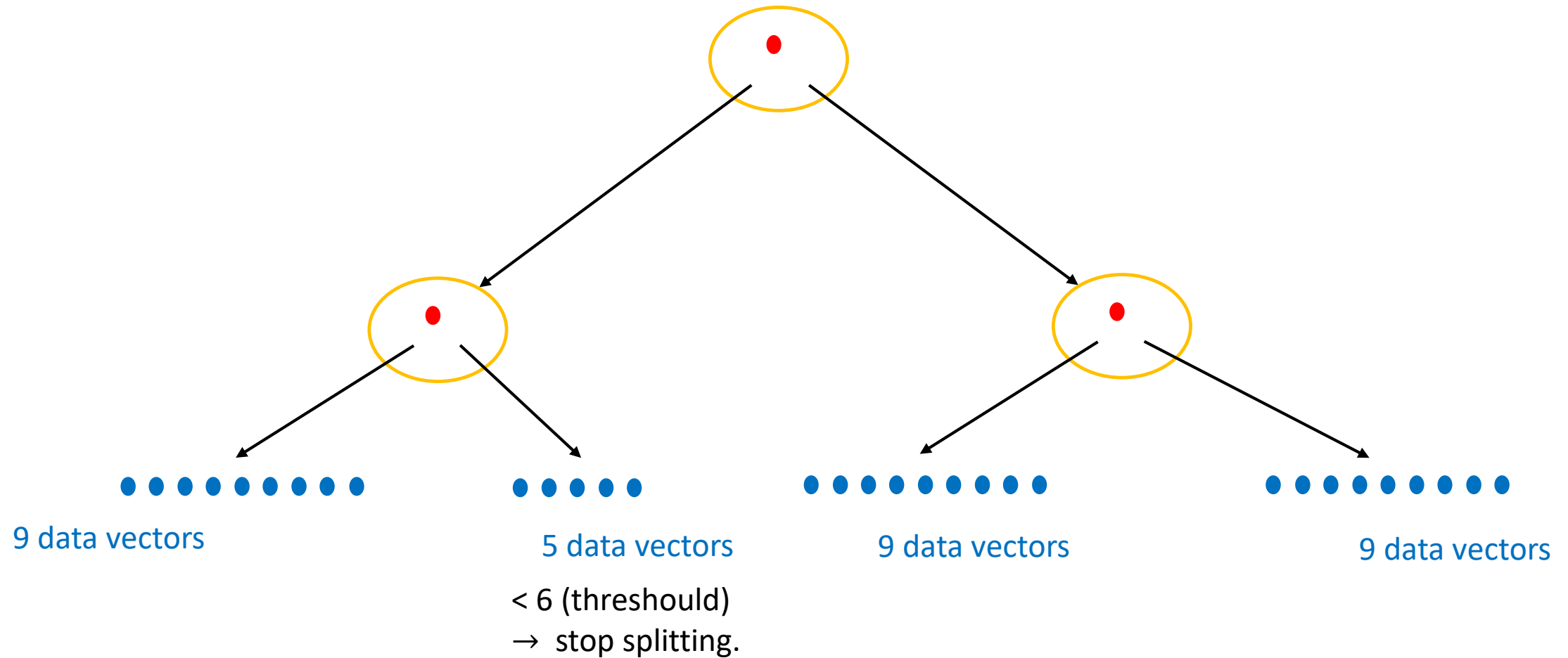
# Build Index Tree (CPU) - Illustration



# Build Index Tree (CPU) - Illustration

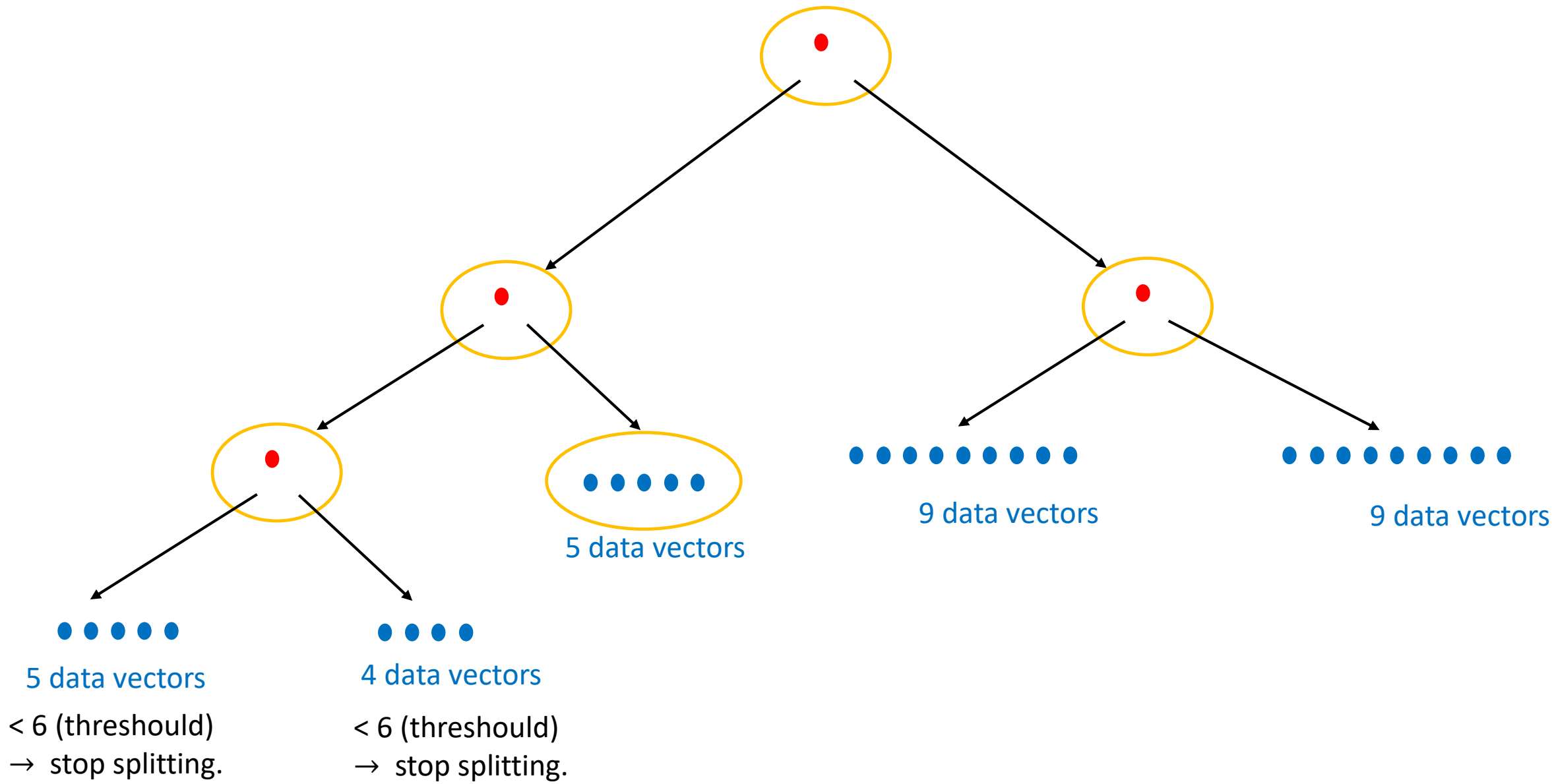


# Build Index Tree (CPU) - Illustration



