SAGE: Self-Tuning Approximation for Graphics Engines

Mehrzad Samadi[1], Janghaeng Lee[1], D. Anoushe Jamshidi[1], Amir Hormati[2], and Scott Mahlke[1]

University of Michigan[1], Google Inc.[2]

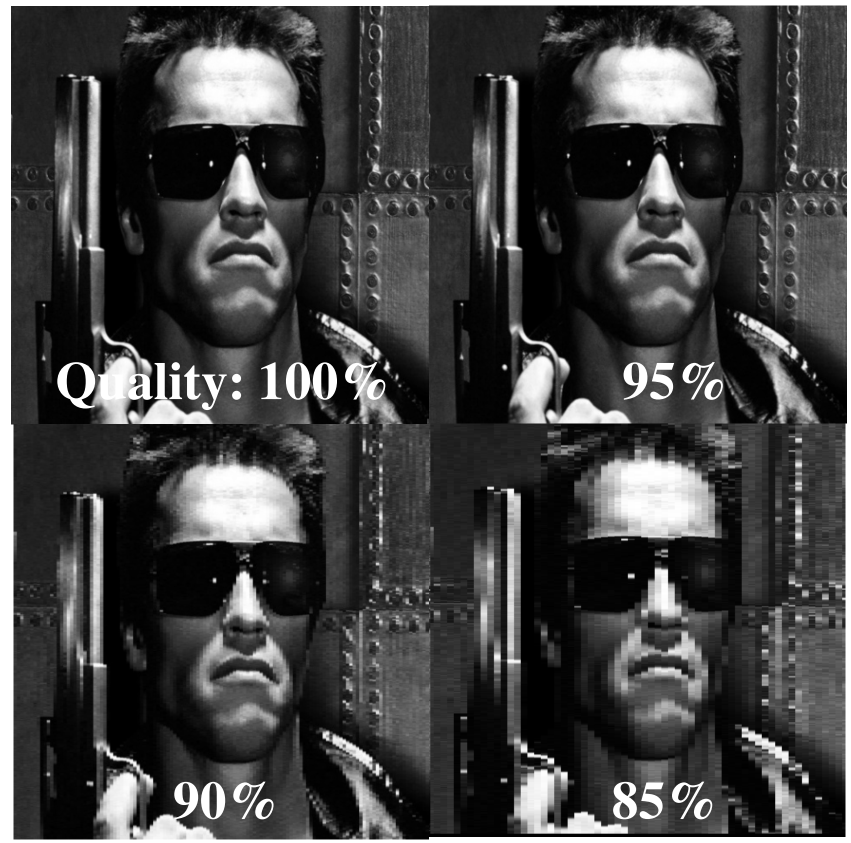
December 2013

University of Michigan
Electrical Engineering and Computer Science

Compilers creating custom processors

Approximate Computing

- Different domains:
  - Machine Learning
  - Image Processing
  - Video Processing
  - Physical Simulation
  - …



Less work → Higher performance
Lower power consumption

Ubiquitous Graphics Processing Units

- Wide range of devices

Super Computers

Servers

Desktops

Cell Phones

- Mostly regular applications
- Works on large data sets

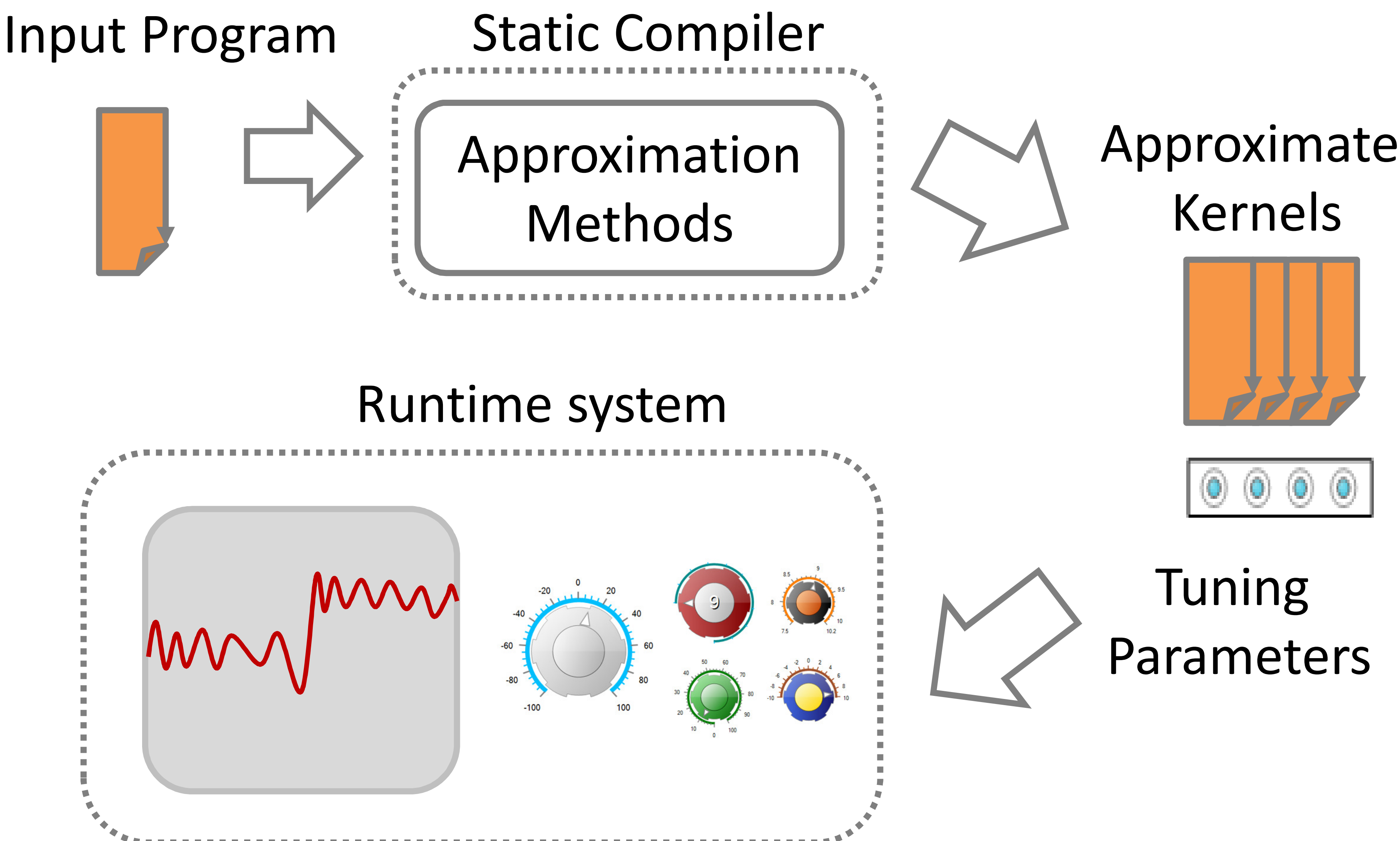
Good opportunity for automatic approximation

SAGE Framework

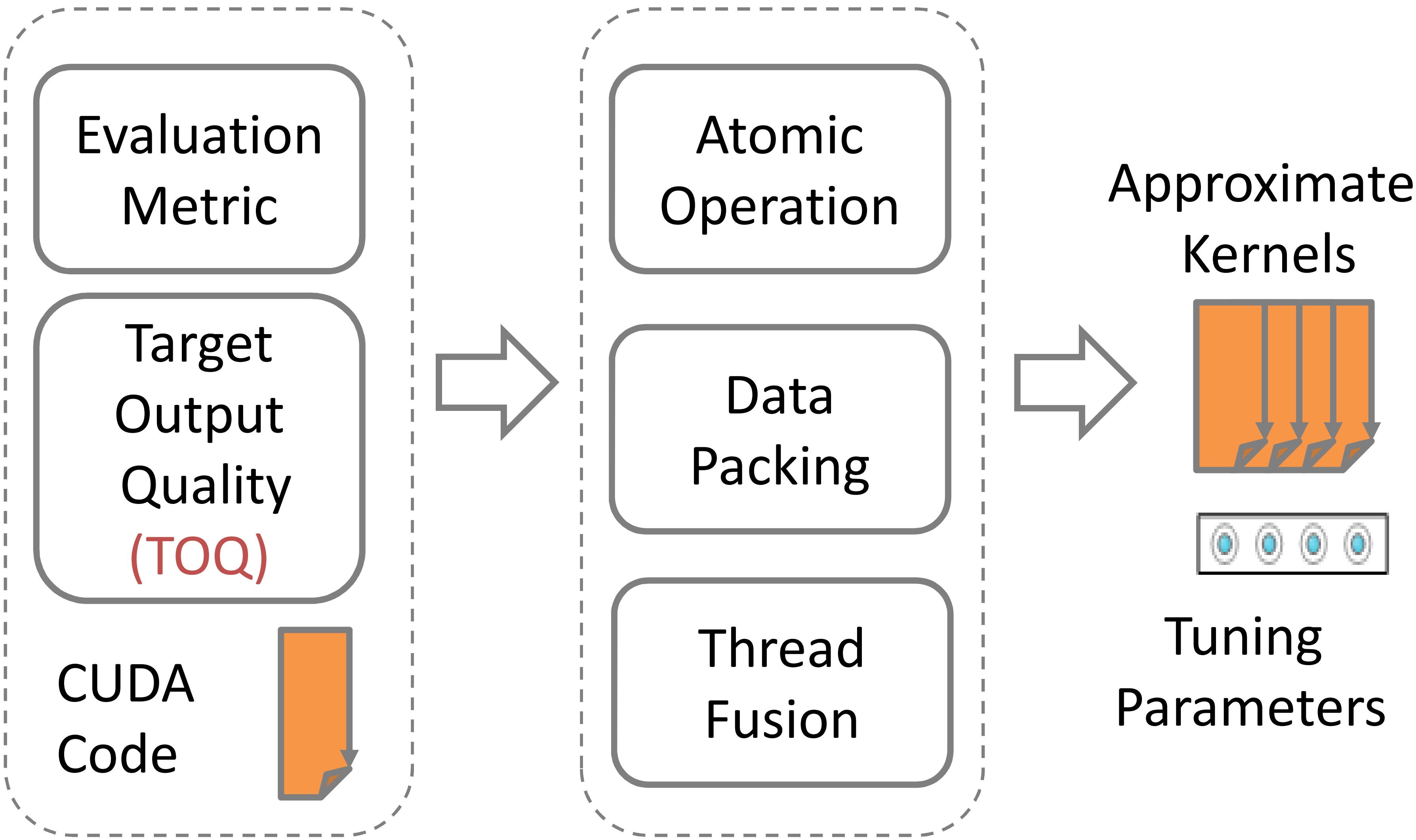
- Si
  -
  -
- Self-Tuning Approximation on Graphics Engines
  - Write the program once
  - Automatic approximation
  - Self-tuning dynamically

