LangCon 2021

Parallelformers

빅모델 배포를 향한 여정

고현웅 (TUNiB)

kevin.ko@tunib.ai

Who am I?

안녕하세요, 제 이름은 고현웅입니다. 저는...

"Make NLP easier for Everyone."

사용하기 쉬운 오픈소스를 개발해왔습니다.

Pororo, Kss, Kochat, Openchat, Parallelformers 등 개발

Transformers, DeepSpeed, GPT-Neo 등에 컨트리뷰트

누구에게나 열려있는 커뮤니티를 운영해왔습니다.

대규모(160+명) NLP 논문리뷰 스터디 「집현전」 운영자

대한민국 대표 챗봇 개발자 커뮤니티 「챗봇코리아」 운영진

"Be a creator not a programmer."

연구조직에서 나와 스타트업을 공동창업했습니다.

올해 초까지는 Kakao Brain에서 다양한 자연어처리 연구수행

최근 멋진 멤버들과 TUNiB을 창업해서 오픈도메인 챗봇 개발중

초거대 언어모델 엔지니어링에 관심이 많습니다.

초거대 언어모델은 이제 막 개척되기 시작한 가장 최신의 연구분야

난이도가 높고 인력이 부족한 빅모델 엔지니어링 영역에 도전 중

Why is NLP so hard nowadays?

Why is NLP so hard nowadays?

코퍼스는 많아지고 진입 장벽도 낮아졌는데

자연어처리는 왜 아직도 어려운가?

Why is NLP so hard nowadays?

Language Conference

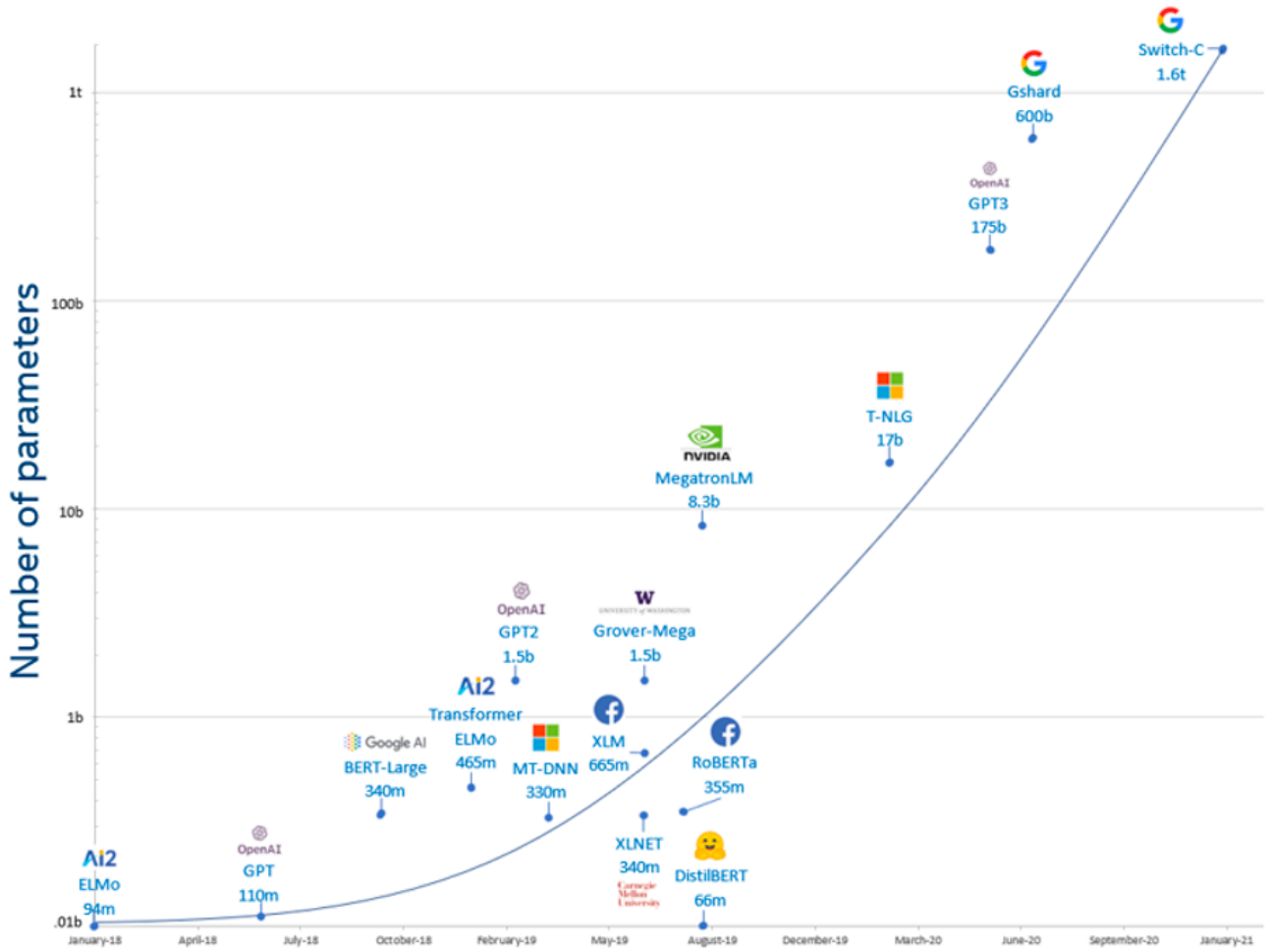


Figure 1: Exponential growth of number of parameters in DL models

요즘 딥러닝 (NLP) 한장 요약

개인적으로는 쉬워지기 보다는 점점 더 어려워지는듯한 느낌.

기술도 기술이지만, GPU 장비의 압박이...

https://towardsdatascience.com/the-rise-of-cognitive-ai-a29d2b724ccc

Why is NLP so hard nowadays?

Language Conference

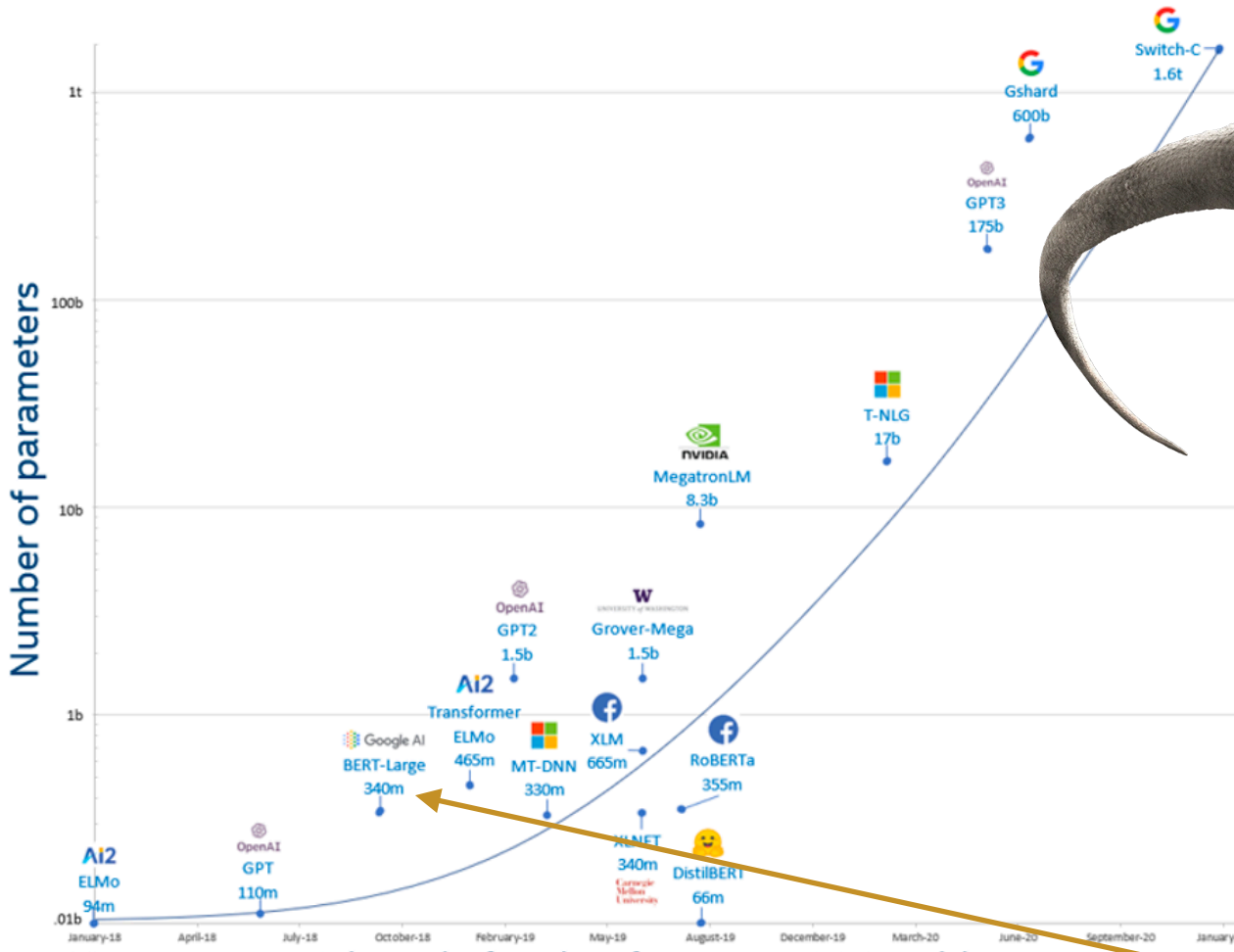


Figure 1: Exponential growth of number of parameters in DL models

GPT-3

;;;

BERT

???: "BERT 같이 작은 모델은..."

2년 전엔 분명 큰 모델이였는데...?

https://towardsdatascience.com/the-rise-of-cognitive-ai-a29d2b724ccc

Why is NLP so hard nowadays?

Language Conference

1. 모델의 아키텍처 같은 것들이 생각보다 별로 중요하지 않더라.

(복잡한 구조가 필요 없고, 그동안의 변화가 생각보다 큰 의미가 없었다.)

Pay Attention to MLPs

Hanxiao Liu, Zihang Dai, David R
Google Research, Brain T
{hanxiaol,zihangd,davidso,qv

MLP-Mixer: An all-MLP Architecture for Vision

Ilya Tolstikhin*, Neil Houlsby*, Alexander Kolesnikov*, Lucas Beyer*,
Xiaohua Zhai, Thomas Unterthiner, Jessica Yung, Andreas Steiner,
Jakob Uszkoreit, Mario Lucic, Alexey Dosovitskiy
*equal contribution
Google Research, Brain Team

Do Transformer Modifications Transfer Across Implementations and Applications?

Sharan Narang* Hyung Won Chung Yi Tay William Fedus
Thibault Fevry† Michael Matena † Karishma Malkan† Noah Fiedel
Noam Shazeer Zhenzhong Lan† Yanqi Zhou Wei Li
Nan Ding Jake Marcus Adam Roberts Colin Raffel

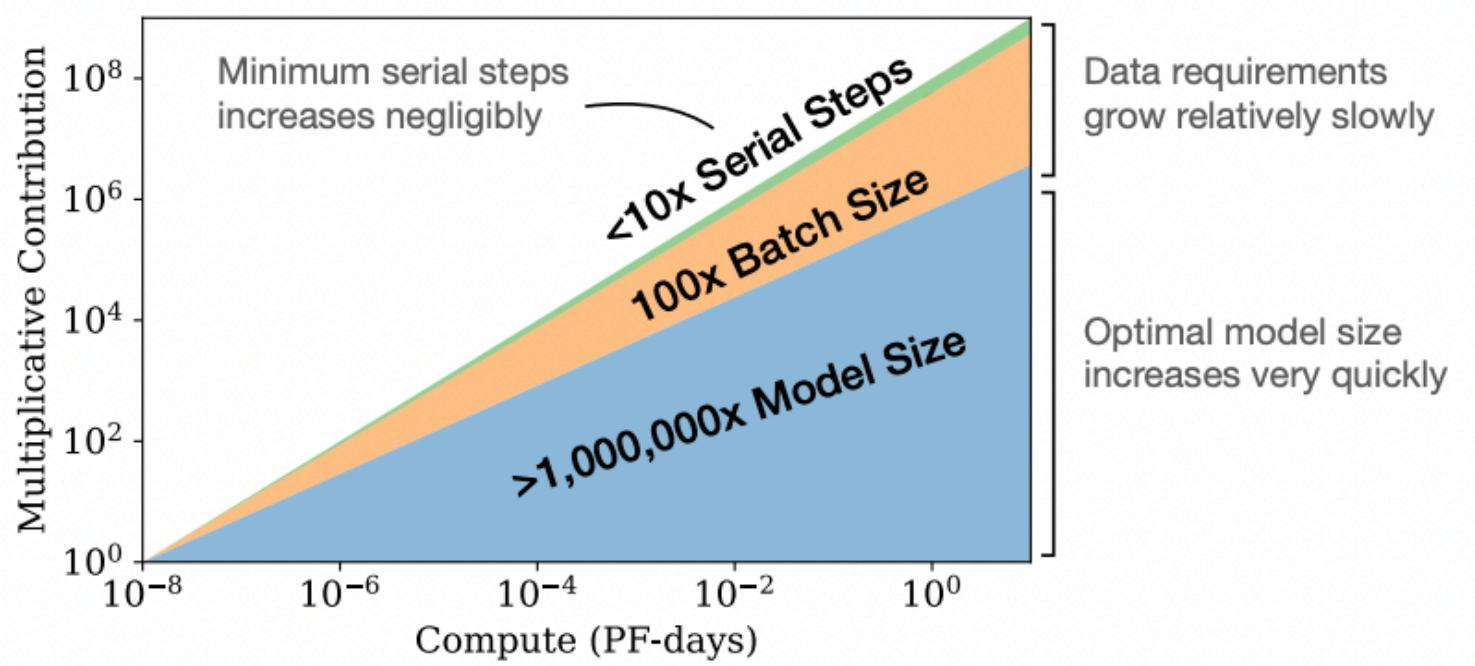
Google Research

Pay Attention to MLPs (https://arxiv.org/abs/2105.08050) / MLP-Mixer (https://arxiv.org/abs/2105.01601)
Do Transformer Modifications Transfer Across Implemenations and Applications? (https://arxiv.org/abs/2102.11972)

Why is NLP so hard nowadays?

2. 결국 관건은 모델과 데이터의 크기, 그들이 곧 성능과 비례한다.

(이전에도 중요한건 알았지만 엄청 키우니까 마법같은 일이 일어나더라. ~~왜 잘되는지는 모른다고 카더라...~~)



Scaling Laws for Neural Language Models (https://arxiv.org/abs/2001.08361)

Why is NLP so hard nowadays?

이대로 간다면, 아마도 몇년 뒤?

깨갱...

GPT-3

???: "GPT-3 같이 작은 모델은..."

GPT-? (가칭)

Why is NLP so hard nowadays?

남들이 바라보는 Large-scale LM

GPT-2

GPT-3

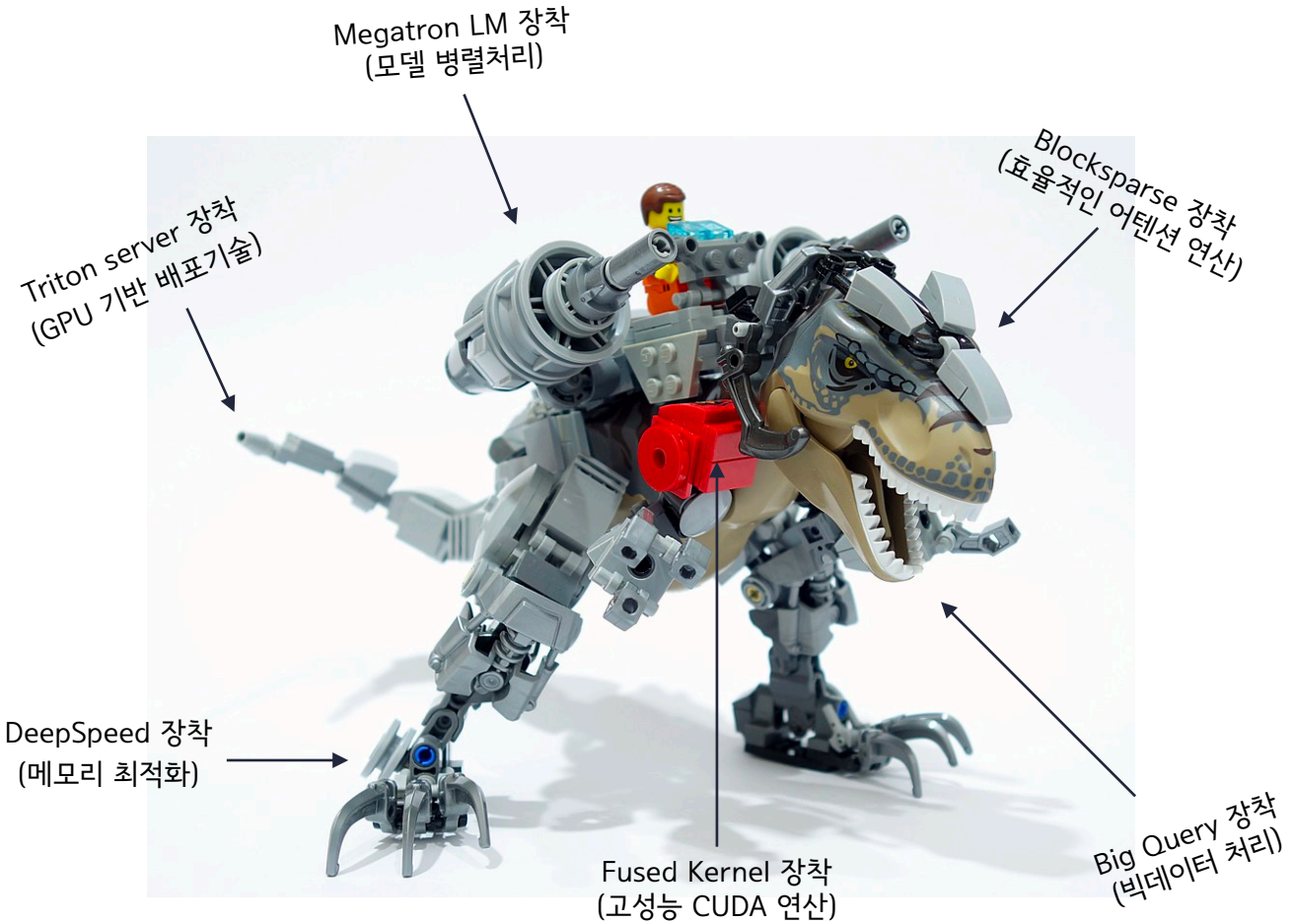
에이... 예전이랑 똑같은데 크기만 키웠네~

Why is NLP so hard nowadays?

Language Conference

하지만 현실은...?

GPT-2



GPT-3

실제로 사용하려면 답도 없는 하드코어 엔지니어링이 필요함...

(~~개발중에 모르는게 생겼다구요? 괜찮아요! 어차피 주변에 물어봐도 아무도 모릅니다!~~)

Why is NLP so hard nowadays?

Language Model의 성능은 모델과 데이터의 크기에 비례한다.

따라서 큰 모델과 큰 데이터를 잘 다룰 수 있어야 한다.

Why is NLP so hard nowadays?

Language Model의 성능은 모델과 데이터의 크기에 비례한다.

따라서 큰 모델과 큰 데이터를 잘 다룰 수 있어야 한다.

이러한 추세가 요즘의 NLP를 더욱 더 어렵게 만들고 있다.

고가의 장비도, 고급 엔지니어링 스킬도 갖추기 어렵기 때문

Why is NLP so hard nowadays?

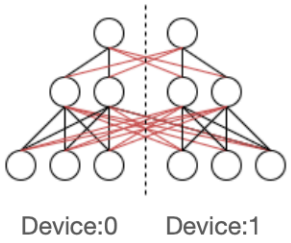
Parallelformers

그래서 만들었습니다.

Design Principles

Design Principles

Parallelformers의 4가지 Design Principles !



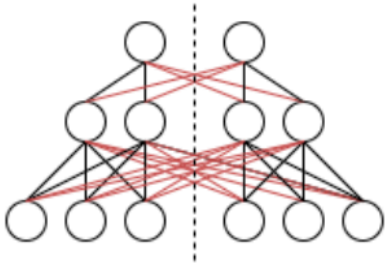
efficient
model parallelism

Deployment

Scalability

Simplicity

Design Principles



efficient
model parallelism

Parallelformers 효율적인 모델 병렬화 도구.

여러대의 GPU에 모델을 쪼개서 올리기 위해 사용합니다.

Deployment

Parallelformers 추론과 배포를 위해 개발된 도구.

(현재는 Huggingface와 협력하여 학습 기능도 개발하고 있습니다.)

Design Principles

Scalability

Parallelformers 확장성이 뛰어난 도구입니다.

Huggingface Transformers에 존재하는 70개의 모델 중 68개 모델에 대한 병렬화를 지원합니다.

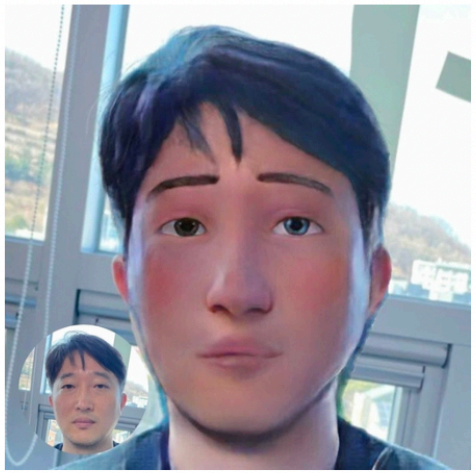
Simplicity

Parallelformers 사용하기 쉬운 도구입니다.

코드 한줄만으로 모델을 병렬화 할 수 있습니다.

Motivation

Motivation



ryan.ai
(Kyubyong Park)

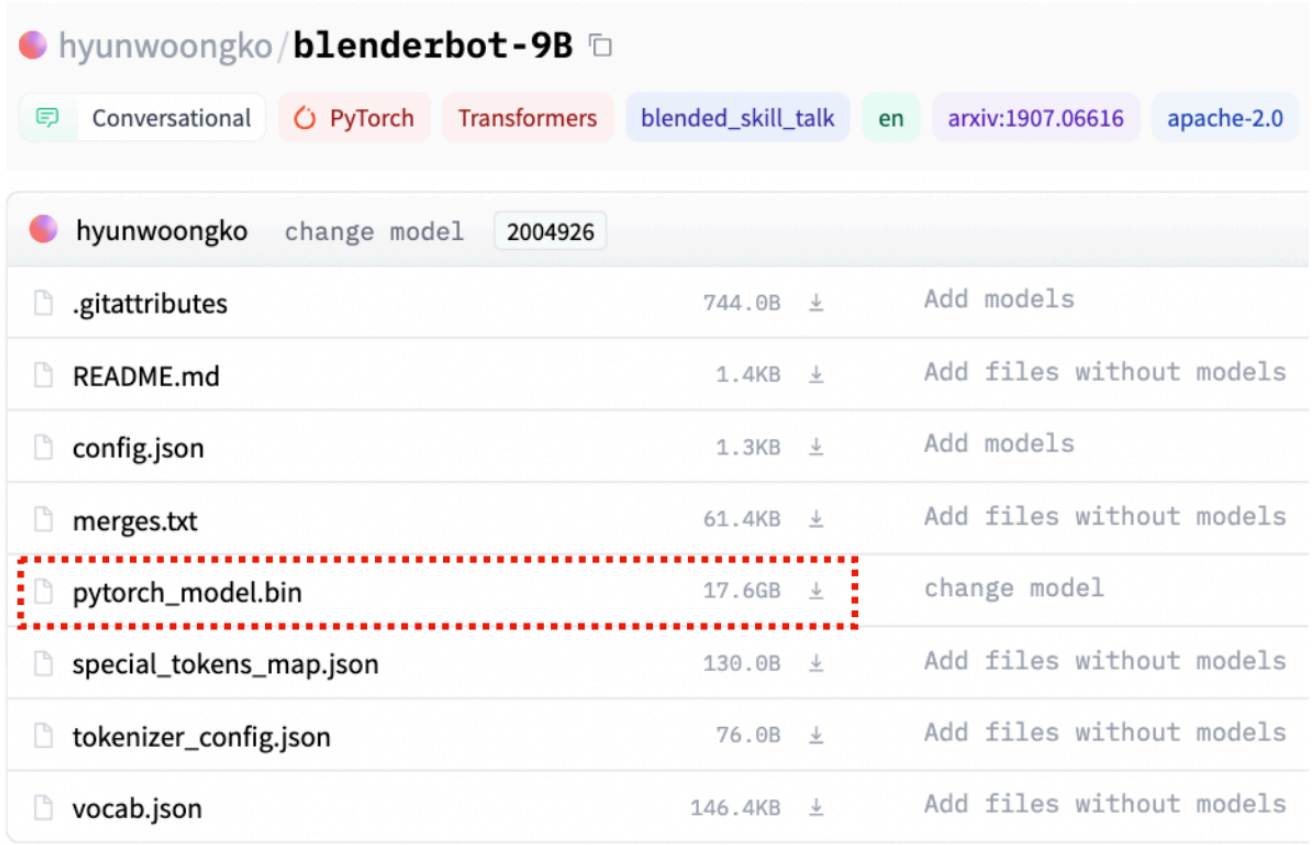
Blenderbot-9B, GPTNeo-3B 등이 요즘 가장 좋다고 알려져 있어요. 우리가 좋은 대화모델을 만들려면 기존 모델들을 사용해보고 분석해보는 시간을 가져야 할 것 같아요.

(별일 아니라는 듯이) 얼른 웹서버에 한번 배포해보세요.

알겠슴다~ (이때까지만 해도 진짜 별일 아닐줄 알았음 😂)

kevin.ko
(Hyunwoong Ko)

Motivation



순수 모델의 용량만 17.6GB

실험결과 웹서버에 배포하려면 22GB 이상의 용량이 필요.

사내에서 사용 중인 Google Cloud Platform (GCP)에서 22GB 이상의 커다란 VRAM 용량을 가진건 A100 (40GB)뿐.
배포하고자 했던 모델들(GPT-Neo, Blenderbot 등)을 모두 배포하려면 비싼 A100 3~4장이 필요. 🤔

시간당 13,500원, 하루에 324,000원, 한달에 약 1,000만원

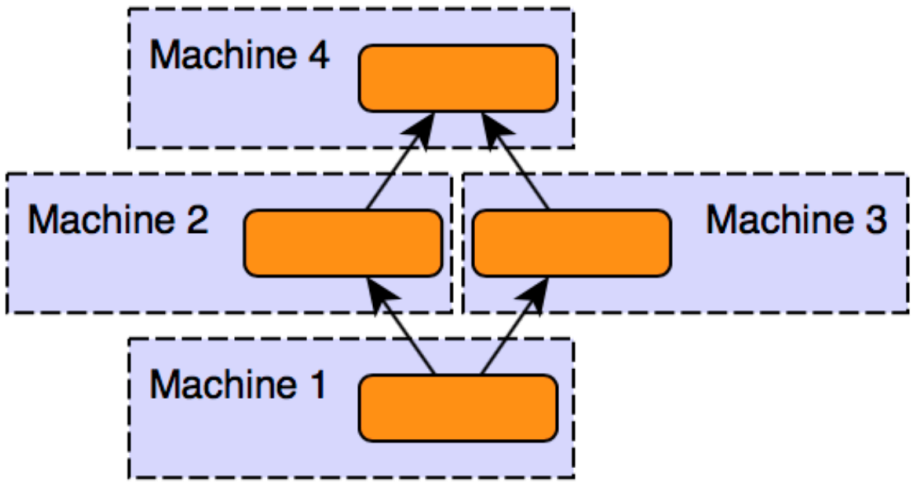
그러나 크기가 작은 GPU 여러대는 크기가 큰 GPU 1대에 비해서 동일용량 대비 훨씬 저렴함.

A100 3대 (120GB)가 아니라 T4 8대 (120GB)를 이용하면 배포에 필요한 비용을 크게 줄일 수 있음. 🤌

한달에 약 1,000만원 → 300만원으로 절약 가능!

Background

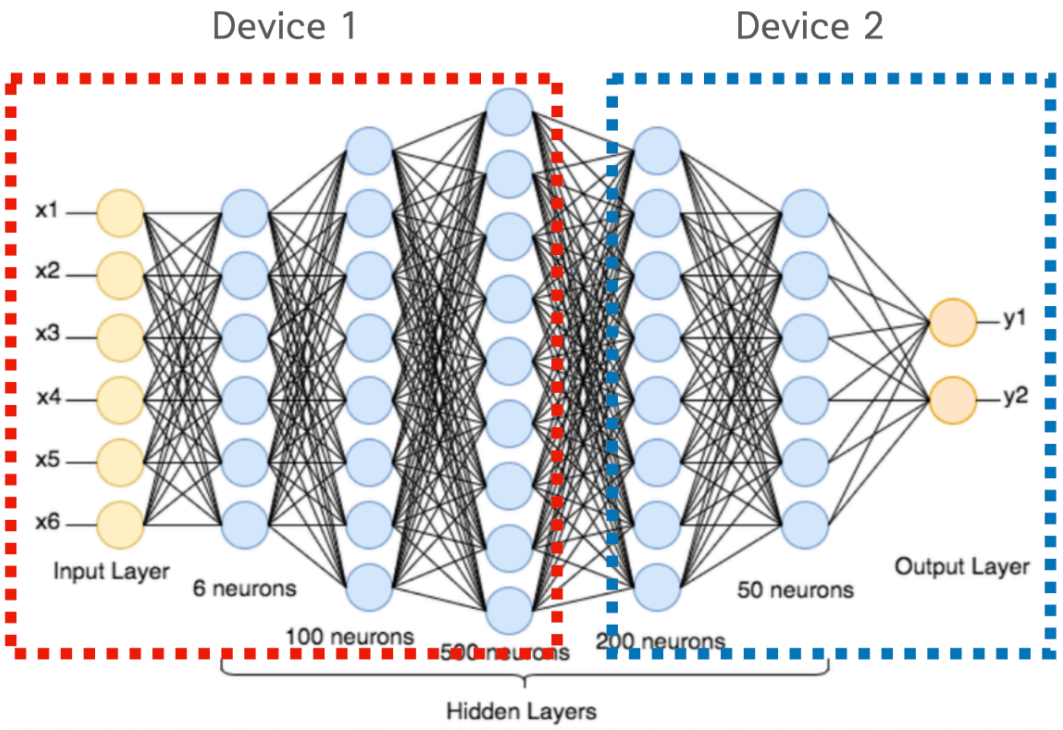
모델 병렬처리: 모델의 각 부분을 서로 다른 GPU에 올려놓고 처리하는 방법



https://xiandong79.github.io/Intro-Distributed-Deep-Learning

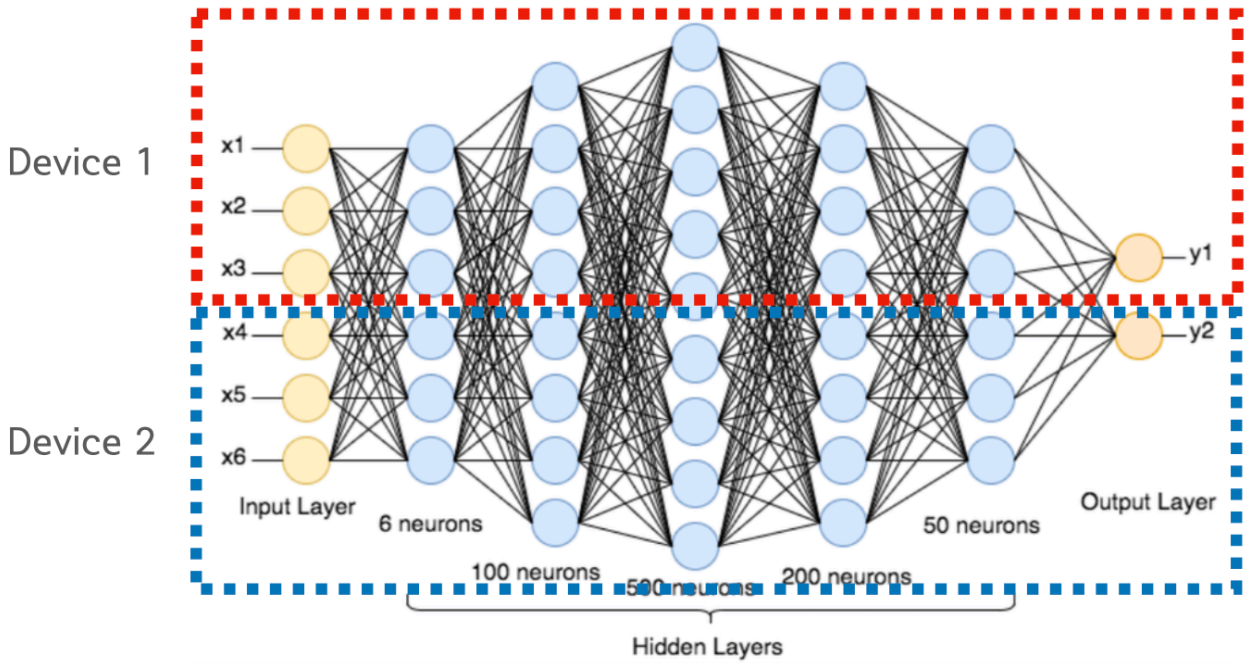
Inter-layer 병렬처리: 신경망의 특정 층을 특정 GPU에 올려놓는 방식

e.g. GPipe, PipeDream ... 등이 있음



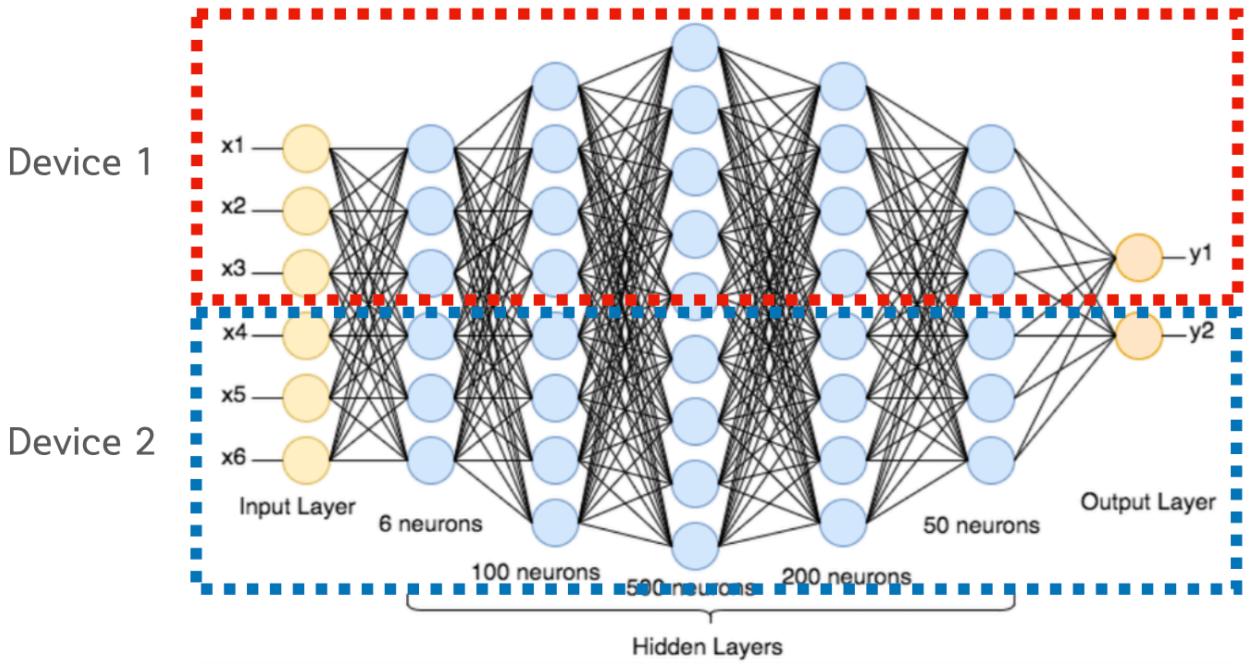
Intra-layer 병렬처리: 신경망 내의 텐서를 쪼개서 올려놓는 방식

e.g. Megatron-LM ... 등이 있음

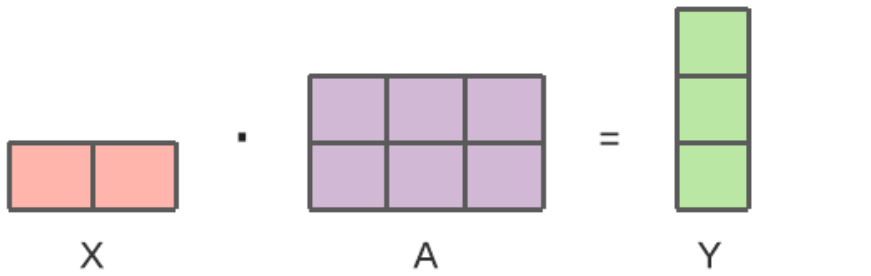


Intra-layer 병렬처리: 신경망 내의 텐서를 쪼개서 올려놓는 방식

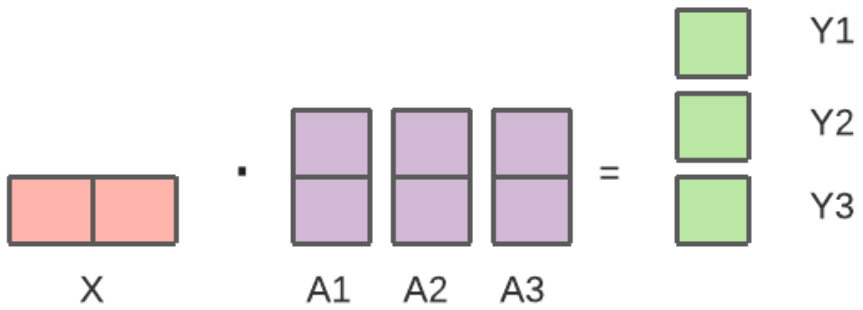
Parallelformers가 모델을 병렬화 하는 방식 !



