ANNOY-GPU: Approximate Nearest Neighbor Oh Yeah – GPU

Final Project Presentation

吳永玹

Outline

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

Goal

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

Motivation

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

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

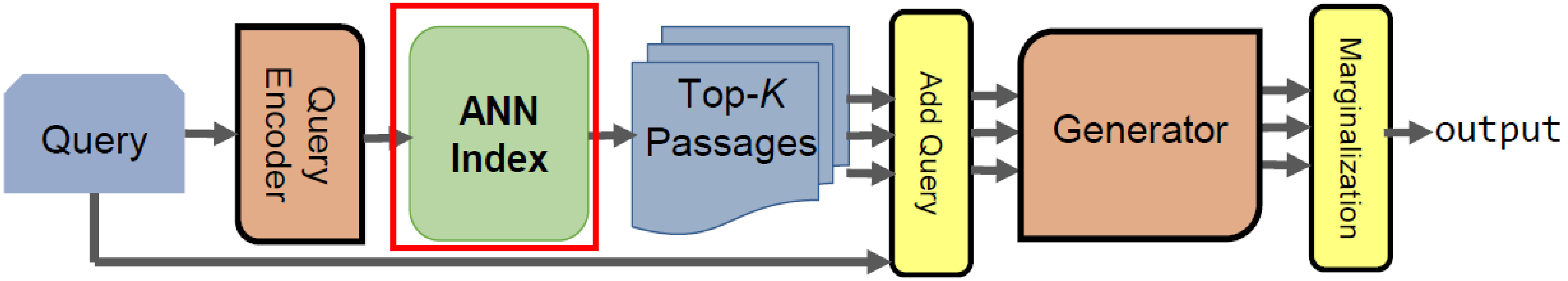
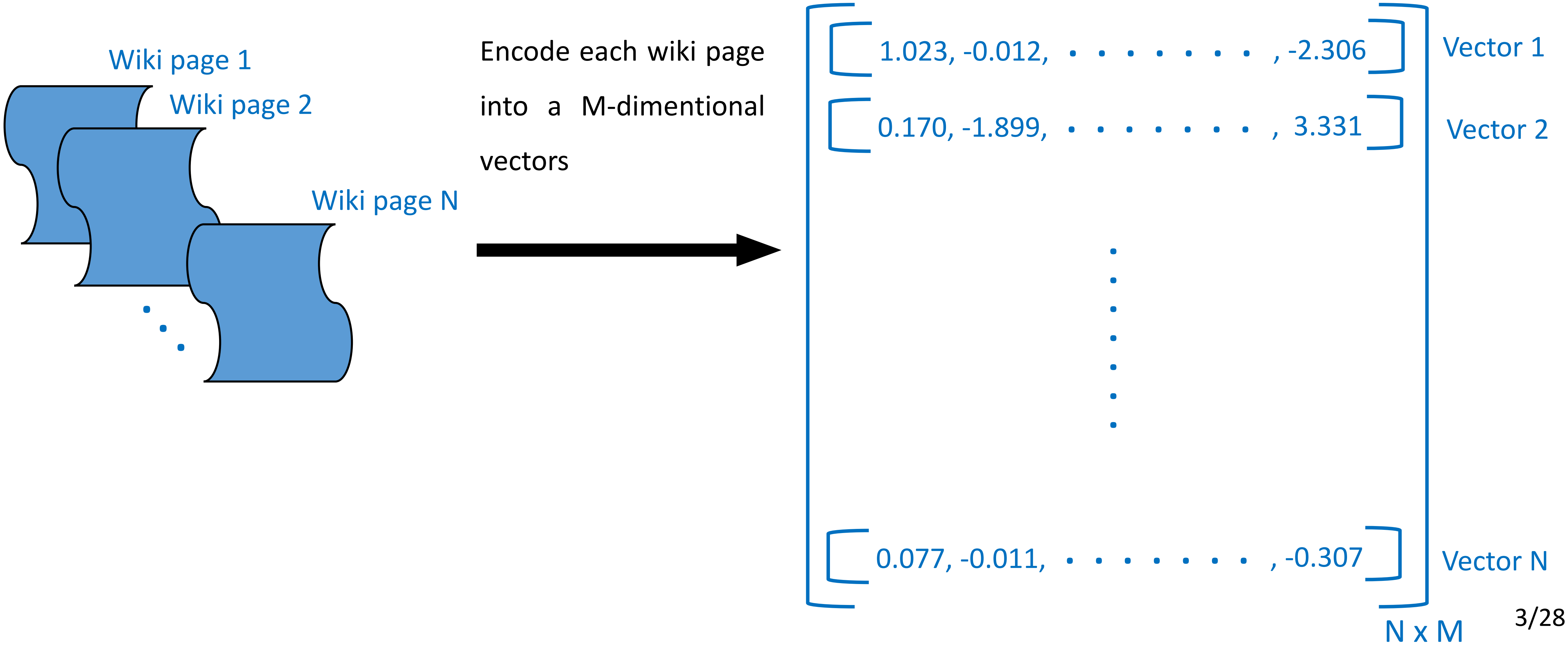


Image source: Michael Glass, Gaetano Rossiello, Md Faisal Mahbub Chowdhury, Ankita Rajaram Naik1, Pengshan Cai1, Alfio Gliozzo. *Re2G: Retrieve, Rerank, Generate*, 2022.

ANN semantic retrival steps

Step 1



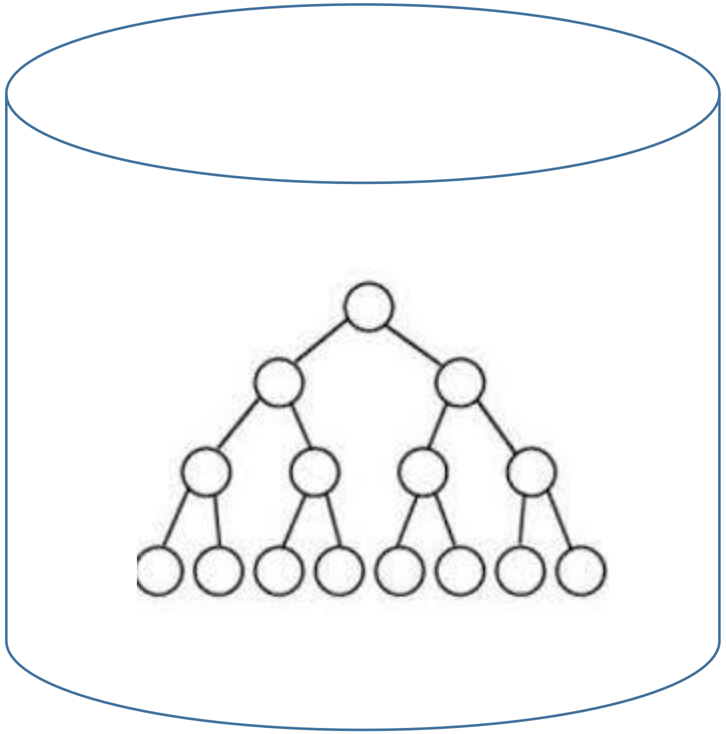
ANN semantic retrival steps

Step 2

$$\begin{bmatrix} [\,1.023, -0.012, \cdots\cdots, -2.306\,] \\ [\,0.170, -1.899, \cdots\cdots, 3.331\,] \\ \vdots \\ [\,0.077, -0.011, \cdots\cdots, -0.307\,] \end{bmatrix} \begin{array}{l} \text{Vector 1} \\ \text{Vector 2} \\ \\ \text{Vector N} \end{array}$$

N x M

Build ANN index tree on disk



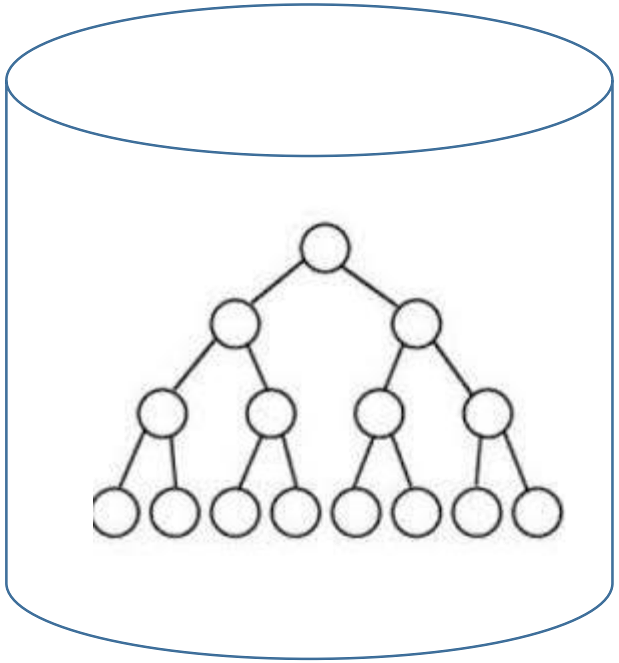
ANN semantic retrival steps

Step 3

Fast semantic retival using the built ANN index tree.

Encode the query into vectors using same encoder model.

