# tensor_encoding API

Jakub Konečný // konkey@google.com // May 2020

# Warm-up

# Motivating example

```
model_weight = ...  # Some weight(s) of a model needed by many workers.
```

# Motivating example

```
model_weight = ...  # Some weight(s) of a model needed by many workers.
# IDEA: Replace broadcasting model_weight with something smaller.
```

# Motivating example

```
model_weight = ...  # Some weight(s) of a model needed by many workers.
# IDEA: Replace broadcasting model_weight with something smaller.

encoder = ...  # An instance of te.core.SimpleEncoder.
```

```
from tensorflow_model_optimization import tensor_encoding as te
```

# Motivating example

```
model_weight = ...  # Some weight(s) of a model needed by many workers.
# IDEA: Replace broadcasting model_weight with something smaller.

encoder = ...  # An instance of te.core.SimpleEncoder.

# In a "central" context:
encoded_weight, _ = encoder.encode(model_weight)
```

# Motivating example

```
model_weight = ...  # Some weight(s) of a model needed by many workers.
# IDEA: Replace broadcasting model_weight with something smaller.

encoder = ...  # An instance of te.core.SimpleEncoder.

# In a “central” context:
encoded_weight, _ = encoder.encode(model_weight)

# The encoded_weight is a smaller Tensor or a collection of Tensors.
# Broadcast encoded_weight -- ideally needing less time.
```

# Motivating example

```
model_weight = ...  # Some weight(s) of a model needed by many workers.
# IDEA: Replace broadcasting model_weight with something smaller.

encoder = ...  # An instance of te.core.SimpleEncoder.

# In a “central” context:
encoded_weight, _ = encoder.encode(model_weight)

# The encoded_weight is a smaller Tensor or a collection of Tensors.
# Broadcast encoded_weight -- ideally needing less time.

# In a “worker” context:
decoded_model_weight = encoder.decode(encoded_weight)
```

# Motivating example

```
model_weight = ...  # Some weight(s) of a model needed by many workers.
# IDEA: Replace broadcasting model_weight with something smaller.

encoder = ...  # An instance of te.core.SimpleEncoder.

encoder
- Can hide any invertible, potentially lossy, Tensor transformations.
    - Easy for researcher to implement; only pure TF needed
```

# Motivating example

```
model_weight = ...  # Some weight(s) of a model needed by many workers.
# IDEA: Replace broadcasting model_weight with something smaller.

encoder = ...  # An instance of te.core.SimpleEncoder.

encoder
- Can hide any invertible, potentially lossy, Tensor transformations.
    - Easy for researcher to implement; only pure TF needed
- Provides a simple interface for platforms to integrate with
    - Distributed training
    - Federated learning
```

# Motivating example

```
model_weight = ...  # Some weight(s) of a model needed by many workers.
# IDEA: Replace broadcasting model_weight with something smaller.

encoder = ...  # An instance of te.core.SimpleEncoder.
```

**encoder**

- Can hide any invertible, potentially lossy, Tensor transformations.
    - Easy for researcher to implement; only pure TF needed
- Provides a simple interface for platforms to integrate with
    - Distributed training
    - Federated learning

Similar pattern available for an encoded reduce/gather (te.core.GatherEncoder)

# tensor_encoding API

# tensor_encoding API

General TF tool for

**invertible**, potentially **lossy**, transformations

**encode / decode methods**

**decode(encode(x)) != x**

# Design objectives

**Organize the space of lossy compression transformations in TensorFlow**

→ Target multiple use cases

**Smoothen the “research → production” pipeline**

→ Different APIs for researcher and platform

→ Novel research can impact multiple platforms

# Design objectives

Organize the space of lossy compression transformations in TensorFlow

Use cases:

- **Gradient compression in distributed training**

Recent Survey [Apr 2020]

Compressed Communication for Distributed Deep Learning: Survey and Quantitative Evaluation

# Design objectives

Organize the space of lossy compression transformations in TensorFlow

Use cases:

- **Gradient compression in distributed training**

Only one of potentially many use-cases...

# Design objectives

Organize the space of lossy compression transformations in TensorFlow

Use cases:

- **Gradient compression in distributed training**
- **Model broadcast in distributed training**

# Design objectives

Organize the space of lossy compression transformations in TensorFlow

Use cases:

- **Gradient compression in distributed training**
- **Model broadcast in distributed training**
- Update compression for federated learning
- Model state compression for federated learning
- Trained model compression for on-disk storage
- Differential privacy + compression (e.g. cpSGD)
- Secure Aggregation protocol (CCS paper, compression)
- ...

Distributed training

Federated learning

Mobile deployment

# Design objectives

Smoothen the “research → production” pipeline

**Goal:** A platform integrates with **`tensor_encoding`** once
→ Anything a researcher contributes is immediately available to **every** platform

# Design objectives

Smoothen the “research → production” pipeline

**Goal:** A platform integrates with **`tensor_encoding`** once
→ Anything a researcher contributes is immediately available to **every** platform

```
Effectively design two API surfaces
- For researchers
- For platforms

… and become the glue in between.
```

# Implementation layers

- Research surface API
    - **`te.core.EncodingStageInterface`**
        - Relatively simple invertible transformations
        - Map single Tensor to a collection of Tensors and back
    - Needed by someone who wants to implement a novel compression algorithm

# Implementation layers

- Research surface API
  - `te.core.EncodingStageInterface`
    - Relatively simple invertible transformations
    - Map single Tensor to a collection of Tensors and back
  - Needed by someone who wants to implement a novel compression algorithm
- Glue layer
  - Composition (tree structure) of `te.core.EncodingStageInterface` instances
  - Hides as much complexity as possible
  - Needed only by `tensor_encoding` API authors / maintainers

# Implementation layers

- Research surface API
  - **`te.core.EncodingStageInterface`**
    - Relatively simple invertible transformations
    - Map single Tensor to a collection of Tensors and back
  - Needed by someone who wants to implement a novel compression algorithm
- Glue layer
  - Composition (tree structure) of **`te.core.EncodingStageInterface`** instances
  - Hides as much complexity as possible
  - Needed only by **`tensor_encoding`** API authors / maintainers
- Platform surface API
  - Specializes to a usage pattern (broadcast / reduce / secure aggregation / …)
  - Needed by deployment platforms

# Implementation layers

- Research surface API
  - **te.core.EncodingStageInterface**
    - Relatively simple invertible transfor
    - Map single Tensor to a collection of
  - Needed by someone who wants to imple
- Glue layer
  - Composition (tree structure) of **te.core**
  - Hides as much complexity as possible
  - Needed only by **tensor_encoding** API
- Platform surface API
  - Specializes to a usage pattern (broadcas
  - Needed by deployment platforms

All functional
**"tensors in - tensors out"**
transforms

No **tf.Variable** creation
No Python's user-execution
side-effects

# Platform surface API

Specialized to top-level use cases

- Broadcast → **`te.core.SimpleEncoder`**
- Reduce / Gather → **`te.core.GatherEncoder`**

# Explanation flow

1. [researcher] **`te.core.EncodingStageInterface`**
2. [researcher] **`te.core.AdaptiveEncodingStageInterface`**
3. [platform] **`te.core.SimpleEncoder`**
4. [platform] **`te.core.GatherEncoder`**