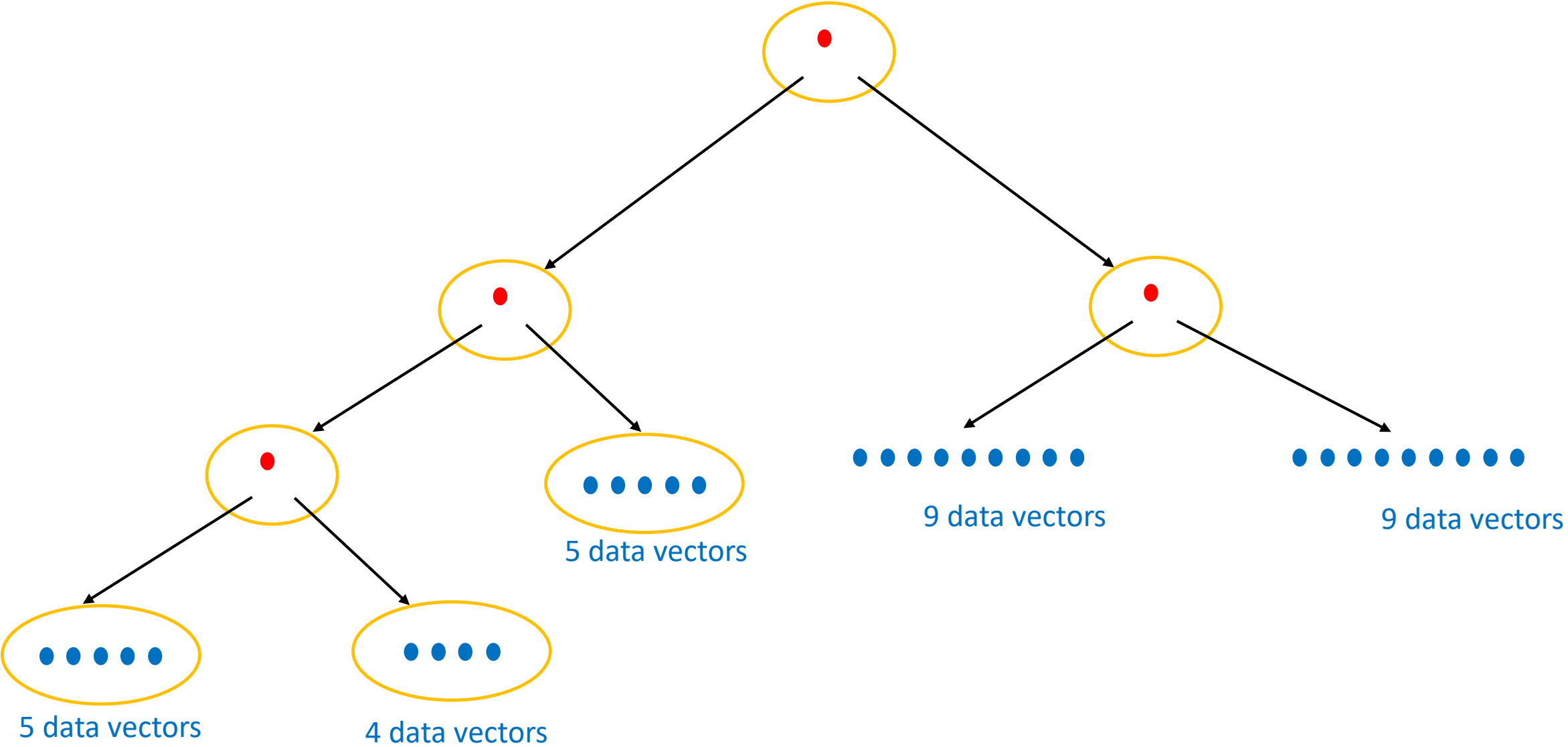
- A group will stop splitting when its data vector number  $<$  a threshold. Assume threshold is 6.
- Will create a Node to hold the group of vectors that can no longer be splitted.