Overview

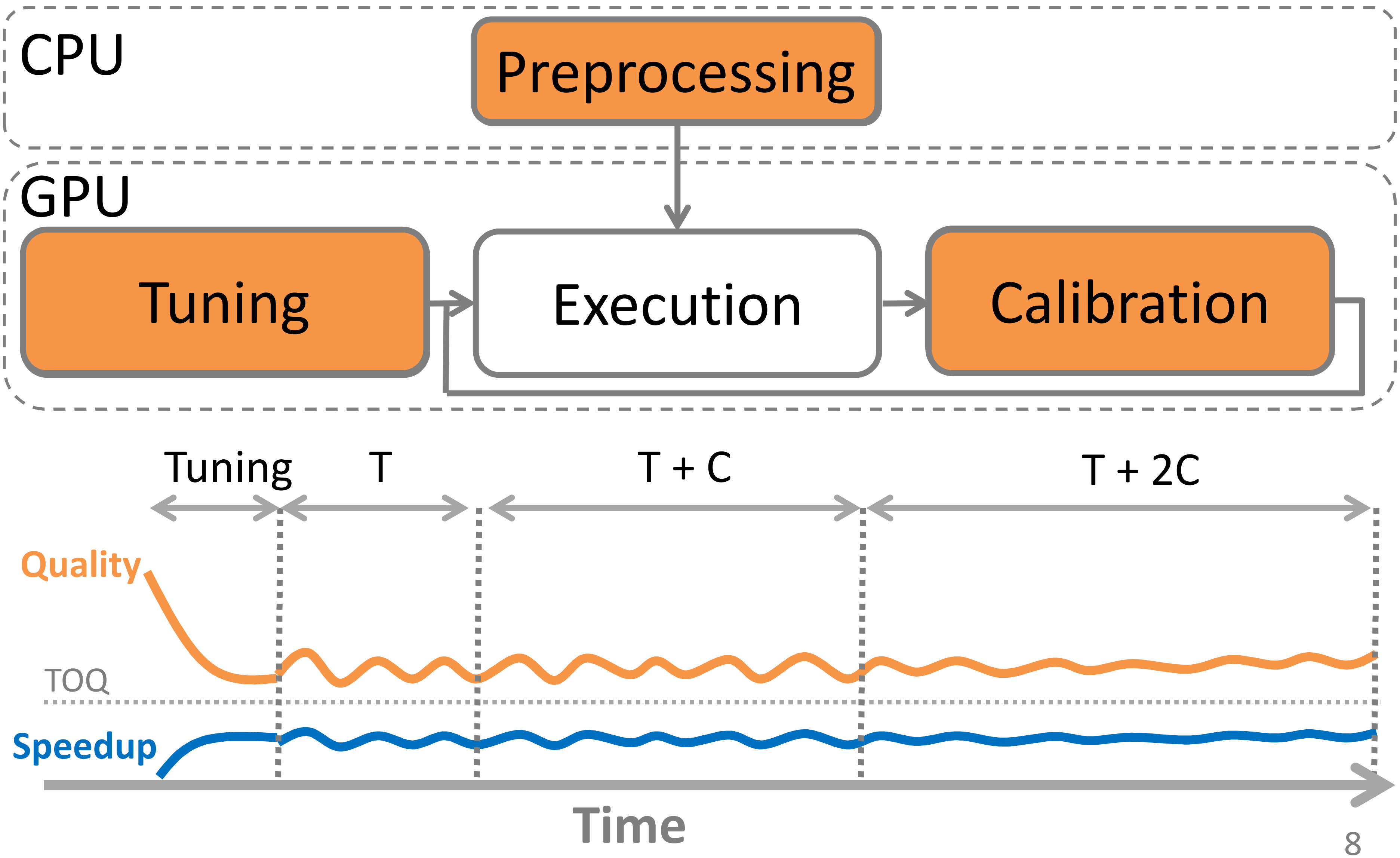
SAGE Framework



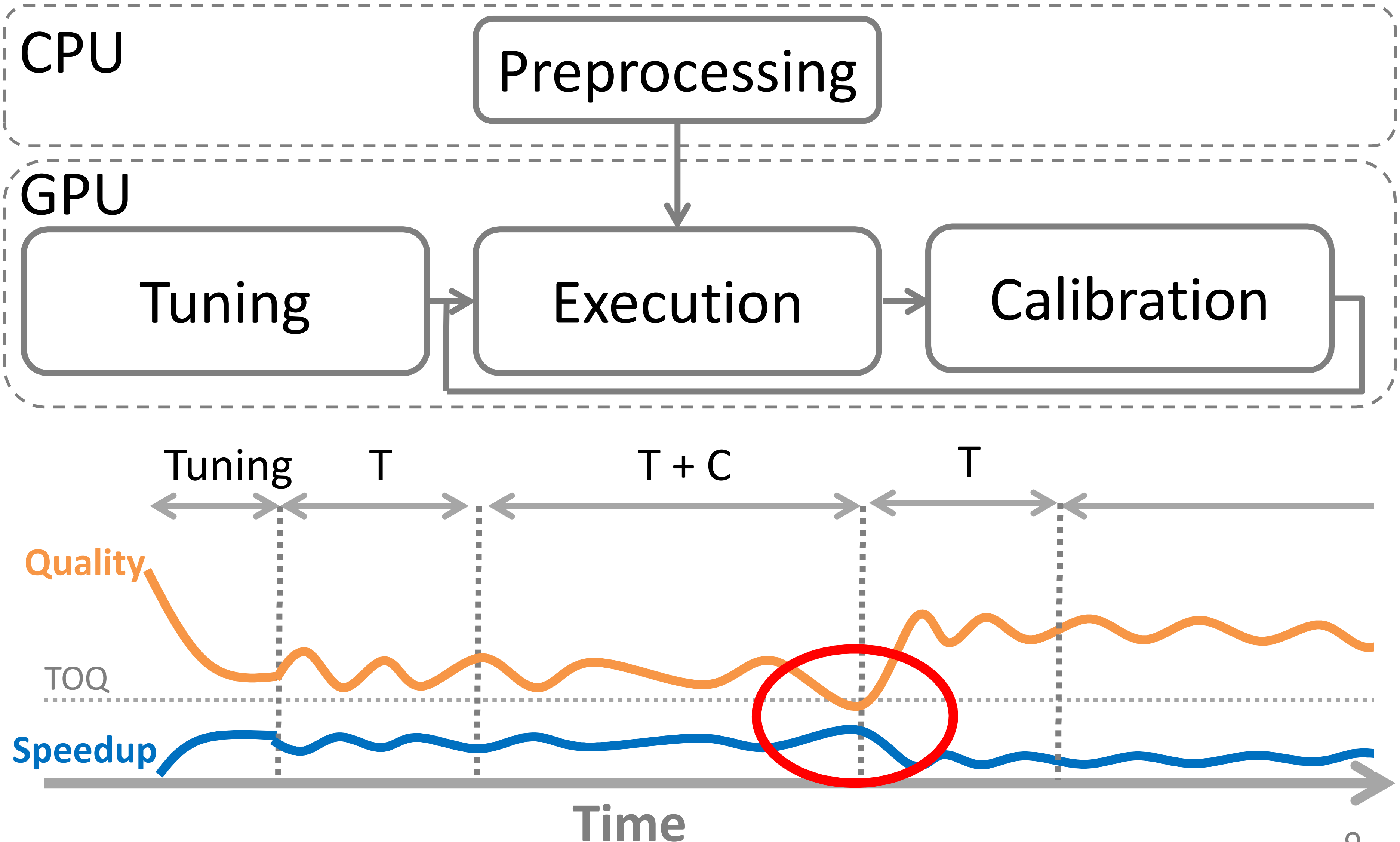
Static Compilation



Runtime System

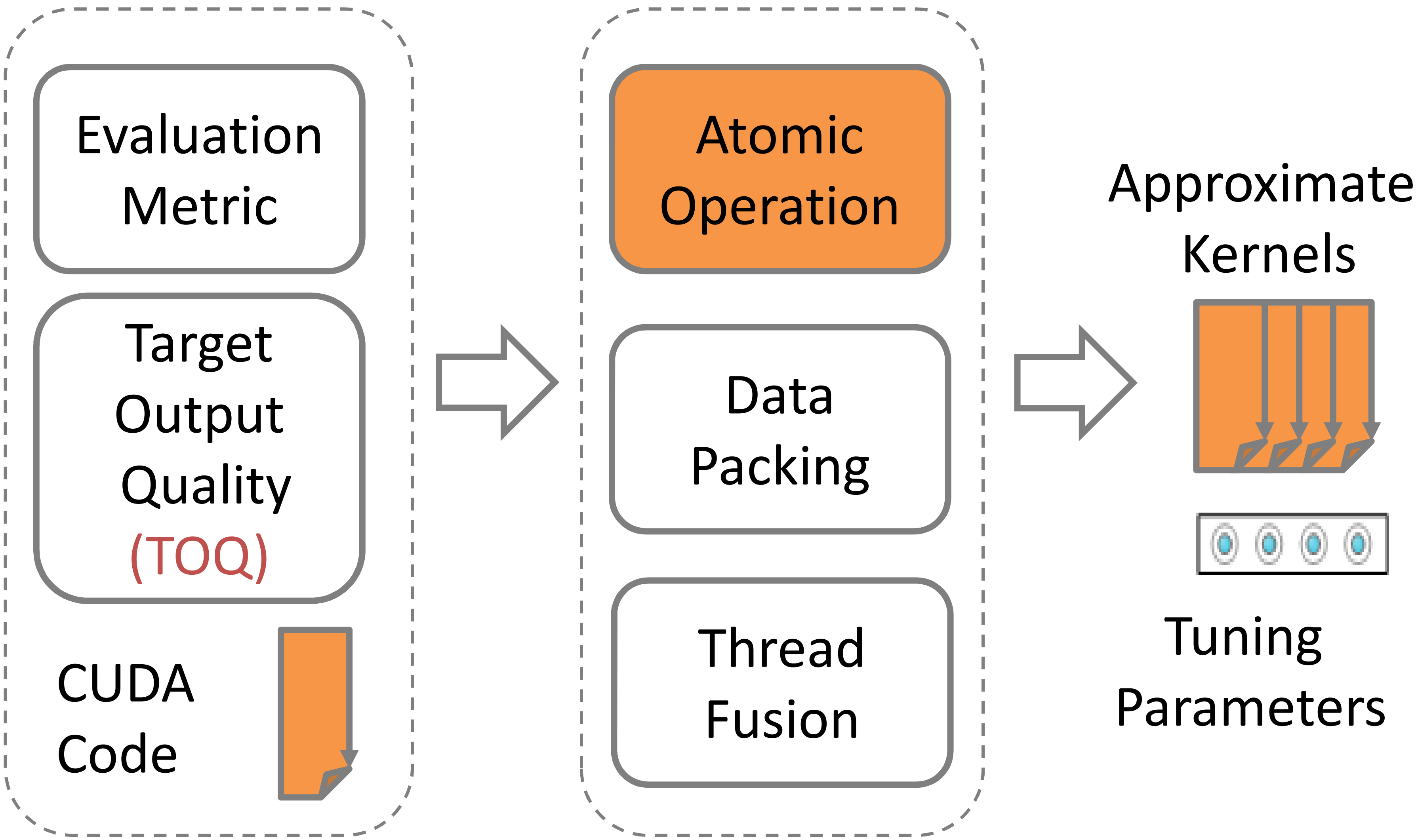


Runtime System



Approximation Methods

Approximation Methods

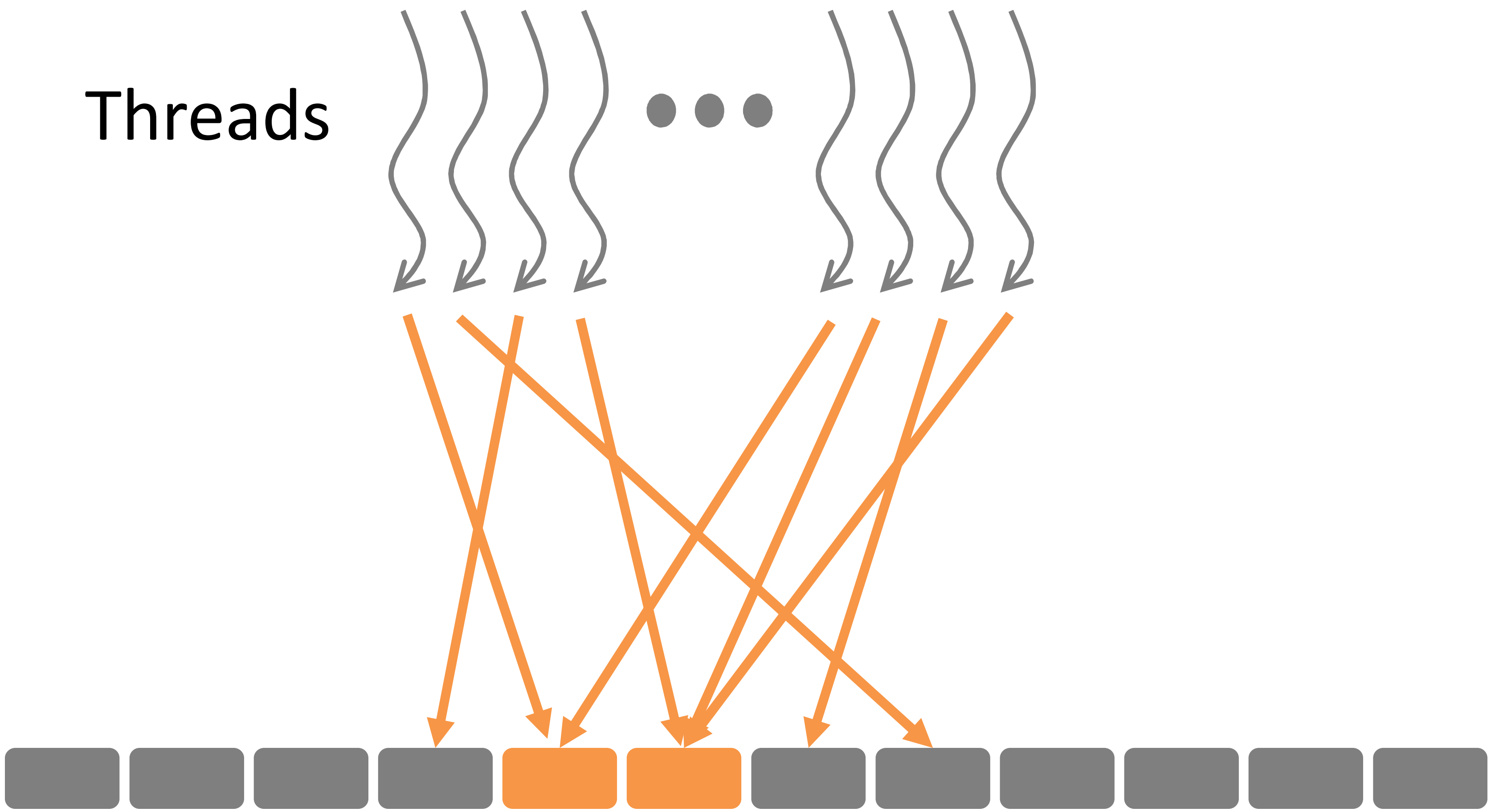


Atomic Operations

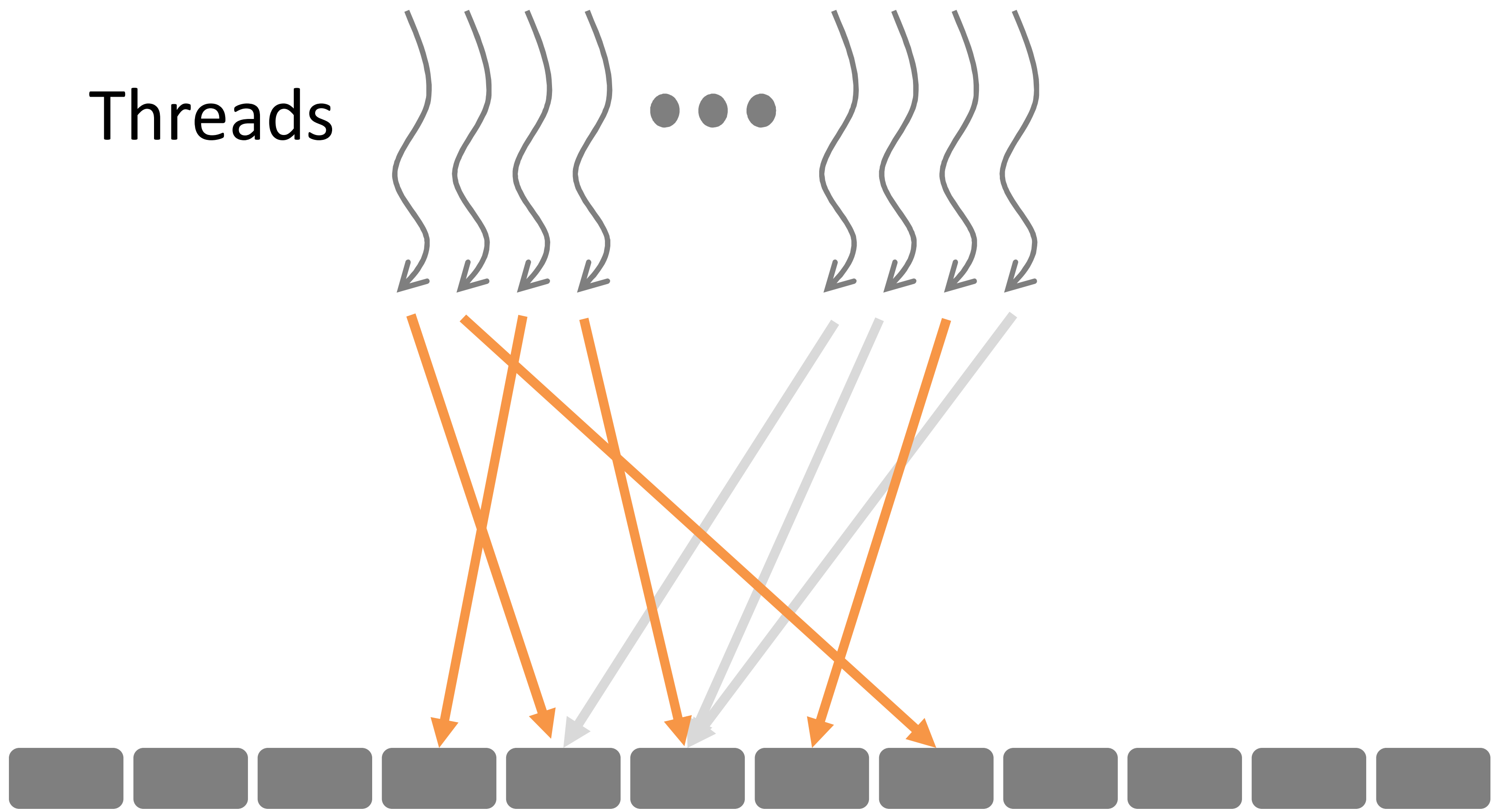
- Atomic operations update a memory location such that the update appears to happen atomically

```
// Compute histogram of colors in an image
__global__ void histogram(int n, int* color, int* bucket)
 int tid = threadIdx.x + blockDim.x * blockIdx.x;
 int nThreads = gridDim.x * blockDim.x;
 for ( int i = tid ; tid < n; tid += nThreads)
   int c = colors[i];
   atomicAdd(&bucket[c], 1);
```