# Research surface API

**`te.core.EncodingStageInterface`** for researchers to implement

Should be relatively simple, meant to compose with other implementations

# Research surface API

**`te.core.EncodingStageInterface`** for researchers to implement

Should be relatively simple, meant to compose with other implementations

Examples:

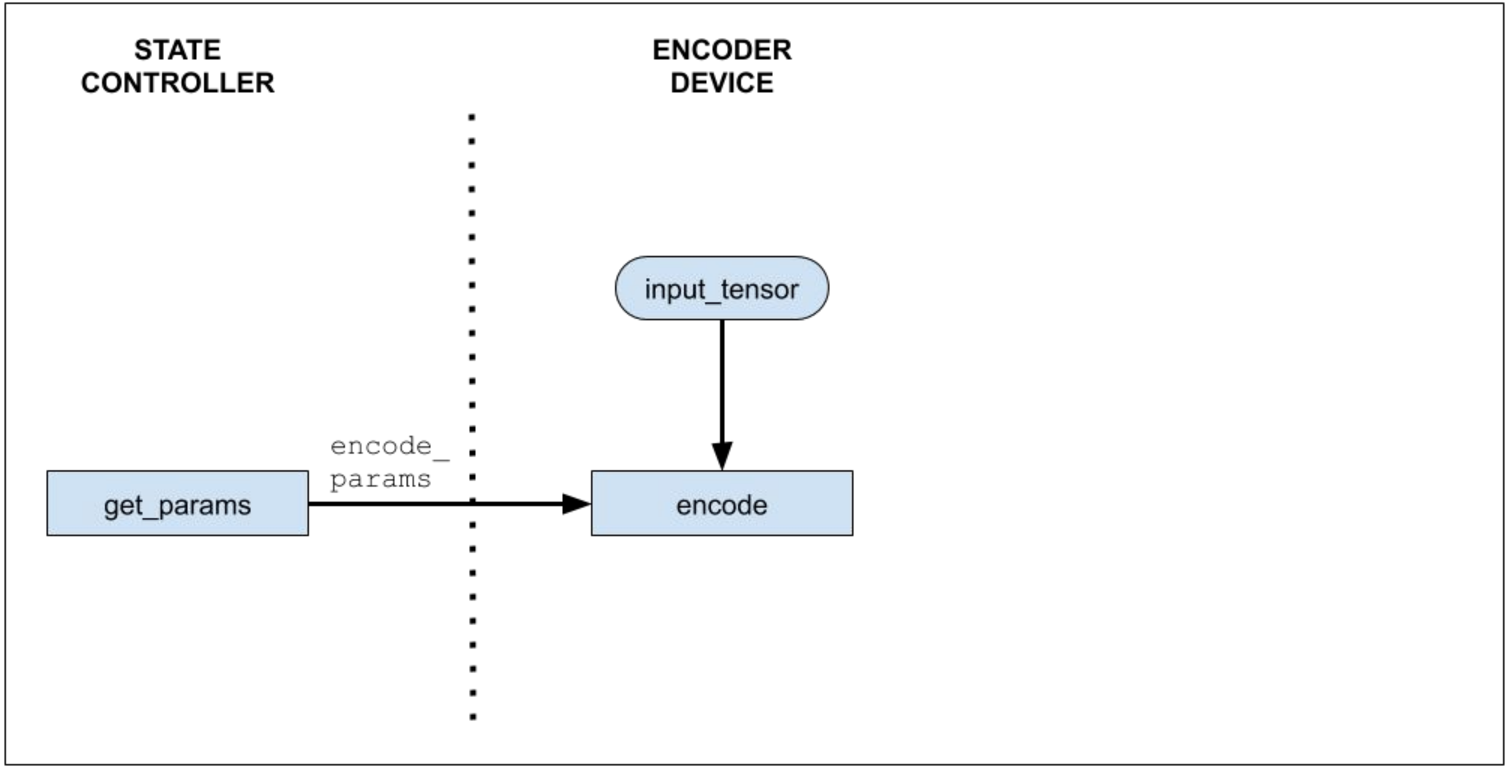
- Quantization (uniform, adaptive, … )
- Hadamard transform (random, deterministic)
- Integer bitpacking
- Subsampling (random, top-k, … )
- Sparse representation
- …

# EncodingStageInterface

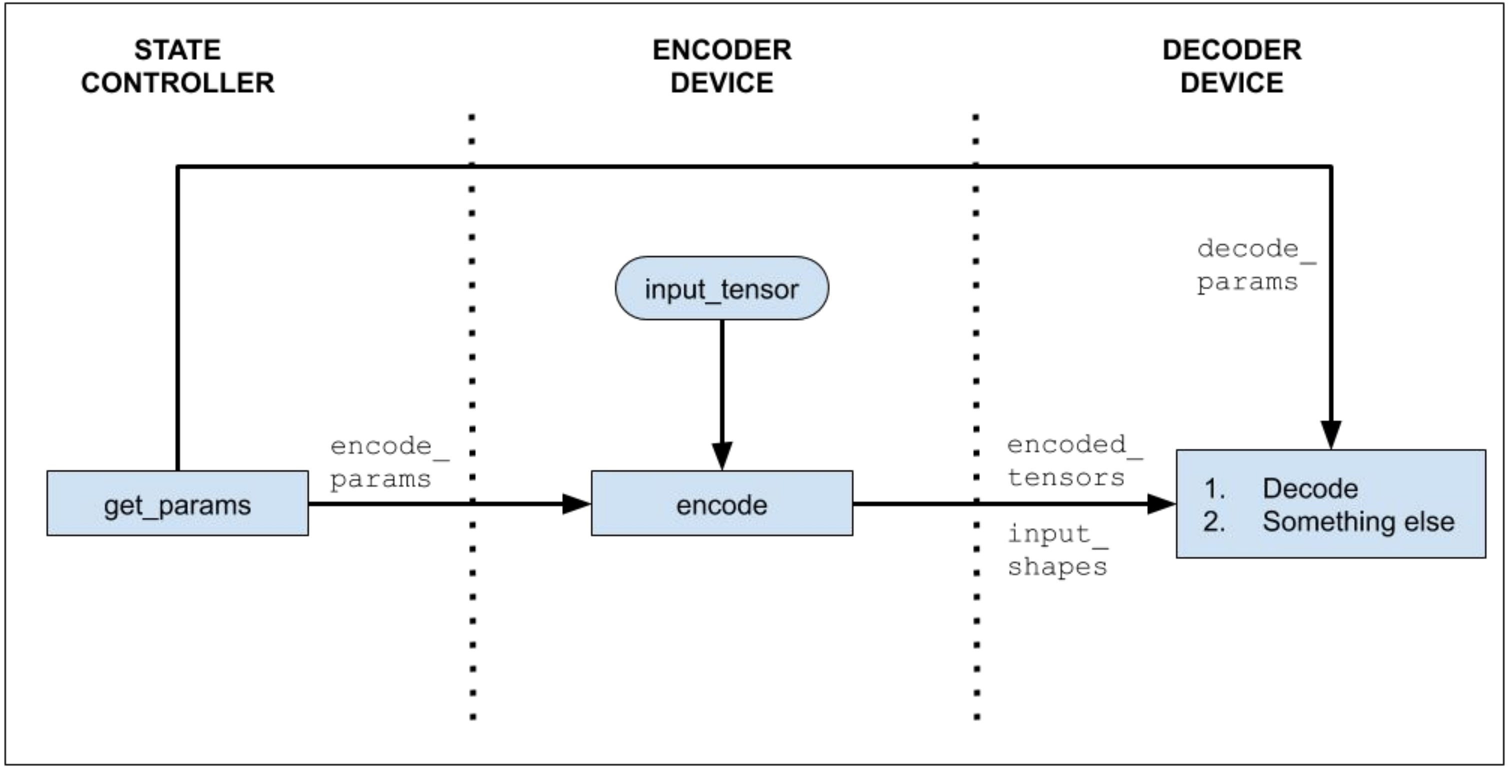
- get_params():
  → encode_params, decode_params
- encode(x, encode_params):
  → encoded_tensors
- decode(encoded_tensors,
  decode_params,
  num_summands=None,
  shape=None):
  → decoded_x

