LLNL-PRES-670952

# Exploration of Lossy Compression for Application-level Checkpoint/Restart


Naoto Sasaki[1], Kento Sato[3],

Toshio Endo[1,2], Satoshi Matsuoka[1,2]

[1] Tokyo institute of technology

[2] Global Scientific Information and Computing Center

[3] Lawrence Livermore National Laboratory

# Needs for Fault Tolerance

**The scale of HPC systems are exponentially growing**

- exa-scale supercomputers in about 2020
- The failure rate increases as systems size grows

Applications' users want to continue its computation even on a failure

**Checkpoint/Restart technique is widely used as fault tolerant function**

- But this technique has problems

# Needs for Reduction in Checkpoint Time

## Checkpoint/Restart

→Data of memory is stored in the disk

→High I/O cost

**On TSUBAME2.5**
Memory capacity : about 116TB
I/O throughput : about 8GB/s
↓
Checkpoint time : about 4 hours

## MTBF(Mean Time Between Failure) is reduced by expansion in the scale of HPC systems

- MTBF is projected to shrink to over 30min in 2020 [※1]

If MTBF < Checkpoint time
an application may not be able to run !
↓
Needs for reduction in checkpoint time !

**Applications' users need to reduce checkpoint time**

※1 : Peter Kogge, Editor & Study Lead (2008)
ExaScale Computing Study: Technology Challenges in Achieving ExaScale Systems

# To Reduce Checkpoint Time

**There are techniques to reduce checkpoint size**

- Compression
- Incremental checkpointing
    - This stores only differences with the last checkpoint

**Compression can be combined with incremental checkpointing**

- In addition, the effect of incremental checkpointing may be limited in scientific applications

**We focus on compression for checkpoint image data**

# Lossless and Lossy Compression

gzip, bzip2, etc.

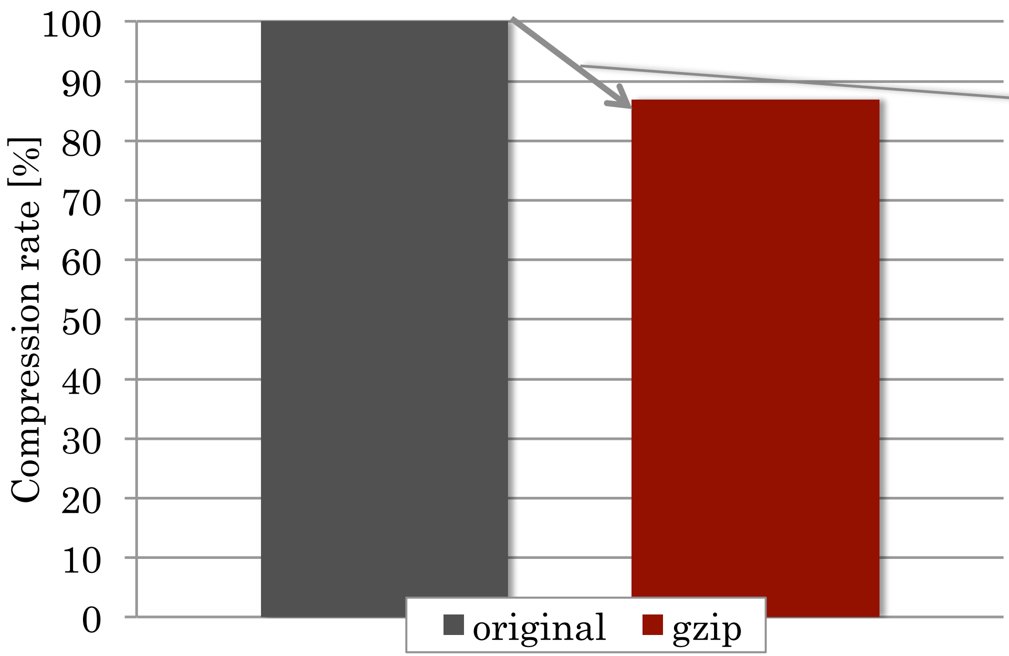
**Features of lossless**

- No data loss
- Low compression rate without bias
    - Scientific data has a randomness

jpeg, mp4, etc.

**Features of lossy**

- High compression rate
- Errors are introduced



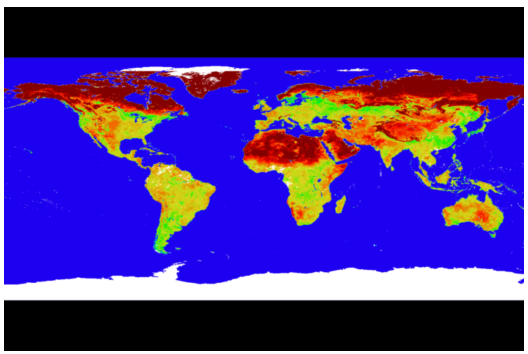

If we apply lossless compression to floating point arrays, the compression rate is limited

We focus on lossy compression

# Discussion on Errors Introduced by Lossy Methods

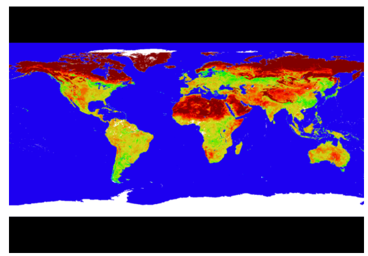
**Errors may be acceptable if we examine processes for developing real scientific applications**

- Scientific model and sensors also introduce errors
- We need to investigate whether the errors are acceptable



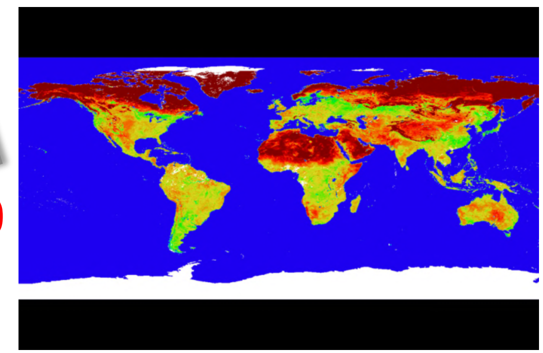
gzip 2.19MB

1/7



original 14.7MB

1/100



jpeg2000 0.153MB

(citation of images : http://svs.gsfc.nasa.gov/vis/a000000/a002400/a002478/)

**Don't apply lossy compression to data that must not have an error (e.g. pointer)**

**We apply lossy compression to checkpoint data**

- The calculation continues with data including errors

# Outline of Our Study

## Purpose

- To reduce checkpoint time, lossy compression is applied to checkpoint data then checkpoint size is reduced

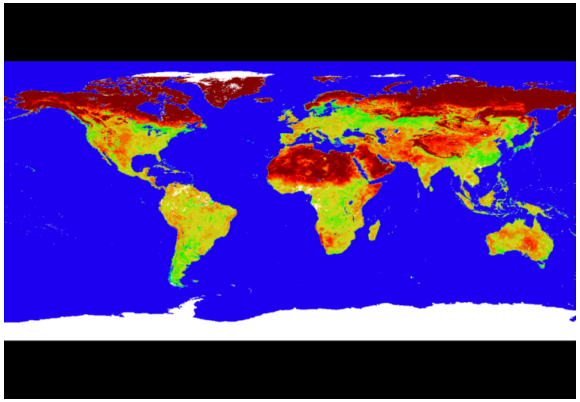
## Proposed Approach

1. We apply wavelet transformation, quantization and encoding to a target data, then store the data in a recoverable format
2. We apply *gzip* to the recoverable format data

## Contribution

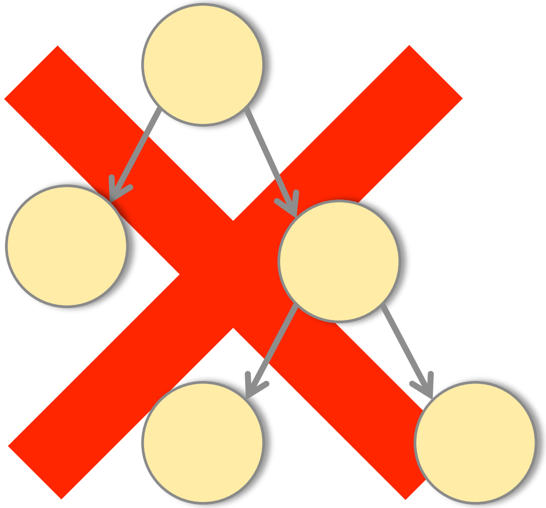
- We apply our approach to real climate application, NICAM, then overall checkpoint time included compression time is reduced by 81% with 1.2% relative error on average in particular situation

# Assumption for Our Approach



**We assume application−level checkpoint**

- We utilize that difference between neighbor values
- Target data are an arrays of physical quantities
  - We target 1,2 or 3D mesh data represented by floating point arrays



**There are data to which must not be applied our approach because errors are introduced**

- Data structure including pointers (e.g. tree)

Users specify a range of data to which are applied our approach

# Motivation of Wavelet Transformation

**Lossless compression is effective in data that have redundancy**

- Scientific data has a randomness
- We need to make redundancy in the scientific data

**To make much redundancy and make errors small…**

- The target data should have dense and small values

**The scientific data does not spatially changed much**

To make good use of this feature…

**We focus on wavelet transformation**

# About Wavelet Transformation

**Wavelet transformation is a technique of frequency analysis**

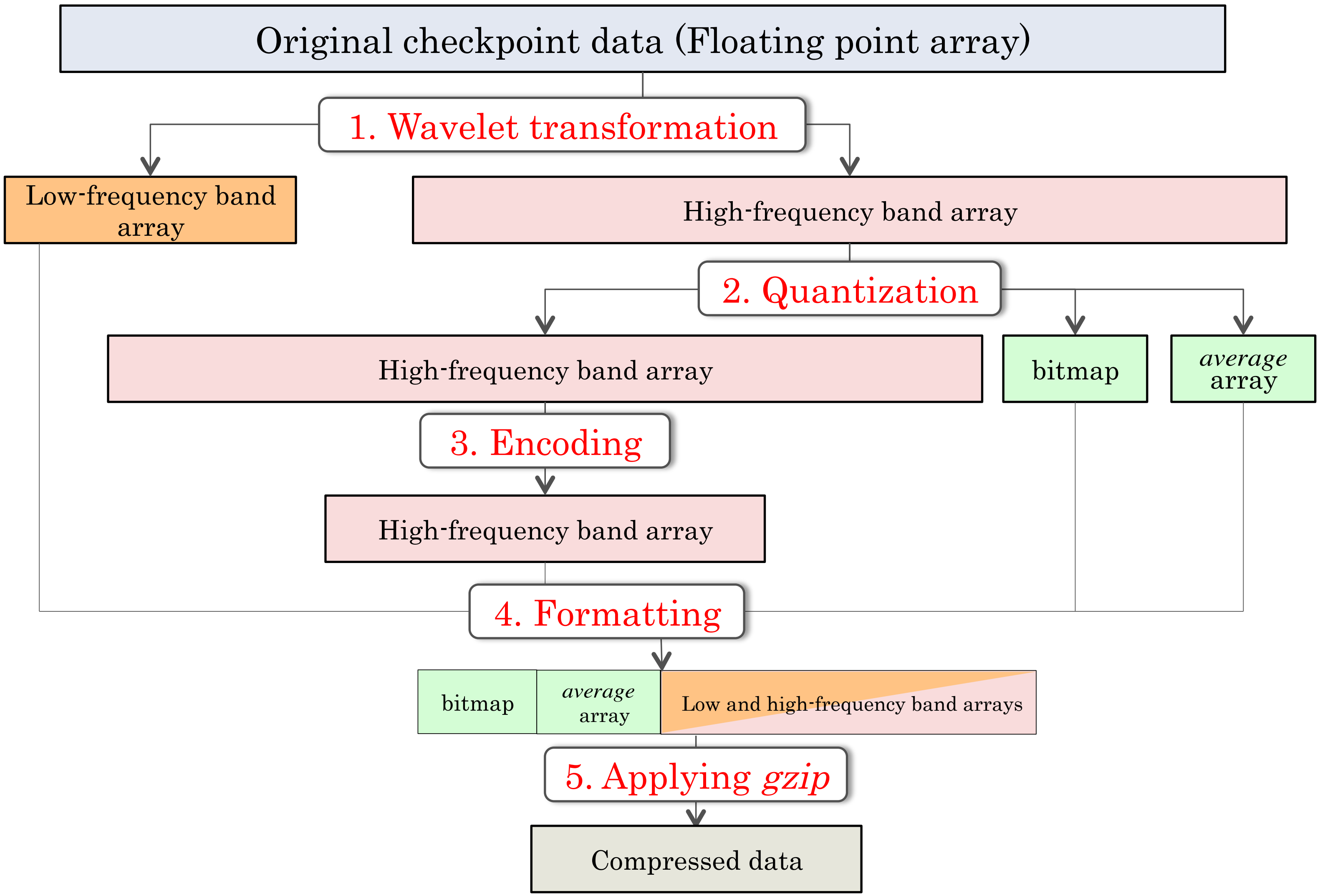
We suspect that compression that uses wavelet transformation is efficient in applications that uses physical quantities (e.g. pressure, temperature)