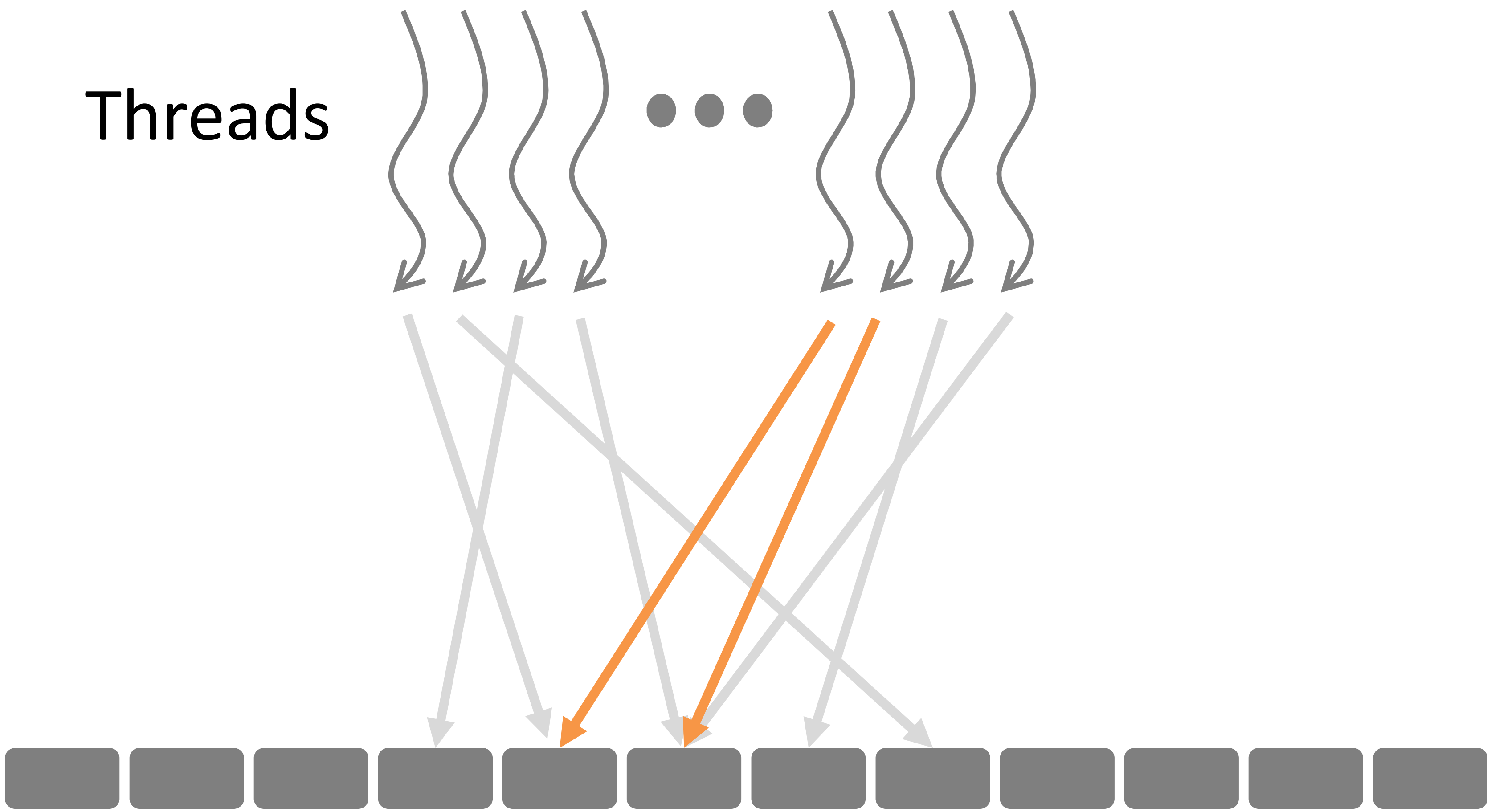
Atomic Operations



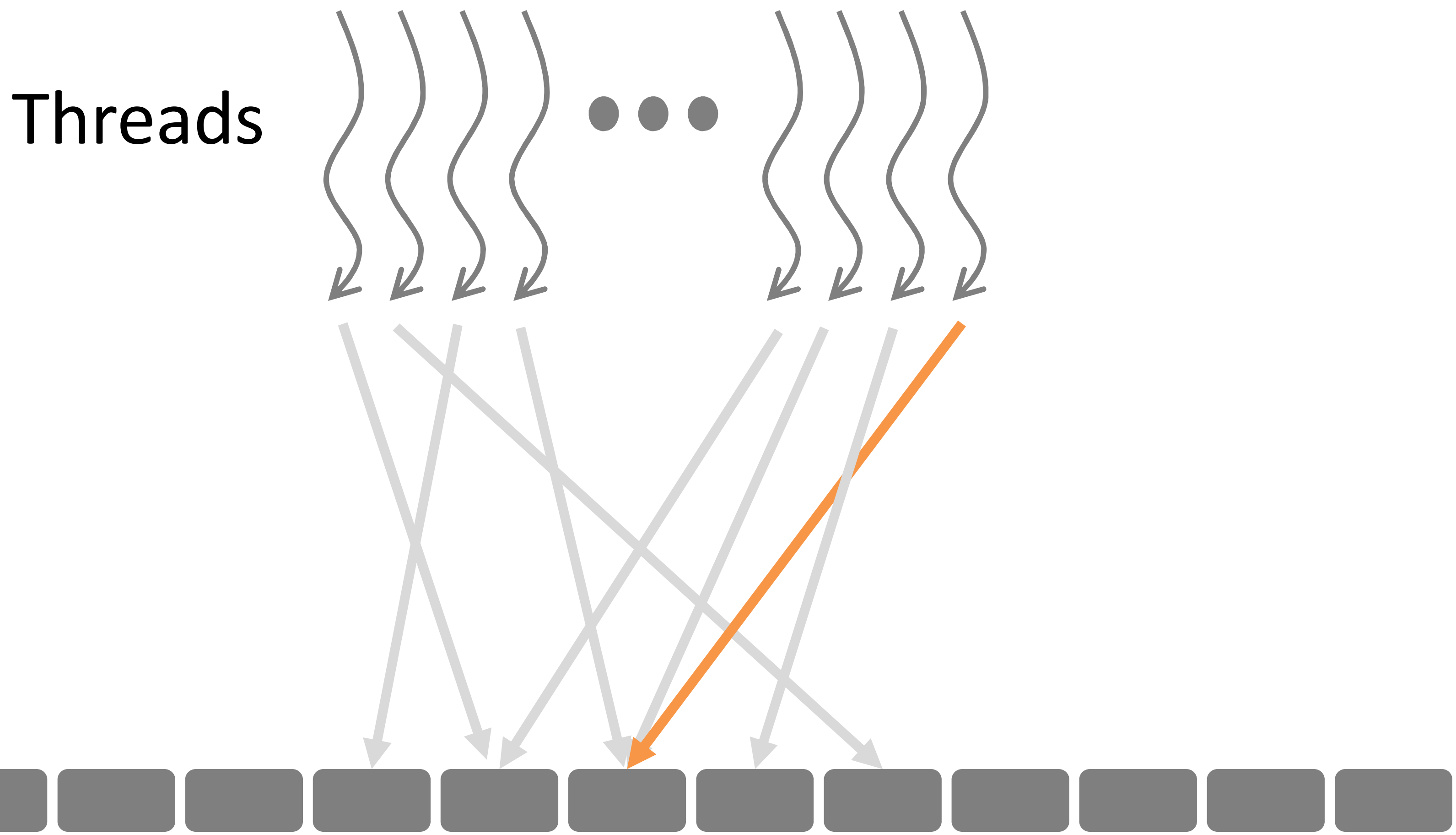
Atomic Operations



Atomic Operations



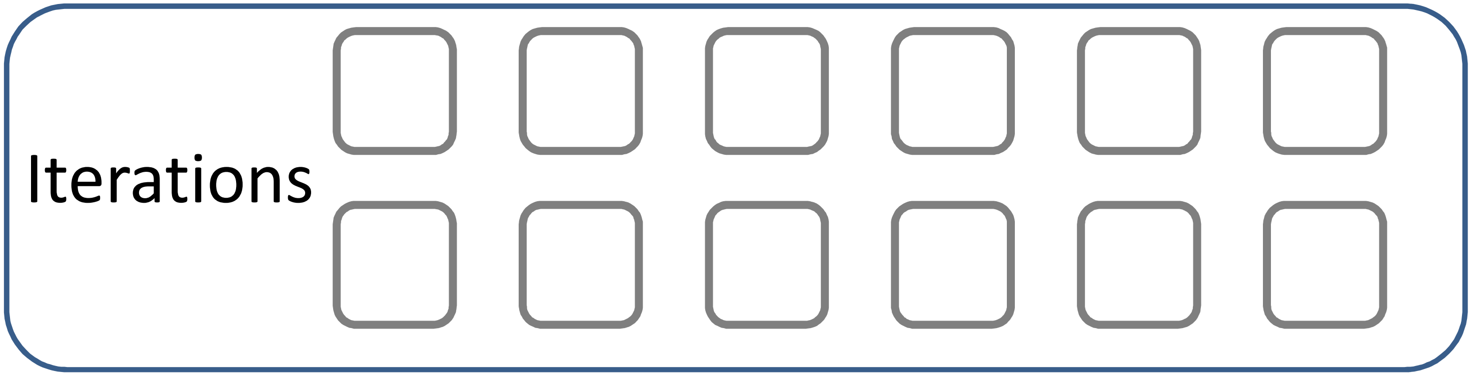
Atomic Operations



Atomic Add

Slowdown
14
12
10
8
6
4
2
0
0
2
4
6
8
10
12
14
16
18
20
22
24
26
28
30
Conflicts per Warp

Atomic Operation Tuning

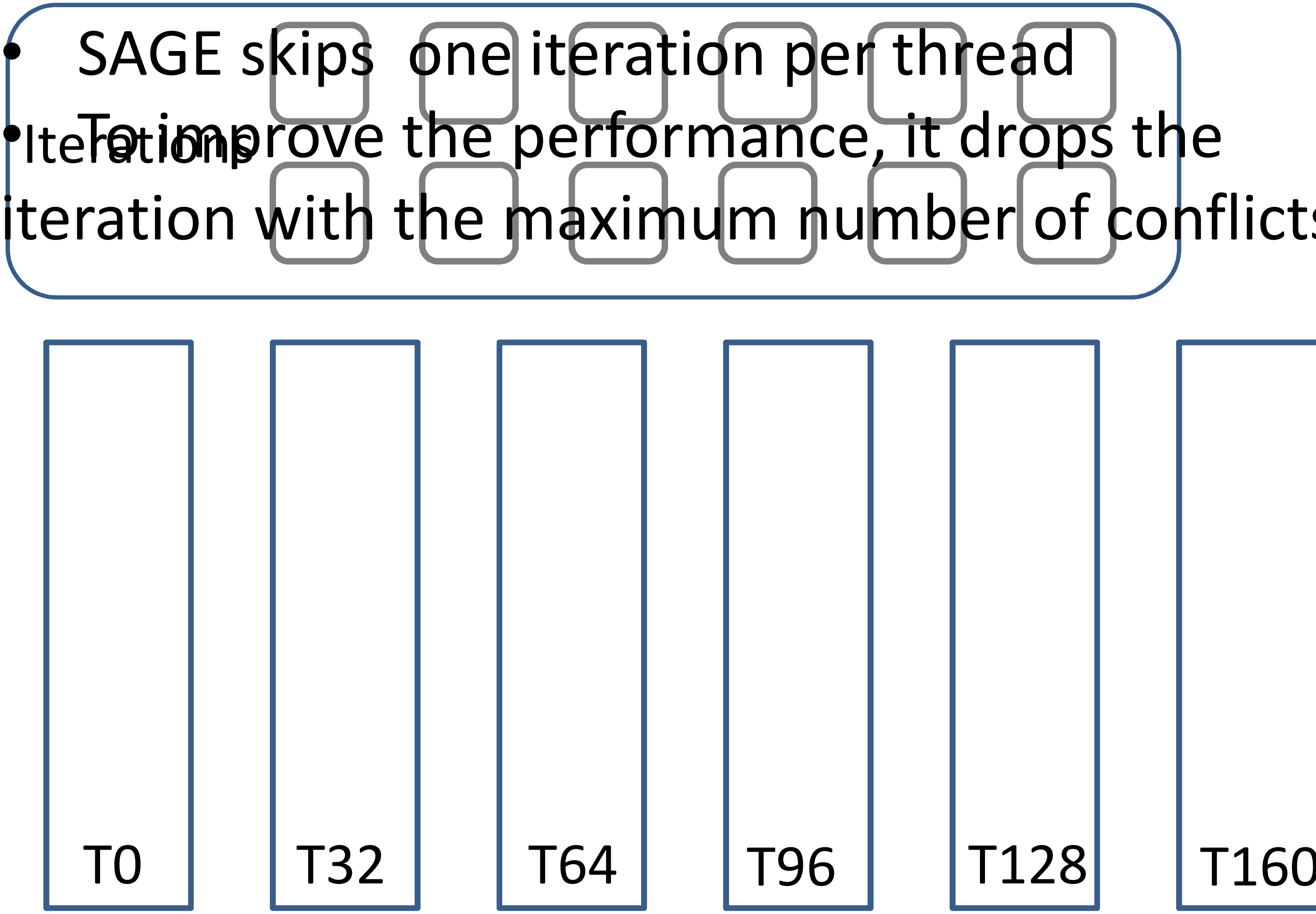



```
// Compute histogram of colors in an image
__global__ void histogram(int n, int* color, int* bucket)
 int tid = threadIdx.x + blockDim.x * blockIdx.x;
 int nThreads = gridDim.x * blockDim.x;
 for ( int i = tid ; tid < n; tid += nThreads)
   int c = colors[i];
   atomicAdd(&bucket[c], 1);
```

T0 T32 T64 T96 T128 T160

Atomic Operation Tuning

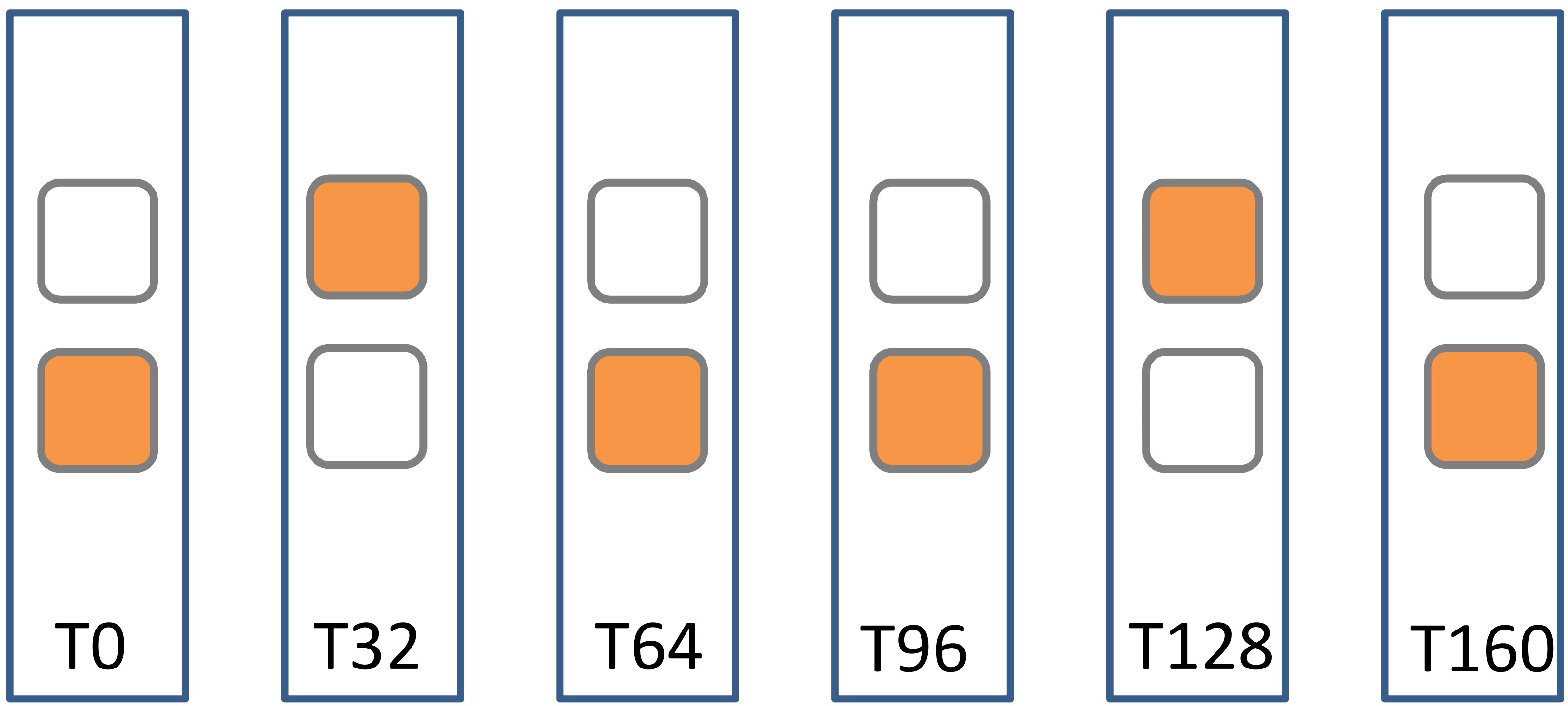
- SAGE skips one iteration per thread
- To improve the performance, it drops the iteration with the maximum number of conflicts



Atomic Operation Tuning

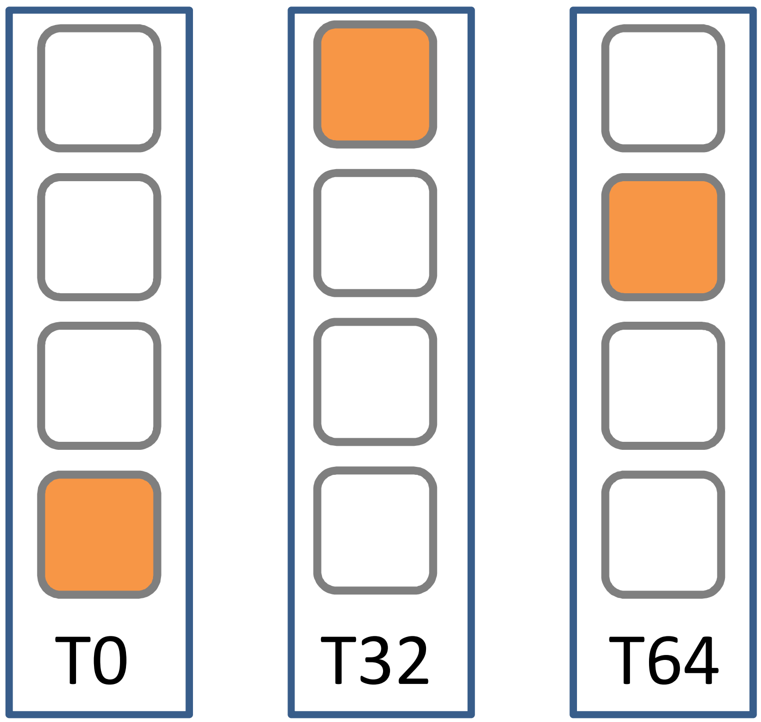
- SAGE skips one iteration per thread
- To improve the performance, it drops the iteration with the maximum number of conflicts