# **EncodingStageInterface** (1/3)

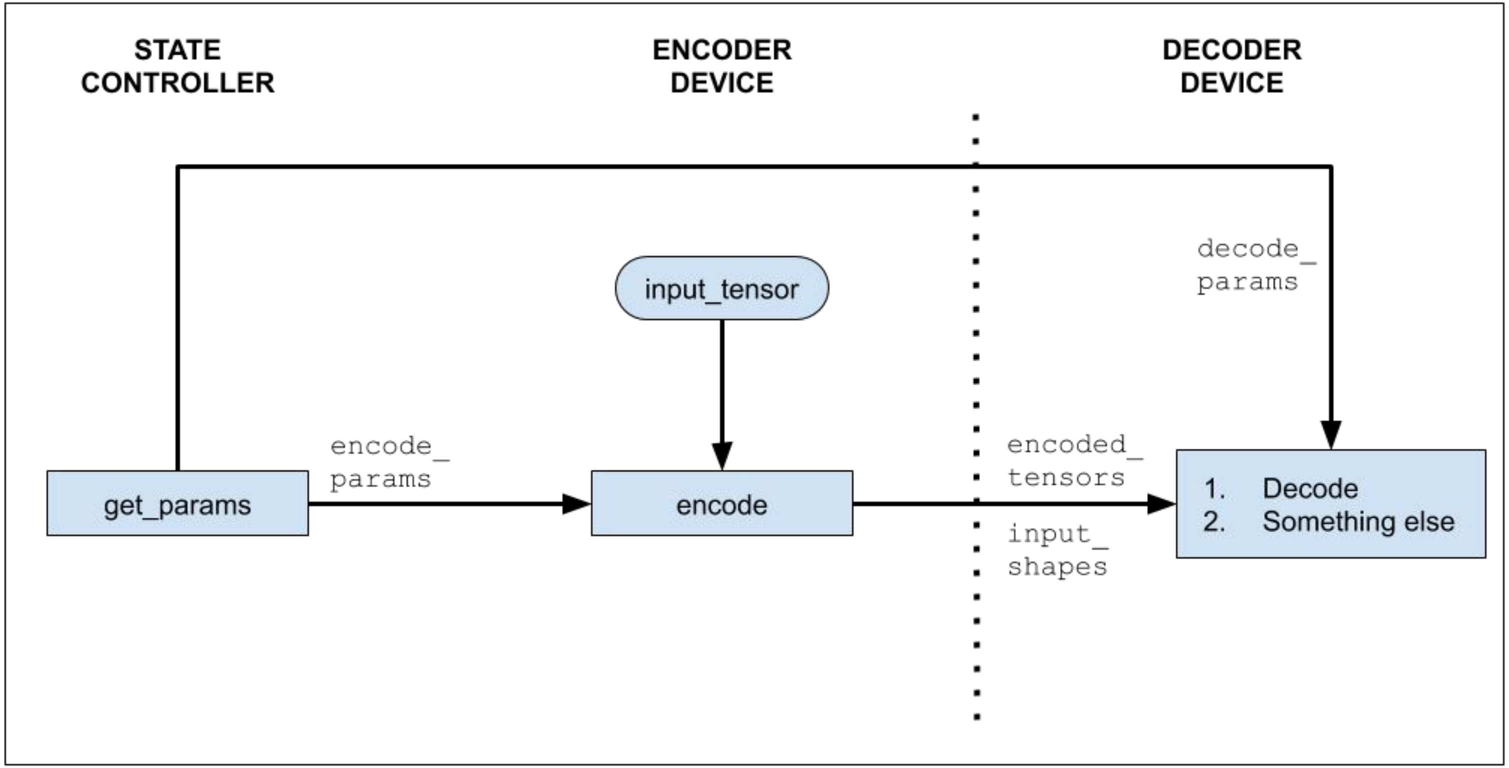
# **EncodingStageInterface** (2/3)



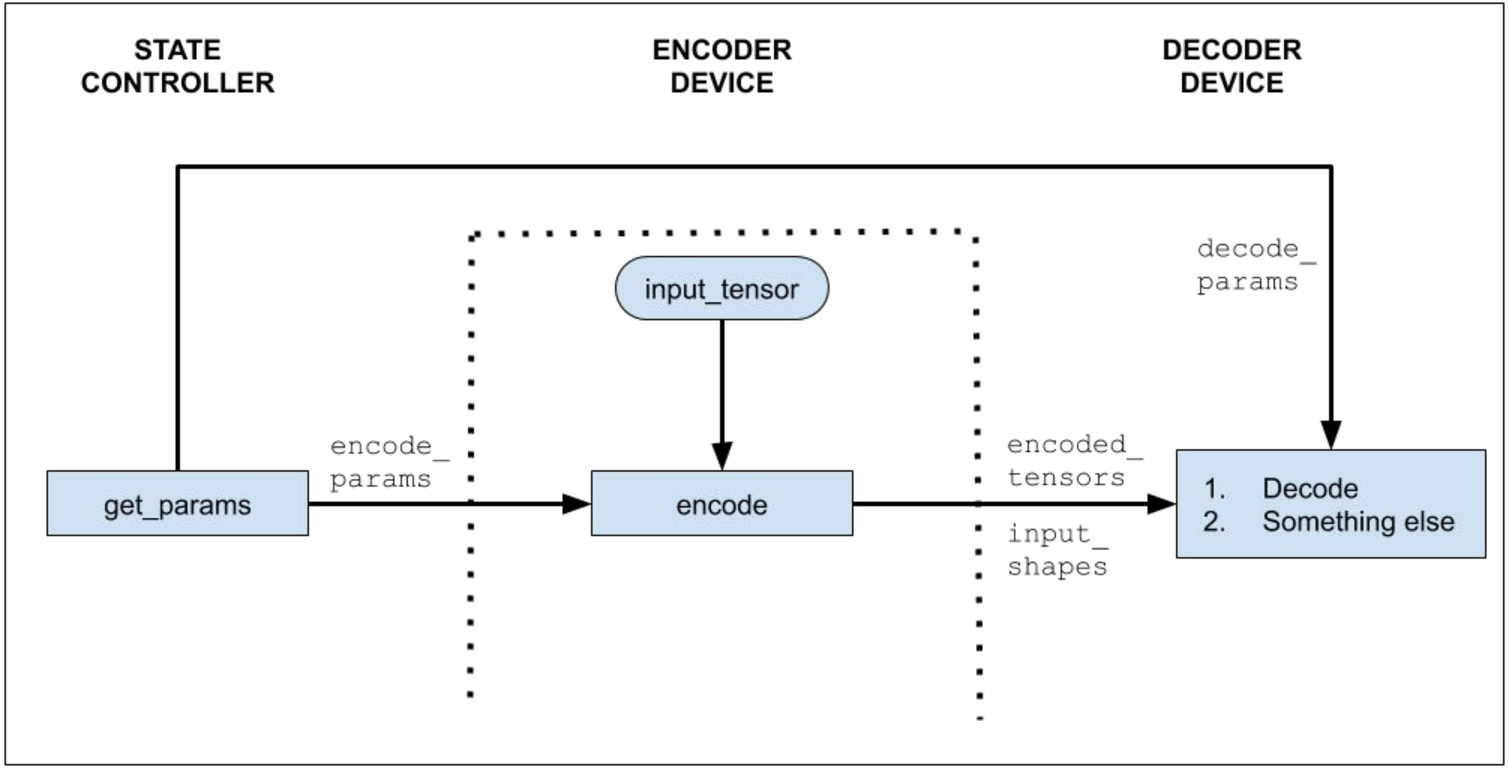
# EncodingStageInterface (3/3)