**Multiple resolution analysis is effective in compression**

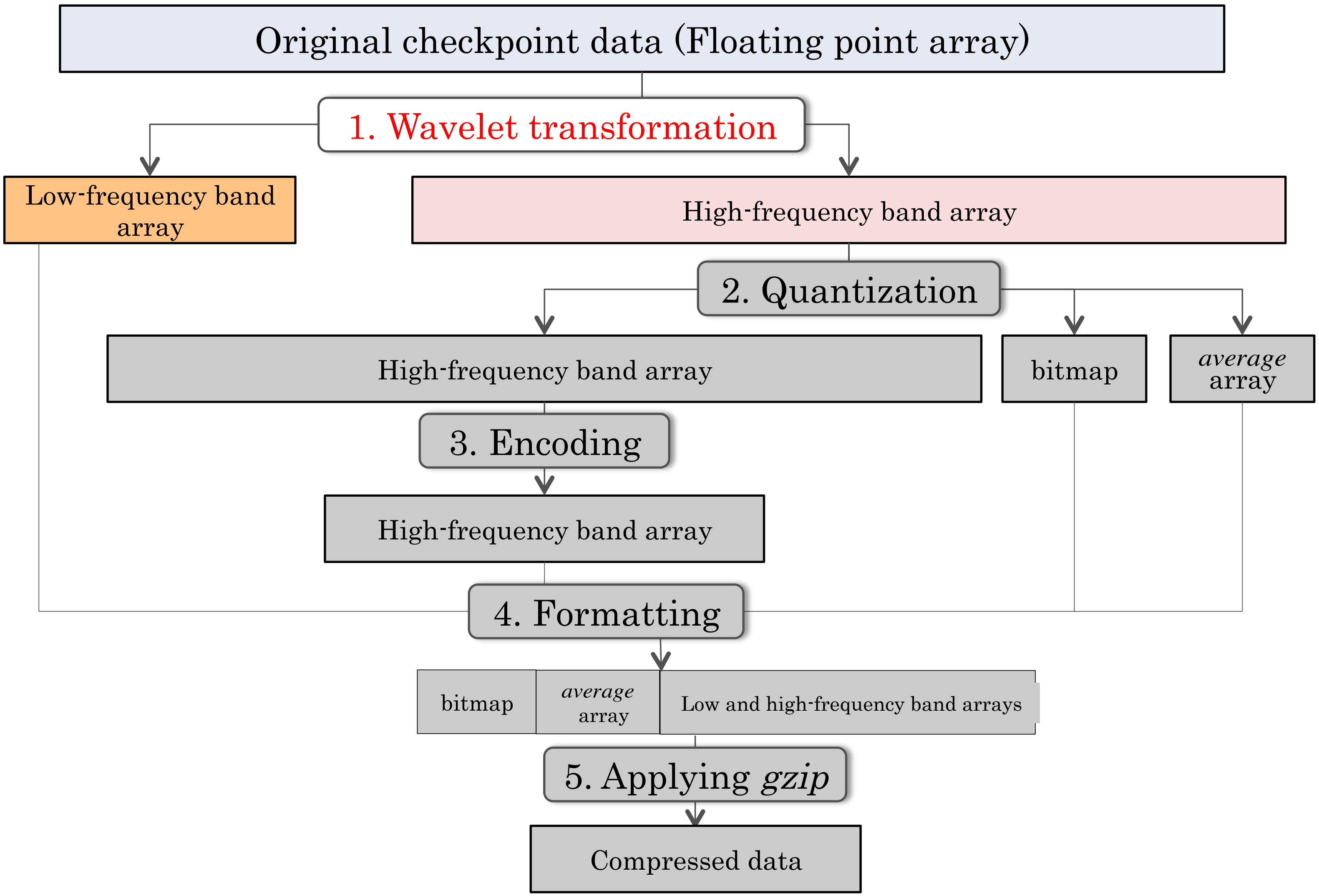
- JPEG2000 uses this technique
- It is known that this technique is effective in *smooth* data
  - This "*smooth*" means the difference between neighbor values is small

※Wavelet transformation itself is NOT compression method, but we use it for preprocessing

# Proposal Approach: Lossy Compression Based On Wavelet



# Wavelet Transformation



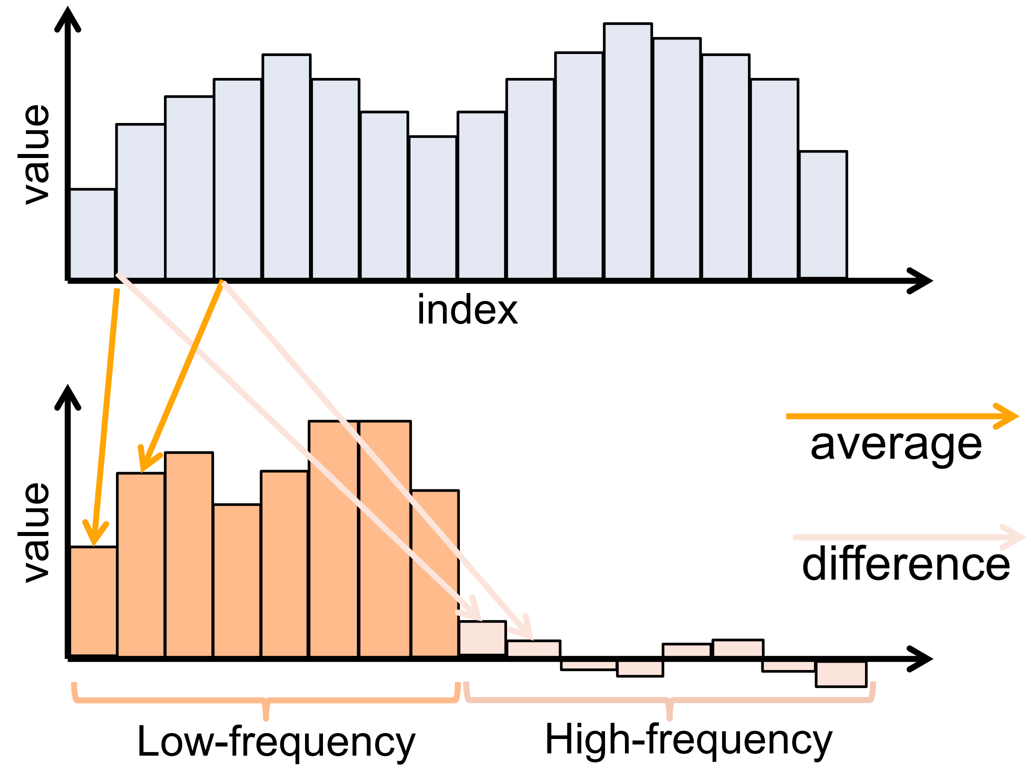
# 1D Wavelet Transformation in Our Approach

**We use average of two neighbor values and difference between two neighbor values**

Original 1D array

Wavelet transformation

Transformed array




**In high-frequency band, most of values are close to zero**

→We expect that an introduced error is small even if the precision of values in high-frequency band region is dropped