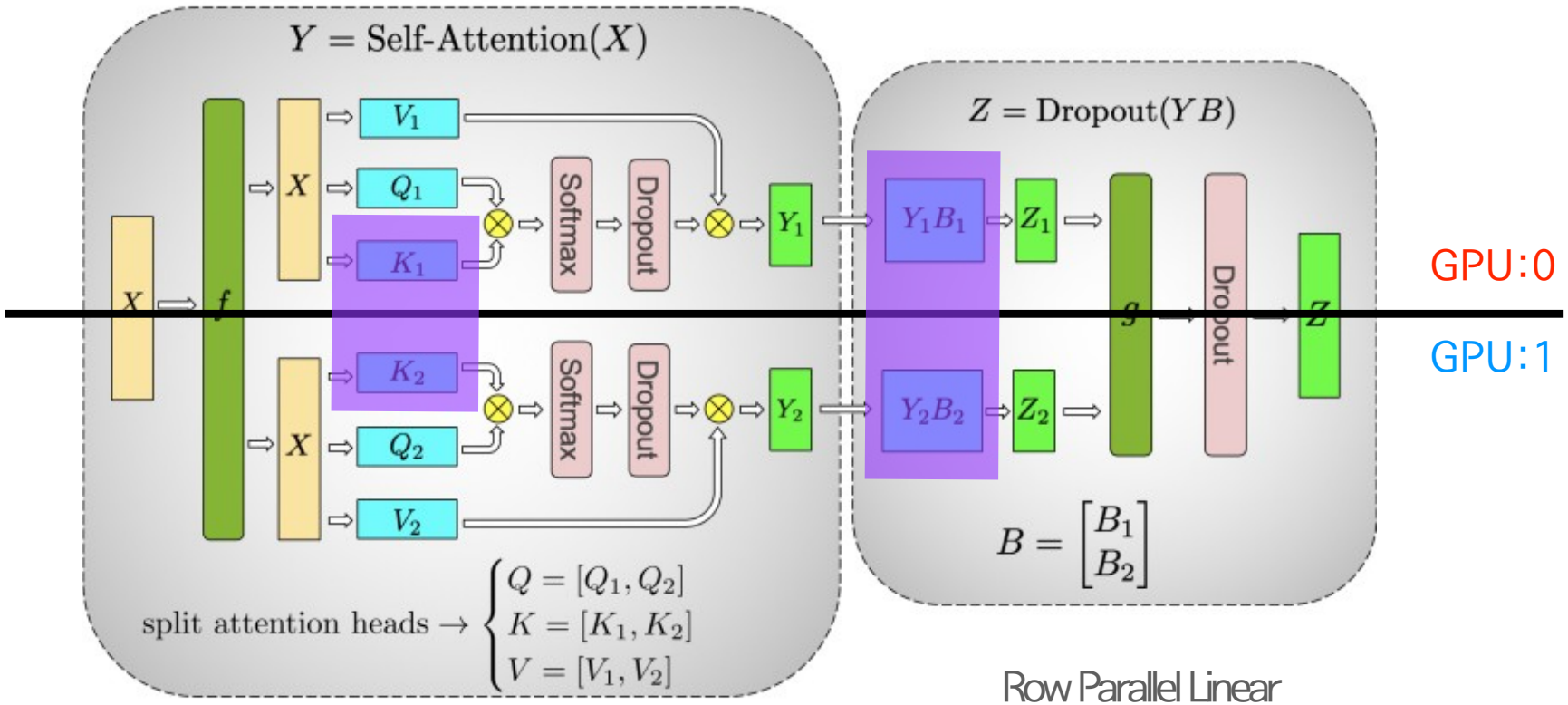
Parallelformers는 Megatron-LM의 병렬처리 기법을 이용했어요!



is equivalent to



https://huggingface.co/transformers/parallelism.html#tensor-parallelism



Column Parallel Linear

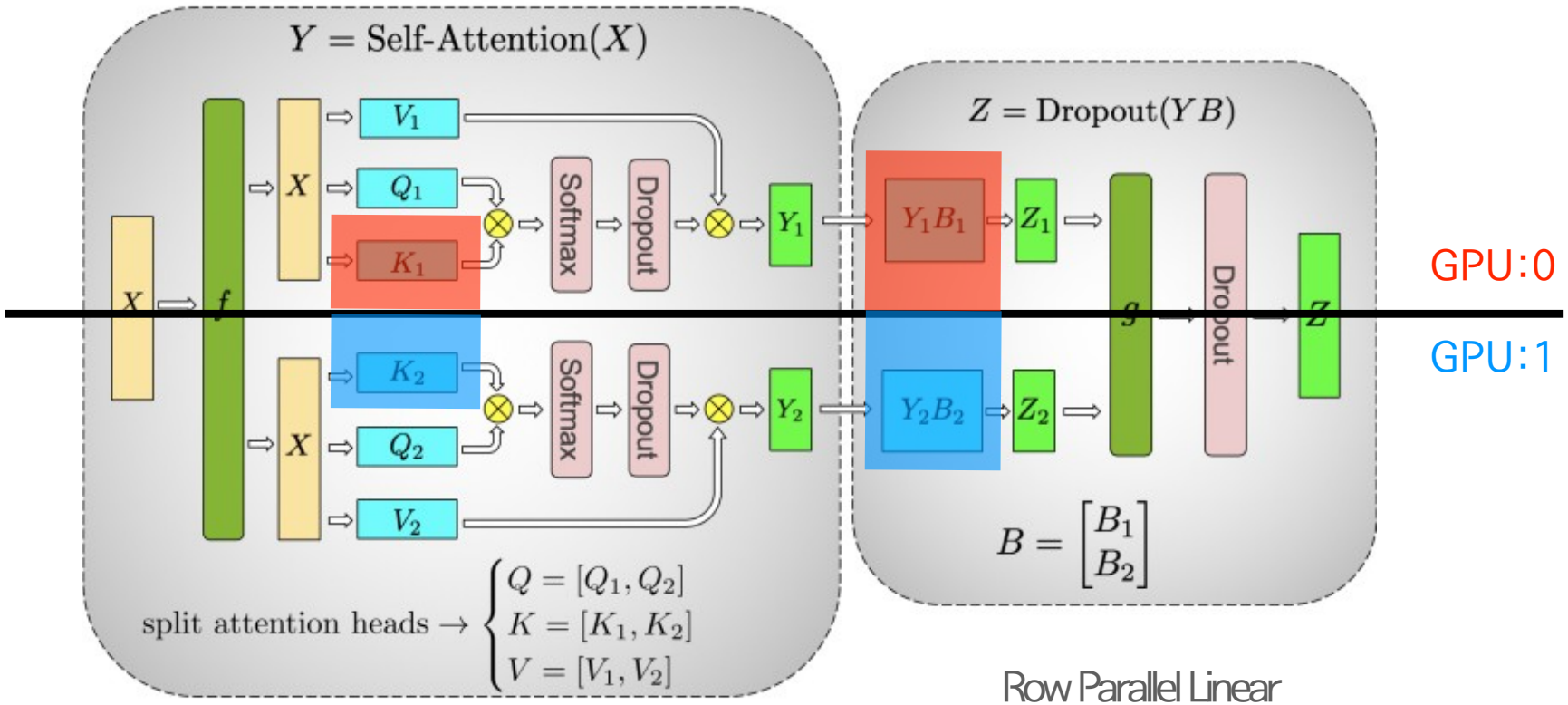
[512, 512] → [512, 256], [512, 256]

Row Parallel Linear

[512, 512] → [256, 512], [256, 512]

Megatron-LM: Training Multi-Billion Parameter Language Models Using Model Parallelism (https://arxiv.org/abs/1909.08053)

Background



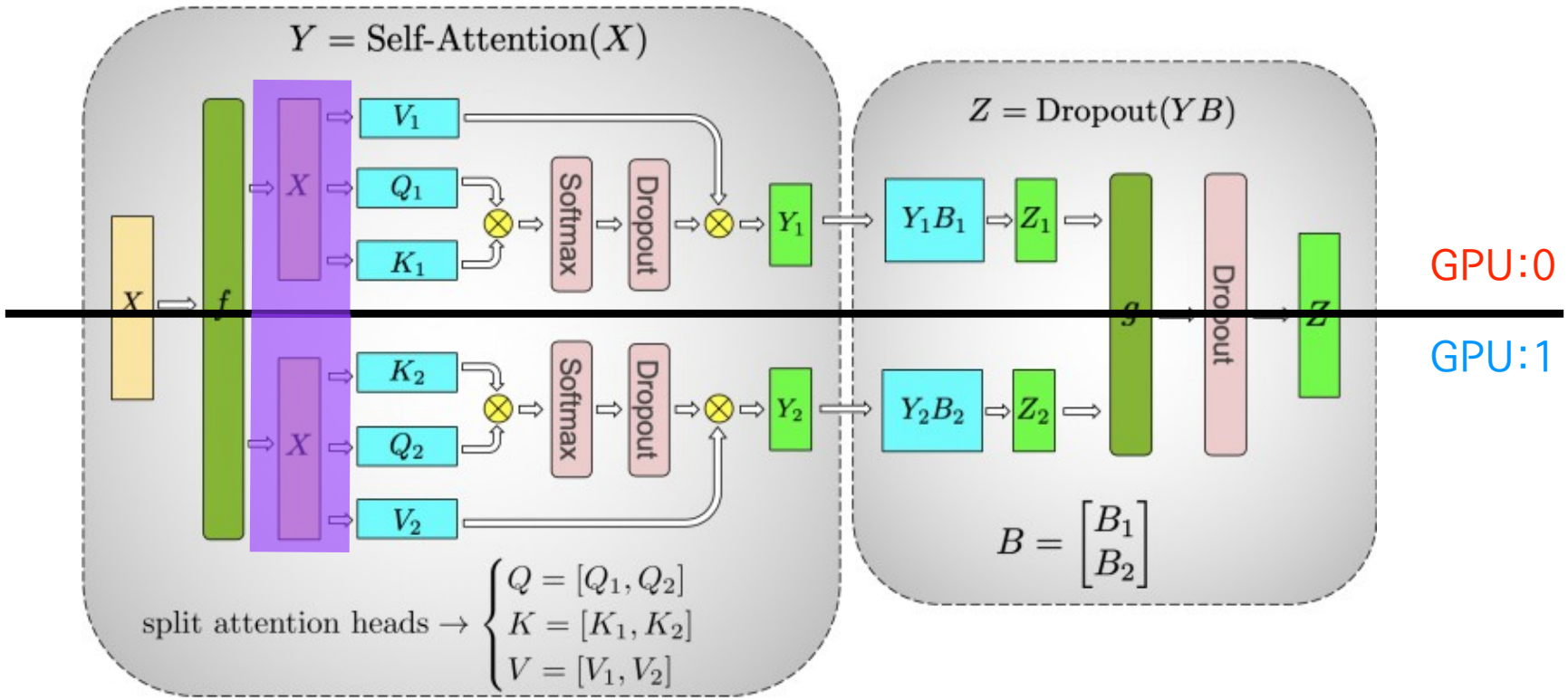
Column Parallel Linear

[512, 512] → [512, 256], [512, 256]

Row Parallel Linear

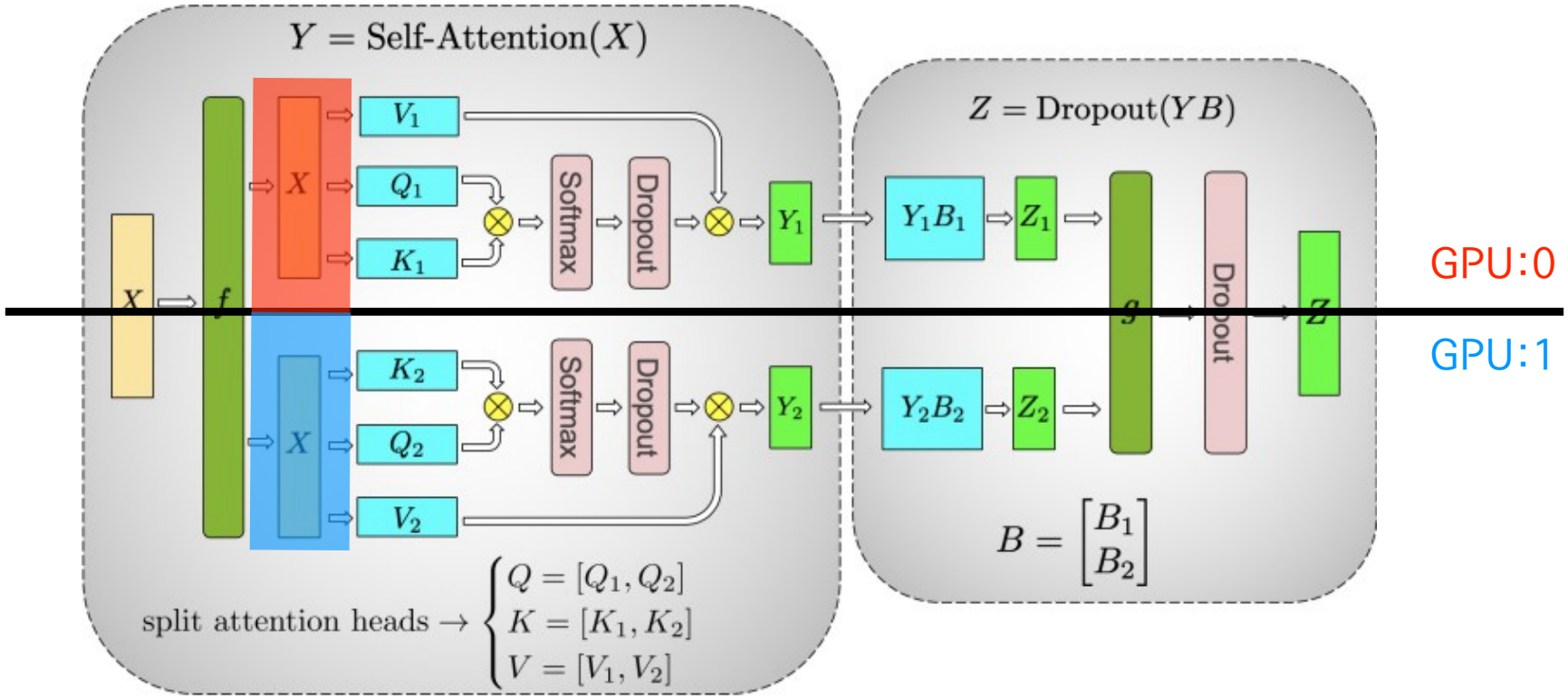
[512, 512] → [256, 512], [256, 512]

Megatron-LM: Training Multi-Billion Parameter Language Models Using Model Parallelism (https://arxiv.org/abs/1909.08053)



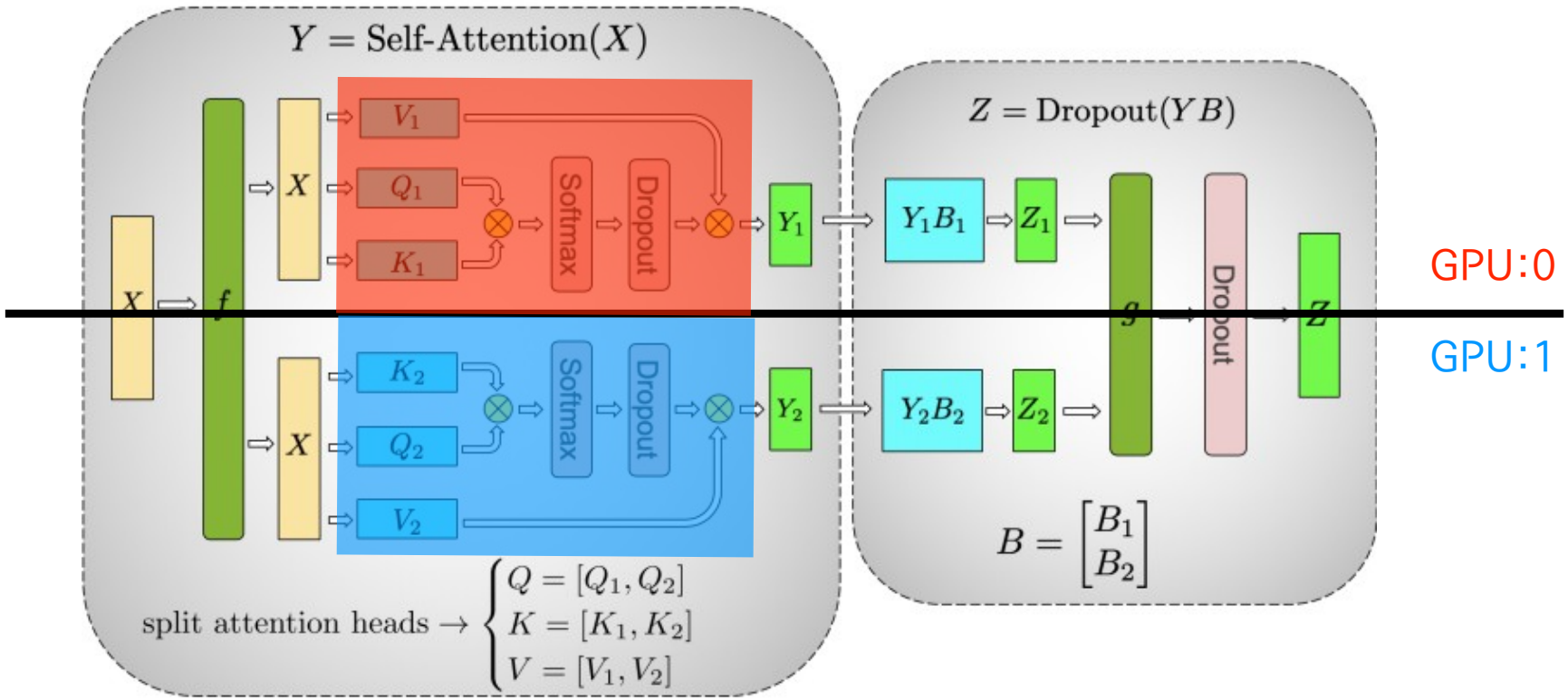
동일한 입력을 복사하여
각 디바이스로 복사

Megatron-LM: Training Multi-Billion Parameter Language Models Using Model Parallelism (https://arxiv.org/abs/1909.08053)



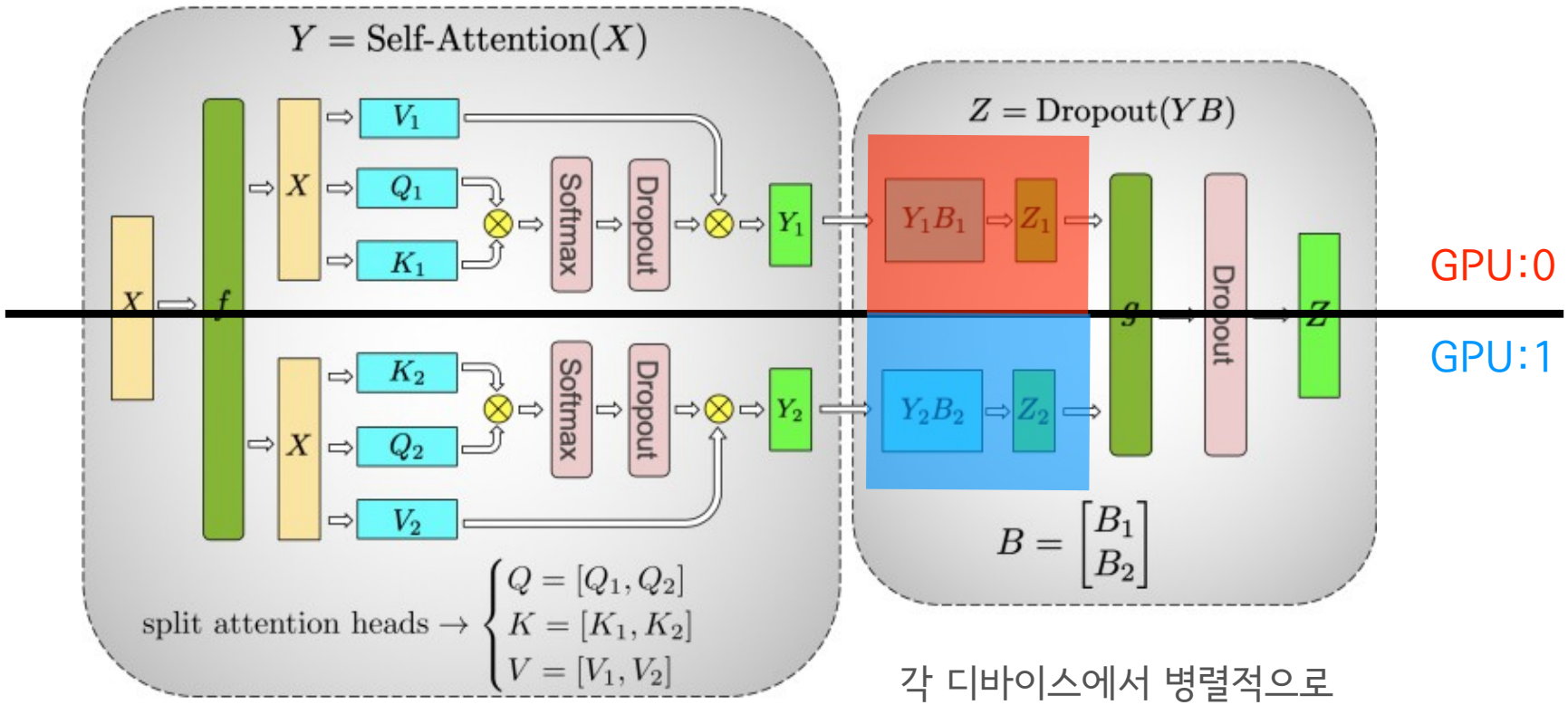
동일한 입력을 복사하여
각 디바이스로 복사

Megatron-LM: Training Multi-Billion Parameter Language Models Using Model Parallelism (https://arxiv.org/abs/1909.08053)



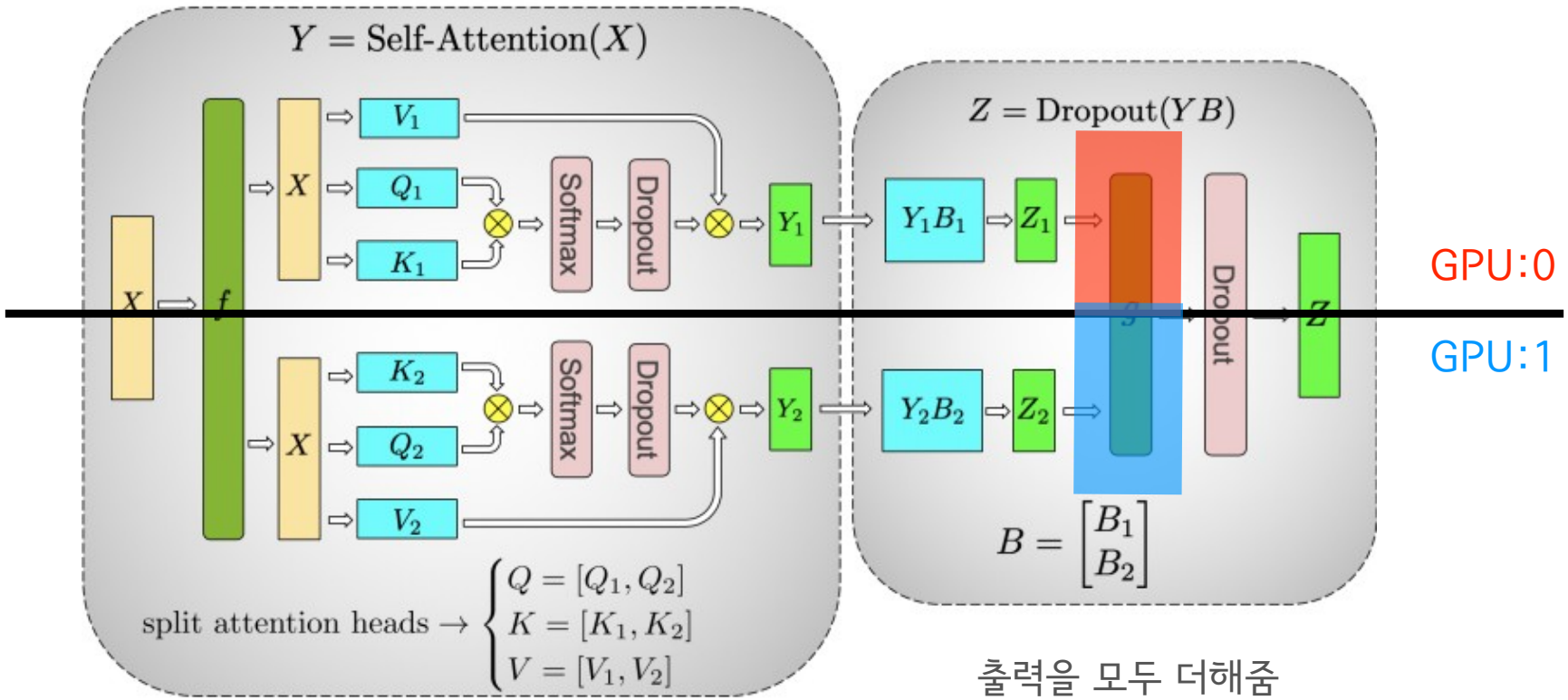
각 디바이스에서 병렬적으로
멀티헤드 어텐션 수행

Megatron-LM: Training Multi-Billion Parameter Language Models Using Model Parallelism (https://arxiv.org/abs/1909.08053)



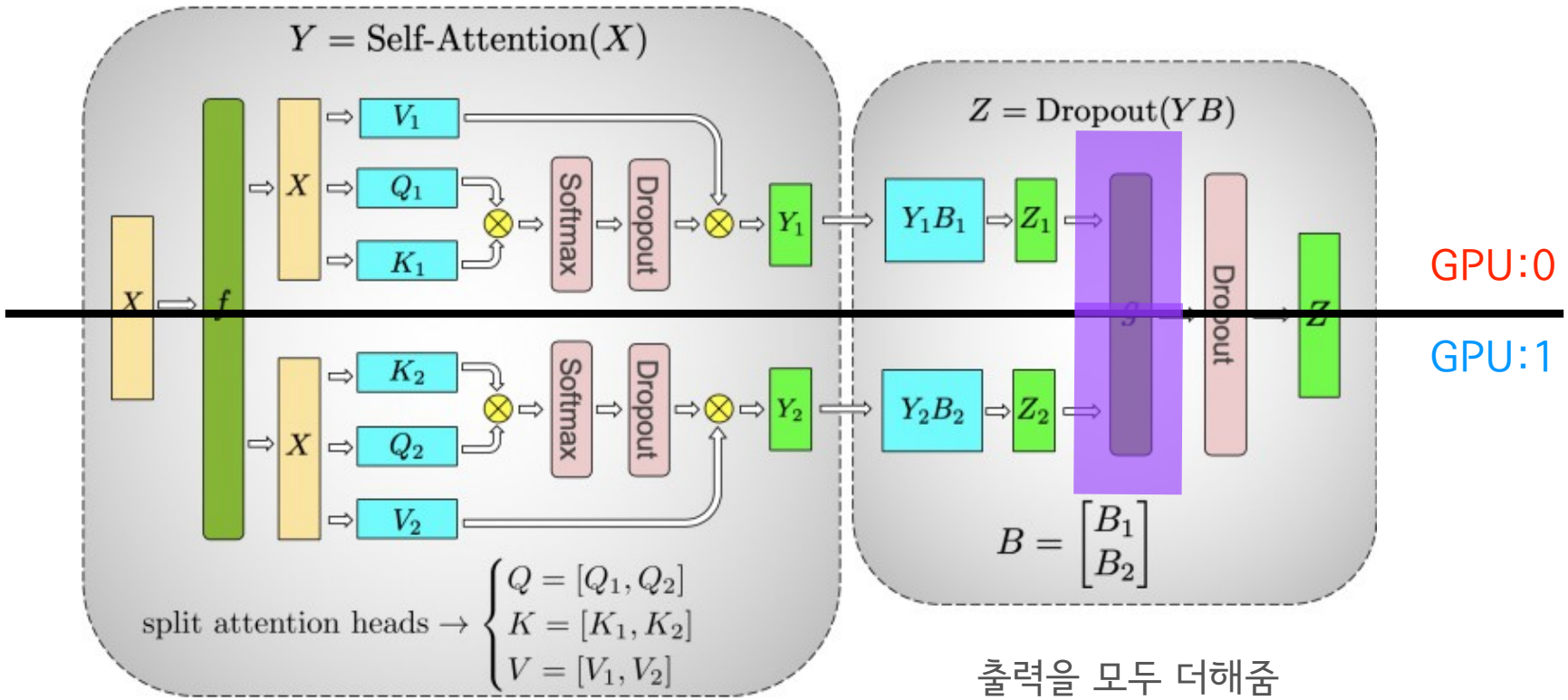
각 디바이스에서 병렬적으로
멀티헤드 어텐션 수행

Megatron-LM: Training Multi-Billion Parameter Language Models Using Model Parallelism (https://arxiv.org/abs/1909.08053)