# EncodingStageInterface (for broadcast)



# **EncodingStageInterface** (for reduce)



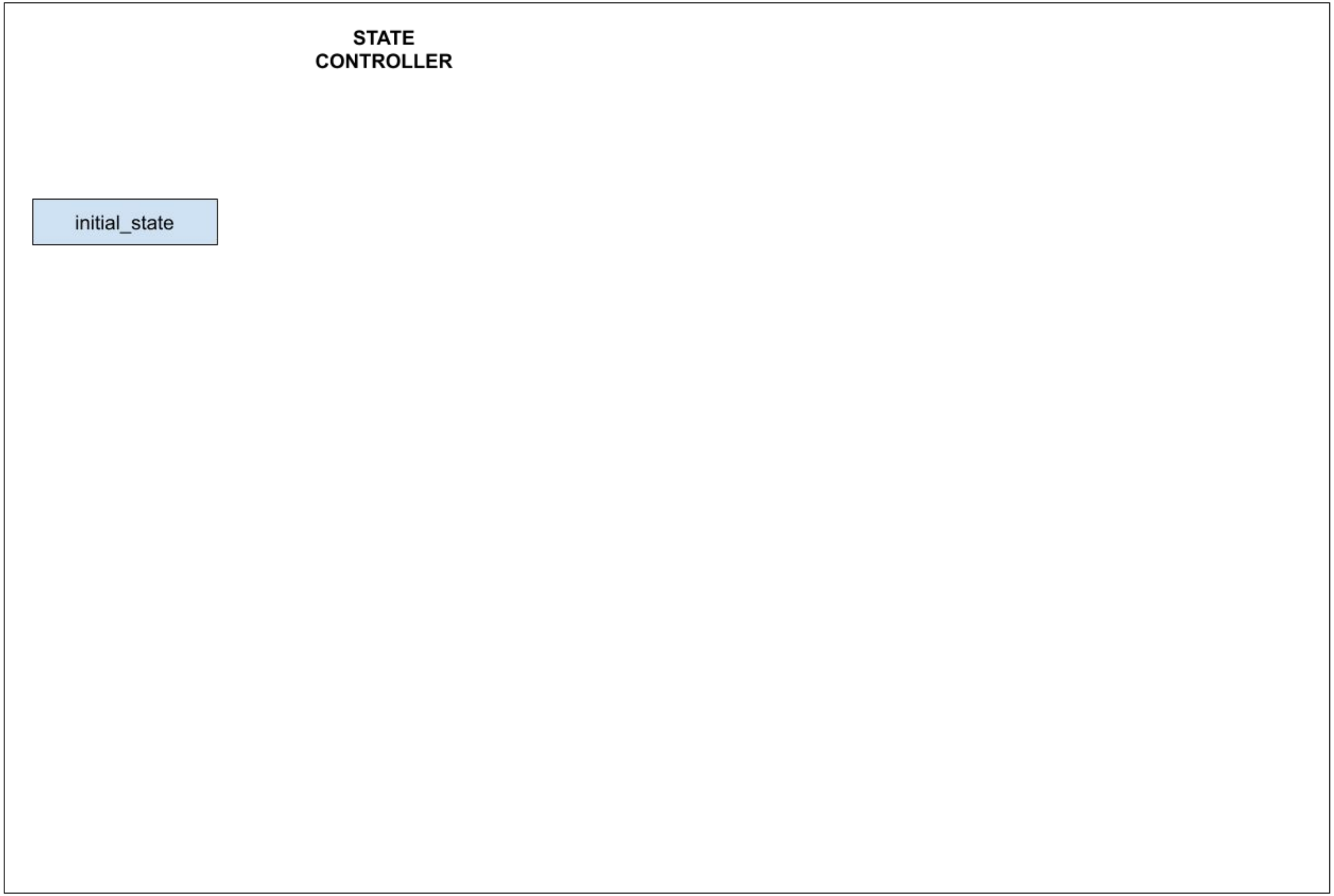
# `AdaptiveEncodingStageInterface`

Example uses:

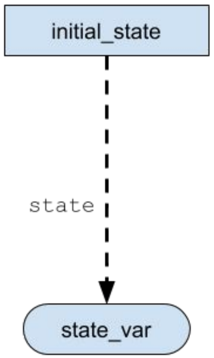
- Adaptive gradient clipping for differential privacy (paper)
- Adaptive quantization range / bitwidth
  - Needs to be known in advance -- for Secure Aggregation for instance (paper)
- Progressively less aggressive compression as model convergences
- …