# Multi-dimensional Wavelet Transformation

**In multi-dimensional array, we apply 1D wavelet transformation to each dimension**

**In case of 2D array**

- # of low…1
- # of high…3

**In case of 3D array**

- # of low…1
- # of high…7

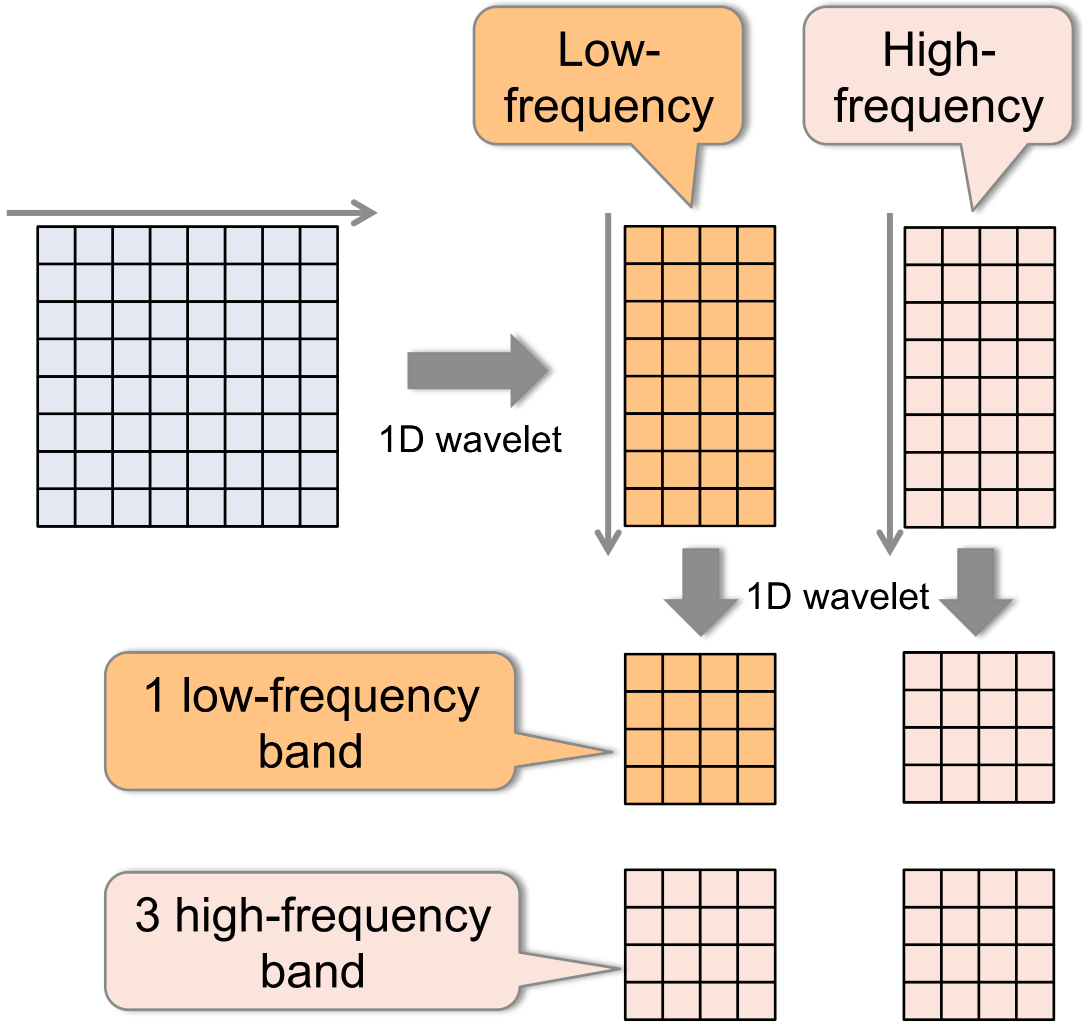
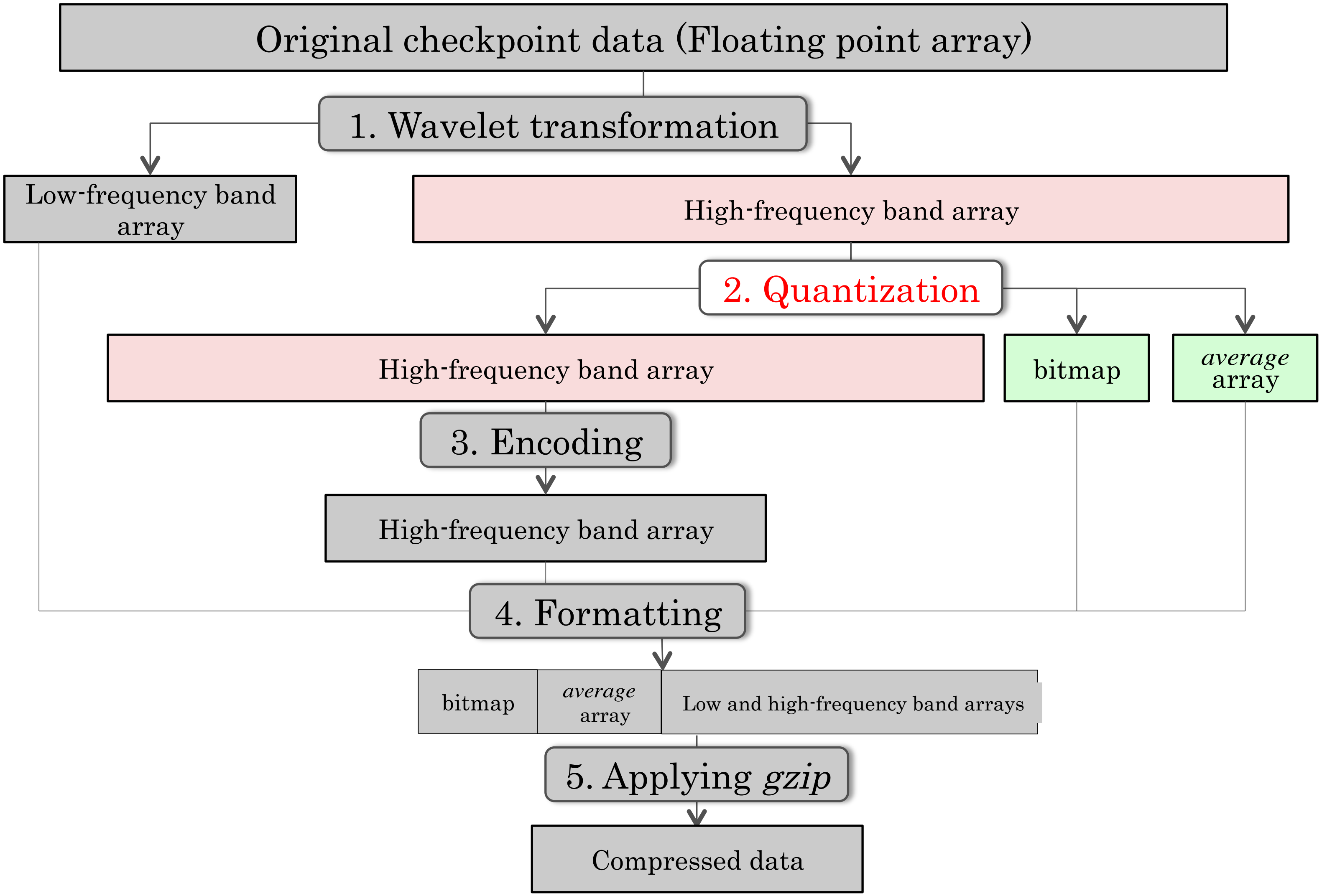


