FHE & Cheddar: A Tutorial on Fully Homomorphic Encryption

Live coding: Implementing FHE applications with Cheddar

ISCA 2026 Tutorial · Sunday, June 28, 2026

SCALE LAB

Seoul National University

Setup: Launch Docker and Jupyter Server

Requirements

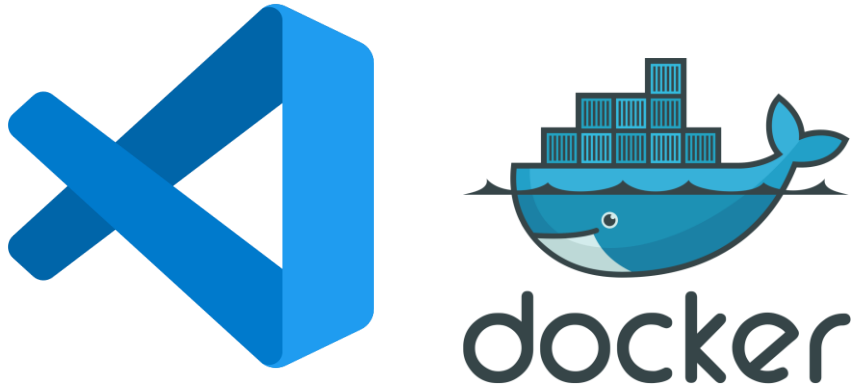
- NVIDIA server or consumer GPU (Pascal or later) with at least 16 GB of GPU memory[1)]
  - Check GPU driver version (>=520.61.05) with `nvidia-smi --query-gpu=driver_version`

```
(cheddar) wonseok@gpu14:~/isca-tutorial/cheddar-fhe$ nvidia-smi --query-gpu=driver_version
driver_version
590.48.01
```

- Docker and nvidia-container-toolkit (install Docker, install toolkit)
  - Check with `nvidia-container-toolkit --version`

```
(cheddar) wonseok@gpu14:~/isca-tutorial/cheddar-fhe$ nvidia-container-toolkit --version
NVIDIA Container Runtime Hook version 1.19.1
commit: 09ceee5dde66ba9ce25c7cc69b1ebd5e6e3266fa
```

- **VS Code (recommended)** or another editor



[1)] We prepared one additional DGX spark machine. If you don't have any access to one, reach out to us.

Pull Docker Image and Run

- Pull the prebuilt Docker image and launch the Jupyter server

terminal

```
docker pull ghcr.io/choiwons/cheddar-tutorial:isca2026
docker run --rm -it --gpus=all -e JUPYTER_PORT=9999 \
-p 127.0.0.1:9999:9999 ghcr.io/choiwons/cheddar-tutorial:isca2026
```

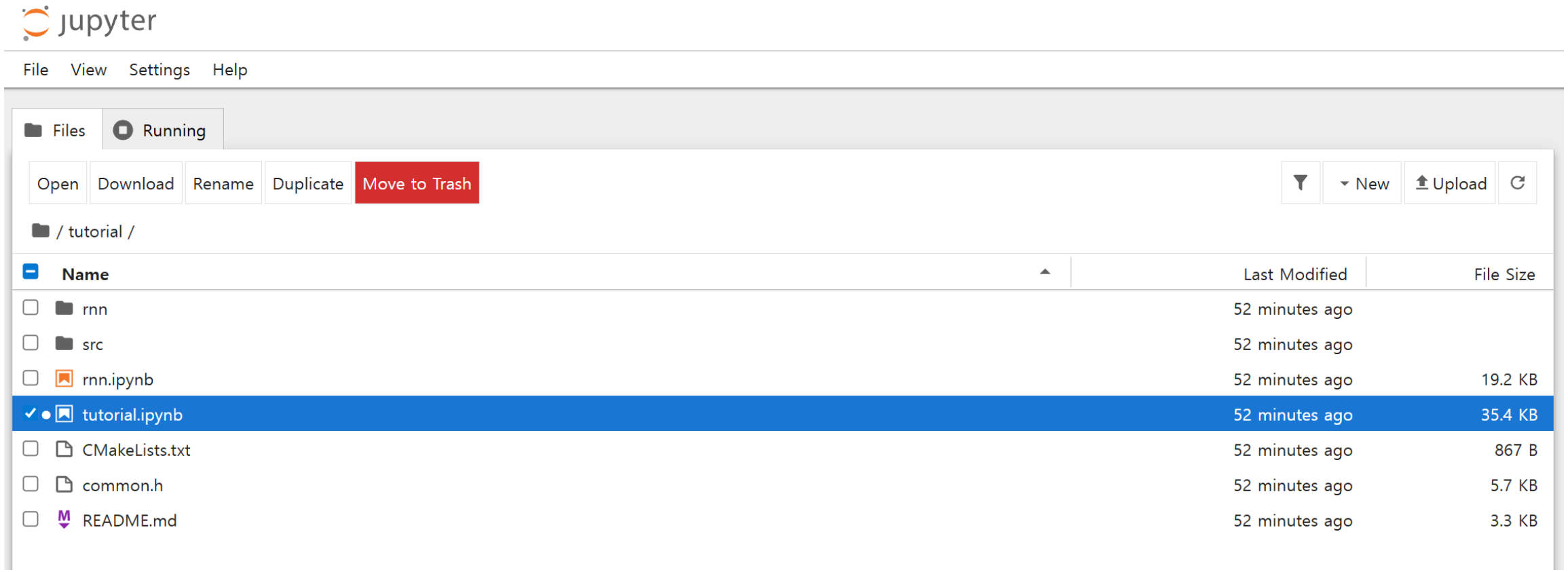
```
[I 2026-06-23 09:07:15.667 ServerApp] jupyter_lsp | extension was successfully linked.
[I 2026-06-23 09:07:15.668 ServerApp] jupyter_server_terminals | extension was successfully linked.
[I 2026-06-23 09:07:15.670 ServerApp] jupyterlab | extension was successfully linked.
[I 2026-06-23 09:07:15.671 ServerApp] notebook | extension was successfully linked.
[I 2026-06-23 09:07:15.683 ServerApp] Writing Jupyter server cookie secret to /root/.local/share/jupyter/runtime/jupyter_cookie_secret
[I 2026-06-23 09:07:15.785 ServerApp] notebook_shim | extension was successfully linked.
[I 2026-06-23 09:07:15.791 ServerApp] notebook_shim | extension was successfully loaded.
[I 2026-06-23 09:07:15.792 ServerApp] jupyter_lsp | extension was successfully loaded.
[I 2026-06-23 09:07:15.792 ServerApp] jupyter_server_terminals | extension was successfully loaded.
[I 2026-06-23 09:07:15.793 LabApp] JupyterLab extension loaded from /usr/local/lib/python3.10/dist-packages/jupyterlab
[I 2026-06-23 09:07:15.793 LabApp] JupyterLab application directory is /usr/local/share/jupyter/lab
[I 2026-06-23 09:07:15.793 LabApp] Extension Manager is 'pypi'.
[I 2026-06-23 09:07:15.827 ServerApp] jupyterlab | extension was successfully loaded.
[I 2026-06-23 09:07:15.828 ServerApp] notebook | extension was successfully loaded.
[I 2026-06-23 09:07:15.829 ServerApp] Serving notebooks from local directory: /cheddar
[I 2026-06-23 09:07:15.829 ServerApp] Jupyter Server 2.20.0 is running at:
[I 2026-06-23 09:07:15.829 ServerApp] http://0.0.0.0:9999/tree?token=be68c5873c492f7160699ce617bc12e2c1bfb5ca46713265
[I 2026-06-23 09:07:15.829 ServerApp]     http://127.0.0.1:9999/tree?token=be68c5873c492f7160699
[I 2026-06-23 09:07:15.829 ServerApp] Use Control-C to stop this server and shut down all kerne
[C 2026-06-23 09:07:15.830 ServerApp]

    To access the server, open this file in a browser:
        file:///root/.local/share/jupyter/runtime/jpserver-1-open.html
    Or copy and paste one of these URLs:
        http://54856e7a4530:9999/tree?token=be68c5873c492f7160699ce617bc12e2c1bfb5ca46713265
        http://127.0.0.1:9999/tree?token=be68c5873c492f7160699ce617bc12e2c1bfb5ca46713265
    The server is listening on all interfaces, so any hostname or IP of this machine will work.
[I 2026-06-23 09:07:15.835 ServerApp] Skipped non-installed server(s): basedpyright, bash-language-server, dockerfile-language-server-nodejs, javascript-typescript-language-server, unified-language-server, vscode-css-languageserver-bin, vscode-html-languageserver-bin, vscode-json-languageserver-bin, yaml-language-server
```

Ctrl+Click this link

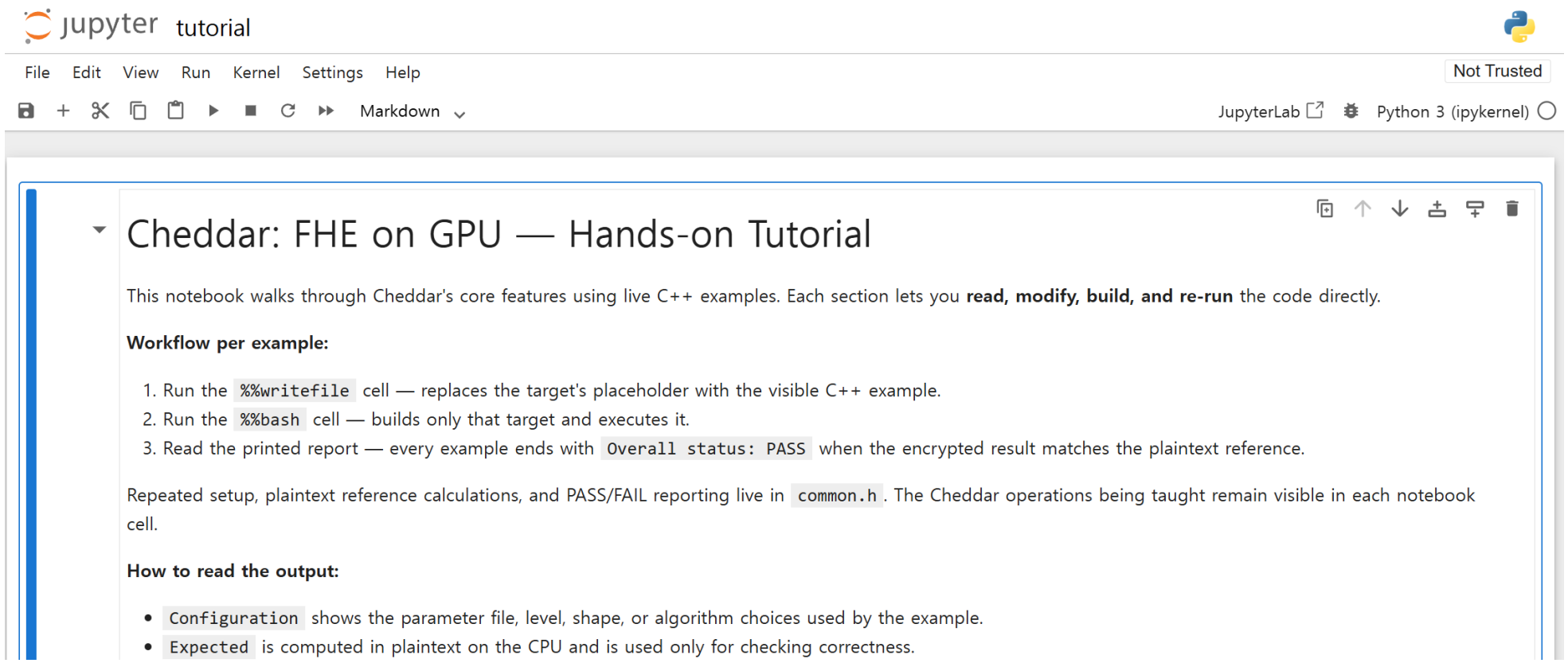
Open Jupyter notebook

- Open the Jupyter notebook at `/tutorial/tutorial.ipynb`



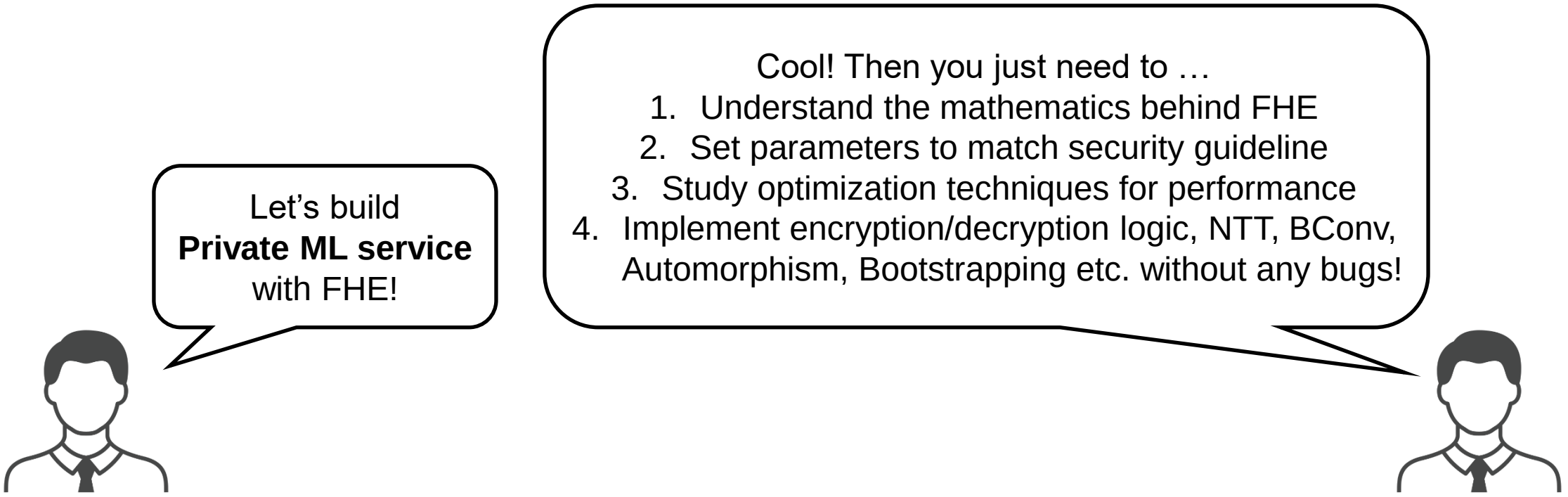
Open Jupyter notebook

- Run each cell with `Shift+Enter`
- The cells write and build C++ source code automatically (`%%writefile` and `%%bash`)

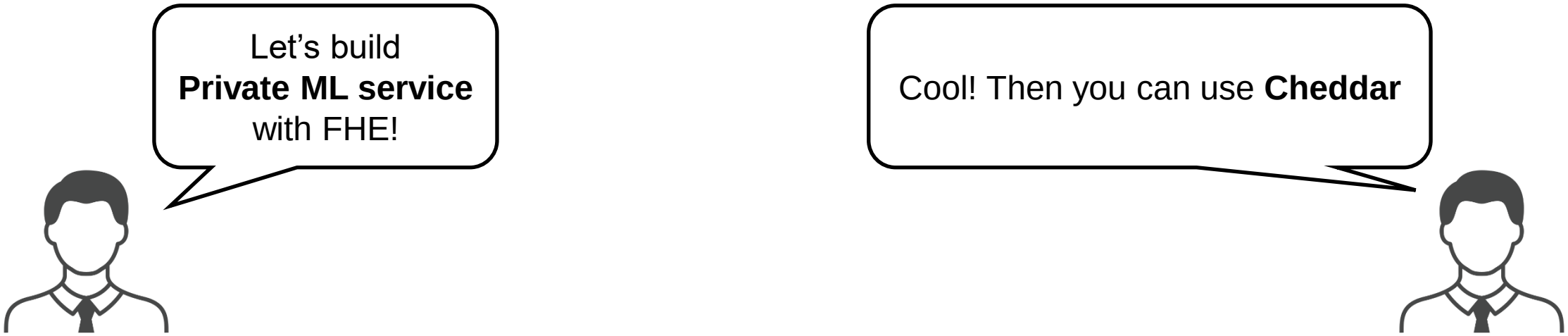


Cheddar: High Performance CKKS Library

Private ML


Let’s build
Private ML service
with FHE!
Cool! Then you just need to …
1. Understand the mathematics behind FHE
2. Set parameters to match security guideline
3. Study optimization techniques for performance
4. Implement encryption/decryption logic, NTT, BConv, Automorphism, Bootstrapping etc. without any bugs!

Private ML


Let’s build
Private ML service
with FHE!
Cool! Then you can use **Cheddar**

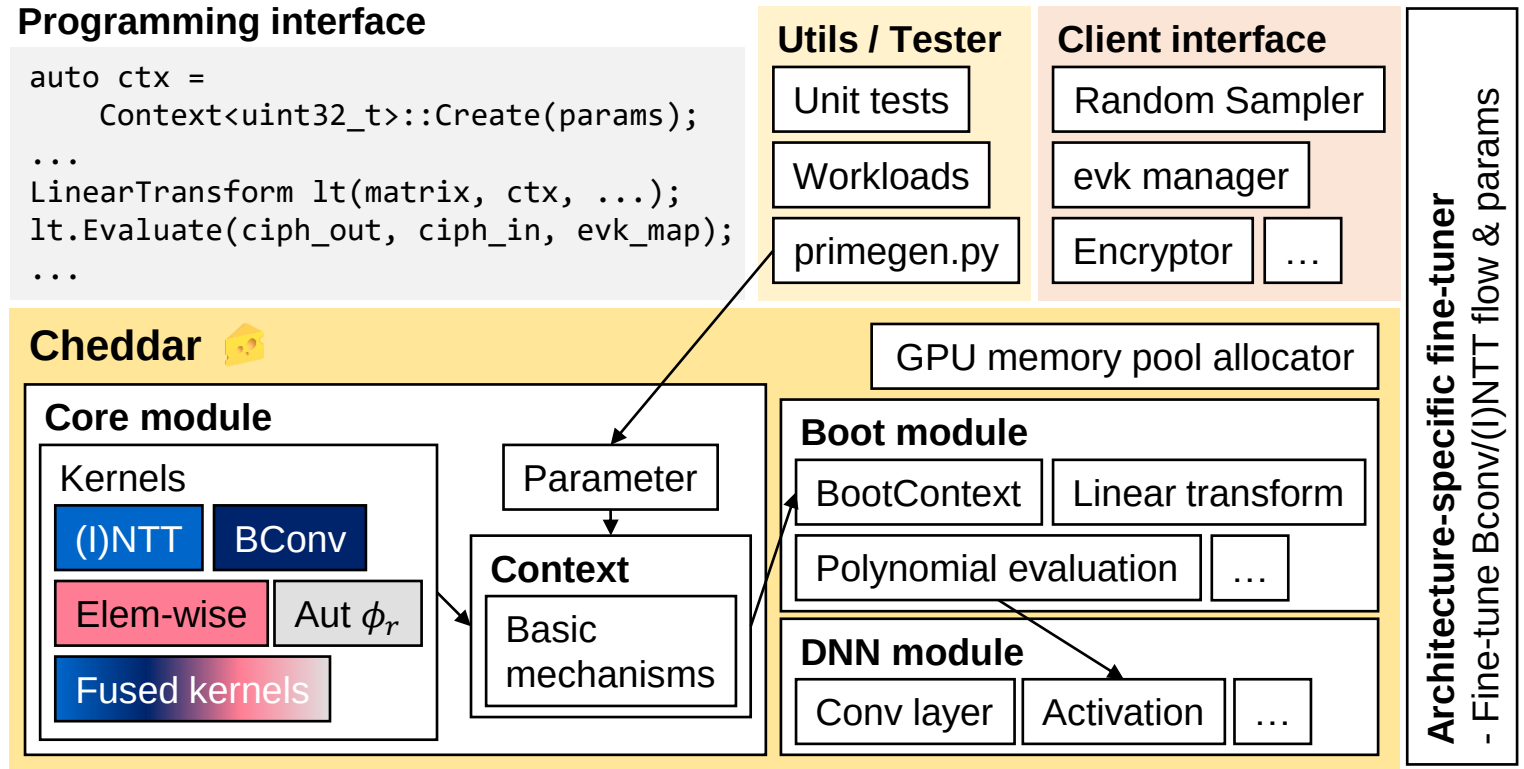
Cheddar [ASPLOS 2026]