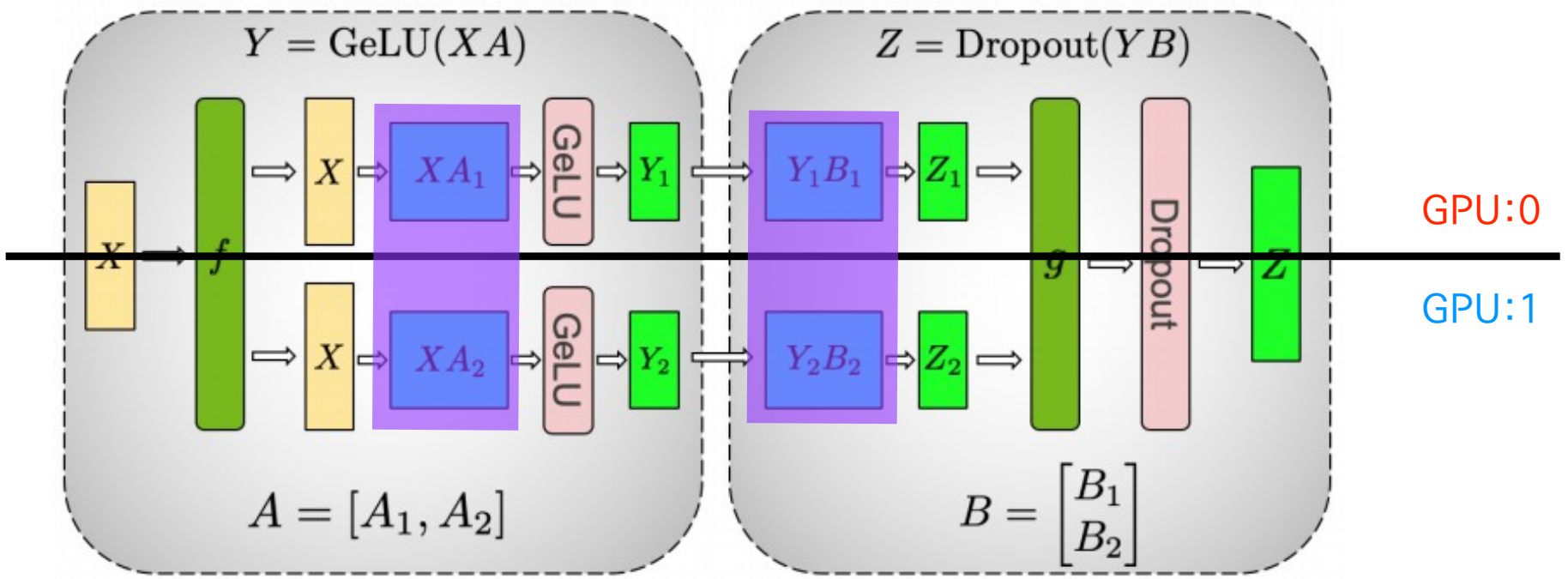
출력을 모두 더해줌

Megatron-LM: Training Multi-Billion Parameter Language Models Using Model Parallelism (https://arxiv.org/abs/1909.08053)



출력을 모두 더해줌

Megatron-LM: Training Multi-Billion Parameter Language Models Using Model Parallelism (https://arxiv.org/abs/1909.08053)



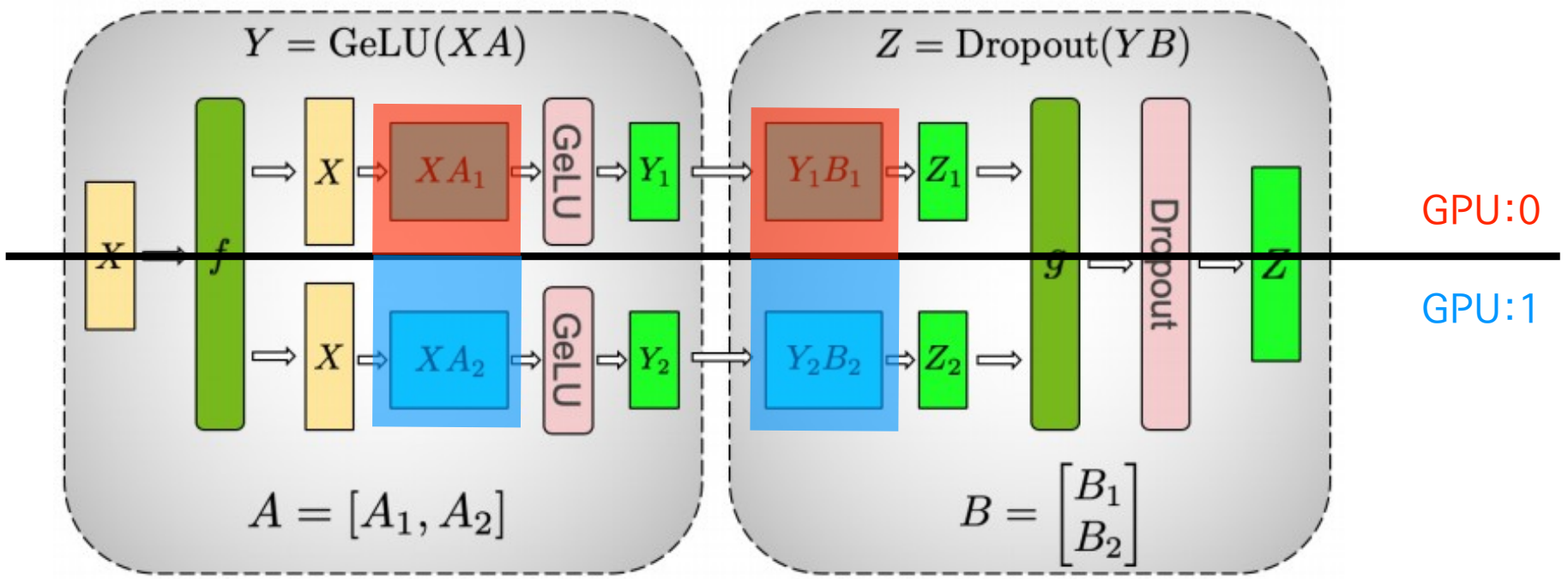
Column Parallel Linear

[512, 512] → [512, 256], [512, 256]

Row Parallel Linear

[512, 512] → [256, 512], [256, 512]

Megatron-LM: Training Multi-Billion Parameter Language Models Using Model Parallelism (https://arxiv.org/abs/1909.08053)



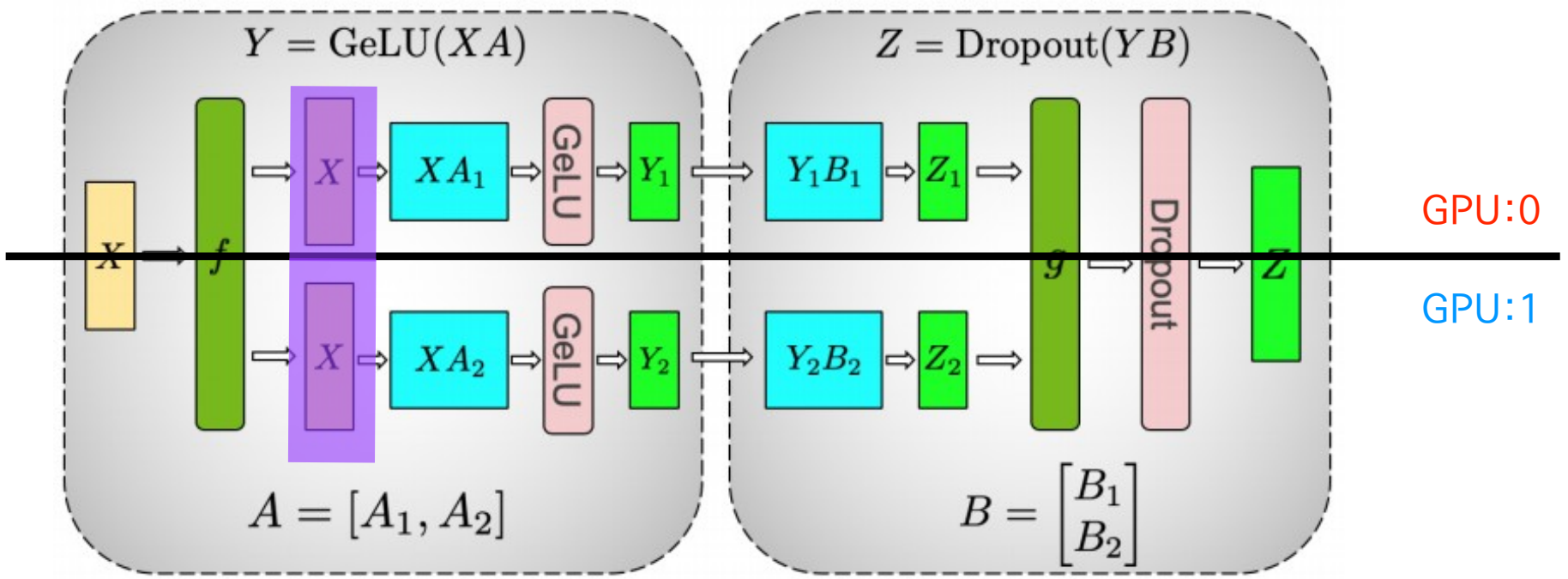
Column Parallel Linear

[512, 512] → [512, 256], [512, 256]

Row Parallel Linear

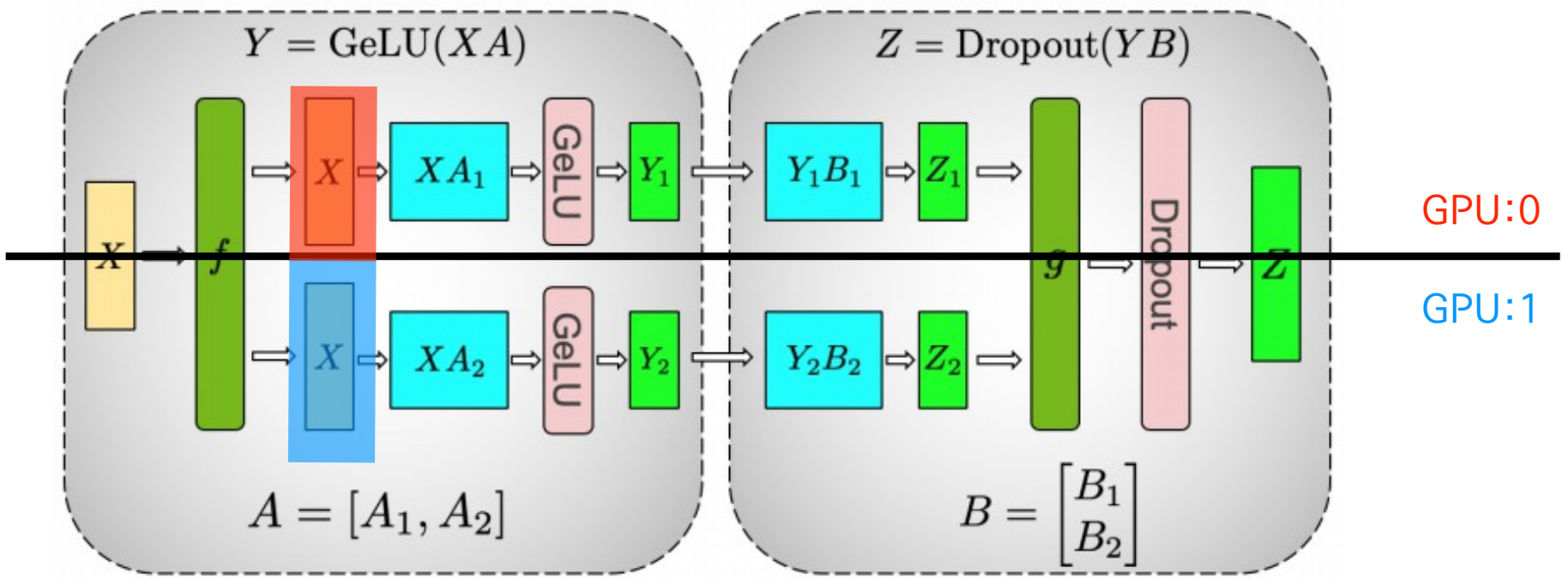
[512, 512] → [256, 512], [256, 512]

Megatron-LM: Training Multi-Billion Parameter Language Models Using Model Parallelism (https://arxiv.org/abs/1909.08053)



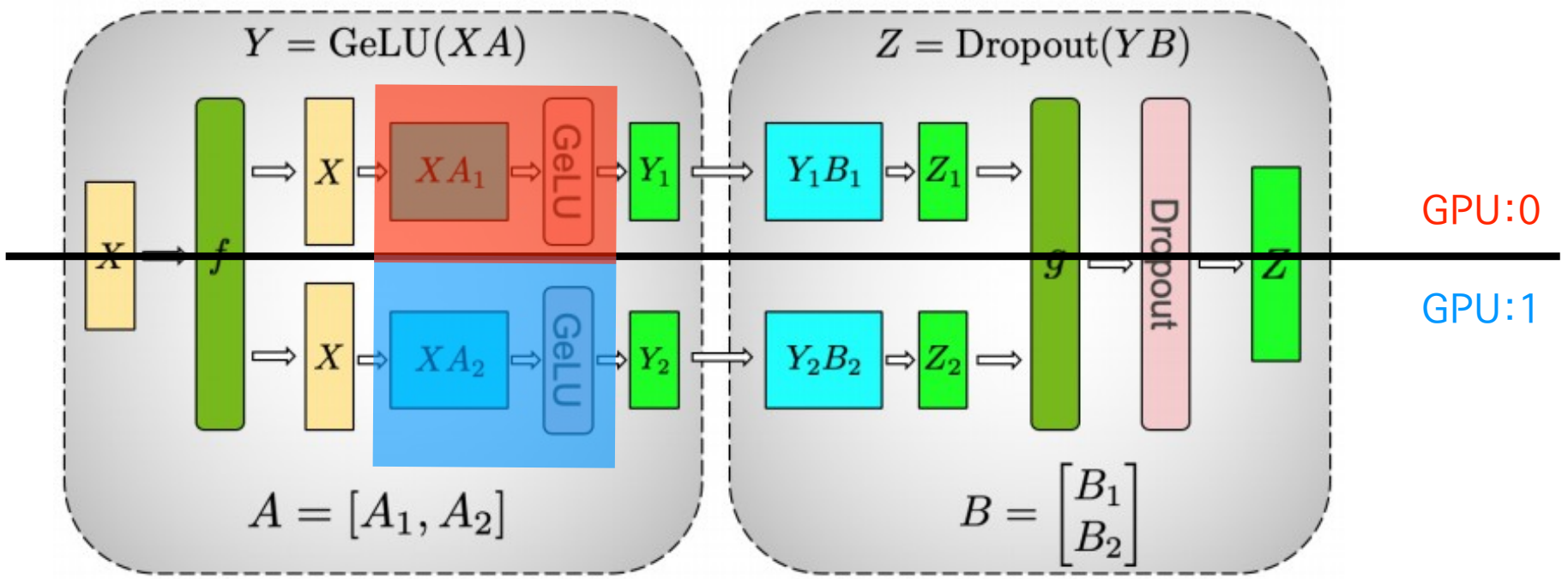
동일한 입력을 복사하여
각 디바이스로 복사

Megatron-LM: Training Multi-Billion Parameter Language Models Using Model Parallelism (https://arxiv.org/abs/1909.08053)



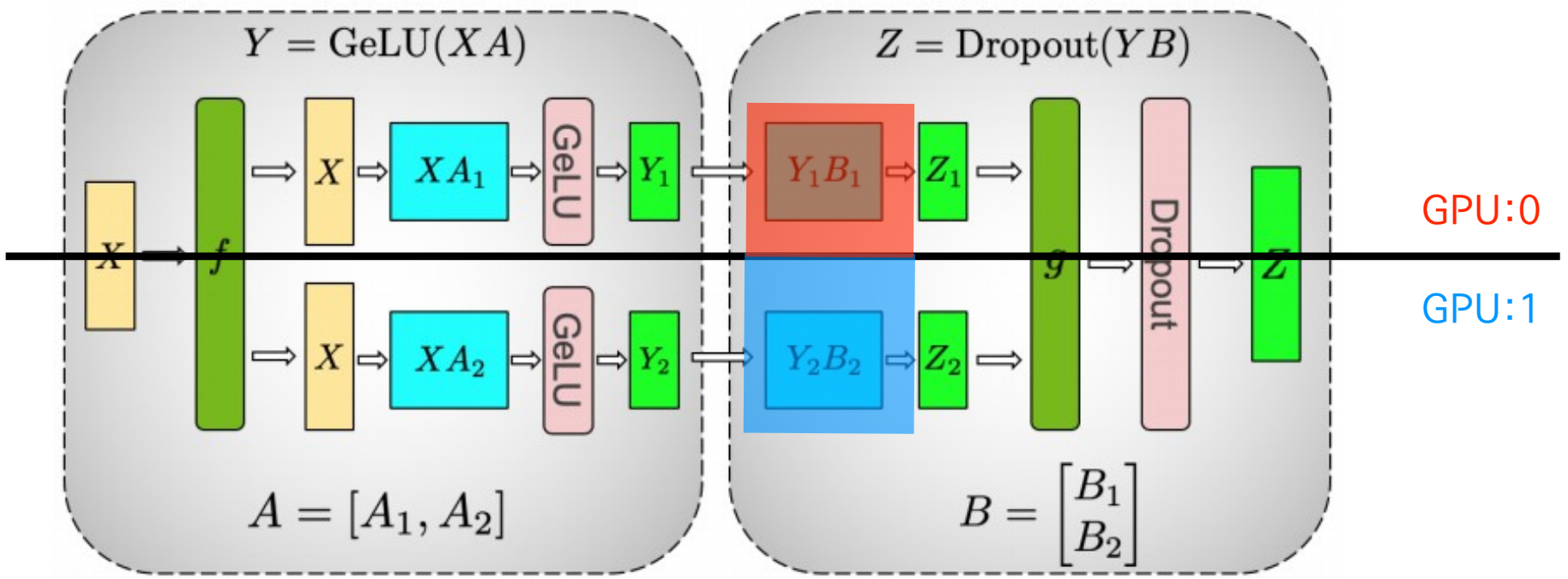
동일한 입력을 복사하여
각 디바이스로 복사

Megatron-LM: Training Multi-Billion Parameter Language Models Using Model Parallelism (https://arxiv.org/abs/1909.08053)



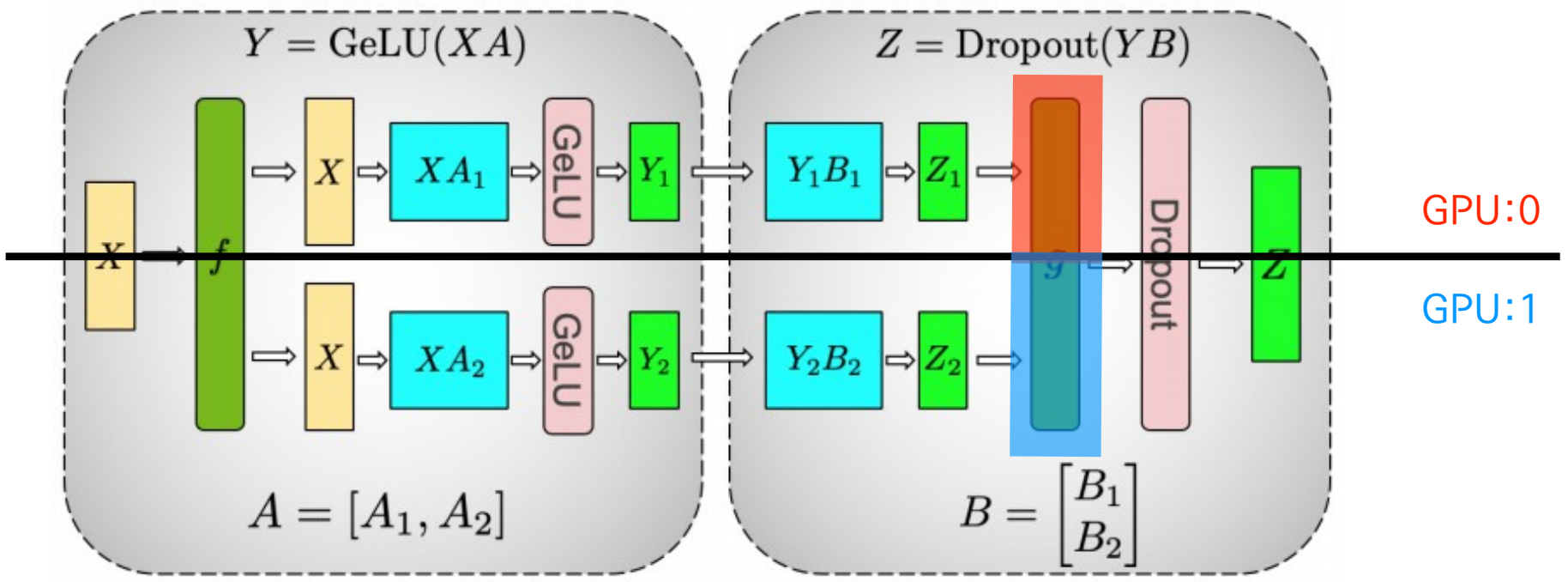
각 디바이스에서 병렬적으로
MLP 연산 수행

Megatron-LM: Training Multi-Billion Parameter Language Models Using Model Parallelism (https://arxiv.org/abs/1909.08053)



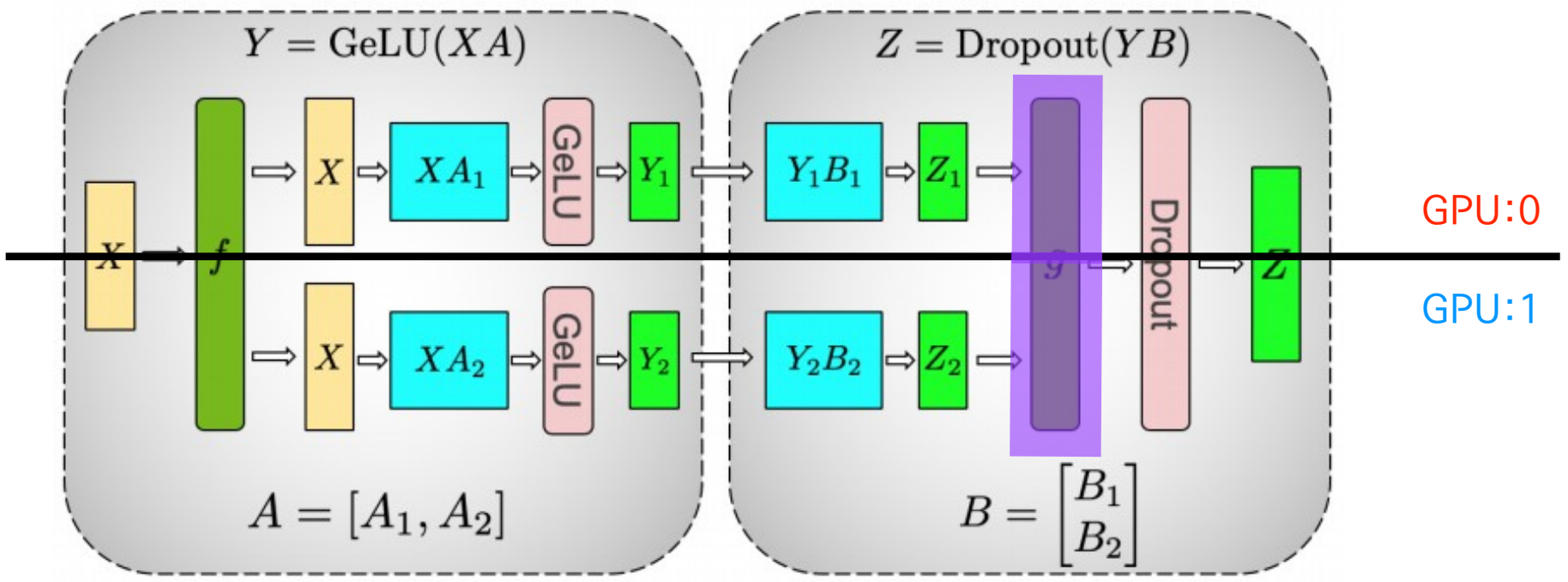
각 디바이스에서 병렬적으로
MLP 연산 수행

Megatron-LM: Training Multi-Billion Parameter Language Models Using Model Parallelism (https://arxiv.org/abs/1909.08053)



출력을 모두 더해줌

Megatron-LM: Training Multi-Billion Parameter Language Models Using Model Parallelism (https://arxiv.org/abs/1909.08053)



출력을 모두 더해줌

Megatron-LM: Training Multi-Billion Parameter Language Models Using Model Parallelism (https://arxiv.org/abs/1909.08053)

DeepSpeed: ZeRO 등 최신의 메모리 최적화 기법들을 제공하는 도구.

→ Megatron-LM + Kernel Fusion 기반의 모델병렬화 기능 지원함

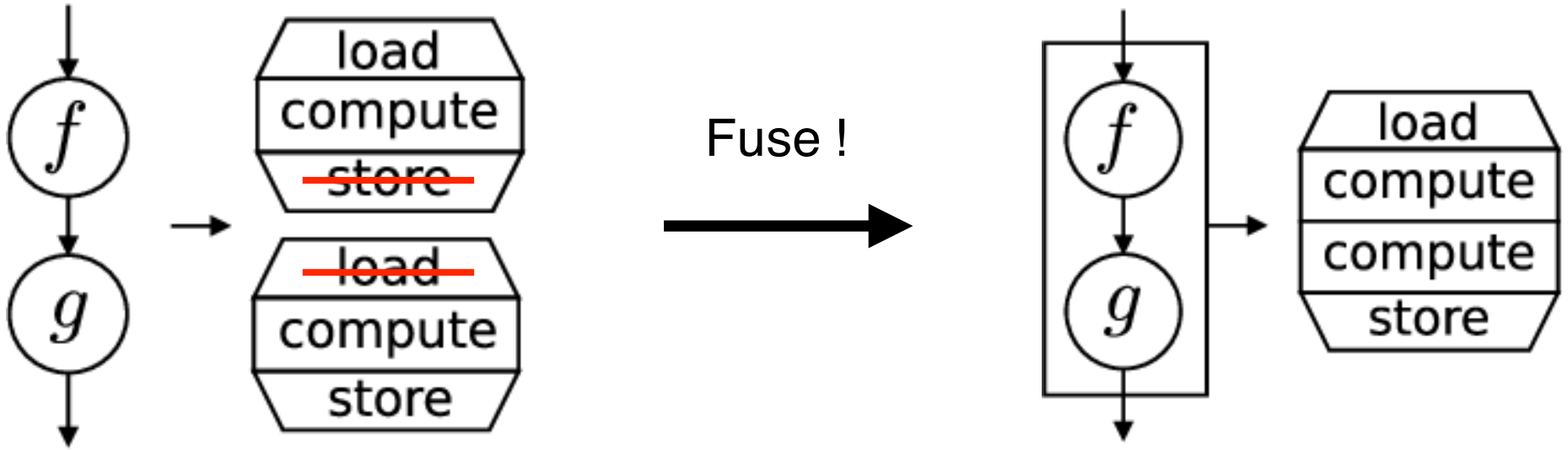
https://dl.acm.org/doi/10.1145/3394486.3406703

Faster Transformer: 자체 커널을 제공하는 모델 추론용 도구.

→ Megatron-LM + Kernel Fusion 기반의 모델병렬화 기능 지원함

https://github.com/NVIDIA/FasterTransformer

이들 도구들은 주로 Kernel Fusion을 통한 속도개선에 방점이 있음.



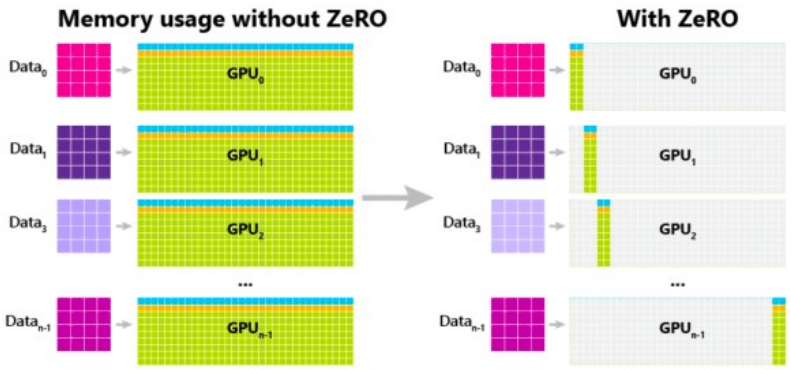
*Kernel이란?: CUDA 코드 중에서 GPU 디바이스에서 동작하는 영역을 의미

Problems

Problem 1: Scalability

1. Lack of Scalability: 속도는 빠르지만 지원하는 모델이 너무 적음

블렌더봇과 같은 Sequence to sequence 모델은 병렬화가 불가능함



DeepSpeed-inference:
GPT-2, GPT-Neo, BERT 지원

NVIDIA®

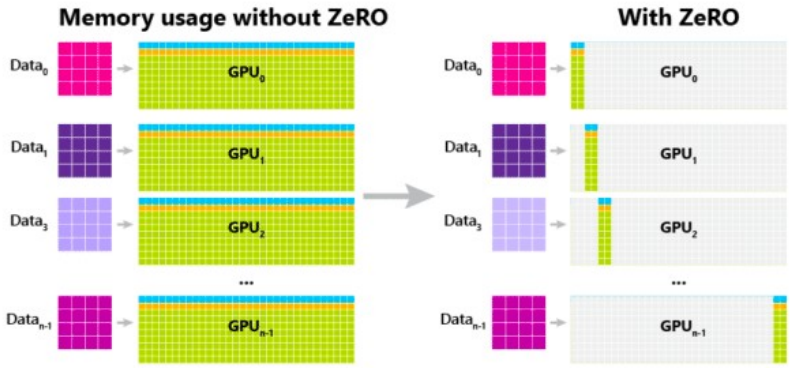
Faster Transformer:
GPT-2, XLNet, BERT 지원

Problem 2: Deployment

Language Conference

2. Deployment: DeepSpeed-inference는 웹서버에 배포 불가능

인퍼런스 전용 모듈인데도, (Data Parallel에 사용하는) 학습용 런처 를 사용하고 있기 때문



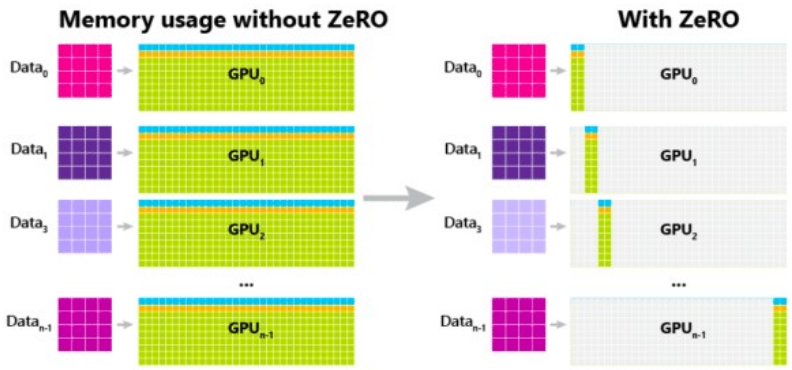
DeepSpeed-inference: 웹서버에 배포 불가능

Problem 3: Memory-Inefficiency

Language Conference

3. Memory-Inefficienty: DeepSpeed-inference는 메모리 비효율적임

GPU와 CPU 메모리 모든 영역에서 비효율적인 방식으로 병렬처리가 이루어지고 있었음



DeepSpeed-inference: 메모리 비효율적

Problem 4: Simplicity

4. Lack of Simplicity: Faster Transformer는 사용법이 너무 복잡함

사용법이 어려워서 이 분야의 입문자는 사용하는게 거의 불가능하다고 여겨짐

Faster Transformer: 사용법이 어려움

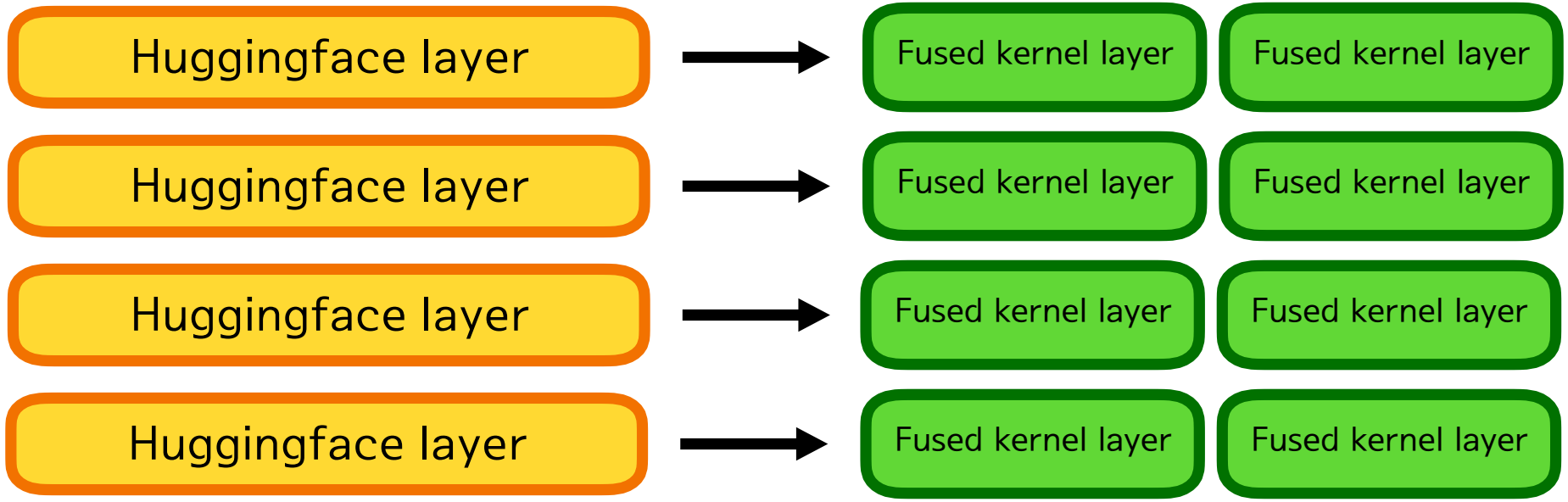
How to solve?