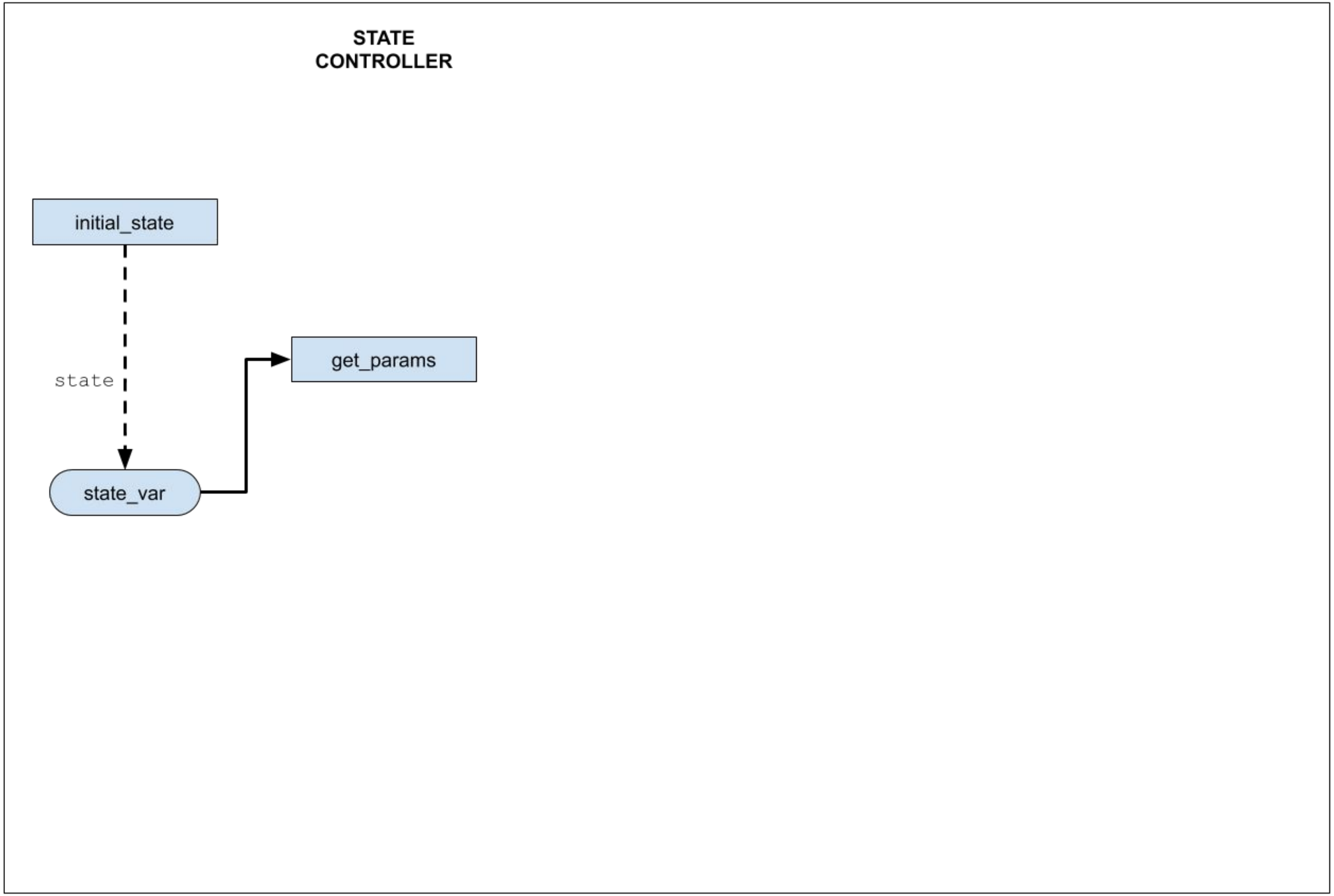
# AdaptiveEncodingStageInterface

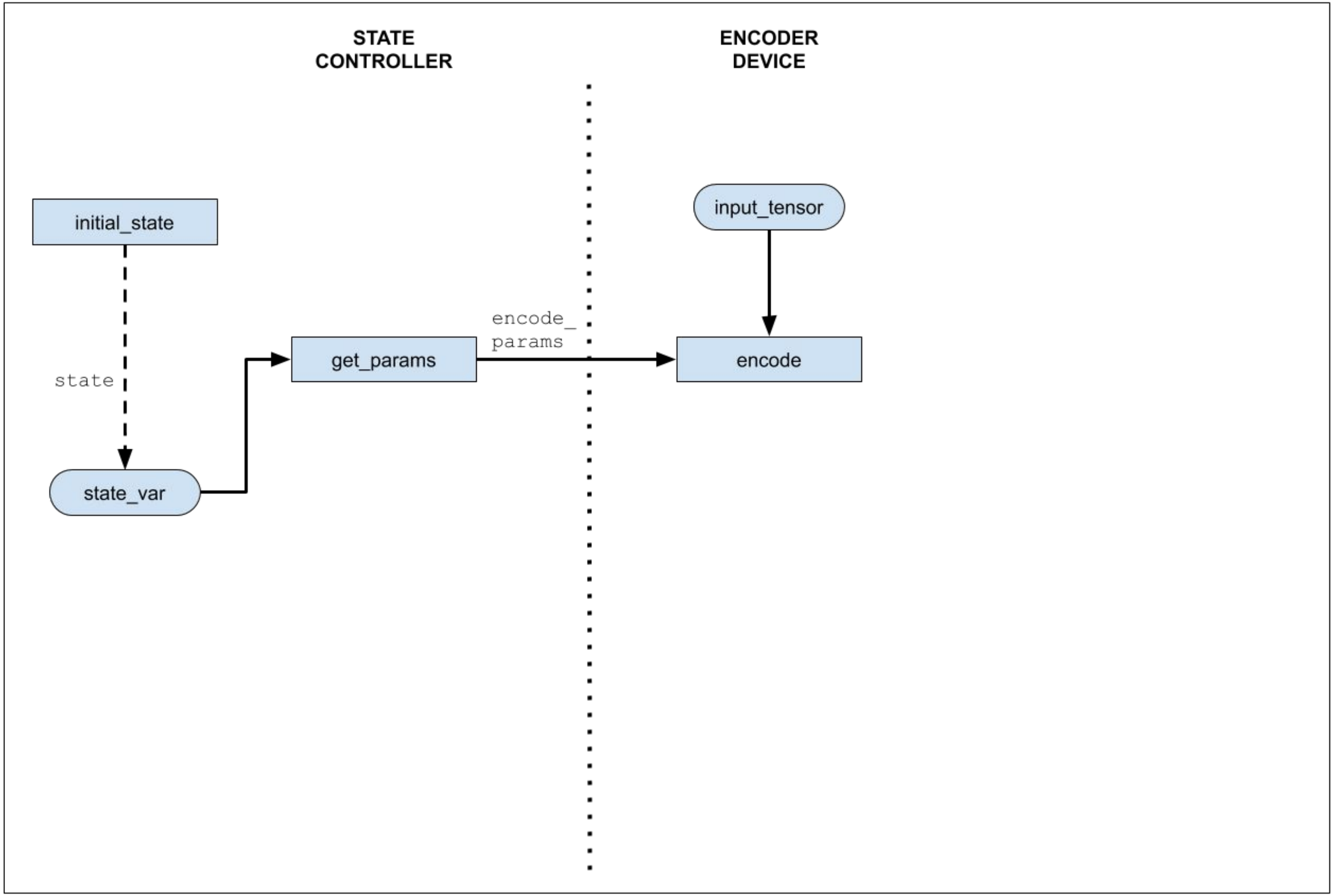
- `initial_state():`
  → `state`
- `get_params(state):`
  → `encode_params, decode_params`
- `encode(x, encode_params):`
  → `encoded_tensors, state_update_tensors`
- `decode(encoded_tensors, decode_params, num_summands=None, shape=None):`
  → `decoded_x`
- `update_state(state, state_update_tensors):`
  → `state`