It drops 50% of iterations

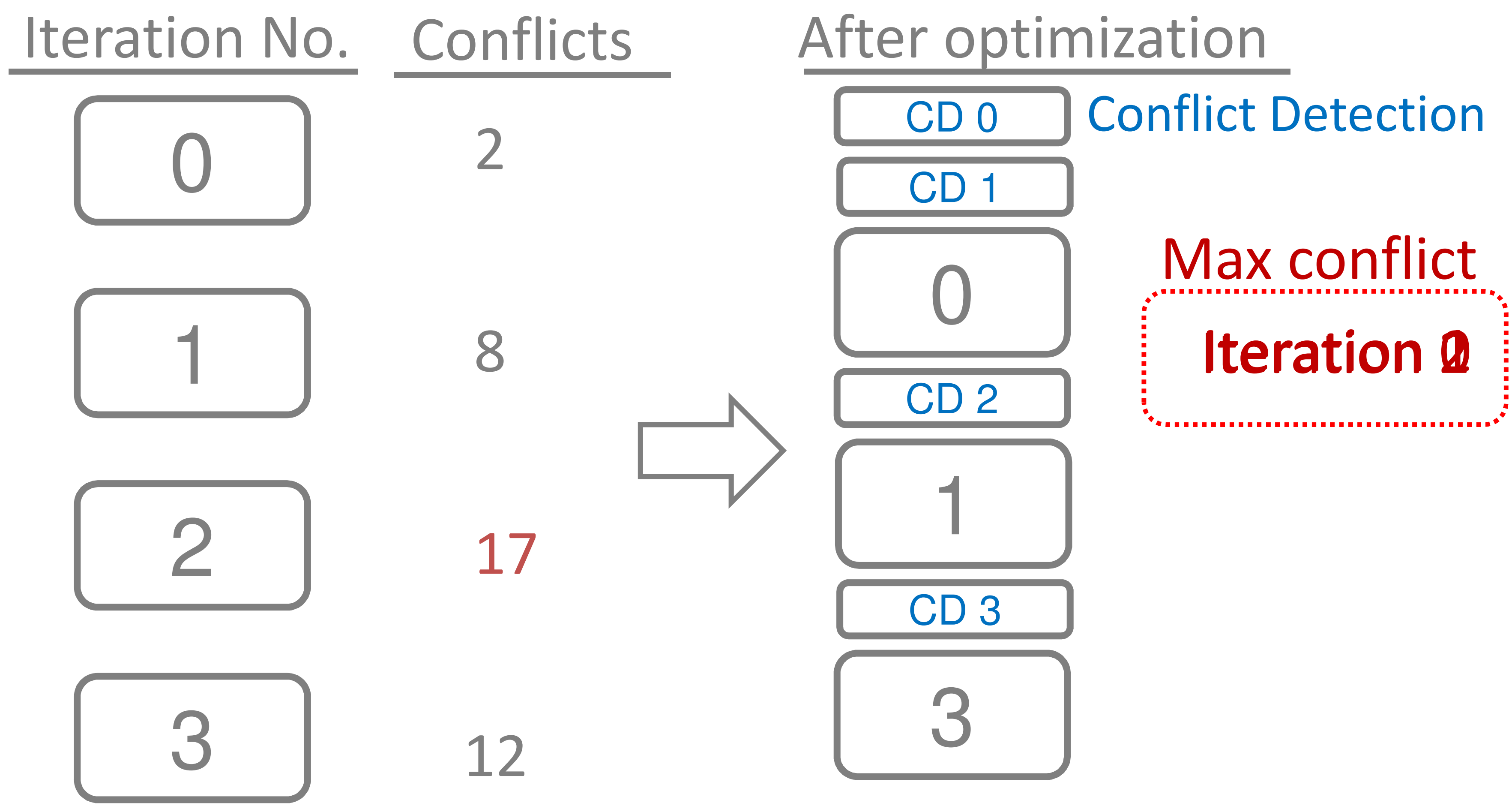


Atomic Operation Tuning

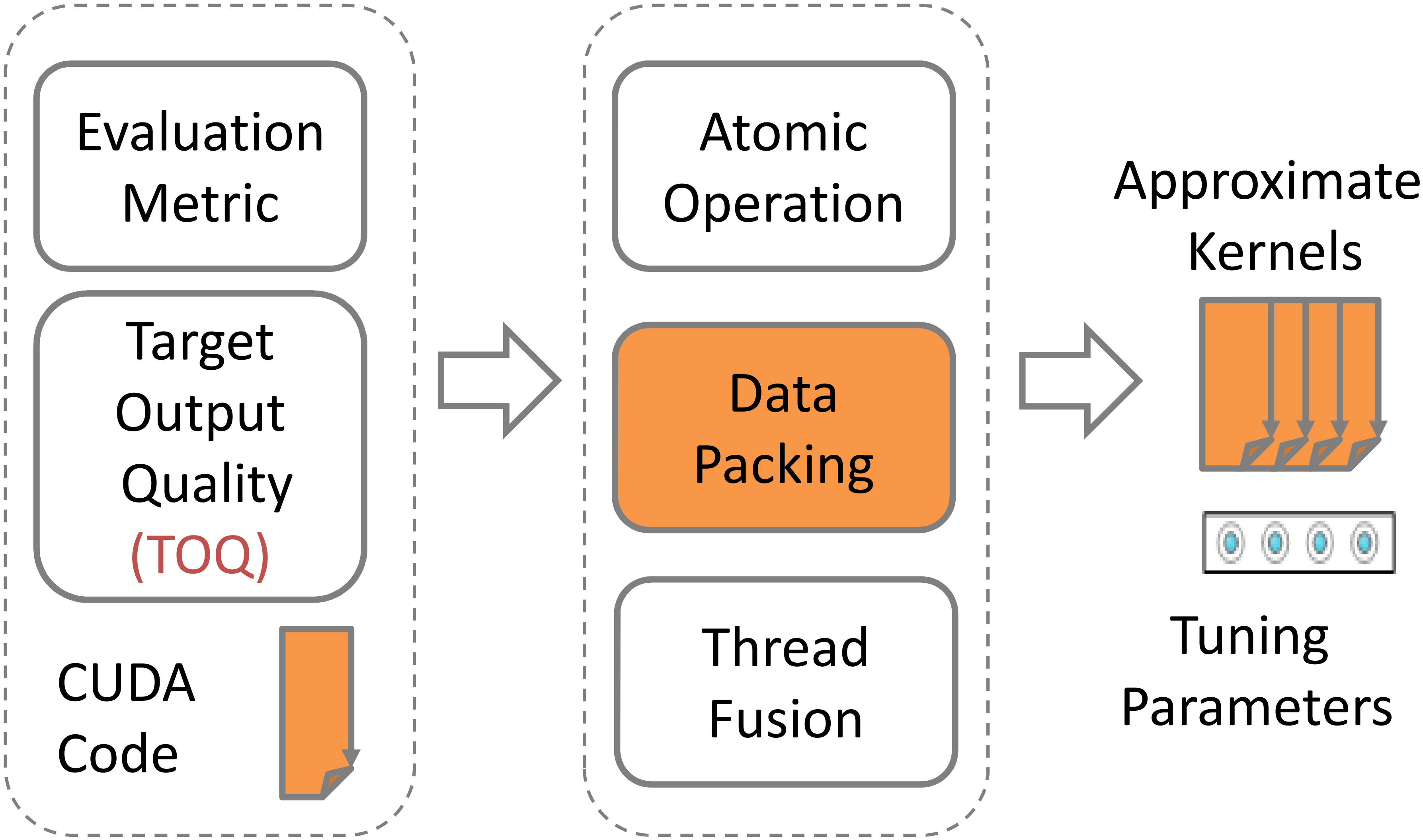
Drop rate goes down to 25%



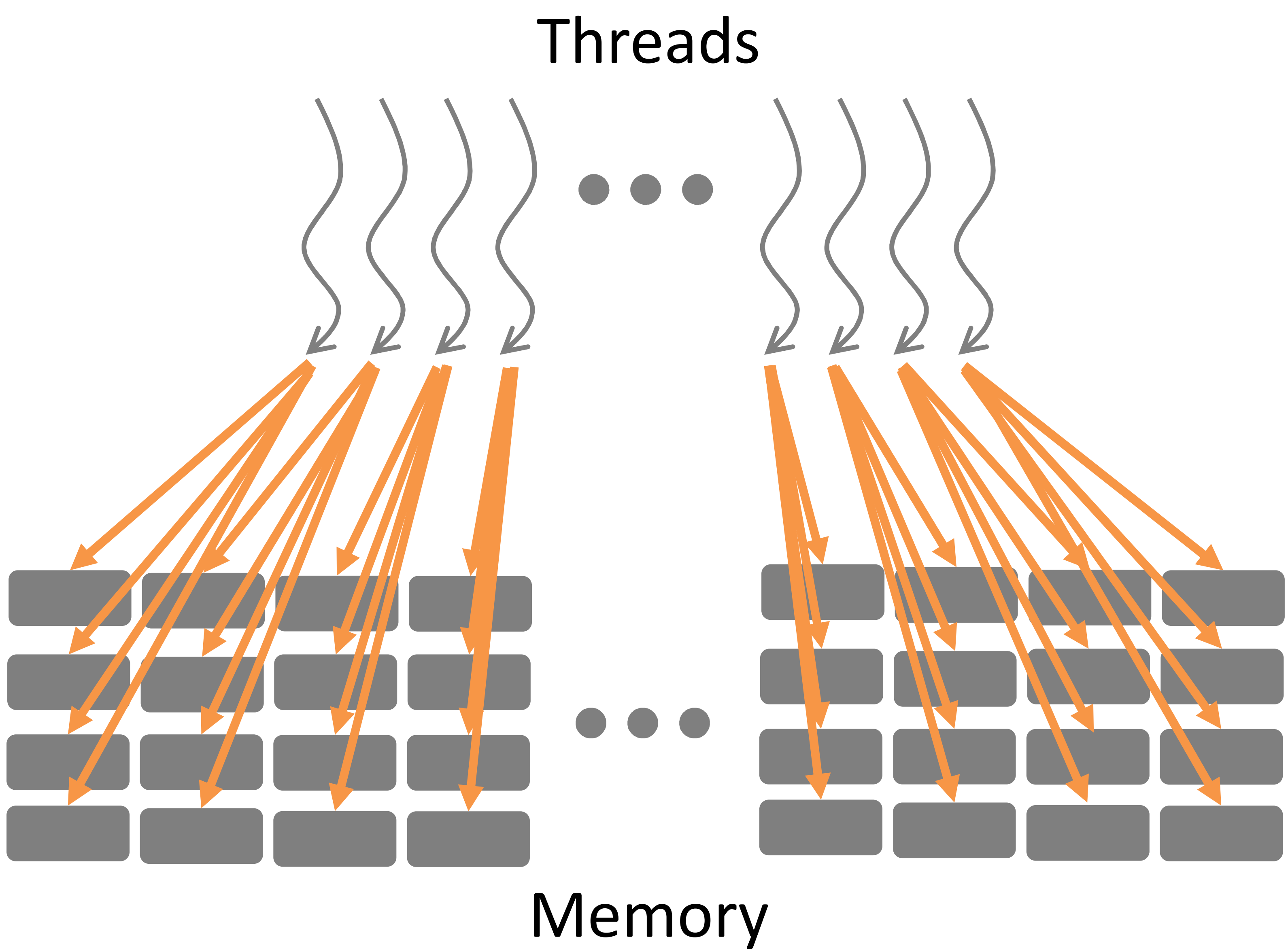
Dropping One Iteration



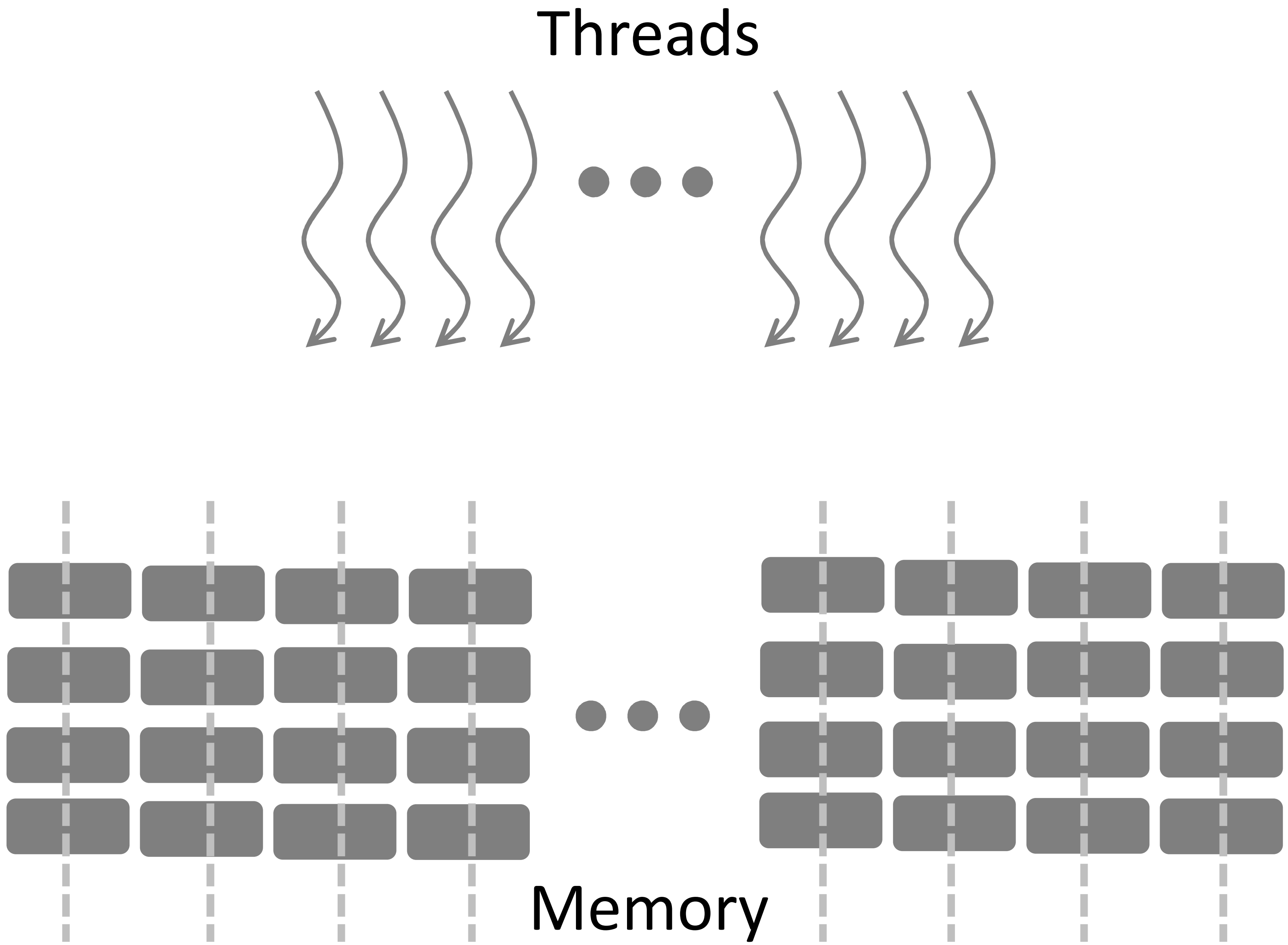
Approximation Methods



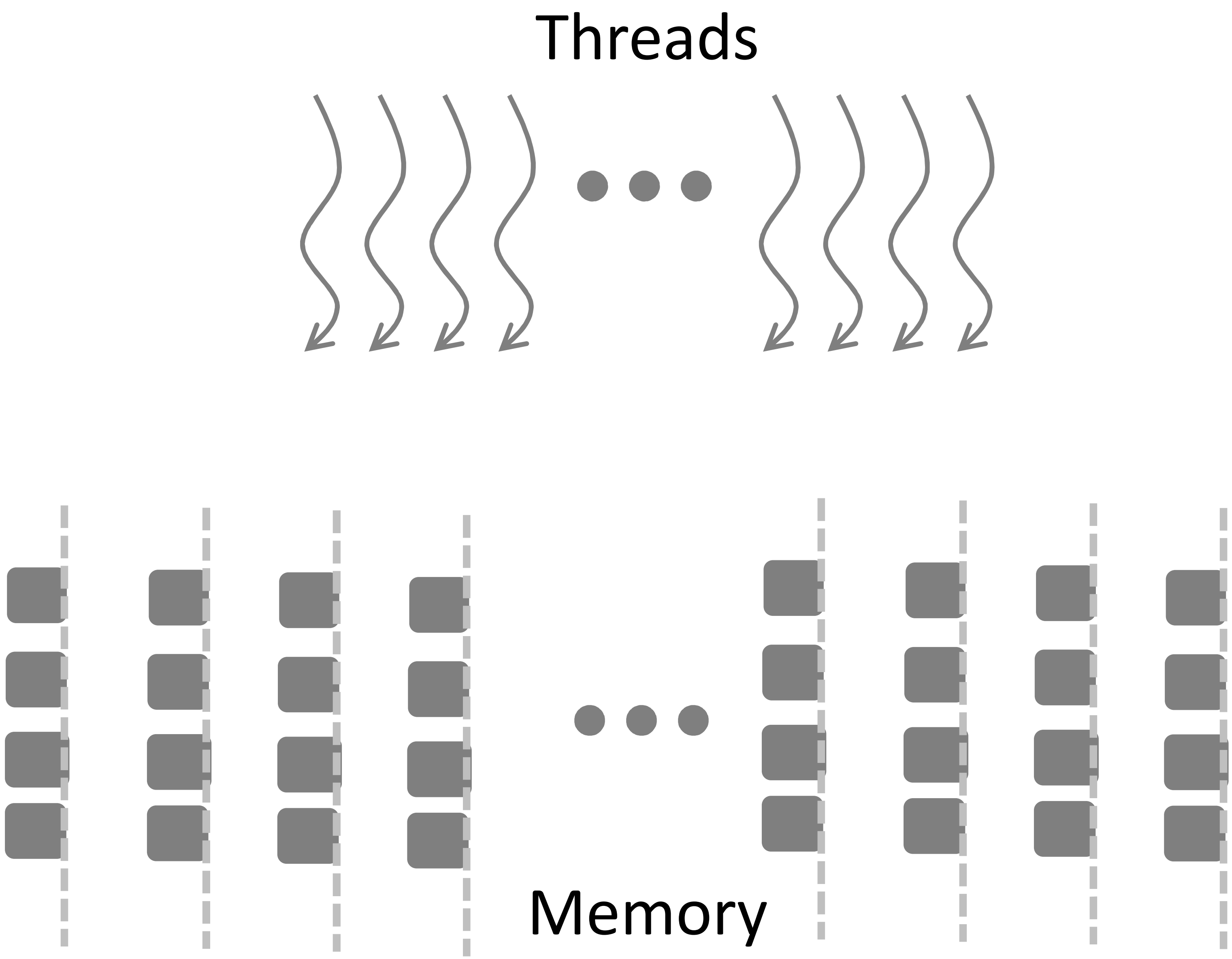
Data Packing



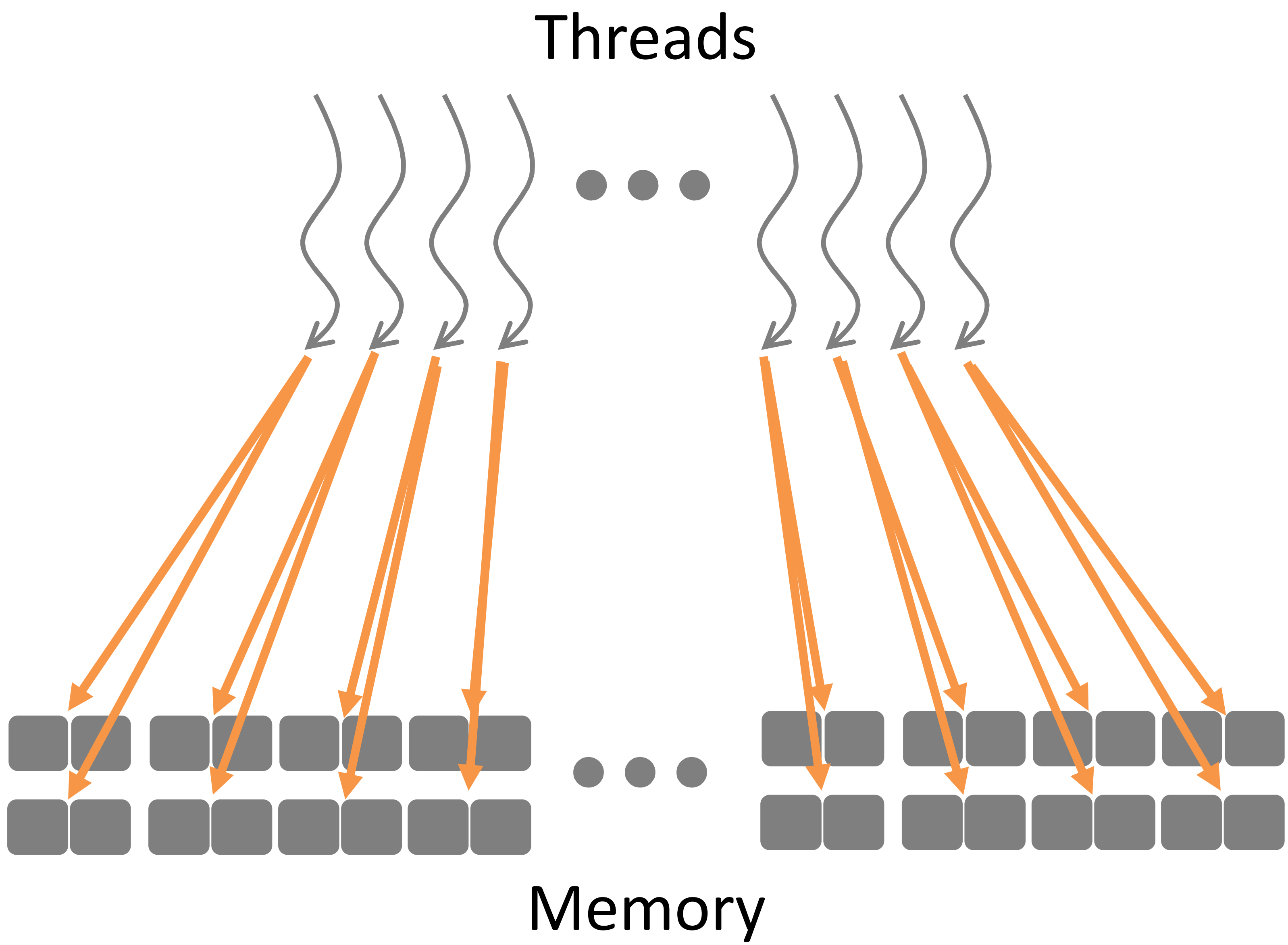
Data Packing



Data Packing

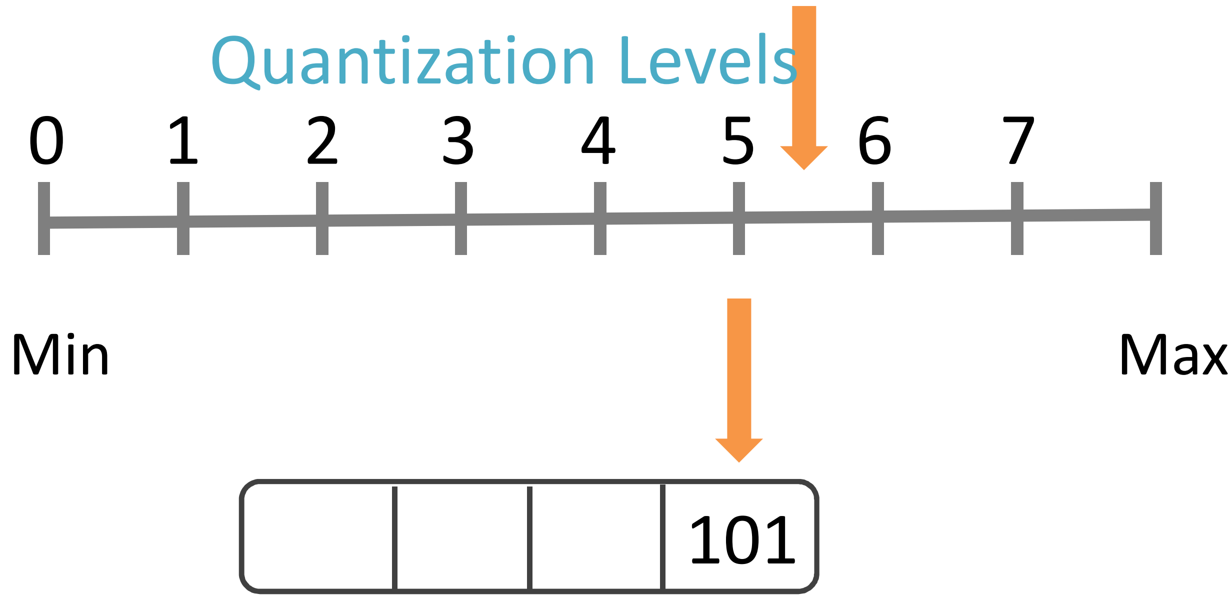


Data Packing



Quantization

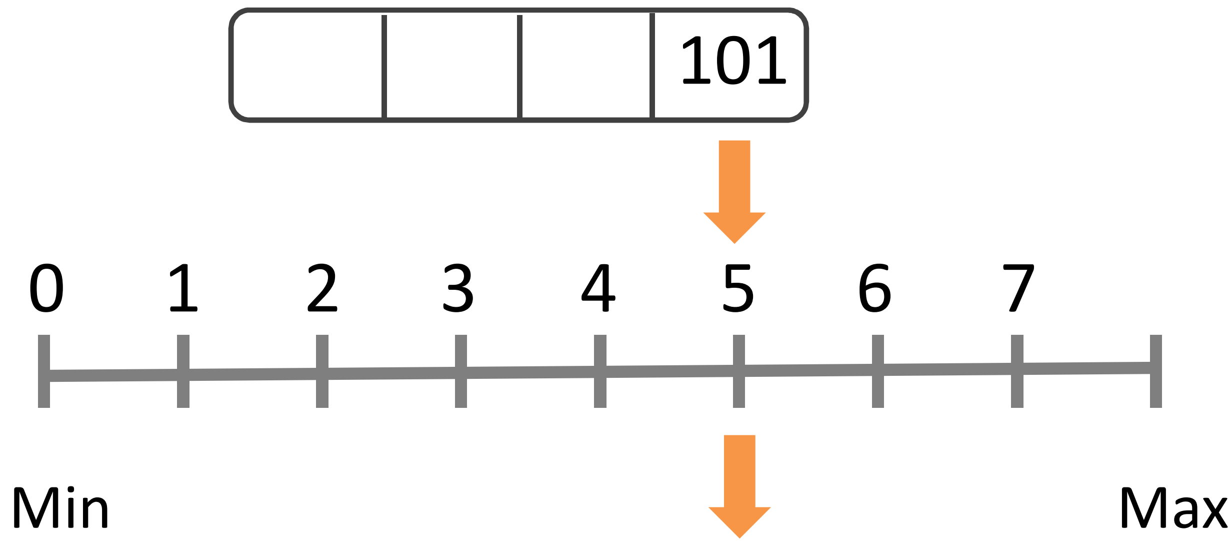
