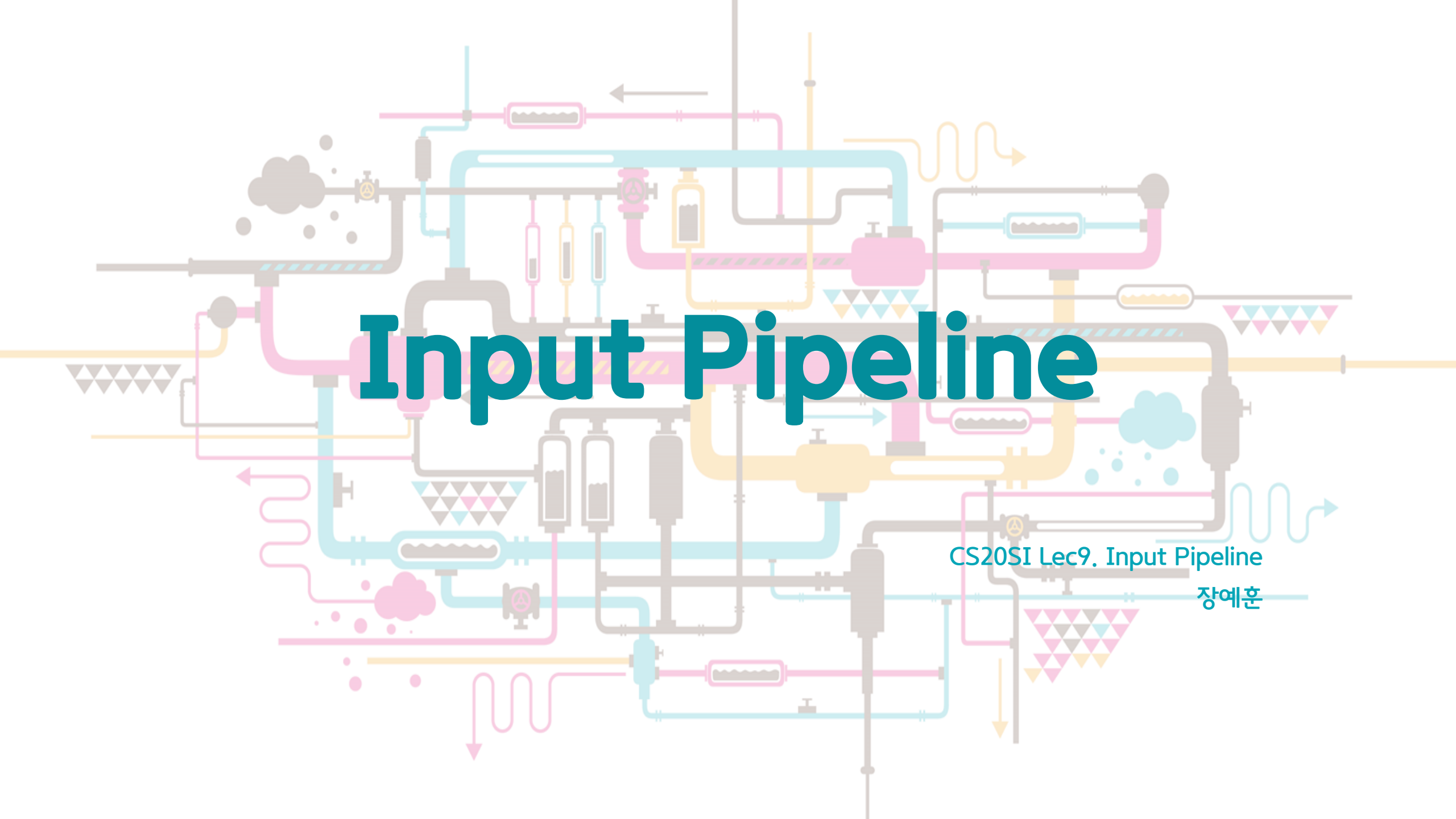


Input Pipeline

CS20SI Lec9. Input Pipeline

장예훈

Pipeline?