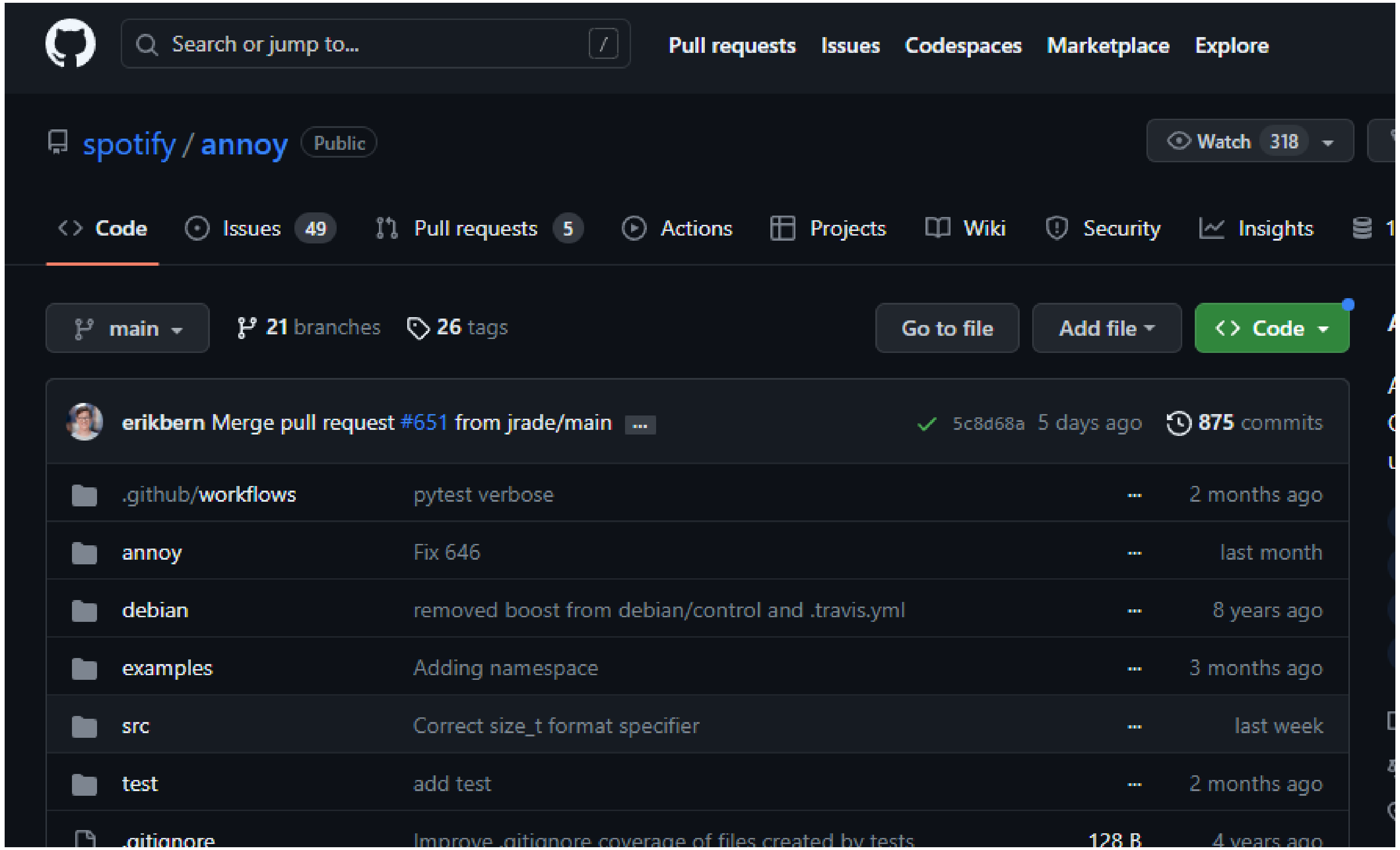
Query → vector →



→ Top K nearest neighbors

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.**

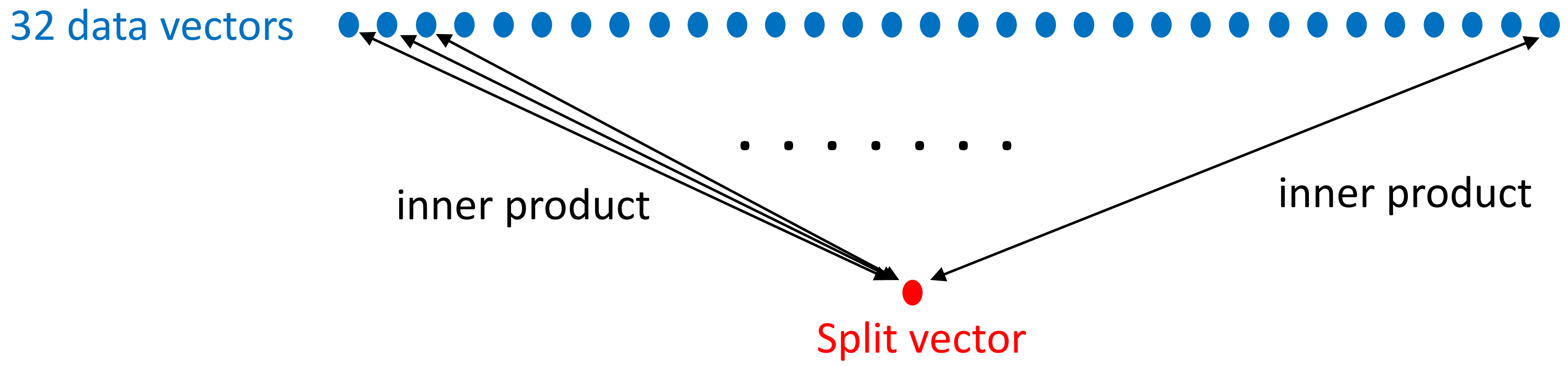


Build Index Tree (CPU) - Illustration

32 data vectors

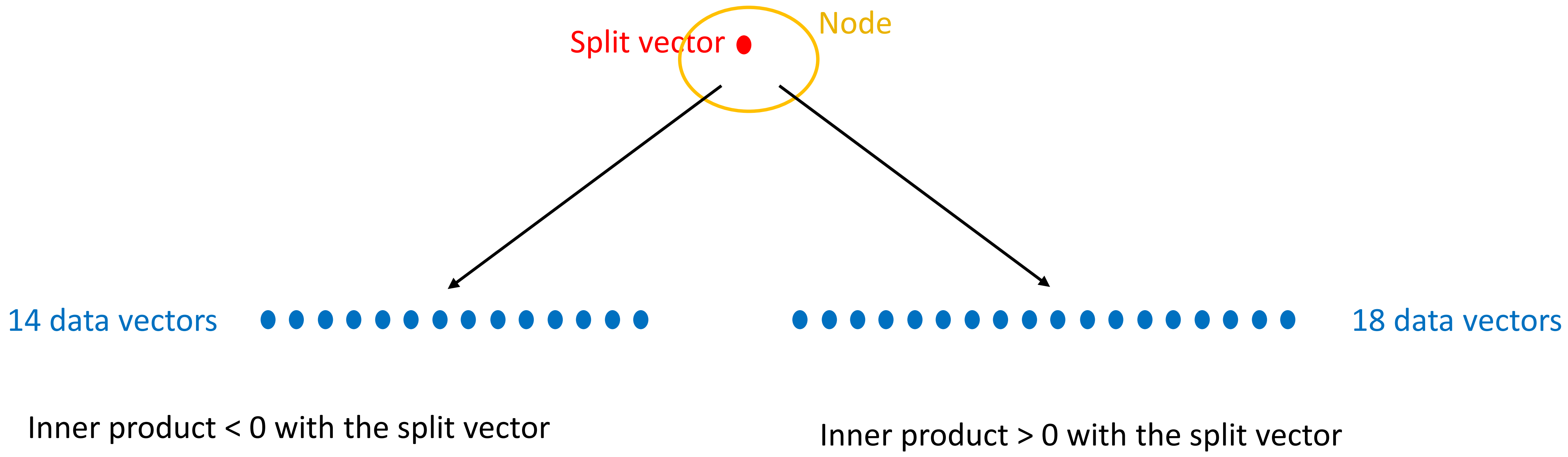
- **We want to build index tree for a dataset of 32 vectors.**

Build Index Tree (CPU) - Illustration



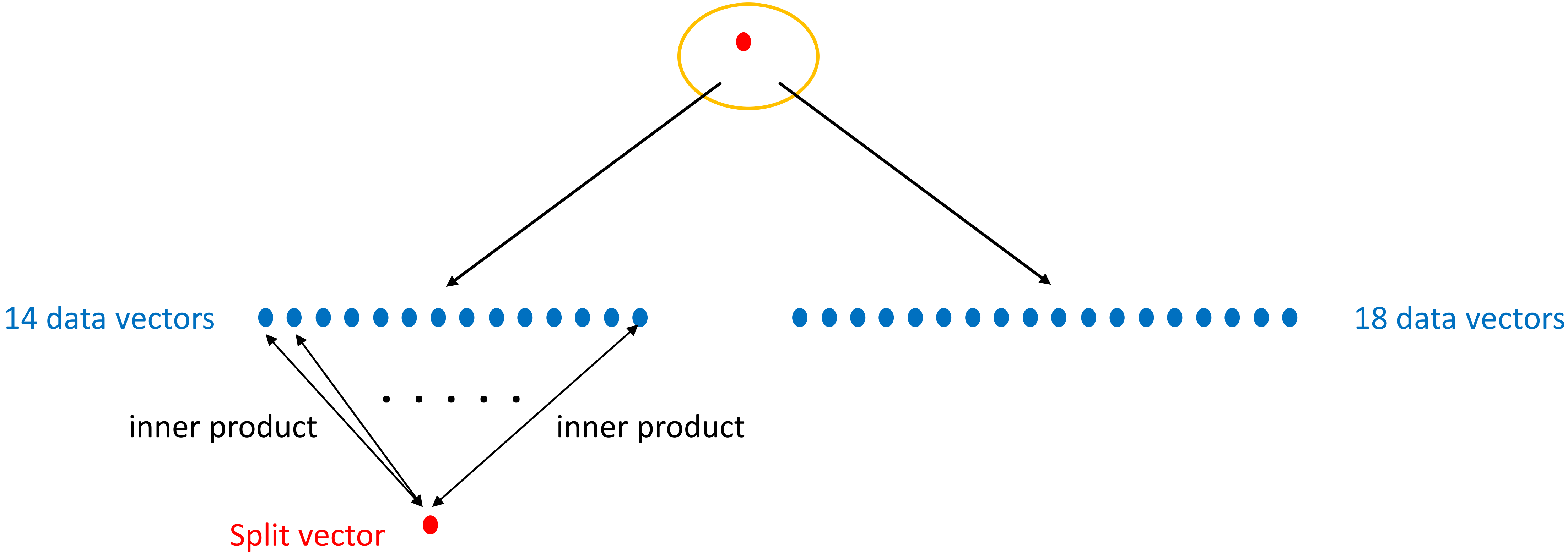
- **Find a split vector that evenly divides the group of vectors (by dot product value > or < 0).**