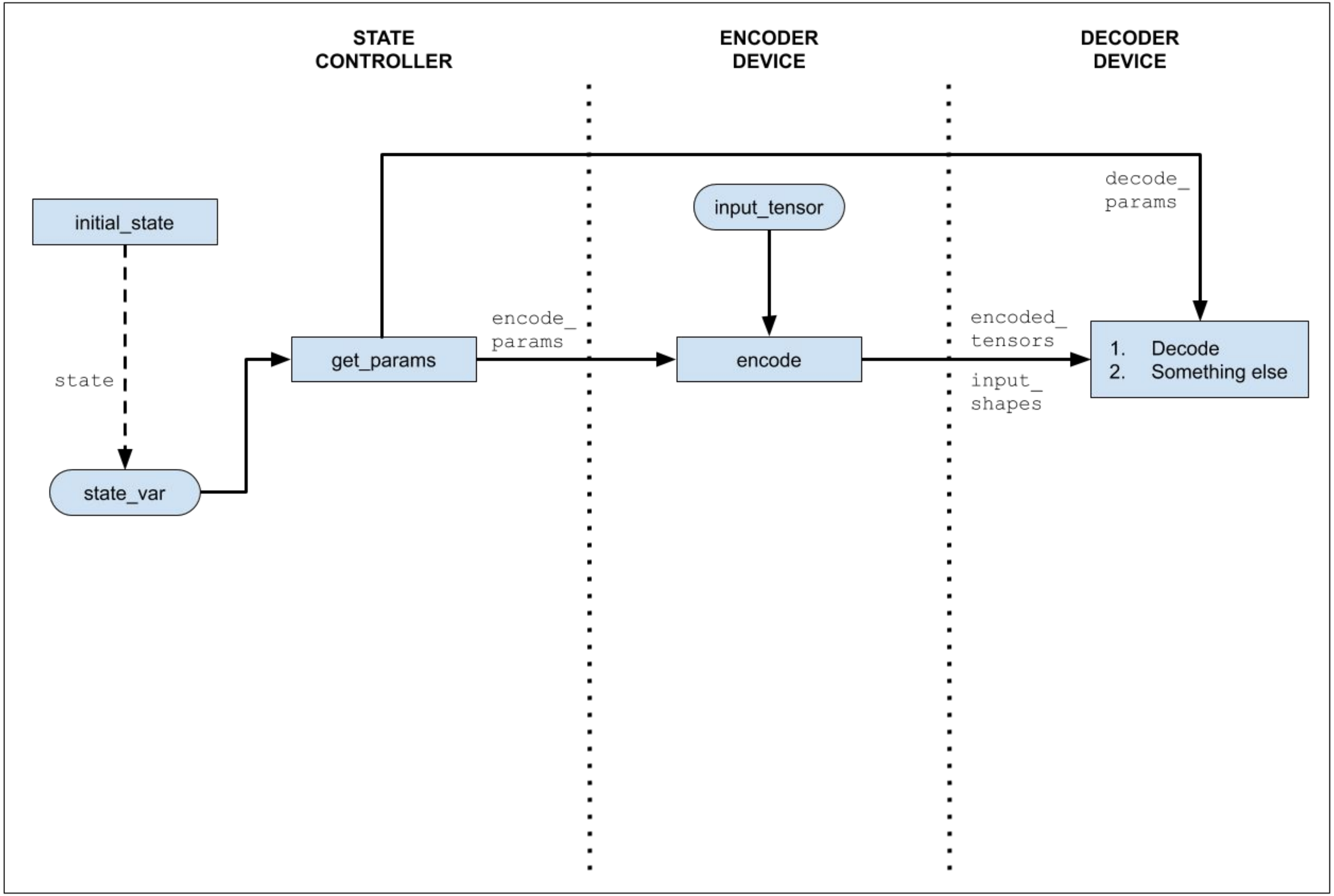

STATE
CONTROLLER
initial_state

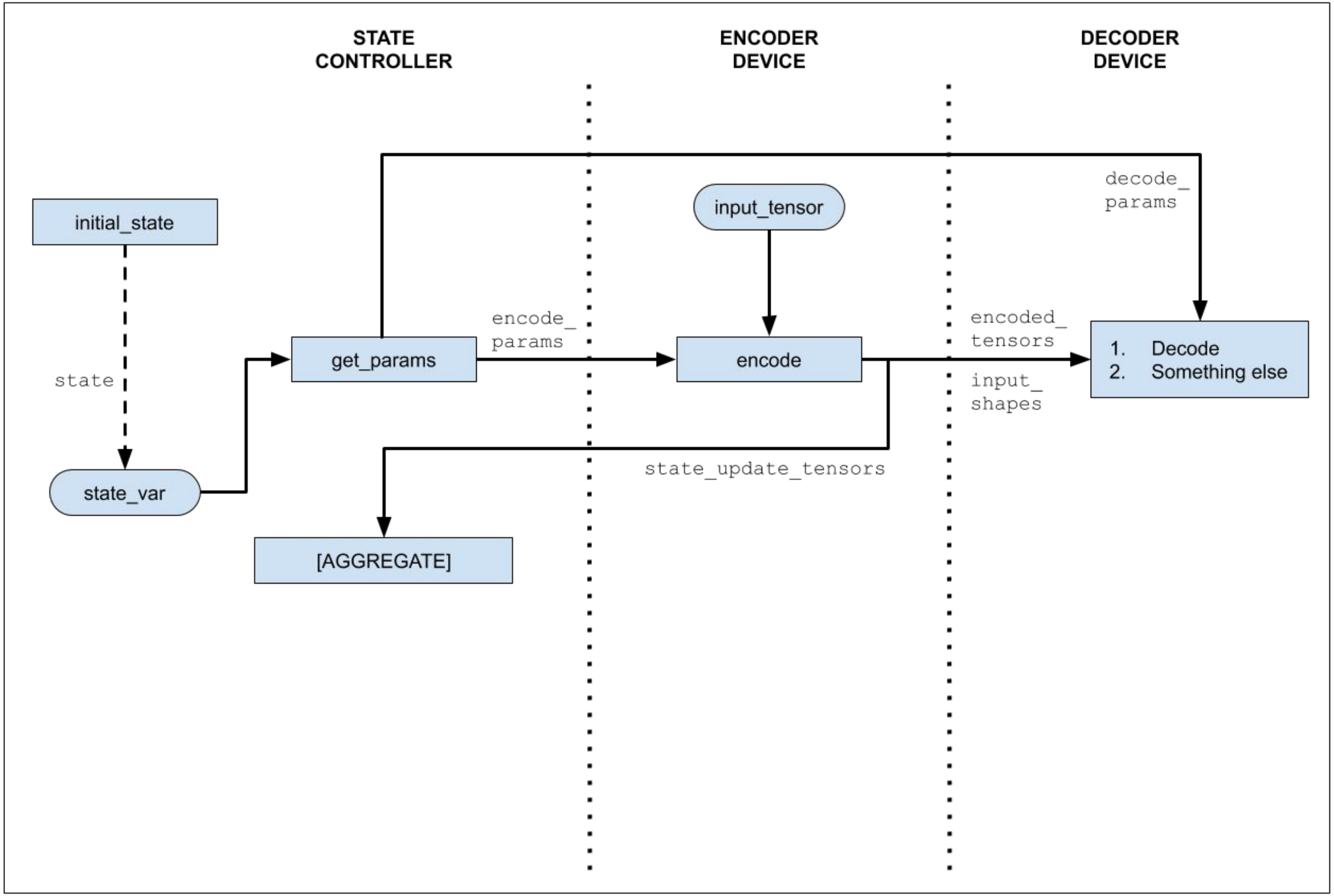
**STATE CONTROLLER**

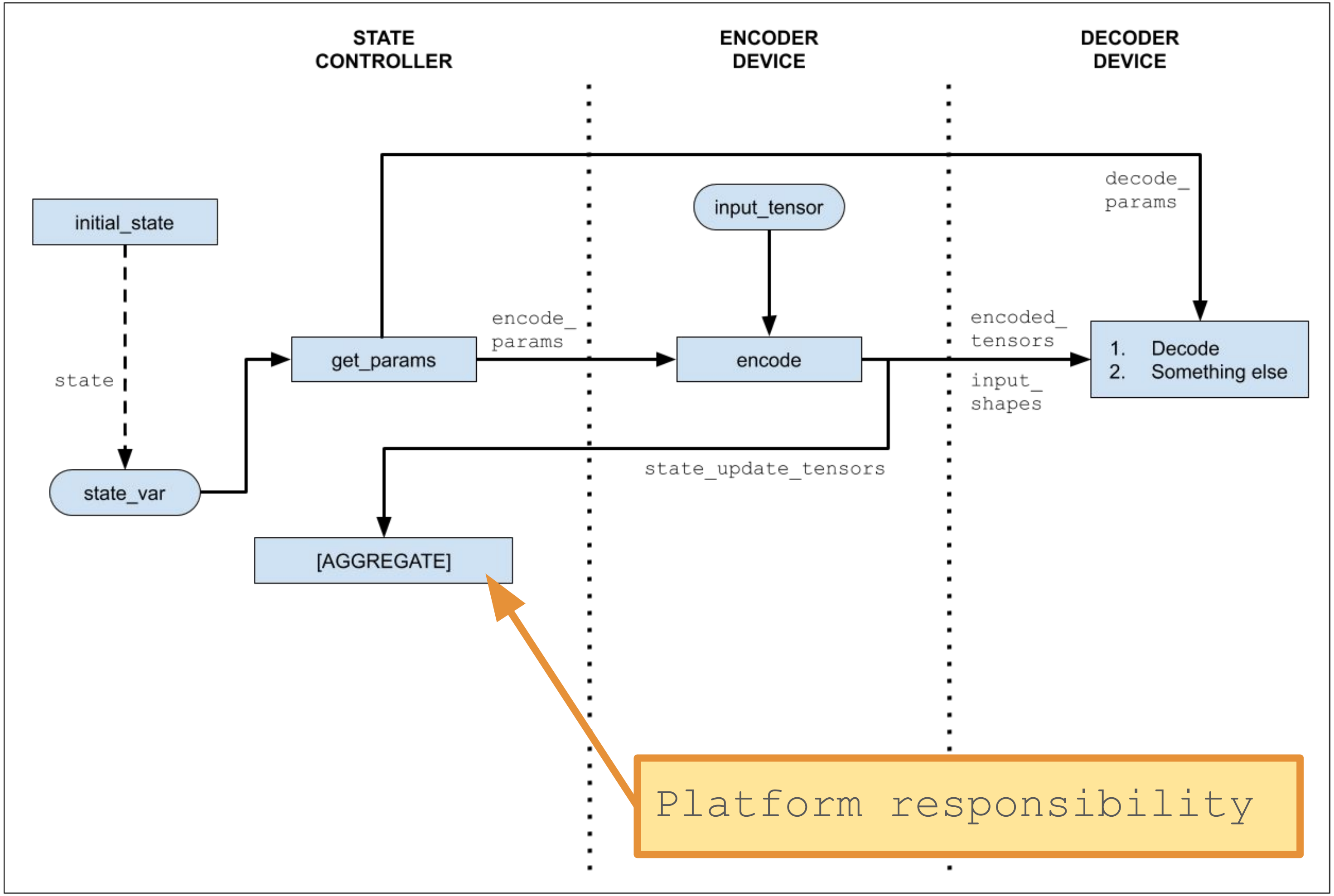