- Cheddar is a high-performance **GPU library for FHE**
- Key features

Fast C++/CUDA-based evaluation

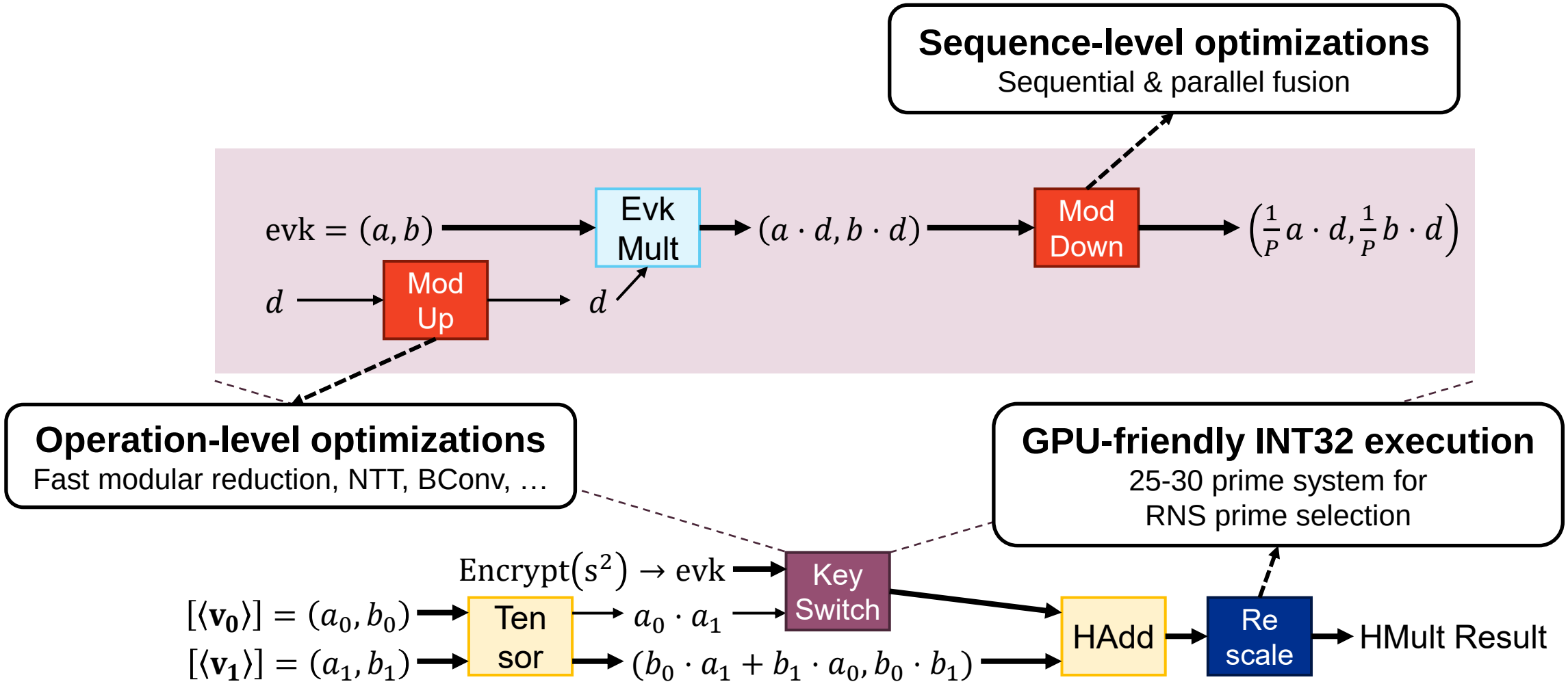
Optimizations for GPU architecture

Simple APIs



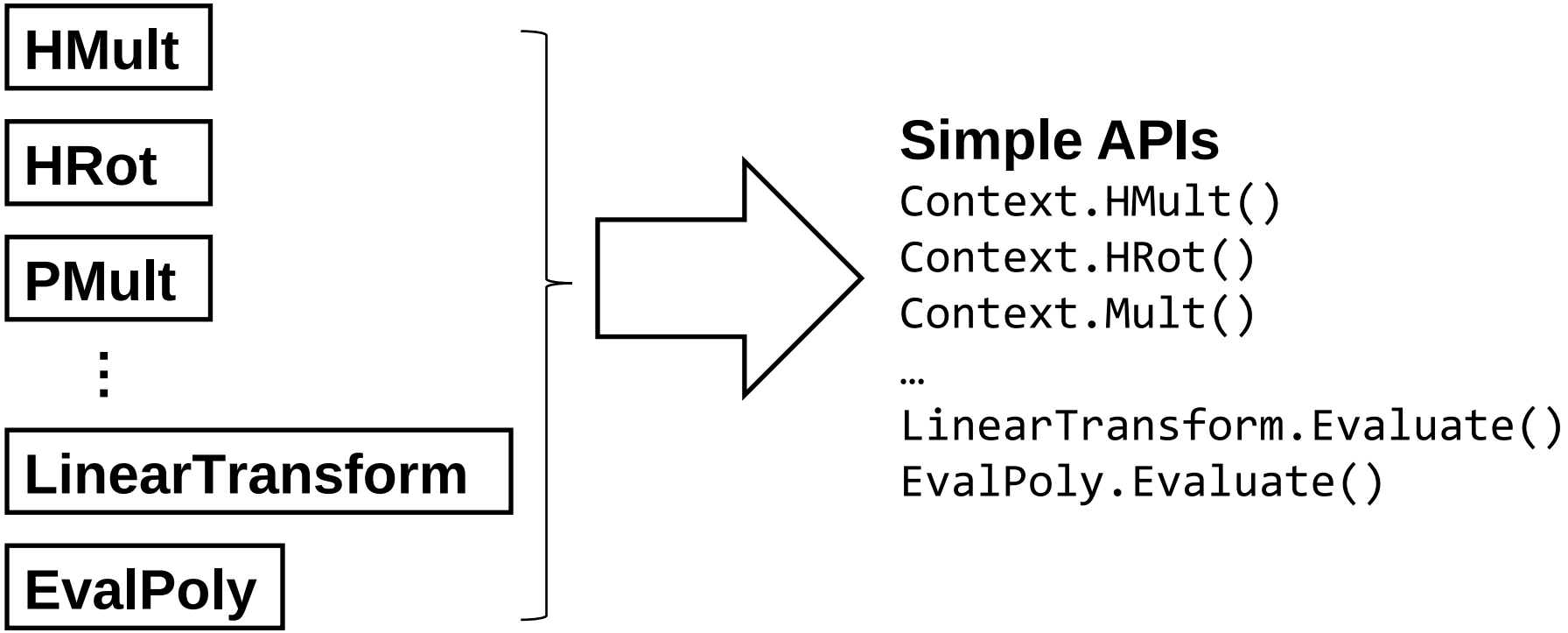
- Cheddar's high performance enables practical FHE services
  - **One of the fastest open-source GPU implementations of CKKS**

HMult with Key-Switching



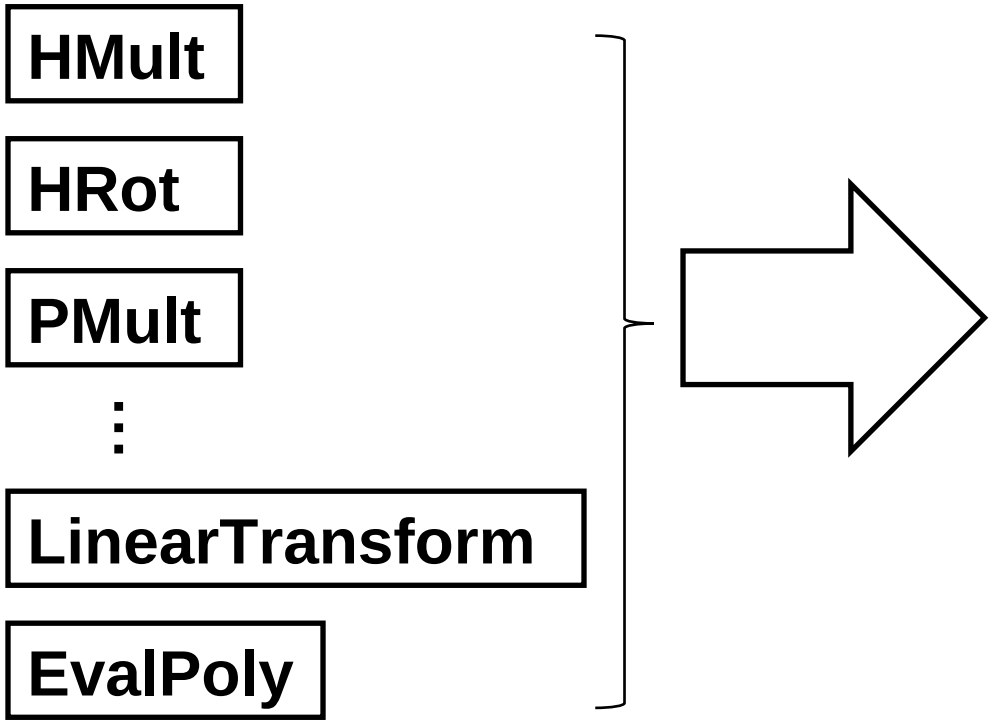
Fast but Simple

- **Simple APIs** hide complex algorithms and low-level optimizations behind a convenient interface



Fast but Simple

- **Simple APIs** hide complex algorithms and low-level optimizations behind a convenient interface
- **Easy INT32/INT64 execution-mode switching** via a template-based structure
  - INT32 mode = Faster, best for getting started
  - INT64 mode = Slower, but compatible with parameters from existing libraries (e.g., Lattigo)



Simple APIs with Templates

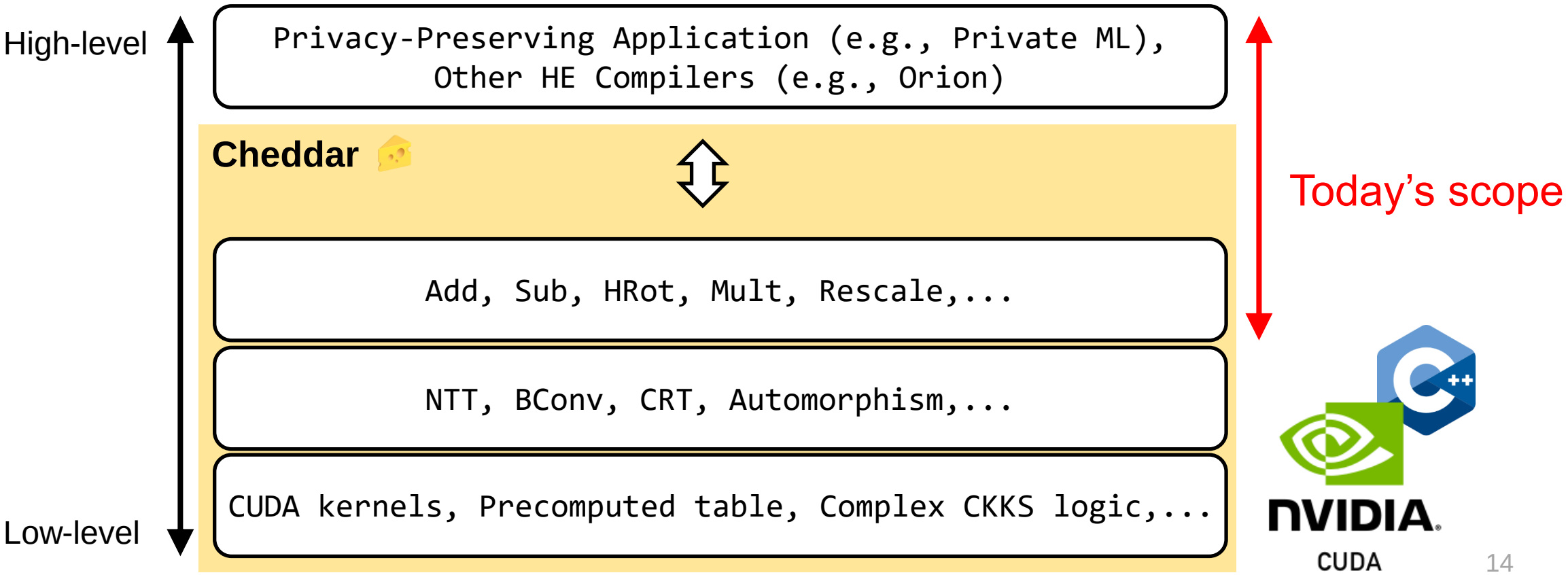
```
Context<word>.HMult()
Context<word>.HRot()
Context<word>.Mult()
…
LinearTransform<word>.Evaluate()
EvalPoly<word>.Evaluate()
```

w/

```
using word = uint32_t;→INT32 mode
using word = uint64_t;→INT64 mode
```

Scope of Cheddar

- You can build your privacy-preserving application on top of **Cheddar**

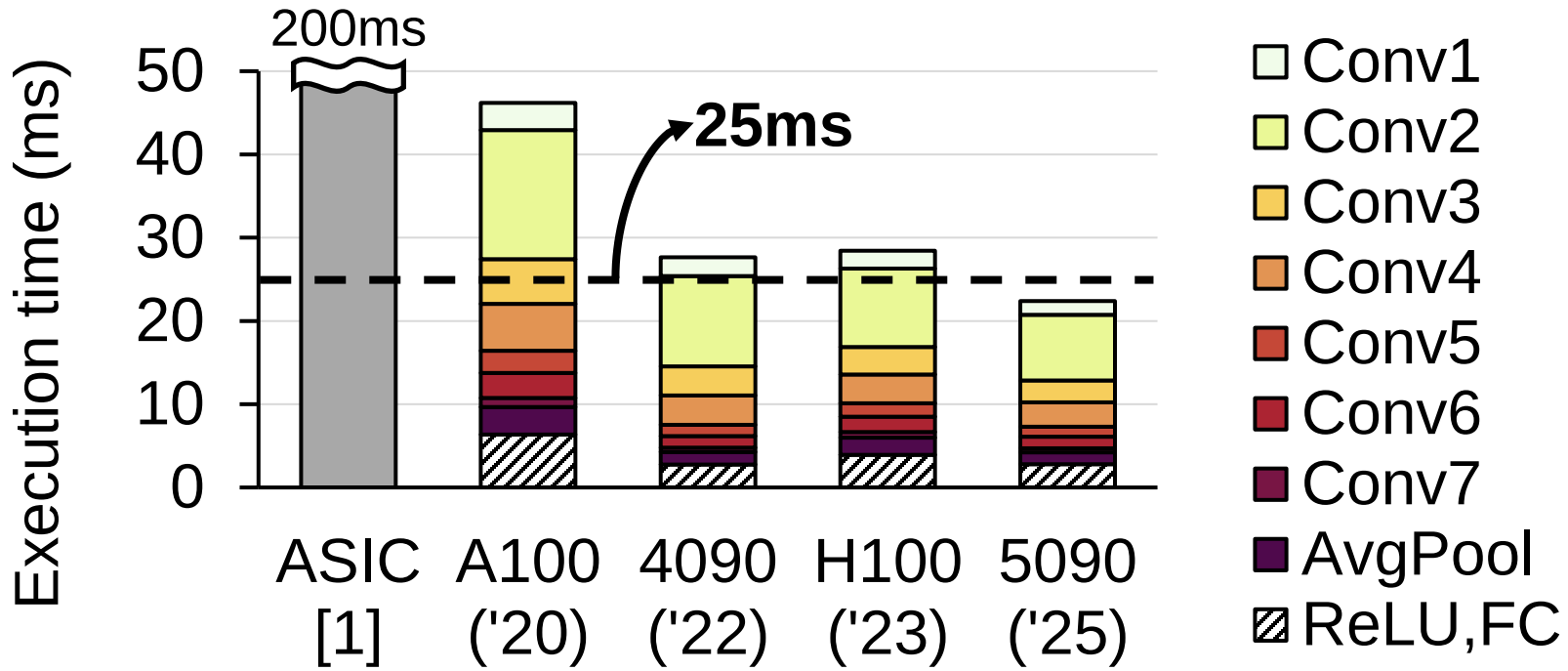


Where to use Cheddar?

- For your research
  - Prototype, experiment, and evaluate faster
- For your service
  - Use end-to-end, or only where privacy matters
- Just for your study
  - Hands-on entry point to FHE

Encrypted 7-Layer CNN (CNN7) inference

- A 7-Layer CNN (CNN7) consists of **7 convolutional layers,** each **followed by ReLU** activation



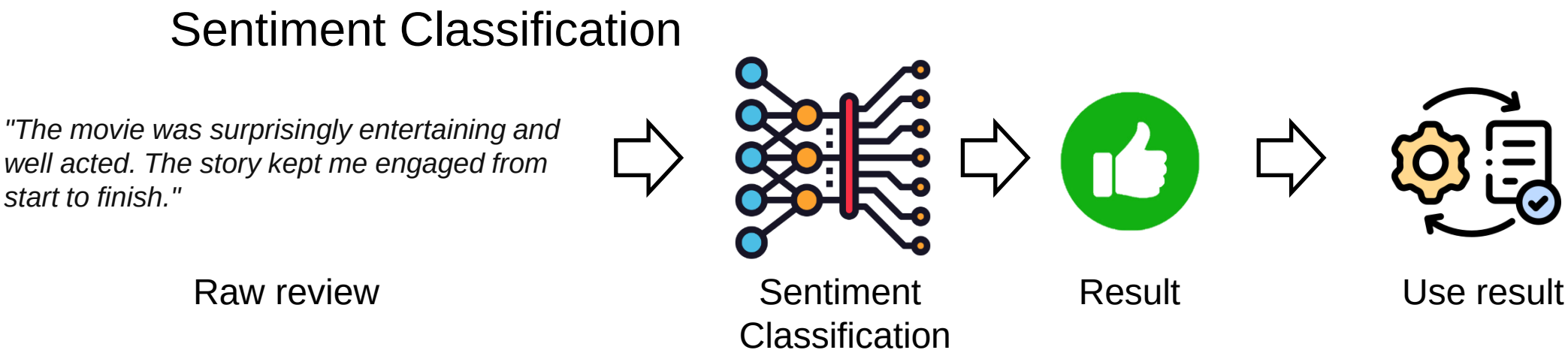
[1] Cousins, David Bruce, et al. "TREBUCHET Fully Homomorphic Encryption Accelerator: Phase Two Performance Estimation Results."

[IEEE MICRO]
"A Five-Year Journey to Accelerate Homomorphic Encryption With GPUs, Demonstrated by Sub-25-ms Convolutional Neural Network Inference,"
W. Choi, S. Kim, J. Park, J. Ahn

From Encryption to Application Building

Sentiment Classification

- **Goal**: Implement movie review sentiment classification

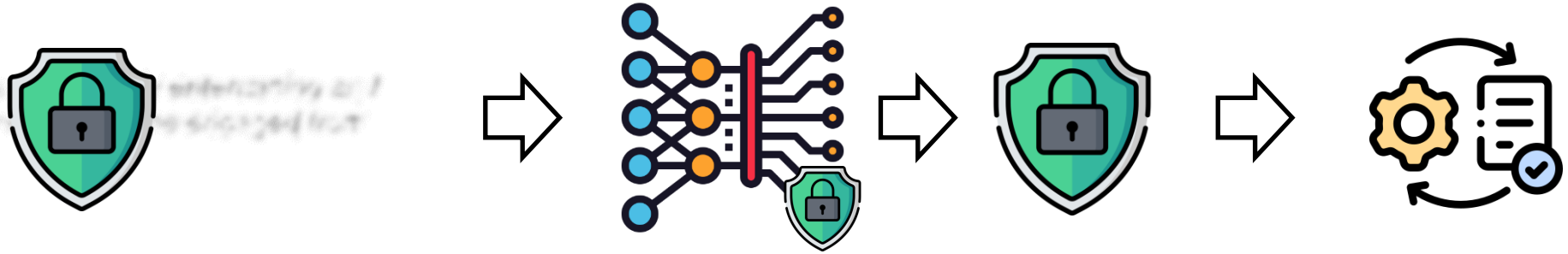


Server can see the client’s review!

Sentiment Classification with FHE

- **Goal**: Implement movie review sentiment classification **without any privacy leakage**

Private Sentiment Classification



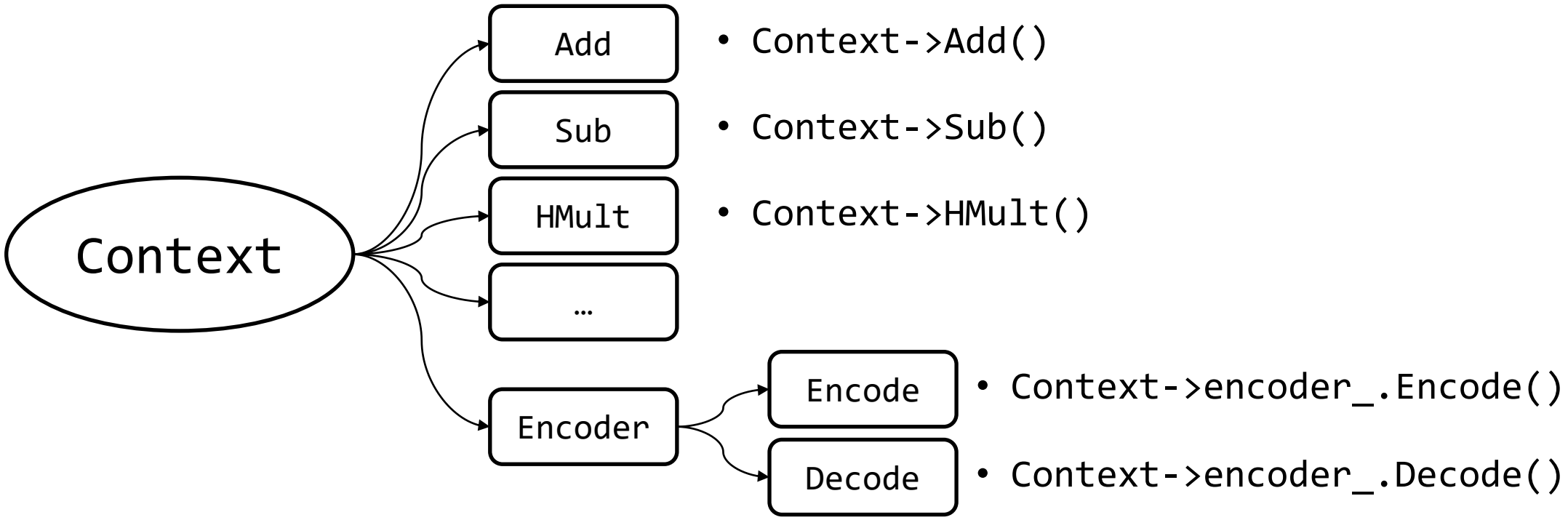
Server **can't** see the client's review!