Build Index Tree (CPU) - Illustration

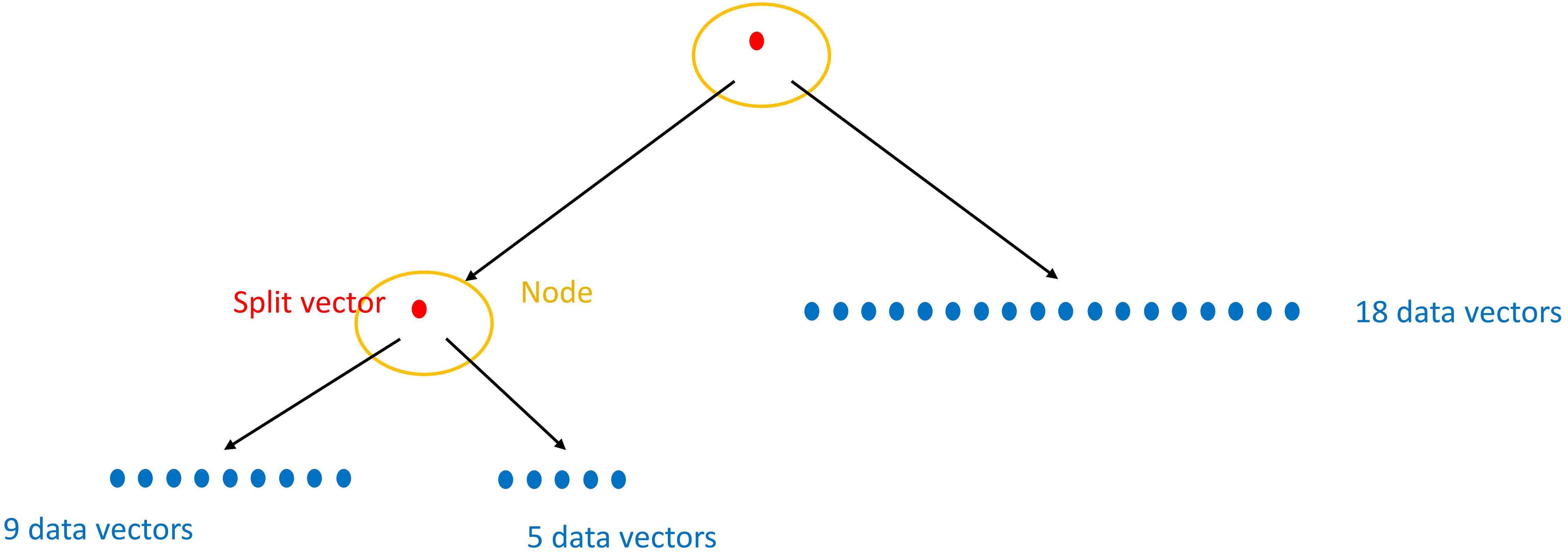


- **Split the group of 32 data vector into 2 groups based on whether their inner product value with the split vector is larger than or smaller than 0.**
- **In case inner product is 0 → random.**
- **Compute imbalance: if $max(\frac{n_{left}}{n_{left}+n_{right}}, \frac{n_{right}}{n_{left}+n_{right}})$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