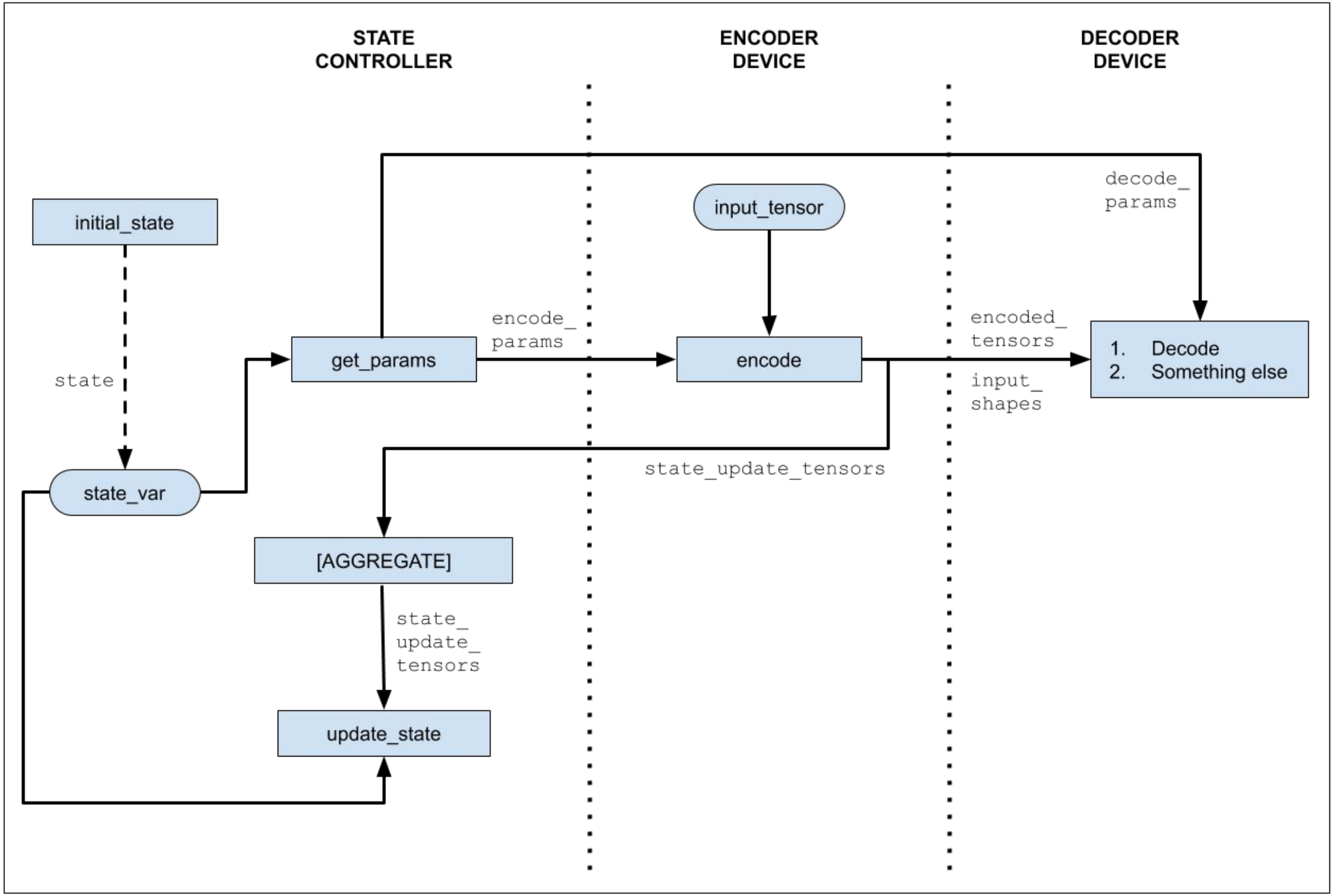

STATE
CONTROLLER
initial_state
state
state_var
get_params


STATE
CONTROLLER
ENCODER
DEVICE
initial_state
state
state_var
get_params
encode_
params
input_tensor
encode