Hello FHE with Cheddar

- Outline
  - **How to set up computing environment**
  - How to add/mult/sub/rotate ciphertexts
  - How to compute matrix multiplication
  - How to compute non-linear functions (e.g. ReLU)

Context

- `Context` is an environment for encrypted computation
- `Create()` is the only entry point for constructing a `Context` (factory method)




```
static std::shared_ptr<Context<word>> Create(const Parameter<word> &param);
```

CKKS Parameters

- Parameters decide:
  - How many values fit in one ciphertext
  - How much computation can be performed
  - How accurate the decrypted result will be
  - How secure the computation is

CKKS Parameters

- Parameters decide:
  - How many values fit in one ciphertext → `log_degree`
  - How much computation can be performed → `default_encryption_level`
  - How accurate the decrypted result will be → `base_scale`
  - How secure the computation is → **Requires careful parameter selection**

Paper, lattice estimator, …

include/core/Parameter.h

```
// Constructor of Parameter
Parameter(int log_degree, double base_scale,
          int default_encryption_level,
          std::vector<std::pair<int, int>> level_config,
          const std::vector<word> &main_primes,
          const std::vector<word> &aux_primes,
          const std::vector<word> &ter_primes = std::vector<word>{},
          const std::pair<int, int> &additional_base = kZeroPair);
```

CKKS Parameters

- Parameters decide:
  - How many values fit in one ciphertext → `log_degree`
  - How much computation can be performed → `default_encryption_level`
  - How accurate the decrypted result will be → `base_scale`
  - How secure the computation is → **Requires careful parameter selection**
- We provide secure parameter presets in `parameters/*`

parameters/tutorial_param.json

```
“log_degree”: 16,
“log_default_scale”: 40,
“default_encryption_level”: 12,
“level_config”: …
“main_primes”: …
“aux_primes”: ……
```

include/core/Parameter.h

```
// Constructor of Parameter
Parameter(int log_degree, double base_scale,
          int default_encryption_level,
          std::vector<std::pair<int, int>> level_config,
          const std::vector<word> &main_primes,
          const std::vector<word> &aux_primes,
          const std::vector<word> &ter_primes = std::vector<word>{},
          const std::pair<int, int> &additional_base = kZeroPair);
```

Parameter preset

CKKS Parameters

- Parameters decide:
  - How many values fit in one ciphertext → `log_degree`
  - How much computation can be performed → `default_encryption_level`
  - How accurate the decrypted result will be → `base_scale`
  - How secure the computation is → **Requires careful parameter selection**
- We provide secure parameter presets in `parameters/*`

parameters/tutorial_param.json

```
“log_degree”: 16,
“log_default_scale”: 40,
“default_encryption_level”: 12,
“level_config”: …
“main_primes”: …
“aux_primes”: ……
```

`LoadJson`
`&BuildParameter`

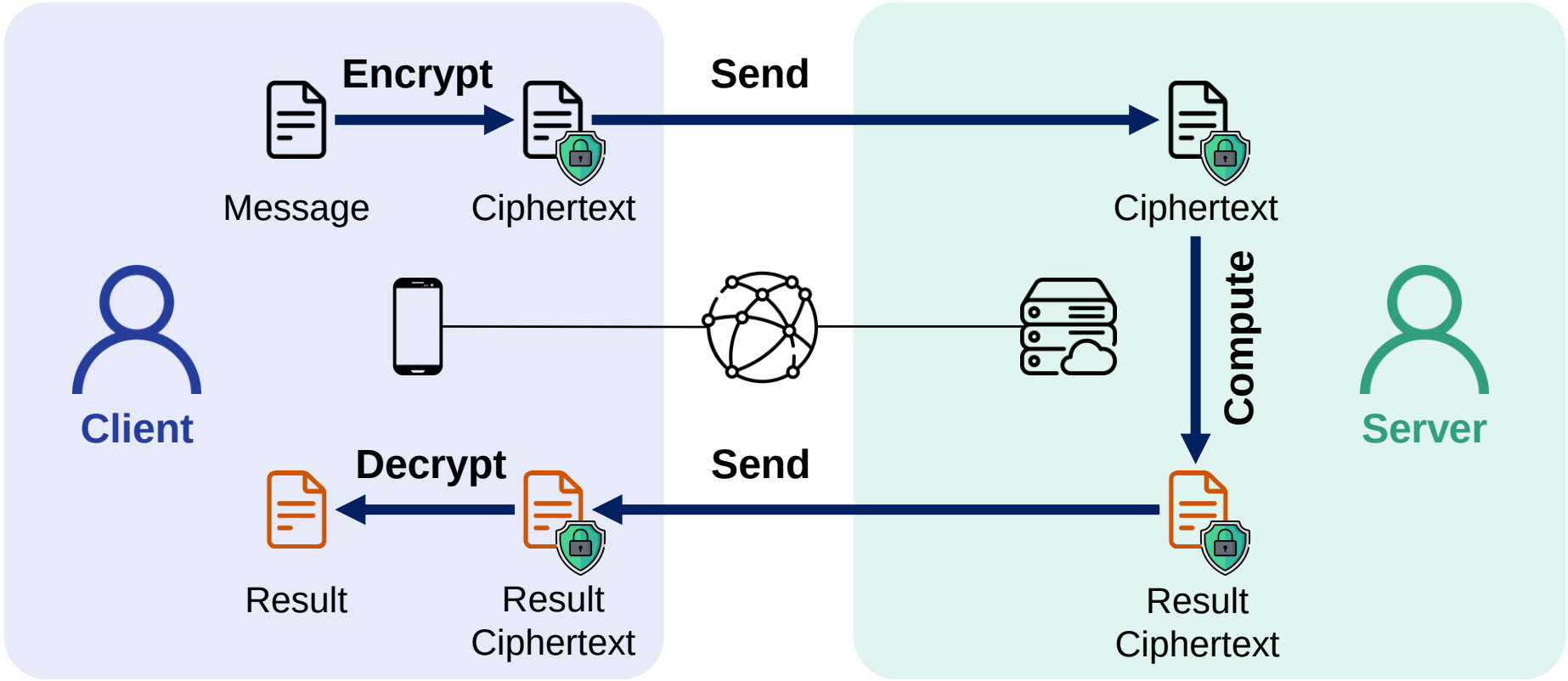
include/core/Parameter.h

```
// Constructor of Parameter
Parameter(int log_degree, double base_scale,
          int default_encryption_level,
          std::vector<std::pair<int, int>> level_config,
          const std::vector<word> &main_primes,
          const std::vector<word> &aux_primes,
          const std::vector<word> &ter_primes = std::vector<word>{},
          const std::pair<int, int> &additional_base = kZeroPair);
```

Parameter preset

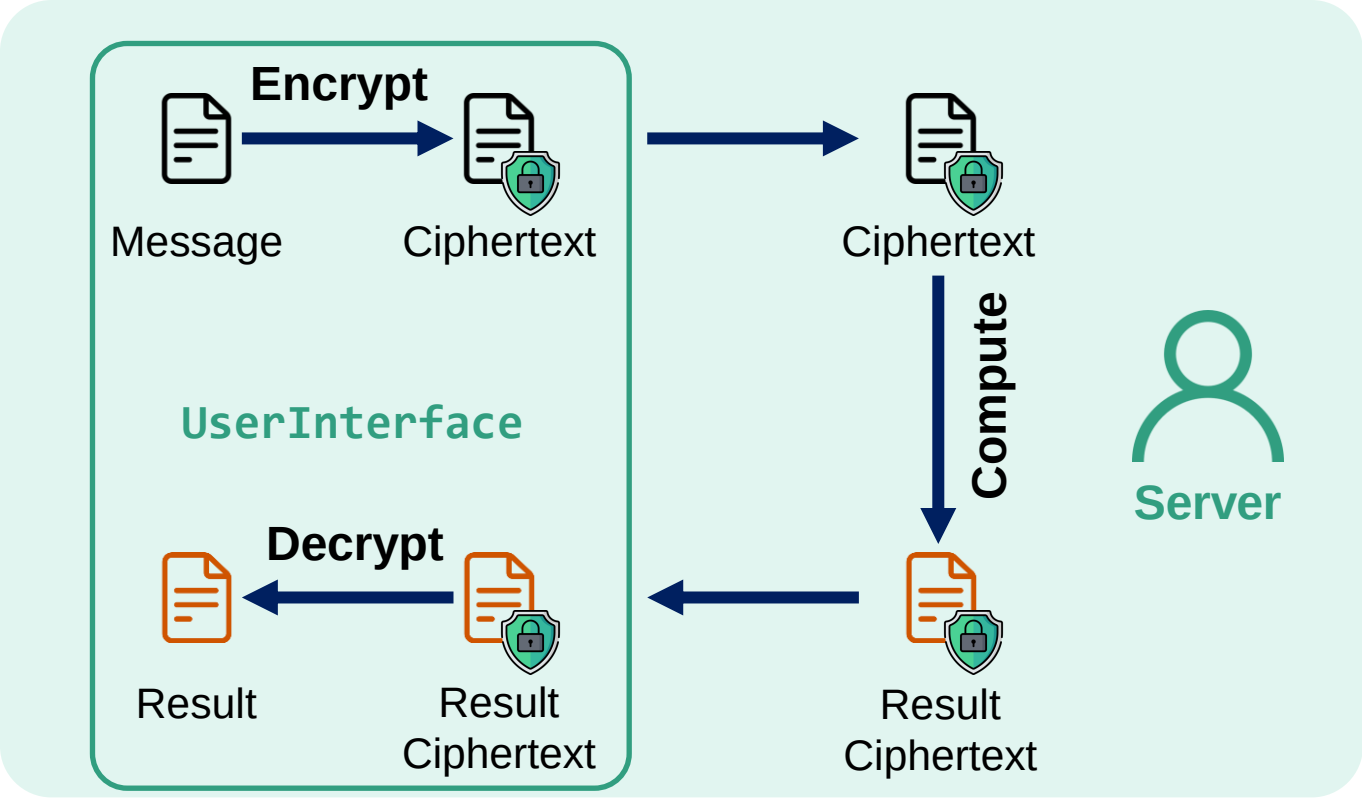
UserInterface

- Typical client-server pipeline
- Hard to test interactively



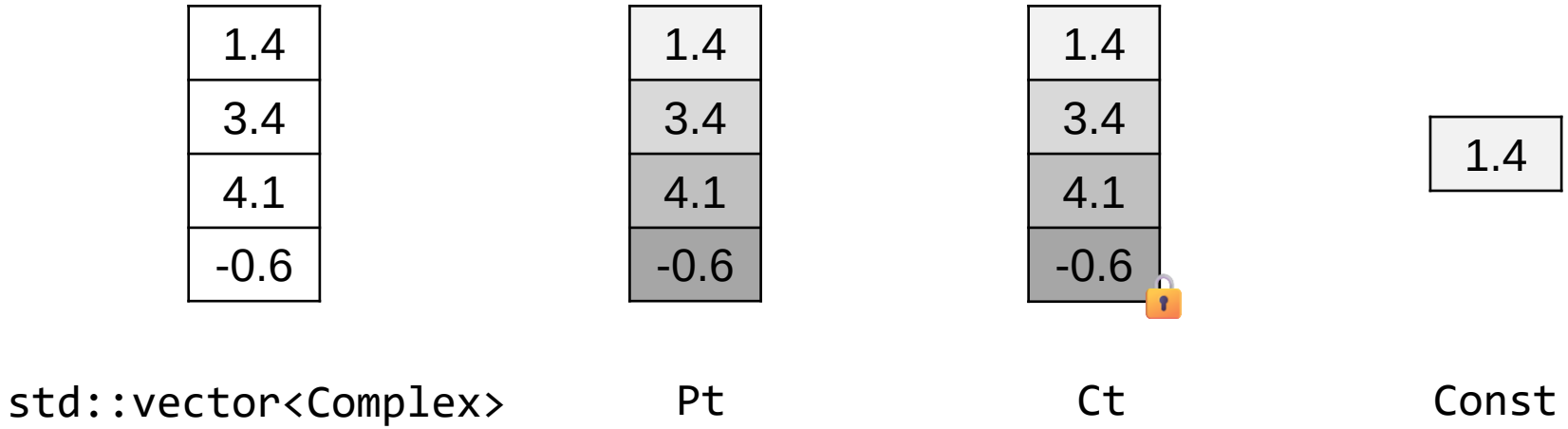
UserInterface

- Testing pipeline: `UserInterface` supports
  - **Test-only** server-side secret key sampling
  - Server-side encryption/decryption
  - Generating evaluation keys for Key Switching



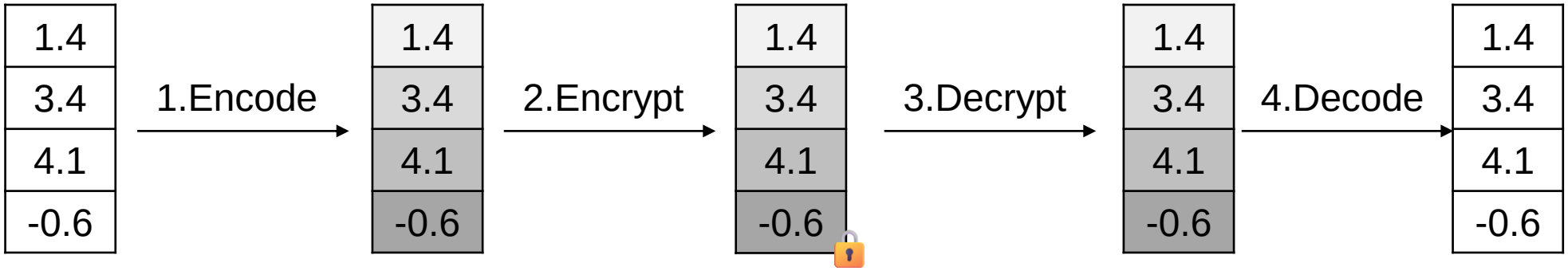
Data Types Cheat Sheet

- Cheddar provides various data types for CKKS in `include/core/Container.h` and `include/core/Type.h`
  - `std::complex<double>(Complex):` data type for messages
  - `Constant<word>(Const):` encoded constant value (not encrypted)
  - `Plaintext<word>(Pt):` encoded vector of messages (not encrypted)
  - `Ciphertext<word>(Ct):` encrypted vector of messages
  - `EvaluationKey<word>(Evk):` special key material used for Key Switching



CKKS Encryption Round Trip

- Cheddar provides simple APIs for encode/decode and encryption/decryption
  1. Encode → `Encoder<word>.Encode()`
  2. Encrypt → `UserInterface<word>.Encrypt()`
  3. Decrypt → `UserInterface<word>.Decrypt()`
  4. Decode → `Encoder<word>.Decode()`



`std::vector<Complex>` `Pt` `Ct` `Pt` `std::vector<Complex>`

Hello FHE with Cheddar

- Outline
  - How to set up computing environment
  - **How to add/mult/sub/rotate ciphertexts**
  - How to compute matrix multiplication
  - How to compute non-linear functions (e.g. ReLU)

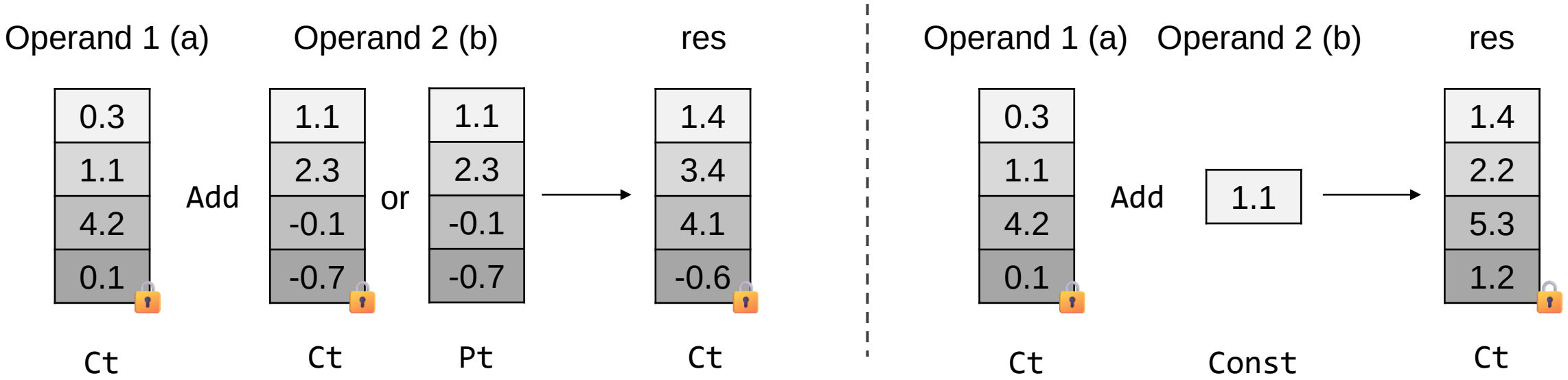
Homomorphic Operations

- Cheddar supports basic homomorphic operations
  - Addition/Subtraction with `Ct/Pt/Const`
  - Multiplication with `Ct/Pt/Const`
  - Negation, Conjugation
  - Cyclic rotation
- More advanced operations are composed of these basic operations
  - Linear transform
  - Polynomial evaluation
  - Bootstrapping

Homomorphic Addition

- `Add/Sub` supports `Ct-Ct`, `Ct-Pt`, and `Ct-Const` addition/subtraction

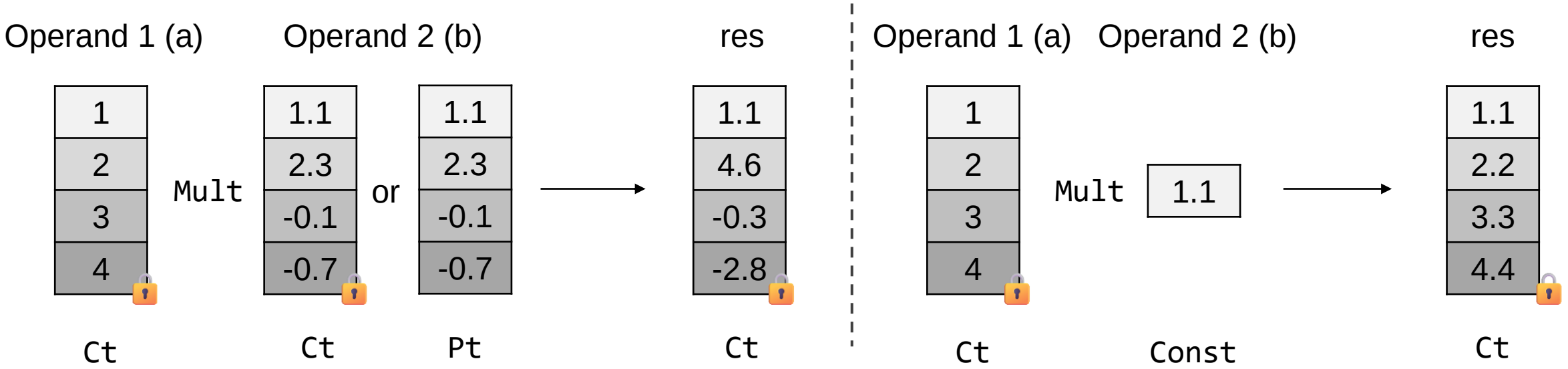
```
void Add(Ct &res, const Ct &a, const Ct &b) const;
void Add(Ct &res, const Ct &a, const Pt &b) const;
void Add(Ct &res, const Ct &a, const Const &b) const;
```



Homomorphic Multiplication

- Supports `Ct-Ct` and `Ct-Pt/Const` multiplication

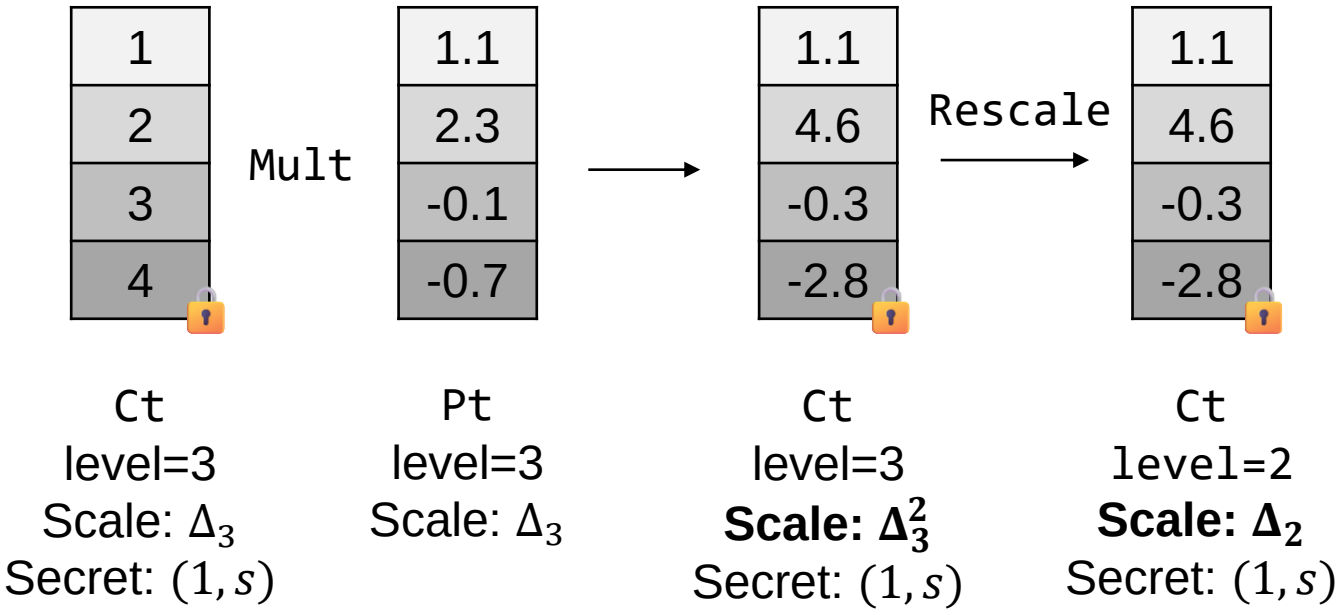
```
void Mult(Ct &res, const Ct &a, const Ct &b) const;
void Mult(Ct &res, const Ct &a, const Pt &b) const;
void Mult(Ct &res, const Ct &a, const Const &b) const;
```



Homomorphic Multiplication

- Supports `Ct-Ct` and `Ct-Pt/Const` multiplication
- `Rescale` is required after multiplication