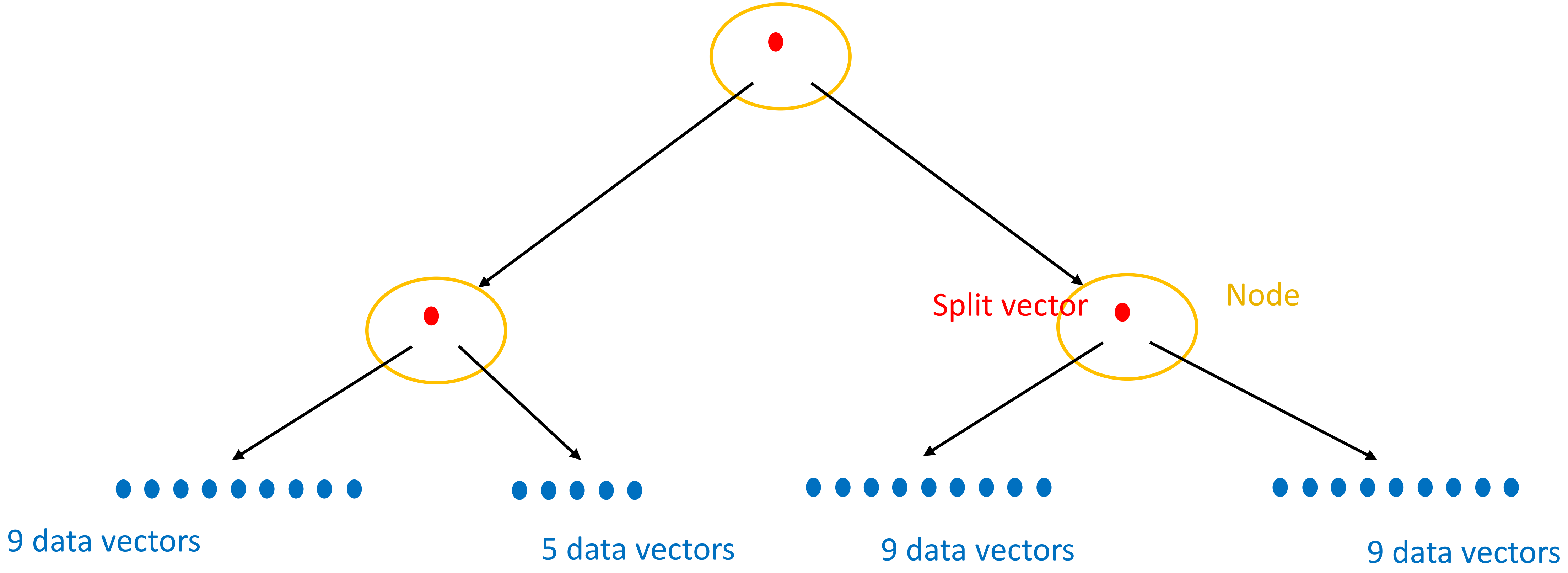
Build Index Tree (CPU) - Illustration



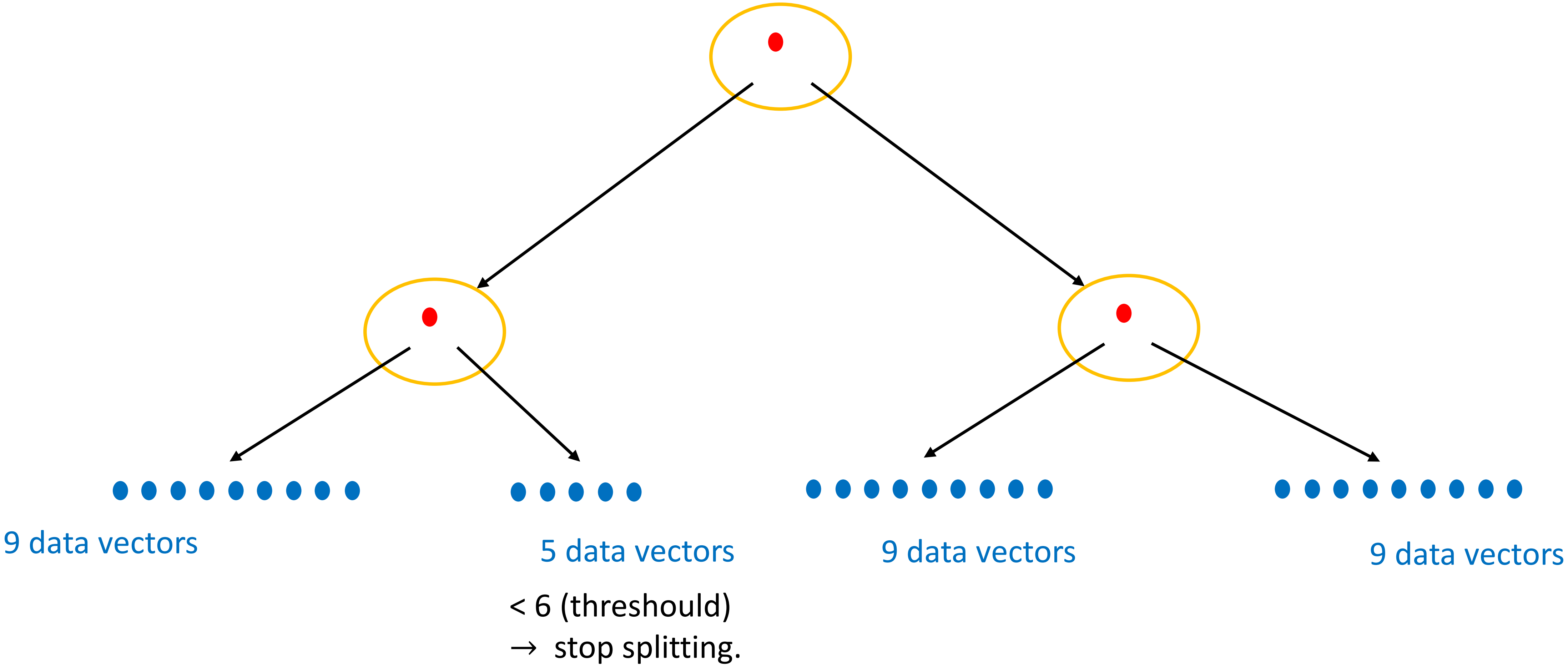
Build Index Tree (CPU) - Illustration



Build Index Tree (CPU) - Illustration

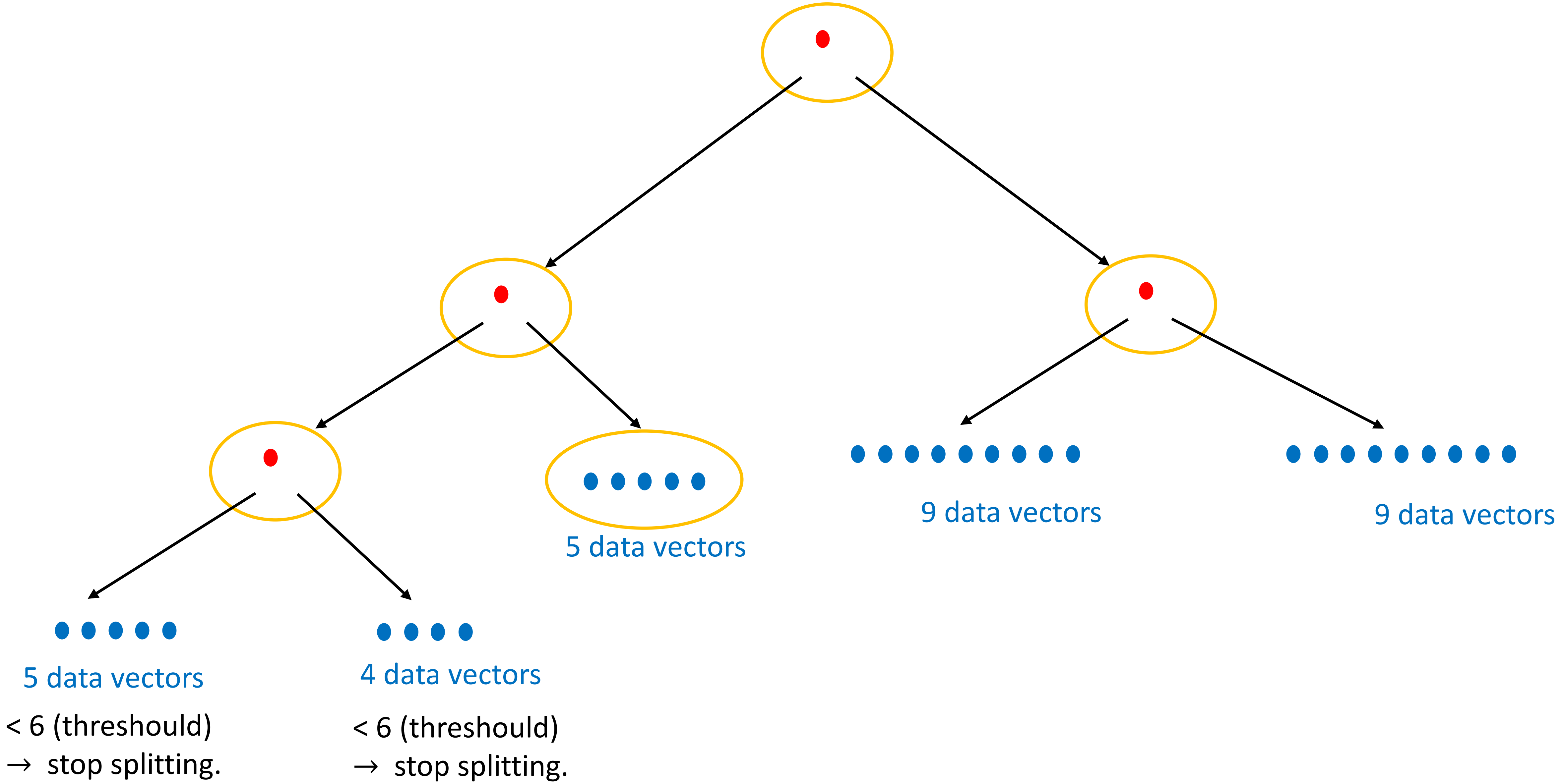


Build Index Tree (CPU) - Illustration

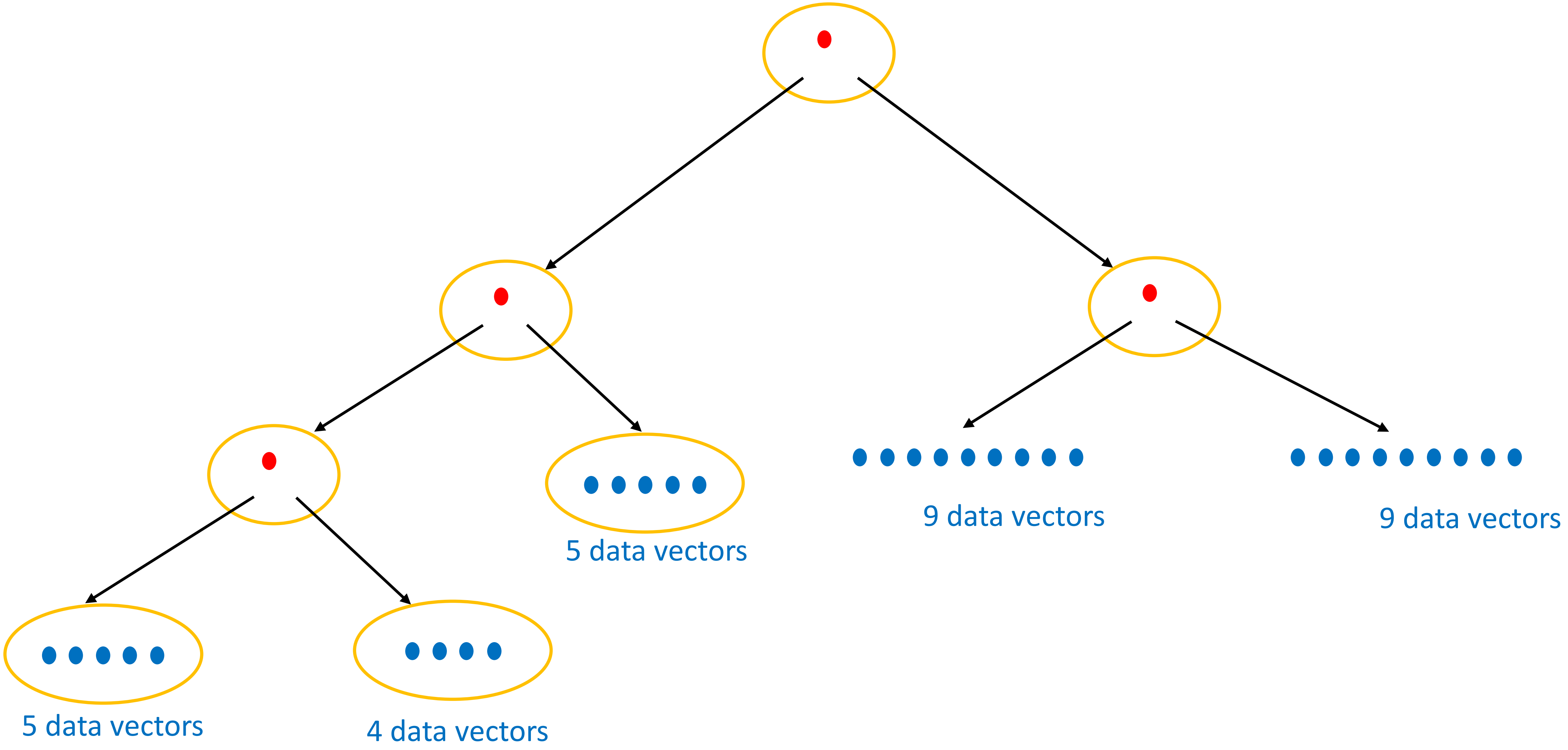


- **A group will stop splitting when its data vector numbr < a threshould. Assume threshould is 6.**
- **Will create a Node to hold the group of vectors that can no longer be splitted.**