Solution 1: Scalability - No Fused CUDA kernel

Solution 1: Scalability - No Fused CUDA kernel

기존 도구들의 병렬화 방식

(1) Slice Tensor + (2) Replace Layer !



Solution 1: Scalability - No Fused CUDA kernel

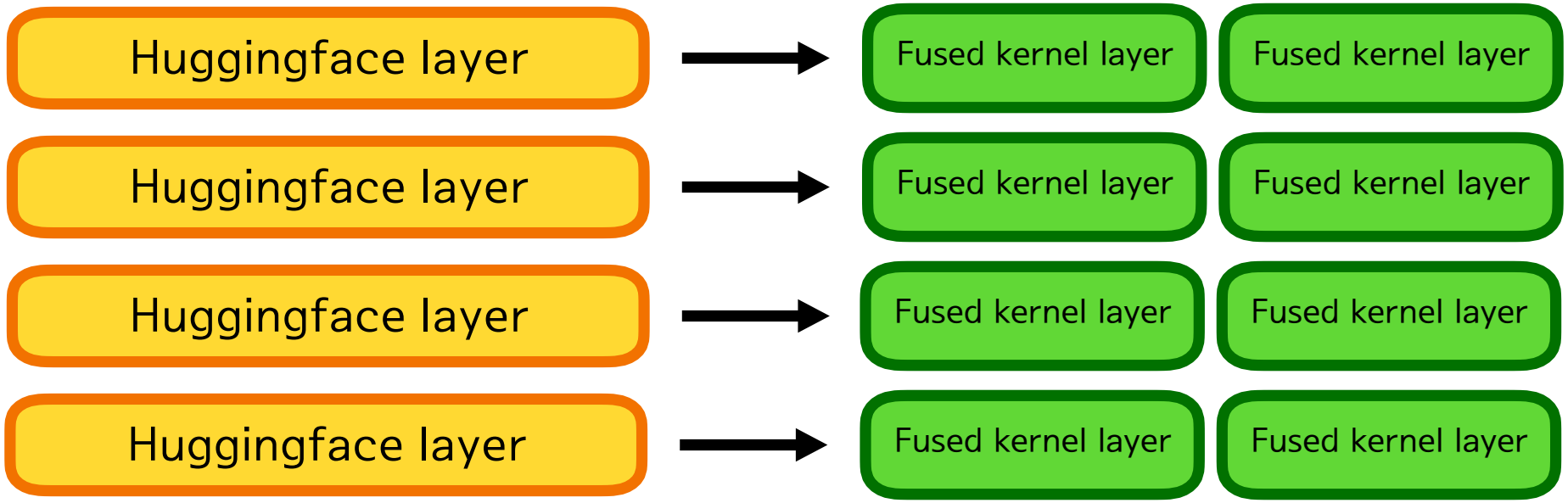
Language Conference

1) 모델 병렬화 가능한 파라미터를 불러온다.

2) 불러온 파라미터를 Megatron-LM 방식으로 Slice한다.

3) Slice된 파라미터로 Fused Kernel Layer를 생성한다.

4) 생성된 Layer로 기존의 Transformer Layer를 교체한다.



Solution 1: Scalability - No Fused CUDA kernel

Language Conference

1) 모델 병렬화 가능한 파라미터를 불러온다.

2) 불러온 파라미터를 Megatron-LM 방식으로 Slice한다.

3) Slice된 파라미터로 Fused Kernel Layer를 생성한다.

4) 생성된 Layer로 기존의 Transformer Layer를 교체한다.

```
>>> bert_model = BertModel.from_pretrained("bert-base-cased")

>>> # Attention layer 0
>>> query = bert_model.encoder.layer[0].attention.self.query.weight
>>> key = bert_model.encoder.layer[0].attention.self.key.weight
>>> value = bert_model.encoder.layer[0].attention.self.value.weight
>>> out_proj = bert_model.encoder.layer[0].attention.out.dense.wieght

>>> # MLP Layer 0
>>> mlp_h_to_4h = bert_model.encoder.layer[0].encoder.intermediate.dense.weight
>>> mlp_4h_to_h = bert_model.encoder.layer[0].encoder.output.dense.weight
```

Solution 1: Scalability - No Fused CUDA kernel

1) 모델 병렬화 가능한 파라미터를 불러온다.

2) 불러온 파라미터를 Megatron-LM 방식으로 Slice한다.

3) Slice된 파라미터로 Fused Kernel Layer를 생성한다.

4) 생성된 Layer로 기존의 Transformer Layer를 교체한다.

```
>>> def slice(weight, dim=0):
...     chunks = weight.chunk(dim=dim)
...     chunks[gpu_index].to(torch.cuda.current_device)  # scatter params
...     return chunks

>>> new_query = slice(query, dim=1)  # col
>>> new_key = slice(key, dim=1)  # col
>>> new_value = slice(value, dim=1)  # col
>>> new_out_proj = slice(out_proj, dim=0)  # row

>>> new_mlp_h_to_4h = slice(mlp_h_to_4h, dim=1)  # col
>>> new_mlp_4h_to_4h = slice(mlp_4h_to_h, dim=0)  # row
```

Solution 1: Scalability - No Fused CUDA kernel

Language Conference

1) 모델 병렬화 가능한 파라미터를 불러온다.

2) 불러온 파라미터를 Megatron-LM 방식으로 Slice한다.

3) Slice된 파라미터로 Fused Kernel Layer를 생성한다.

4) 생성된 Layer로 기존의 Transformer Layer를 교체한다.

```
>>> fused_kernel_layer = FusedKernelTransformerLayer(
...     qeury=new_query,
...     key=new_key,
...     value=new_value,
...     out_proj=new_out_proj,
...     mlp_h_to_4h=new_mlp_h_to_4h,
...     mlp_4h_to_h=new_mlp_4h_to_h,
...     attn_layer_norm= bert_model.encoder.layer[0].attention.out.LayerNorm.wieght,
...     mlp_layer_norm= bert_model.encoder.layer[0].output.LayerNorm.wieght,
... )
```

Solution 1: Scalability - No Fused CUDA kernel

1) 모델 병렬화 가능한 파라미터를 불러온다.

2) 불러온 파라미터를 Megatron-LM 방식으로 Slice한다.

3) Slice된 파라미터로 Fused Kernel Layer를 생성한다.

4) 생성된 Layer로 기존의 Transformer Layer를 교체한다.

```
>>> fused_kernel_layer = FusedKernelTransformerLayer(
...     qeury=new_query,
...     key=new_key,
...     value=new_value,
...     out_proj=new_out_proj,
...     mlp_h_to_4h=new_mlp_h_to_4h,
...     mlp_4h_to_h=new_mlp_4h_to_h,
...     attn_layer_norm= bert_model.encoder.layer[0].attention.out.LayerNorm.wieght,
...     mlp_layer_norm= bert_model.encoder.layer[0].output.LayerNorm.wieght,
... )
```

병렬화 불가능한 영역

Solution 1: Scalability - No Fused CUDA kernel

Language Conference

1) 모델 병렬화 가능한 파라미터를 불러온다.

2) 불러온 파라미터를 Megatron-LM 방식으로 Slice한다.

3) Slice된 파라미터로 Fused Kernel Layer를 생성한다.

4) 생성된 Layer로 기존의 Transformer Layer를 교체한다.

```
>>> # Replace
>>> bert_model.encoder.layer[0] = fused_kernel_layer

>>> # fused_kernel_layer의 forward 함수가 실행됨
>>> bert_model.encoder.layer[0].forward(input_tensors)
```

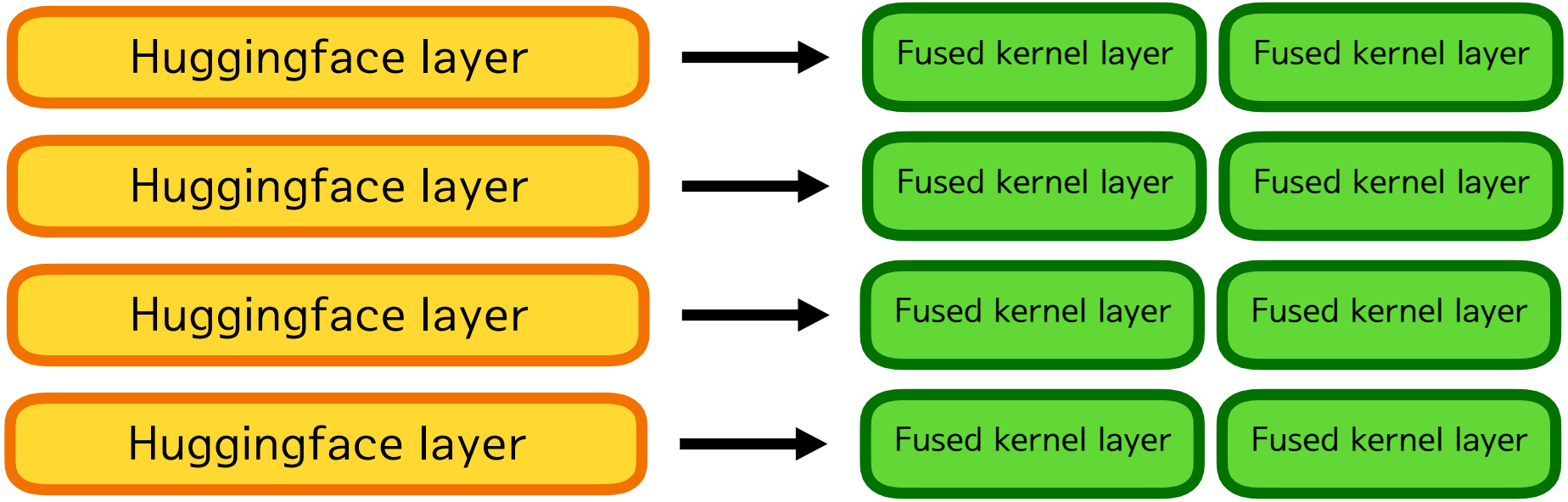
Solution 1: Scalability - No Fused CUDA kernel

1) 모델 병렬화 가능한 파라미터를 불러온다.

2) 불러온 파라미터를 Megatron-LM 방식으로 Slice한다.

3) Slice된 파라미터로 Fused Kernel Layer를 생성한다.

4) 생성된 Layer로 기존의 Transformer Layer를 교체한다.



Solution 1: Scalability - No Fused CUDA kernel

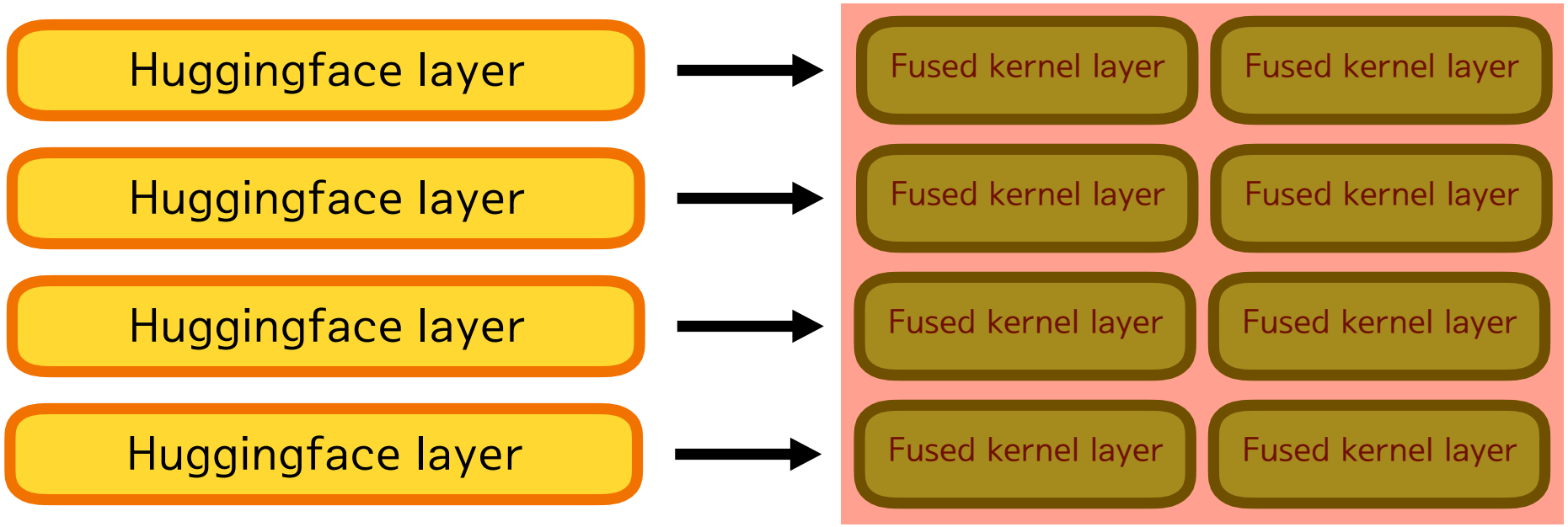
Language Conference

1) 모델 병렬화 가능한 파라미터를 불러온다.

2) 불러온 파라미터를 Megatron-LM 방식으로 Slice한다.

3) Slice된 파라미터로 Fused Kernel Layer를 생성한다.

4) 생성된 Layer로 기존의 Transformer Layer를 교체한다.



Solution 1: Scalability - No Fused CUDA kernel

1) 모델 병렬화 가능한 파라미터를 불러온다.

2) 불러온 파라미터를 Megatron-LM 방식으로 Slice한다.

3) Slice된 파라미터로 Fused Kernel Layer를 생성한다.

4) 생성된 Layer로 기존의 Transformer Layer를 교체한다.

```
>>> # Replace
>>> bert_model.encoder.layer[0] = fused_kernel_layer

>>> # fused_kernel_layer의 forward 함수가 실행됨
>>> bert_model.encoder.layer[0].forward(input_tensors)
```

Solution 1: Scalability - No Fused CUDA kernel

Language Conference

1) 모델 병렬화 가능한 파라미터를 불러온다.

2) 불러온 파라미터를 Megatron-LM 방식으로 Slice한다.

3) Slice된 파라미터로 Fused Kernel Layer를 생성한다.

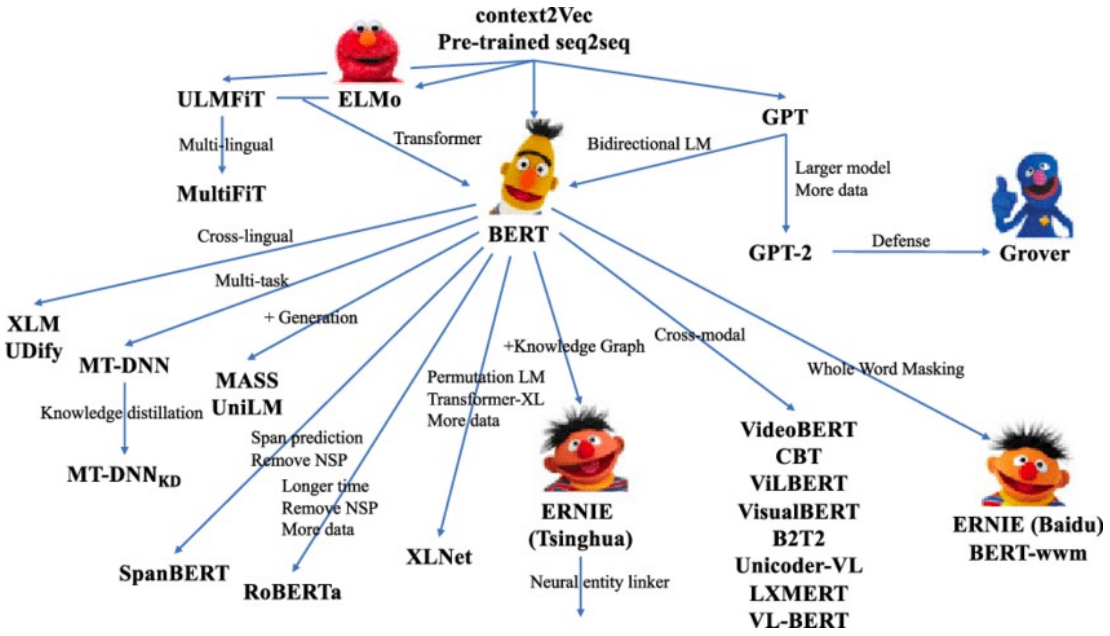
4) 생성된 Layer로 기존의 Transformer Layer를 교체한다.

```
>>> # Replace
>>> bert_model.encoder.layer[0] = fused_kernel_layer

>>> # fused_kernel_layer의 forward 함수가 실행됨
>>> bert_model.encoder.layer[0].forward(input_tensors)
```

Solution 1: Scalability - No Fused CUDA kernel

Language Conference

문제는 Language Model이 많아도 너무 많다는 것.




```
>>> fused_kernel_layer = FusedKernelTransformerLayer(
...     qeury=new_query,
...     key=new_key,
...     value=new_value,
...     out_proj=new_out_proj,
...     mlp_h_to_4h=new_mlp_h_to_4h,
...     mlp_4h_to_h=new_mlp_4h_to_h,
...     attn_layer_norm= bert_model.encoder.layer[0].attention...
...     mlp_layer_norm= bert_model.encoder.layer[0].output...
... )
```

→ Fused CUDA Kernel을 사용하기 위해 위의 클래스를 사용

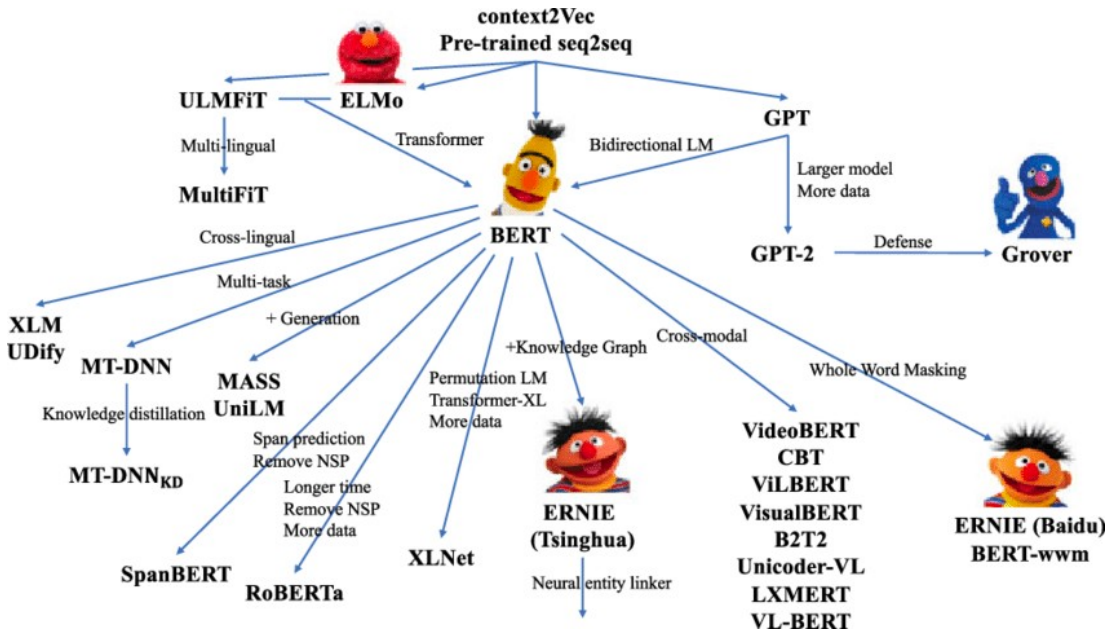
→ 특정 모델을 지원하려면 CUDA로 해당 모델의 로직을 구현해야 함

→ 모든 모델의 로직을 CUDA로 전부 다시 구현하는 것은 사실상 불가능

→ 그래서 대표적인 모델 2~3개만 지원하고 있는 것

Solution 1: Scalability - No Fused CUDA kernel

문제는 Language Model이 많아도 너무 많다는 것.



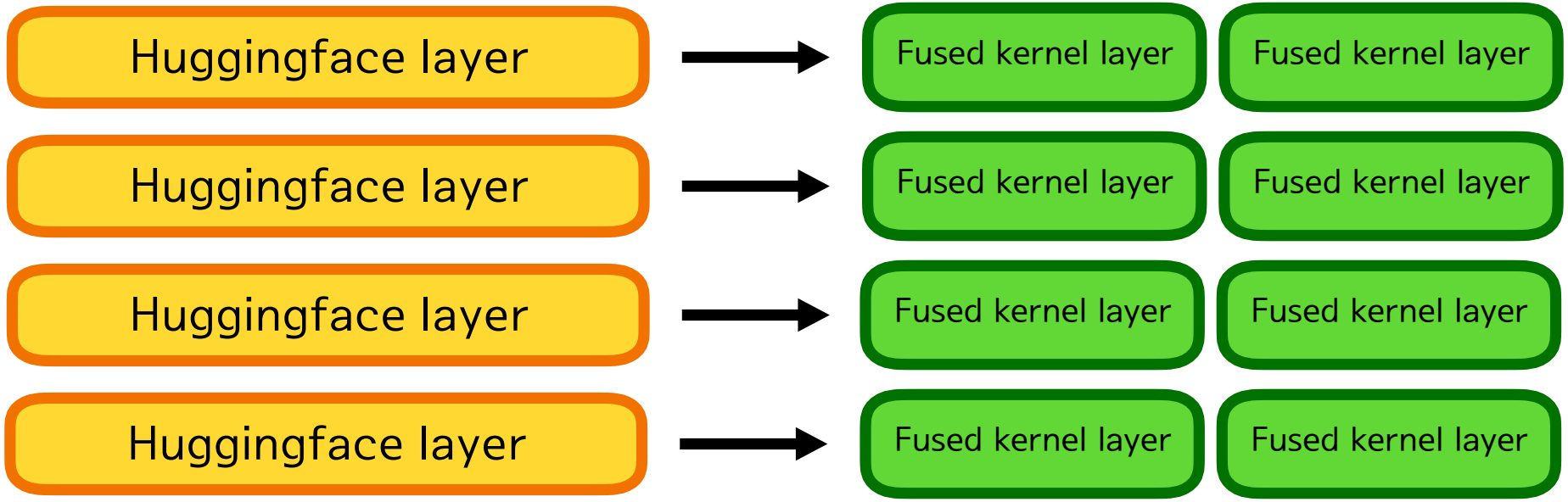
그러면 그냥 Fused kernel을 안쓰면 되는거 아닌가...?

kevin.ko

Solution 1: Scalability - No Fused CUDA kernel

기존 도구들의 병렬화 방식

(1) Slice Tensor + (2) Replace Layer !

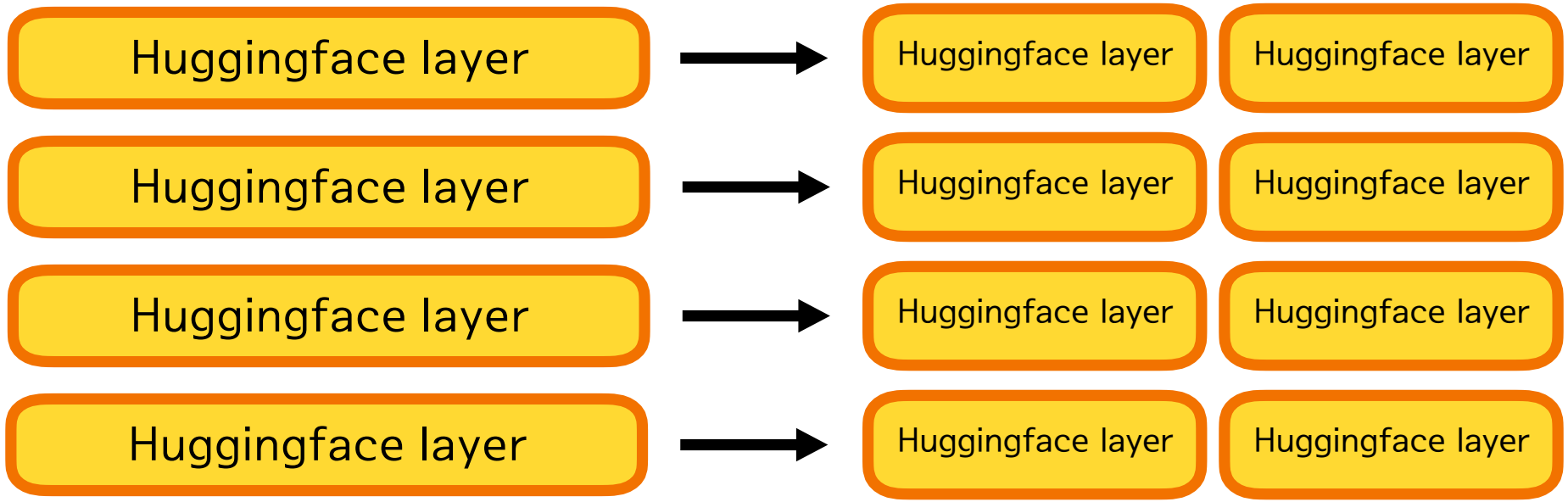


Solution 1: Scalability - No Fused CUDA kernel

Language Conference

Parallelformers의 병렬화 방식

(1) Slice Tensor + (2) Replace Tensor !



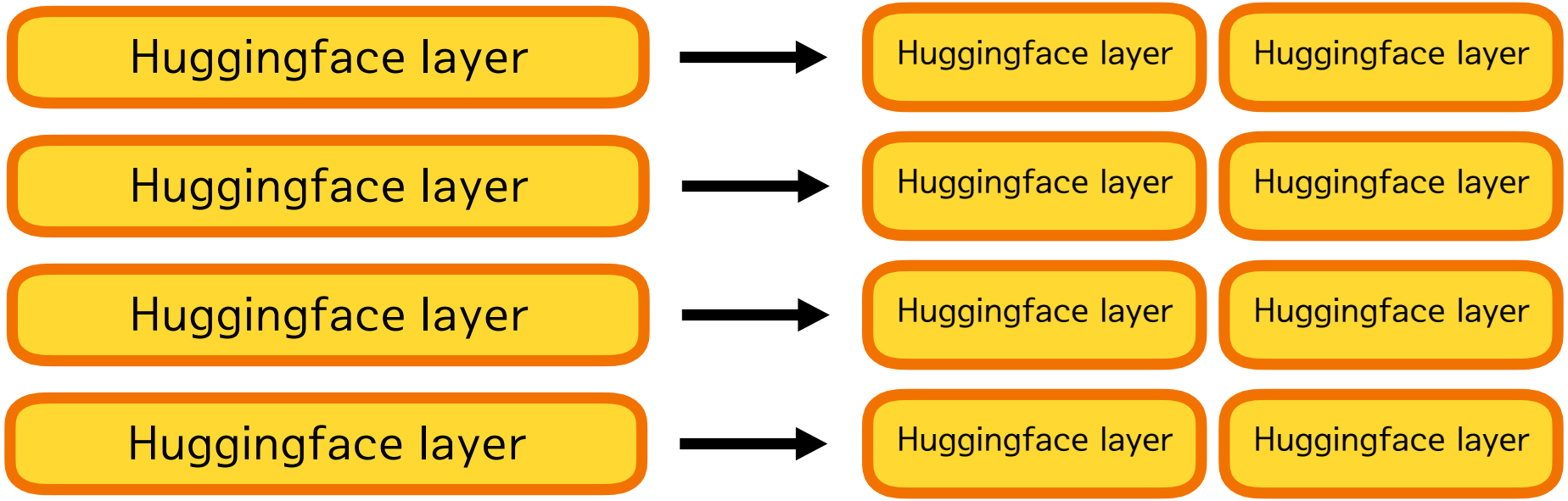
Solution 1: Scalability - No Fused CUDA kernel

Language Conference

1) 모델 병렬화 가능한 파라미터를 불러온다.

2) 불러온 파라미터를 Megatron-LM 방식으로 Slice한다.

3) Slice된 파라미터를 기존 Layer에 삽입한다.



Solution 1: Scalability - No Fused CUDA kernel

Language Conference

1) 모델 병렬화 가능한 파라미터를 불러온다. (동일)

2) 불러온 파라미터를 Megatron-LM 방식으로 Slice한다.

3) Slice된 파라미터를 기존 Layer에 삽입한다.

```
>>> bert_model = BertModel.from_pretrained("bert-base-cased")

>>> # Attention layer 0
>>> query = bert_model.encoder.layer[0].attention.self.query.weight
>>> key = bert_model.encoder.layer[0].attention.self.key.weight
>>> value = bert_model.encoder.layer[0].attention.self.value.weight
>>> out_proj = bert_model.encoder.layer[0].attention.out.dense.wieght

>>> # MLP Layer 0
>>> mlp_h_to_4h = bert_model.encoder.layer[0].encoder.intermediate.dense.weight
>>> mlp_4h_to_h = bert_model.encoder.layer[0].encoder.output.dense.weight
```

Solution 1: Scalability - No Fused CUDA kernel

Language Conference

1) 모델 병렬화 가능한 파라미터를 불러온다. (동일)

2) 불러온 파라미터를 Megatron-LM 방식으로 Slice한다. (동일)

3) Slice된 파라미터를 기존 Layer에 삽입한다.

```
>>> def slice(weight, dim=0):
...     chunks = weight.chunk(dim=dim)
...     chunks[gpu_index].to(torch.cuda.current_device)  # scatter params
...     return chunks

>>> new_query = slice(query, dim=0)  # row
>>> new_key = slice(key, dim=0)  # row
>>> new_value = slice(value, dim=0)  # row
>>> new_out_proj = slice(out_proj, dim=1)  # col

>>> new_mlp_h_to_4h = slice(mlp_h_to_4h, dim=0)  # row
>>> new_mlp_4h_to_4h = slice(mlp_4h_to_h, dim=1)  # col
```

Solution 1: Scalability - No Fused CUDA kernel

Language Conference

1) 모델 병렬화 가능한 파라미터를 불러온다. (동일)

2) 불러온 파라미터를 Megatron-LM 방식으로 Slice한다. (동일)

3) Slice된 파라미터를 기존 Layer에 삽입한다. (여기가 핵심!)

```
>>> # Inject
>>> bert_model.encoder.layer[0].attention.self.query.weight = new_query
>>> bert_model.encoder.layer[0].attention.self.key.weight = new_key
>>> bert_model.encoder.layer[0].attention.self.value.weight = new_value
>>> bert_model.encoder.layer[0].attention.self.out.weight = new_out_proj
... (생략)

>>> # bert_model.encoder.layer의 forward 함수가 실행됨
>>> bert_model.encoder.layer[0].forward(input_tensors)
```

Solution 1: Scalability - No Fused CUDA kernel

Language Conference

1) 모델 병렬화 가능한 파라미터를 불러온다. (동일)

2) 불러온 파라미터를 Megatron-LM 방식으로 Slice한다. (동일)

3) Slice된 파라미터를 기존 Layer에 삽입한다. (여기가 핵심!)

```
>>> # Inject
>>> bert_model.encoder.layer[0].attention.self.query.weight = new_query
>>> bert_model.encoder.layer[0].attention.self.key.weight = new_key
>>> bert_model.encoder.layer[0].attention.self.value.weight = new_value
>>> bert_model.encoder.layer[0].attention.self.out.weight = new_out_proj
... (생략)

>>> # bert_model.encoder.layer의 forward 함수가 실행됨
>>> bert_model.encoder.layer[0].forward(input_tensors)
```

Solution 1: Scalability - No Fused CUDA kernel

Language Conference

Kernel fusion

```python
def forward(
    self,
    hidden_states,
    attention_mask=None,
    head_mask=None,
    encoder_hidden_states=None,
    encoder_attention_mask=None,
    past_key_value=None,
    output_attentions=False,
):
    mixed_query_layer = self.query(hidden_states)

    # If this is instantiated as a cross-attention module, the keys
    # and values come from an encoder; the attention mask needs to be
    # such that the encoder's padding tokens are not attended to.
    is_cross_attention = encoder_hidden_states is not None

    if is_cross_attention and past_key_value is not None:
        # reuse k,v, cross_attentions
        key_layer = past_key_value[0]
        value_layer = past_key_value[1]
        attention_mask = encoder_attention_mask
    elif is_cross_attention:
        key_layer = self.transpose_for_scores(self.key(encoder_hidden_states))
        value_layer = self.transpose_for_scores(self.value(encoder_hidden_states))
        attention_mask = encoder_attention_mask
```

Parallelformers

```python
def forward(
    self,
    hidden_states,
    attention_mask=None,
    head_mask=None,
    encoder_hidden_states=None,
    encoder_attention_mask=None,
    past_key_value=None,
    output_attentions=False,
):
    mixed_query_layer = self.query(hidden_states)

    # If this is instantiated as a cross-attention module, the keys
    # and values come from an encoder; the attention mask needs to be
    # such that the encoder's padding tokens are not attended to.
    is_cross_attention = encoder_hidden_states is not None

    if is_cross_attention and past_key_value is not None:
        # reuse k,v, cross_attentions
        key_layer = past_key_value[0]
        value_layer = past_key_value[1]
        attention_mask = encoder_attention_mask
    elif is_cross_attention:
        key_layer = self.transpose_for_scores(self.key(encoder_hidden_states))
        value_layer = self.transpose_for_scores(self.value(encoder_hidden_states))
        attention_mask = encoder_attention_mask
```

Replacement Areas:

Solution 1: Scalability - No Fused CUDA kernel

Language Conference

Kernel fusion

```
def forward(
    self,
    hidden_states,
    attention_mask=None,
    head_mask=None,
    encoder_hidden_states=None,
    encoder_attention_mask=None,
    past_key_value=None,
    output_attentions=False,
):
    mixed_query_layer = self.query(hidden_states)

    # If this is instantiated as a cross-attention module, the keys
    # and values come from an encoder; the attention mask needs to be
    # such that the encoder's padding tokens are not attended to.
    is_cross_attention = encoder_hidden_states is not None

    if is_cross_attention and past_key_value is not None:
        # reuse k,v, cross_attentions
        key_layer = past_key_value[0]
        value_layer = past_key_value[1]
        attention_mask = encoder_attention_mask
    elif is_cross_attention:
        key_layer = self.transpose_for_scores(self.key(encoder_hidden_states))
        value_layer = self.transpose_for_scores(self.value(encoder_hidden_states))
        attention_mask = encoder_attention_mask
```

Replacement Areas:

Parallelformers

```
def forward(
    self,
    hidden_states,
    attention_mask=None,
    head_mask=None,
    encoder_hidden_states=None,
    encoder_attention_mask=None,
    past_key_value=None,
    output_attentions=False,
):
    mixed_query_layer = self.query(hidden_states)

    # If this is instantiated as a cross-attention module, the keys
    # and values come from an encoder; the attention mask needs to be
    # such that the encoder's padding tokens are not attended to.
    is_cross_attention = encoder_hidden_states is not None

    if is_cross_attention and past_key_value is not None:
        # reuse k,v, cross_attentions
        key_layer = past_key_value[0]
        value_layer = past_key_value[1]
        attention_mask = encoder_attention_mask
    elif is_cross_attention:
        key_layer = self.transpose_for_scores(self.key(encoder_hidden_states))
        value_layer = self.transpose_for_scores(self.value(encoder_hidden_states))
        attention_mask = encoder_attention_mask
```

각종 모델이 가진 특수한 메커니즘에
관한 코드를 전부 그대로 활용 !