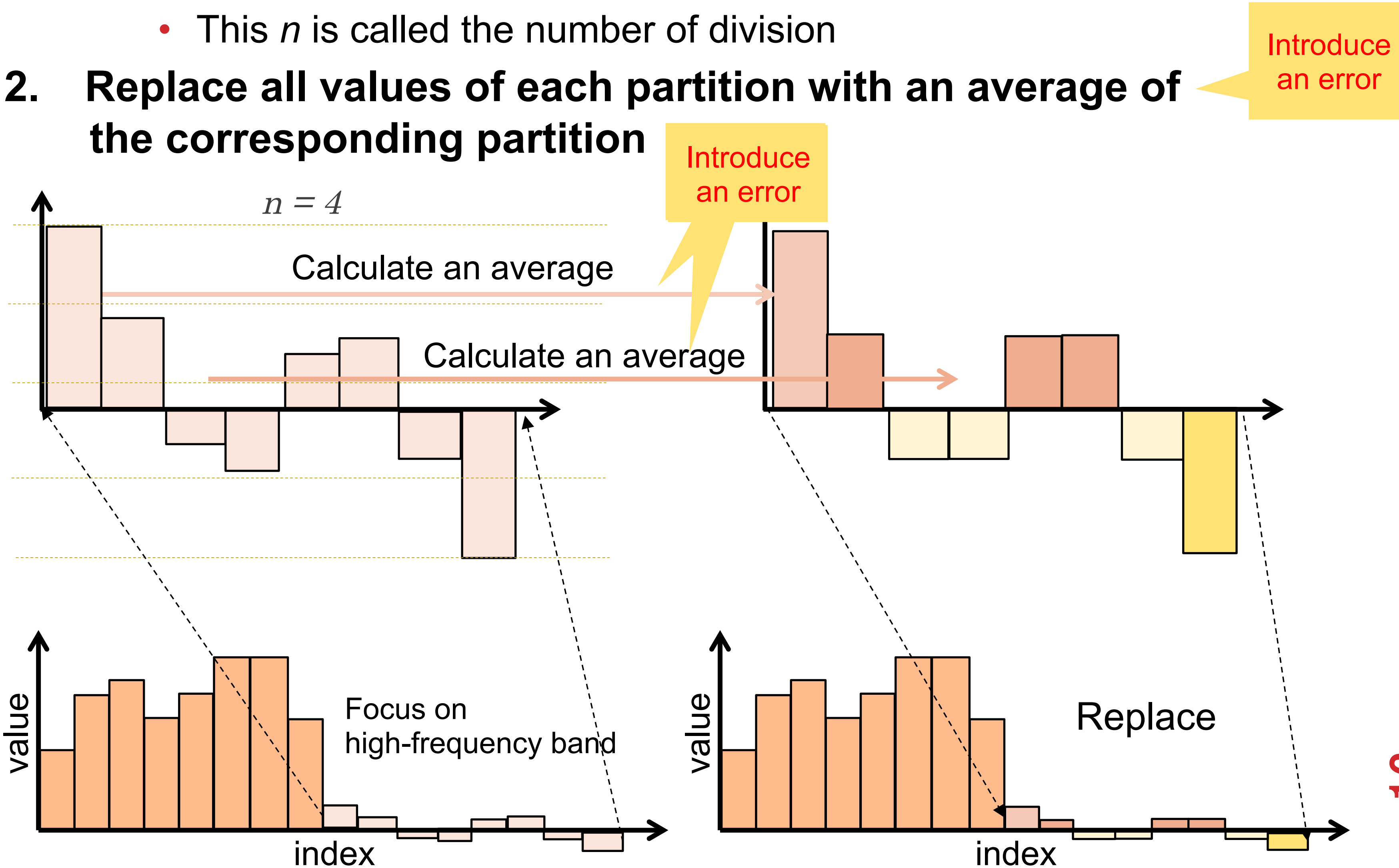

Fig : an example of wavelet transformation for a 2D array

# Quantization



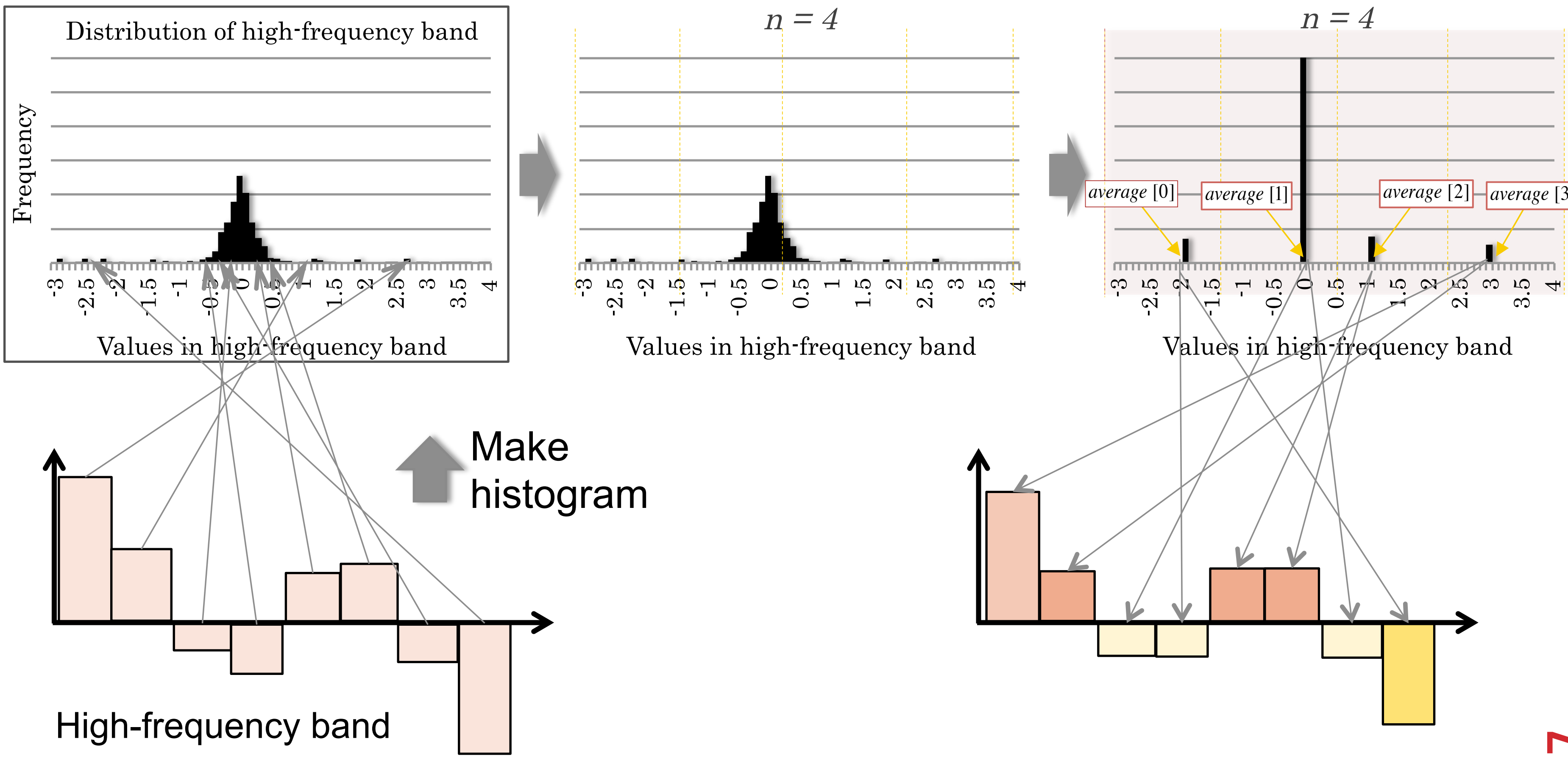
# Simple Quantization

1. **Divide high-frequency band values into *n* partitions**
   - This *n* is called the number of division
2. **Replace all values of each partition with an average of the corresponding partition**



# Problems of Simple Quantization

## Simple quantization introduces large errors



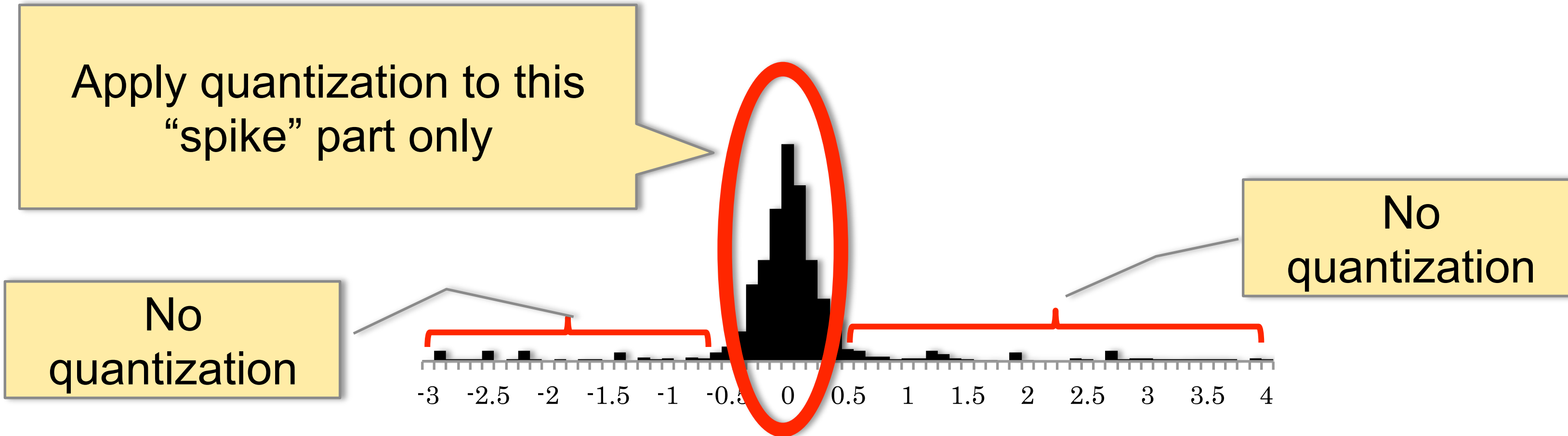
# To reduce Errors

## Target data is expected to be smooth

- Most of values in high-frequency band are close to zero
  - These make a "*spike*" in the distribution

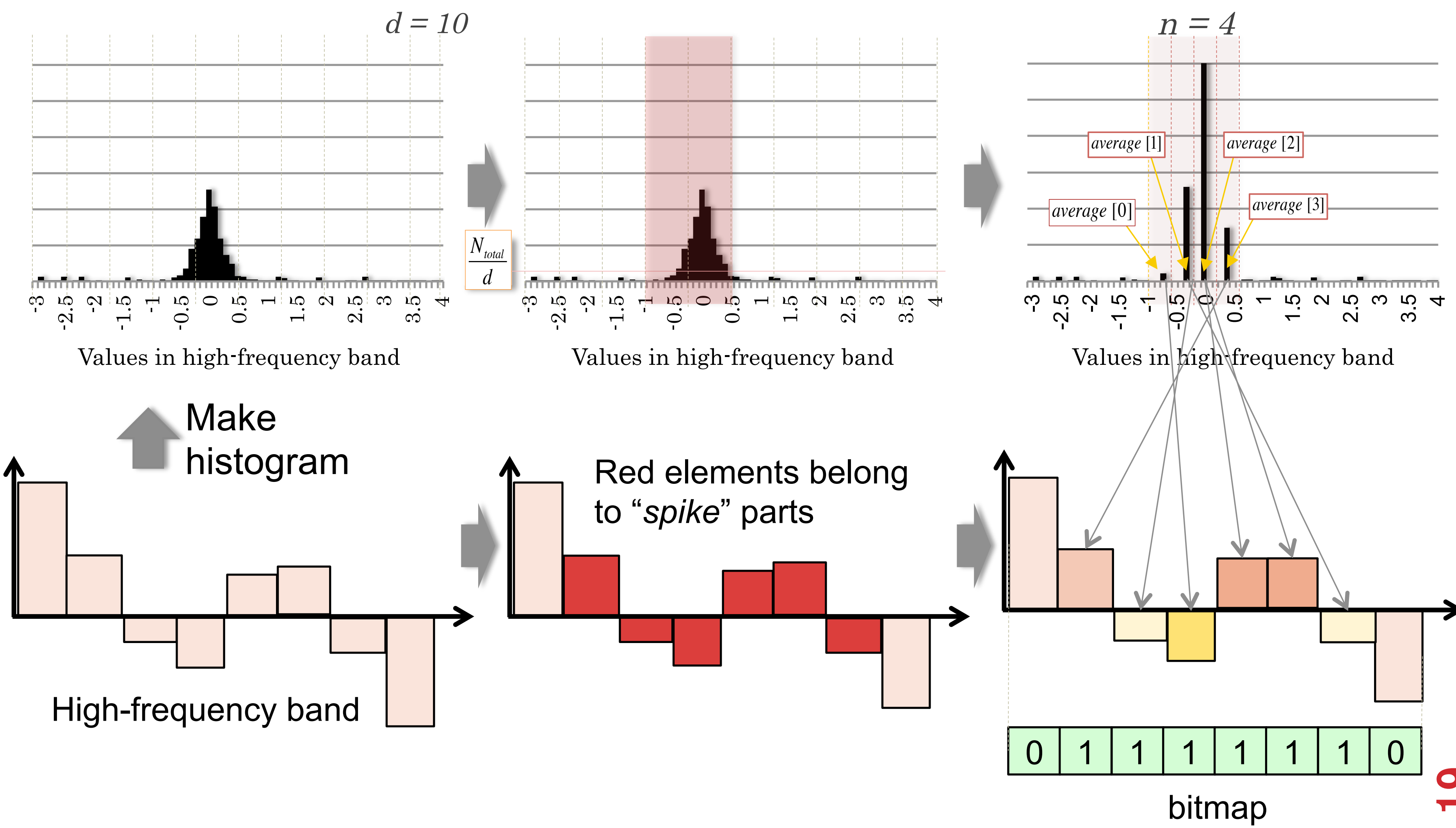
## To reduce an error, we apply the quantization to the "*spike*" parts only

- An impact on compression rate is low because the spike parts consist of most of values in high-frequency band



# Proposed Quantization

## This method is improved version of simple one



# Difference in quantization methods

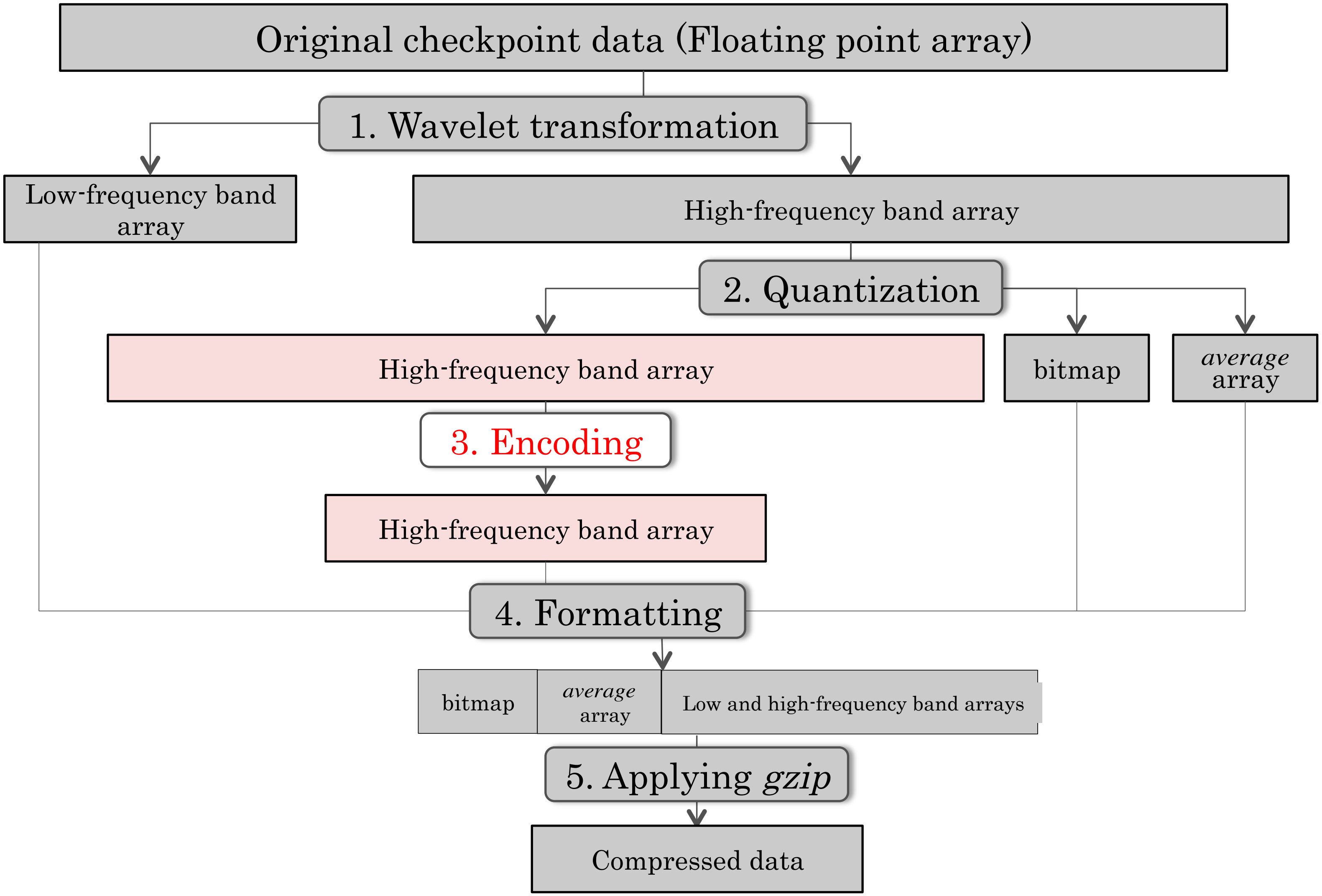
## Simple quantization

- Replace all values in high-frequency band
  - →Introduce large errors
  - →High compression rate because of less type of values