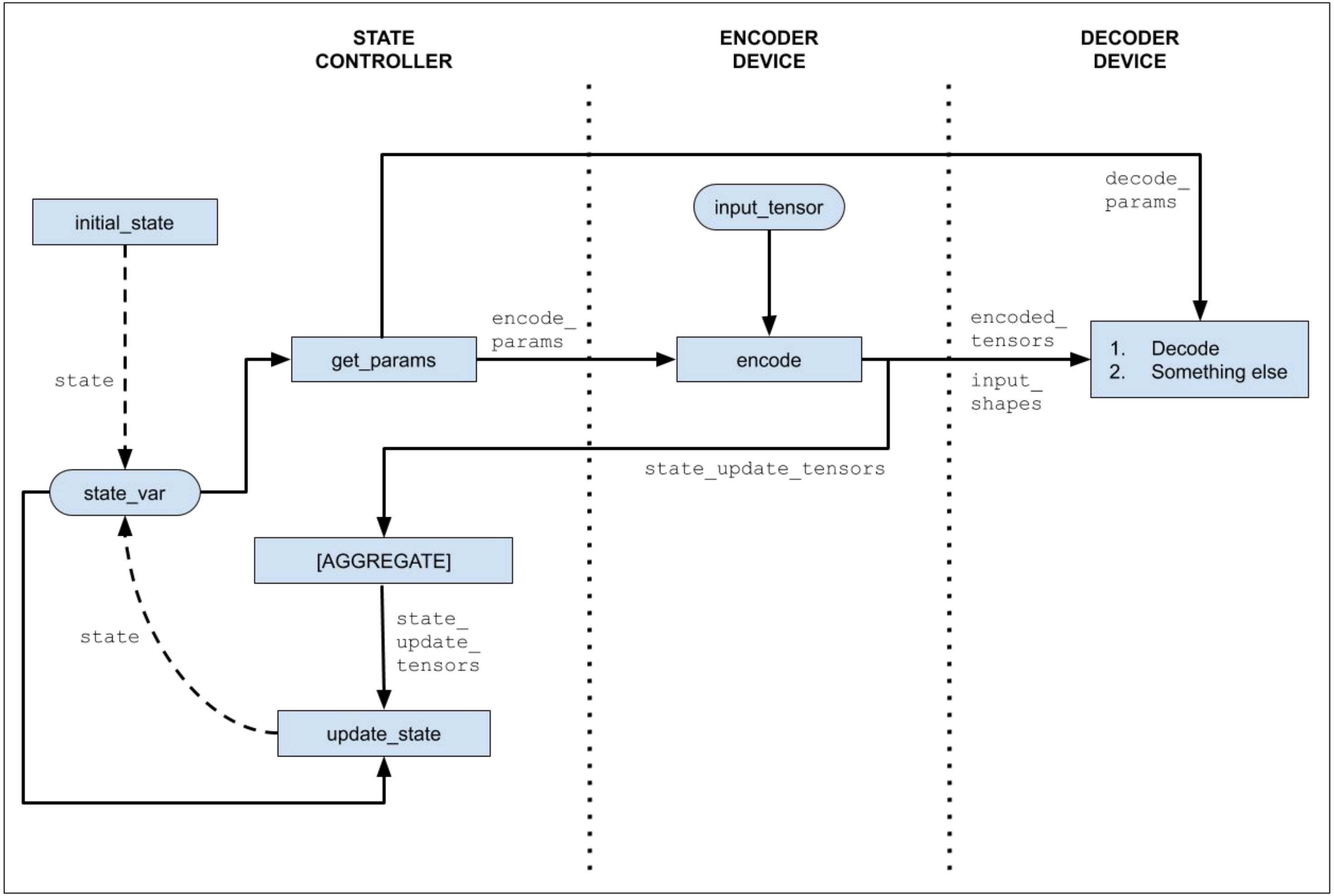

STATE CONTROLLER
ENCODER DEVICE
DECODER DEVICE
initial_state
state
state_var
get_params
encode_ params
decode_ params
input_tensor
encode
encoded_ tensors
input_ shapes
1. Decode
2. Something else


STATE CONTROLLER
ENCODER DEVICE
DECODER DEVICE
initial_state
state
state_var
get_params
encode_ params
decode_ params
input_tensor
encode
encoded_ tensors
input_ shapes
1. Decode
2. Something else
state_update_tensors
[AGGREGATE]


STATE CONTROLLER
ENCODER DEVICE
DECODER DEVICE
initial_state
state
state_var
get_params
encode_
params
input_tensor
encode
encoded_
tensors
input_
shapes
decode_
params
1. Decode
2. Something else
state_update_tensors
[AGGREGATE]
Platform responsibility



STATE CONTROLLER
ENCODER DEVICE
DECODER DEVICE
initial_state
state
state_var
get_params
encode_ params
decode_ params
input_tensor
encode
encoded_ tensors
input_ shapes
1. Decode
2. Something else
state_update_tensors
[AGGREGATE]
state_ update_ tensors
update_state


STATE CONTROLLER
ENCODER DEVICE
DECODER DEVICE
initial_state
state
state_var
get_params
encode_ params
input_tensor
encode
decode_ params
encoded_ tensors
input_ shapes
1. Decode
2. Something else
state_update_tensors
[AGGREGATE]
state_ update_ tensors
state
update_state

# Platform surface API

Specialize **`(Adaptive)EncodingStageInterface`** to top-level use cases

- Broadcast → **`te.core.SimpleEncoder`**
- Gather → **`te.core.GatherEncoder`**

# SimpleEncoder

- `initial_state()`
  → `state`
- `encode(x, state)`
  → `encoded_x, state`
- `decode(encoded_x)`
  → `decoded_x`

# **GatherEncoder**

Specializes to encode an input compatible with `tf.TensorSpec`.

Primarily for **`SUM`** or **`MEAN`** reduction

→ For instance maps to [`tf.distribute.ReduceOp`]

**Efficiency:** Decomposes decoding part based on its commutativity with sum

→ **`decode_before_sum`**

→ **`decode_after_sum`**

# GatherEncoder

- `initial_state():`
  → `state`
- `get_params(state):`
  → `encode_params, decode_before_sum_params, decode_after_sum_params`
- `encode(x, encode_params):`
  → `encoded_x, state_update_tensors`
- `decode_before_sum(encoded_x, decode_before_sum_params):`
  → `part_decoded_x`
- `decode_after_sum(part_decoded_x, decode_after_sum_params, num_summands):`
  → `decoded_x`
- `update_state(state, state_update_tensors):`
  → `state`

# tensor_encoding API

General TF tool for

**invertible**, potentially **lossy**, transformations

**encode / decode methods**

**decode(encode(x)) != x**