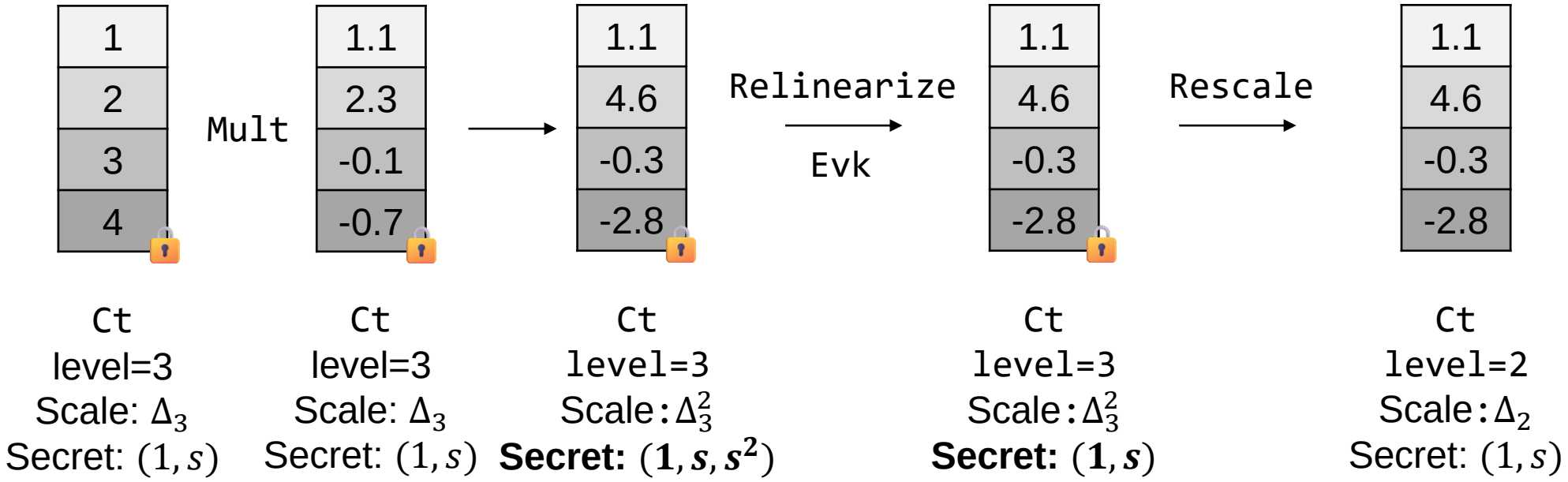
```
void Rescale(Ct &res, const Ct &a) const;
```



Homomorphic Multiplication

- Supports `Ct-Ct` and `Ct-Pt/Const` multiplication
- `Rescale` is required after multiplication
- `Ct-Ct` multiplication requires `Relinearization` with `Evk`

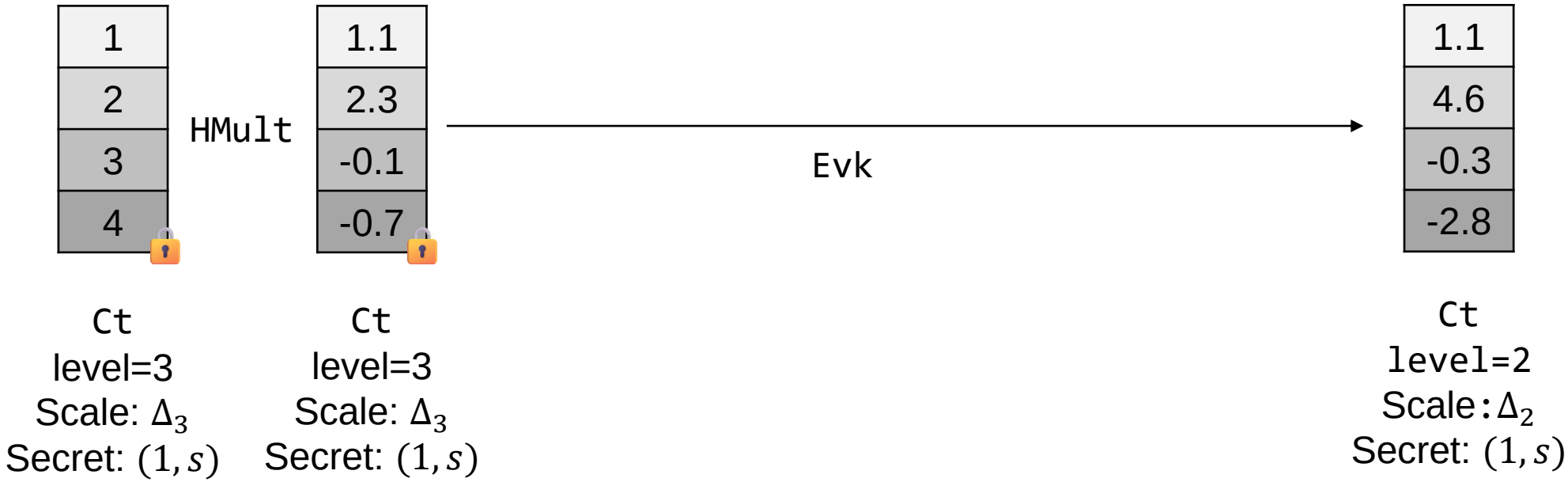
```
void Relinearize(Ct &res, const Ct &a, const Evk &key) const;
```



Homomorphic Multiplication

- Supports `Ct-Ct` and `Ct-Pt/Const` multiplication
- `Rescale` is required after multiplication
- `Ct-Ct` multiplication requires `Relinearization` with `Evk`
- `HMult` fuses `Mult+Relinearize+Rescale`

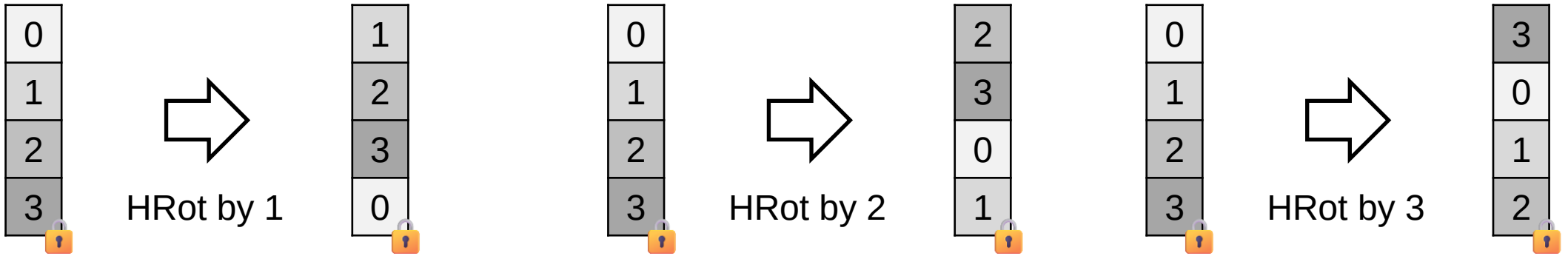
```
void HMult(Ct &res, const Ct &a, const Ct &b, const Evk &mult_key, bool rescale = true) const;
```



Homomorphic Rotation

- `HRot` rotates an encrypted vector by a given amount using a specific `Evk`
- You can get `Evk` with `ui->GetEvkMap().GetRotationKey(amount)`
  - Or directly `ui->GetRotationKey(amount)`

```
void HRot(Ct &res, const Ct &a, const Evk &rot_key, int rot_dist) const;
```



Need `Evk` for 1

Need `Evk` for 2

Need `Evk` for 3

EvkMap

- `Ct-Ct` Multiplication, Rotation, Conjugation require `Evk` for KeySwitching
- All `Evks` are managed by `EvkMap` in `UserInterface`

  e.g.) `ui->GetEvkMap().GetMultiplicationKey()` or directly `ui->GetMultiplicationKey()`

  `ui->GetEvkMap().GetRotationKey(amount)` or `ui->GetRotationKey(amount)`

`EvkMap`

Key	Value
kMultiplicationKeyIndex	Evk for multiplication
1	Evk for rotation amount of 1
2	Evk for rotation amount of 2
32	Evk for rotation amount of 32
…	…

Multiplication key is prepared by default

Need to prepare explicitly with `ui->PrepareRotationKey(amount, level)`

Hello FHE with Cheddar

- Outline
  - How to set up computing environment
  - How to add/mult/sub/rotate ciphertexts
  - **How to compute matrix multiplication**
  - How to compute non-linear functions (e.g. ReLU)

Matrix-Vector Multiplication

- Matrix (unencrypted)-Vector (unencrypted) multiplication (GEMV)

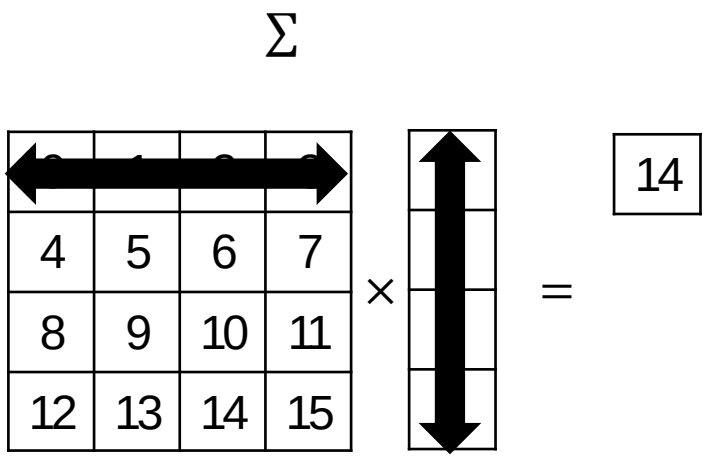
0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

$\times$

0
1
2
3

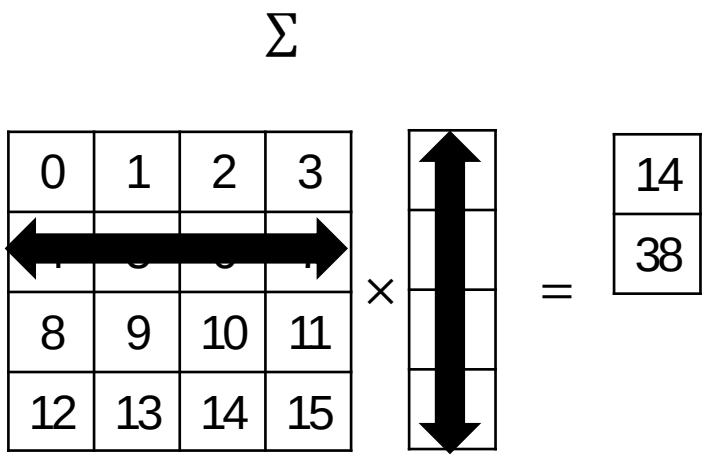
Matrix-Vector Multiplication

- Matrix (unencrypted)-Vector (unencrypted) multiplication (GEMV)



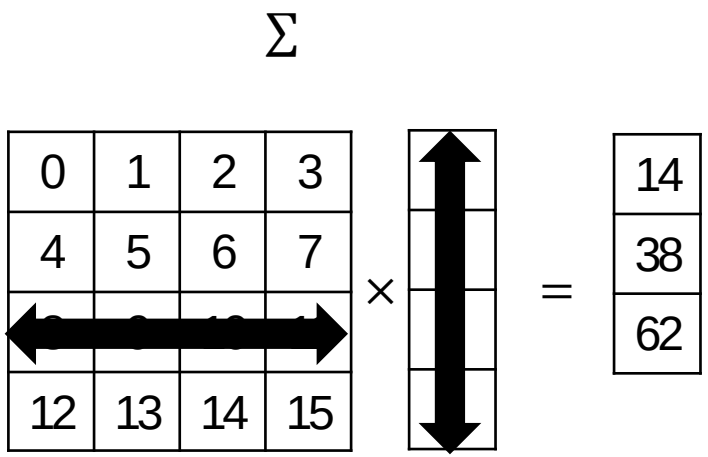
Matrix-Vector Multiplication

- Matrix (unencrypted)-Vector (unencrypted) multiplication (GEMV)



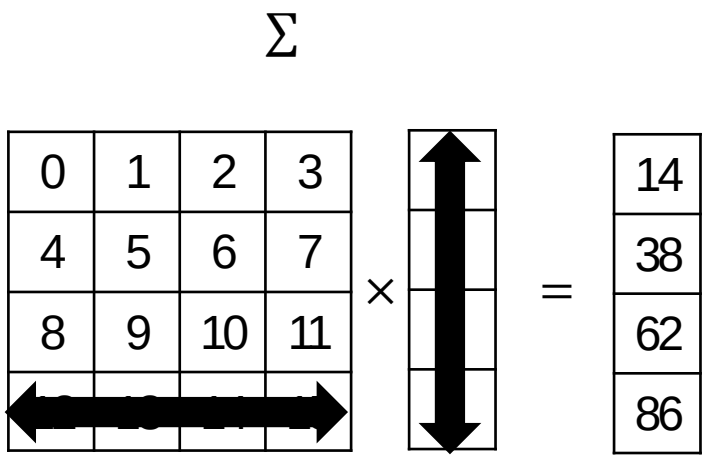
Matrix-Vector Multiplication

- Matrix (unencrypted)-Vector (unencrypted) multiplication (GEMV)



Matrix-Vector Multiplication

- Matrix (unencrypted)-Vector (unencrypted) multiplication (GEMV)



Matrix-Vector Multiplication

- Matrix (unencrypted)-Vector (unencrypted) multiplication (GEMV)

0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

$\times$

0
1
2
3

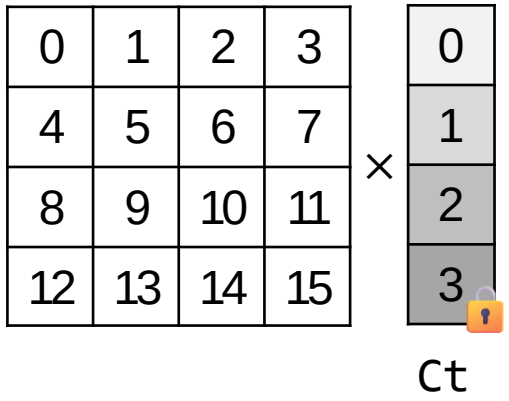
$=$

14
38
62
86

Matrix-Vector Multiplication

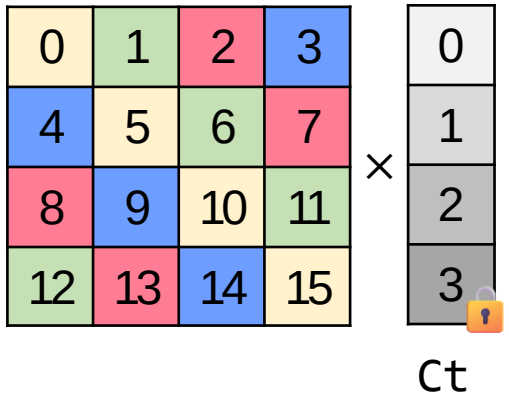
- Matrix (unencrypted)-Vector (**encrypted**) multiplication (GEMV)

How to pack matrix as `Pt`?



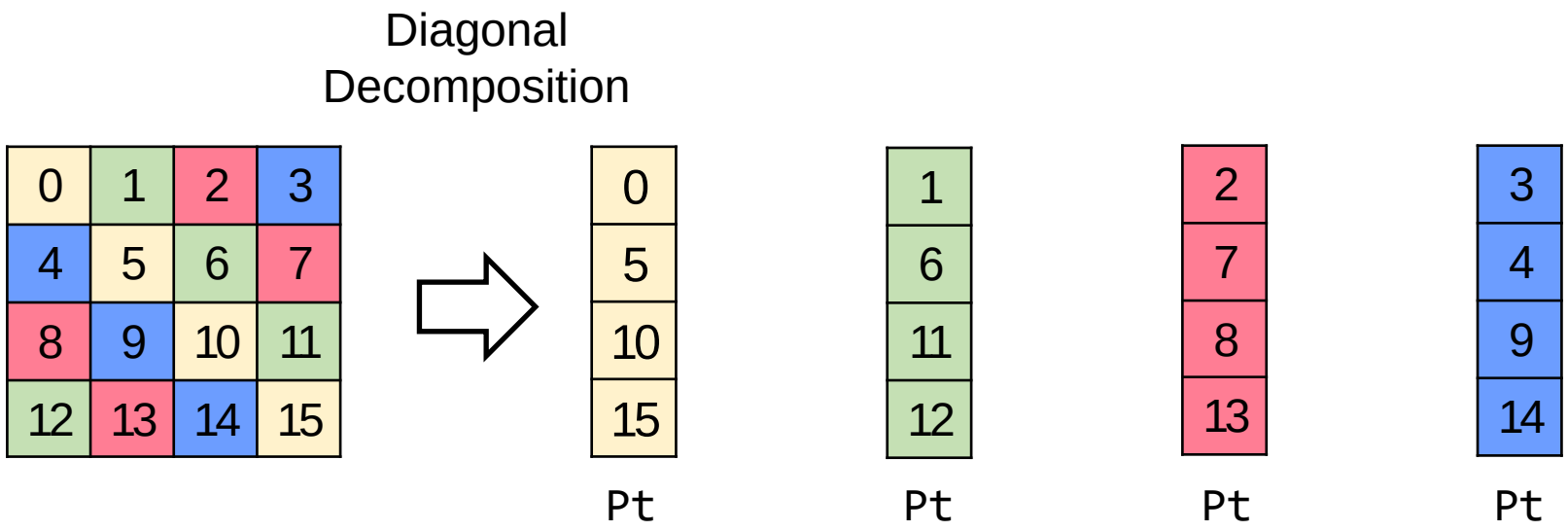
Matrix-Vector Multiplication

- Matrix (unencrypted)-Vector (**encrypted**) multiplication (GEMV) → **Diagonal method**



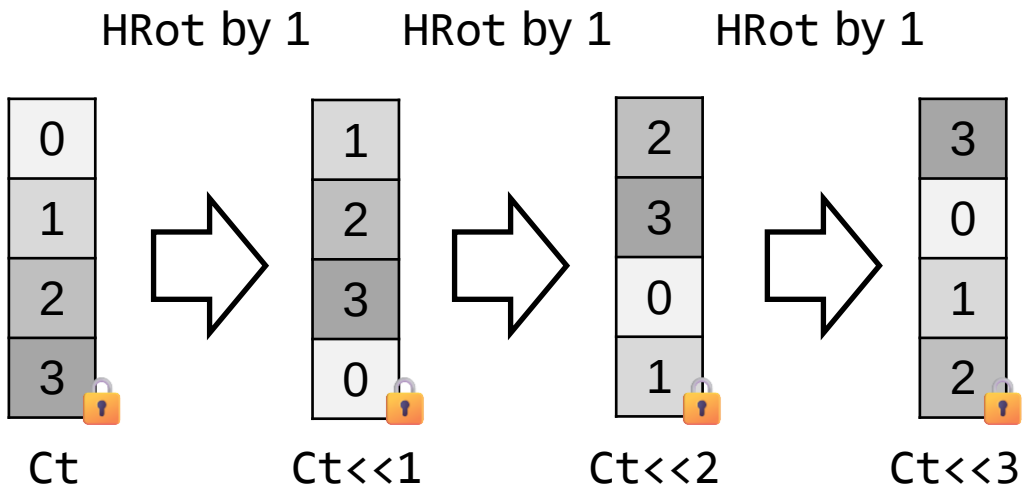
Matrix-Vector Multiplication

- Matrix (unencrypted)-Vector (**encrypted**) multiplication (GEMV) → **Diagonal method**



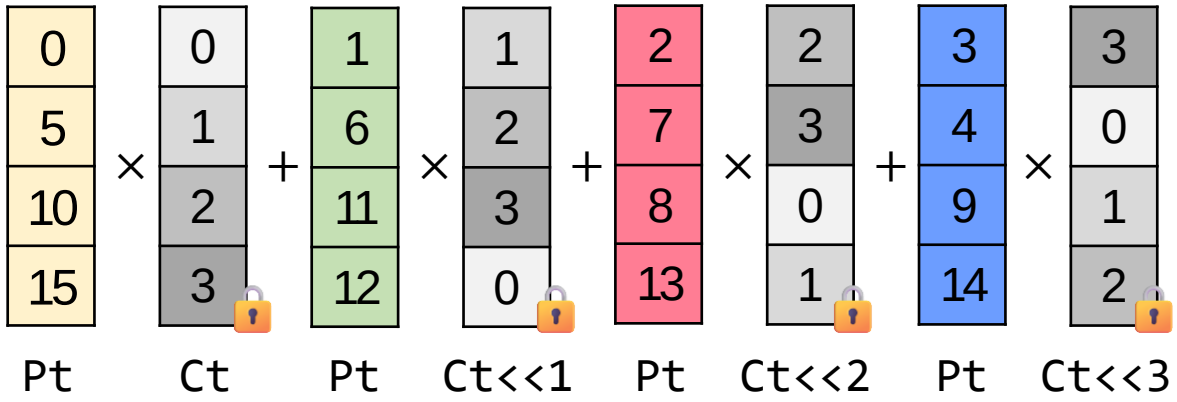
Matrix-Vector Multiplication

- Matrix (unencrypted)-Vector (**encrypted**) multiplication (GEMV) → **Diagonal method**



Matrix-Vector Multiplication

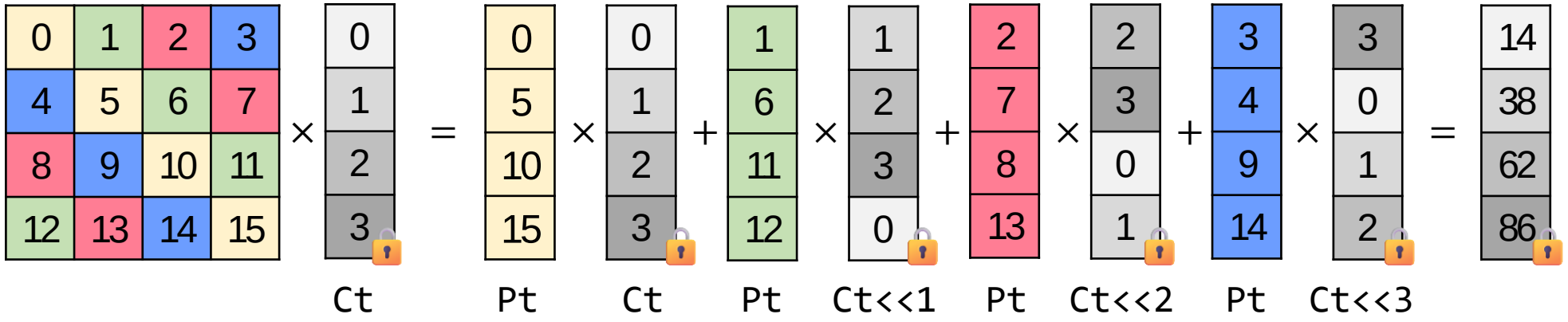
- Matrix (unencrypted)-Vector (**encrypted**) multiplication (GEMV) → **Diagonal method**