## Proposed quantization

- Replace parts of values in high-frequency band
  - →Introduce small errors
  - →Low compression rate by lack of regularity

# Encoding

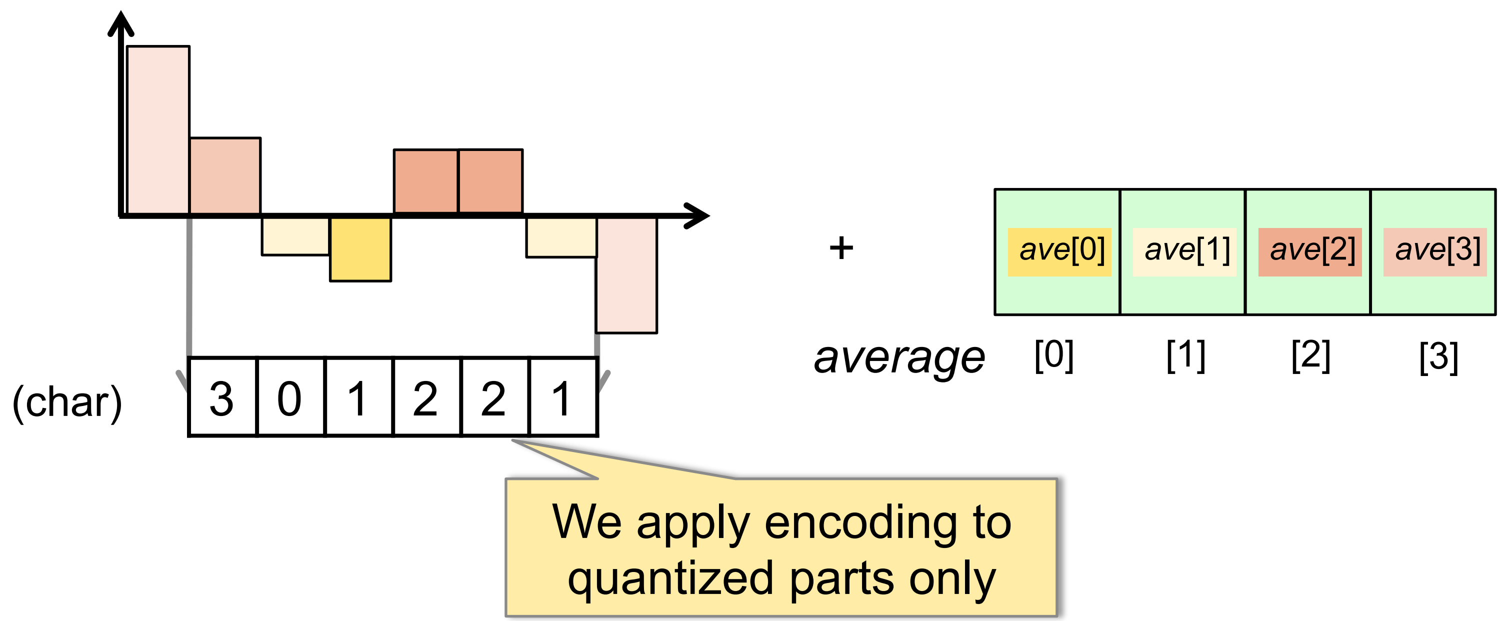

Original checkpoint data (Floating point array)
1. Wavelet transformation
Low-frequency band array
High-frequency band array
2. Quantization
High-frequency band array
bitmap
*average* array
3. Encoding
High-frequency band array
4. Formatting
bitmap
*average* array
Low and high-frequency band arrays
5. Applying *gzip*
Compressed data

# Encoding

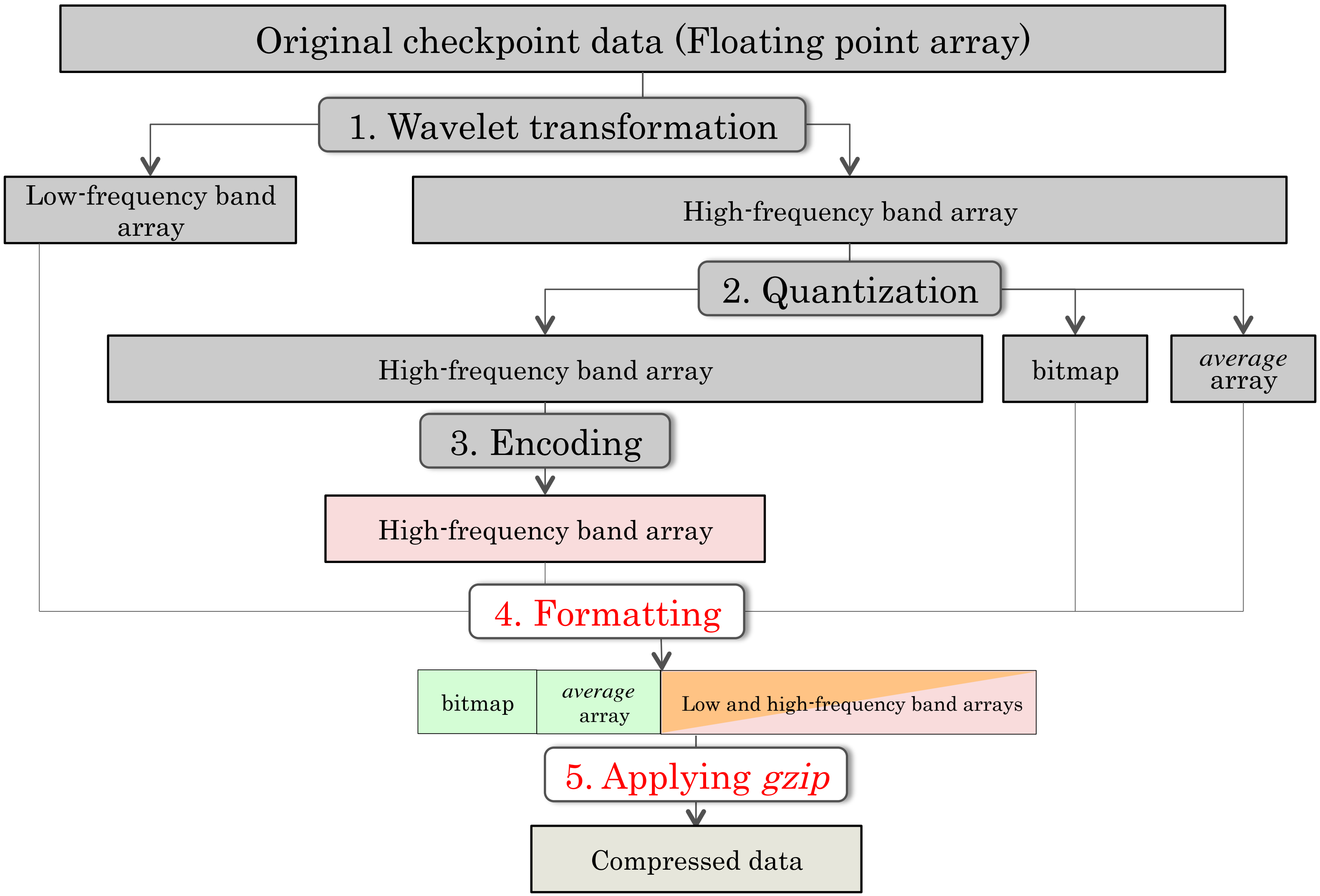
## In quantization step, all or part of high-frequency band are replaced with *n* kinds of values

- *n* kinds of *double* values are replaced with corresponding *char* values
  - In case of *double,* data size becomes 1/8
  - In case of *float,* data size becomes 1/4

## In recovery, an *average* array is required



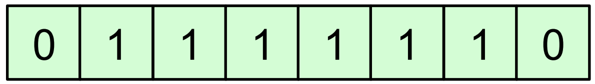
# Formatting


Original checkpoint data (Floating point array)
1. Wavelet transformation
Low-frequency band array
High-frequency band array
2. Quantization
High-frequency band array
bitmap
*average* array
3. Encoding
High-frequency band array
4. Formatting
bitmap
*average* array
Low and high-frequency band arrays
5. Applying *gzip*
Compressed data