- 파이프 라인

  한 데이터 처리 단계의 출력이 다음 단계의 입력으로 이어지는 형태로 연결된 구조

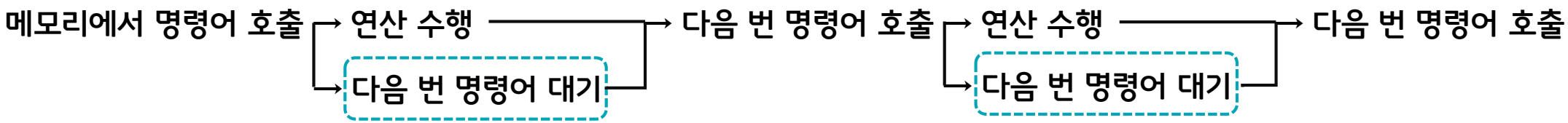
  이 때, 시스템의 효율을 높이기 위해 명령문을 수행하면서 몇 가지의 특수한 작업들을 병렬 처리하도록 설계

- 파이프 라인의 기능
  - 파이프 라인이 없다면?

    메모리에서 명령어 호출 → 연산 수행 → 다음 번 명령어 호출 → 연산 수행

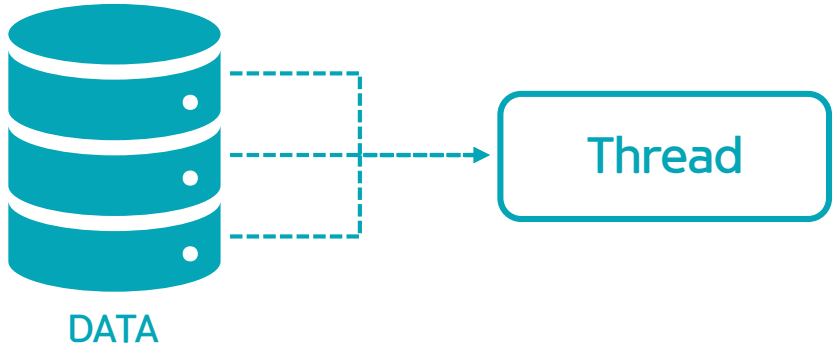
    다음 명령어를 호출하는 동안 산술 연산 부분은 명령어가 도착되기를 기다리며 쉬고 있음

  - 파이프 라인을 사용하면?

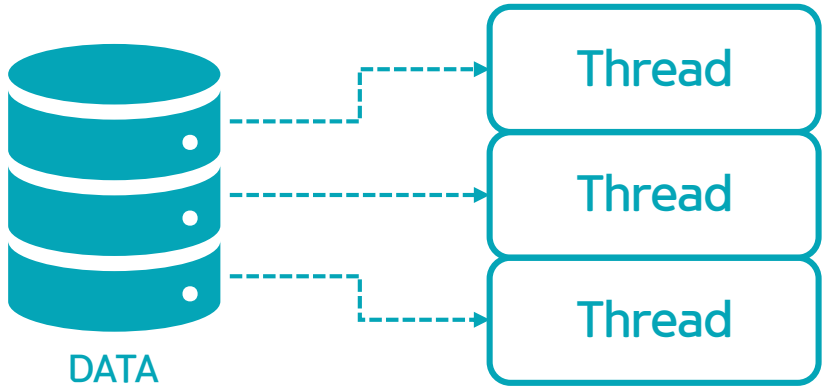


Queues

- TF “Important TensorFlow objects for computing tensors asynchronously in a graph.”
- Input Pipeline에서 Multi Threads는 Data Phase를 읽는 과정에서 병목 현상을 줄여줌