<<a: HRot by a ×: Mult +: Add

Matrix-Vector Multiplication

- Matrix (unencrypted)-Vector (**encrypted**) multiplication (GEMV) → **Diagonal method**
- GEMV = `HRots + Mults + Adds`



<<a: HRot by a ×: Mult +: Add

LinearTransform

- `LinearTransform` performs GEMV (`include/extension/LinearTransform.h`)
- `StripedMatrix` is a diagonally indexed matrix (`include/extension/StripedMatrix.h`)
- Set (bs,gs) ≈ $(\sqrt{N}, \sqrt{N})$ for $N \times N$ matrix (bs×gs=N)
- For now, ignore `pre_rotation` and `additional_pt_rot`

include/extension/LinearTransform.h

```
// Constructor of LinearTransform
LinearTransform(ConstContextPtr<word> context, const StripedMatrix &matrix,
                int pt_level, double pt_scale, int bs, int gs = 1,
                int pre_rotation = 0, int additional_pt_rot = 0);
```

LinearTransform

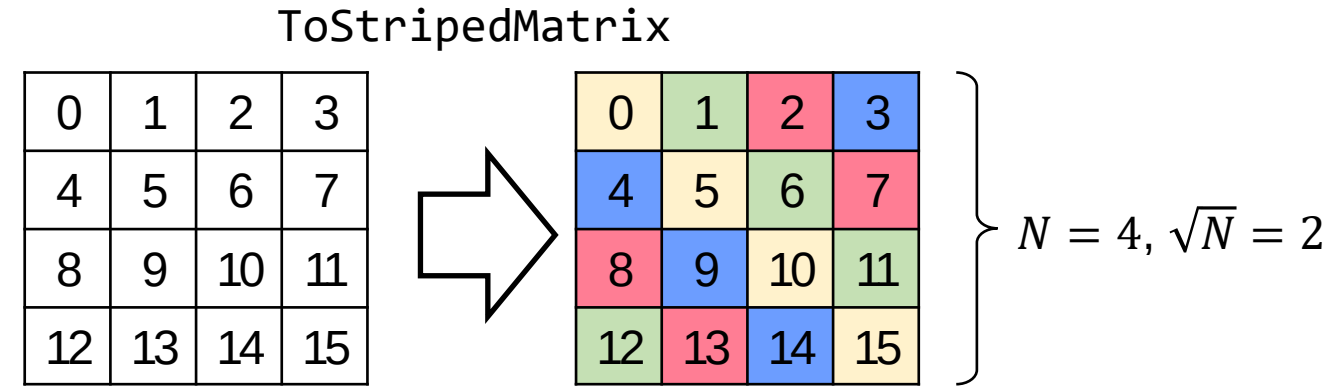
- `Matrix = std::vector<std::vector<double>>`

0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

```
Matrix a = {
  {0,  1,  2,  3},
  {4,  5,  6,  7},
  {8,  9, 10, 11},
  {12, 13, 14, 15},
};
```

LinearTransform

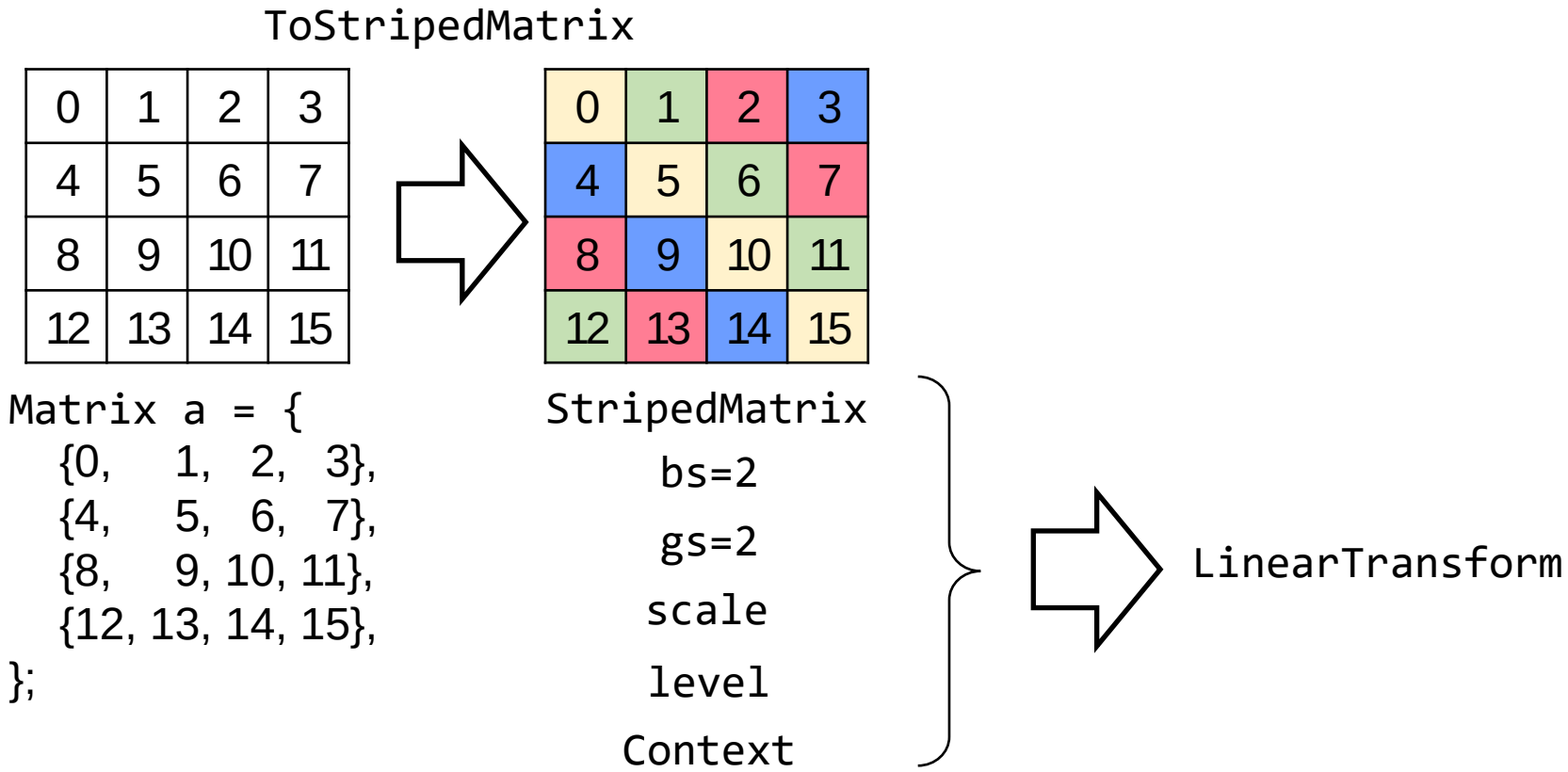
- `Matrix = std::vector<std::vector<double>>`
- We provide `ToStripedMatrix` helper in `tutorial/common.h`




```
Matrix a = {
   {0,  1,  2,  3},
   {4,  5,  6,  7},
   {8,  9, 10, 11},
   {12, 13, 14, 15},
};
```

LinearTransform

- `Matrix = std::vector<std::vector<double>>`
- We provide `ToStripedMatrix` helper in `tutorial/common.h`



LinearTransform

- `AddRequiredRotation` compute the rotation keys needed during LinearTransform

include/extension/LinearTransform.h

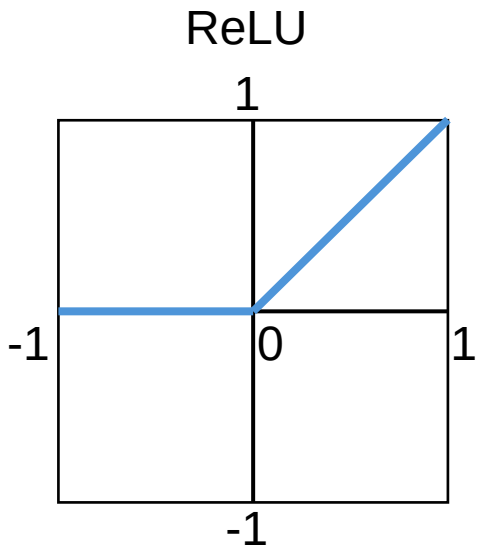
```
// Constructor of LinearTransform
LinearTransform(ConstContextPtr<word> context, const StripedMatrix &matrix,
                int pt_level, double pt_scale, int bs, int gs = 1,
                int pre_rotation = 0, int additional_pt_rot = 0);
…
// For preparing rotation keys needed by LinearTransform
void AddRequiredRotations(EvkRequest &req, bool min_ks = false) const;
…
// Evaluator
void Evaluate(ConstContextPtr<word> context, Ct &res, const Ct &input,
              const EvkMap<word> &evk_map, bool min_ks = false) const;
```

Hello FHE with Cheddar

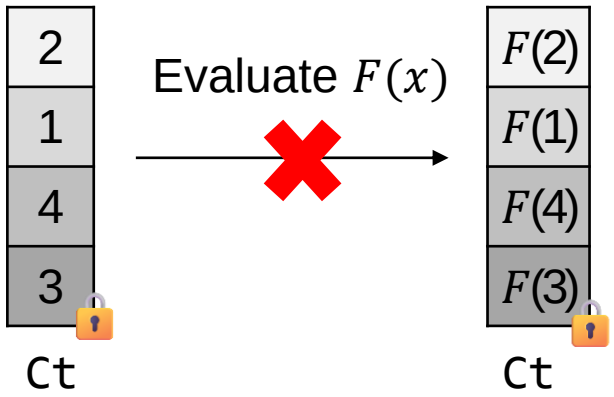
- Outline
  - How to set up computing environment
  - How to add/mult/sub/rotate ciphertexts
  - How to compute matrix multiplication
  - **How to compute non-linear functions (e.g. ReLU)**

Non-linear operation

- Non-linear operation is hard to evaluate as CKKS only supports `Add` and `Mult`

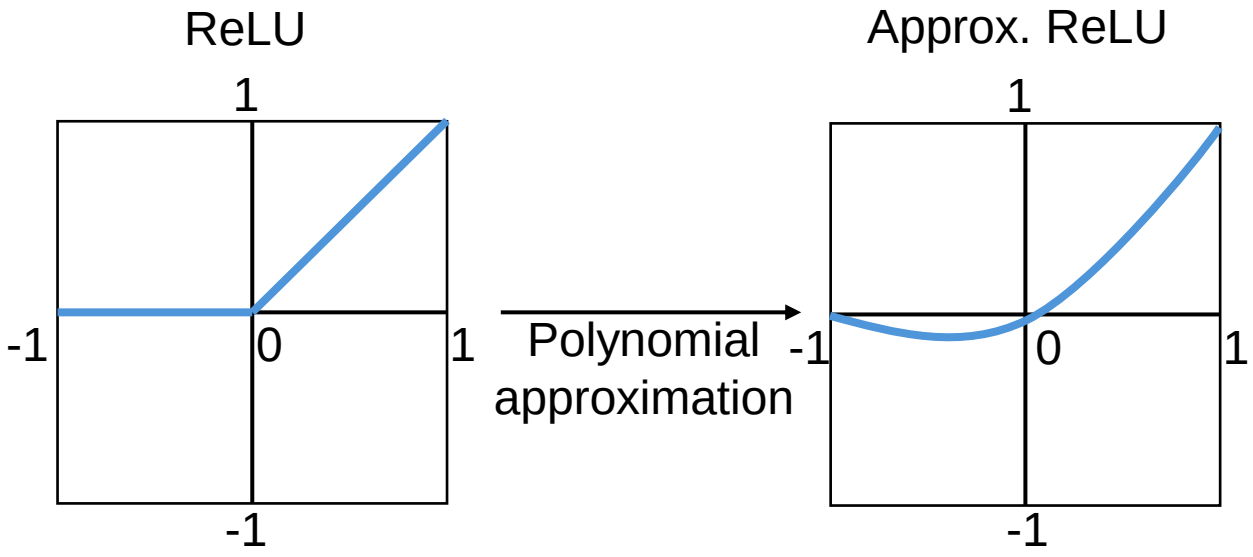


$$F(x) = \frac{x + xsgn(x)}{2}$$



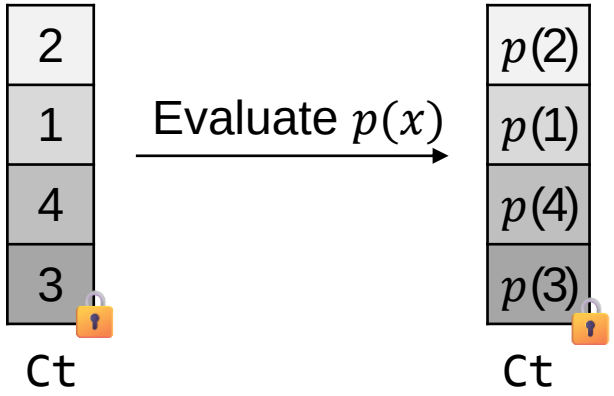
Polynomial Evaluation

- Non-linear operation is hard to evaluate as CKKS only supports `Add` and `Mult`
- In CKKS, non-linear functions are approximated with polynomials



$$F(x) = \frac{x + xsgn(x)}{2}$$

$$p(x) = ax^2 + bx + c$$



Polynomial Evaluation

- The evaluation order matters in polynomial evaluation

e.g.) Evaluate $p(x) = ax^{11} + bx^{10} + cx^9 + dx^8 + ex^7 + fx^6 + gx^5 + hx^4 + ix^3 + jx^2 + kx + l$

Horner method: $p(x) = \big(\cdots\big((ax + b)x + cx\big) + \cdots + k\big)x + l$

Level consumption: $D = 11$
Multiplication: $D = 11$

Paterson-Stockmeyer: $p(x) = Q_0 + Q_1 x^4 + Q_2 x^8$

$$Q_0 = l + kx + jx^2 + ix^3$$
$$Q_1 = h + gx + fx^2 + ex^3$$
$$Q_2 = d + cx + bx^2 + ax^3$$

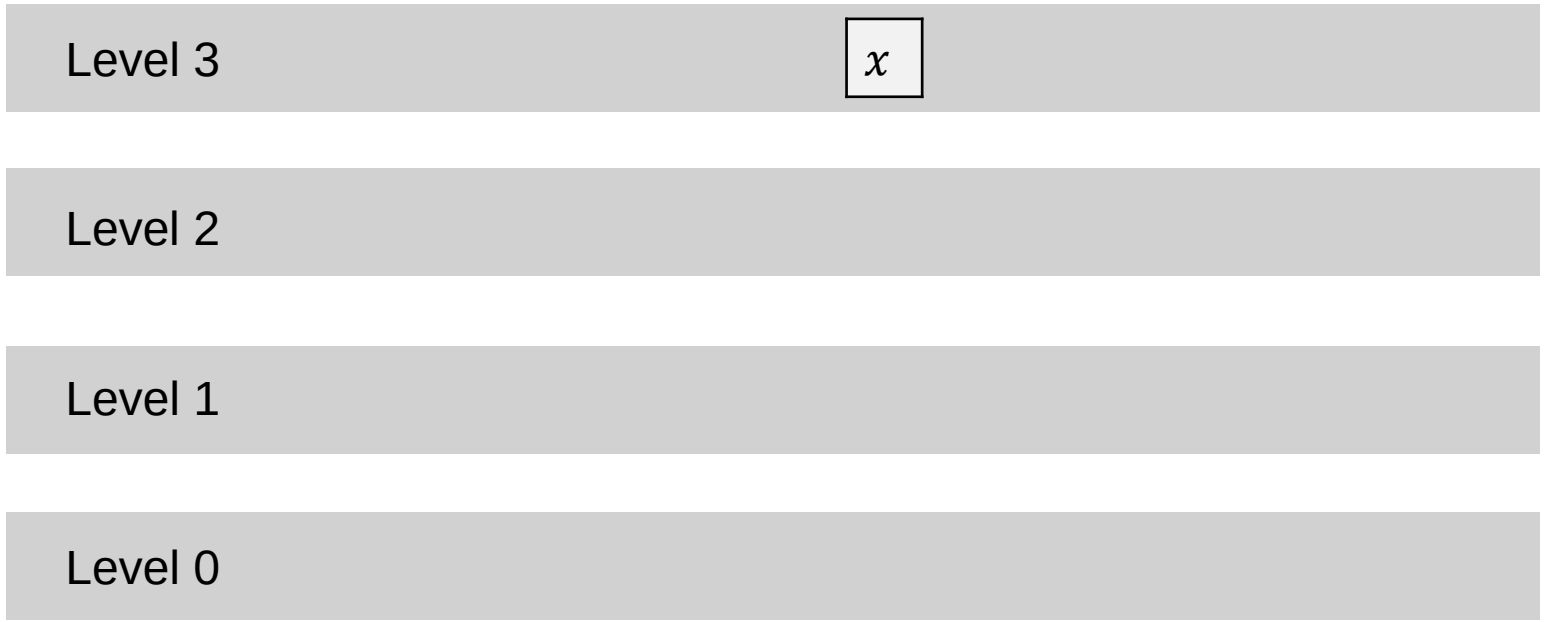
Level consumption: $\approx \lceil \log_2 \mathrm{D} \rceil = 4$
Multiplication: $\approx 2\sqrt{D}$

Polynomial Evaluation

- `EvalPoly` evaluates polynomial using an evaluation tree for performance, numerical stability, lower level consumption (defined in `include/extension/EvalPoly.h`)

e.g.) Evaluate $p(x) = a \cdot x^5 + \mathrm{b} \cdot x^4 + c \cdot x^3 + d \cdot x^2 + e \cdot x + f$

$$= (a \cdot x + b) \cdot x^4 + c \cdot x^3 + d \cdot x^2 + e \cdot x + f$$



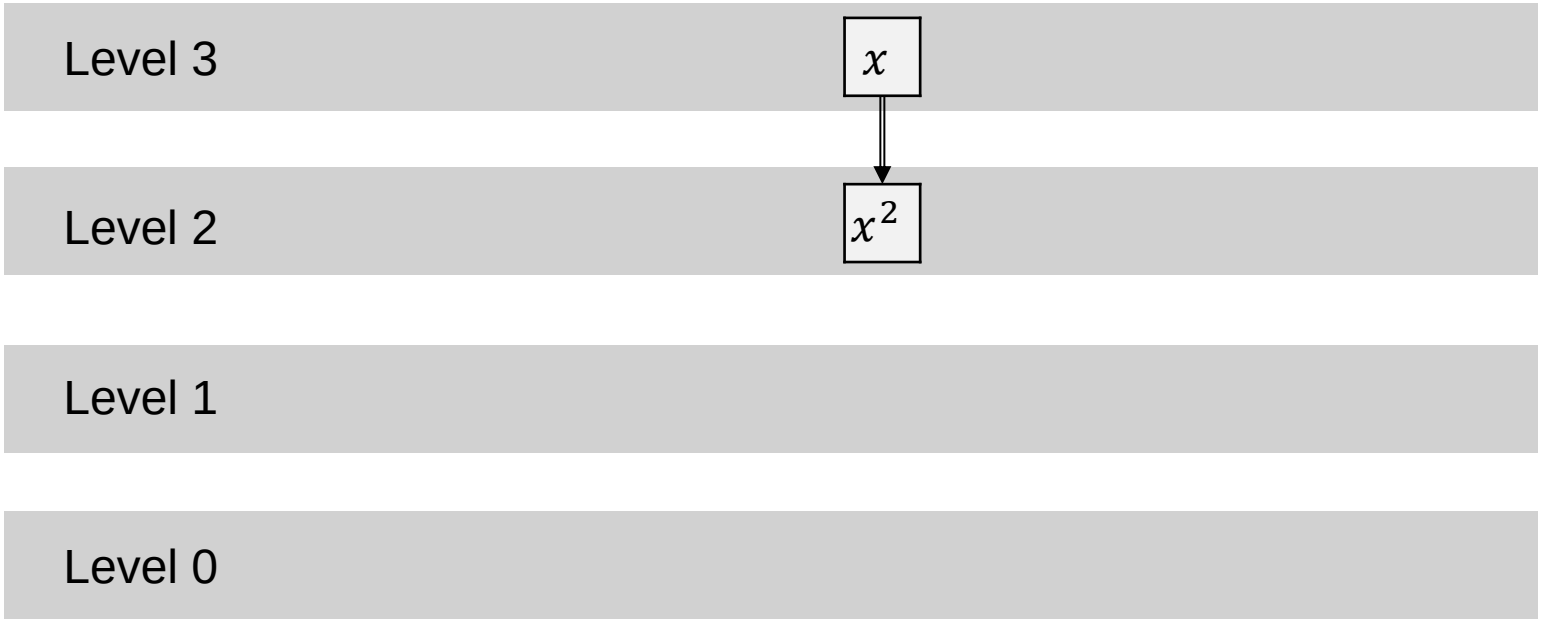
Step 1. Prepare basis

Polynomial Evaluation

- `EvalPoly` evaluates polynomial using an evaluation tree for performance, numerical stability, lower level consumption (defined in `include/extension/EvalPoly.h`)

e.g.) Evaluate $p(x) = a \cdot x^5 + \mathrm{b} \cdot x^4 + c \cdot x^3 + d \cdot x^2 + e \cdot x + f$