# Build Index Tree (CPU) - Illustration



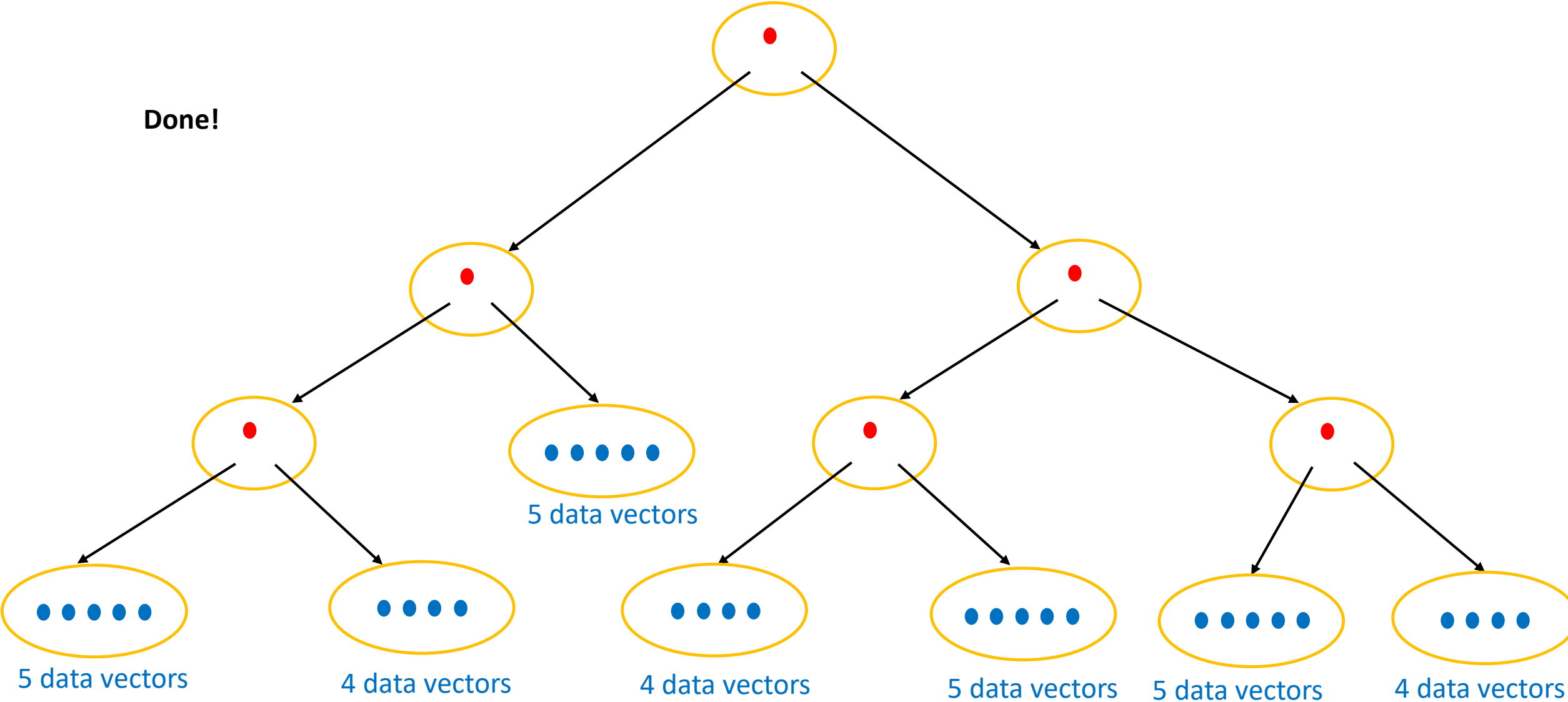


# Build Index Tree (CPU) - Illustration

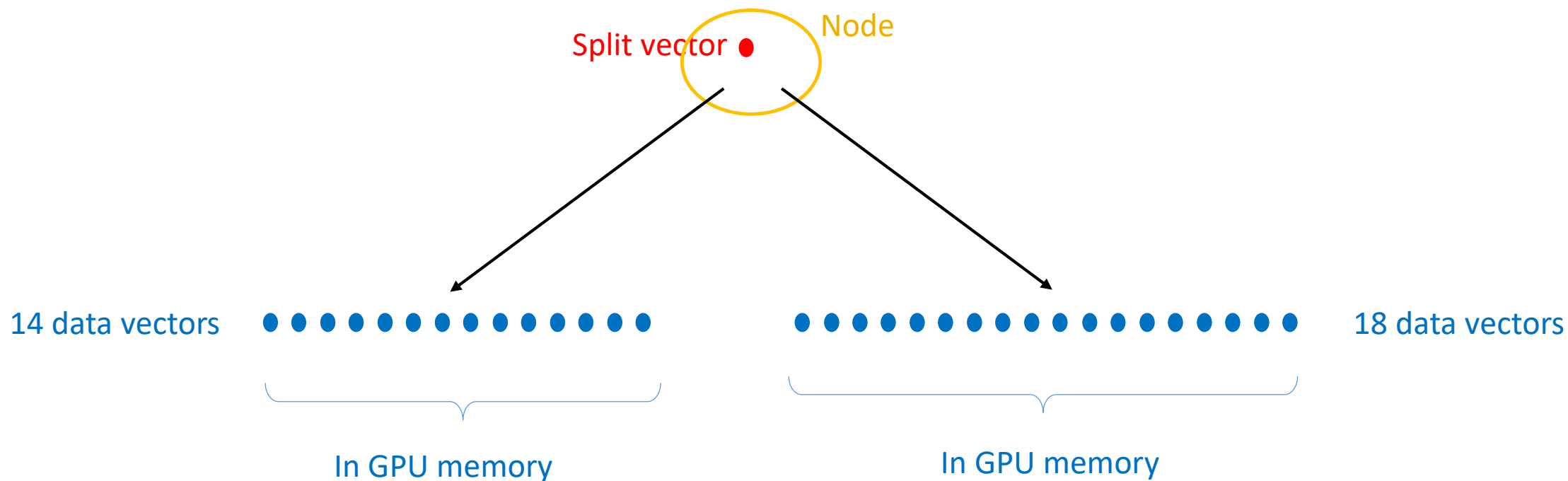


# Build Index Tree (CPU) - Illustration

Done!

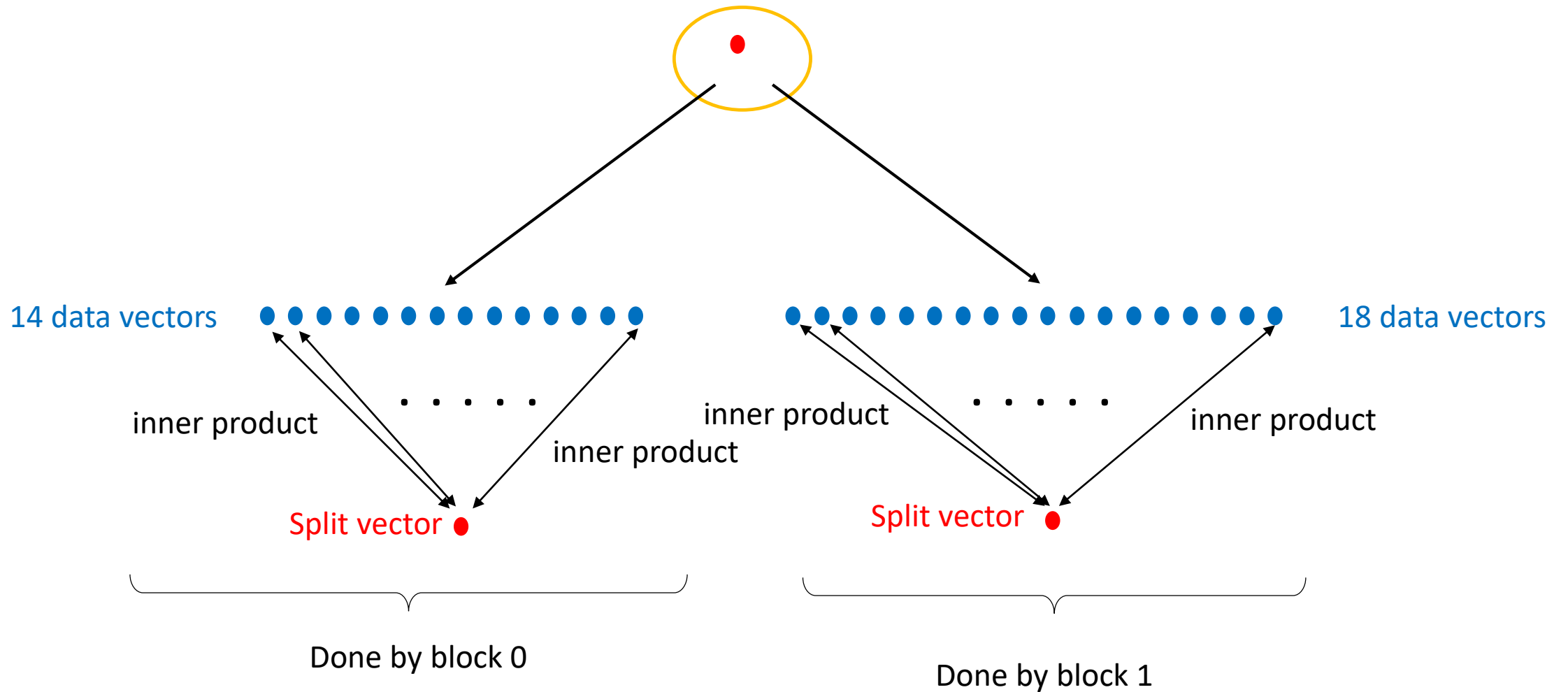


# Build Index Tree (GPU) - Illustration



- Data vectors are loaded to GPU before building process begins.
- Assume all vectors can fit into GPU memory for now.
- Tree is built on host disk not on GPU memory.
- Computed split vector by GPU will need to be sent back to host in each iteration to update the tree.