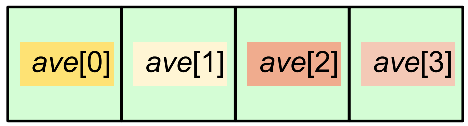
# Recoverable Format

## Required data in restart

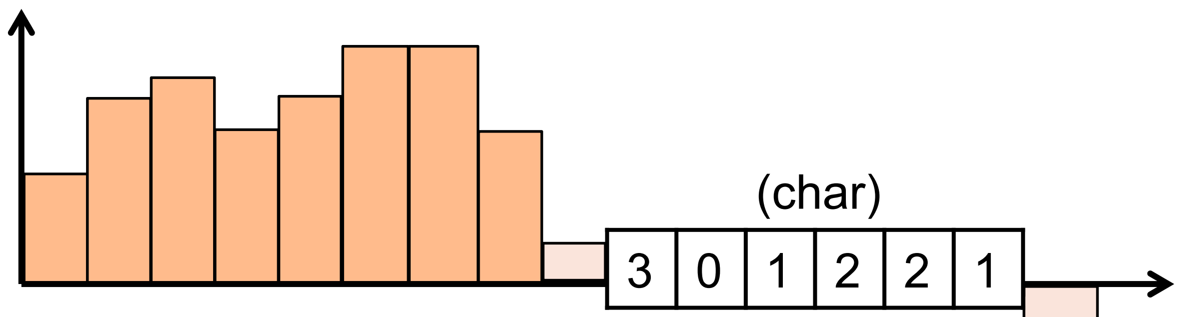
- Bitmap



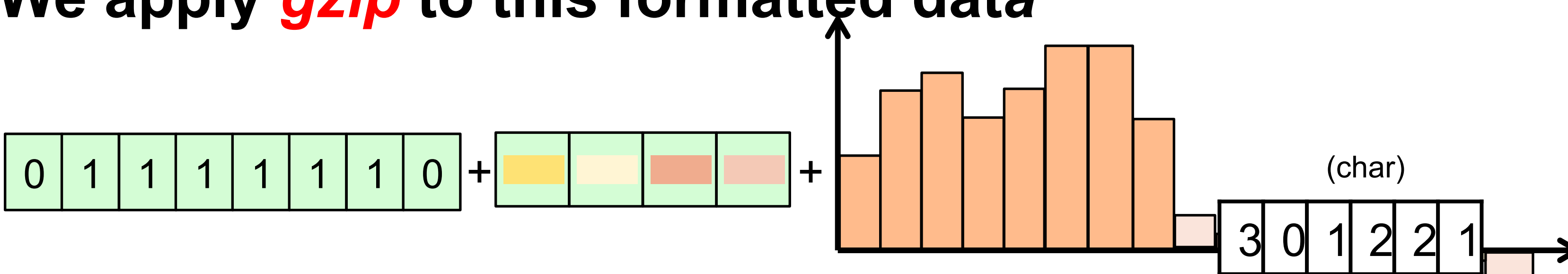

- Average array




- *Char* and *double* data to which is applied our approach

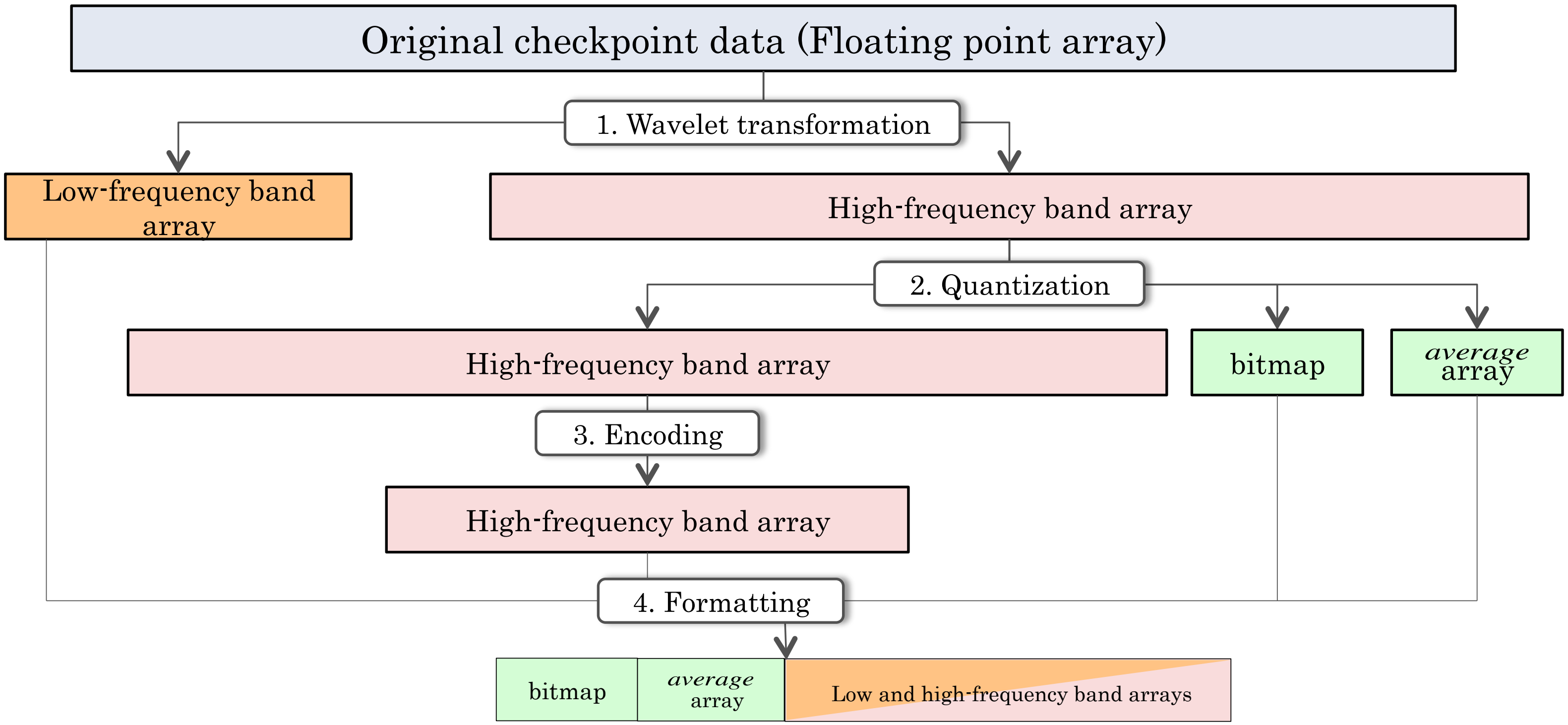



## We apply *gzip* to this formatted data



# Computational Complexity

## Our compression algorithm contains only single loop that processes all or part of arrays

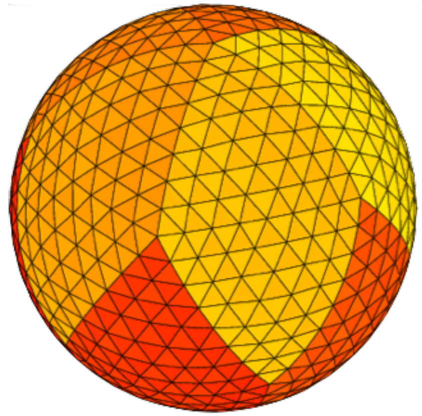



An algorithm of our approach has computational complexity *O(s)* with checkpoint size *s*

# Evaluation Environment

**To estimate a impact of our approach, we evaluate…**

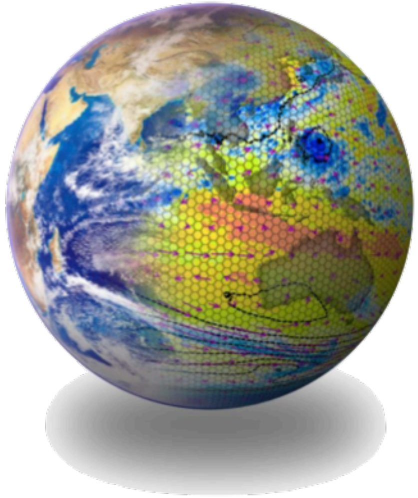
- Compression time
- Compression rate
- The degree of errors



**Our approach is applied to real climate simulation, NICAM[M.Satoh, 2008]**

- Target physical quantities are pressure, temperature and velocity.
    - double precision, 3Darray, 1156*82*2
- The data is too smooth in initial state

→apply our approach after 720 steps from initial state



Machine spec

| | |
|---|---|
| CPU | Intel Core i7-3930K 6 cores 3.20GHz |
| Memory size | 16GB |

(citation of image : HPCS2014 全球大気シミュレーションにおいて、どこまで解像度が必要か？)

# Metrics for Evaluation

## Compression rate

$$CR = \frac{CS_{compressed}}{CS_{original}} \times 100[\%]$$

$CS_{compressed}$ : checkpoint size with compression

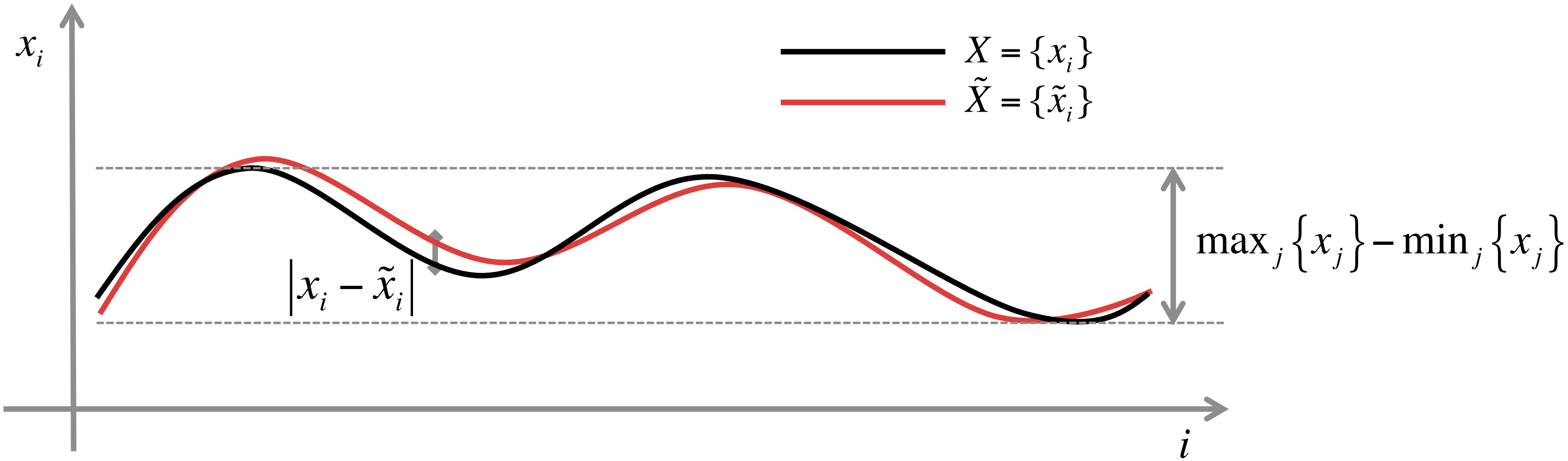
$CS_{original}$ : original checkpoint size

## Relative error

$$RE_i = \frac{|x_i - \tilde{x}_i|}{\max_j\{x_j\} - \min_j\{x_j\}}$$

$X = \{x_i\}$ : original data

$\tilde{X} = \{\tilde{x}_i\}$ : data with our approach



# Evaluation of Compression Rate

Original checkpoint data (Floating point array)

Apply *gzip*

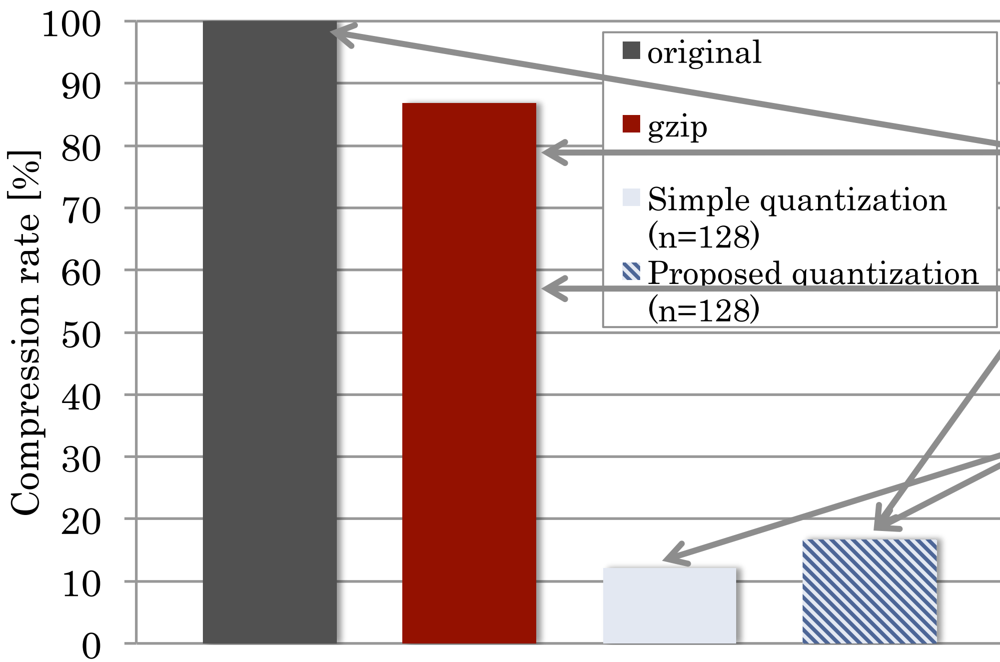
Compressed checkpoint data with *gzip*

Apply our approach with simple quantization
( The number of division *n is 128* )