<Single Thread>



<Multi Threads>

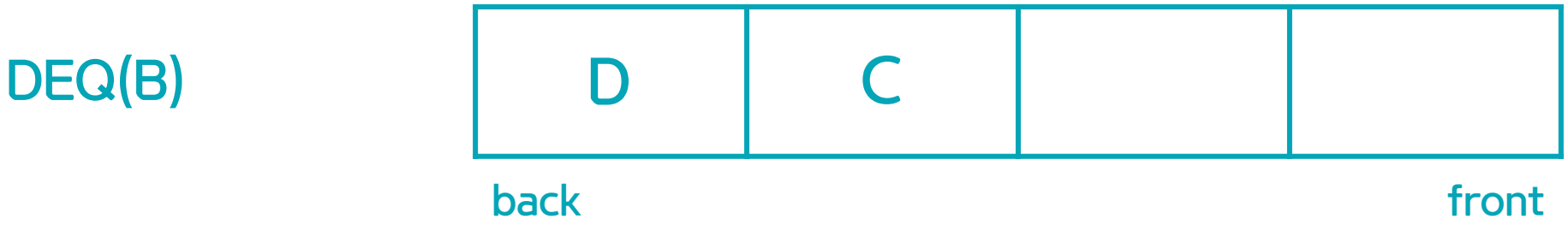
Queues

- Queue?
  - [kju:] 1. (무엇을 기다리는 사람·자동차 등의) 줄
  - 컴퓨터의 기본적인 자료 구조의 한가지
  - 먼저 집어 넣은 데이터가 먼저 나오는 FIFO (First In First Out)구조로 저장하는 형식

    cf) 나중에 집어 넣은 데이터가 먼저 나오는(LIFO) 스택(Stack)과 반대 개념
  - 프린터의 출력 처리나 윈도우 시스템의 메시지 처리기, 프로세스 관리 등
- put(insert)과 get(delete)을 이용하여 구현
- 오버플로우(Overflow) / 언더플로우(Underflow)

Queues

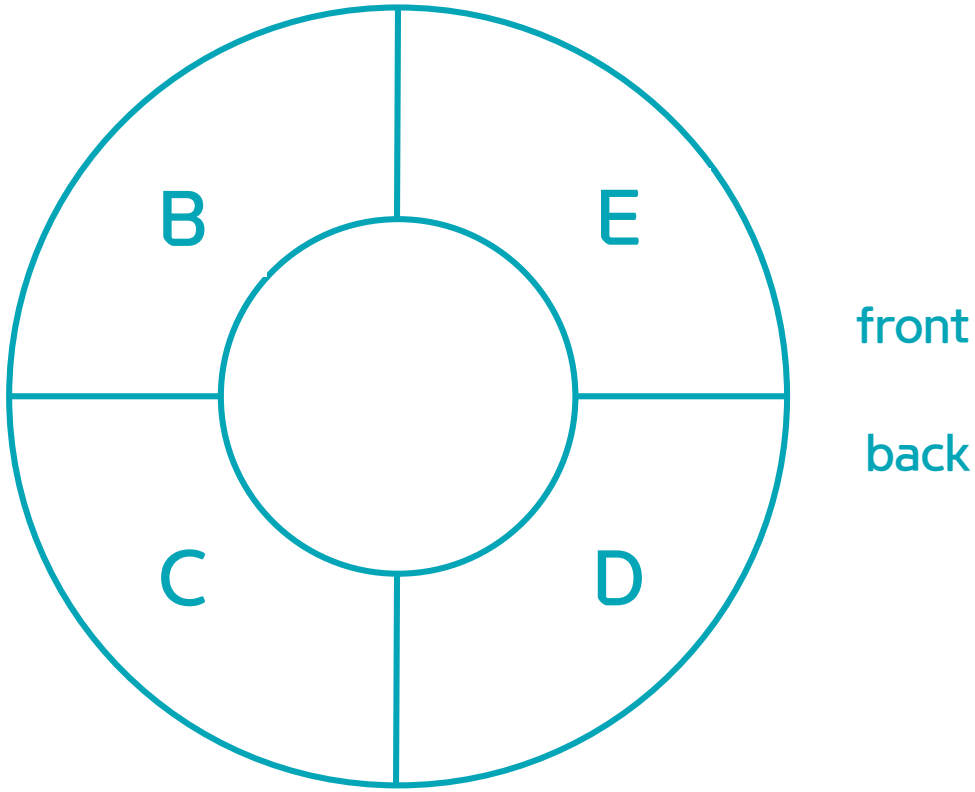
- 선형
  - 막대 모양으로 된 큐
  - 제한된 크기
  - 빈 공간을 사용하려면 모든 자료를 꺼내거나 자료를 한 칸씩 옮겨야 함



Queues

- 환형
  - 선형 큐의 문제점 보안
  - front가 큐의 끝에 닿으면 큐의 맨 앞으로 자료를 보내어 원형으로 연결하는 방식

ENQ(E)