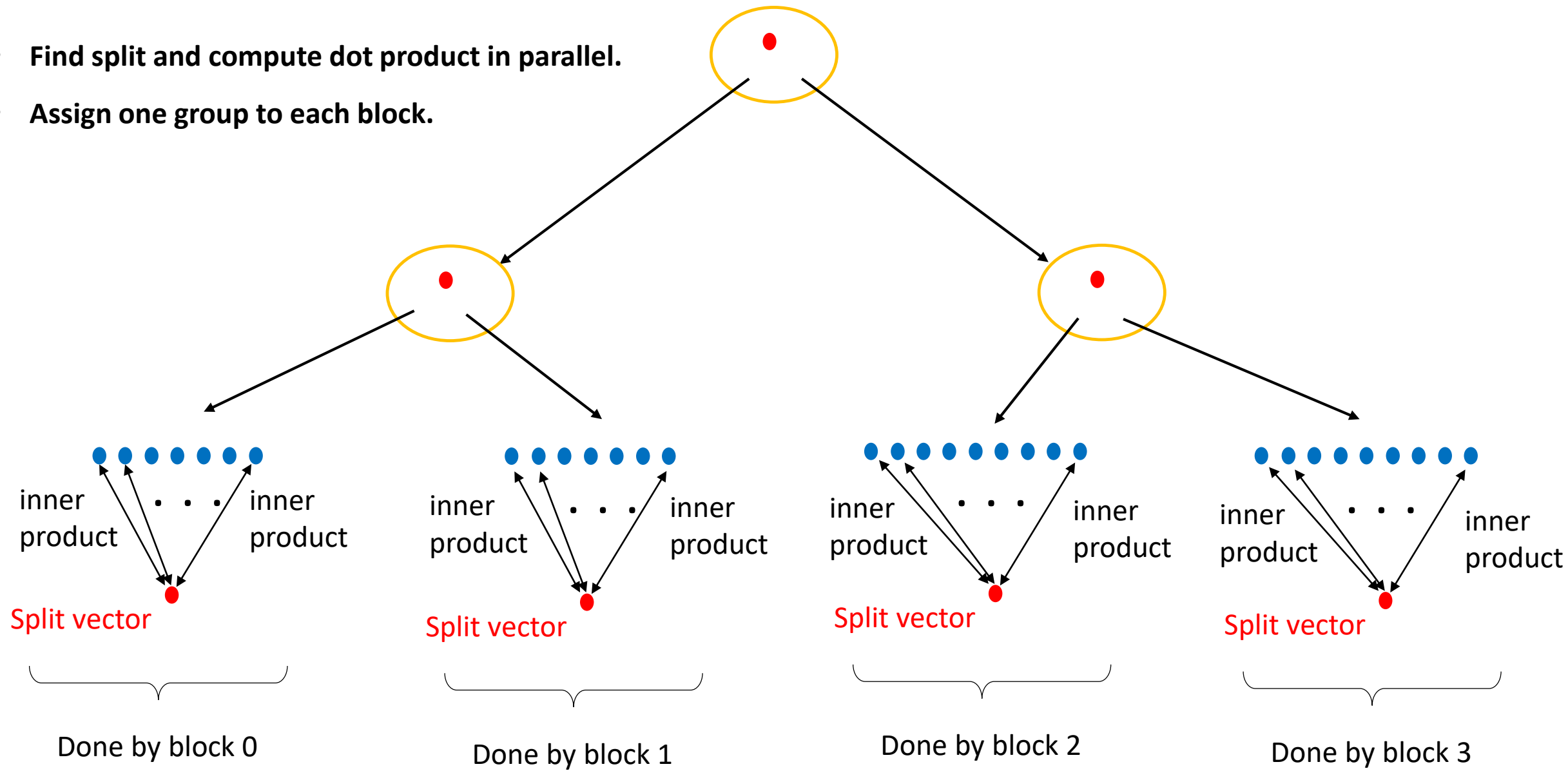
# Build Index Tree (GPU) - Illustration



- Find split and compute dot product in parallel.
- Assign one group to each block.