- Preprocessing finds min and max of the input sets and packs the data



- During execution, each thread unpacks the data and transforms the quantization level to data by applying a linear transformation

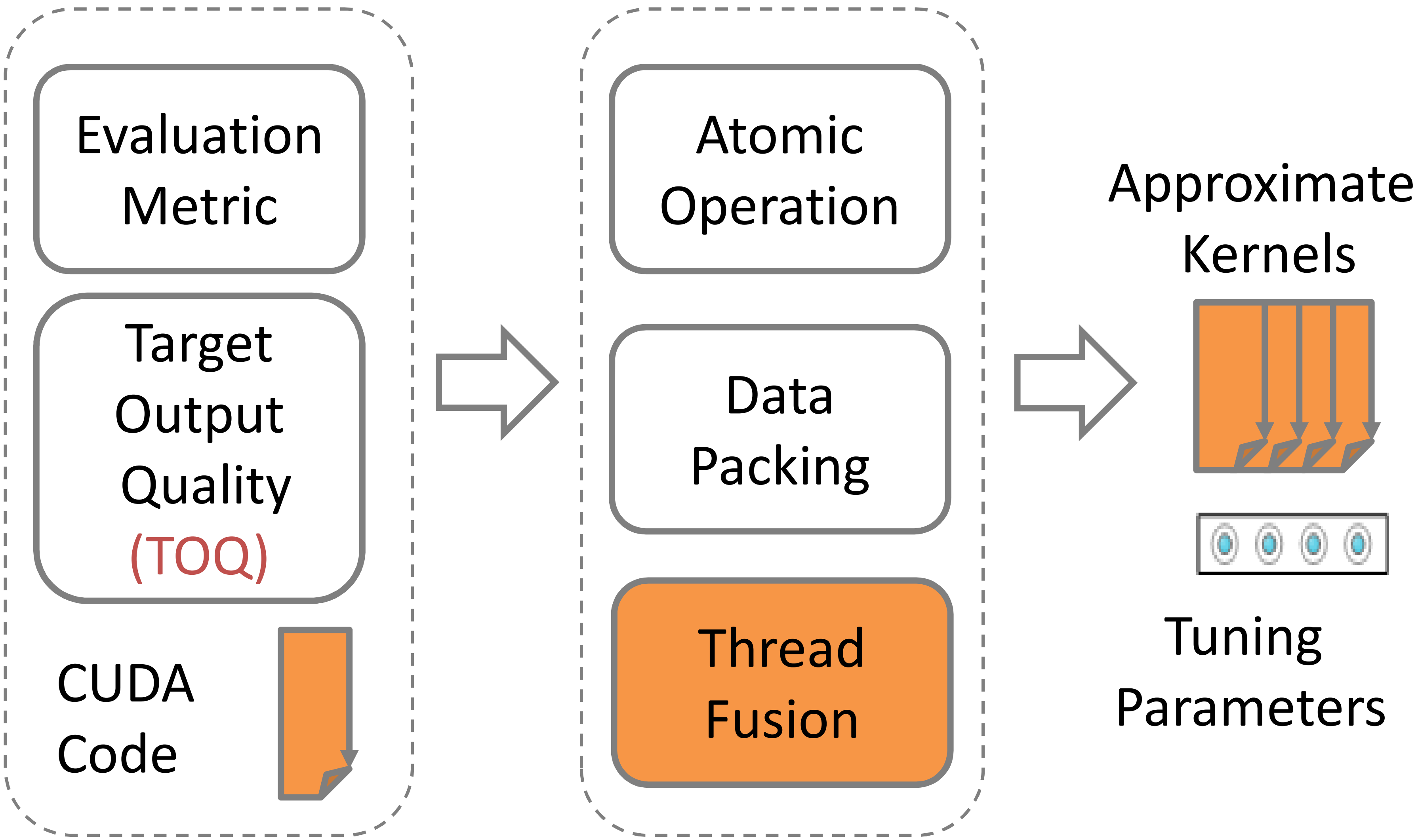
Quantization

- Preprocessing finds max and min of the input sets and packs the data

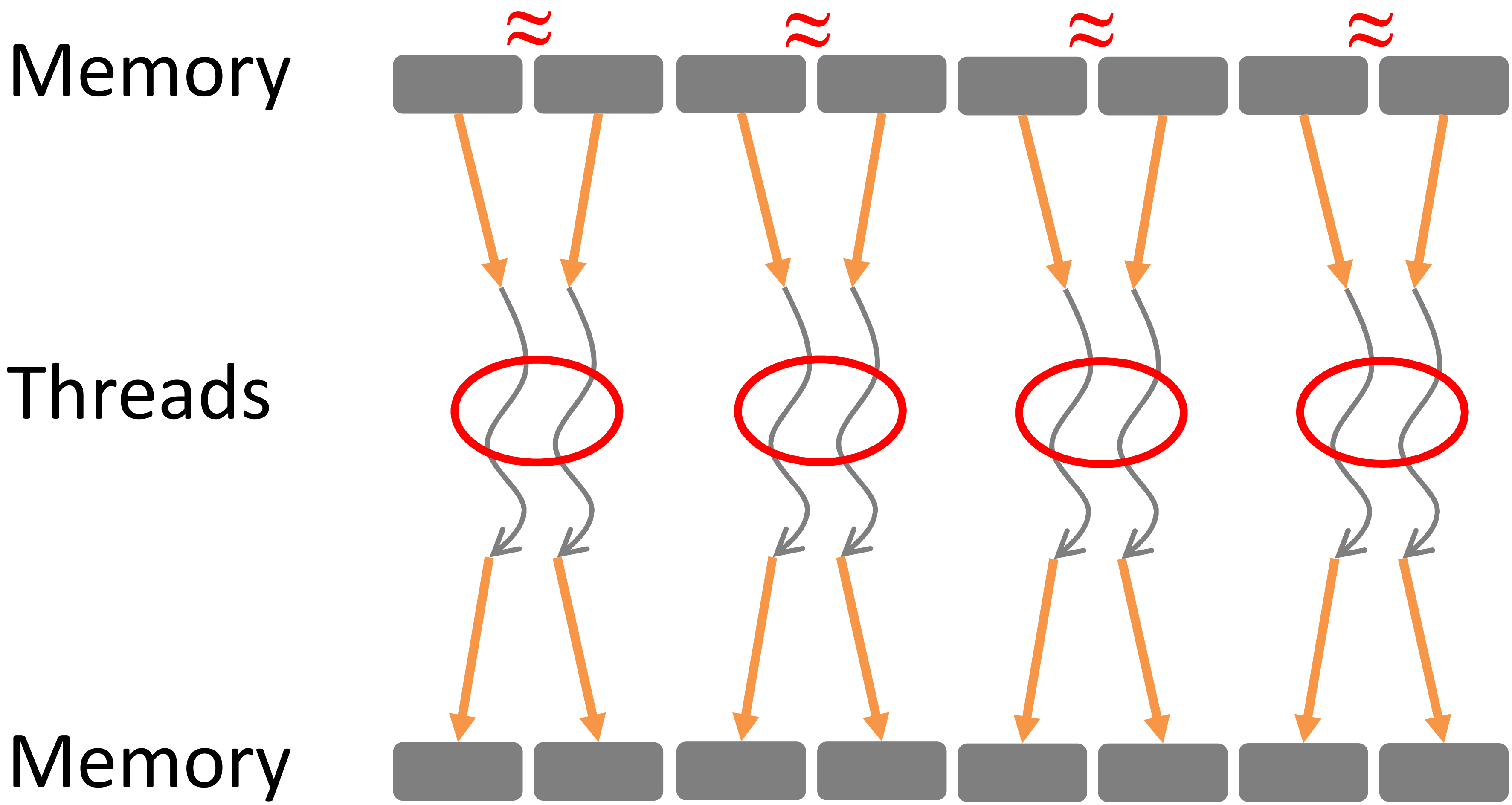


- During execution, each thread unpacks the data and transforms the quantization level to data by applying a linear transformation