Compressed checkpoint data with simple quantization

Apply our approach with proposal quantization
( The number of division *n is 128* )

Compressed checkpoint data with proposal quantization




If we apply *gzip* to scientific checkpoint data directly, the size is reduced by about 13%

approach reduces checkpoint size by

Simple quantization achieves better compression rate, but introduces a larger error than proposal quantization

$$RE_i = \frac{x_i - \tilde{x}_i}{\max_j\{x_j\} - \min_j\{x_j\}}$$

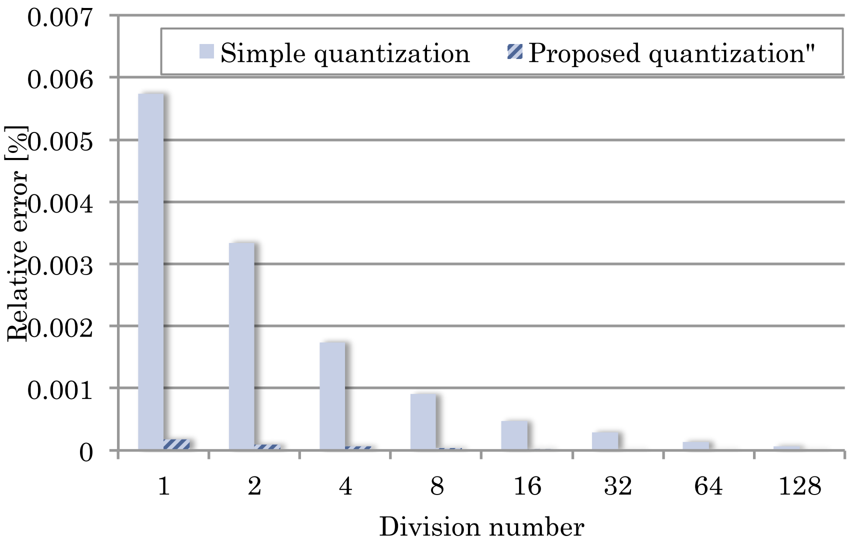
# Evaluation of Errors

**Errors are reduced with # of division (*n*) increasing**

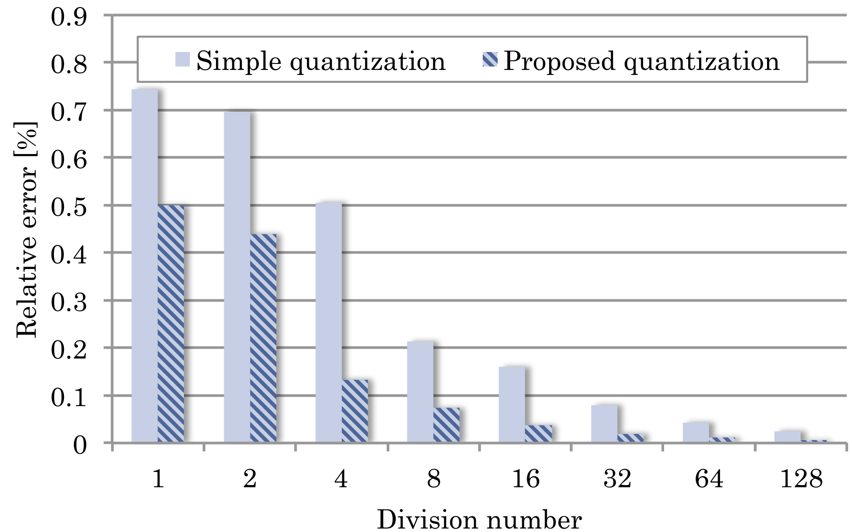
- Errors are reduced by about 98% at $n$ = 128 compared with $n$ = 1

**Proposed quantization reduces an error compared with simple one**

- The degree of reduction in errors is different depending on arrays



An average error on pressure array



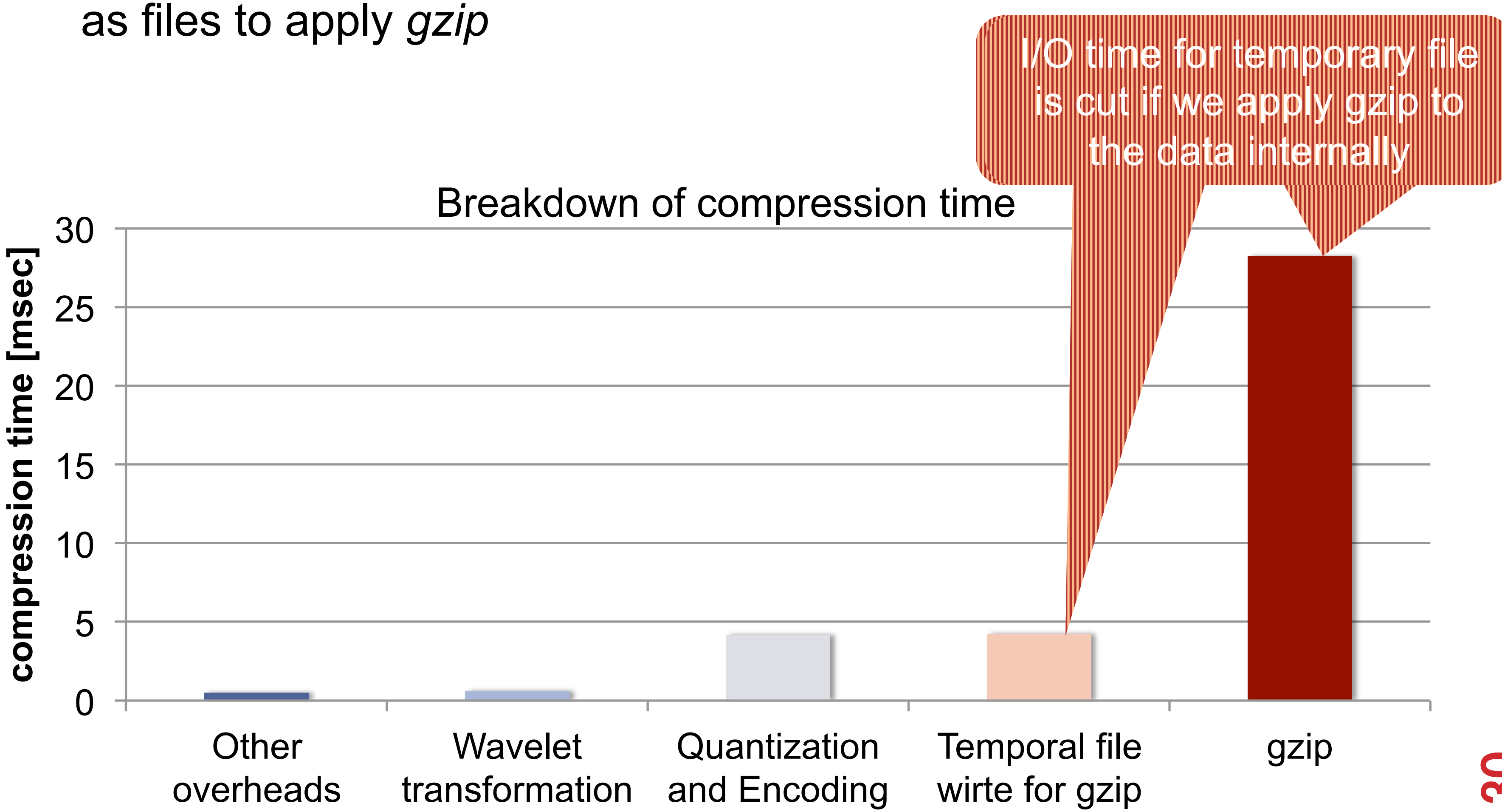
An average error on temperature array

On all variables, maximum errors are within 5% and average errors are within 1.2%

# Evaluation of Compression Time

## The figure shows breakdown of compression time

- The current implementation writes temporary file checkpoint data as files to apply *gzip*



# Estimation on Massively Parallel Case

## Assumptions for compression time

- I/O throughput…20GB/s
- Checkpoint size that each process has…about 1.5MB

  →Total checkpoint size…about (1.5 × # of parallelism)MB

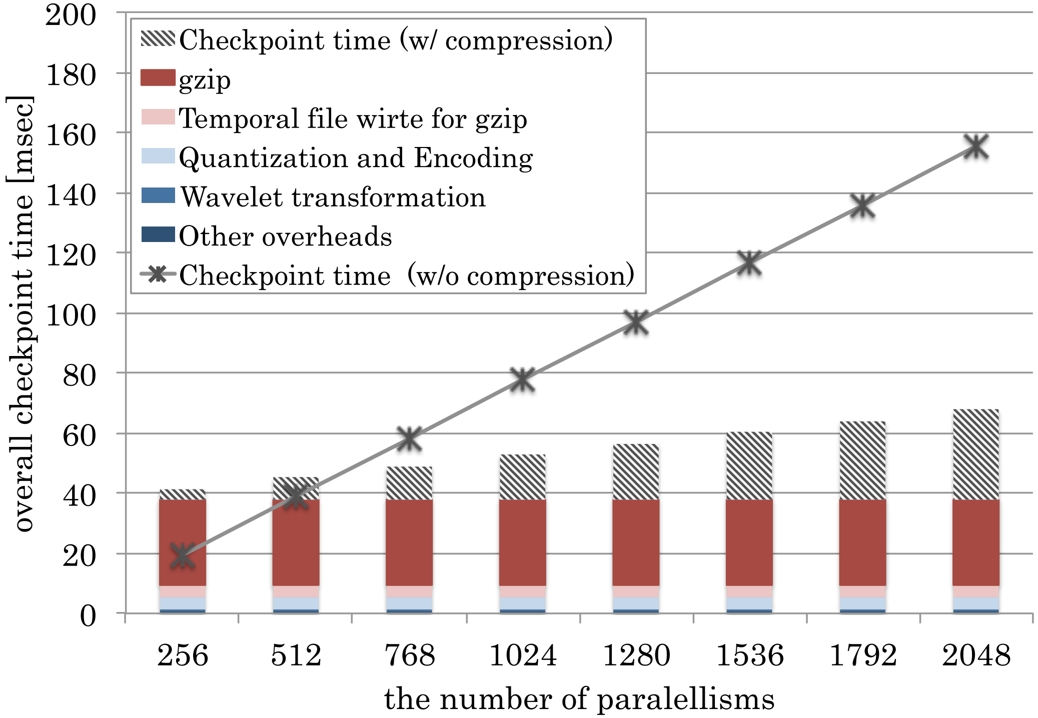
## Actual survey

- Compression time
- Compression rate

## Calculation from assumption

- I/O time

$$\frac{\text{Total checkpoint size}(\times \text{compression rate})}{\text{I/O Throughput}}$$



# Estimation on Massively Pa

If compression time is negligible by increasing # of parallelism, I/O time reduces by about 81%