$$= (a \cdot x + b) \cdot x^4 + c \cdot x^3 + d \cdot x^2 + e \cdot x + f$$



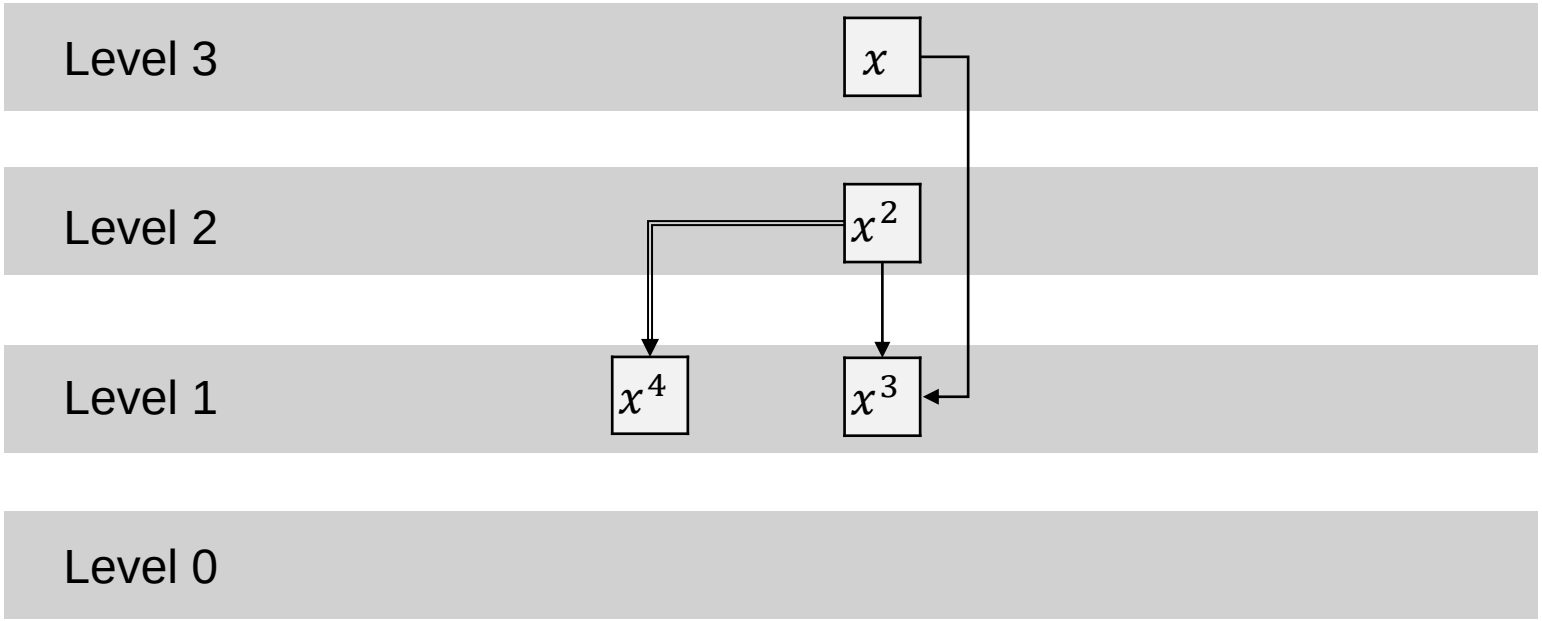
Step 1. Prepare basis

Polynomial Evaluation

- `EvalPoly` evaluates polynomial using an evaluation tree for performance, numerical stability, lower level consumption (defined in `include/extension/EvalPoly.h`)

e.g.) Evaluate $p(x) = a \cdot x^5 + \mathrm{b} \cdot x^4 + c \cdot x^3 + d \cdot x^2 + e \cdot x + f$

$$= (a \cdot x + b) \cdot x^4 + c \cdot x^3 + d \cdot x^2 + e \cdot x + f$$



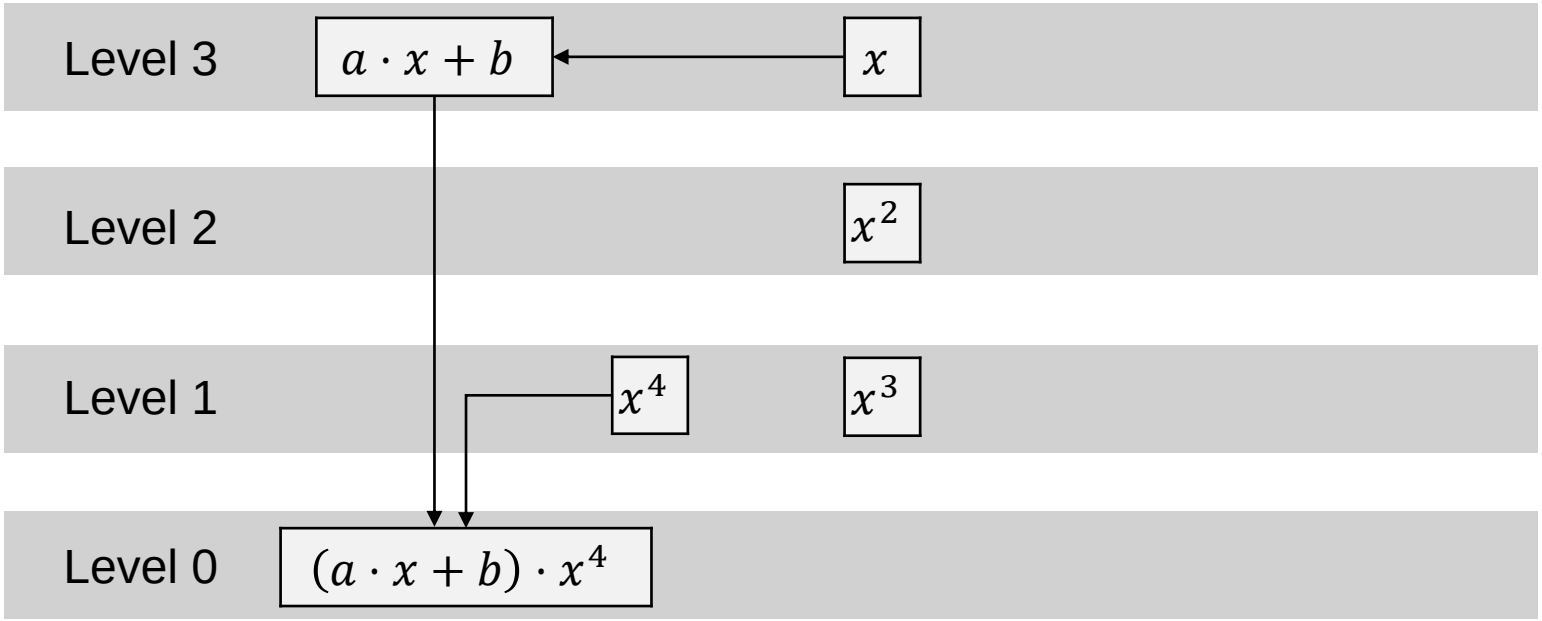
Step 1. Prepare basis

Polynomial Evaluation

- `EvalPoly` evaluates polynomial using an evaluation tree for performance, numerical stability, lower level consumption (defined in `include/extension/EvalPoly.h`)

e.g.) Evaluate $p(x) = a \cdot x^5 + \mathrm{b} \cdot x^4 + c \cdot x^3 + d \cdot x^2 + e \cdot x + f$

$= (a \cdot x + b) \cdot x^4 + c \cdot x^3 + d \cdot x^2 + e \cdot x + f$



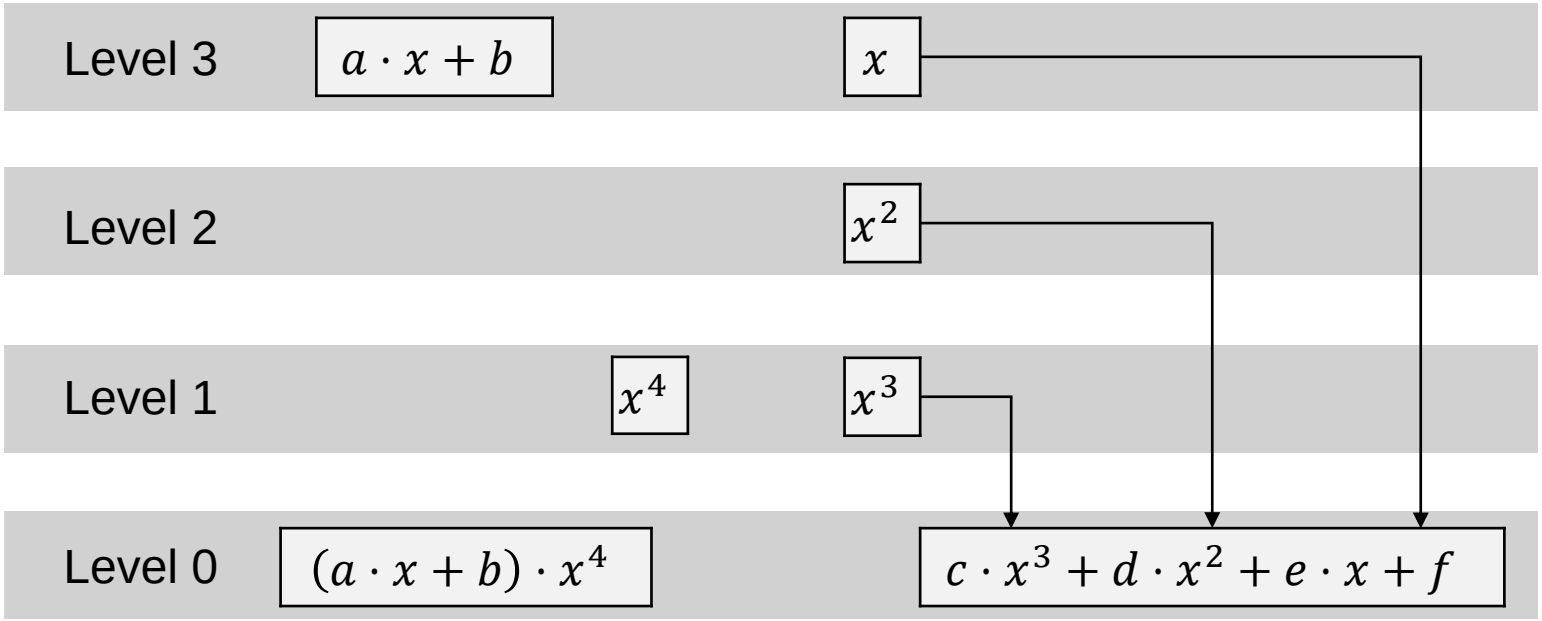
Step 2. Evaluation tree

Polynomial Evaluation

- `EvalPoly` evaluates polynomial using an evaluation tree for performance, numerical stability, lower level consumption (defined in `include/extension/EvalPoly.h`)

e.g.) Evaluate $p(x) = a \cdot x^5 + \mathrm{b} \cdot x^4 + c \cdot x^3 + d \cdot x^2 + e \cdot x + f$

$$= (a \cdot x + b) \cdot x^4 + c \cdot x^3 + d \cdot x^2 + e \cdot x + f$$



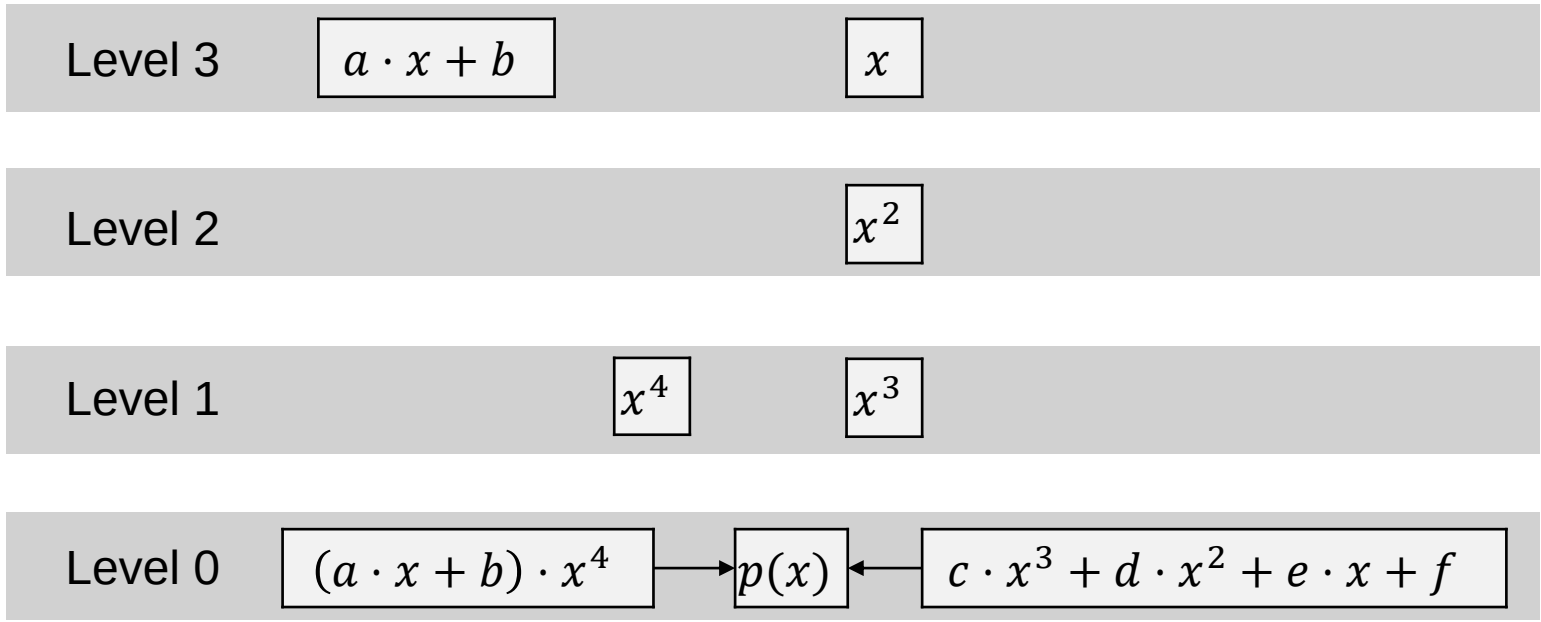
Step 2. Evaluation tree

Polynomial Evaluation

- `EvalPoly` evaluates polynomial using an evaluation tree for performance, numerical stability, lower level consumption (defined in `include/extension/EvalPoly.h`)

e.g.) Evaluate $p(x) = a \cdot x^5 + \mathrm{b} \cdot x^4 + c \cdot x^3 + d \cdot x^2 + e \cdot x + f$

$= (a \cdot x + b) \cdot x^4 + c \cdot x^3 + d \cdot x^2 + e \cdot x + f$



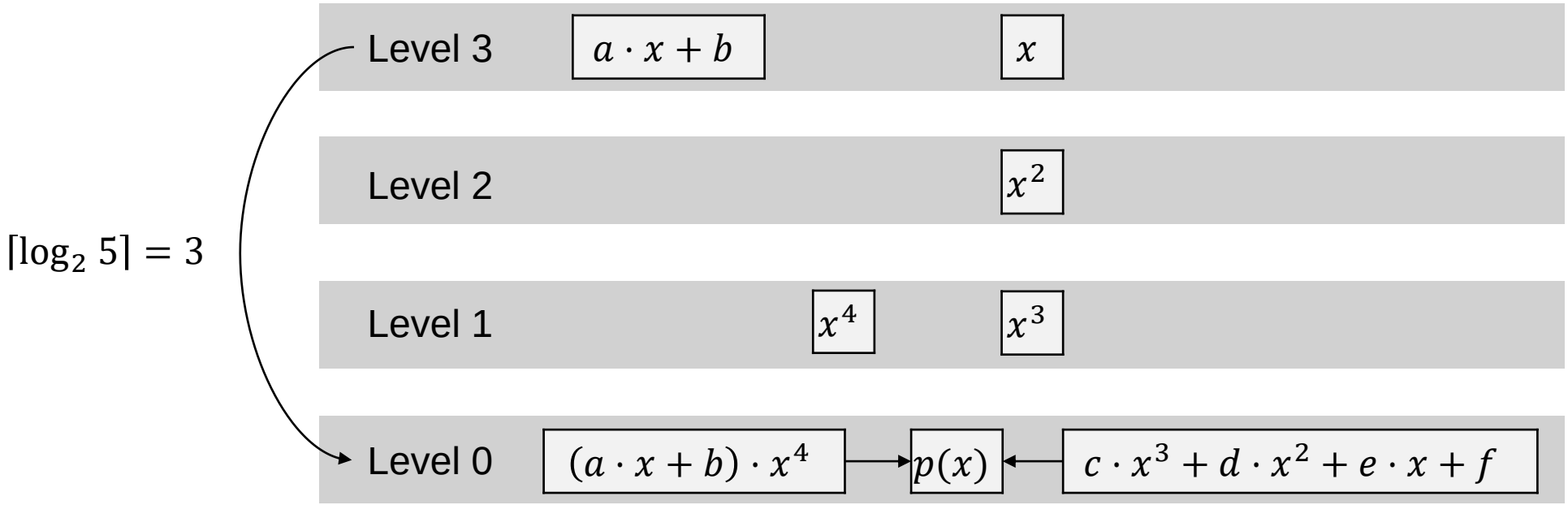
Step 2. Evaluation tree

Polynomial Evaluation

- `EvalPoly` evaluates polynomial using an evaluation tree for performance, numerical stability, lower level consumption (defined in `include/extension/EvalPoly.h`)

e.g.) Evaluate $p(x) = a \cdot x^5 + \mathrm{b} \cdot x^4 + c \cdot x^3 + d \cdot x^2 + e \cdot x + f$
$= (a \cdot x + b) \cdot x^4 + c \cdot x^3 + d \cdot x^2 + e \cdot x + f$

Consumes $\lceil \log_2 \mathrm{D} \rceil$ levels for evaluating a D-degree polynomial



Step 2. Evaluation tree

EvalPoly

- `EvalPoly` is defined in `include/extension/EvalPoly.h`
- $1.7 + 2.1x + 3.1x^3$ → `std::vector<double> coeff = {1.7, 2.1, 0, 3.1};`

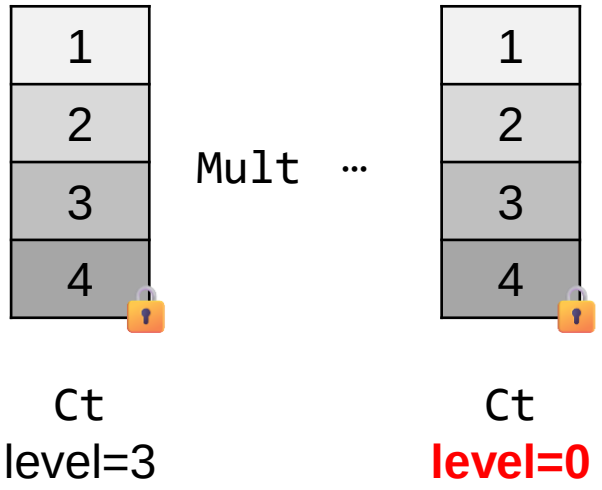
include/extension/EvalPoly.h

```
// Constructor of EvalPoly
EvalPoly(const std::vector<double> &coefficients, int input_level,
         double input_scale, double target_scale, bool chebyshev = false);
…
// Compile evaluation tree
void Compile(ConstContextPtr<word> context);
…
// Evaluator
void Evaluate(ConstContextPtr<word> context, Ct &res, const Ct &input,
              const Evk &mult_key) const;
```

Boot

- `Boot` restores the level, allowing an unlimited number of multiplications

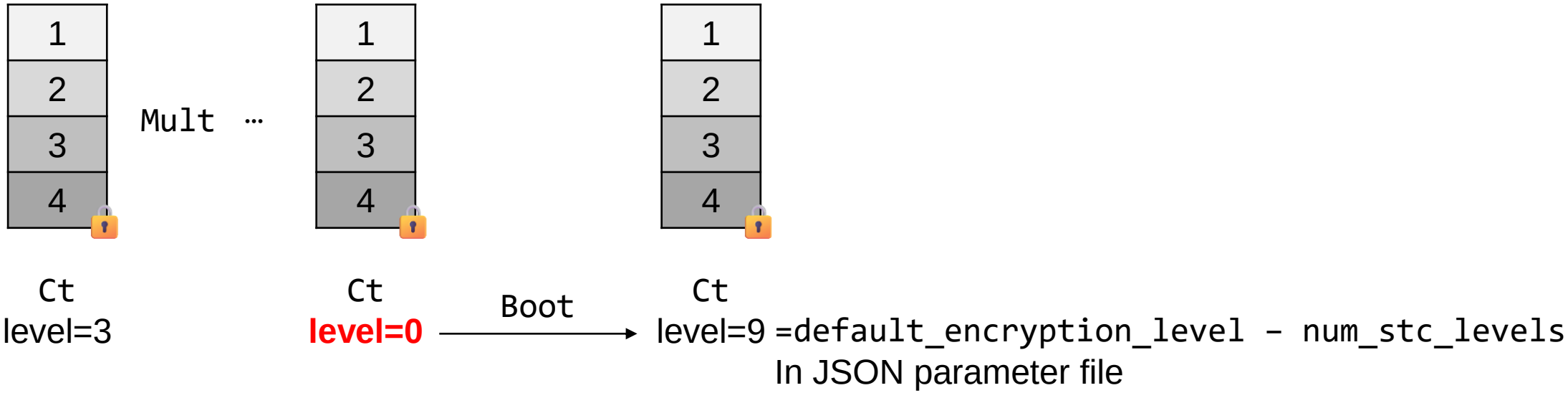
```
void Boot(Ct &res, const Ct &input, const EvkMap<word> &evk_map, bool min_ks = false) const;
```



Boot

- `Boot` restores the level, allowing an unlimited number of multiplications

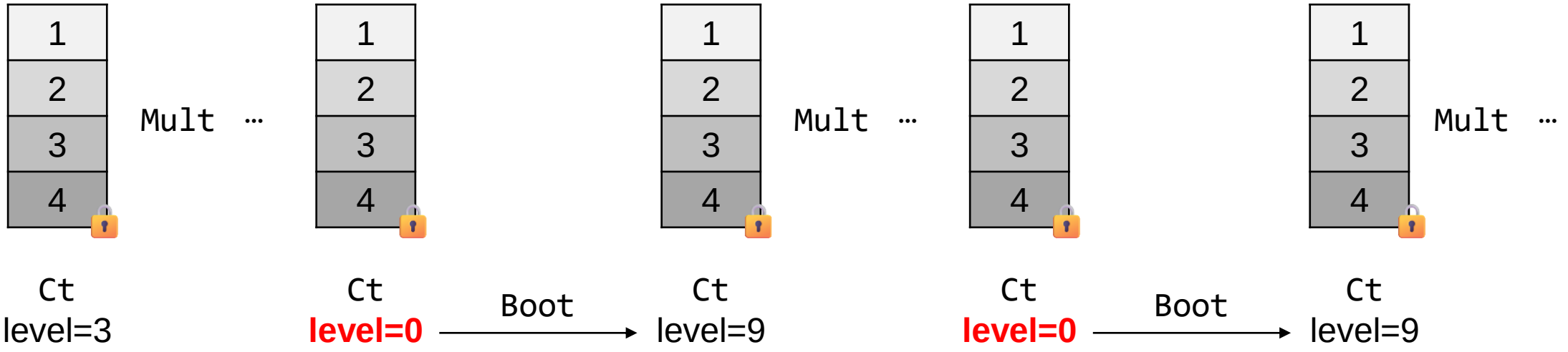
```
void Boot(Ct &res, const Ct &input, const EvkMap<word> &evk_map, bool min_ks = false) const;
```