Build Index Tree (CPU) - Illustration

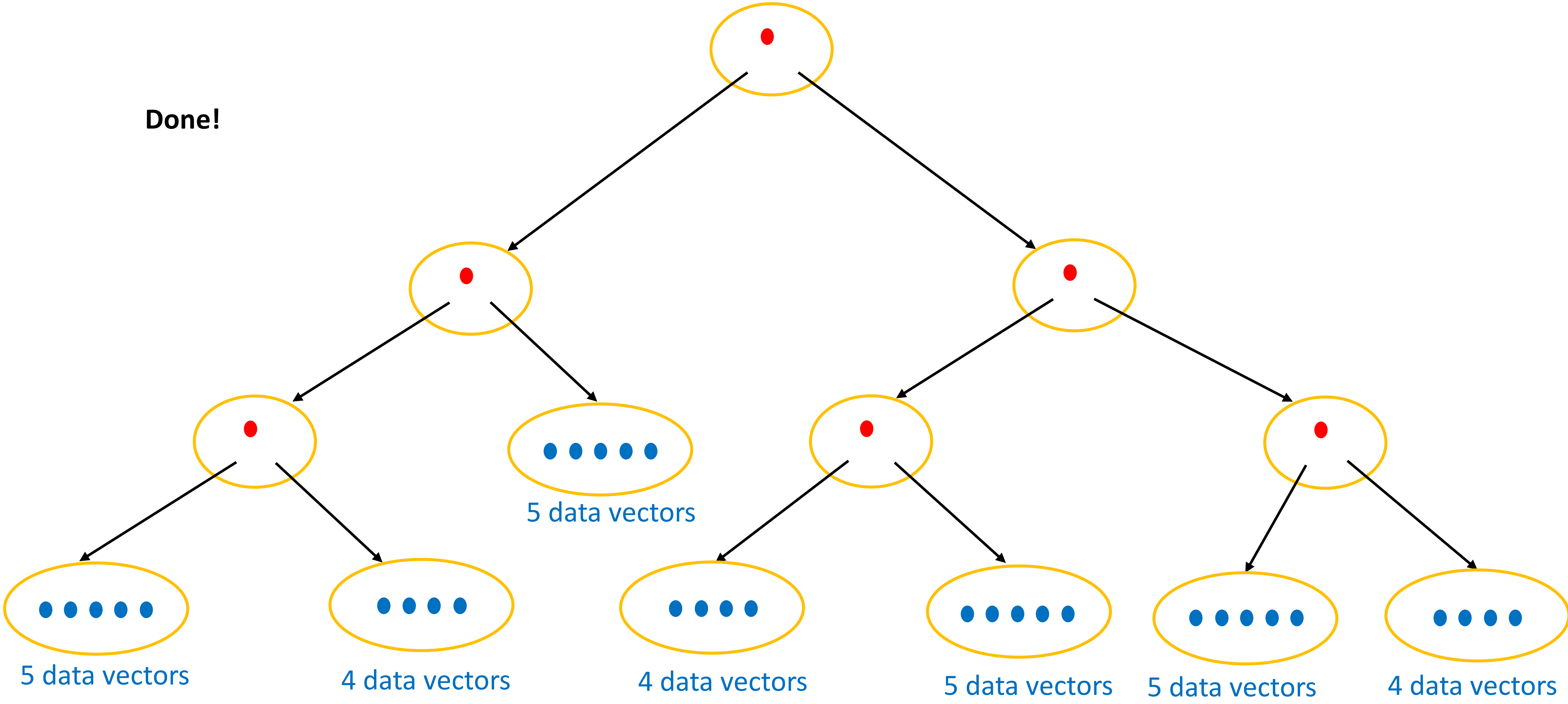


Build Index Tree (CPU) - Illustration

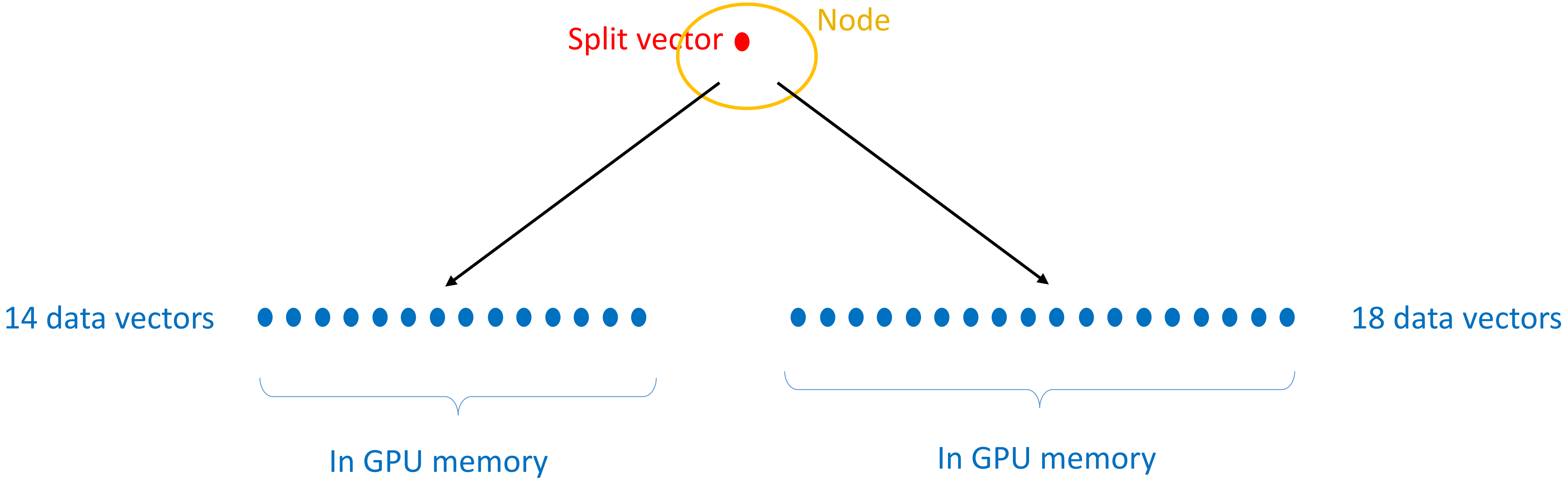


Build Index Tree (CPU) - Illustration

Done!