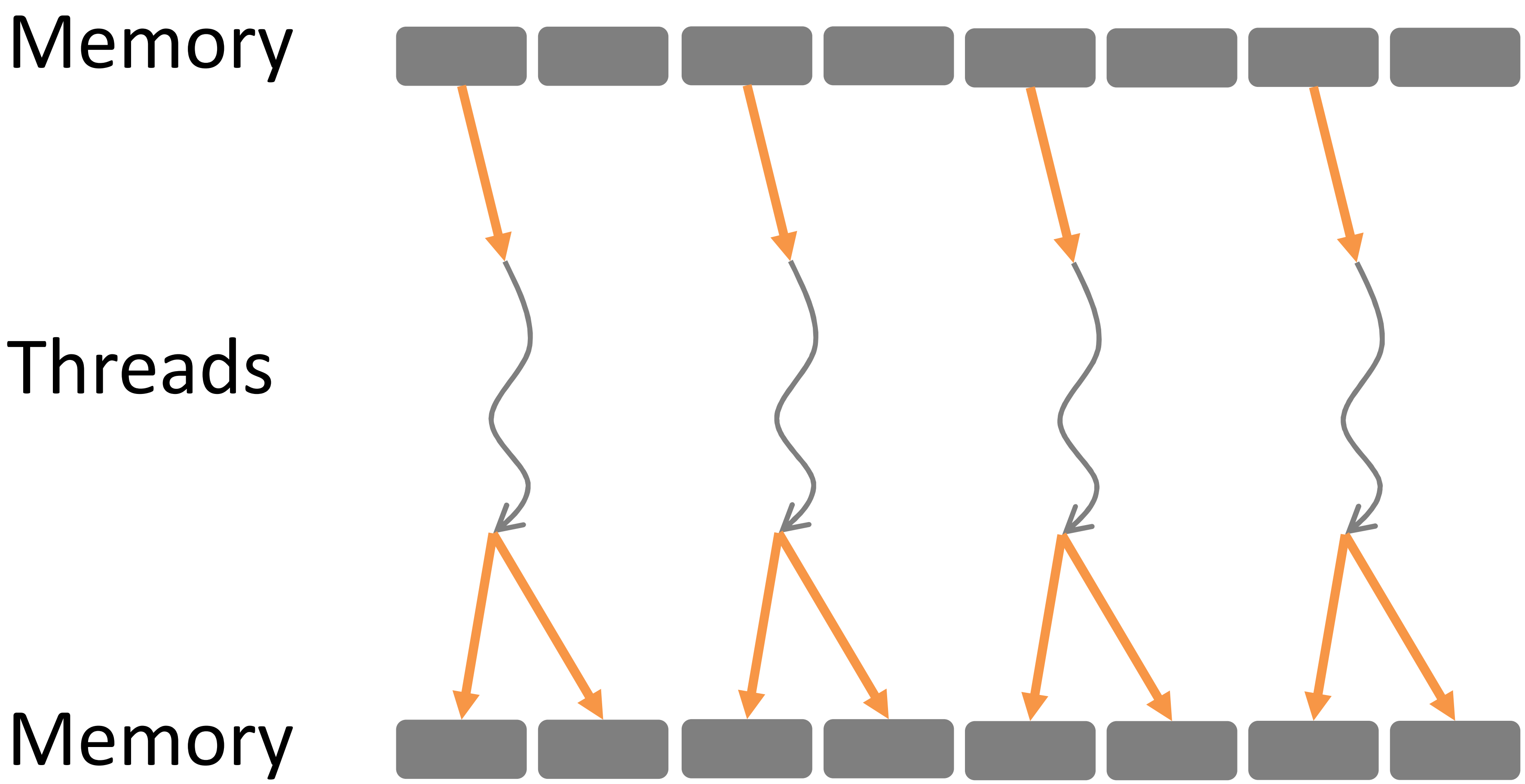
Approximation Methods



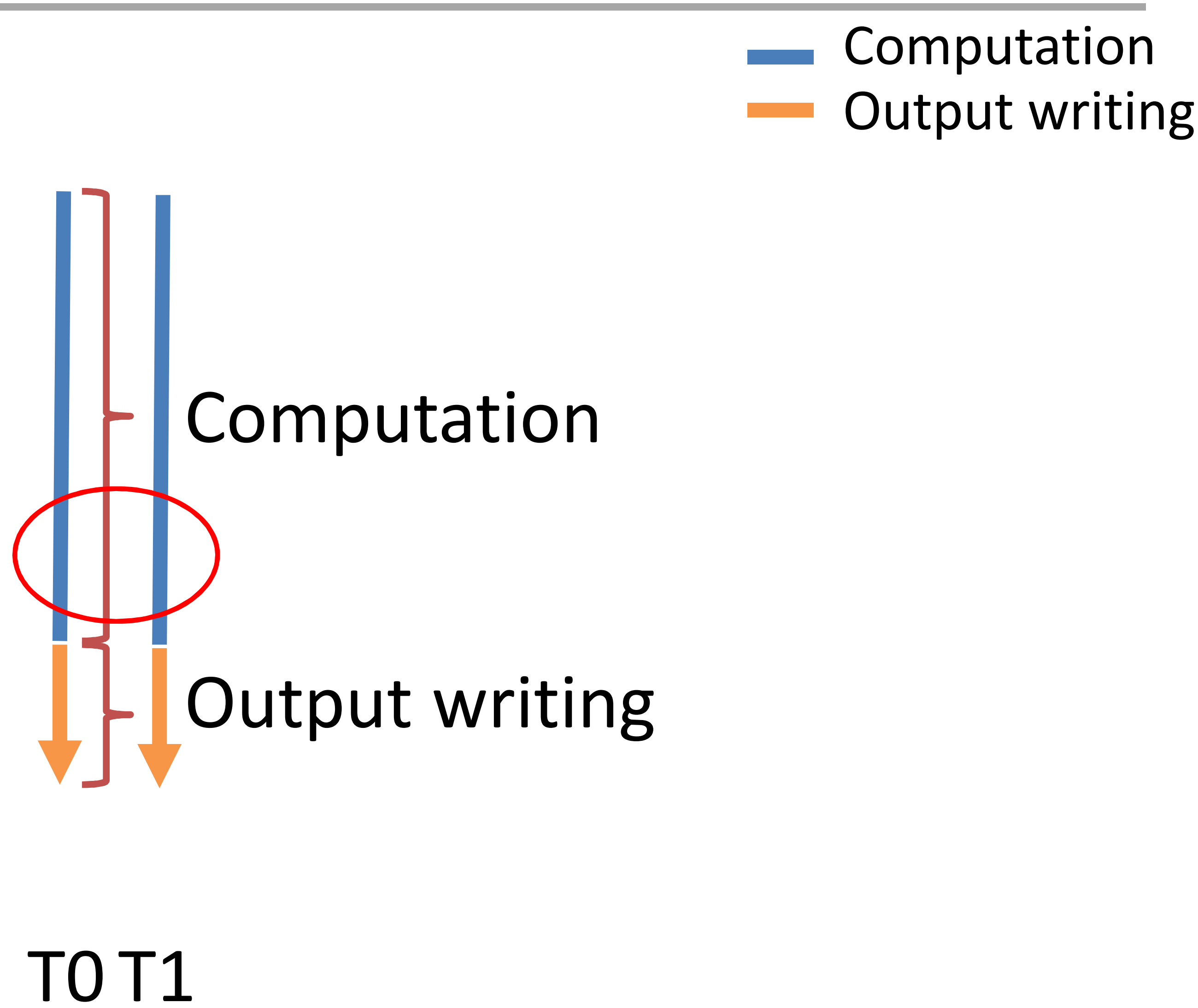
Thread Fusion



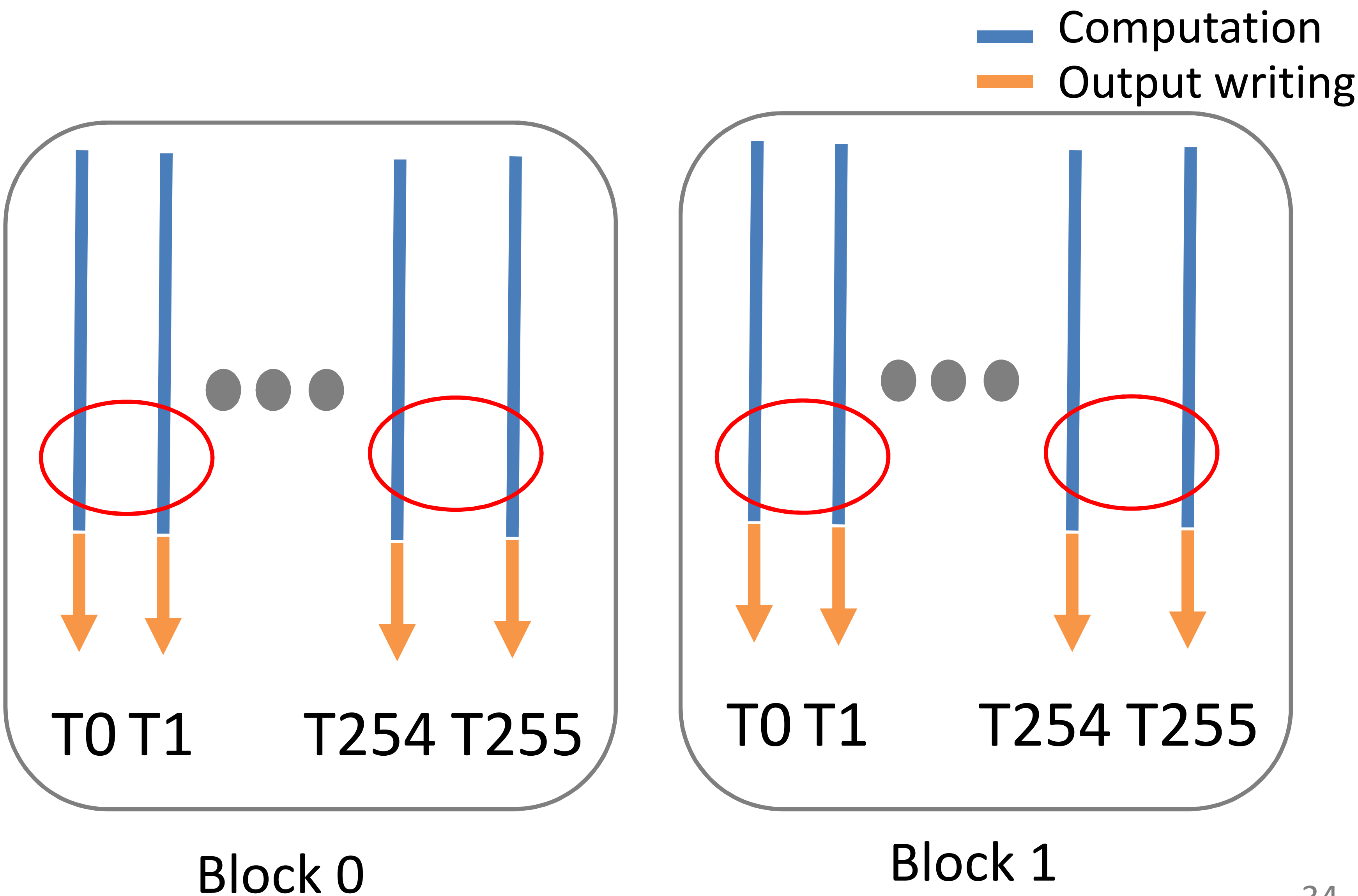
Thread Fusion



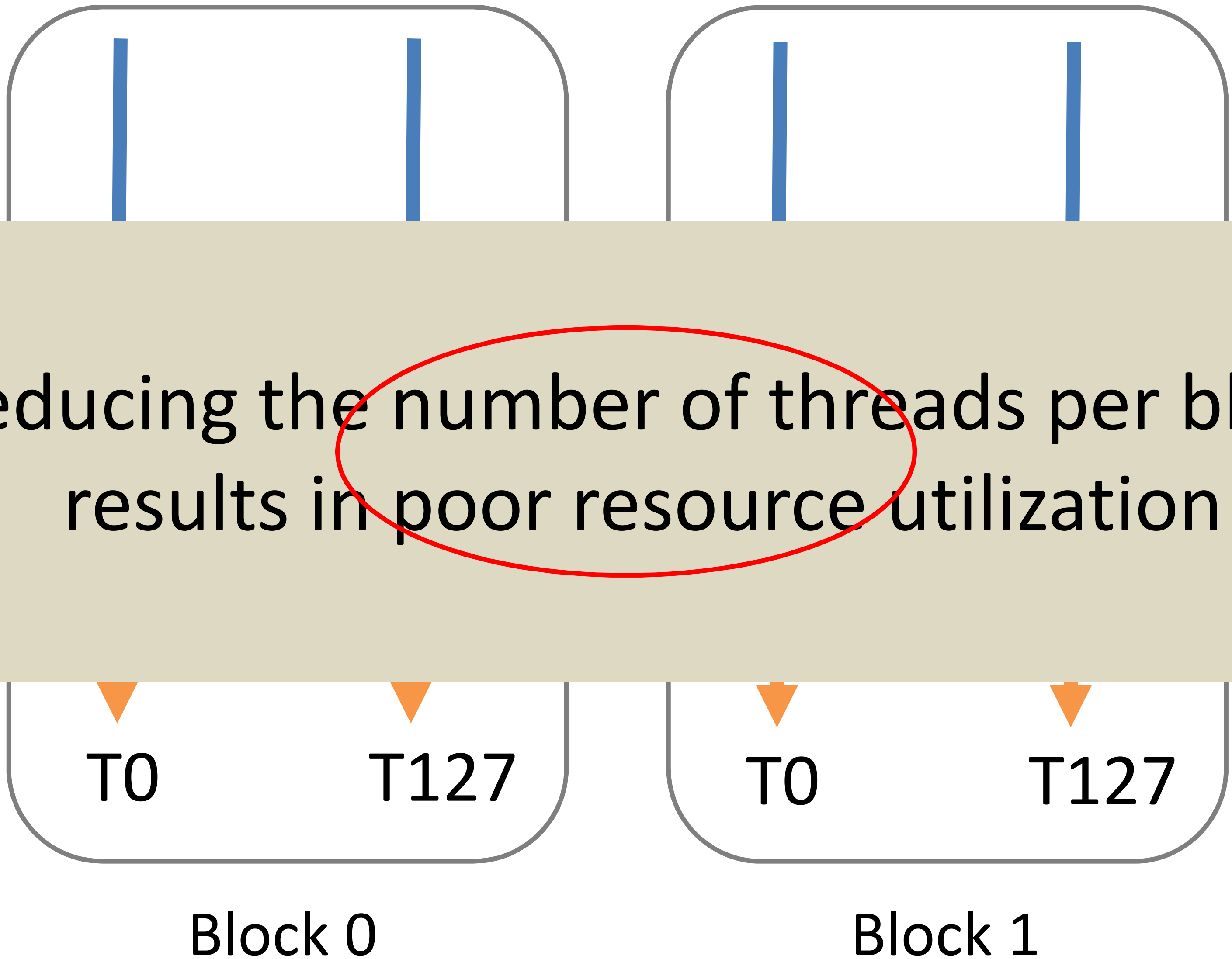
Thread Fusion



Thread Fusion

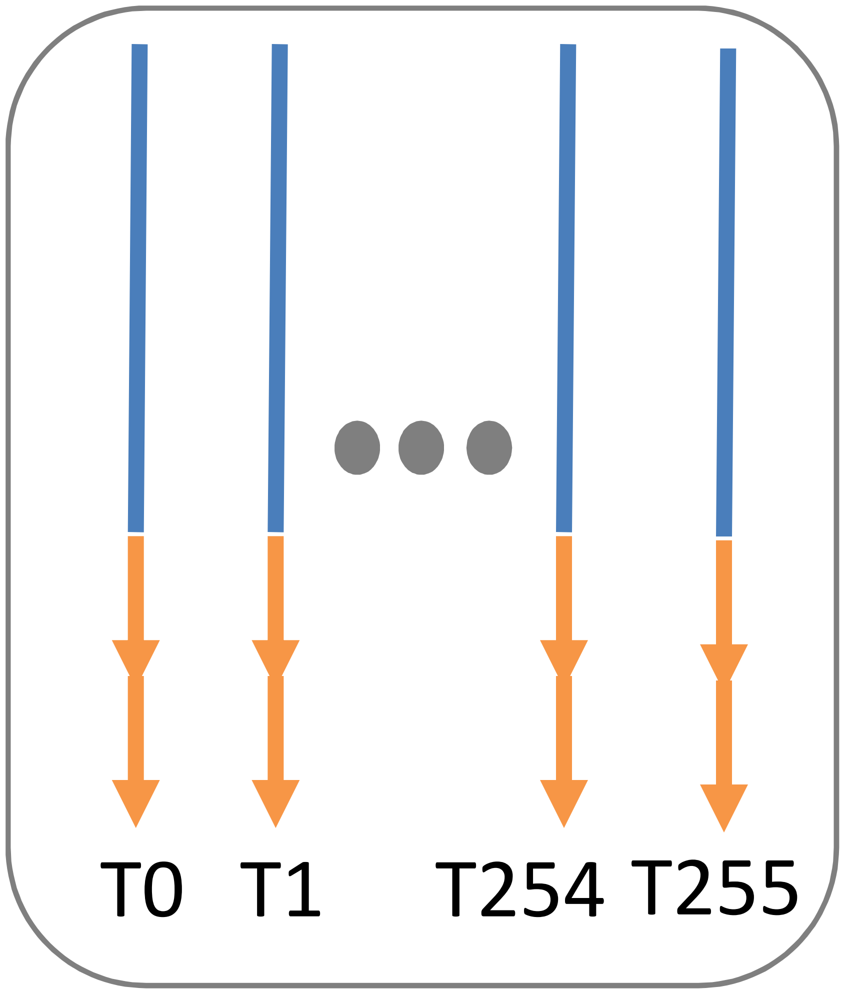


Thread Fusion



Reducing the number of threads per block results in poor resource utilization

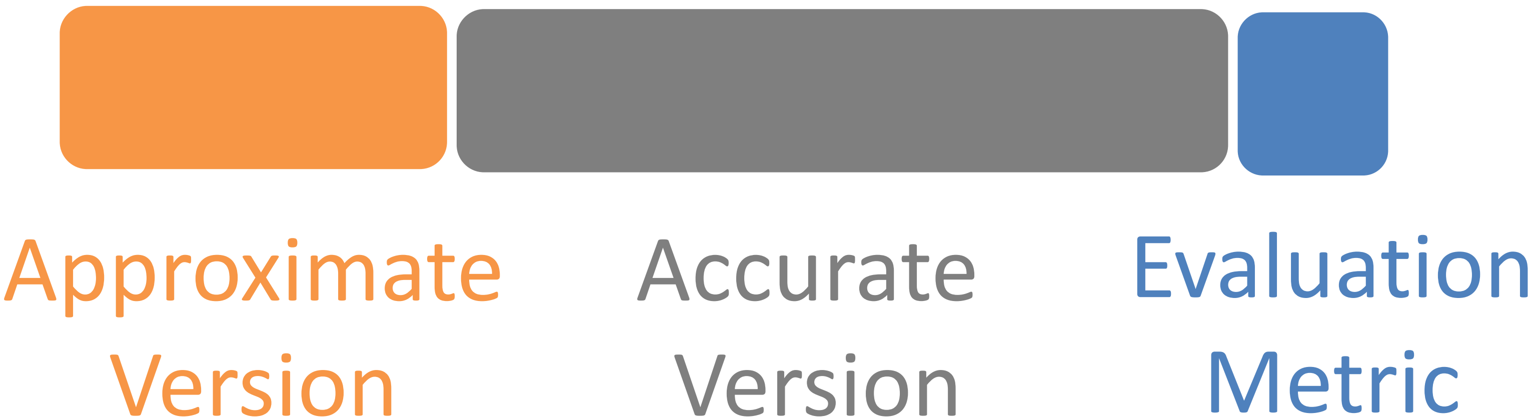
Block Fusion



Block 0 & 1 fused

Runtime

How to Compute Output Quality?