Boot

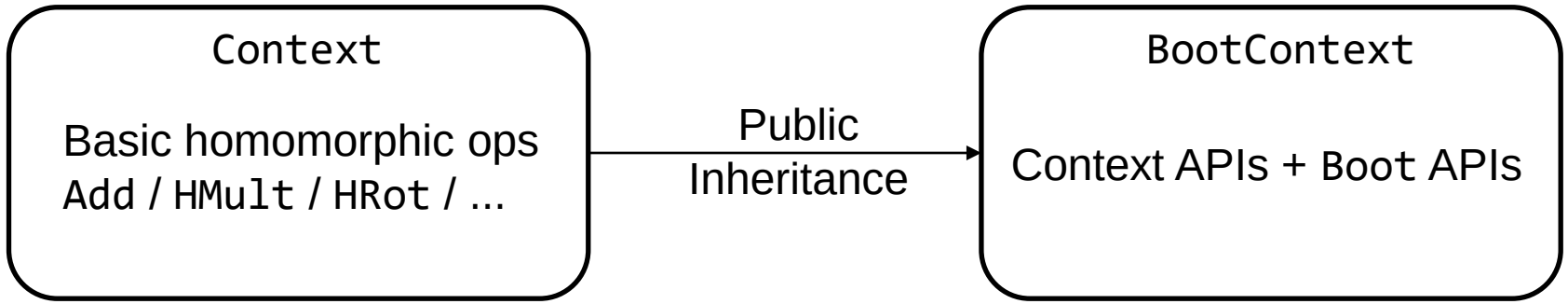
- `Boot` restores the level, allowing an unlimited number of multiplications
- Remaining levels after boot: `default_encryption_level – num_stc_levels`
  e.g.) 12 – 3 = 9 in this parameter

```
void Boot(Ct &res, const Ct &input, const EvkMap<word> &evk_map, bool min_ks = false) const;
```



Boot

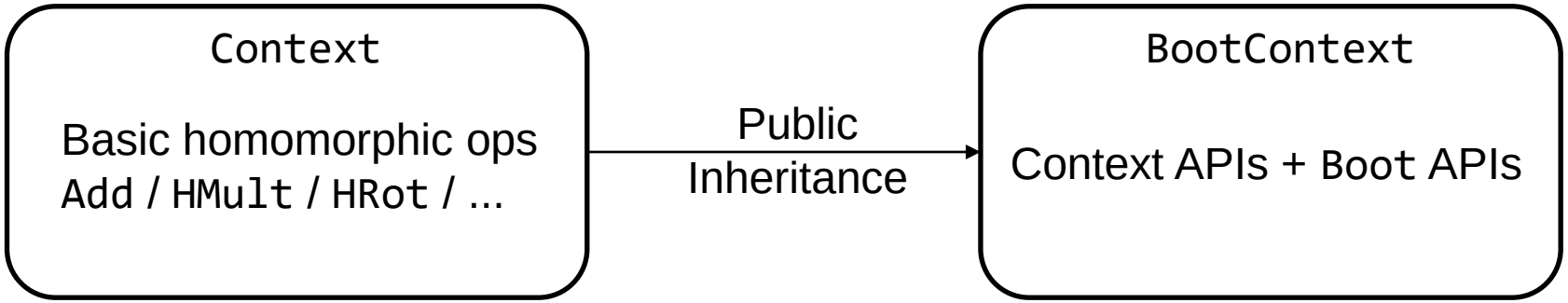
- `BootContext` = `Context` + Bootstrapping support




```
static std::shared_ptr<BootContext<word>> Create(const Parameter<word> &param, const BootParameter &boot_param);
```

Boot

- `BootContext` = `Context` + Bootstrapping support
- `BootContext` needs `BootParam` which contains specific parameters for `Boot`
  - e.g., `num_cts_levels`, `num_stc_levels`, etc.




```
static std::shared_ptr<BootContext<word>> Create(const Parameter<word> &param, const BootParameter &boot_param);
```

include/extension/BootParameter.h

```
BootParameter(int max_level, int num_cts_levels, int num_stc_levels,
              int log_message_ratio = 5);
```

Boot

- `Boot` = `LinearTransform` + `EvalPoly` + …
  - `PrepareEvalMod`: Prepare the polynomial used during `Boot`
  - `PrepareEvalSpecialFFT`: Prepare the linear transform used during `Boot`
- `slots` should be greater than 16

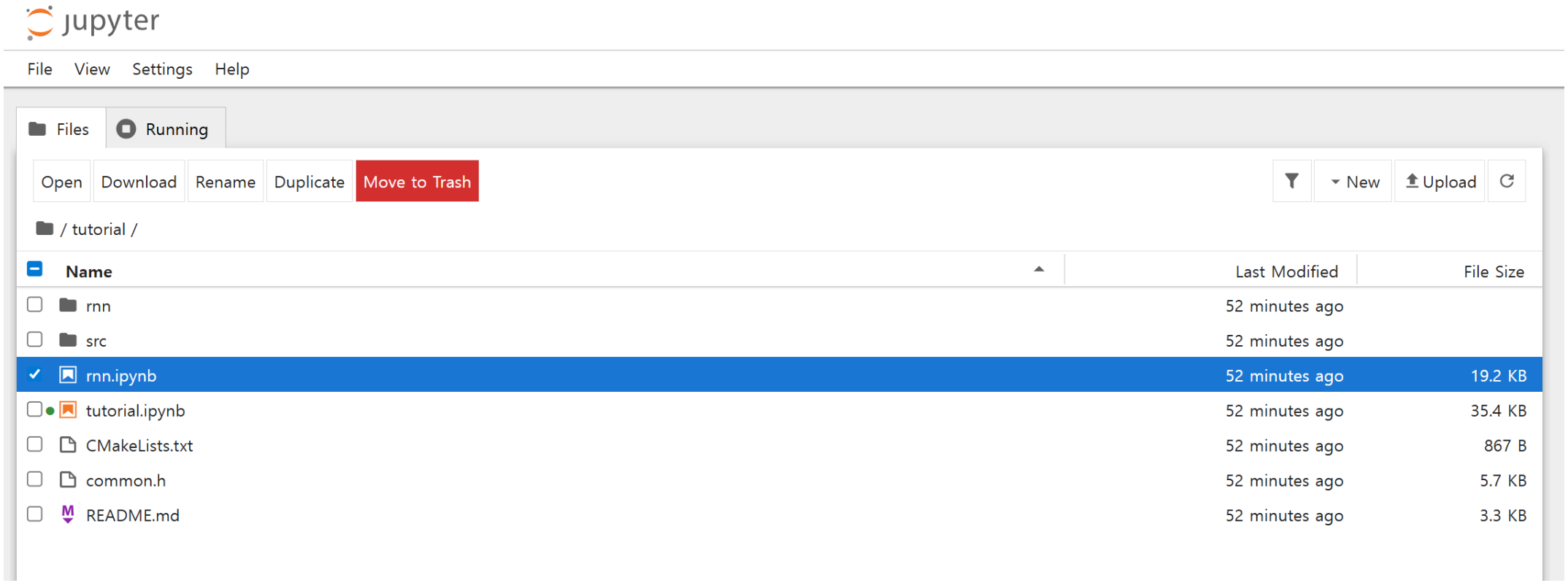
Boot example

```
1  void PrepareBoot(std::shared_ptr<BootContext<word>> boot_context,
2                   UserInterface<word> &ui,
3                   int slots) {
4      // Precompute the two pieces of the bootstrapping circuit.
5      boot_context->PrepareEvalMod();
6      boot_context->PrepareEvalSpecialFFT(slots);
7      // Ask the prepared circuit which rotation keys it needs, then create them.
8      EvkRequest rotations;
9      boot_context->AddRequiredRotations(rotations, slots);
10     ui.PrepareRotationKey(rotations);
11 }
```

FHE-RNN: IMDB Sentiment Classification

Change Jupyter notebook

- Change Jupyter notebook to `/tutorial/rnn.ipynb`



What is IMDB Sentiment Classification?

IMDB is a dataset of movie reviews labeled with binary sentiment labels: **Positive** or **Negative**

- **IMDB Examples**

Positive Review

"The movie was surprisingly entertaining and well acted. The story kept me engaged from start to finish."

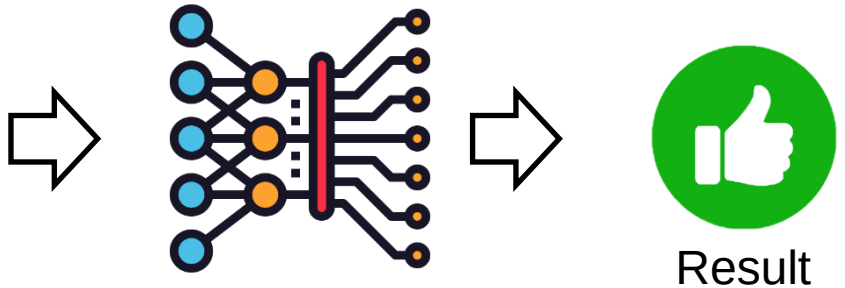
Label: Positive

Negative Review

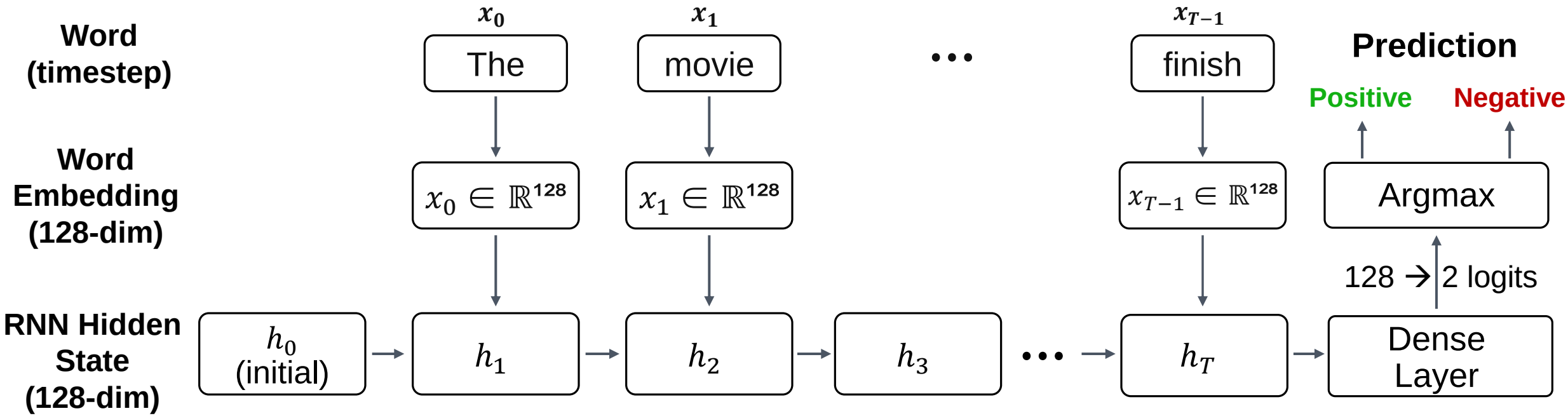
"The plot was boring and predictable. The characters were flat and the ending was disappointing."

Label: Negative

- **Sentiment Classification Example**



RNN Architecture for IMDB Sentiment Classification



RNN Update

$$h_t = f(x_t W + h_{t-1} V + b)$$

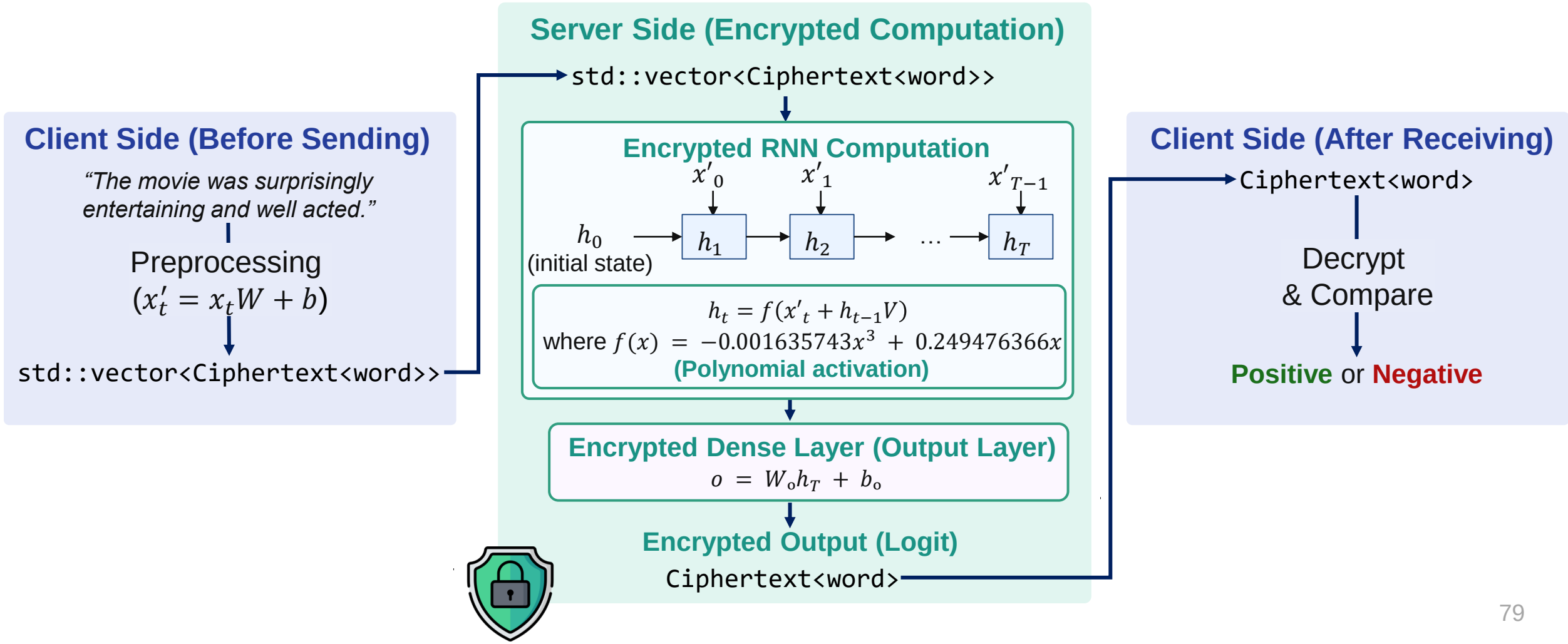
x_t: current word embedding
h_{t-1}: previous hidden state
h_t: current hidden state
f: activation function

- Input: sequence of word embeddings $(x_0, x_1, \ldots, x_{T-1})$
- Each $x_t \in \mathbb{R}^{128}$

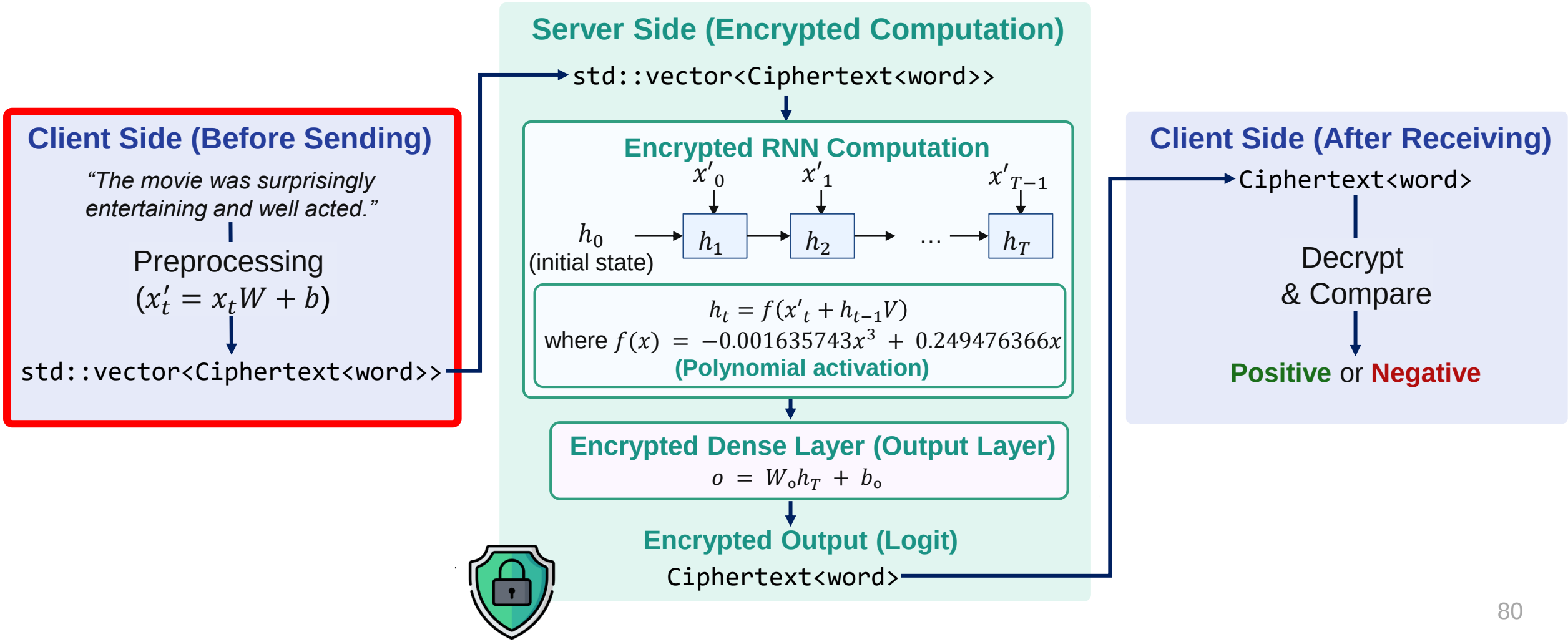
- Hidden state size: 128
- T = 200 (padded/truncated)

- Final hidden state h_T summarizes the entire review
- Used for sentiment classification

Encrypted RNN Inference Scenario



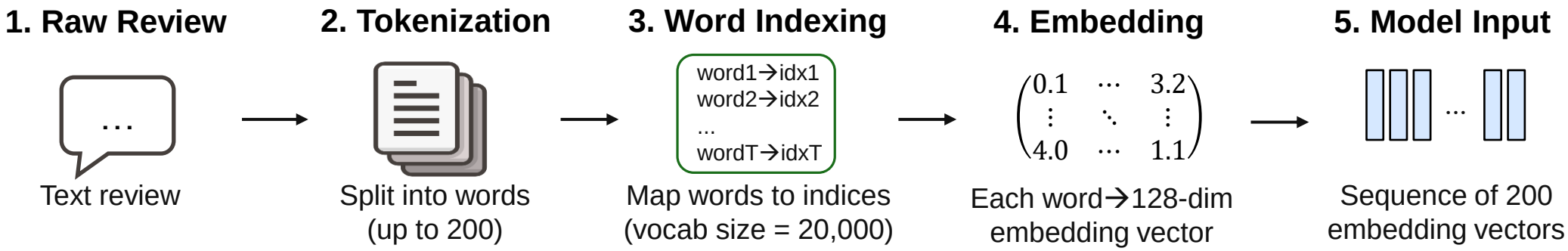
Encrypted RNN Inference Scenario



Input Preprocessing

- The client processes the data (tokenize, embedding, input-to-hidden term)

From Raw Review to Model Input



Example

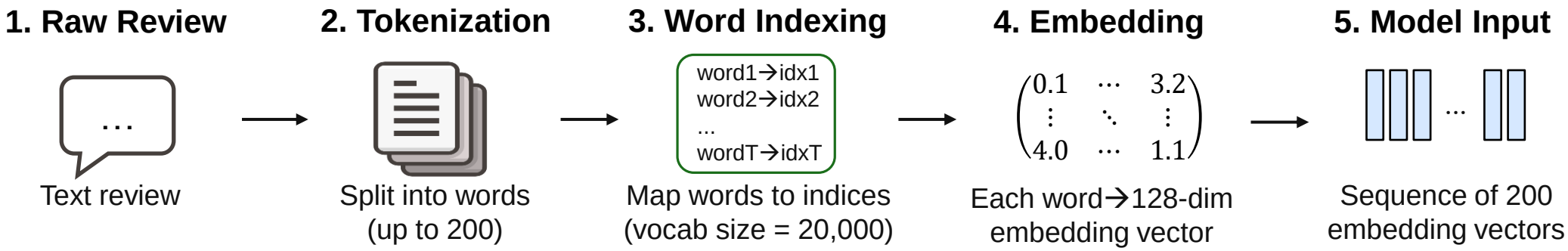
"This movie is good"

```
Input: std::string
```

Input Preprocessing

- The client processes the data (tokenize, embedding, input-to-hidden term)

From Raw Review to Model Input



Example

"This movie is good" → $\begin{bmatrix} 1 \\ 14 \\ 20 \\ 9 \\ 52 \end{bmatrix}$ 5×1

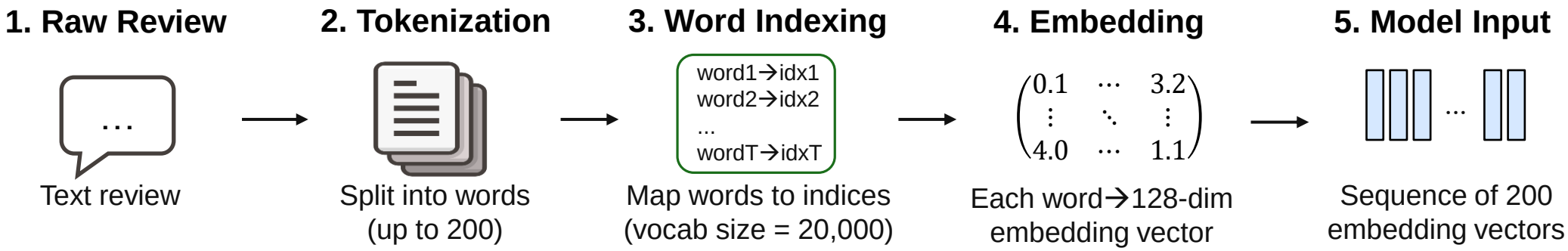
Indexing table
Vocab[START] = 1
Vocab["This"] = 14
Vocab["movie"] = 20
Vocab["is"] = 9
Vocab["good"] = 52

```
Input: std::string
```