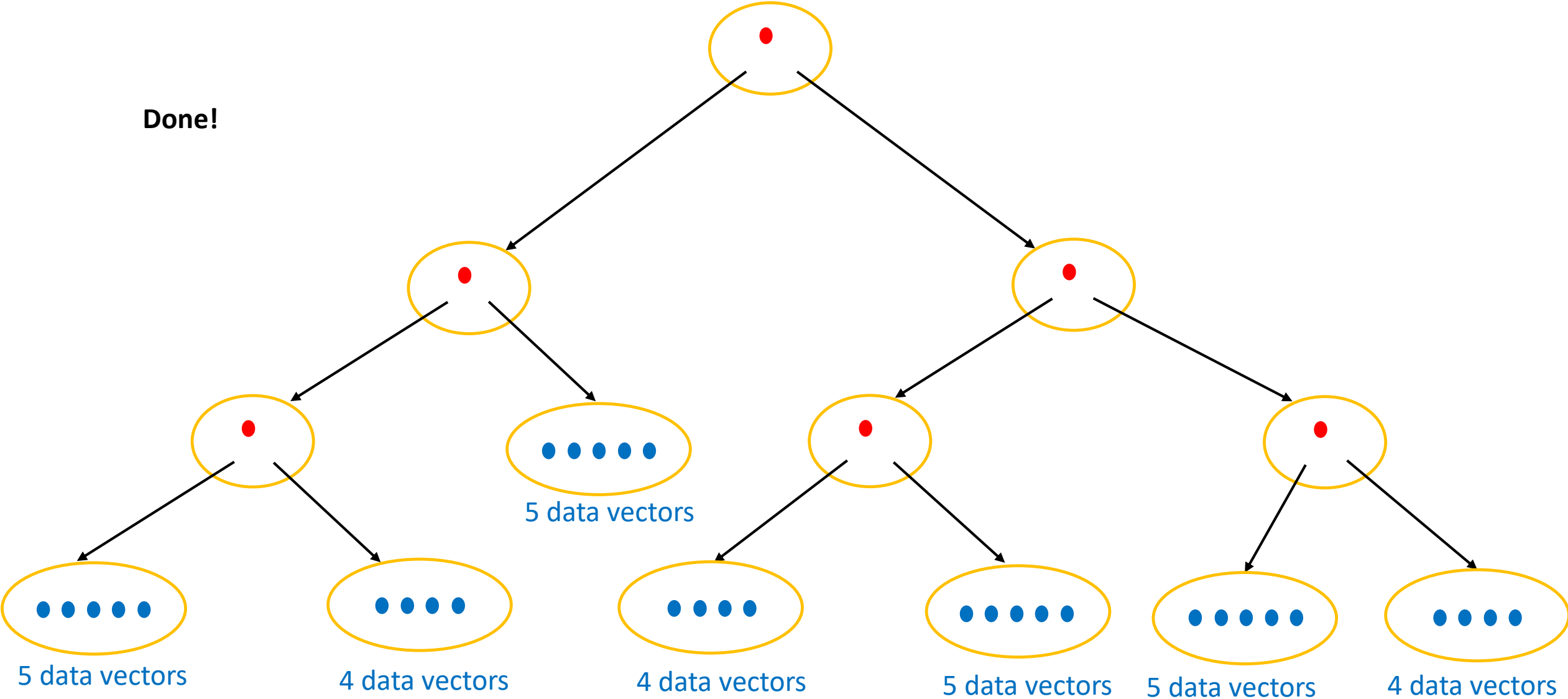
# Build Index Tree (GPU) - Illustration

- Find split and compute dot product in parallel.
- Assign one group to each block.



# Build Index Tree (GPU) - Illustration

Done!



## Build Index Tree (GPU) - Illustration

- **If initially the data vectors are larger than GPU meomry:**
  - **Use CPU to do split when group sizes still larger than GPU memory.**
  - **Switch to GPU when group sizes can fit into GPU.**

- **Build index for  $1 \times 10^6$  768-dimensional float vectors (3 GB < GPU memory).**

```
Done building in 33 secs.  
limit: 10      precision: 11.00% avg time: 0.001643s  
limit: 100     precision: 11.00% avg time: 0.001468s  
limit: 1000    precision: 12.00% avg time: 0.003323s  
limit: 10000   precision: 17.00% avg time: 0.026979s
```

```
Done building in 178 secs.  
limit: 10      precision: 10.00% avg time: 0.001487s  
limit: 100     precision: 10.00% avg time: 0.001389s  
limit: 1000    precision: 10.00% avg time: 0.004077s  
limit: 10000   precision: 12.00% avg time: 0.029807s
```

→ GPU roughly 6 time faster than CPU. Precision are similar.

- **Build index for  $5 \times 10^6$  768-dimensional float vectors (15 GB > GPU memory).**

```
Done building in 337 secs.  
limit: 10      precision: 10.00% avg time: 0.002013s  
limit: 100     precision: 10.00% avg time: 0.001815s  
limit: 1000    precision: 10.00% avg time: 0.003787s  
limit: 10000   precision: 10.00% avg time: 0.031639s
```

```
Done building in 1007 secs.  
limit: 10      precision: 10.00% avg time: 0.001900s  
limit: 100     precision: 10.00% avg time: 0.001730s  
limit: 1000    precision: 10.00% avg time: 0.003715s  
limit: 10000   precision: 10.00% avg time: 0.032597s
```

→ GPU roughly 3 time faster than CPU. Precision are similar.

- 
- **Speedup factor is 1 - 6, depending on dataset size (size of all vectors).**
  - **The speedup factor is larger for dataset that can be fit into GPU memory entirely.**
  - **The speedup factor could be almost 1 for dataset which are much larger than GPU memory.**



# Demo – Test on real-workd data: Wiki pages

- There are 1187751 wiki pages in total.
- Each wiki page will be encoded into 768-dimensional vector.
- Use the transformer model [shibing624/text2vec-base-chinese](#) to encode.