Queues

- 연결 리스트로 구현한 큐 (링크드 큐)
  - 연결 리스트를 사용
  - 큐의 길이를 쉽게 늘릴 수 있음 → 오버플로우(Overflow) 발생 X
  - 필요에 따라 환형으로도 만들 수도 있음
  - 환형으로 만들지 않아도 삽입과 삭제가 제한되지 않아 편리

Queues with TF

- TF “Queues are a powerful mechanism for asynchronous computation using TensorFlow.”
- 예제 코드

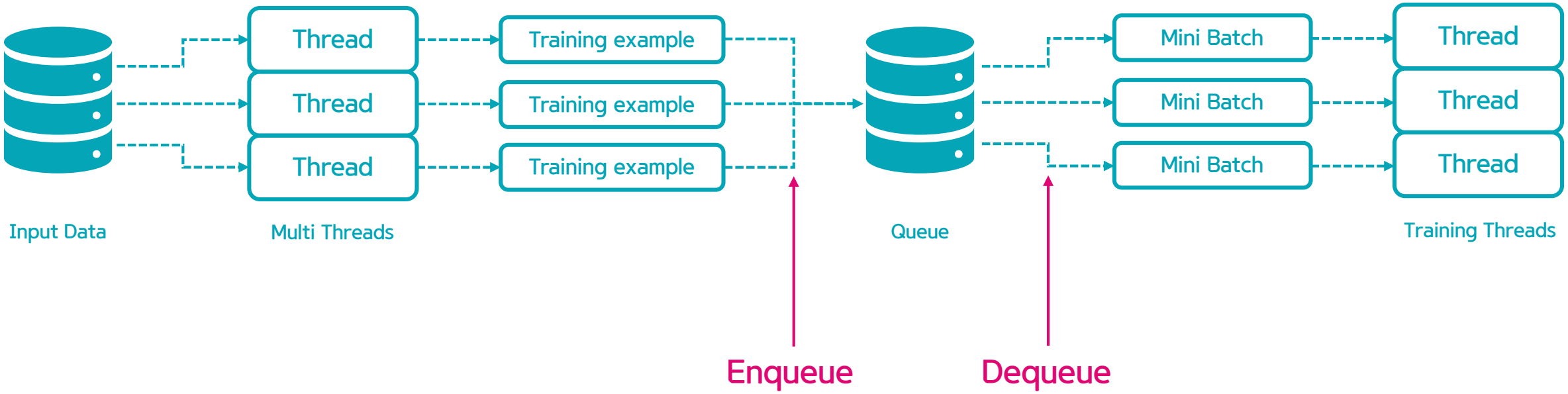

```
q = tf.FIFOQueue(3, "float")
init = q.enqueue_many(([0.,0.,0.],))

x = q.dequeue()
y = x+1
q_inc = q.enqueue([y])

init.run()
q_inc.run()
q_inc.run()
q_inc.run()
q_inc.run()
```

Queues & Threads

- Training Model을 위한 Input을 준비할 때, Queue를 사용하면 얻을 수 있는 결과
  - Multi threads가 Training example을 준비하고 Enqueue
  - Training threads는 Queue에 있는 Mini batch를 Dequeue



Multi Threads with TF

- TensorFlow Session Object는 Multi Threads로 디자인 됨

  → 같은 Session을 쉽게 사용하고, 병렬적으로 실행 가능

- 그러나, Threads를 구동하는 Python program을 구현하는 것이 항상 쉬운 것은 아님

- Threads의 조건
  - 모든 threads들은 항상 함께 멈춰야 함
  - 예외들은 반드시 catch되고, report되어야 함
  - 멈출 때는 queue가 반드시 닫혀야 함

Queues & Threads

- TF document에는 Threading이 Queue를 실행하는데 optional한 것처럼 서술
- 실제로는 Threading이 없으면

  → 프로그램이 Gridlock(하나의 op가 다른 op를 기다림 ≒ Traffic Jam) 에 빠짐 → 프로그램 Crush

- 다행히 TF는 두 개의 class를 제공함으로써 Threading 문제를 도와주고 있음

  → tf.Coordinator & tf.QueueRunner

tf.Coordinator & tf.QueueRunner

- 두 개의 class는 함께 사용되도록 디자인 되었음

- Coordinator class는 multi thread가 함께 멈추고,

  다음 op가 멈추길 기다리는(gridlock에 빠진) 예외를 기록해주는 데 도움을 줌
- QueueRunner class는 같은 queue에 있는 tensor를 enqueue하기 위해