- High overhead
- Tuning should find a good enough approximate version as fast as possible
- Greedy algorithm

Tuning

TOQ = 90%

K(0,0)

Quality = 100%

Speedup = 1x

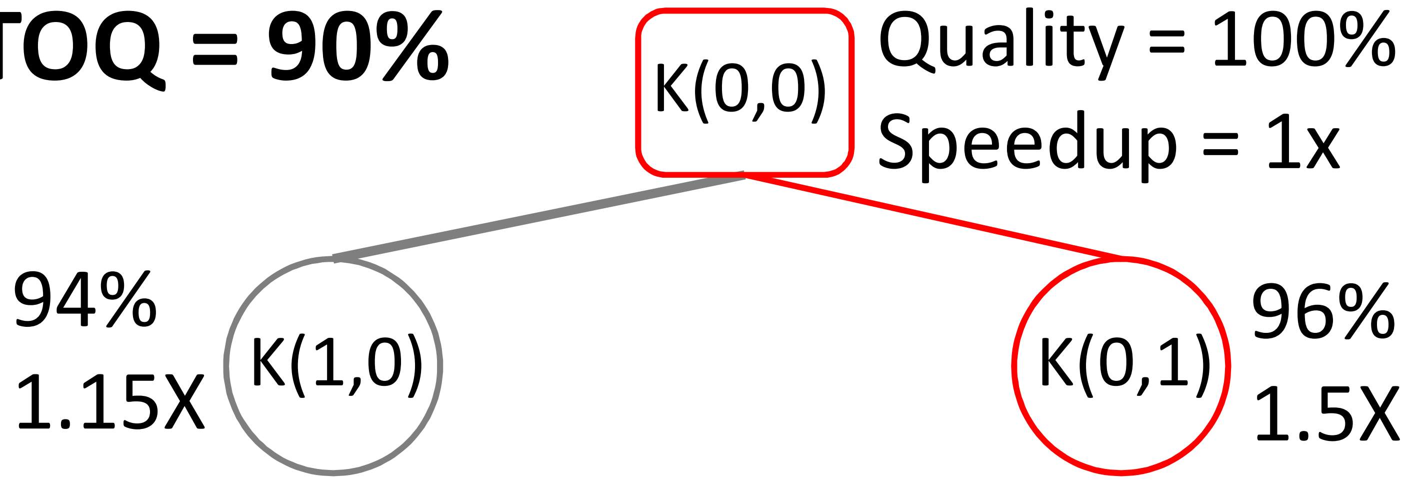
Tuning parameter of the First optimization

Tuning parameter of the Second optimization

K(x,y)

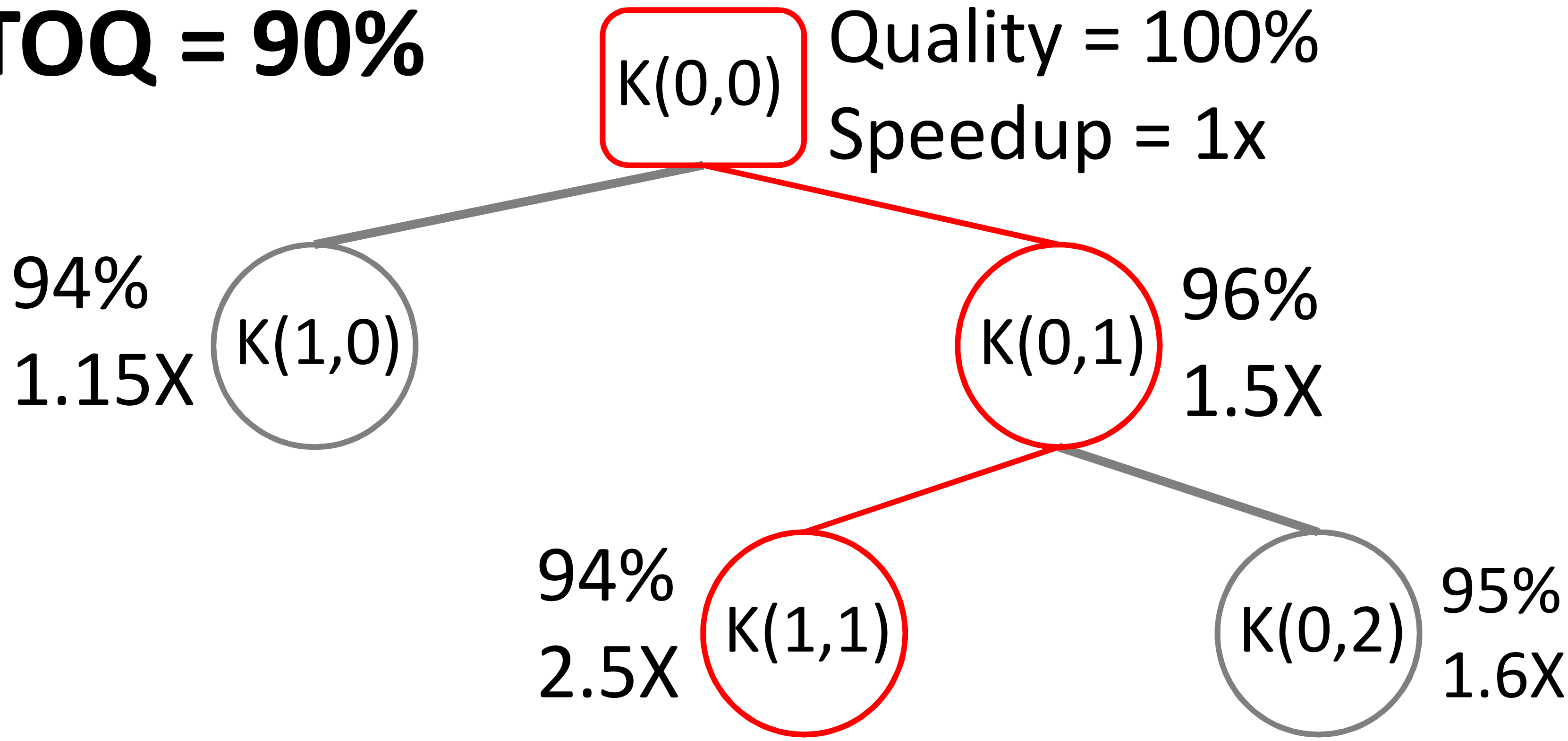
Tuning

TOQ = 90%



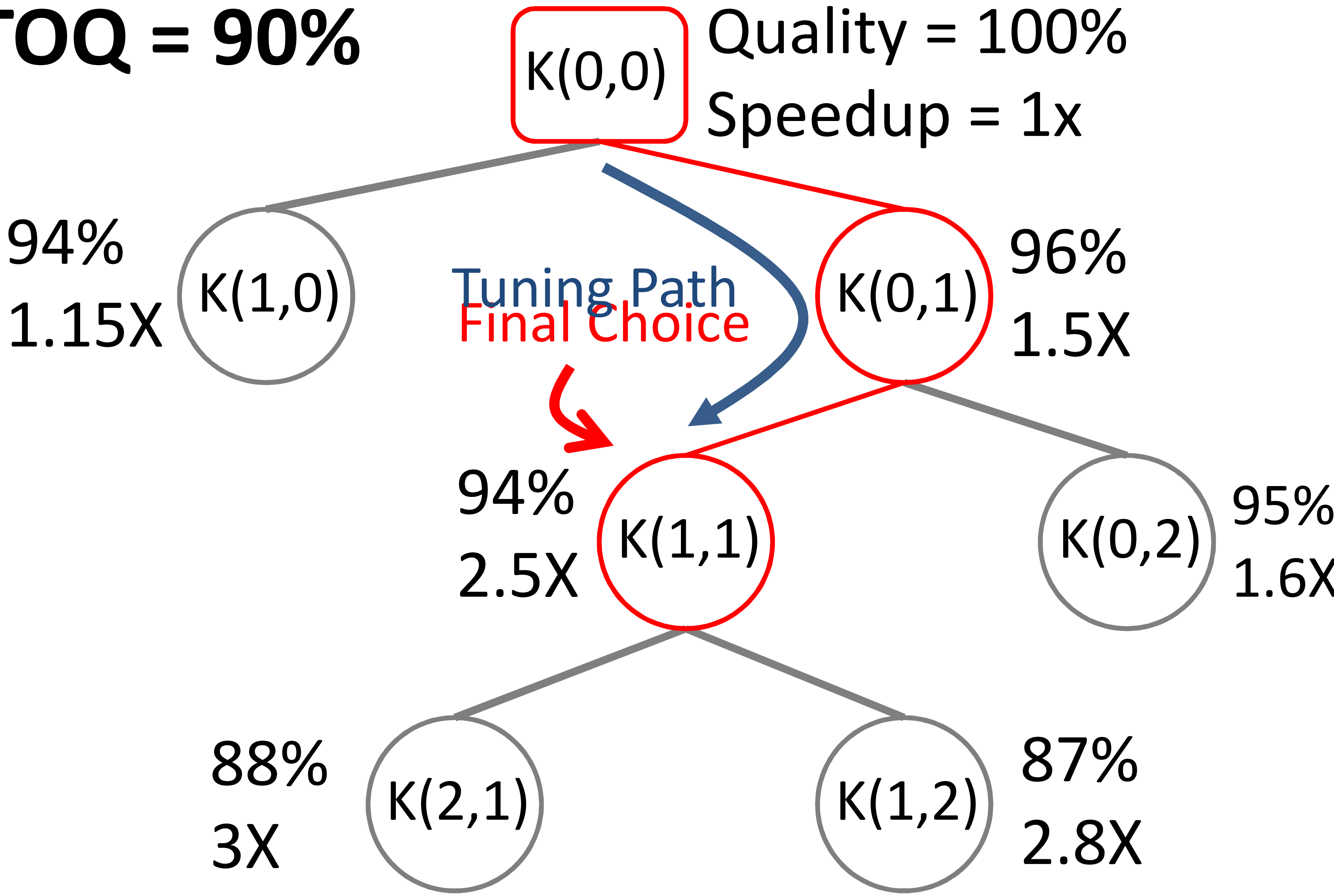
Tuning

TOQ = 90%



Tuning

TOQ = 90%

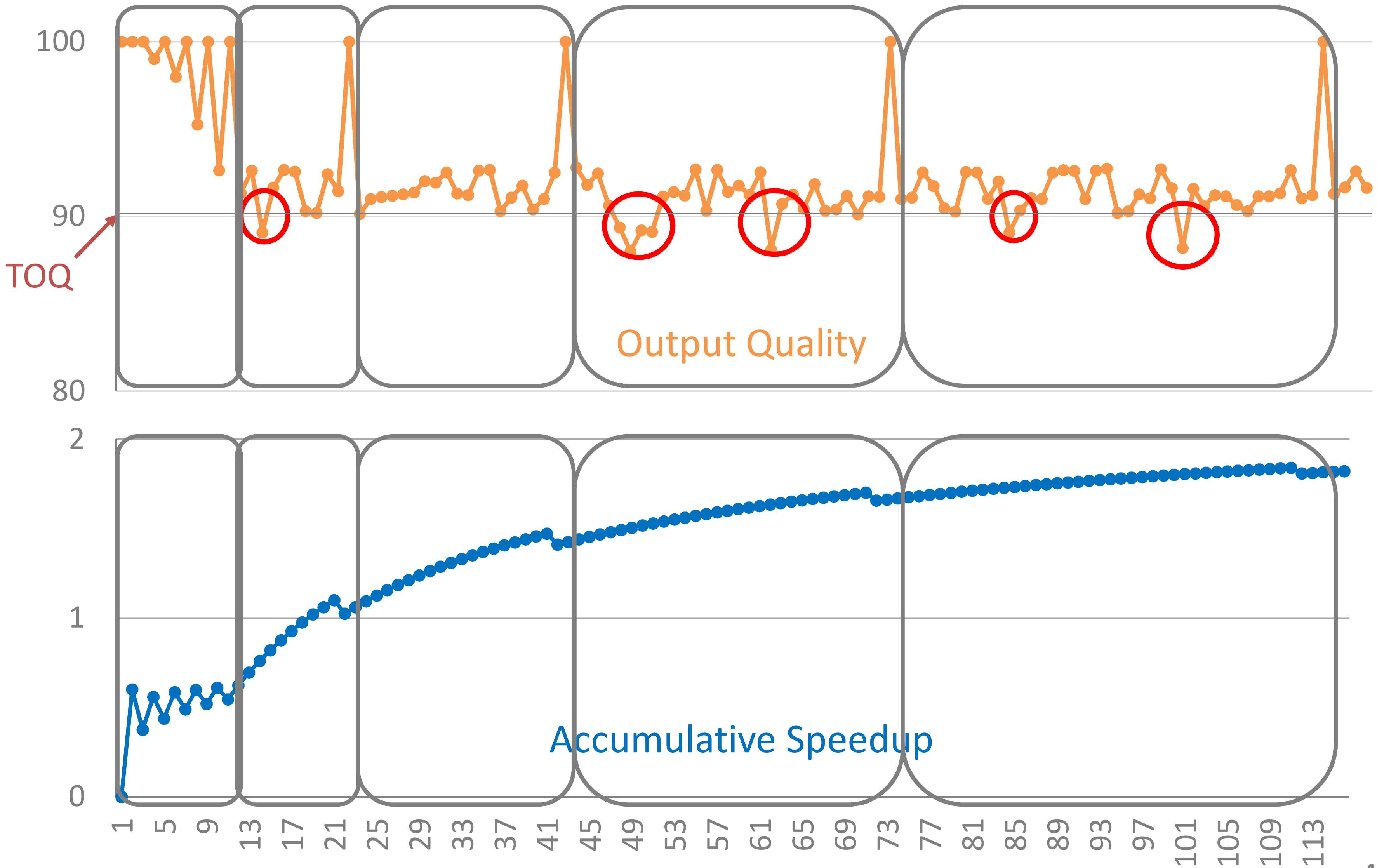


Evaluation

Experimental Setup

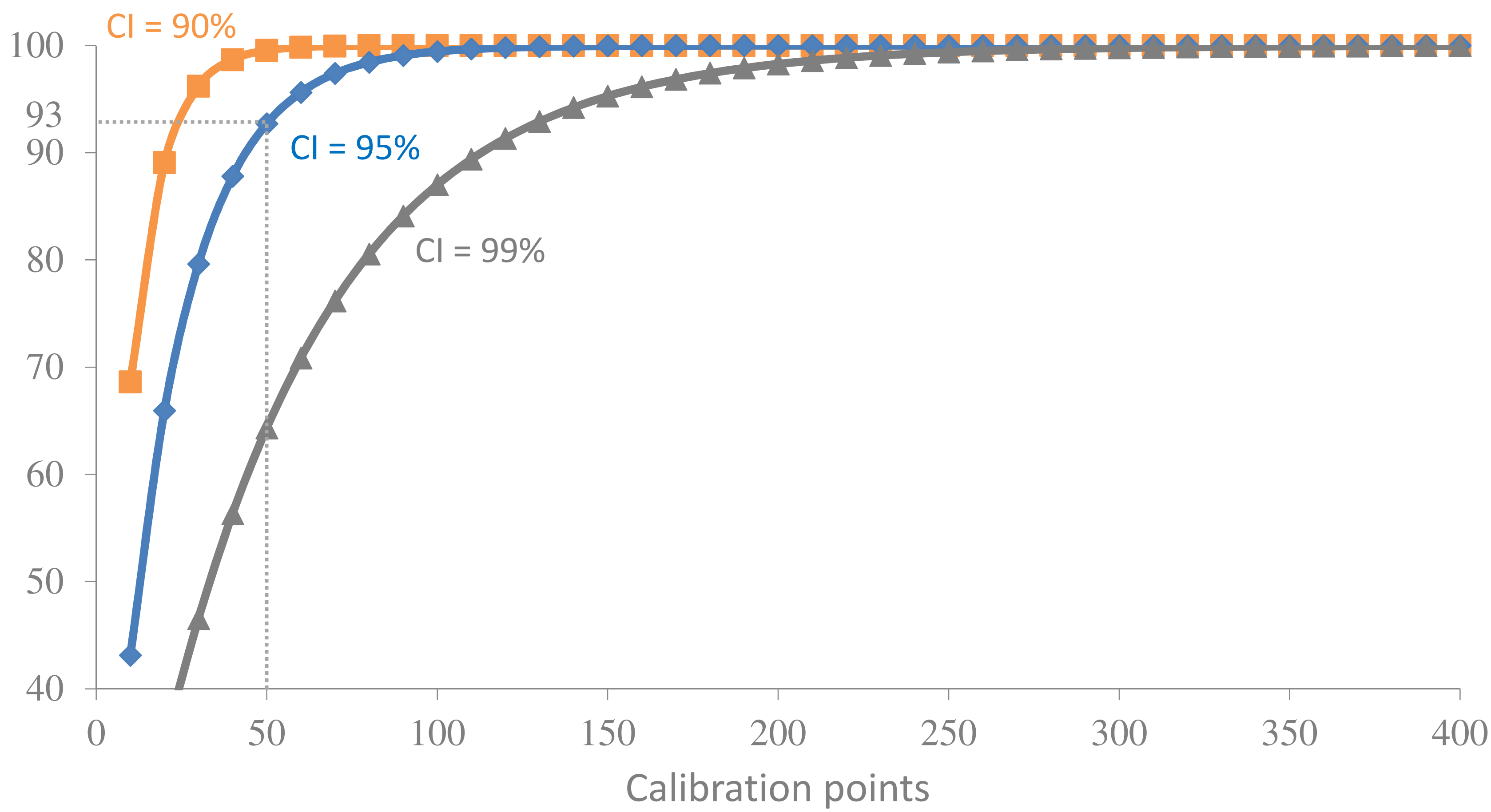
- Backend of Cetus compiler
- GPU
  - NVIDIA GTX 560
    - 2GB GDDR 5
- CPU
  - Intel Core i7
- Benchmarks
  - Image processing
  - Machine Learning