 Home Competitions Discussion Datasets Success Story 永瑛 吳永瑛

2023/04/06: 競賽主辦團隊提供參考程式碼，歡迎使用，請至此下載：<https://github.com/IKMLab/NCKU-AICUP2023-baseline>  
2023/03/30: 開放Public測試資料集下載與答案上傳功能。

## 真相只有一個: 事實文字檢索與查核競賽

已結束

Overview Leaderboard Download Dataset Submission History TEAM MANAGEMENT

### 競賽說明

近年來，新聞媒體及社群網路平台的資訊傳播速度越來越快，這也導致假資訊的問題越來越嚴重，混淆民眾的判斷。然而，目前絕大多數針對假資訊的處理都是基於人工辨識，驗證假資訊的過程繁複且耗時，往往跟不上假資訊的產生速度，如果能將驗證資訊的過程自動化，就能夠可以大幅降低一般民眾接觸到假資訊的頻率。

在AICUP2023中，本競賽將提供一個事實資料庫以及陳述句 (claim)，參賽者需要建立自動化的事實檢索與查核系統，以驗證陳述句的真偽。如果陳述句能夠「支持」或「反對」事實，系統也必須透過檢索資料庫中的文章來提供證據句。需要注意的是，由於事實有可能會隨時間推移而產生變化，參賽者只能使用我們提供的資料庫進行事實驗證。

  
301  
參賽隊伍

  
總獎金  
新台幣 31.00 萬元

開始 3/17/2023 結束 6/2/2023

# Demo – Test on real-workd data: Wiki pages

Print out wiki page content

```
▼ print wiki pages

1 for i, (k, v) in enumerate(pageDict.items()):
2
3     print("pageId: ", k, '\n')
4     for line in v:
5         print(f"line: ", line)
6
7     print("\n#####\n")
8
9     if(i>100):break

#####

pageId: 螫蛱蝶族
line: 螫蛱蝶族（學名：Charaxini）是蛱蝶科螫蛱蝶亞科中的一個族。
line: 大型蝴蝶，分佈於古熱帶界，即歐亞非澳四大洲之熱帶地區。
line:

#####

pageId: 沃德斯登
line: 沃德斯登（英語：Waddesdon /ˈwɒdzdən/）是一座位於英國白金漢郡艾爾斯伯里鎮的村莊。
line: 距艾爾斯伯里6英里。
line: 沃德斯登的經濟主要依賴於絲織品生產。
line:

#####

pageId: 王煥承
line: 王煥成，山西陽高人。
line: 1974年7月參加工作，1974年12月加入中國共產黨。
line: 中共中央黨校大學學歷，農業推廣碩士。
line:
line: 早年為內蒙古自治區清水河縣五良太青年農場插隊知青。
line: 1975年入伍後，歷任北京軍區守備二師戰士、文書、排長、正連職宣傳幹事，內蒙古軍區政治部
line: 1993年復員後，歷任內蒙古自治區人民防空辦公室幹部，內蒙古自治區人民防空通信站副站長等職。
line: 1996年12月，調入中共內蒙古自治區黨委辦公廳，歷任綜合處助理調研員、副處長、處長，助理
line: 2005年3月，任中共內蒙古自治區黨委副秘書長。
line: 2008年3月，兼任中共內蒙古自治區黨委政策研究室主任；2014年4月，再兼任中共內蒙古自治區黨委
line: 2015年1月，任中共內蒙古自治區黨委副秘書長、直屬機關工委書記。
line: 2016年1月，當選為內蒙古自治區政協副主席。
line:

#####
```

Encode each page into 768-dimension vectors.

```
40
41 from text2vec import SentenceModel
42 sm_model = SentenceModel('shibing624/text2vec-base-chinese')

▼ wiki page encode & save

[ ] 1
2     batch_size = 50000 # 1 wiki files
3     text_list = []
4
5     for i, (pageId, lines) in enumerate(pageDict.items()):
6
7         if(i < 20 * batch_size): continue
8         if(i >= 25 * batch_size): break
9
10        text_list.append("".join(lines))
11
12    print("len(text_list): ", len(text_list))

len(text_list): 187751

[ ] 1 vec_array = sm_model.encode(text_list)
2     vec_array.shape

(187751, 768)
```

## Demo – Test on real-workd data: Wiki pages

- **Build index for TBrain wiki page data: 1187751, 768-dimensional float vectors (< GPU memory).**


```
Done building in 58 secs.  
limit: 10      precision: 44.00% avg time: 0.001258s  
limit: 100     precision: 46.00% avg time: 0.001272s  
limit: 1000    precision: 66.00% avg time: 0.003273s  
limit: 10000   precision: 89.00% avg time: 0.021272s
```

```
Done building in 353 secs.  
limit: 10      precision: 45.00% avg time: 0.001838s  
limit: 100     precision: 45.00% avg time: 0.001726s  
limit: 1000    precision: 58.00% avg time: 0.003799s  
limit: 10000   precision: 93.00% avg time: 0.025689s
```


→ GPU roughly 6 time faster than CPU. Precision are similar.


→ We are able to obtain much higher precision scores on real world data than we do in synthetic random data

# Demo – Pip install and run on colab




Search projects



 Ri-chard-Wu ▾

# annoy-gpu 1.0.8

`pip install annoy-gpu`



✓


[Latest version](#)


Released: about 1 hour ago


Approximate Nearest Neighbors in C++/Python optimized for memory usage and loading/saving to disk. Can run with GPU speedup.

Manage project

Navigation

 Project description

 Release history

 Download files

Project description

Note

This project is derived from [Annoy](#). The original project can use multi-thread to accelerate build process. In this project GPU is used to accelerate the build process. This project is still under developing. Currently it only support the Angular metrics.