  cooperating하는 많은 thread를 생성

tf.Coordinator

tf.train.Coordinator.should_stop : 스레드를 중지 해야 하는 경우 True를 반환

tf.train.Coordinator.request_stop : 스레드가 멈추도록 요청

tf.train.Coordinator.join : 지정된 스레드가 중지 될 때까지 대기

```
# Thread body: loop until the coordinator indicates a stop was requested.
# If some condition becomes true, ask the coordinator to stop.
def MyLoop(coord):
  while not coord.should_stop():
    ...do something...
    if ...some condition...:
      coord.request_stop()

# Main thread: create a coordinator.
coord = tf.train.Coordinator()

# Create 10 threads that run 'MyLoop()'
threads = [threading.Thread(target=MyLoop, args=(coord,)) for i in  range(10)]

# Start the threads and wait for all of them to stop.
for t in threads:
  t.start()
coord.join(threads)
```

④ should_stop()이 True를 반환 할 때까지 loop 작동

⑤ request_stop() 호출
→ should_stop이 True를 반환 → loop stop

① Coordinator 개체 생성

② Coordinator를 사용하는 스레드 여러 개 생성

③ 실행

tf.FIFOQueue & tf.RandomShuffleQueue

- Queue class의 Main
- FIFOQueue : dequeue element가 FIFO 순서를 갖도록 Queue를 생성
- RandomShuffleQueue : dequeue element가 Random한 순서를 갖도록 Queue를 생성
- 위의 각 Queue들은 enqueue, enqueue_many, dequeue를 지원
- Data를 읽을 때, enqueue할 때는 한꺼번에 많은 예제를 넣을 수 있지만(enqueue_many), dequeue는 one by one으로 하는 것이 보통 (dequeue_many는 허용 X)
- batch를 dequeue할 때 한꺼번에 많은 element를 얻고 싶다면 → tf.train.batch 사용
- random shuffle 된 batch를 얻고 싶다면 → tf.train.shuffle_batch 사용

tf.PaddingFIFOQueue & tf.PriorityQueue

- FIFOQueue를 paddin으로 tensor의 가변성 있는 batching size 일괄 처리를 지원

  ex) NLP 중 seq2seq 모델을 사용할 때 문장을 batch로 넣길 원함 → 문장마다 길이가 같지 않음

  → tf.PaddingFIFOQueue사용

- PaddingFIFOQueue 특징
  - 다양한 shape의 component를 포함
  - dequeue_many 지원

  → 왜 PaddingFIFOQueue에서만 dequeue_many를 지원하는 지 정확한 이유를 모름

  저자가 TF GitHub에 report된 issue를 읽고 난 후 추측으로는 dequeue_many는 FIFOQueue와 RandomShuffleQueue를 지원하는데 사용되었지만, 사용자들이 실행하는데 수 많은 문제들이 생겨서 TF에서 허용하지 않게 만든 것

Create the Queue

- parameter를 사용하여 독립적으로 Queue 생성 가능
  - min_after_dequeue : 사용자가 dequeue한 후 한 queue안에 있는 element의 최소 수
  - 제한된 capacity : 한 queue 안에 들어갈 수 있는 elements의 최대 수
  - element shape : queue안에 있는 element의 shape, 만약 shape이 None이면 어떤 shape이라도 가능

```
tf.RandomShuffleQueue(capacity, min_after_dequeue, dtypes, shapes=None, names=None,
seed=None, shared_name=None, name='random_shuffle_queue')
```

- 실제로는 queue자체를 단독으로 사용하는 경우는 거의 없음 (코드)

Data Reader

- Lec5에서 우리는 Data Reader에 관해서 배웠지만 사용하기 까다롭고, 모호한 문서는 실제로 도움 X
- TF에서 Data를 읽는 3가지 방법
  - 상수를 통하는 것
    - 그래프를 심각하게 부풀릴 가능성 있음
  - Feed Dict를 통하는 것
    - Data load의 흐름 ① 저장소 → 클라이언트 ② 클라이언트 → 작업자
    - 클라이언트와 작업자가 서로 다른 machine에 있을 때 속도가 느려짐
  - Data Reader를 통해 저장소에서 작업자에게 직접 load