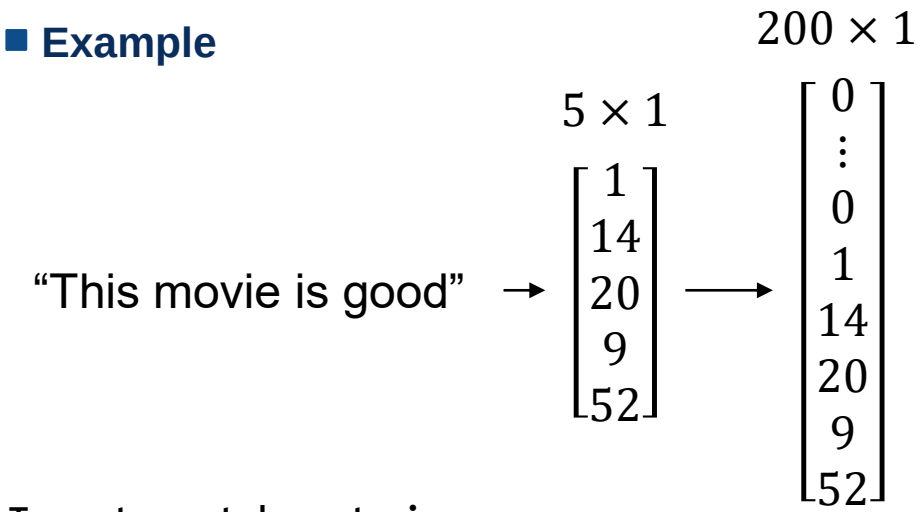
Input Preprocessing

- The client processes the data (tokenize, embedding, input-to-hidden term)

From Raw Review to Model Input



Example

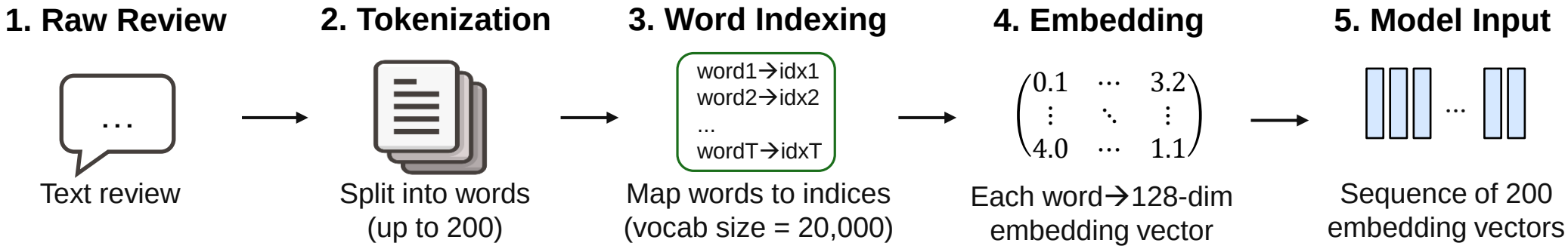


`Input: std::string`

Input Preprocessing

- The client processes the data (tokenize, embedding, input-to-hidden term)

From Raw Review to Model Input



Example

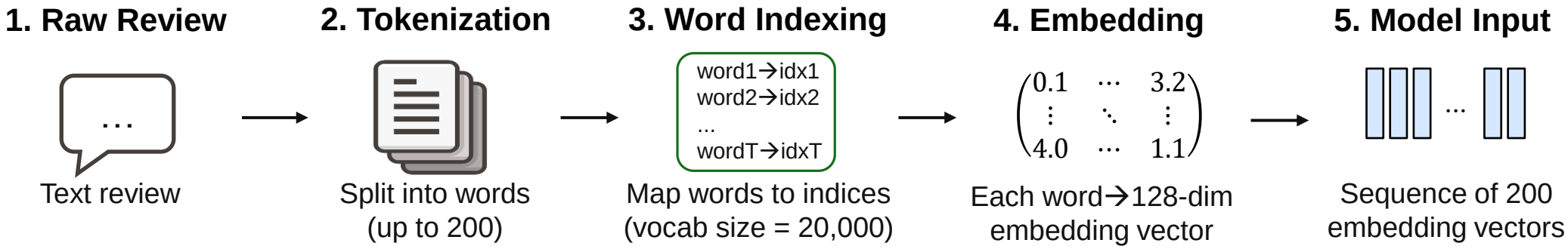
$$\text{“This movie is good”} \rightarrow \underset{5\times 1}{\begin{bmatrix}1\\14\\20\\9\\52\end{bmatrix}} \rightarrow \underset{200\times 1}{\begin{bmatrix}0\\\vdots\\0\\1\\14\\20\\9\\52\end{bmatrix}} \rightarrow \underset{200\times 128}{\begin{bmatrix}1.1 & \cdots & 2.1\\5.1 & \cdots & 3.4\\\vdots & \ddots & \vdots\\0.7 & \cdots & 3.4\end{bmatrix}}$$

`Input: std::string`

Input Preprocessing

- The client processes the data (tokenize, embedding, input-to-hidden term)

From Raw Review to Model Input



Example

$$\text{“This movie is good”} \rightarrow \underset{5\times 1}{\begin{bmatrix}1\\14\\20\\9\\52\end{bmatrix}} \rightarrow \underset{200\times 1}{\begin{bmatrix}0\\\vdots\\0\\1\\14\\20\\9\\52\end{bmatrix}} \rightarrow \underset{200\times 128}{\begin{bmatrix}1.1 & \cdots & 2.1\\5.1 & \cdots & 3.4\\\vdots & \ddots & \vdots\\0.7 & \cdots & 3.4\end{bmatrix}} \xrightarrow{x_t' = x_t W + b} \underset{200\times 128}{\begin{bmatrix}0.1 & \cdots & 0.3\\1.0 & \cdots & 1.2\\\vdots & \ddots & \vdots\\0.2 & \cdots & 5.0\end{bmatrix}}$$

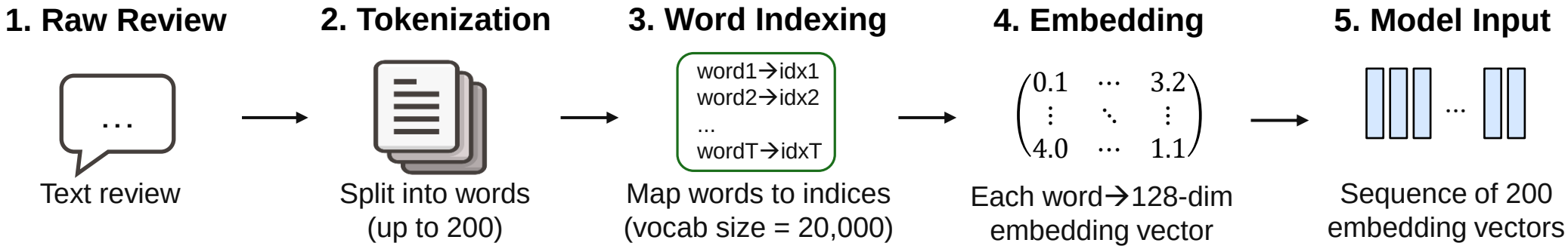
```
Input: std::string
```

$$h_t = f(x_t W + h_{t-1} V + b) = \boldsymbol{f(h_{t-1} V + x_t')}$$

Input Preprocessing

- The client processes the data (tokenize, embedding, input-to-hidden term)

From Raw Review to Model Input



Example

200×1

5×1

"This movie is good" → $\begin{bmatrix} 1 \\ 14 \\ 20 \\ 9 \\ 52 \end{bmatrix}$ → $\begin{bmatrix} 0 \\ \vdots \\ 0 \\ 1 \\ 14 \\ 20 \\ 9 \\ 52 \end{bmatrix}$ → $\begin{bmatrix} 1.1 & \cdots & 2.1 \\ 5.1 & \cdots & 3.4 \\ \vdots & \ddots & \vdots \\ 0.7 & \cdots & 3.4 \end{bmatrix}$ $x_t' = x_t W + b$ → $\begin{bmatrix} 0.1 & \cdots & 0.3 \\ 1.0 & \cdots & 1.2 \\ \vdots & \ddots & \vdots \\ 0.2 & \cdots & 5.0 \end{bmatrix}$ →Ct0 →Ct1 ⋮ →Ct199

200×128 200×128

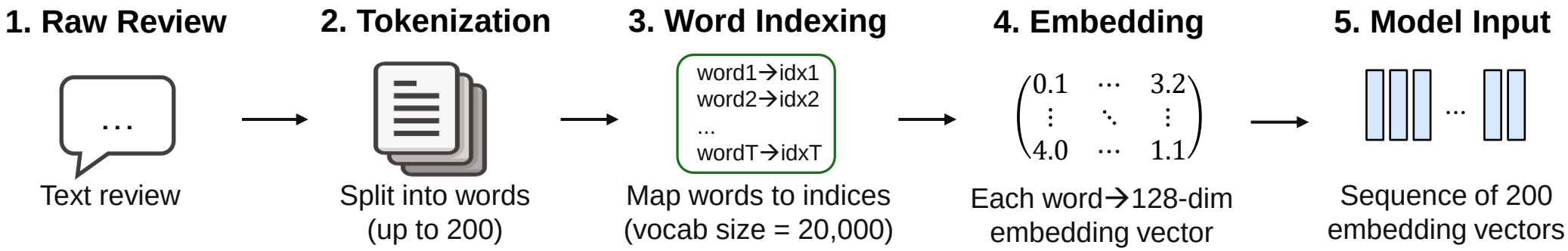
`Input: std::string`

`Output: std::vector<Ct>`

Input Preprocessing

- `PrepareEncryptedInputs(..)` takes an input string and outputs a vector of input ciphertexts

From Raw Review to Model Input



Example

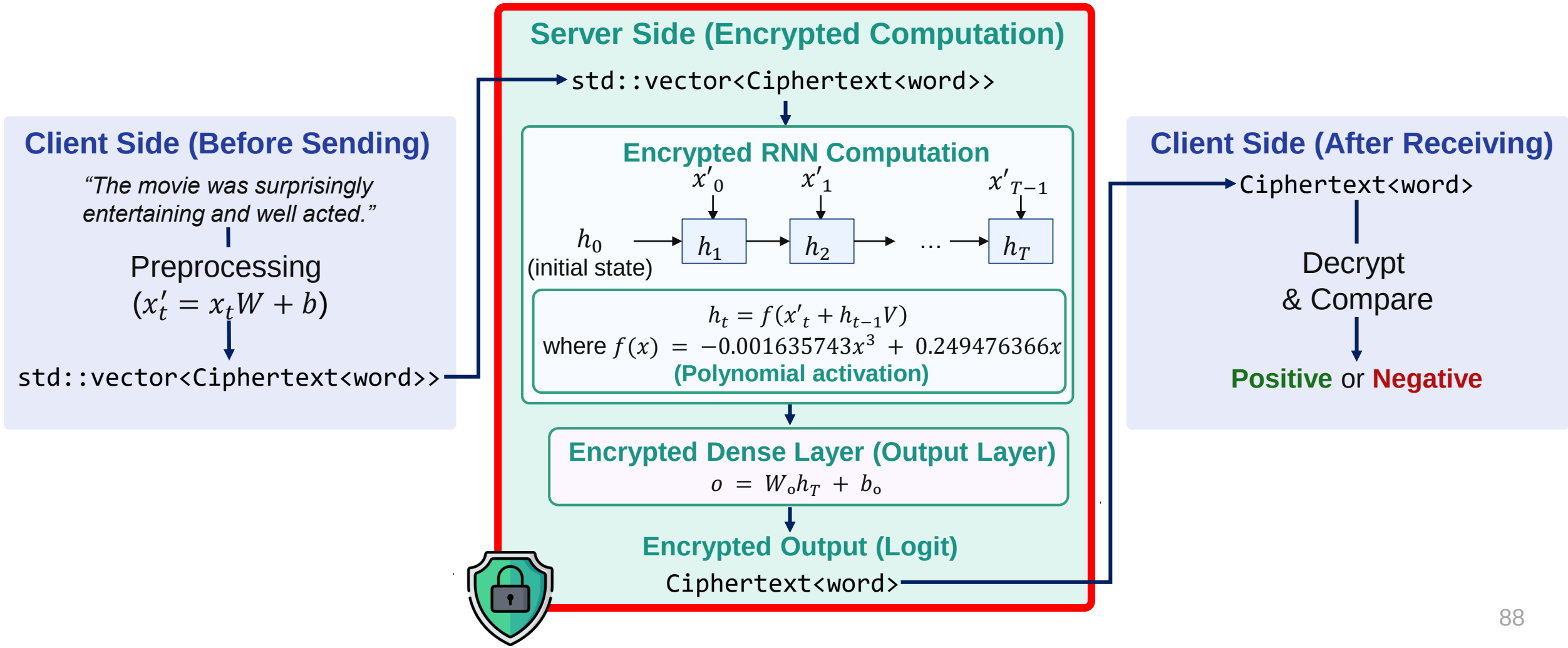
"This movie is good" → `std::vector<Ct> PrepareEncryptedInputs(…)`

```
→Ct0
→Ct1
  ⋮
→Ct199
```

`Input: std::string`

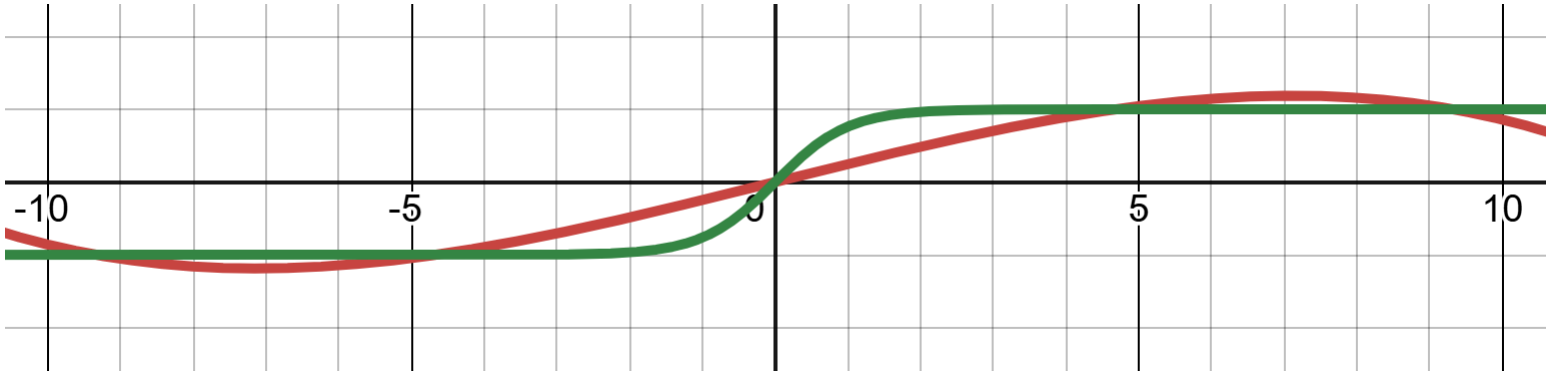
`Output: std::vector<Ct>`

Encrypted RNN Inference Scenario



Polynomial Activation for HE

- RNN update: $h_t = f(x'_t + h_{t-1}V)$
- f = tanh (non-linear)→ low-degree polynomial activation

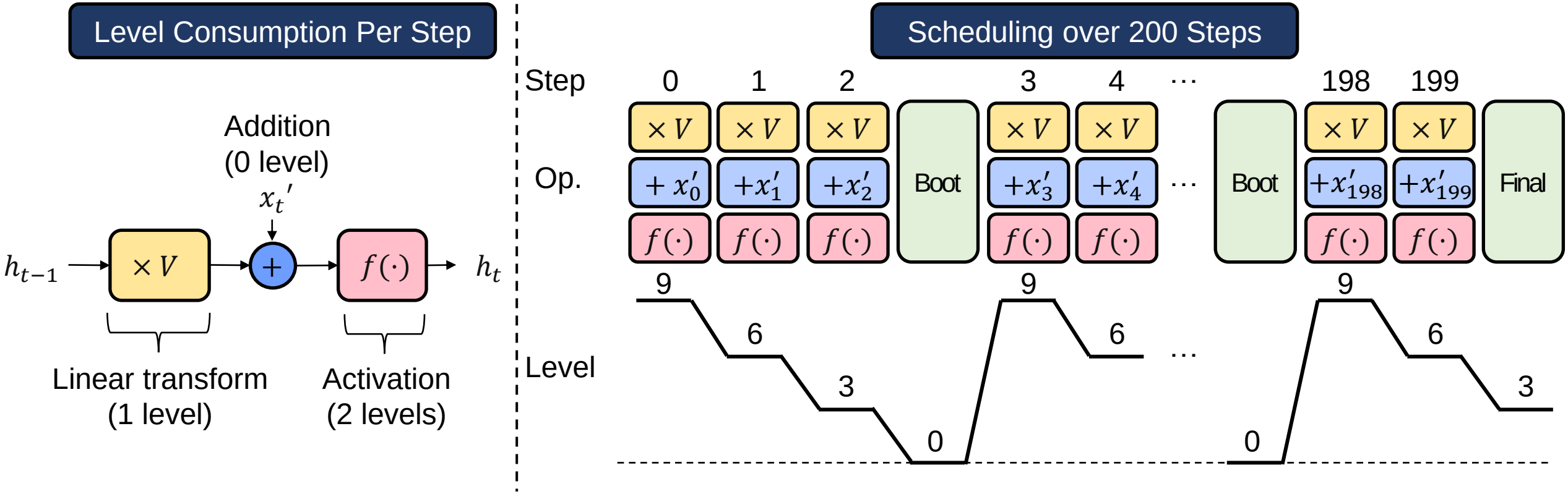


$$y = -0.0016357430300187 48 \cdot x^3 + 0.249476365628036 \cdot x$$

$$y = \tanh(x)$$

Level Scheduling

- RNN update: $h_t = f(x_t' + h_{t-1}V)$ → Consumes 3 levels (linear transform = 1 lv., activation = 2 lv.)
- After `Boot` = 9 level = 3 steps → boot → 3 steps → …
- For the last hidden state, final classification ($h_T W_0 + b_0$) is applied



Model Data Preparation

- We trained a model on IMDB dataset (`tutorial/rnn/rnn_data.bin`)
  - Shows 92% accuracy for both encrypted/unencrypted inference on 128 test samples
- You can test your own review

Examples

```
Custom review
  Text: "The movie was surprisingly entertaining and well acted. The story kept me engaged from start to finish."
  FHE prediction: positive

Custom review
  Text: "The plot was boring and predictable. The characters were flat and the ending was disappointing."
  FHE prediction: negative
```

RNN Inference Logic

- We need to prepare 3 `EvalPoly` instances for poly_tanh (8→6, 5→3, 2→0)

RNN example

```
// Section 3 again: compile the polynomial tanh for each level window.
// Index 0 starts at the highest level. Larger indices handle lower
// levels, matching timestep % kIterationsPerBoot directly.
const std::vector<double> kTanhCoefficients = {
    0.0, 0.249476365628036, 0.0, -0.0016357430301874 8};
std::vector<EvalPoly<word>> activations;
activations.reserve(kIterationsPerBoot);
for (int index = 0; index < kIterationsPerBoot; ++index) {
    const int input_level =
        kWorkingLevel - index * kLevelsPerStep - 1;
    const int output_level = input_level - 2;
    activations.emplace_back(
        kTanhCoefficients, input_level, param->GetScale(input_level),
        param->GetScale(output_level));
    activations.back().Compile(boot_context);
}

```

RNN Inference Logic

- We need to prepare 3 `LinearTransform` instances (9→8, 6→5, 3→2)
- We set (bs, gs) = (8, 16) (bs×gs = 128)

RNN example

```
// Section 4 again: encode W_hidden as a BSGS LinearTransform.
// The matrix is plaintext, while the hidden state h remains encrypted.
int bs = 8;
int gs = 16;
StripedMatrix hidden_striped = ToStripedMatrix(data.hidden_weights);
std::vector<LinearTransform<word>> hidden_transforms;
hidden_transforms.reserve(kIterationsPerBoot);
for (int index = 0; index < kIterationsPerBoot; ++index) {
    const int level = kWorkingLevel - index * kLevelsPerStep;
    hidden_transforms.emplace_back(
        boot_context, hidden_striped, level, param->GetScale(level),
        bs, gs);
}
```

RNN Inference Logic

- **Your task**: Complete the encrypted RNN loop
- For each timestep, bootstrap if needed, apply the hidden-state linear transform, add the encrypted input, and evaluate the polynomial activation

RNN example

```
// Server-side encrypted recurrent loop for one review:
//   h = poly_tanh(W_hidden * h + x_t)
for (int timestep = 0; timestep < kSequenceLength; ++timestep) {
    // TODO: Implement the inference logic of the RNN model.
    // 1. Bootstrap when the level budget is exhausted.
    // 2. Apply the hidden-state linear transform.
    // 3. Add the encrypted input term.
    // 4. Apply the polynomial tanh activation.

}
```

Conclusion

- **From FHE primitives to private ML**
  - Simple APIs for encrypted computation: `Encode/Encrypt/Add/HMult/HRot/Decrypt`
  - Composable building blocks for ML workloads: `LinearTransform/EvalPoly/Boot`
  - End-to-end encrypted sentiment classification
- **Cheddar bridges FHE research and real applications**

FHE & Cheddar: A Tutorial on Fully Homomorphic Encryption

Live coding: Implementing FHE applications with Cheddar

ISCA 2026 Tutorial · Sunday, June 28, 2026

SCALE LAB
Seoul National University

Cheddar open-source repository

`https://github.com/scale-snu/cheddar-fhe`

Thank you!