# Demo – Pip install and run on colab

PRLC-final-project.ipynb

檔案 編輯 檢視畫面 插入 執行階段 工具 說明

+ 程式碼 + 文字

### install & import annoy\_gpu

```
1 !export CUDAHOME=/usr/local/cuda-11.8
2 !pip install annoy_gpu
3
4 import random, time
5 from annoy_gpu import AnnoyIndex
```

Looking in indexes: <https://pypi.org/simple>, <https://us-python.pkg.dev>  
Collecting annoy\_gpu  
Downloading annoy\_gpu-1.0.8.tar.gz (32 kB)  
Installing build dependencies ... done  
Getting requirements to build wheel ... done

### add & save items

```
1 n, f = 10000000, 768
2
3 t = AnnoyIndex(f, 'angular')
4 t.fill_items('test-le7.tree')
5
6 for i in range(n):
7
8     if(i % 1000 == 0): print(f"{i} / {n} = {1.0 *
9
10         v = []
11         for z in range(f):
12             v.append(random.gauss(0, 1))
13         t.add_item(i, v)
14
15 t.save_items()
```

資源 ×

你並未訂閱這項服務。瞭解詳情。  
目前沒有可用的運算單元。本產品不保證免費版本的資源分配。如需購買更多運算單元，請按這裡。  
管理工作階段

需要更多記憶體和磁碟空間嗎？  
升級至 Colab Pro

Python 3 Google Compute Engine 後端  
目前顯示自 下午1:54 以來的資源

系統 RAM  
1.0 / 12.7 GB

磁碟  
23.3 / 107.7 GB

變更執行階段類型

PRLC-final-project.ipynb

檔案 編輯 檢視畫面 插入 執行階段 工具 說明

+ 程式碼 + 文字

### build & precision test

```
1 n, f = 1000000, 768
2
3 def precision_test(q):
4     limits = [10, 100, 1000, 10000]
5     k = 10
6     prec_sum = {}
7     prec_n = 10
8     time_sum = {}
9
10    for i in range(prec_n):
11        j = random.randrange(0, n)
12        closest = set(q.get_nns_by_item(j, k, n))
13        for limit in limits:
14            t0 = time.time()
15            toplist = q.get_nns_by_item(j, k, limit)
16            T = time.time() - t0
17
18            found = len(closest.intersection(toplist))
19            hitrate = 1.0 * found / k
20            prec_sum[limit] = prec_sum.get(limit, 0.0) + hitrate
21            time_sum[limit] = time_sum.get(limit, 0.0) + T
22
23    for limit in limits:
24        print('limit: %-9d precision: %6.2f%% avg time: %.6fs'
25              % (limit, 100.0 * prec_sum[limit] / (i + 1), time_sum[lim
26
27    q = AnnoyIndex(f, 'angular')
28    q.load_items('test-le6.tree')
29    q.build(30)
30    precision_test(q)
```

limit: 10 precision: 10.00% avg time: 0.002227s  
limit: 100 precision: 10.00% avg time: 0.002058s  
limit: 1000 precision: 11.00% avg time: 0.003194s  
limit: 10000 precision: 13.00% avg time: 0.016867s

資源 ×

你並未訂閱這項服務。瞭解詳情。  
目前沒有可用的運算單元。本產品不保證免費版本的資源分配。如需購買更多運算單元，請按這裡。  
管理工作階段


Python 3 Google Compute Engine 後端 (GPU)  
目前顯示凌晨 1:45 至凌晨 1:57 之間的資源

系統 RAM  
1.5 / 12.7 GB

GPU RAM  
0.1 / 15.0 GB

磁碟  
26.5 / 78.2 GB

# Demo – annoy-gpu Github



Search or jump to...

PullsIssuesCodespaces

Ri-chard-Wu / annoy-gpu Public

Pin

Unwatch 1

<> Code

Issues

Pull requests

Actions

Projects


Wiki

master

Go to file

Add file

<> Code

 Ri-chard-Wu update readme ... 4 minutes ago 5

|               |               |         |               |
|---------------|---------------|---------|---------------|
| annoy_gpu...  | update readme | ...     | 5 minutes ago |
| annoy_gpu     | backup        | ...     | 3 hours ago   |
| debian        | first commit  | ...     | 4 hours ago   |
| dist          | update readme | ...     | 5 minutes ago |
| examples      | first commit  | ...     | 4 hours ago   |
| src           | update readme | ...     | 5 minutes ago |
| test          | update readme | ...     | 5 minutes ago |
| CMakeLists... | first commit  | 917 B   | 4 hours ago   |
| LICENSE       | first commit  | 11.1 KB | 4 hours ago   |
| MANIFEST.in   | first commit  | 102 B   | 4 hours ago   |

README.rst

annoy-gpu

This project is derived from [Annoy](#). The original project can use multi-thread to accelerate build process. In this project GPU is used to accelerate the build process. This project is still under development. Currently it only support the 'angular' metrics.

Install

First set the environment variable "CUDAHOME" to CUDA installation location, e.g. `export CUDAHOME=/usr/local/cuda-10.2`, then run `pip install annoy_gpu` to pull down the latest version from [PyPI](#).

Python code example

Add items and then save it in the file 'test-1e5.tree'.

```
from annoy_gpu import AnnoyIndex
import random, time

n, f = 100000, 768

t = AnnoyIndex(f, 'angular')

t.fill_items('test-1e5.tree')
```

28/28





**Thanks for your attention**