Bulit-in Reader

- tf . TextLineReader
  - 가장 다재다능
  - newlines로 구분 된 파일을 읽고 각 호출과 line을 return

    ex) text files , CSV files
- tf . FixedLengthRecordReader
  - 고정된 length로 읽음

    ex) each MNIST file has 28 x 28 pixels , CIFAR - 10 32 x 32 x 3
- tf . WholeFileReader
- tf . TFRecordReader
  - TFRecord 타입의 파일을 읽음

Data Reader Code

- Data Reader를 사용하기 위해서는 첫째로, tf.train.string_input_producer를 통해 읽고자 하는 모든 파일의 이름은 hold할 수 있는 queue를 만들어야 함 (코드)

```
filename_queue = tf.train.string_input_producer(filenames)
reader = tf.TextLineReader(skip_header_lines=1) # skip the first line in the file
key, value = reader.read(filename_queue)

with tf.Session() as sess:
        coord = tf.train.Coordinator()
        threads = tf.train.start_queue_runners(coord=coord)
        print sess.run(key) # data/heart.csv:2
        print sess.run(value) # 144,0.01,4.41,28.61,Absent,55,28.87,2.06,63,1
        coord.request_stop()
        coord.join(threads)
```

Data Reader Code

- 반환된 value는 string tensor

  만약 모델에 feed하기를 원하는 data의 type이 string이어도 좋다면 괜찮지만,

  거의 모든 경우, 사람들은 string data가 feature들의 vector형으로 바뀌는 것은 원함

- 일단, 데이터를 살펴보면...

sbp	tobacco	ldl	adiposity	famhist	typea	obesity	alcohol	age	chd
160	12	5.73	23.11	Present	49	25.3	97.2	52	1
144	0.01	4.41	28.61	Absent	55	28.87	2.06	63	1
118	0.08	3.48	32.28	Present	52	29.14	3.81	46	0
170	7.5	6.41	38.03	Present	51	31.99	24.26	58	1
134	13.6	3.5	27.78	Present	60	25.99	57.34	49	1
132	6.2	6.47	36.21	Present	62	30.77	14.14	45	0
142	4.05	3.38	16.2	Absent	59	20.81	2.62	38	0
114	4.08	4.59	14.6	Present	62	23.11	6.72	58	1
114	0	3.83	19.4	Present	49	24.86	2.49	29	0
132	0	5.8	30.96	Present	69	30.11	0	53	1

- 처음 9개의 columns는 feature와 연관
- 마지막 1개의 column은 label과 연관

Data Reader Code

- Feature들을 vector로 변환해 주기 위해서 TF CSV Decoder를 사용

```
record_defaults = [[1.0] for _ in range(N_FEATURES)]
record_defaults[4] = [''] # make the fifth feature string
record_defaults.append([1])
content = tf.decode_csv(value, record_defaults=record_defaults)
```

- record_defaults = [[1.0], [1.0], [1.0], [1.0], [''], [1.0], [1.0], [1.0], [1.0], [1]]
  - 10개의 elements
  - 4번째 (5번째) element를 제외하고, 모든 element의 타입은 int나 float

    사람들은 label이 int인 것을 좋아하기 때문에 9번째 (10번째) column을 int로 지정해줌

Data Reader Code

```
record_defaults = [[1.0], [1.0], [1.0], [1.0], [''], [1.0], [1.0], [1.0], [1.0], [1]]
```

- Data를 feed하기 전에 모든 종류의 전처리 가능

  우리의 content = 8개의 float, 1개의 string, 1개의 int, 총 10개의 elements로 구성된 list

- string → float
  - Absent는 0, Present는 1
- 9개의 feature → 모델에 feed할 수 있는 tensor

TFRecord

- Binary파일이 다루고 귀찮다고 생각하는 사람도 많겠지만, Binary파일은 매우 유용
  - Disk cache 사용이 나아지도록 함
  - 더 빨리 작동함
  - 다른 타입의 데이터도 저장 가능 (한 장소에 img와 label을 동시에 저장 가능)
- 수 많은 ML Framework처럼 TF도 TFRecord라고 불리는 자신만의 Binary 데이터 포맷을 갖고 있음
- TFRecord는 직렬화 된 tf.train.Example Product Object임
- Image를 TFRecord로 변환 (코드)