K-Means



100
90
80
TOQ
Output Quality
2
1
0
Accumulative Speedup
1
5
9
13
17
21
25
29
33
37
41
45
49
53
57
61
65
69
73
77
81
85
89
93
97
101
105
109
113

Confidence

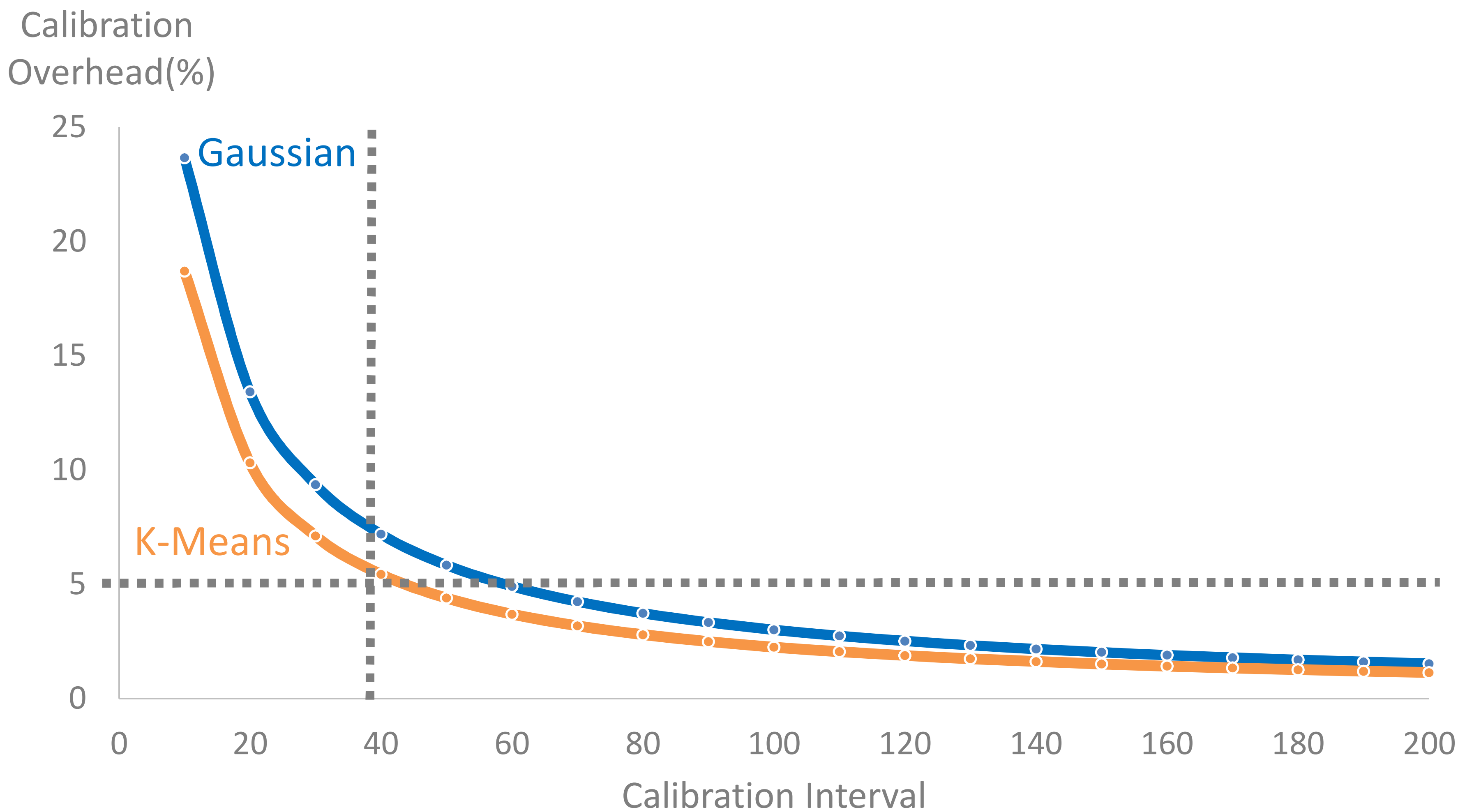


Beta Uniform Binomial

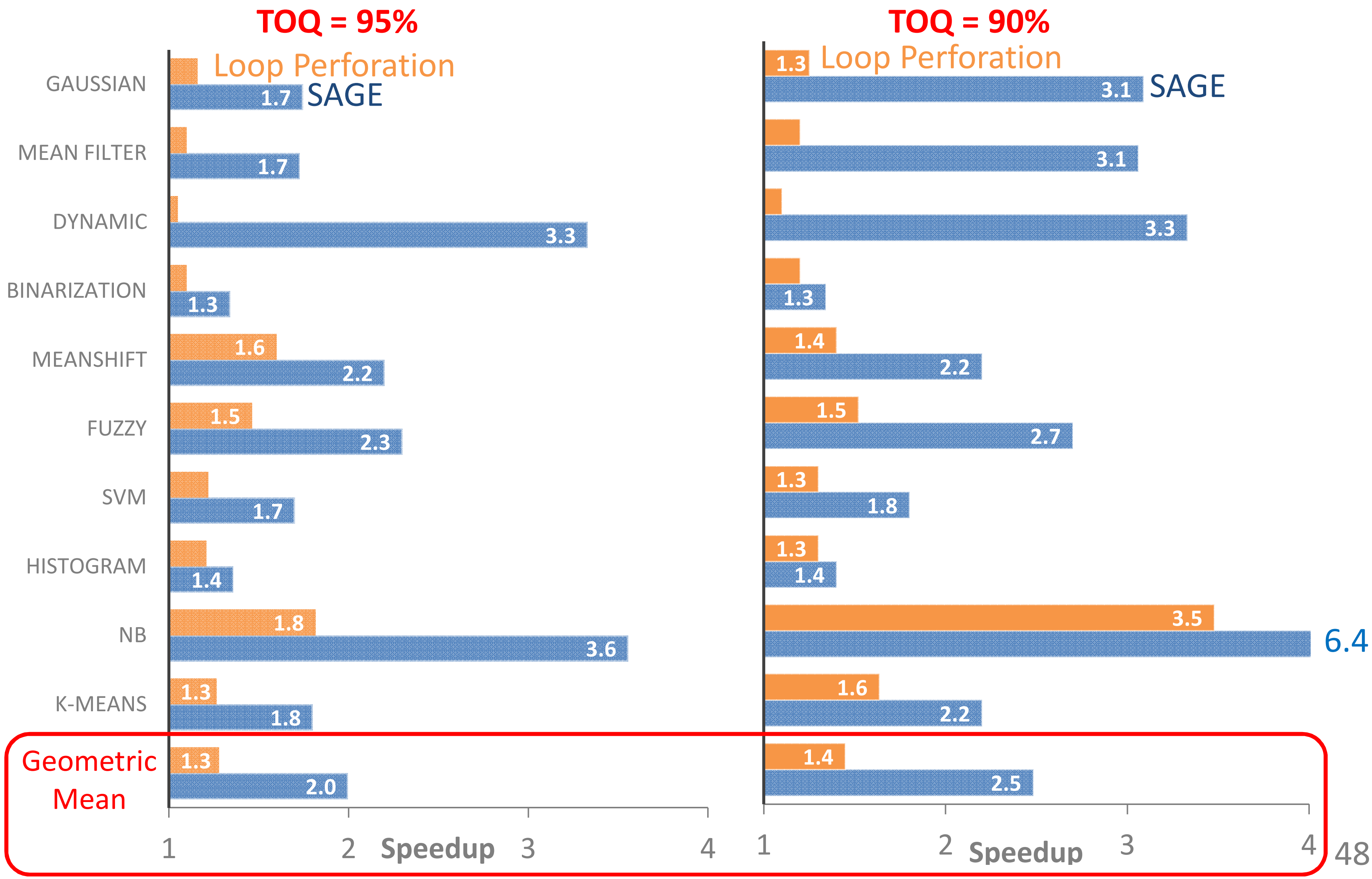
After checking 50 samples, we will be 93% confident that 95% of the outputs satisfy the TOQ threshold

$posterior \propto prior \times likelihood$

Calibration Overhead



Performance



Conclusion

- Automatic approximation is possible
- SAGE automatically generates approximate kernels with different parameters
- Runtime system uses tuning parameters to control the output quality during execution
- 2.5x speedup with less than 10% quality loss compared to the accurate execution

SAGE: Self-Tuning Approximation for Graphics Engines

Mehrzad Samadi[1], Janghaeng Lee[1], D. Anoushe Jamshidi[1], Amir Hormati[2], and Scott Mahlke[1]

University of Michigan[1], Google Inc.[2]

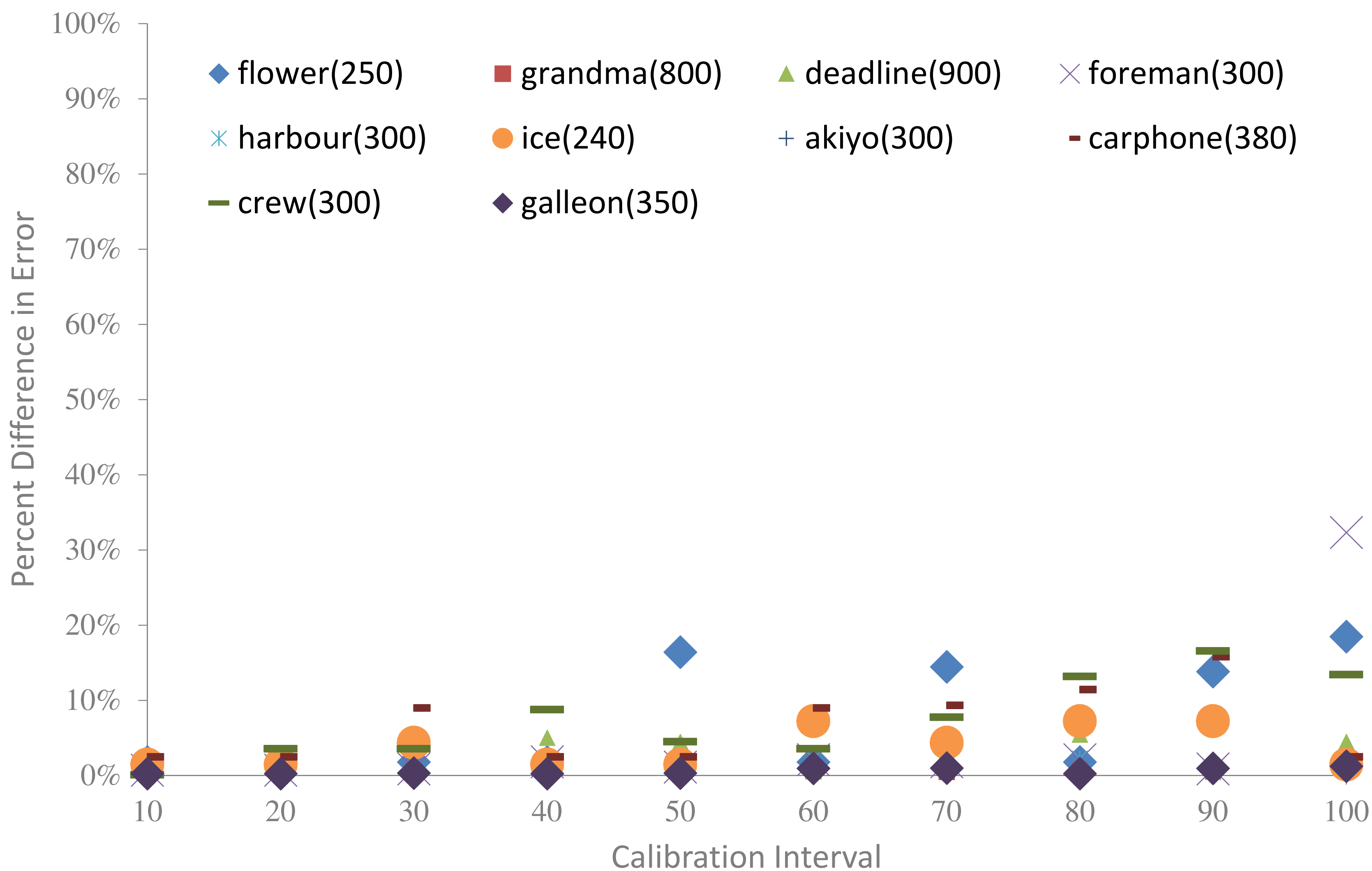
December 2013

University of Michigan
Electrical Engineering and Computer Science

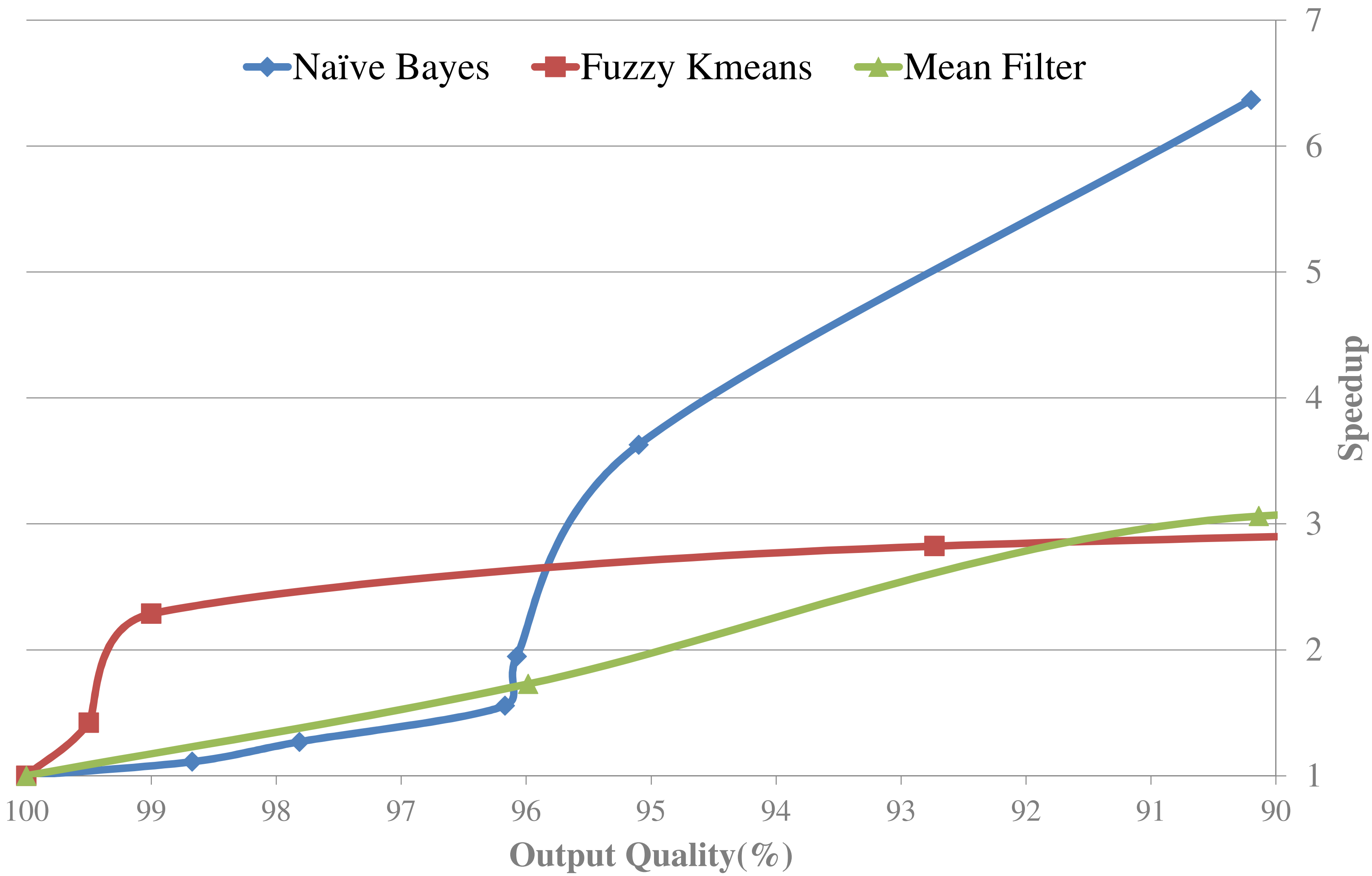
Compilers creating custom processors

Backup Slides

What Does Calibration Miss?



SAGE Can Control The Output Quality



Distribution of Errors