Solution 1: Scalability - No Fused CUDA kernel

Language Conference

Fully Supported Models

- ALBERT
- BART
- BARThez (=BERT)
- BERT
- BERTweet (=BERT)
- BertJapanese (=BERT)
- BertGeneration
- Blenderbot
- Blenderbot Samll
- BORT (=BERT)
- CamemBERT (=RoBERTa)
- CLIP
- CPM
- CTRL
- DeBERTa
- DeBERTa-v2
- DeiT
- DETR
- DialoGPT (=GPT2)
- DistilBERT
- DPR (=BERT)
- ELECTRA
- FlauBERT (=XLM)
- FSMT
- Funnel Transformer
- herBERT (=RoBERTa)
- I-BERT
- LayoutLM
- LED
- Longformer
- LUKE
- LXMERT
- MarianMT
- M2M100
- MBart
- Mobile BERT
- MPNet
- MT5 (=T5)
- Megatron BERT (=BERT)
- Megatron GPT2 (=GPT2)
- OpenAI GPT
- OpenAI GPT2
- GPTNeo
- Hubert
- Pegasus
- PhoBERT (=RoBERTa)
- Reformer
- RetriBERT
- RoBERTa
- RoFormer
- Speech2Text
- T5
- ByT5 (=T5)
- TAPAS
- TransformerXL
- ViT
- VisualBERT
- Wav2Vec2
- XLM
- XLM-RoBERTa (=RoBERTa)
- XLNet
- XLSR-Wave2Vec2

Partly Supported Models

- BigBird
- BigBirdPegasus
- ConvBERT
- ProphetNet
- XLM-ProphetNet

Unsupported Models

- SqueezeBERT
- RAG

Huggingface Transformers의 70개 모델 중 68개의 모델 병렬화 성공,

속도에 대한 이점을 포기했지만 Scalablity의 이점을 얻음.

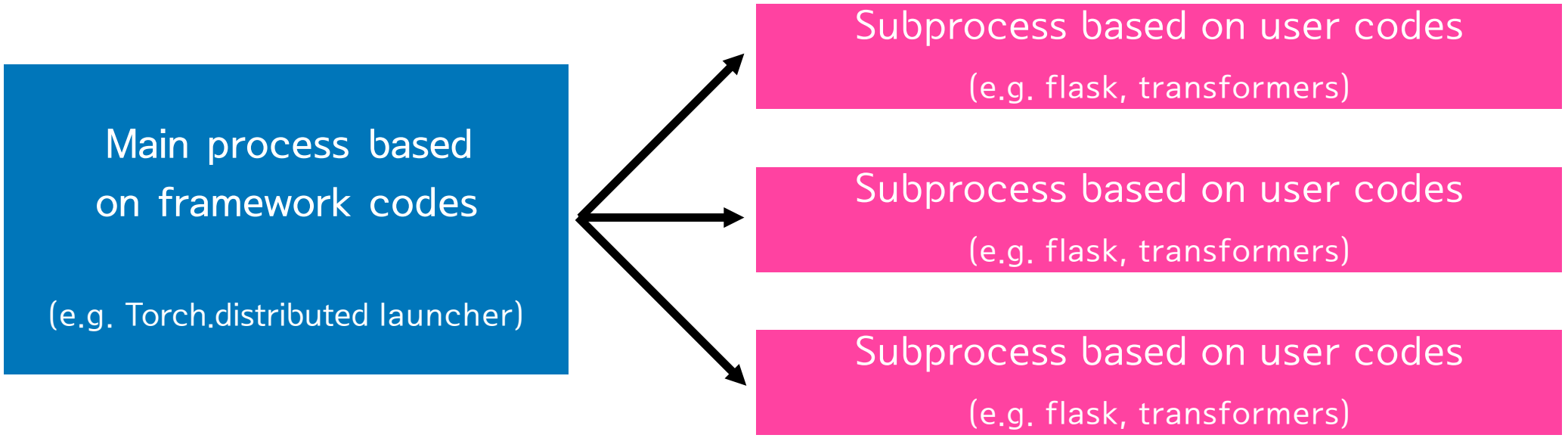
언어모델 뿐만 아닌 ViT, CLIP같은 비전모델, Wav2Vec2와 같은 음성모델도 병렬화 가능

Solution 2: Deployment: Inversion of process control

Solution 2: Deployment - Inversion of process control

Language Conference

기존의 멀티프로세싱 런처는 프레임워크의 코드가 유저의 코드를 여러번 동시에 실행시키는 구조.

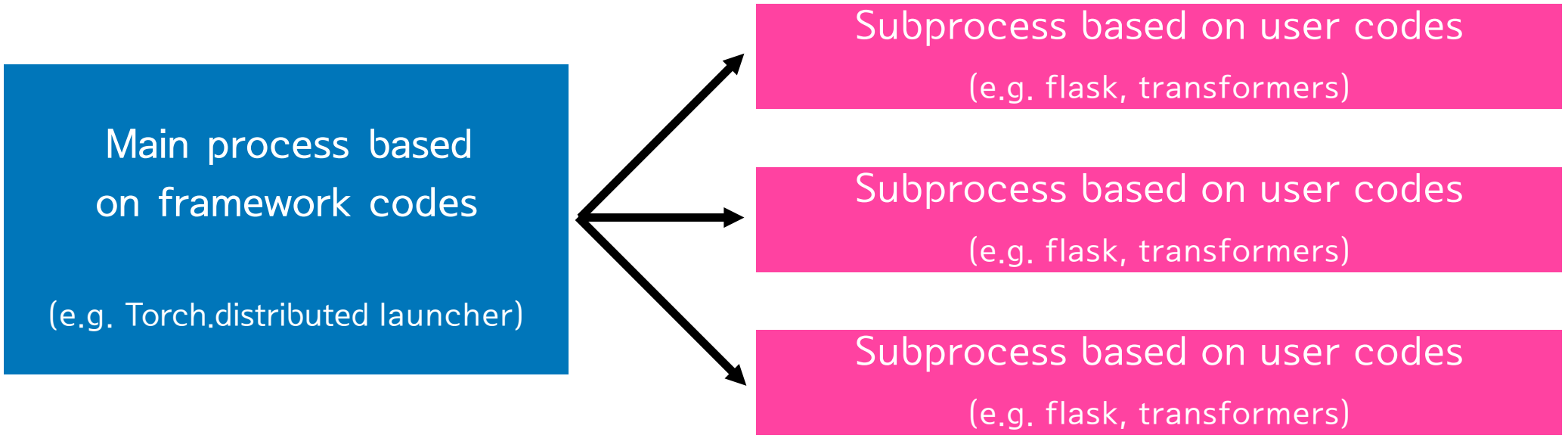


Solution 2: Deployment - Inversion of process control

Language Conference

여기에는 두가지 문제가 있음

1) 병렬화와 관계 없는 부분까지 여러번 실행되는 문제
2) 병렬화를 해제할 수 없는 문제 (Deparallelization 불가)



Solution 2: Deployment - Inversion of process control

Language Conference

여기에는 두가지 문제가 있음

1) 병렬화와 관계 없는 부분까지 여러번 실행되는 문제

2) 병렬화를 해제할 수 없는 문제 (Deparallelization 불가)

```
from transformers import AutoModelForCausalLM, AutoTokenizer

model = AutoModelForCausalLM.from_pretrained("EleutherAI/gpt-neo-2.7B")
tokenizer = AutoTokenizer.from_pretrained("EleutherAI/gpt-neo-2.7B")
```

우리가 흔히 쓰는 Transformers 모델 로딩 코드

Solution 2: Deployment - Inversion of process control

여기에는 두가지 문제가 있음

1) 병렬화와 관계 없는 부분까지 여러번 실행되는 문제

2) 병렬화를 해제할 수 없는 문제 (Deparallelization 불가)

CPU 메모리 초과

Data Parallelism에서 쓰던 런쳐 !!

Solution 2: Deployment - Inversion of process control

여기에는 두가지 문제가 있음

1) 병렬화와 관계 없는 부분까지 여러번 실행되는 문제

2) 병렬화를 해제할 수 없는 문제 (Deparallelization 불가)

```
app.run(host="0.0.0.0", port=5000)
```

우리가 흔히 쓰는 Flask 코드

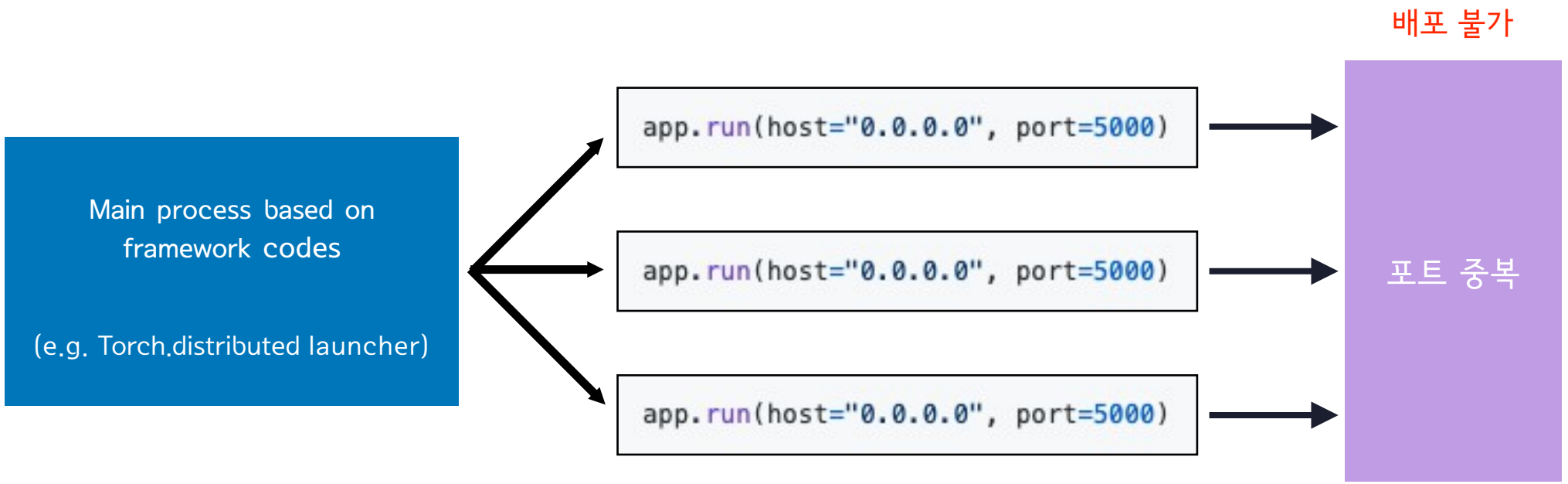
Solution 2: Deployment - Inversion of process control

Language Conference

여기에는 두가지 문제가 있음

1) 병렬화와 관계 없는 부분까지 여러번 실행되는 문제

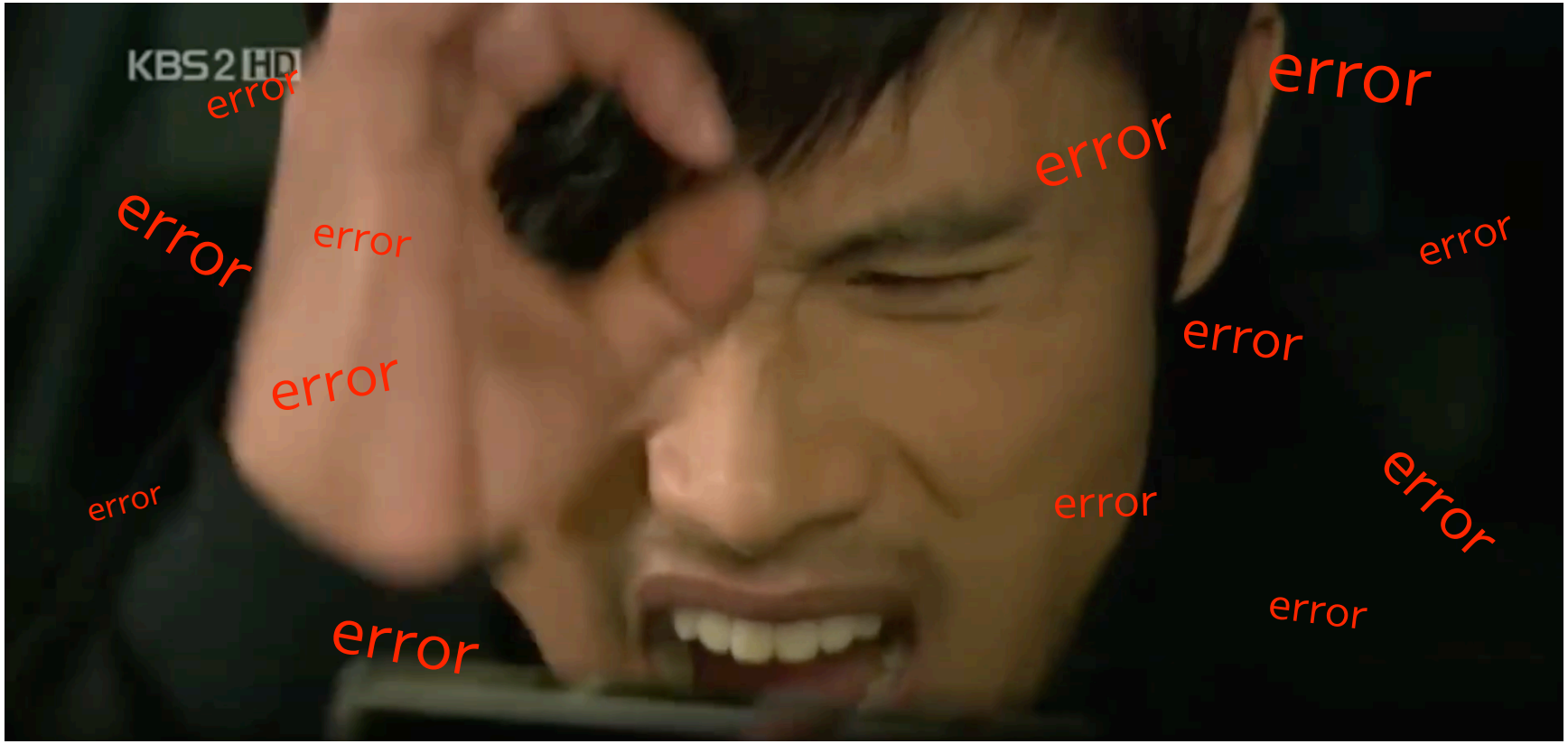
2) 병렬화를 해제할 수 없는 문제 (Deparallelization 불가)



Solution 2: Deployment - Inversion of process control

Language Conference

Solution 2: Deployment - Inversion of process control



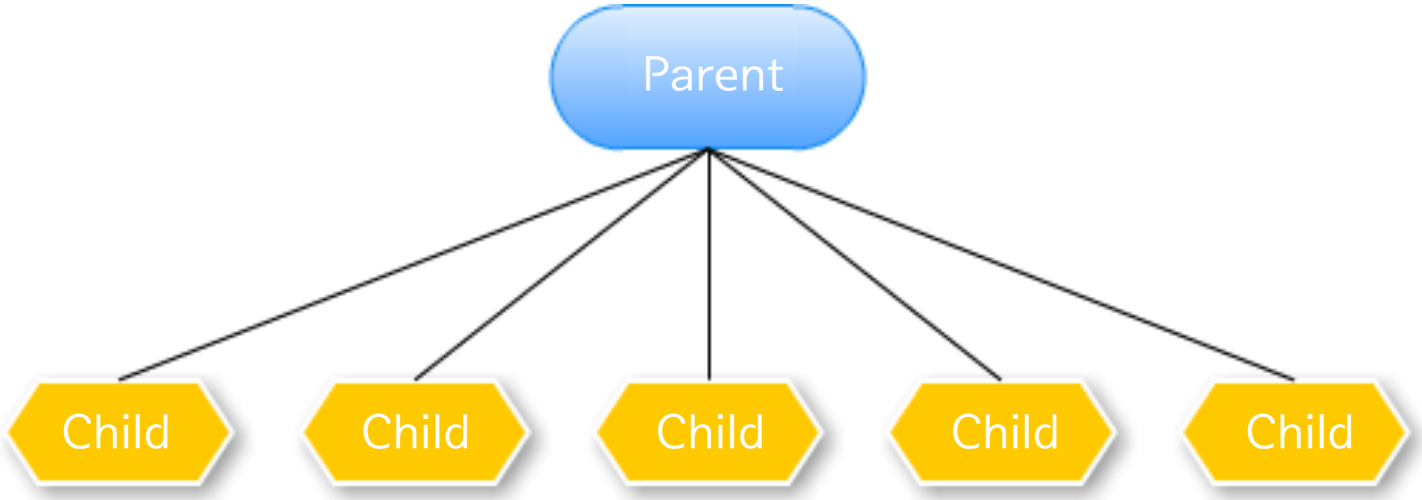
Solution 2: Deployment - Inversion of process control

Language Conference

여기에는 두가지 문제가 있음

1) 병렬화와 관계 없는 부분까지 여러번 실행되는 문제

2) 병렬화를 해제할 수 없는 문제 (Deparallelization 불가)



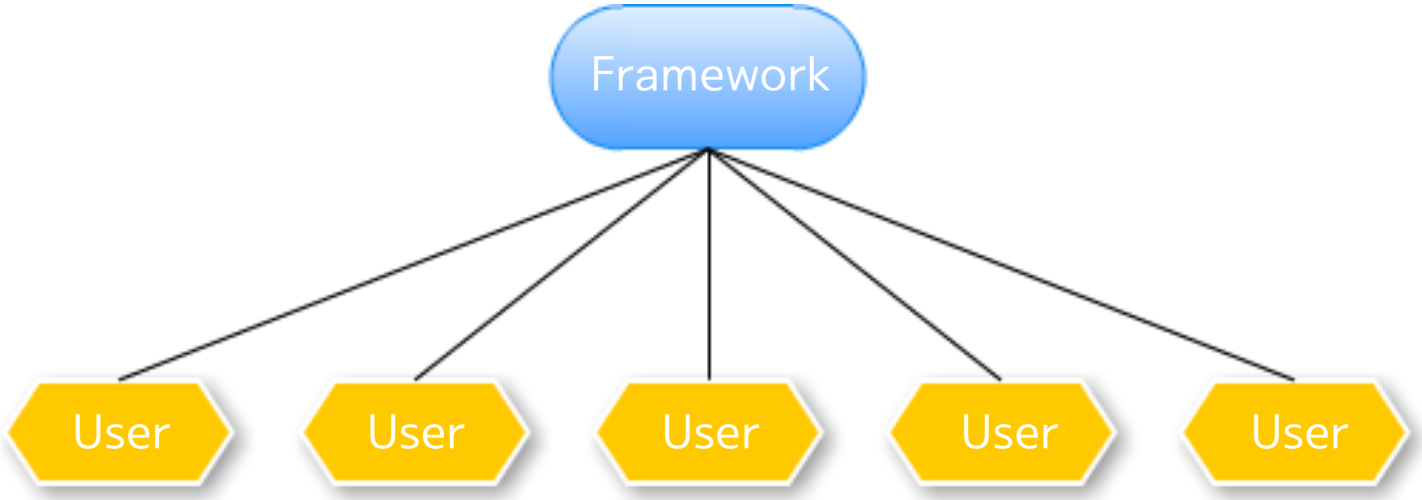
Join은 부모 프로세스만 할 수 있음

Solution 2: Deployment - Inversion of process control

여기에는 두가지 문제가 있음

1) 병렬화와 관계 없는 부분까지 여러번 실행되는 문제

2) 병렬화를 해제할 수 없는 문제 (Deparallelization 불가)



User의 코드에서는 join()을 호출할 수 없음 → 병렬화 해제 불가

Solution 2: Deployment - Inversion of process control

Language Conference

병렬화 해야하는 부분만 병렬적으로 동작하게 하자!

```python
from transformers import AutoModelForCausalLM, AutoTokenizer

model = AutoModelForCausalLM.from_pretrained("EleutherAI/gpt-neo-2.7B")
tokenizer = AutoTokenizer.from_pretrained("EleutherAI/gpt-neo-2.7B")
```

code of deepspeed ...

```python
from flask import Flask

app = Flask(__name__)


@app.route("/generate_text/<text>")
def generate_text(text):
    inputs = tokenizer(text, return_tensors="pt")

    outputs = model.generate(
        **inputs,
        num_beams=5,
        no_repeat_ngram_size=4,
        max_length=15,
    )

    outputs = tokenizer.batch_decode(
        outputs,
        skip_special_tokens=True,
    )

    return {
        "inputs": text,
        "outputs": outputs[0],
    }


app.run(host="0.0.0.0", port=5000)
```

DeepSpeed-Inference

Parallel

```python
from transformers import AutoModelForCausalLM, AutoTokenizer

model = AutoModelForCausalLM.from_pretrained("EleutherAI/gpt-neo-2.7B")
tokenizer = AutoTokenizer.from_pretrained("EleutherAI/gpt-neo-2.7B")

from parallelformers import parallelize

parallelize(model, num_gpus=2, fp16=True, verbose='detail')

from flask import Flask

app = Flask(__name__)


@app.route("/generate_text/<text>")
def generate_text(text):
    inputs = tokenizer(text, return_tensors="pt")

    outputs = model.generate(
        **inputs,
        num_beams=5,
        no_repeat_ngram_size=4,
        max_length=15,
    )

    outputs = tokenizer.batch_decode(
        outputs,
        skip_special_tokens=True,
    )

    return {
        "inputs": text,
        "outputs": outputs[0],
    }


app.run(host="0.0.0.0", port=5000)
```

Parallelformers

Single

Parallel

Single

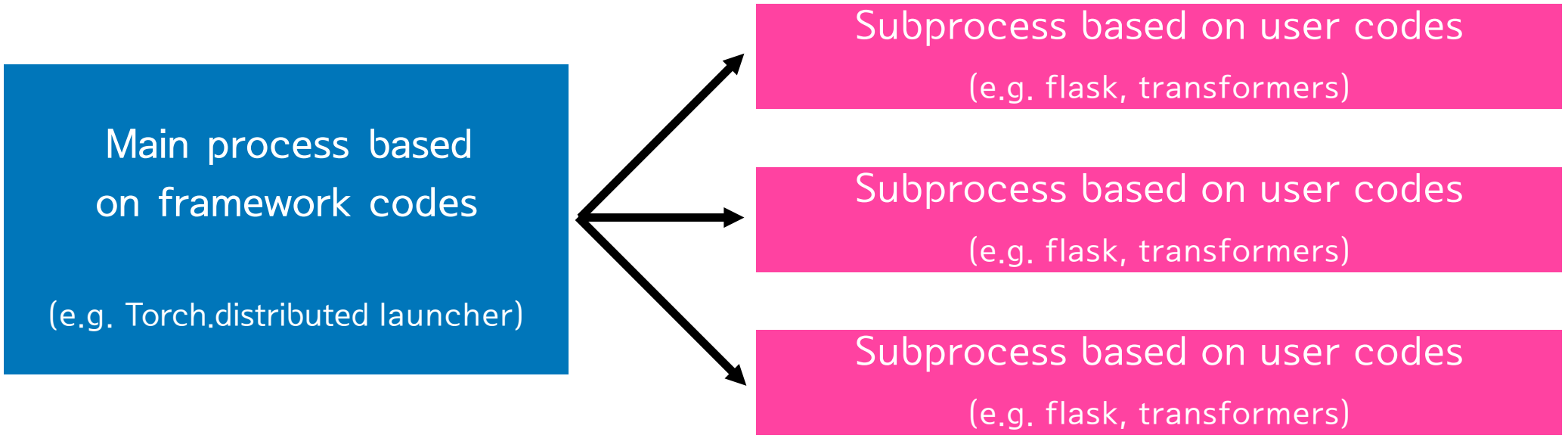
Parallel

Single

Solution 2: Deployment - Inversion of process control

Language Conference

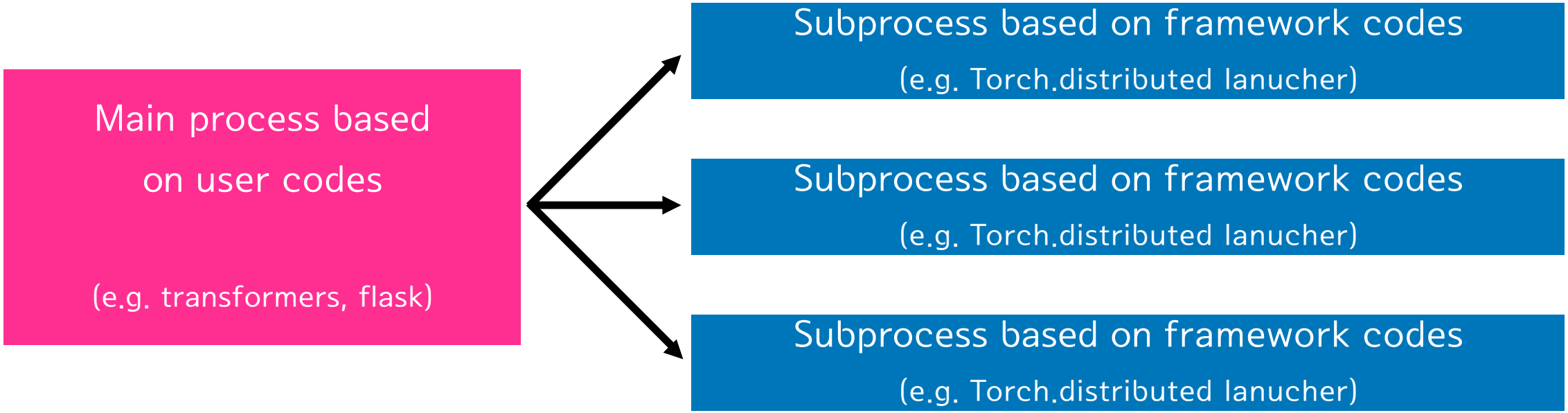
기존의 멀티프로세싱 런처는 프레임워크의 코드가 유저의 코드를 여러번 동시에 실행시키는 구조.



Solution 2: Deployment - Inversion of process control

Parallelformers는 유저의 코드에서 프레임워크의 코드를 여러번 동시에 실행시키는 구조.

= Inversion of process control

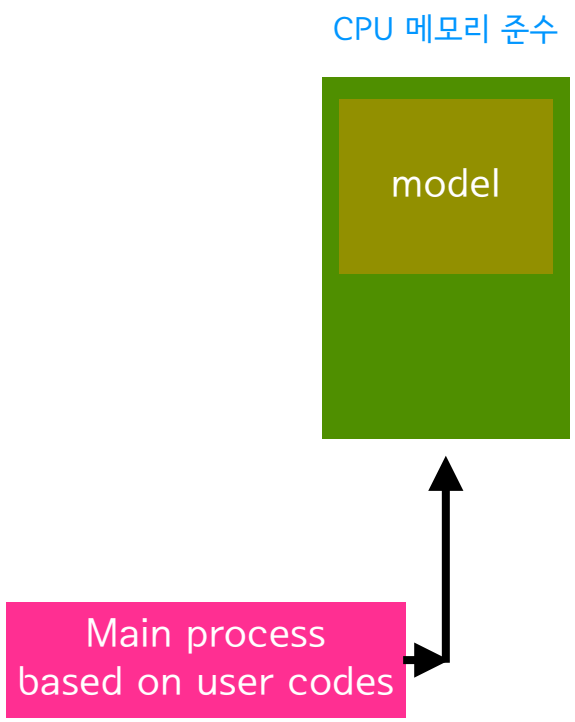


Solution 2: Deployment - Inversion of process control

Language Conference

```python
from transformers import AutoModelForCausalLM, AutoTokenizer

model = AutoModelForCausalLM.from_pretrained("EleutherAI/gpt-neo-2.7B")
tokenizer = AutoTokenizer.from_pretrained("EleutherAI/gpt-neo-2.7B")
```

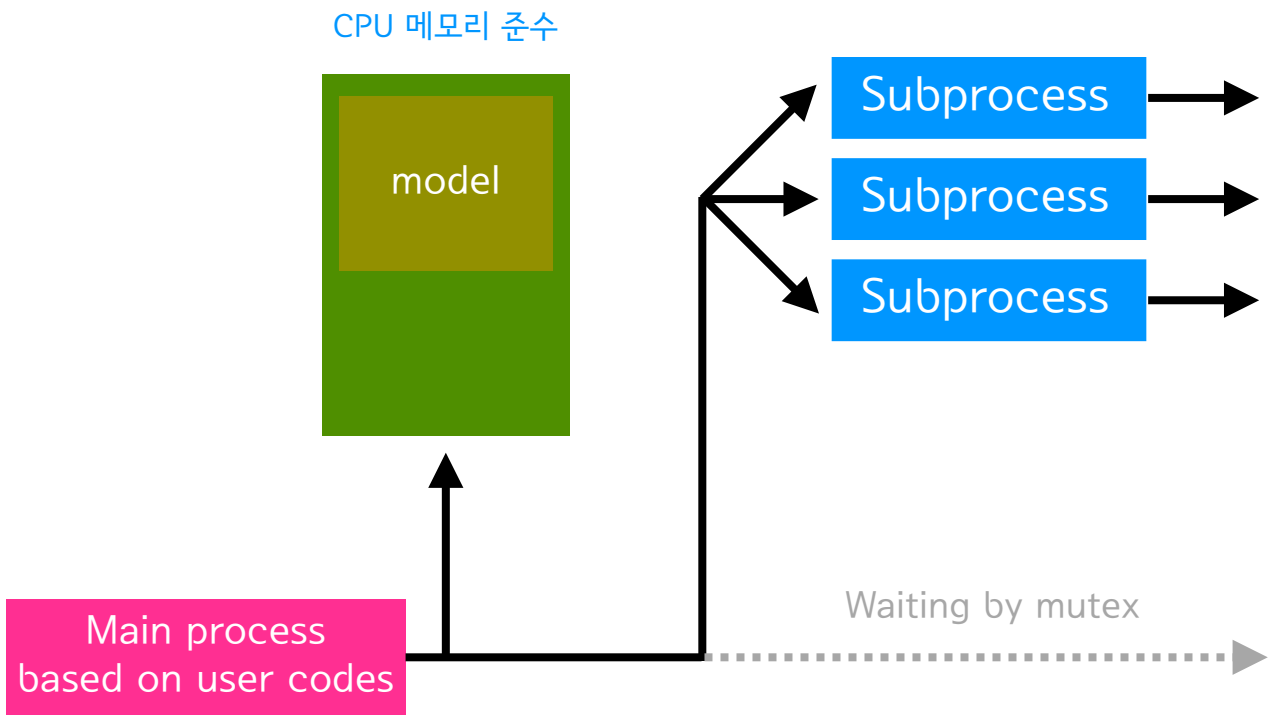
Single



Solution 2: Deployment - Inversion of process control


```python
from parallelformers import parallelize

parallelize(model, num_gpus=2, fp16=True, verbose='detail')
```

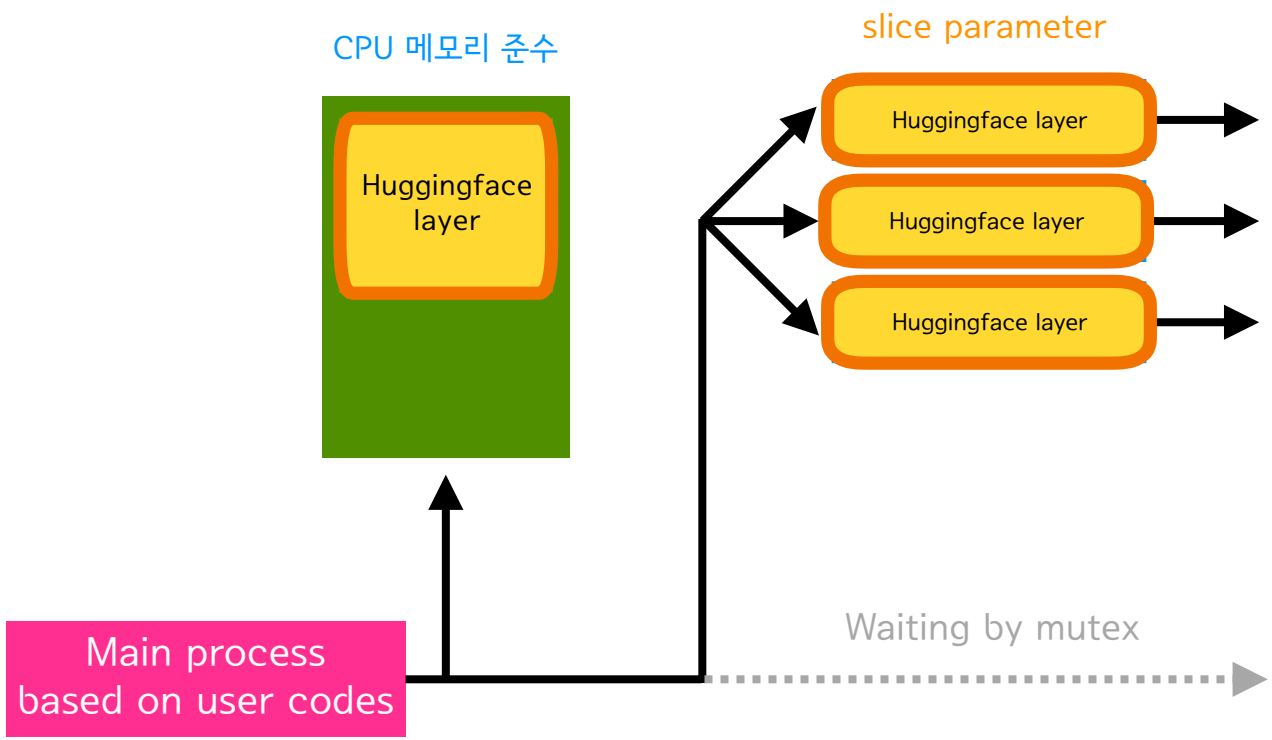
Parallel



Solution 2: Deployment - Inversion of process control


```
from parallelformers import parallelize

parallelize(model, num_gpus=2, fp16=True, verbose='detail')
```