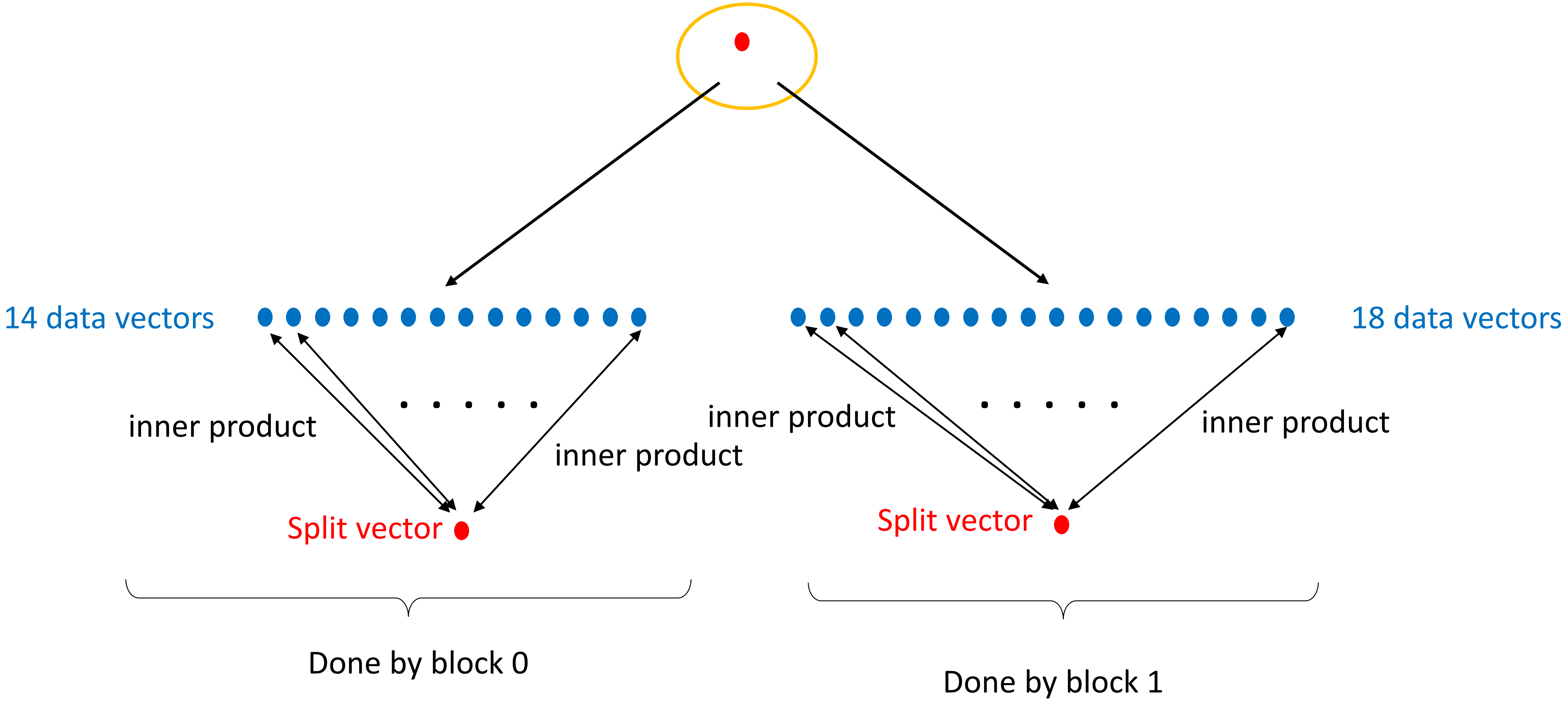


Build Index Tree (GPU) - Illustration



- **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.**

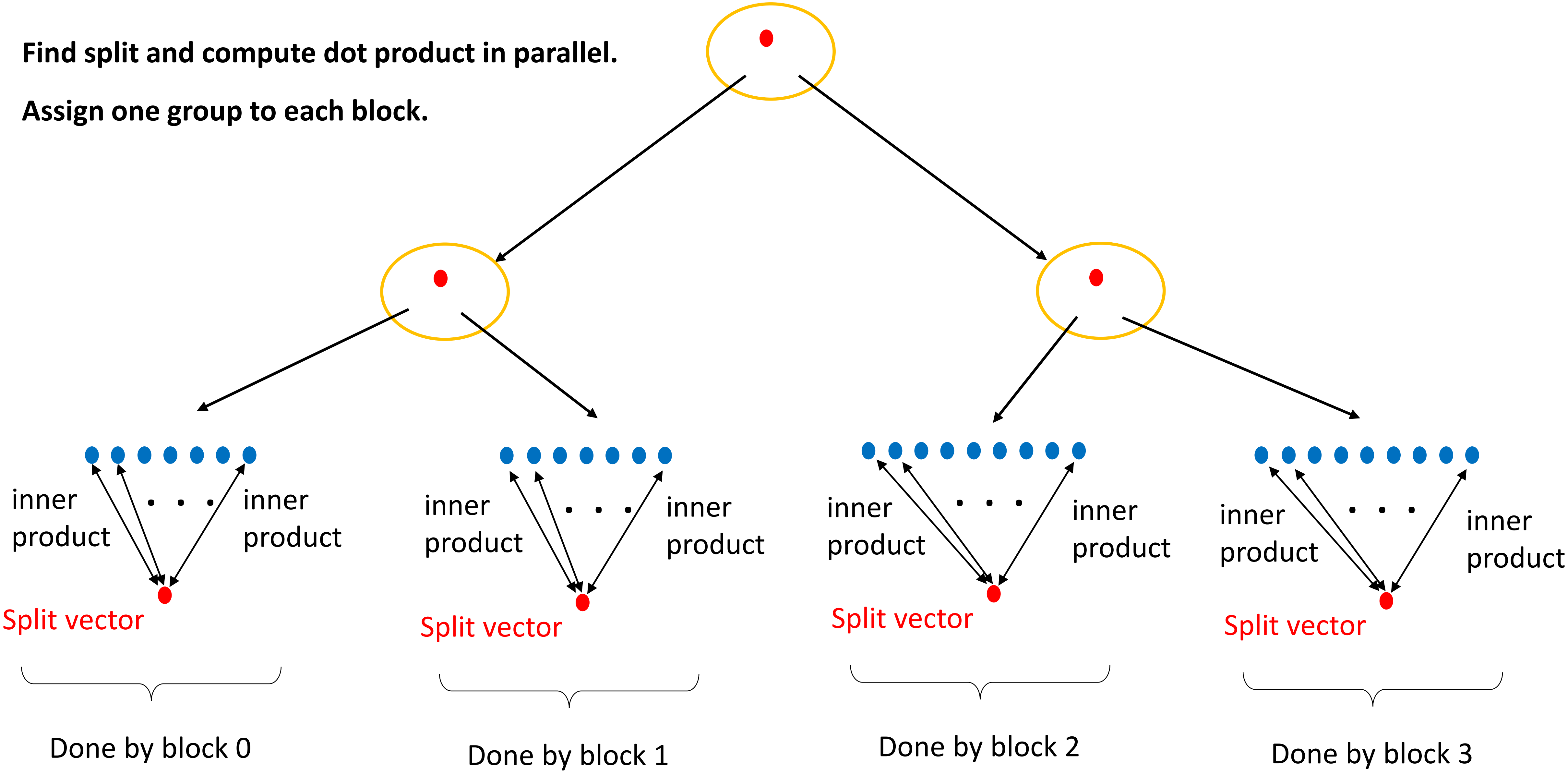
Build Index Tree (GPU) - Illustration



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

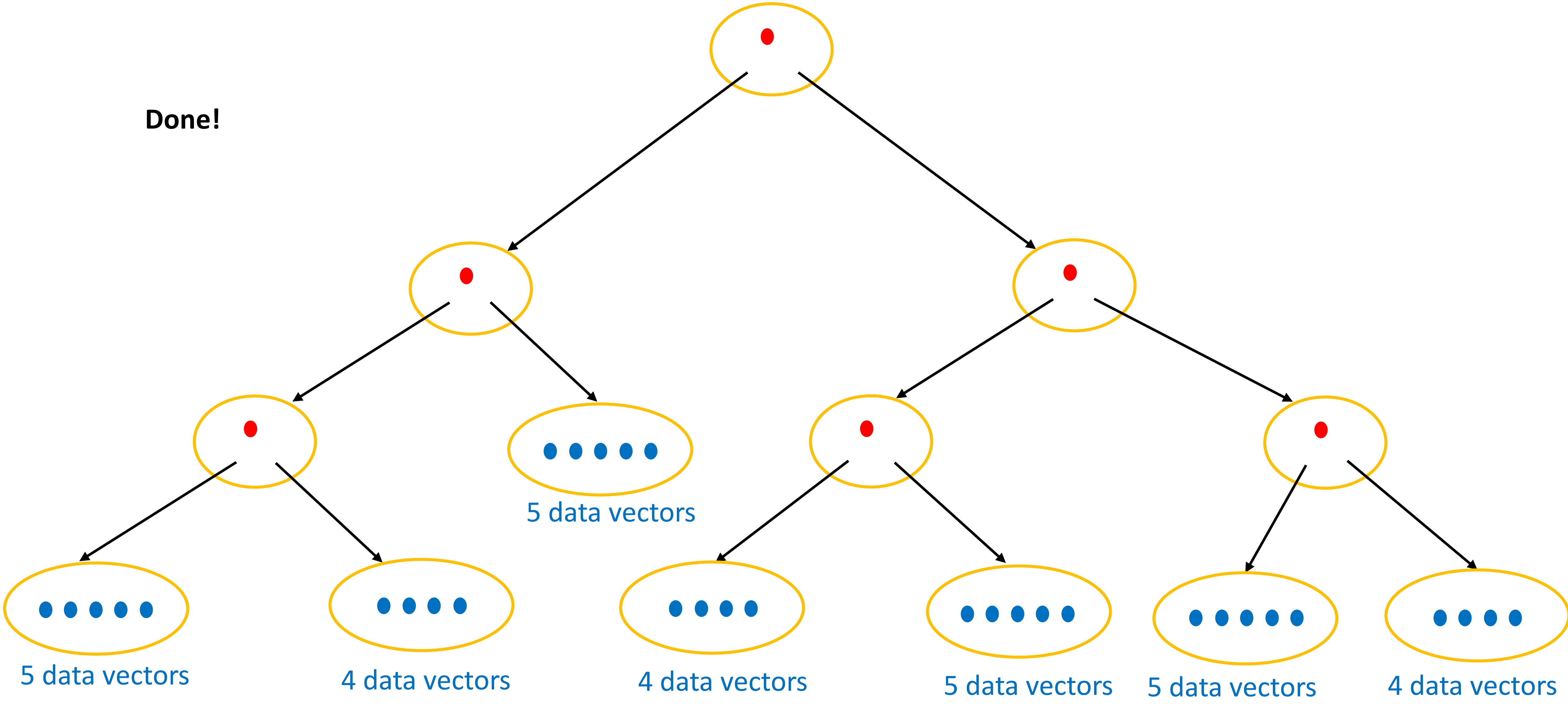
Build Index Tree (GPU) - Illustration

- **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.**

Performance

- **Build index for 1 x 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 x 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.**

Demo – Test on real-workd data: Wiki pages

Print out wiki page content

print wiki pages

```python
for i, (k, v) in enumerate(pageDict.items()):

    print("pageId: ", k, '\n')
    for line in v:
        print("line: ", line)

    print("\n##############################################################\n")

    if(i>100):break
```

```
##############################################################
pageId:  螯蛱蝶族

line:  螯蛱蝶族 （ 學名 ： Charaxini ） 是蛱蝶科螯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裏韋爾的村莊 。
line:  距艾爾斯伯裏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:
line:  早年爲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祕書長 。
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祕書長 、 直屬機關工委書記 。
line:  2016年1月 ， 當選爲內蒙古自治區政協副主席 。
line:
##############################################################
```