**An assumption about compression time**

- I/O throughput…20GB/s
- Checkpoint size that each process has…about 1.5MB

→Total checkpoint size…about (1.5 × # of parallelism)MB

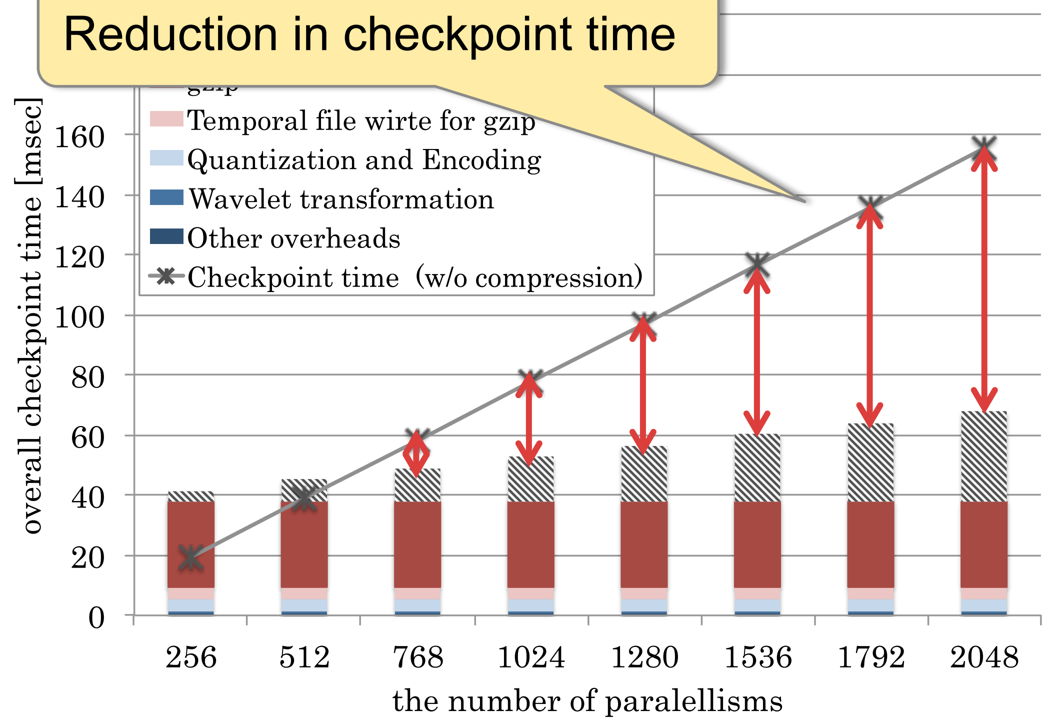
**Each process compresses 1.5MB checkpoint data in spite of # of parallelism**

- Compression time is constant

**I/O time depends on total checkpoint size**

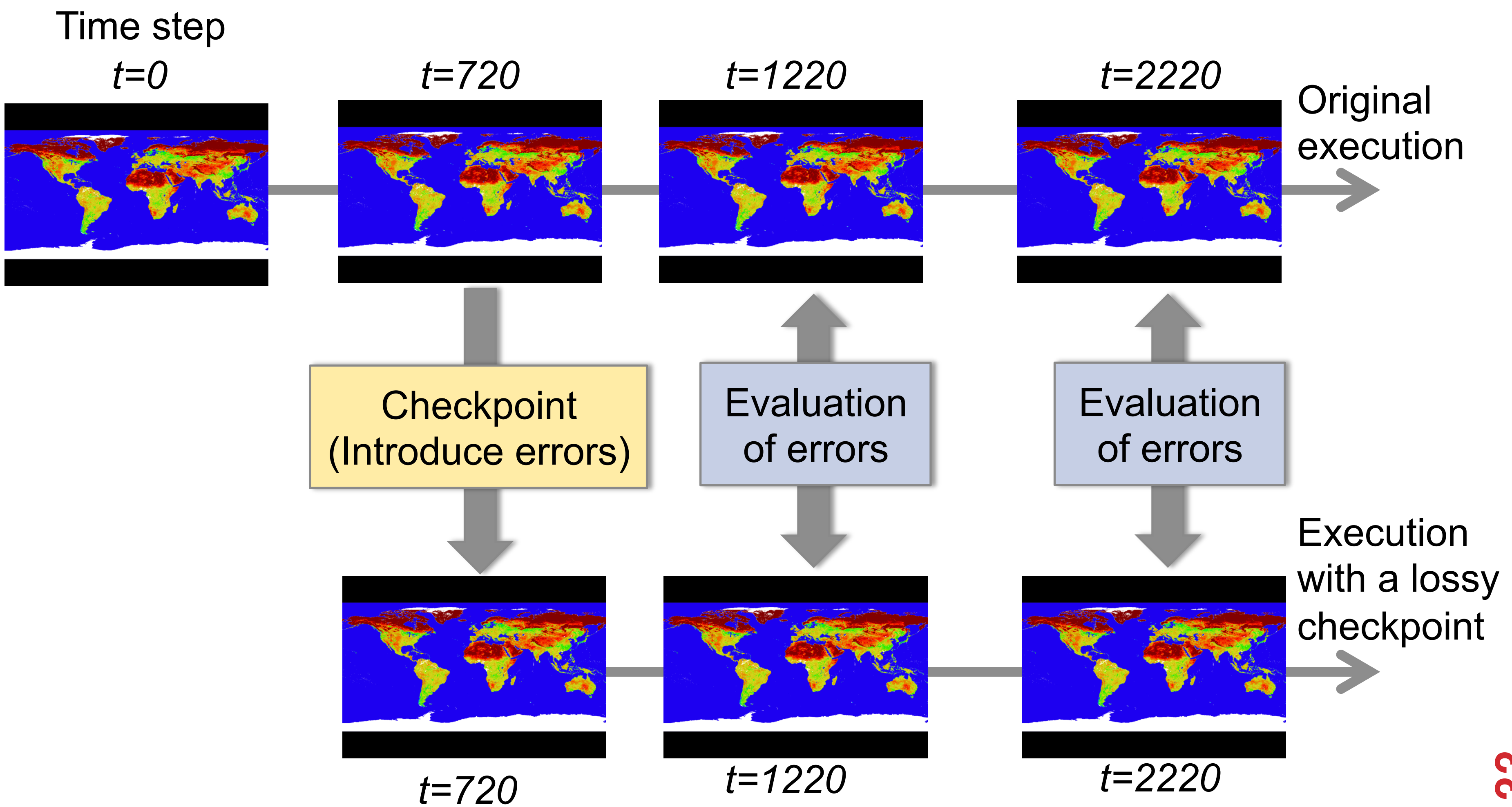
**Our approach takes advantage when # of parallelism increases**

Reduction in checkpoint time



# Evaluation Method for Error Transition

**We evaluate error transition as shown in bottom figure**



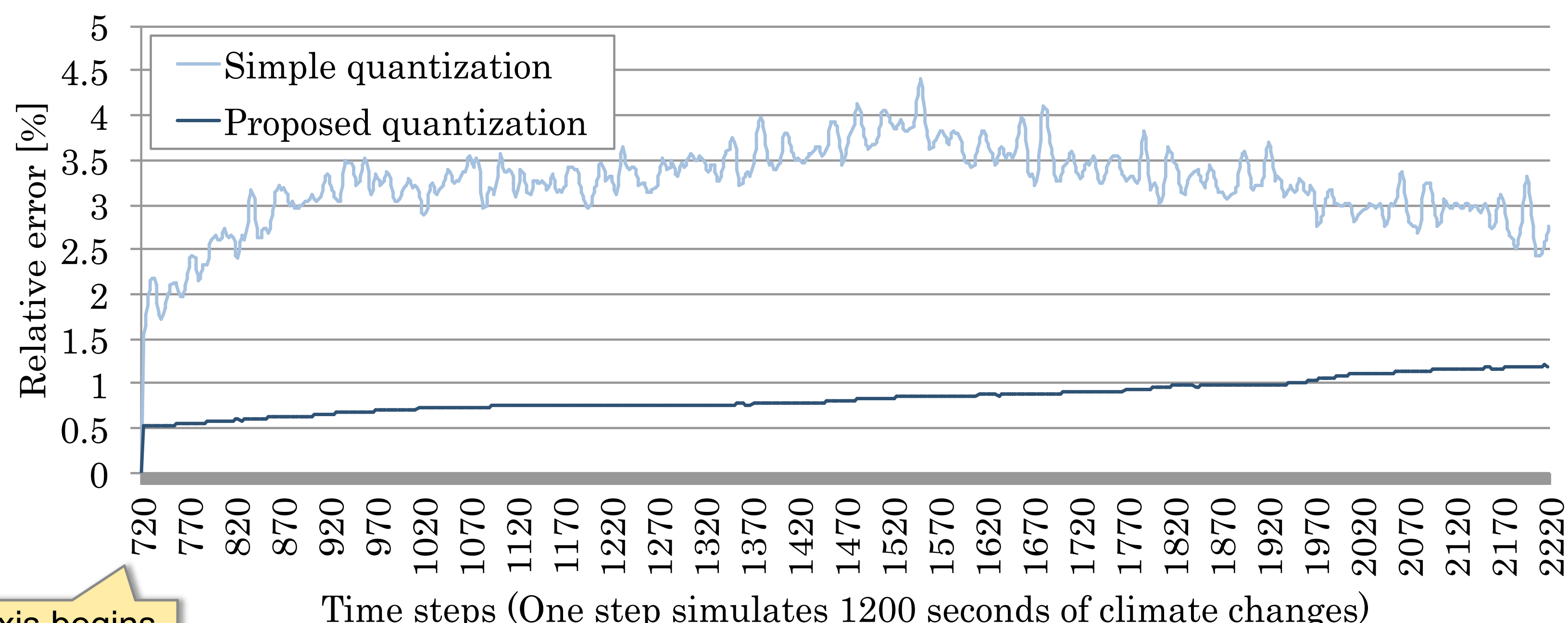
# Evaluation of Error Transition

**Lossy compression is applied to checkpoint data**

→ Applications use the data with errors

→ The errors may diverge even if initial errors are small

**Lossy compression has been becoming feasible for checkpoint image data in an N-body cosmology simulation [※]**




x-axis begins from 720

[※ Xiang Ni, SC, 2014, "Lossy compression for checkpointing: Fallible or feasible?" ]

# Related Work

**Multi-level checkpointing [Bautista-Gomez, SC, 2011]**

- Applications write checkpoint to local storage frequently, and to parallel file system less frequently
- We can combine our approach with this technique

**Incremental checkpointing [Naksinehaboon, CCGRID, 2008]**

- This stores only differences with the last checkpoint
- We can combine our approach with this technique

**MCREngine [Islam, SC, 2012]**

- This study aims to improve compression rate with lossless compression
    - The scheme merges distributed checkpoint images per each variable, and select effective compression methods for each variable

# Conclusion

**Contribution**

- We apply our approach to real climate application, NICAM, then overall checkpoint time included compression time is reduced by 81% with 1.2% relative error on average in particular situation
- We improve compression rate compared to lossless compression with the same degree of inherent errors to scientific simulations, such as sensor errors and model errors

**Future work**

- Improvement of the compression algorithm
    - Reduce compression rate and errors
- Investigation of the feasibility in other applications
- Combination with other efforts

# THANK YOU!

**I’m studying English, so I would appreciate it if you could speak slowly for me…**