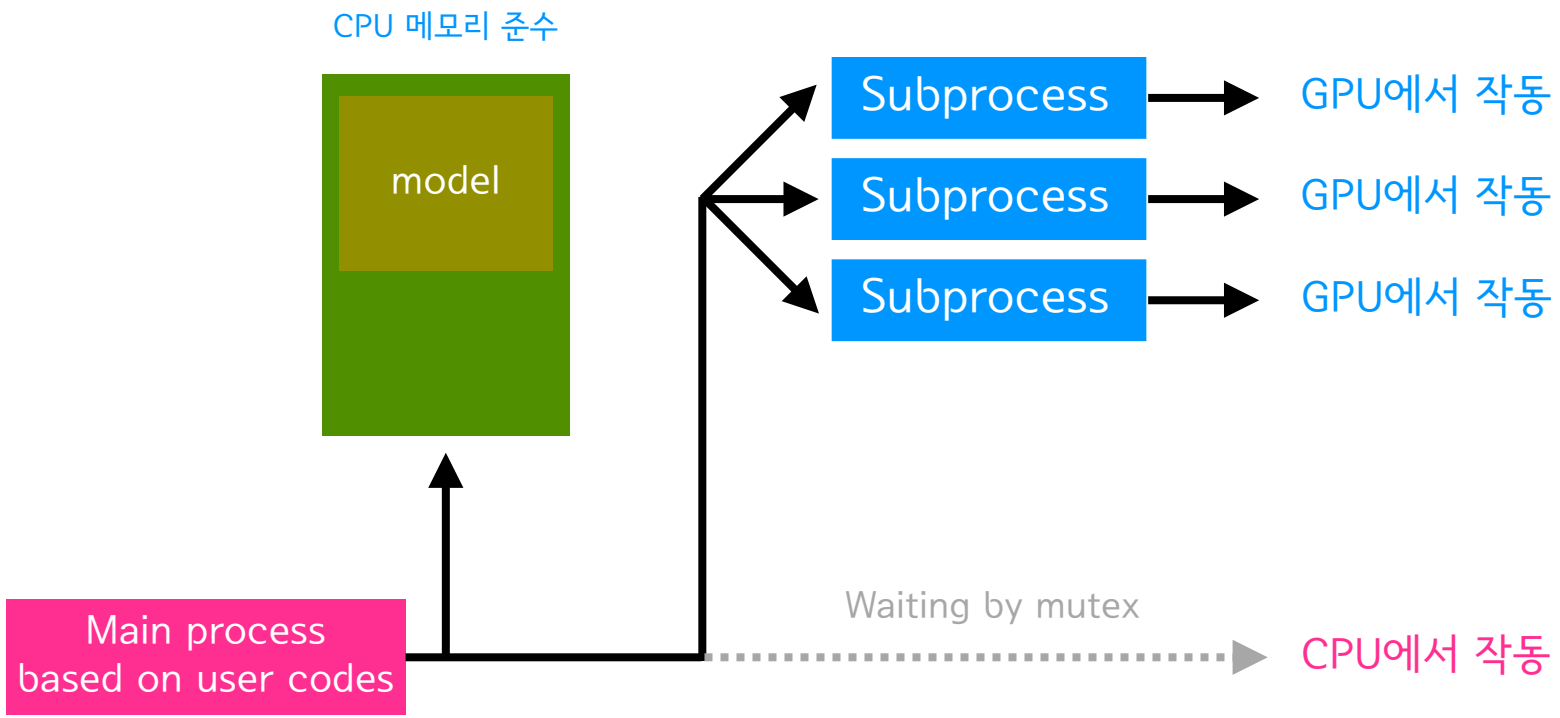
Parallel



Solution 2: Deployment - Inversion of process control


```python
from parallelformers import parallelize

parallelize(model, num_gpus=2, fp16=True, verbose='detail')
```

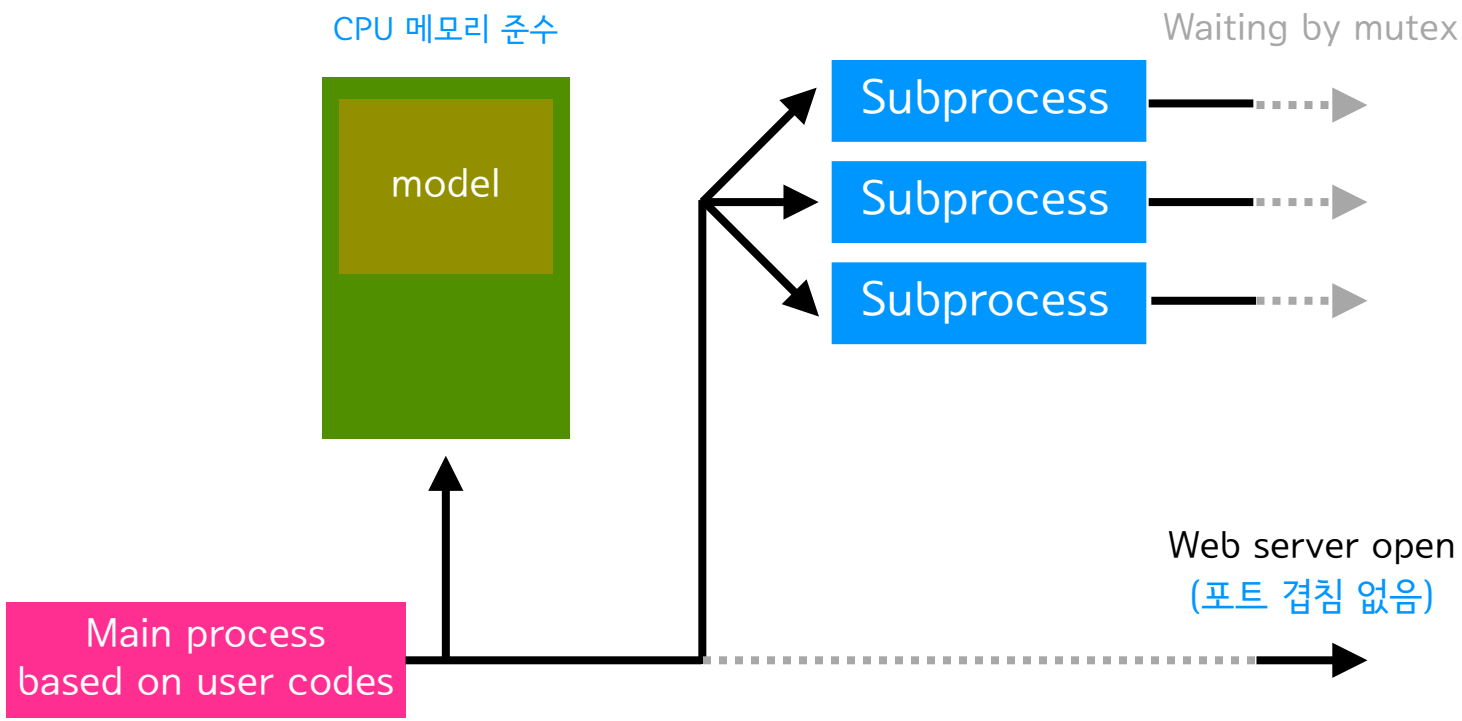
Parallel



Solution 2: Deployment - Inversion of process control

Single

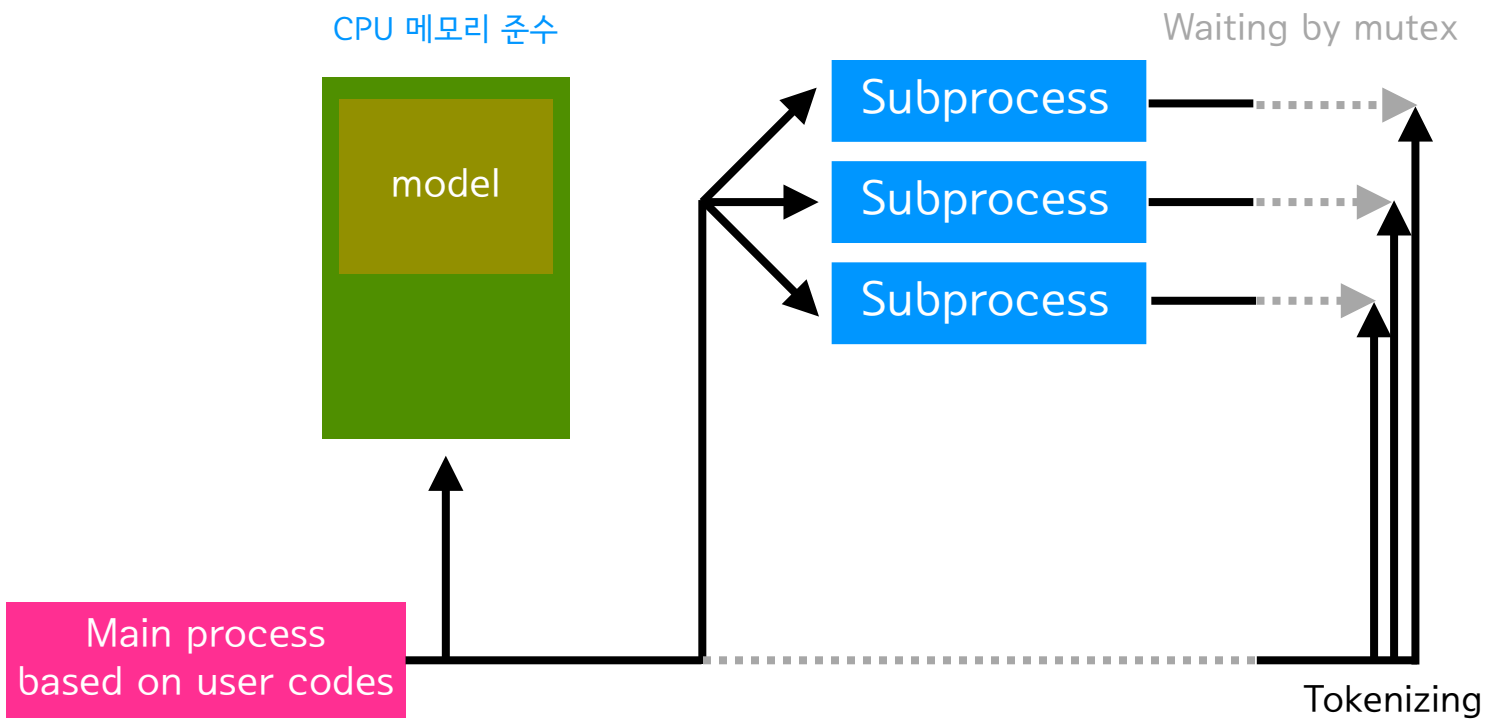
```
app.run(host="0.0.0.0", port=5000)
```



Solution 2: Deployment - Inversion of process control

Single

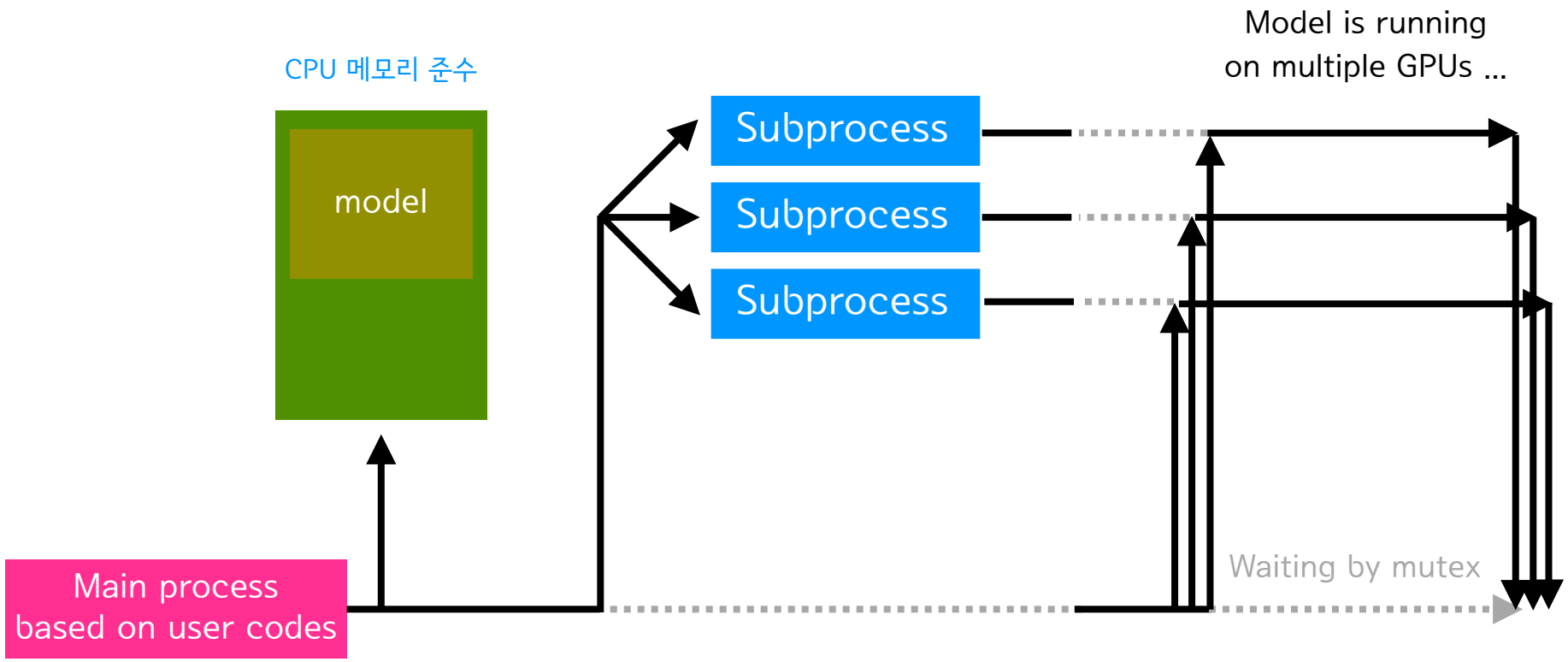
```
@app.route("/generate_text/<text>")
def generate_text(text):
    inputs = tokenizer(text, return_tensors="pt")
```



Solution 2: Deployment - Inversion of process control


```
outputs = model.generate(
    **inputs,
    num_beams=5,
    no_repeat_ngram_size=4,
    max_length=15,
)
```

Parallel

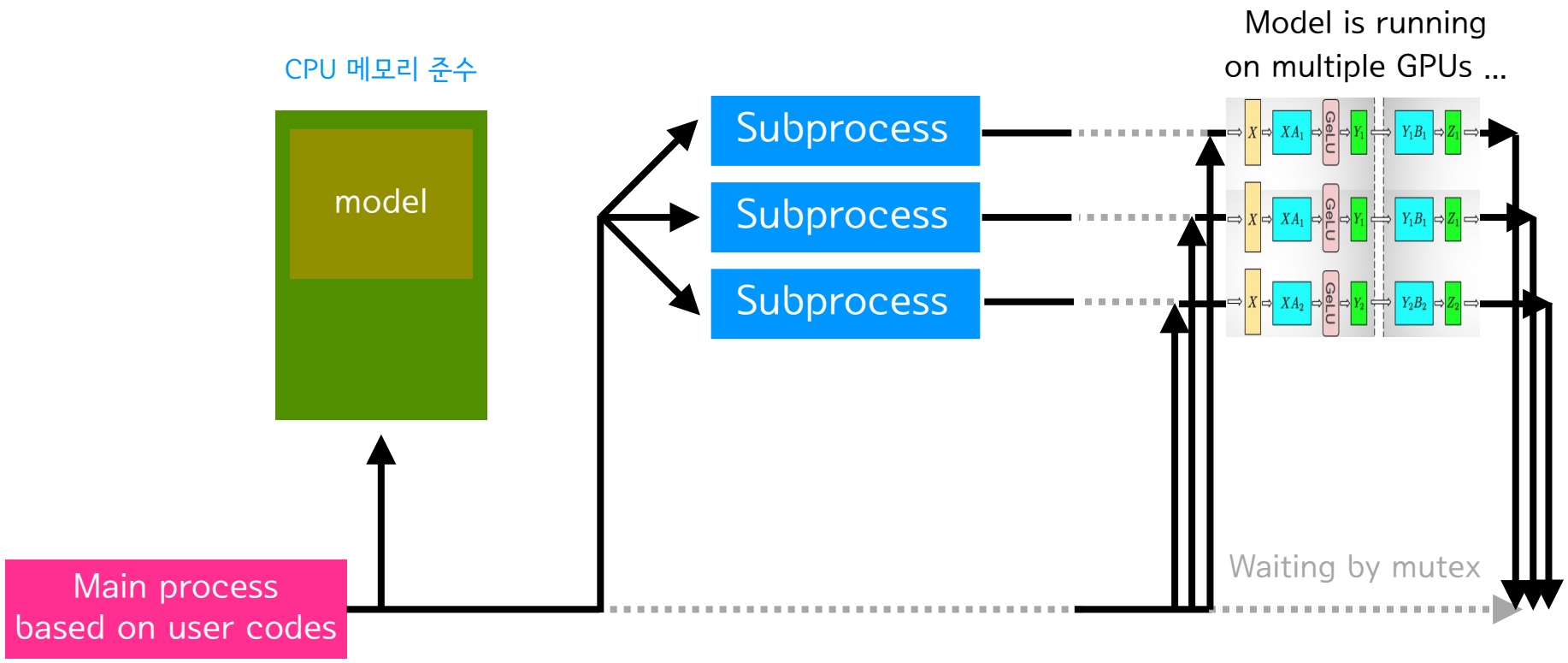


Solution 2: Deployment - Inversion of process control

Language Conference

```
outputs = model.generate(
    **inputs,
    num_beams=5,
    no_repeat_ngram_size=4,
    max_length=15,
)
```

Parallel

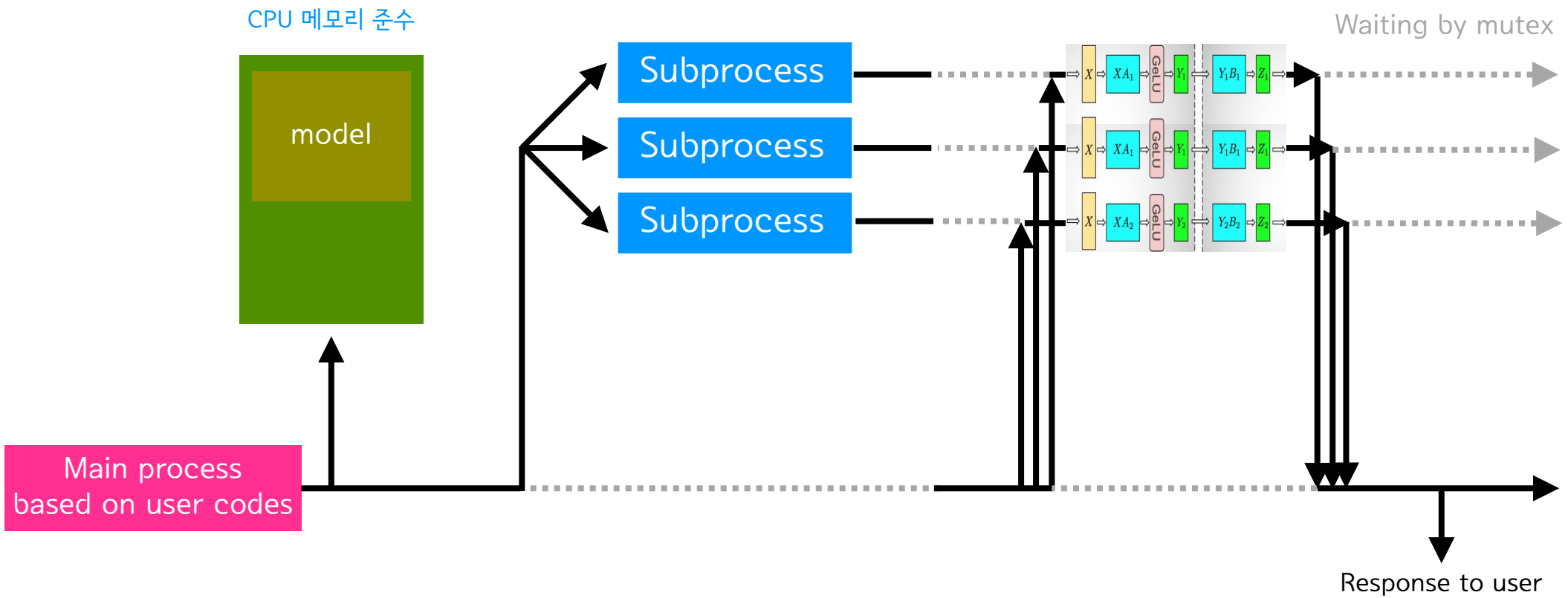


Solution 2: Deployment - Inversion of process control

Language Conference

```
outputs = tokenizer.batch_decode(
    outputs,
    skip_special_tokens=True,
)

return {
    "inputs": text,
    "outputs": outputs[0],
}
```

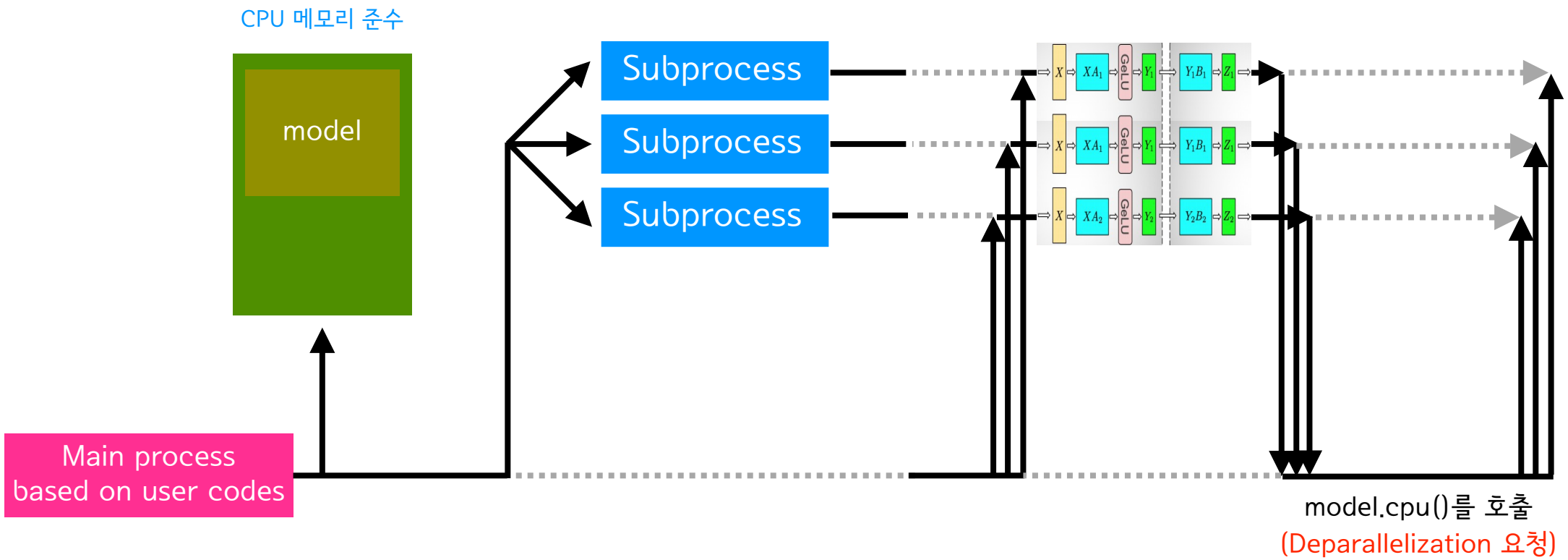
Single



Solution 2: Deployment - Inversion of process control

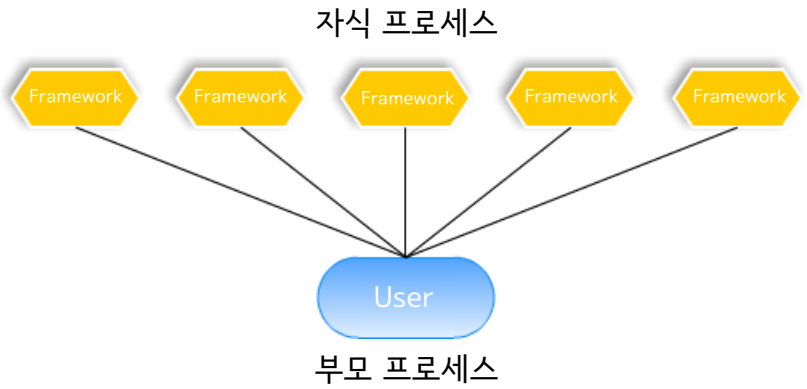
Language Conference

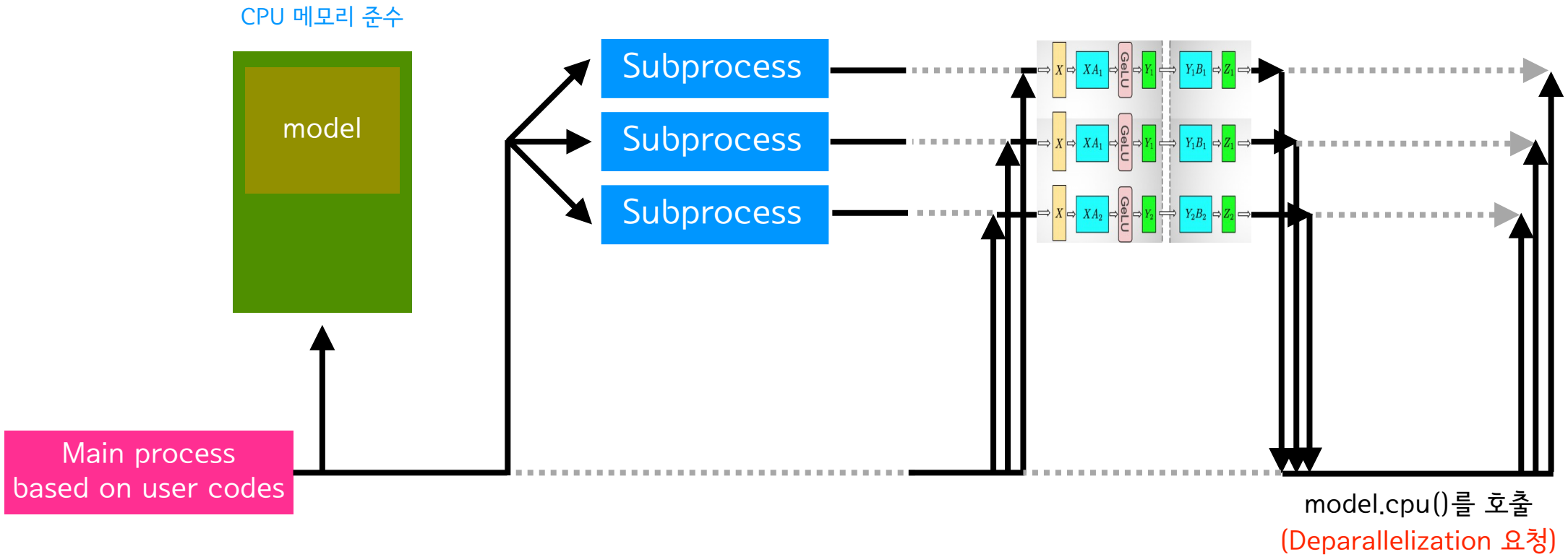
Parallelformers can deparallelize !



Solution 2: Deployment - Inversion of process control

Language Conference

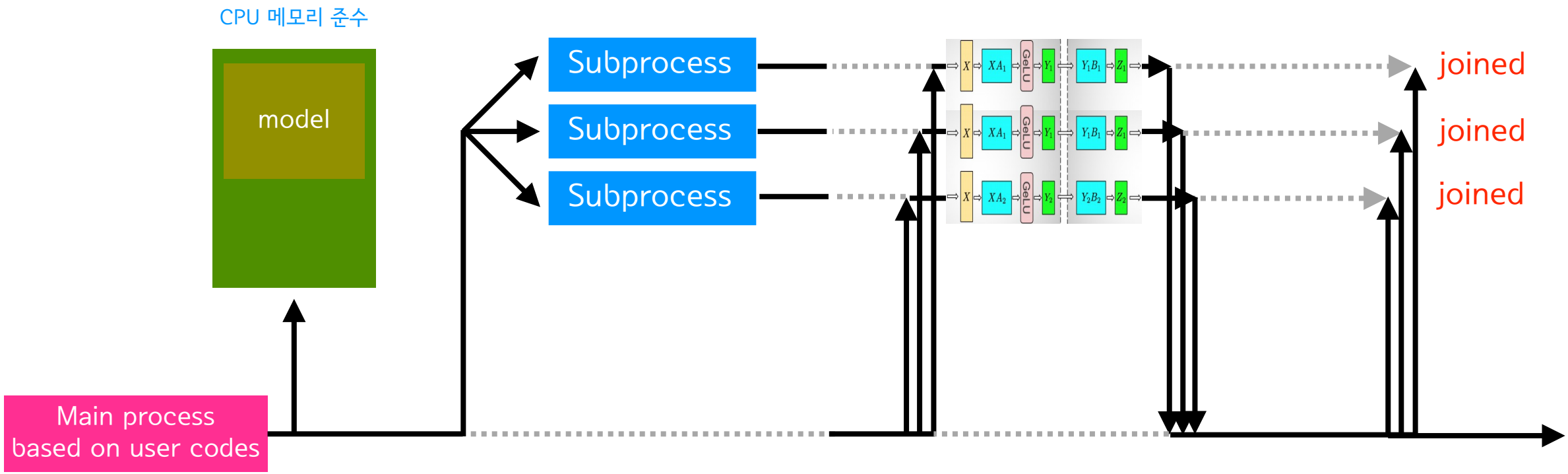




Solution 2: Deployment - Inversion of process control

Language Conference

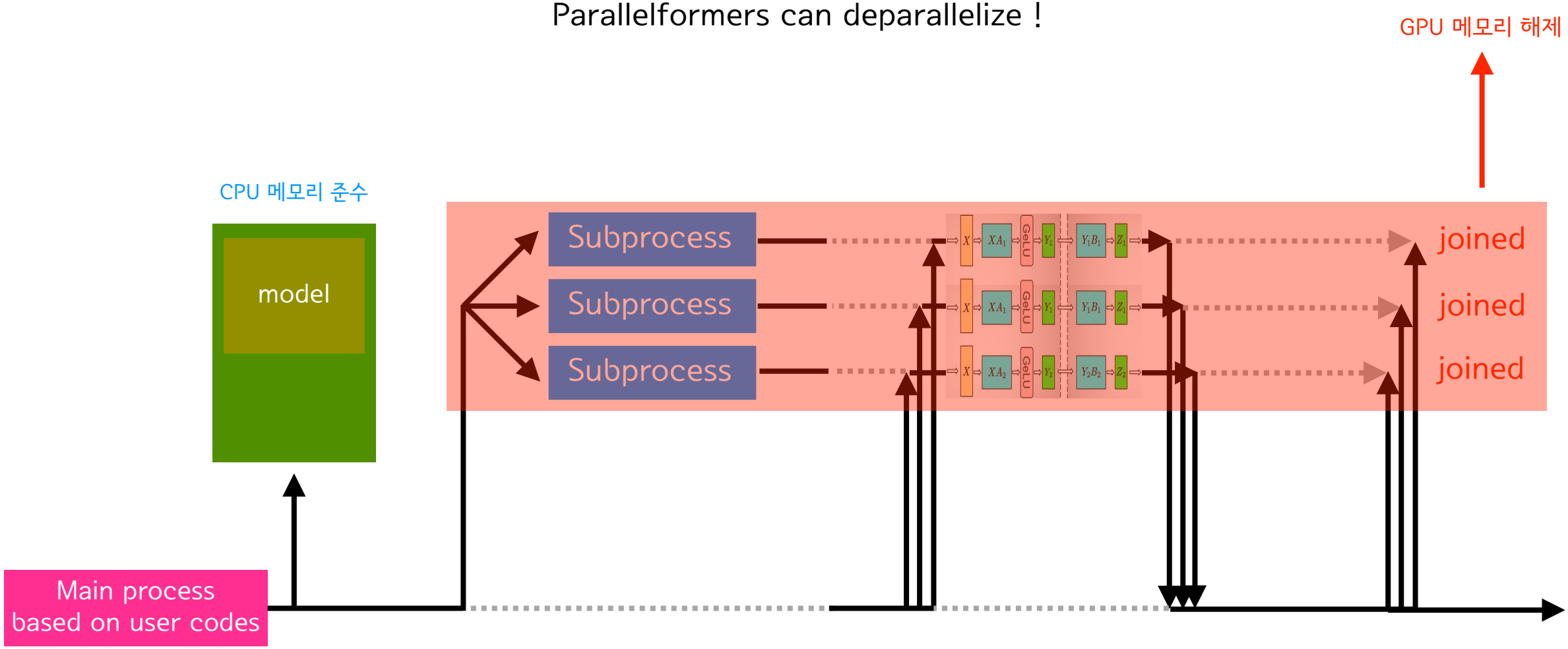
Parallelformers can deparallelize !