Page

Page

Page

Encode each page into 768-dimension vectors.

```python
from text2vec import SentenceModel
sm_model = SentenceModel('shibing624/text2vec-base-chinese')
```

wiki page encode & save

```python

batch_size = 50000 # 1 wiki files
text_list = []

for i, (pageId, lines) in enumerate(pageDict.items()):

    if(i < 20 * batch_size): continue
    if(i >= 25 * batch_size): break

    text_list.append("".join(lines))

print("len(text_list): ", len(text_list))
```

```
len(text_list):  187751
```

```python
vec_array = sm_model.encode(text_list)
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).**

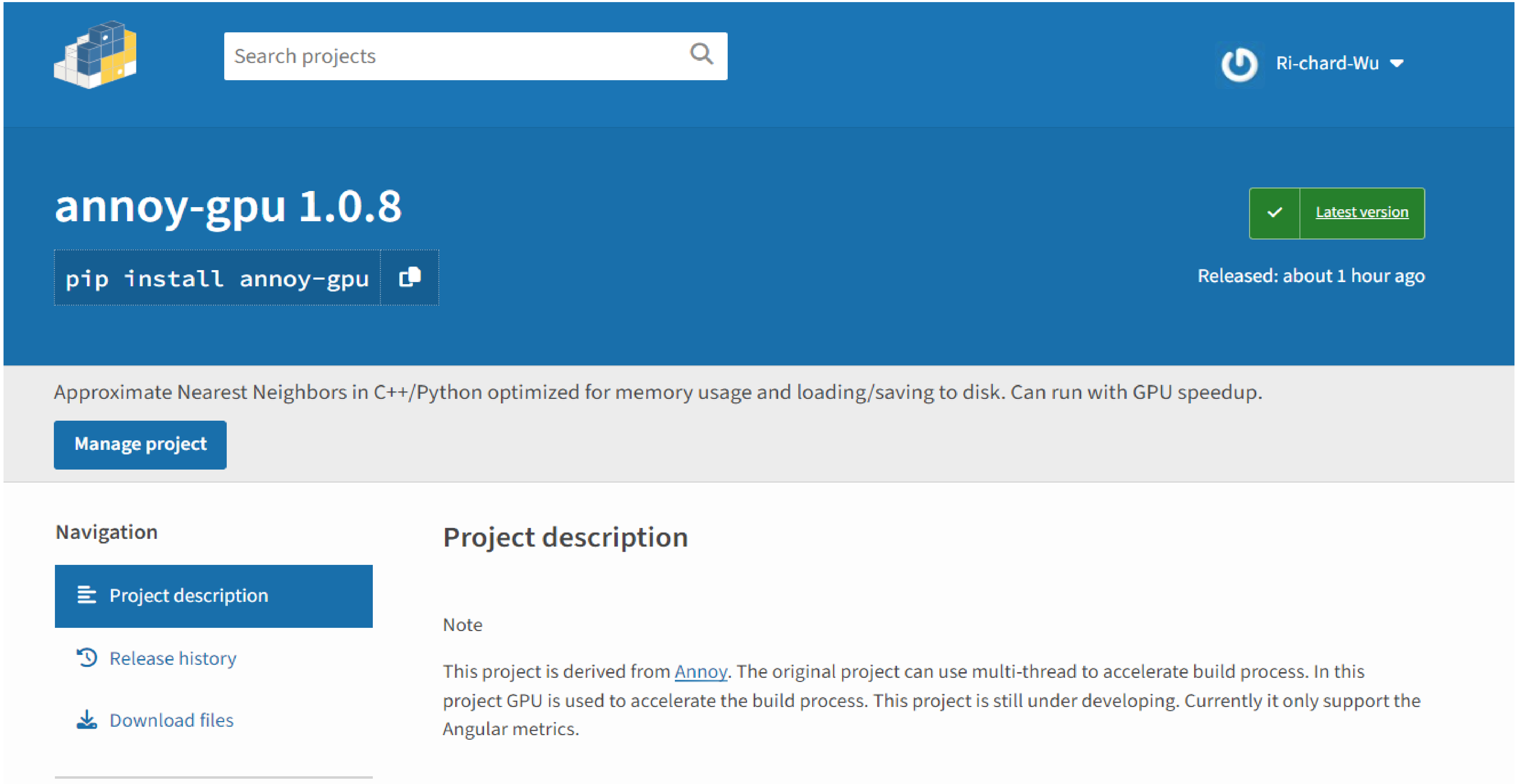
```
Done building in 58 secs.
limit: 10        precision:  44.00% avg time: 0.001258s
limit: 100       precision:  46.00% avg time: 0.001272s
limit: 1000      precision:  66.00% avg time: 0.003273s
limit: 10000     precision:  89.00% avg time: 0.021272s
```