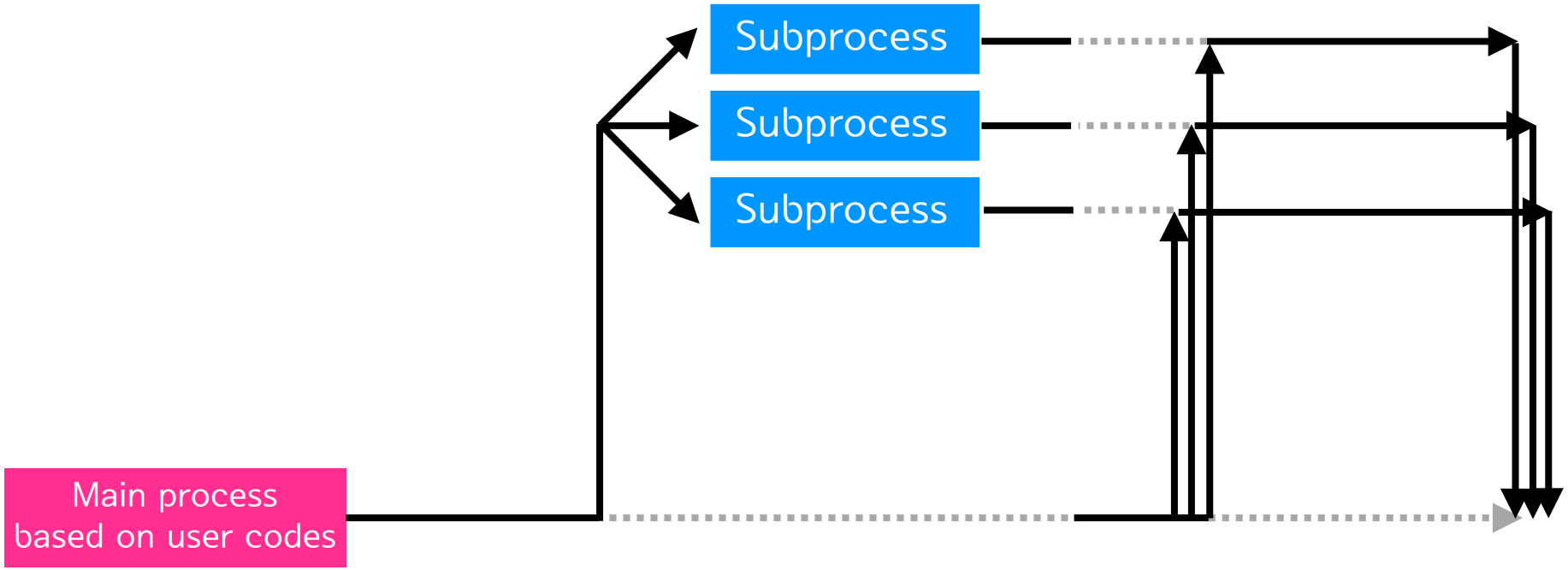
Solution 2: Deployment - Inversion of process control

Language Conference

Parallelformers can deparallelize !

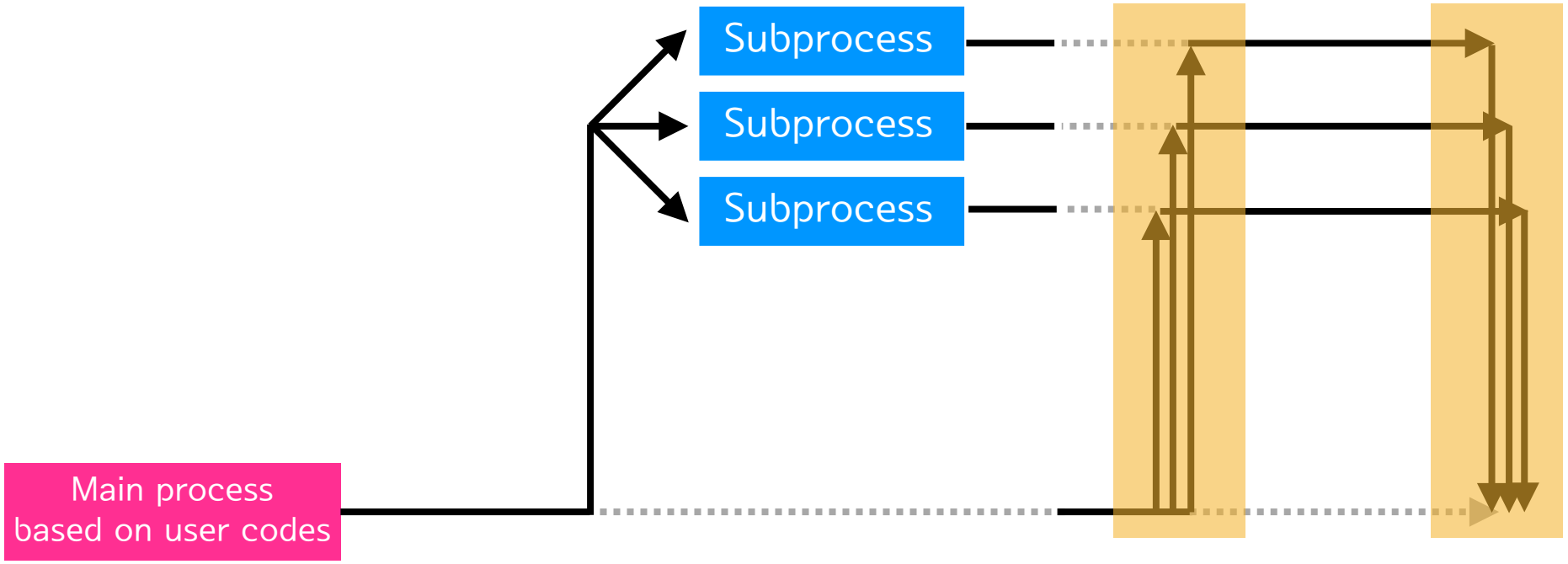


Solution 2: Deployment - Inversion of process control



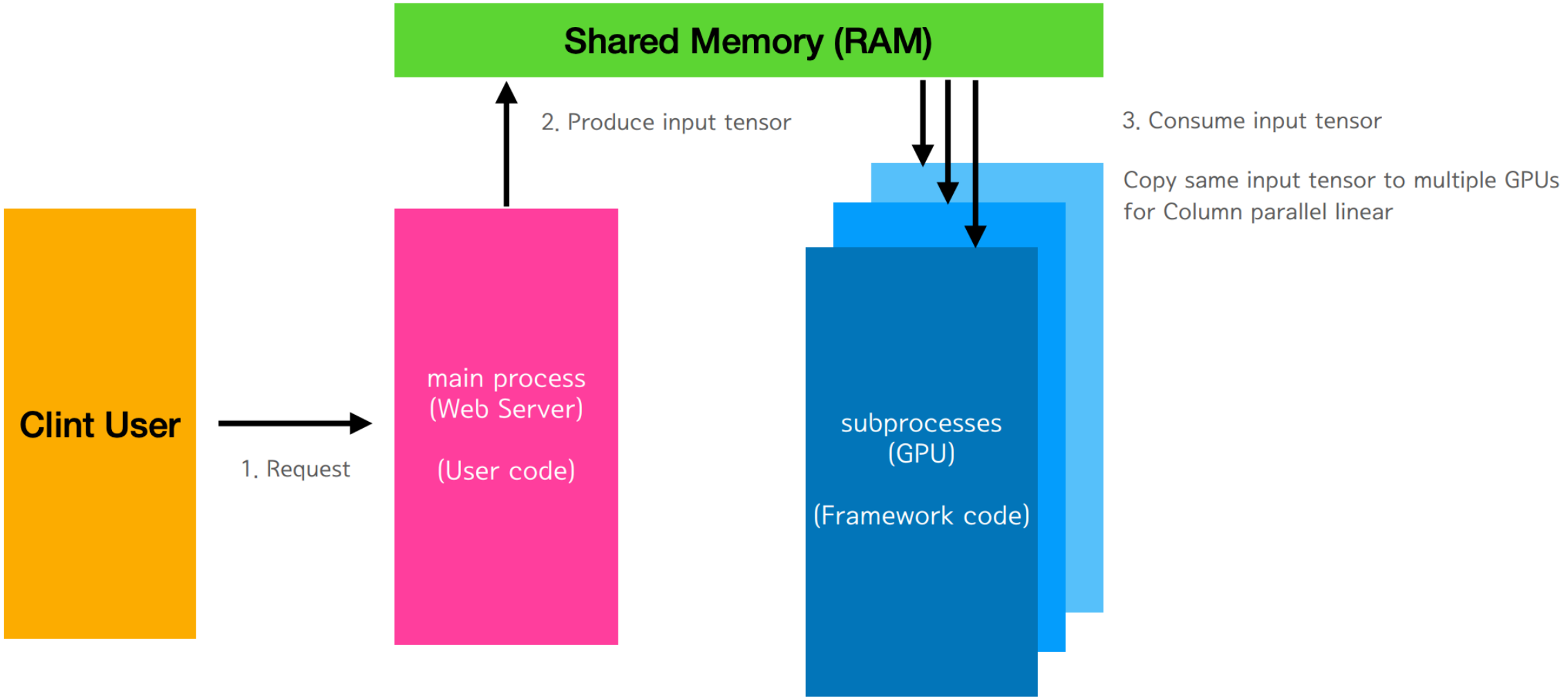
Process 간 데이터 전송은 어떻게 할까?

Solution 2: Deployment - Inversion of process control

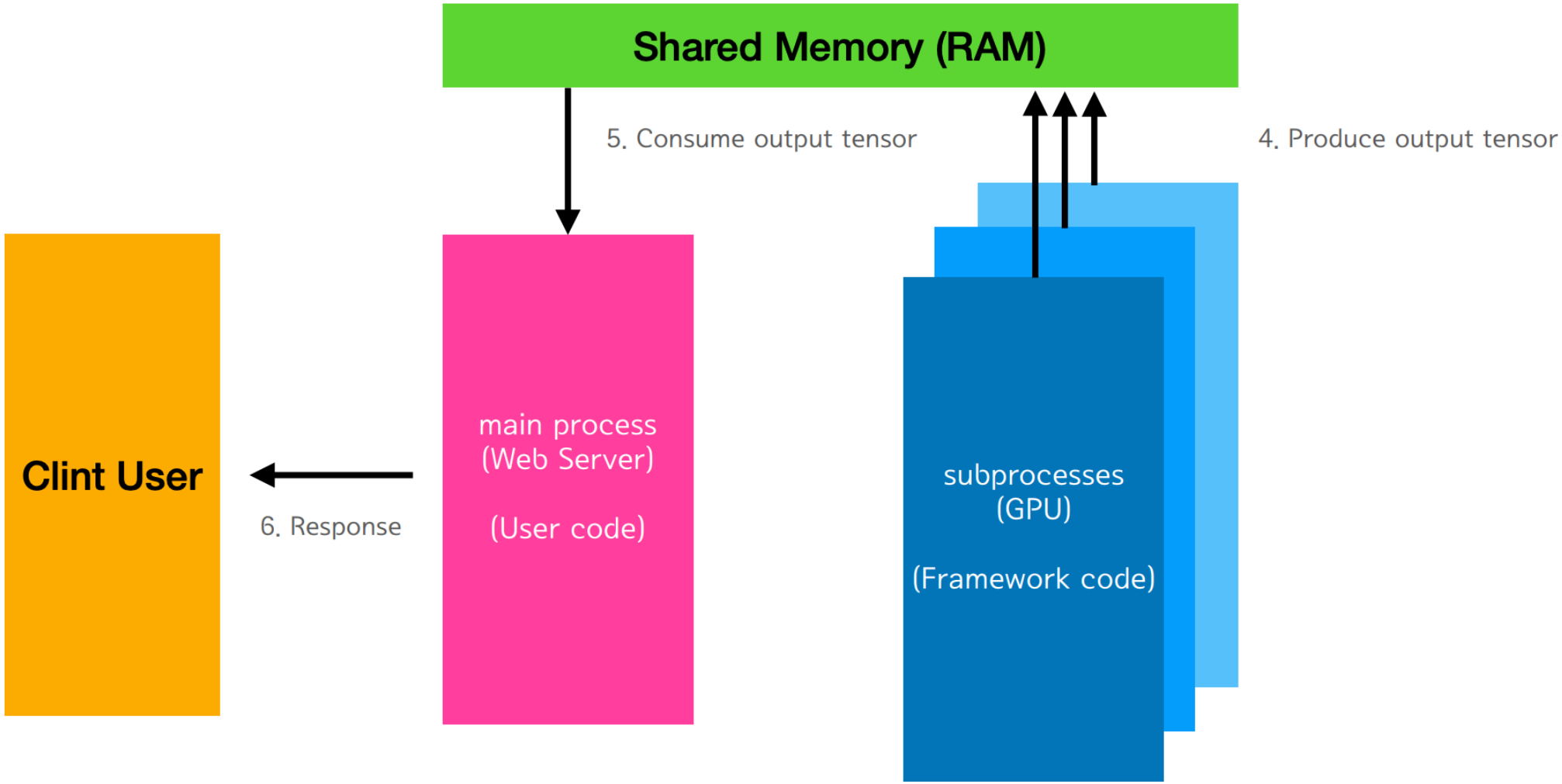


Process 간 데이터 전송은 어떻게 할까?

Solution 2: Deployment - Inversion of process control

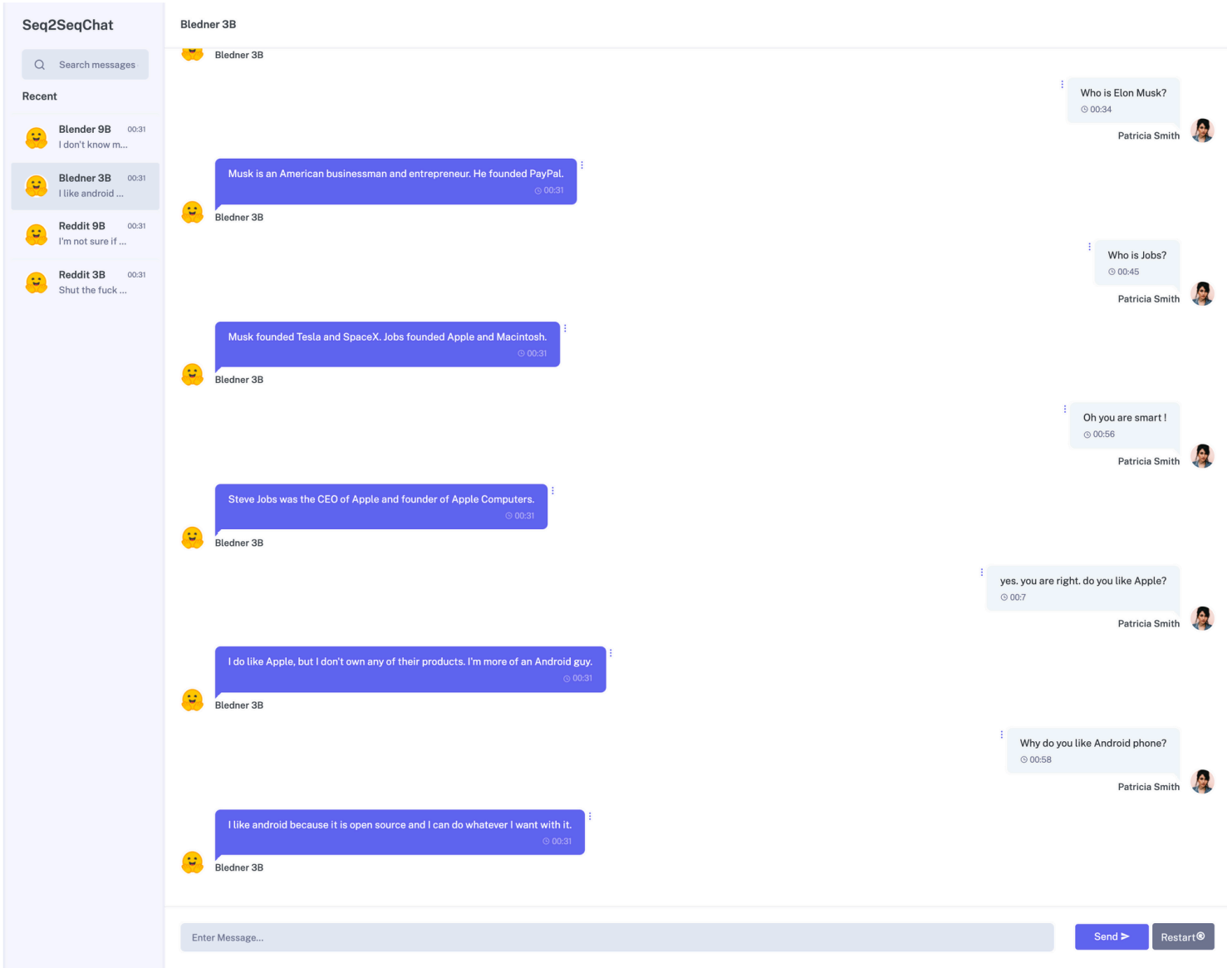


Solution 2: Deployment - Inversion of process control



Solution 2: Deployment - Inversion of process control

Language Conference



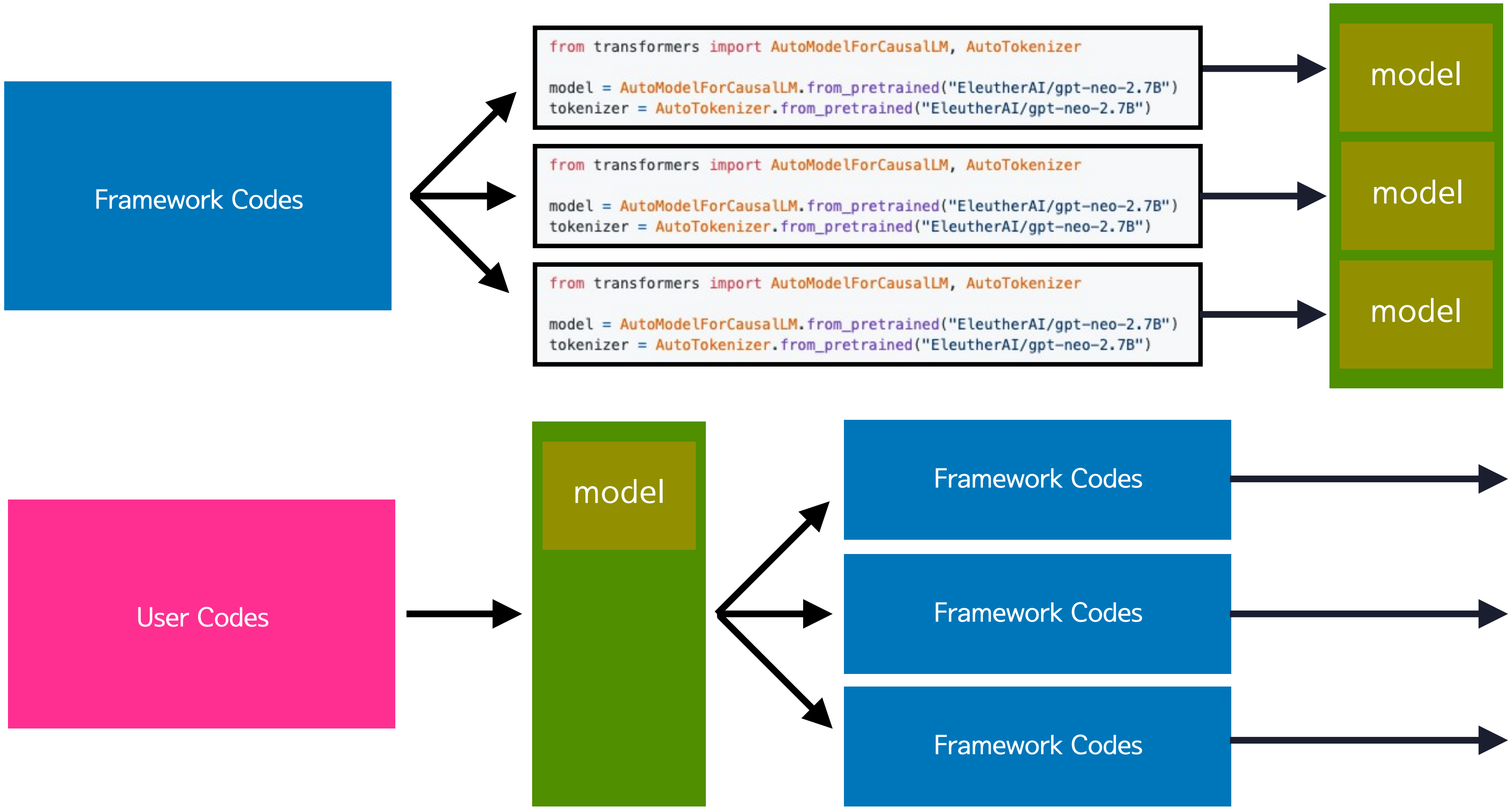
병렬화된 모델을 성공적으로

웹서버에 배포할 수 있었음 🎉🎉

Solution 3: Memory Inefficiency - Lazy GPU Allocation

Solution 3: Memory Inefficiency - Lazy GPU Allocation

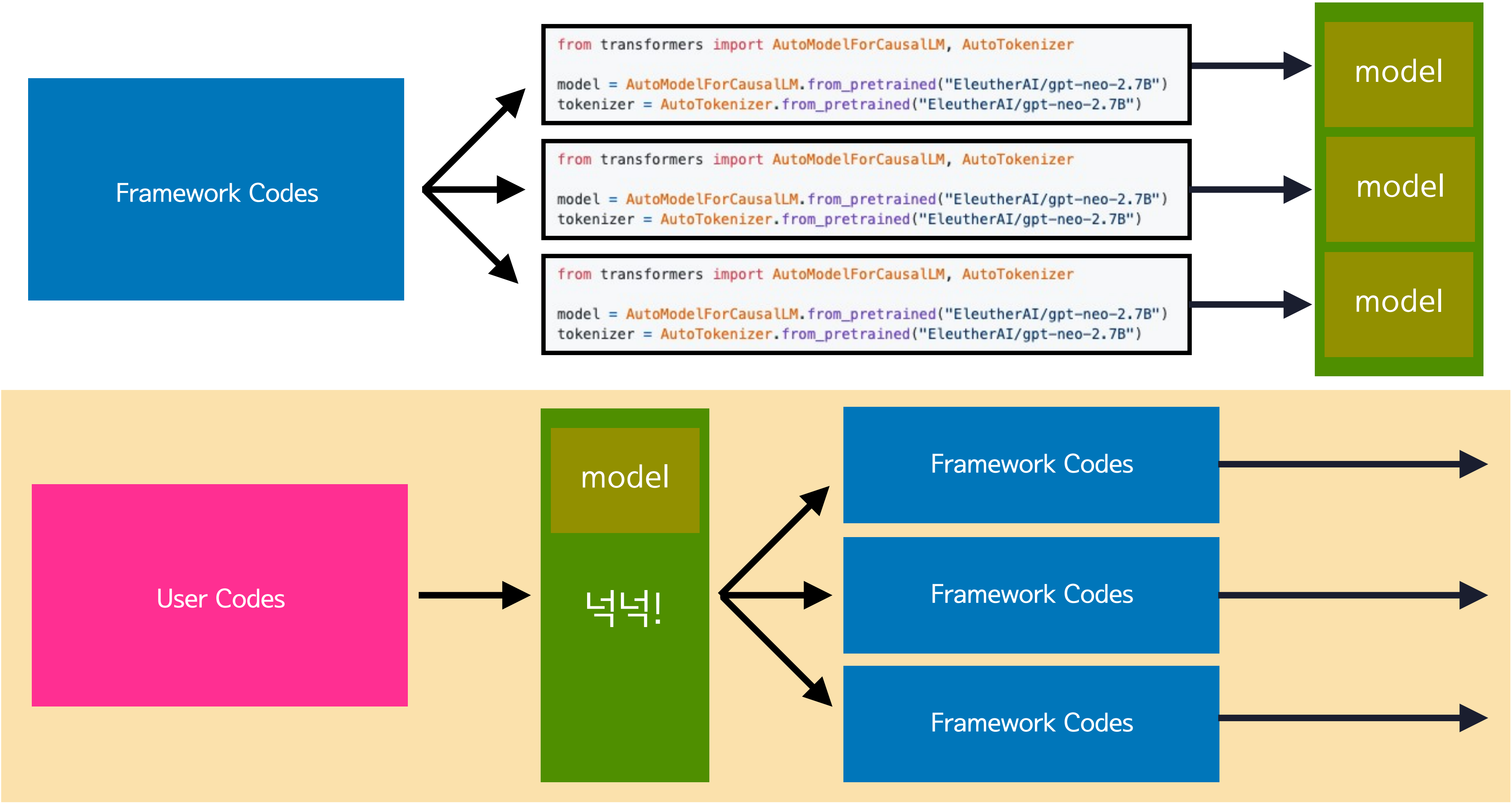
Language Conference



CPU 메모리 비효율성은 Inversion of process control로 이미 해결하였음.

Solution 3: Memory Inefficiency - Lazy GPU Allocation

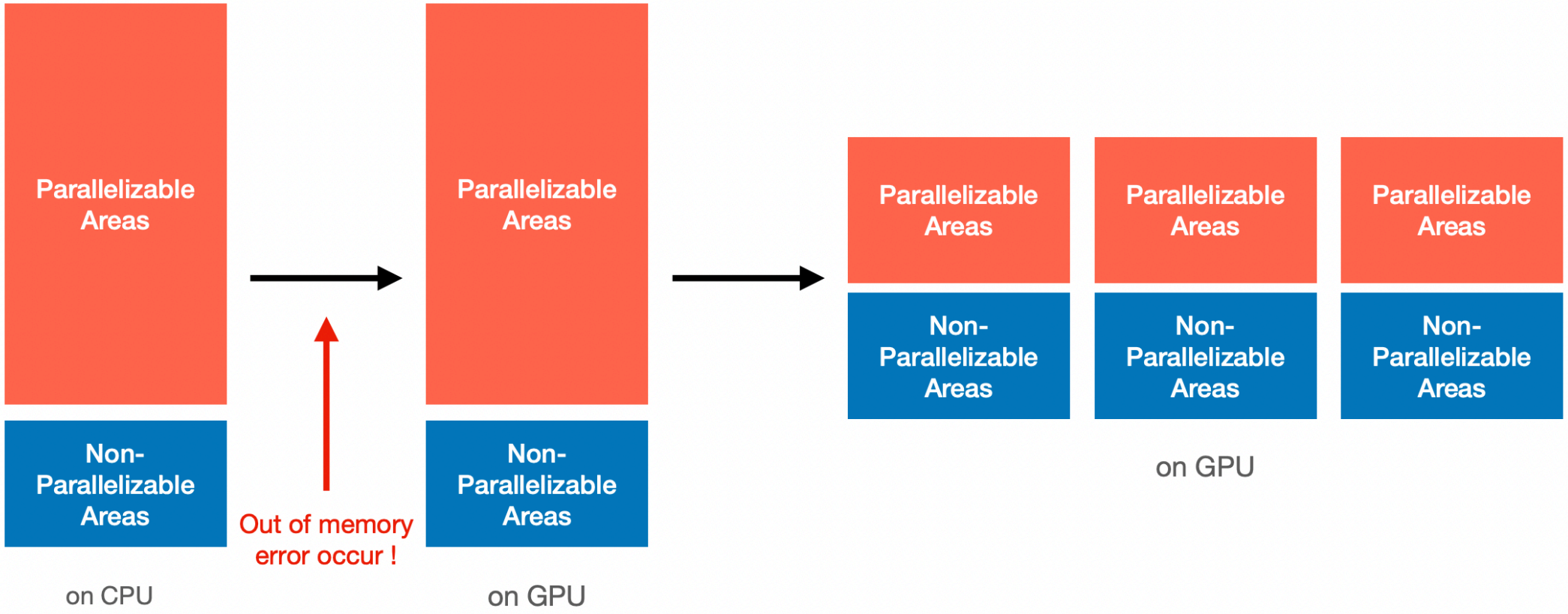
Language Conference



CPU 메모리 비효율성은 Inversion of process control로 이미 해결하였음.

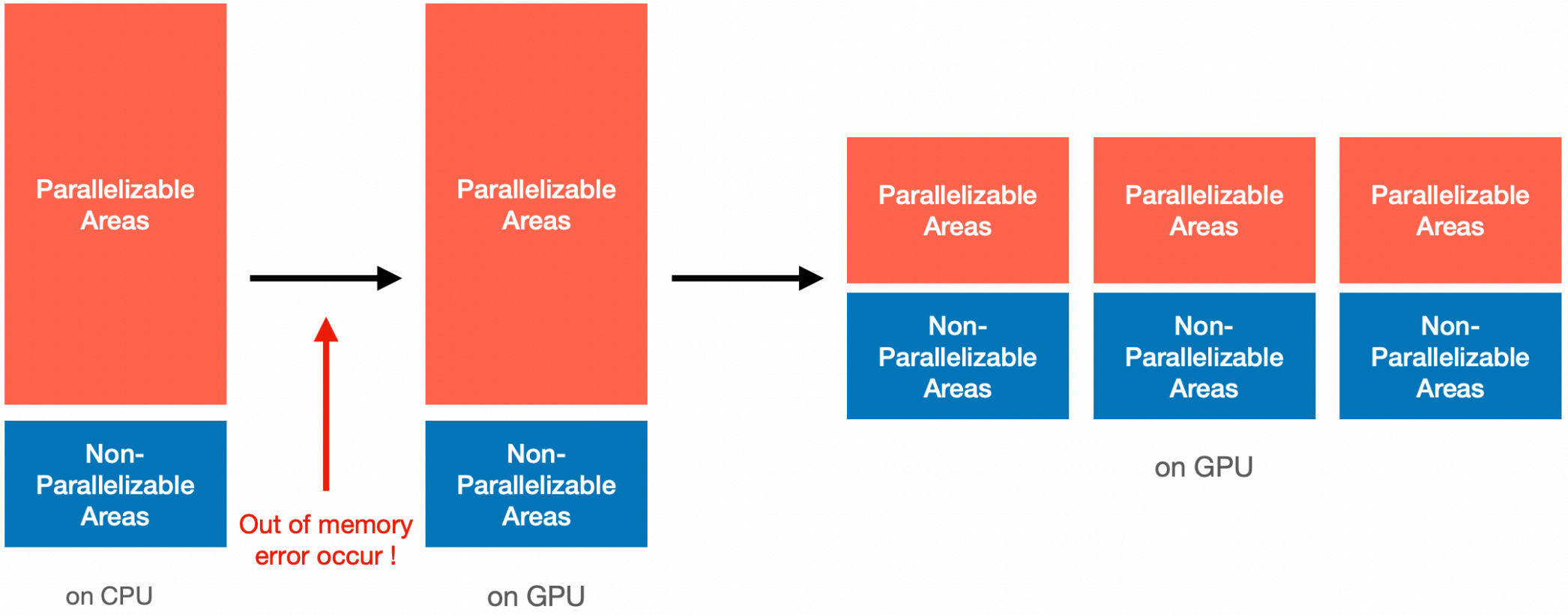
Solution 3: Memory Inefficiency - Lazy GPU Allocation

Language Conference



DeepSpeed-inference는 모델의 모든 파라미터를 GPU에 올려놓고나서 병렬화를 시작

Solution 3: Memory Inefficiency - Lazy GPU Allocation

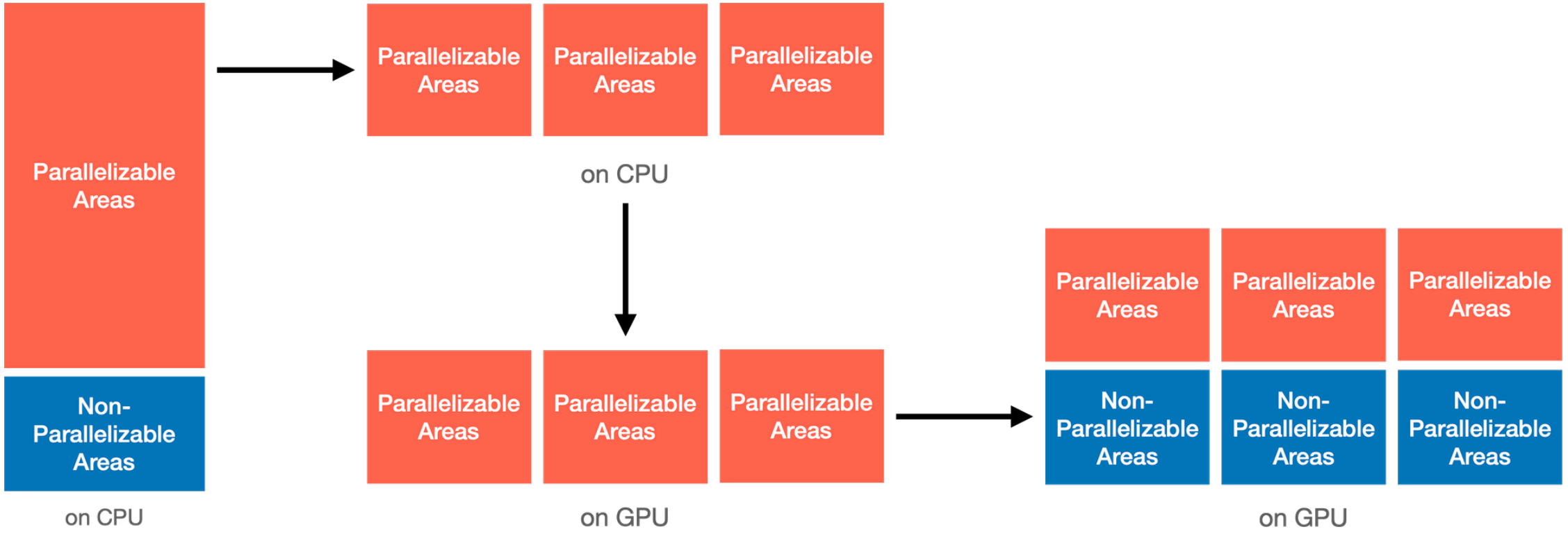


DeepSpeed-inference는 모델의 모든 파라미터를 GPU에 올려놓고나서 병렬화를 시작

→ 정작 GPU 메모리가 부족할 때는 병렬화가 불가능함. (애초에 이게 가능했다면 병렬화를 할 필요가...)

Solution 3: Memory Inefficiency - Lazy GPU Allocation

Language Conference



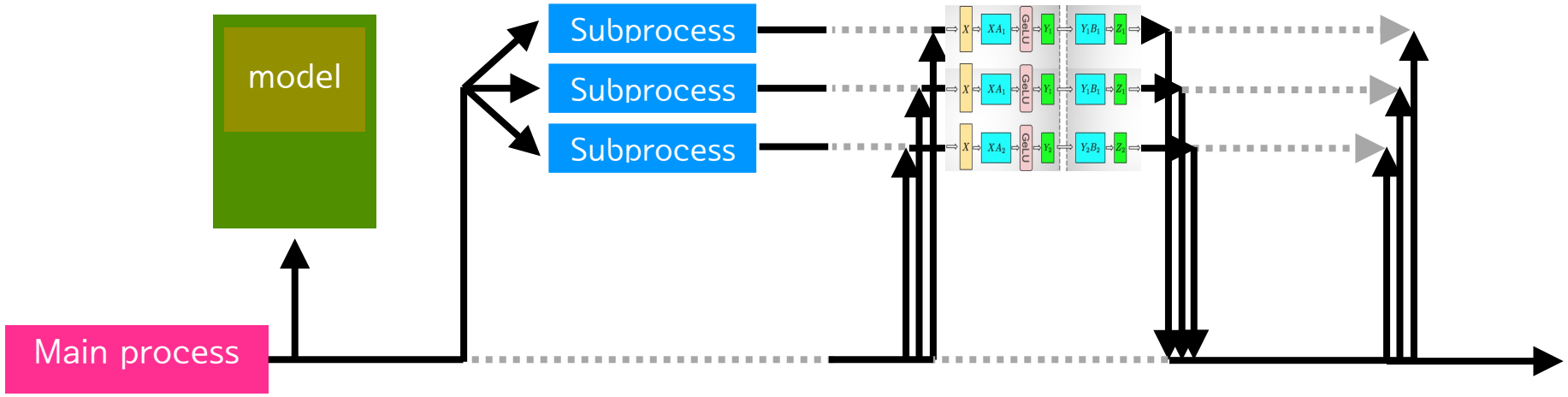
Parallelformers는 모델의 모든 파라미터를 CPU에 올려놓고나서 병렬화를 시작

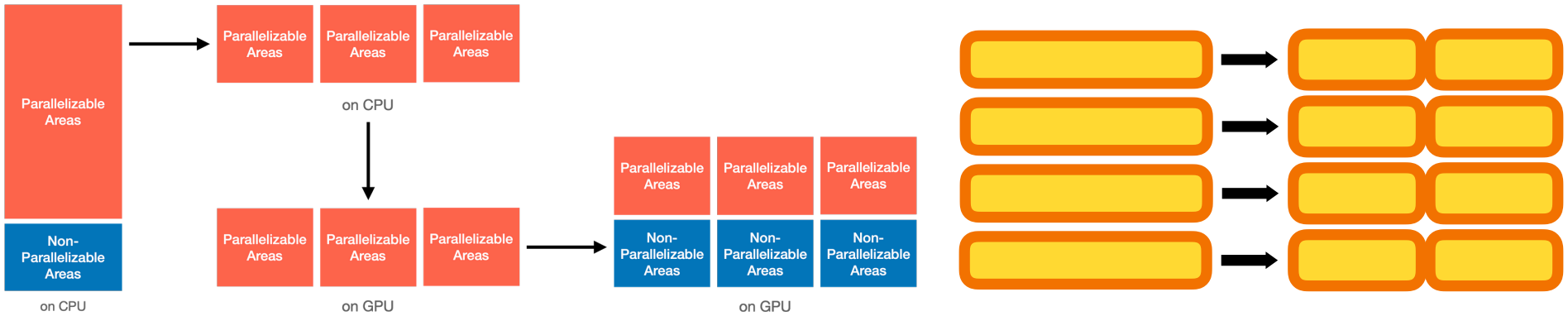
→ CPU 메모리는 일반적으로 GPU 메모리보다 크기 때문에 커다란 모델도 작은 GPU로 처리 가능

Solution 4: Simplicity - Method Hijacking

Solution 4: Simplicity - Method Hijacking

Language Conference





사용자한테 이런 것들을 설명해주고 직접 하라고 하면 사용하기 매우 어려워짐

좋은 도구라면 사용자가 구현에 대해 아예 모르더라도 손쉽게 원하는 기능을 이용할 수 있어야 함

Solution 4: Simplicity - Method Hijacking

Language Conference

```python
from transformers import AutoModelForCausalLM, AutoTokenizer

model = AutoModelForCausalLM.from_pretrained("EleutherAI/gpt-neo-2.7B")
tokenizer = AutoTokenizer.from_pretrained("EleutherAI/gpt-neo-2.7B")

from flask import Flask

app = Flask(__name__)


@app.route("/generate_text/<text>")
def generate_text(text):
    inputs = tokenizer(text, return_tensors="pt")

    outputs = model.generate(
        **inputs,
        num_beams=5,
        no_repeat_ngram_size=4,
        max_length=15,
    )

    outputs = tokenizer.batch_decode(
        outputs,
        skip_special_tokens=True,
    )

    return {
        "inputs": text,
        "outputs": outputs[0],
    }


app.run(host="0.0.0.0", port=5000)
```

사용자는 Parallelformers를 사용하는 방법을 배울 필요가 없음

(import 제외) 코드 1줄만 실행하면 모든 처리가 자동으로 진행되어 병렬화가 완료됨.

Solution 4: Simplicity - Method Hijacking

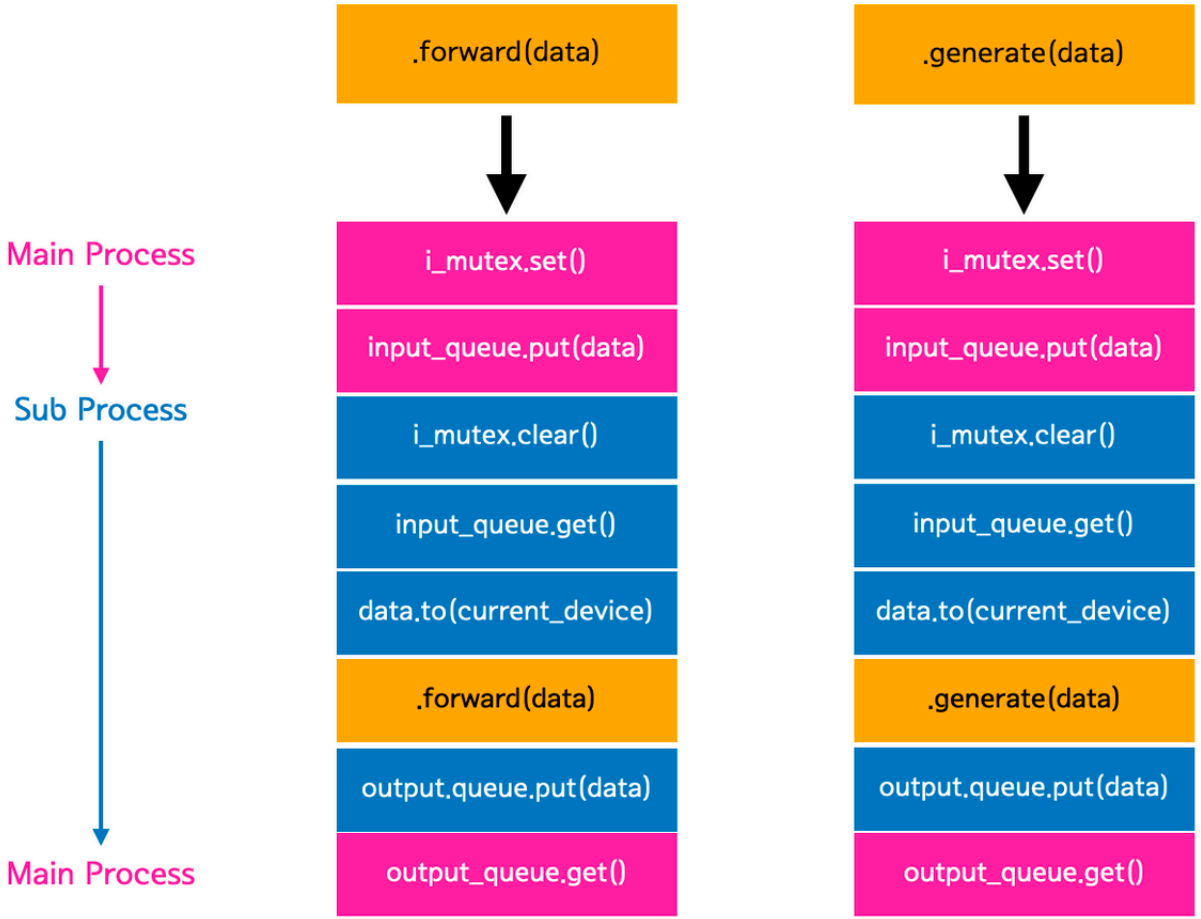
Language Conference

```python
from transformers import AutoModelForCausalLM, AutoTokenizer

model = AutoModelForCausalLM.from_pretrained("EleutherAI/gpt-neo-2.7B")
tokenizer = AutoTokenizer.from_pretrained("EleutherAI/gpt-neo-2.7B")

from parallelformers import parallelize

parallelize(model, num_gpus=2, fp16=True, verbose='detail')

from flask import Flask

app = Flask(__name__)


@app.route("/generate_text/<text>")
def generate_text(text):
    inputs = tokenizer(text, return_tensors="pt")

    outputs = model.generate(
        **inputs,
        num_beams=5,
        no_repeat_ngram_size=4,
        max_length=15,
    )

    outputs = tokenizer.batch_decode(
        outputs,
        skip_special_tokens=True,
    )

    return {
        "inputs": text,
        "outputs": outputs[0],
    }


app.run(host="0.0.0.0", port=5000)
```

사용자는 Parallelformers를 사용하는 방법을 배울 필요가 없음

(import 제외) 코드 1줄만 실행하면 모든 처리가 자동으로 진행되어 병렬화가 완료됨.

Solution 4: Simplicity - Method Hijacking

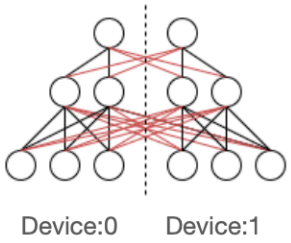


Parallelformers는 Method Hijacking (Proxy)를 통해 이러한 것을 가능하게 했음.

사용자가 기존에 사용하던 메소드를 호출하면, 코드의 흐름을 하이재킹하여 필요한 작업들을 함께 수행해줌.

Solution 4: Simplicity - Method Hijacking

Parallelformers의 4가지 Design Principles !



efficient
model parallelism

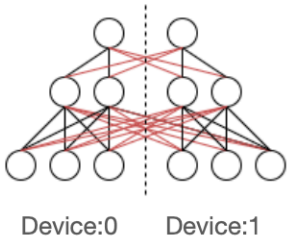
Deployment

Scalability

Simplicity

Solution 4: Simplicity - Method Hijacking

Parallelformers의 4가지 Design Principles !



Lazy GPU Allocation

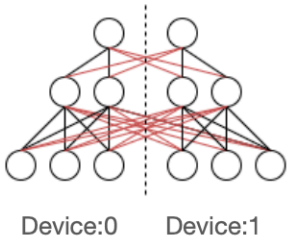
Deployment

Scalability

Simplicity

Solution 4: Simplicity - Method Hijacking

Parallelformers의 4가지 Design Principles !



Lazy GPU Allocation

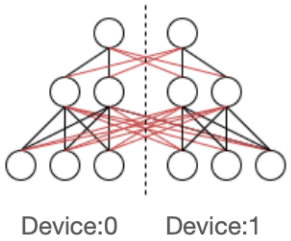
Inversion of process control

Scalability

Simplicity

Solution 4: Simplicity - Method Hijacking

Parallelformers의 4가지 Design Principles !



Lazy GPU Allocation

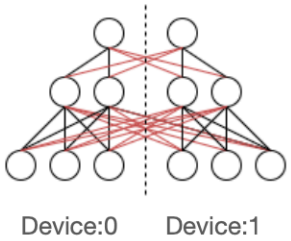
Inversion of process control

No fused CUDA kernel

Simplicity

Solution 4: Simplicity - Method Hijacking

Parallelformers의 4가지 Design Principles !



Lazy GPU Allocation

Inversion of process control

No fused CUDA kernel

Method Hijcaking

Usages & Issues

1. Create a HuggingFace transformers model.

You don't need to call `.half()` or `.cuda()` as those functions will be invoked automatically. It is more memory efficient to start parallelization on the CPU.

```
from transformers import AutoModelForCausalLM, AutoTokenizer

model = AutoModelForCausalLM.from_pretrained("EleutherAI/gpt-neo-2.7B")
tokenizer = AutoTokenizer.from_pretrained("EleutherAI/gpt-neo-2.7B")
```

2. Put the `model` in the `parallelize()` function.

```
from parallelformers import parallelize

parallelize(model, num_gpus=2, fp16=True, verbose='detail')
```

```
|===========================================================================|
|                  PyTorch CUDA memory summary, device ID 0                 |
|---------------------------------------------------------------------------|
|            CUDA OOMs: 0            |        cudaMalloc retries: 0         |
|===========================================================================|
|        Metric         | Cur Usage  | Peak Usage | Tot Alloc  | Tot Freed  |
|---------------------------------------------------------------------------|
| Allocated memory      |    2721 MB |    2967 MB |    2967 MB |  251905 KB |
|       from large pool |    2720 MB |    2966 MB |    2966 MB |  251904 KB |
|       from small pool |       1 MB |       1 MB |       1 MB |       1 KB |
|---------------------------------------------------------------------------|

GPU:0 => 2.72GB
```

```
|===========================================================================|
|                  PyTorch CUDA memory summary, device ID 1                 |
|---------------------------------------------------------------------------|
|            CUDA OOMs: 0            |        cudaMalloc retries: 0         |
|===========================================================================|
|        Metric         | Cur Usage  | Peak Usage | Tot Alloc  | Tot Freed  |
|---------------------------------------------------------------------------|
| Allocated memory      |    2721 MB |    2967 MB |    2967 MB |  251905 KB |
|       from large pool |    2720 MB |    2966 MB |    2966 MB |  251904 KB |
|       from small pool |       1 MB |       1 MB |       1 MB |       1 KB |
|---------------------------------------------------------------------------|

GPU:1 => 2.72GB
```

3. Do Inference as usual.

You don't have to call `.cuda()` when creating input tokens. **Note that you should input both input tokens and attention masks to the model.** (`**inputs` is the recommended way for this)

```
inputs = tokenizer("Parallelformers is", return_tensors="pt")

outputs = model.generate(
    **inputs,
    num_beams=5,
    no_repeat_ngram_size=4,
    max_length=15,
)

print(f"Output: {tokenizer.batch_decode(outputs)[0]}")
```

```
Output: Parallelformers is an open-source library for parallel programming ...
```

4. Deploy the model to the server as usual.

The parallelization process does not affect the web server because they are automatically synchronized.

```
from flask import Flask

app = Flask(__name__)


@app.route("/generate_text/<text>")
def generate_text(text):
    inputs = tokenizer(text, return_tensors="pt")

    outputs = model.generate(
        **inputs,
        num_beams=5,
        no_repeat_ngram_size=4,
        max_length=15,
    )

    outputs = tokenizer.batch_decode(
        outputs,
        skip_special_tokens=True,
    )

    return {
        "inputs": text,
        "outputs": outputs[0],
    }


app.run(host="0.0.0.0", port=5000)
```

You can send a request to the web server as follows:

```
$ curl -X get "YOUR_IP:5000/generate_text/Messi"
```

And the following result should be returned.

```
{"inputs": "Messi", "outputs": "Messi is the best player in the world right now. He is the"}
```

5. Check the current GPU states.

You can check GPU states using `.memory_allocated()`, `.memory_reserved()` and `.memory_chached()` to make sure the parallelization is successful.

```
model.memory_allocated()
model.memory_reserved()
model.memory_chached()
```

```
{'cuda:0':XXXXXX, 'cuda:1':XXXXXX}
```

6. Manage the model parallelization states.

You can manage model parallelization states using `.cuda()`, `.cpu()` and `.to()`. **The model parallelization process ends if you call those functions.**

```
model.cuda()

print(torch.cuda.memory_summary(0))
print(torch.cuda.memory_summary(1))
```

Check the allocated memory status using `torch.cuda.memory_summary()`.

```
|===========================================================================|
|                  PyTorch CUDA memory summary, device ID 0                 |
|---------------------------------------------------------------------------|
|            CUDA OOMs: 0            |        cudaMalloc retries: 0         |
|===========================================================================|
|        Metric         | Cur Usage  | Peak Usage | Tot Alloc  | Tot Freed  |
|---------------------------------------------------------------------------|
| Allocated memory      |    5121 MB |    5121 MB |    5121 MB |    1024 B  |
|       from large pool |    5120 MB |    5120 MB |    5120 MB |       0 B  |
|       from small pool |       1 MB |       1 MB |       1 MB |    1024 B  |
|---------------------------------------------------------------------------|

GPU0 => 5.12GB
```

```
|===========================================================================|
|                  PyTorch CUDA memory summary, device ID 1                 |
|---------------------------------------------------------------------------|
|            CUDA OOMs: 0            |        cudaMalloc retries: 0         |
|===========================================================================|
|        Metric         | Cur Usage  | Peak Usage | Tot Alloc  | Tot Freed  |
|---------------------------------------------------------------------------|
| Allocated memory      |       0 B  |    1024 B  |    1024 B  |    1024 B  |
|       from large pool |       0 B  |       0 B  |       0 B  |       0 B  |
|       from small pool |       0 B  |    1024 B  |    1024 B  |    1024 B  |
|---------------------------------------------------------------------------|

GPU1 => 0.00GB
```

If you switch to the CPU mode, it works like this.

```
model.cpu()

print(torch.cuda.memory_summary(0))
print(torch.cuda.memory_summary(1))
```

```
|===========================================================================|
|                  PyTorch CUDA memory summary, device ID 0                 |
|---------------------------------------------------------------------------|
|            CUDA OOMs: 0            |        cudaMalloc retries: 0         |
|===========================================================================|
|        Metric         | Cur Usage  | Peak Usage | Tot Alloc  | Tot Freed  |
|---------------------------------------------------------------------------|
| Allocated memory      |       0 B  |    5121 MB |    5121 MB |    5121 MB |
|       from large pool |       0 B  |    5120 MB |    5120 MB |    5120 MB |
|       from small pool |       0 B  |       1 MB |       1 MB |       1 MB |
|---------------------------------------------------------------------------|

GPU0 => 0.00GB
```

```
|===========================================================================|
|                  PyTorch CUDA memory summary, device ID 1                 |
|---------------------------------------------------------------------------|
|            CUDA OOMs: 0            |        cudaMalloc retries: 0         |
|===========================================================================|
|        Metric         | Cur Usage  | Peak Usage | Tot Alloc  | Tot Freed  |
|---------------------------------------------------------------------------|
| Allocated memory      |       0 B  |    1024 B  |    1024 B  |    1024 B  |
|       from large pool |       0 B  |       0 B  |       0 B  |       0 B  |
|       from small pool |       0 B  |    1024 B  |    1024 B  |    1024 B  |
|---------------------------------------------------------------------------|

GPU1 => 0.00GB
```

Q. Why doesn't the GPU usage decrease by exactly *n* times when I parallelize on *n* GPUs?

There are three possible reasons.

1. There are non-parallelizable areas in the model. For example, embedding, normalization and lm head layers can NOT be parallelized, resulting that they are copied to all GPUs.
2. We need to allocate shared memory areas for inter-process communication. Since this shared memory is allocated across all GPU processes, the GPU usage should increase.
3. When input tensors are copied to all GPUs, the GPU usage can increase.

Q. How many GPUs are good to use when parallelizing a model?

We recommend you keep the number of GPUs as least as possible.

Q. Can I parallelize multiple models on same GPUs?

Yes. The following is example of multiple model parallelization. Note it is helpful to change the `master_port` if you want to parallelize multiple models on the same main process.

```
# example of multiple model parallelization

parallelize(model_1, num_gpus=4, fp16=True, master_port=29500)
parallelize(model_2, num_gpus=4, fp16=True, master_port=29501)
```

Q. Why are some models not supported?

There are several factors. Models are partly supported or not supported if they ...

1. have dynamically changed layers.

We only can parallelize static layers because the parallelization process should be completed before the forward pass. But some models' layers (e.g., `BigBird's Self-Attention`) can change dynamically during the forward pass and ends up to unparallelization. For example, `BigBirdPegasus` contains `BigBird's Self-Attention` in its encoder layers, so they can't be parallelized.

2. have convolutional layers.

The convolution operation is not compatible with the tensor slicing method. For example, the attention layers of `ConvBERT` and all the layers of `SqueezeBERT` consist of convolutions, so they can not be parallelized. It is worth mentioning that although OpenAI's `GPT1` and `GPT2` also use convolutional operations, they can be parallelized because they actually perform matrix multiplication-based operations rather than actual convolutional operations. (Check the implementations of the `transformers.modeling_utils.Conv1D` layer)

3. have n-gram attention layers.

We conducted several parallelization experiments with `ProphetNet` that adopts the N-gram attention mechanism. Unfortunately, we found the results after the parallelization are not the same as the original representations for some reason.

4. adopt `EncoderDecoderModel`.

The `EncoderDecoderModel` framework conflicts with our `AutoPolicy` mechanism. Therefore, when using the `EncoderDecoderModel` framework, you have to write your own custom policy objects.

5. can not be serialized.

When transfering a model to other processes, the model's weights must be serialized. Thus, the models that are not serializable such as `RAG` are do not support parallelization.

Q. Can I use it on Docker?

Yes, but please note the followings.

1. Issue about multiple model parallelization in Docker containers

In Docker containsers, you can't parallelize multiple models on the same GPUs. For example, If you parallelize first model to GPUs `[0, 1, 2, 3]`, you can not parallelize second model to GPUs `[0, 1, 2, 3]` again.

This is a bug between Docker and `multiprocessing` package in the Python. If you already have semophores on your main process and you are using the `multiprocessing` module with `spawn` method, Python try to check leaked semaphores (please check here), and this leaked semaphore checker makes some problems in the Docker containers. (you can check more details here, And we are currently working to solve this issue)

2. Issue about shared memory size

In addition, you need to increase the shared memory size if you want to use distributed backed such as `nccl` in Docker containers (The default is set to 2mb, but it is too small)

Future works

Future works

Language Conference

I would really appreciate if we can work together to bring the parallel implementation design on your side to deepspeed and we can merge it with the high-performance kernels for the different models.

Thanks,
Reza

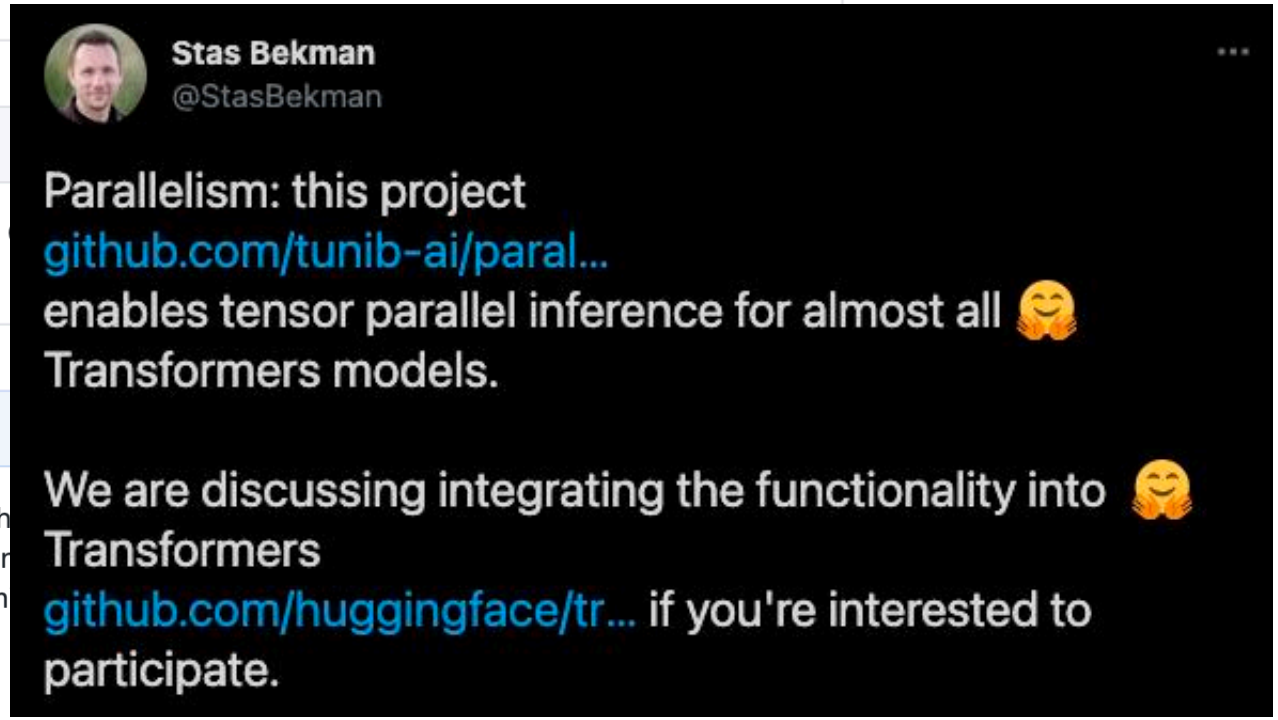
❤️ 1

RezaYazdaniAminabadi commented on 21 Jul

By the way, you will realize that with this PR the memory of the
before.

hyunwoongko commented on 22 Jul

Thanks for the positive reply. **@stas00** and I were discussing th
maybe we're thinking of this as part of the DeepSpeed and Trar
DeepSpeed team. For example, I and you implement and comm
Transformers.



Microsoft DeepSpeed 팀에서 먼저 협업을 제안했고, Huggingface Transformers 팀과 Integration 작업을 시작했어요!

Step 1. Collaborate DeepSpeed and TUNiB to move Paralleformers Tensor MP

The method of replacing the existing layer uses the scalable method of parallelformers. This does not change the entire transformer layer, but a method to replace a few linear layers with a sliced linear layer or a sliced all-reduce linear layer. Since DeepSpeed's Tensor MP replaced the entire Transformer layer, it could not reflect the specific mechanism of each model.

Firstly, I will implement this method and PR to DeepSpeed. (And this is what the DeepSpeed team wants me to do. refer to here) Ultimately, it's a good idea to archive parallelformers after most of the mechanisms of parallelformers are moved in DeepSpeed. It's a pity that our toolkit will be archived, but I think user accessibility is much more important because I want more people to easily use the large model. Parallelformers are less accessible compared to HF Transformers and MS DeepSpeed.

Step 2. Collaborate DeepSpeed and TUNiB about fused CUDA kernel

However, it is quite challenging to combine it with the CUDA kernel in the training process. In my opinion, it would not be difficult to implement forward pass, but the problem is backward. There is currently no backward pass implementation in the Tensor MP kernel in DeepSpeed. Because currently, Tensor MP is provided as inferences, DeepSpeed team didn't need to implement backward pass. Unfortunately, since I do not understand the CUDA code at a high level, it will be difficult for me to write the CUDA backward code myself.

Therefore, collaboration with DeepSpeed should be made in this part. It would be nice if we could collaborate with DeepSpeed and discuss about backward implementation of the DeepSpeed Tensor MP kernel. If this is impossible, it may be difficult to use the CUDA kernel during the training process.

Step 3. Collaborate Huggingface and TUNiB about transformers

In this step, we will add the newly implemented Tensor MP kernel by DeepSpeed and TUNiB into the HuggingFace. I think it will be similar to the Policy I implemented in parallelformers.

There are two methods to add to HuggingFace side.

1. Like `modeling_MODEL.py` and `tokenization_MODEL.py` in each model directory, we can create `parallel_MODEL.py` about the parallelization policy.

2. Alternatively, it is also worth considering about utilizing config.json. However, this can also be a fairly large work because every config.json files uploaded to hub needs to be changed.

Step 4. Collaborate Huggingface and TUNiB about DP, DDP, PP

Once Tensor MP is done, we will be able to proceed with combining it with DP and DDP. At the same time, It would be good to consider about implementing PP using `nn.ModuleList`. In my opinion, the existing PP based on `nn.Sequntial` is not

Future works

Language Conference

With Microsoft

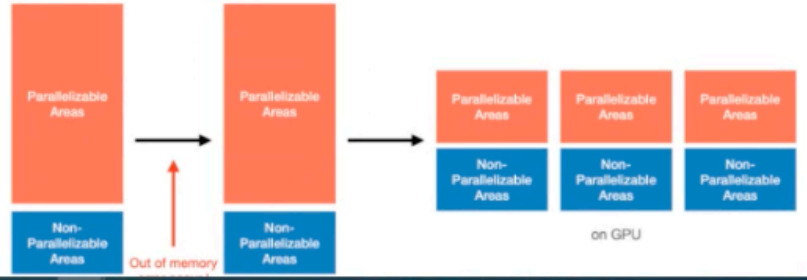
To solve this problem, we reversed the traditional processing method. We changed the structure so that we could call the framework code from the user's code several times, simultaneously. We called this method the "Inversion of process control".

[Figure 8] In Parallelformers, only certain parts operate in parallel unlike in DeepSpeed, where all parts operate in parallel

By applying the 'Inversion of process control' method, it's possible to execute only parts of the user's code that need to run multiple times. This prevents repeated loading of the same model or opening the server multiple times to the same port. As a result, we were able to successfully deploy the parallelized models to the web server. Also, because parallelization starts with the user's code, we could break parallelization whenever we wanted.

Step3: Solving GPU memory problems - Lazy GPU Allocation

We could deploy various models to the web server through 'Inversion of process control'. However, DeepSpeed had another problem.



With Huggingface

hyunwoongko 12 days ago Author

I thought of following code.

```
if token_type_ids is None:
    if hasattr(self, "token_type_ids"):
        buffered_token_type_ids = self.token_type_ids[:, :seq_length]
        buffered_token_type_ids_expanded = buffered_token_type_ids.expand(input_shape[0], se
        token_type_ids = buffered_token_type_ids_expanded
    elif model_parallel:  # <-- new variable
        token_type_ids = torch.zeros(input_shape, dtype=torch.long, device=os.getenv("LOCAL_
    else:
        token_type_ids = torch.zeros(input_shape, dtype=torch.long, device=self.position_ids
```

hyunwoongko 12 days ago Author

If model parallel is not applied, the LOCAL_RANK variable does not exist.

Load more...

stas00 12 days ago • edited Contributor

but still I'd prefer to abstract `os.getenv("LOCAL_RANK"))` and include meaningful exceptions should it be invalid for whatever reason. i.e. `transfromers` needs to have a defined API to get the rank and not rely just on env var.

It'd also hide all that checking if it's not defined. Let's perhaps start with a helper util in `modeling_utils.py` to add the abstraction

hyunwoongko 12 days ago Author

You are right. It would be good to implement mpu like nvidia megatron and provide it as a utility function.

hyunwoongko 12 days ago Author

https://github.com/NVIDIA/Megatron-LM/blob/main/megatron/mpu/initialize.py#L255

stas00 12 days ago • edited Contributor

we will absolutely need to implement MPU anyway, so let's to do it right from the get-going!

You have a choice of Megatron or deepspeed MPU versions, I may have seen some other.

I think I may have even started to port one while trying to plug my PP PR into Deepspeed's 3D. yes, I did:
https://github.com/huggingface/transformers/pull/9765/files#diff-48e672da3865f77a2e1d38954e8e075c0f1e02c7306f163847b9b8ecc56ede24
I see I took it from megatron.

stas00 reviewed 12 days ago View changes

src/transformers/models/bart/modeling_bart.py

```
@@ -253,7 +253,7 @@ def forward(
            attn_output = attn_output.view(bsz, self.num_heads, tgt_len, self.head_dim)
            attn_output = attn_output.transpose(1, 2)
-           attn_output = attn_output.reshape(bsz, tgt_len, embed_dim)
+           attn_output = attn_output.reshape(bsz, tgt_len, self.embed_dim)
```

stas00 12 days ago • edited Contributor

if we do that, I propose we change the earlier code:

```
- bsz, tgt_len, embed_dim = hidden_states.size()
+ bsz, tgt_len, _ = hidden_states.size()
```

so that there will be no confusing `embed_dim` local variable hanging around.

that way we know we already use `self.embed_dim`

hyunwoongko 12 days ago Author

Great.

Reply...

Resolve conversation

stas00 reviewed 12 days ago View changes

src/transformers/models/prophetnet/modeling_prophetnet.py

```
@@ -762,7 +762,7 @@ def forward(
            attn_output = (
                attn_output.view(batch_size, self.num_attn_heads, tgt_len, self.head_dim)
                .transpose(1, 2)
```

Q & A