```
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
```

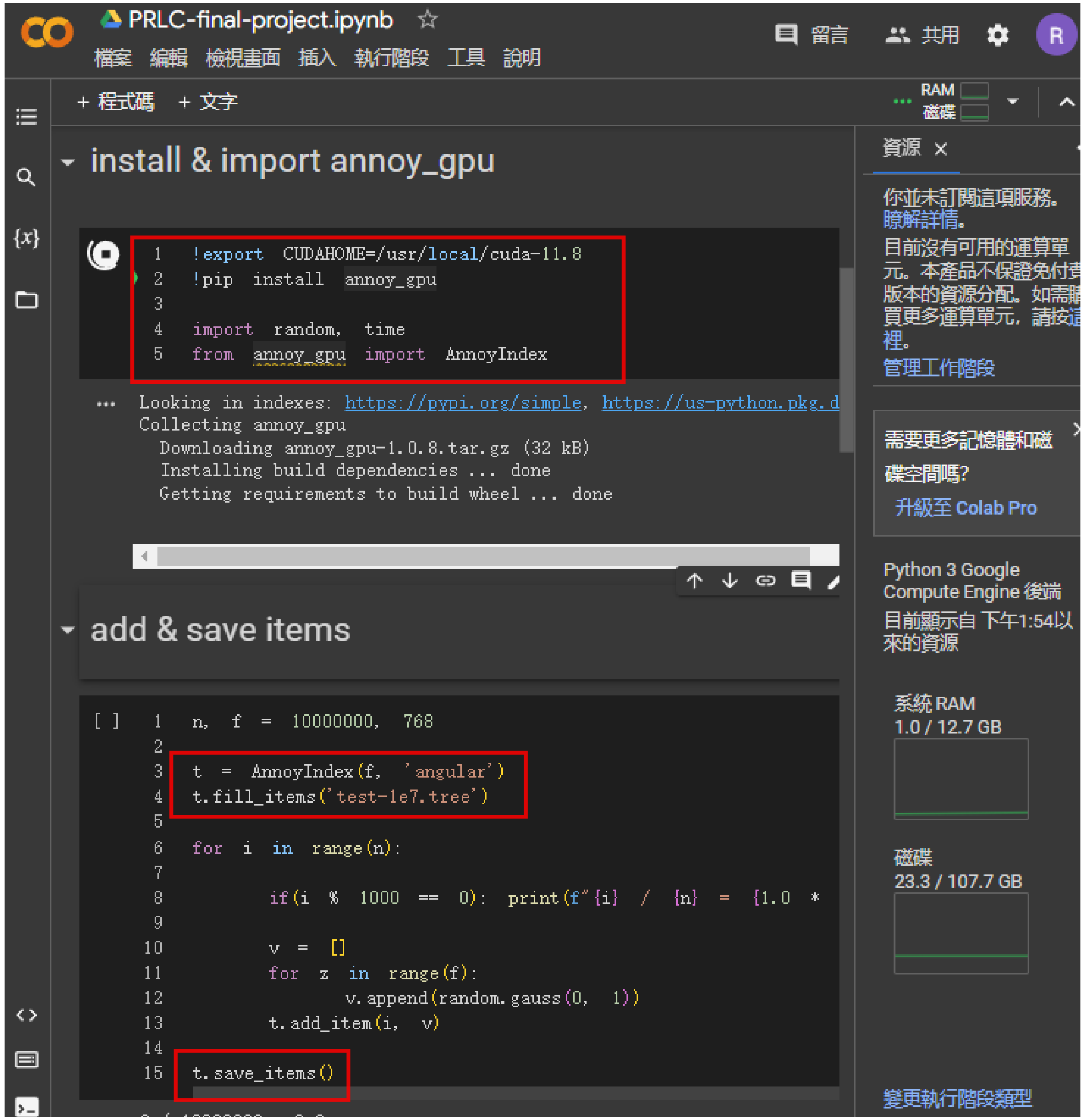
→ GPU roughly 6 time faster than CPU. Precision are similar.

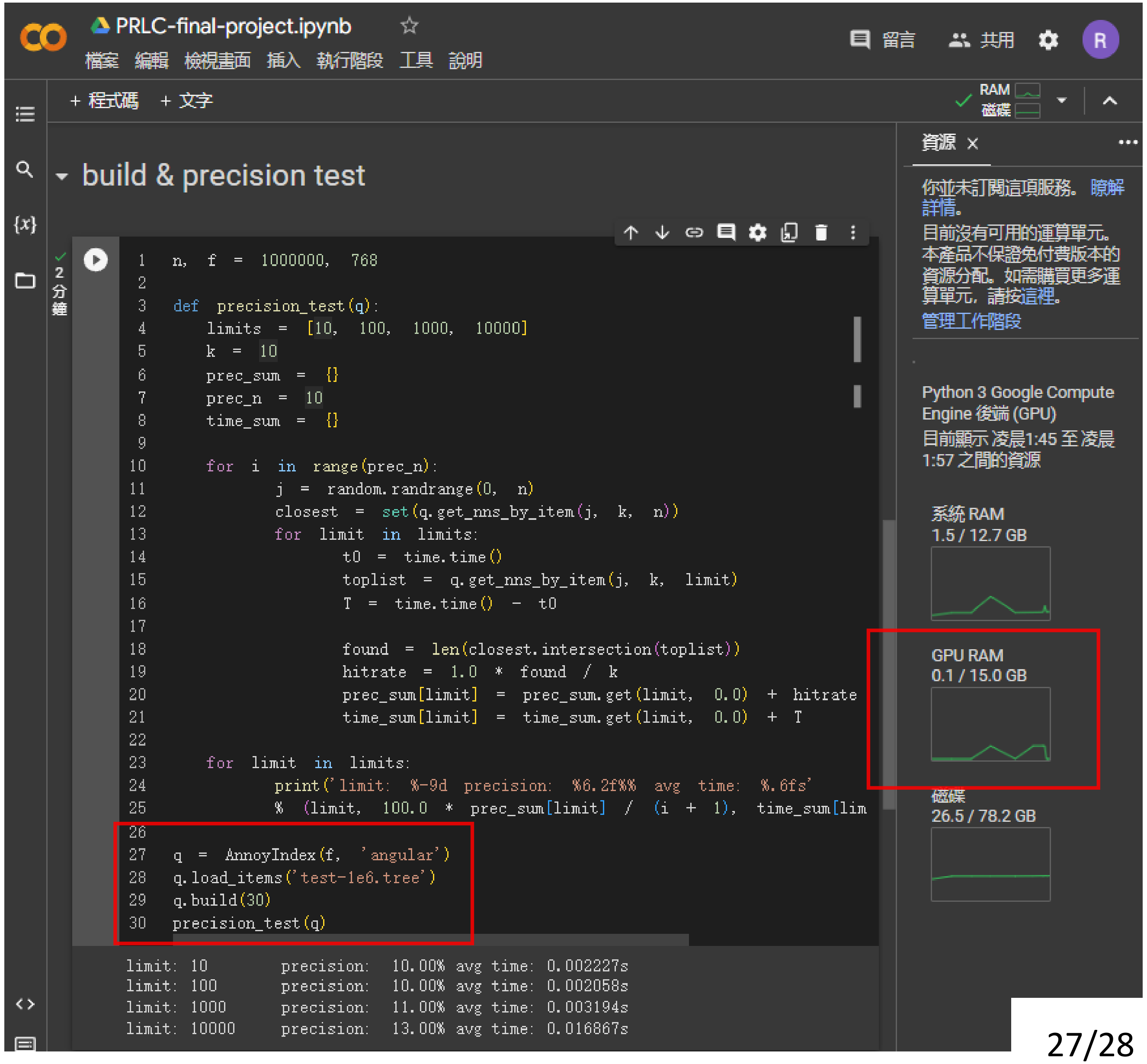
→ We are able to obtain much higher precision scores on real world data than we do in synthetic random data

Demo – Pip install and run on colab

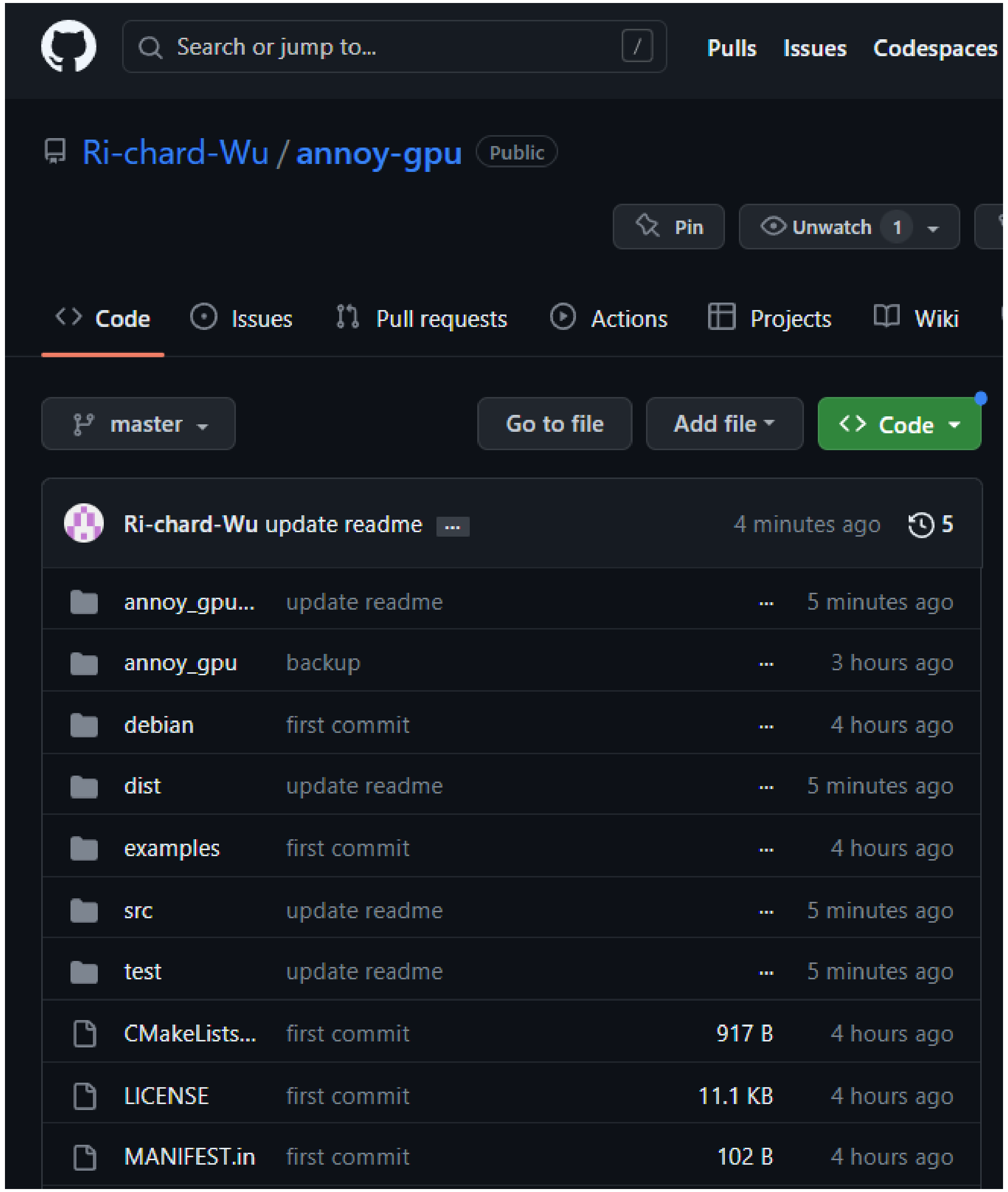


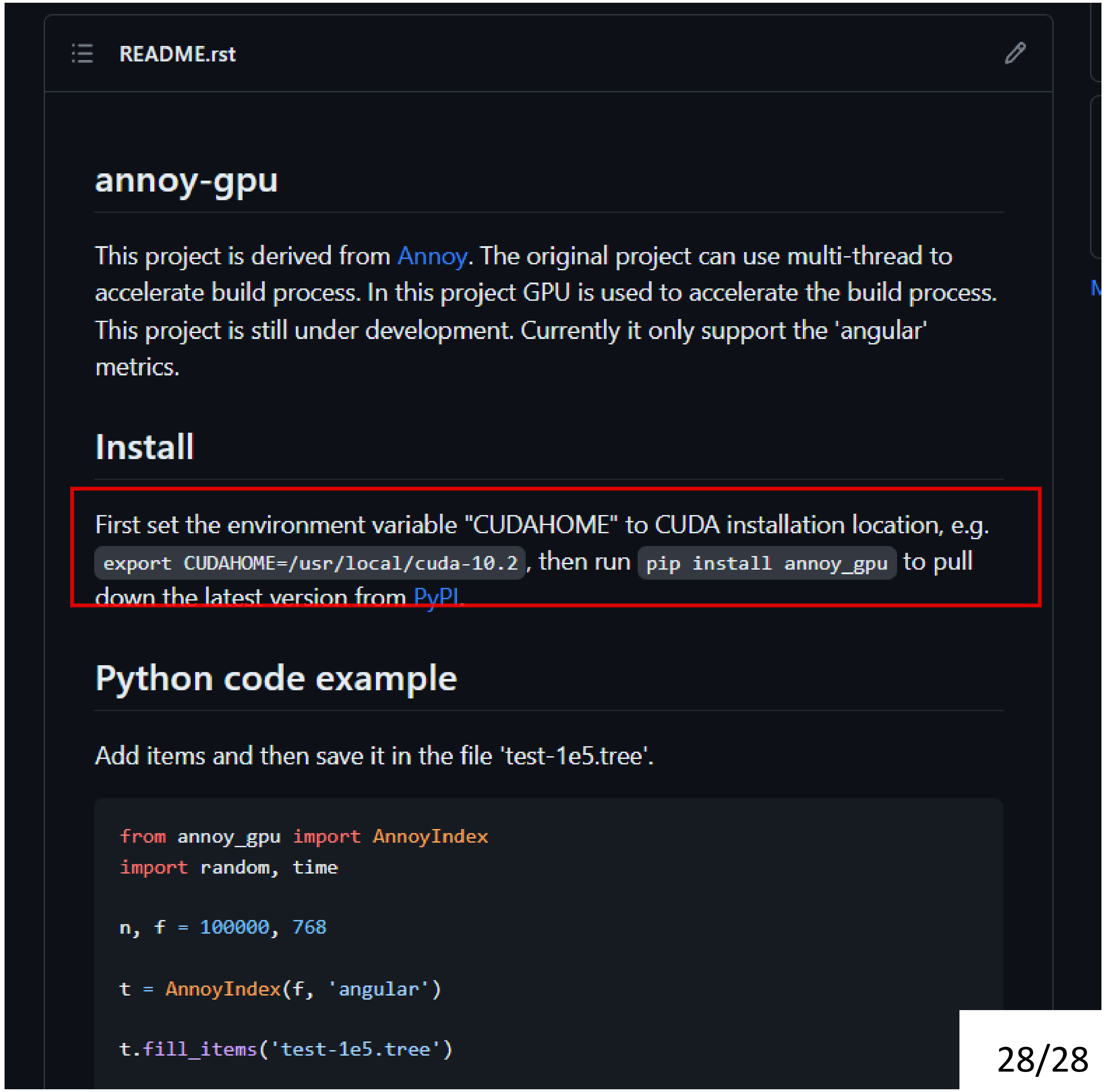
Demo – Pip install and run on colab





Demo – annoy-gpu Github





Thanks for your attention

Image source: https://github.com/spotify/annoy