Visualization of Data Movements and Accesses

Til Mohr

High Performance Computing

RWTH AACHEN UNIVERSITY

Sum of Matrix Elements

Listing 1: Matrix Summation

```
let matrix = Matrix::random(2, 2);
let mut sum = 0;
for column in 0..2 {
  for row in 0..2 {
    sum += matrix.get(row, column);
  }
}
sum
```

Matrix:

	0	1
0	1	2
1	3	4

Sum of Matrix Elements

Listing 1: Matrix Summation

```
let matrix = Matrix::random(2, 2);
let mut sum = 0;
for column in 0..2 {
  for row in 0..2 {
    sum += matrix.get(row, column);
  }
}
sum
```

Matrix:

	0	1
0	1	2
1	3	4

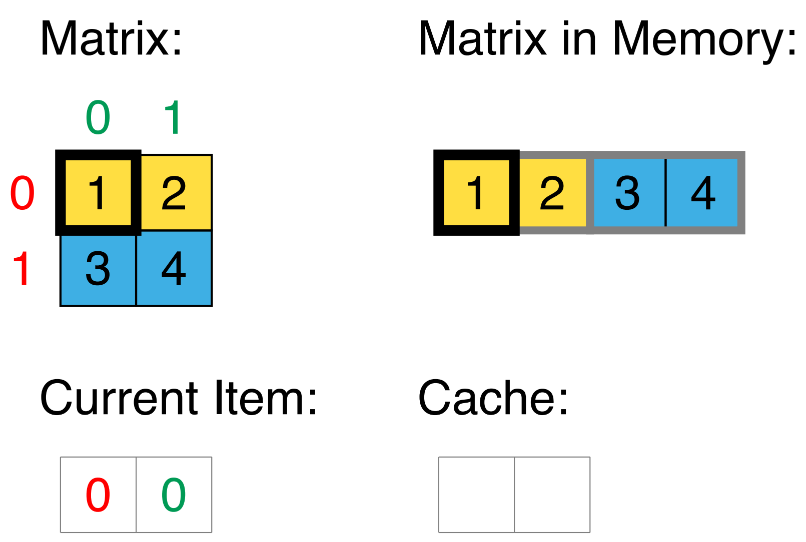
Matrix in Memory:

1	2	3	4

Sum of Matrix Elements

Listing 1: Matrix Summation

```
let matrix = Matrix::random(2, 2);
let mut sum = 0;
for column in 0..2 {
  for row in 0..2 {
    sum += matrix.get(row, column);
  }
}
sum
```

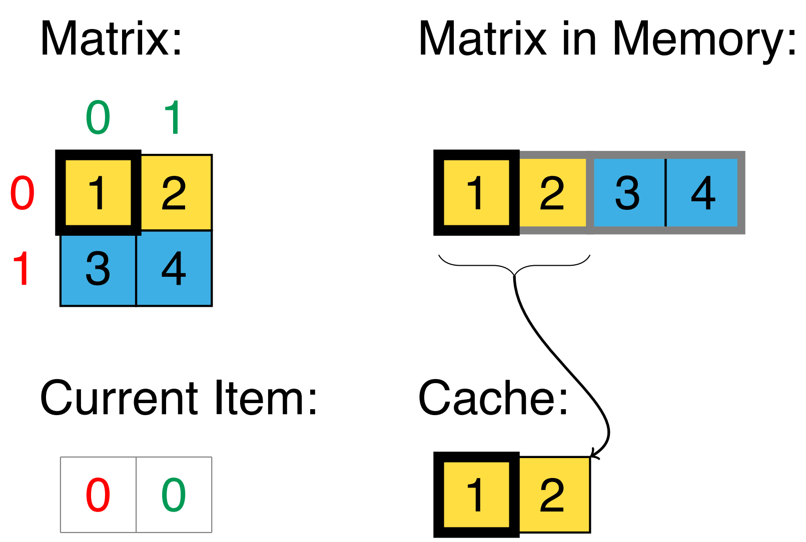


Number of cache misses: **0**

Sum of Matrix Elements

Listing 1: Matrix Summation

```
let matrix = Matrix::random(2, 2);
let mut sum = 0;
for column in 0..2 {
  for row in 0..2 {
    sum += matrix.get(row, column);
  }
}
sum
```

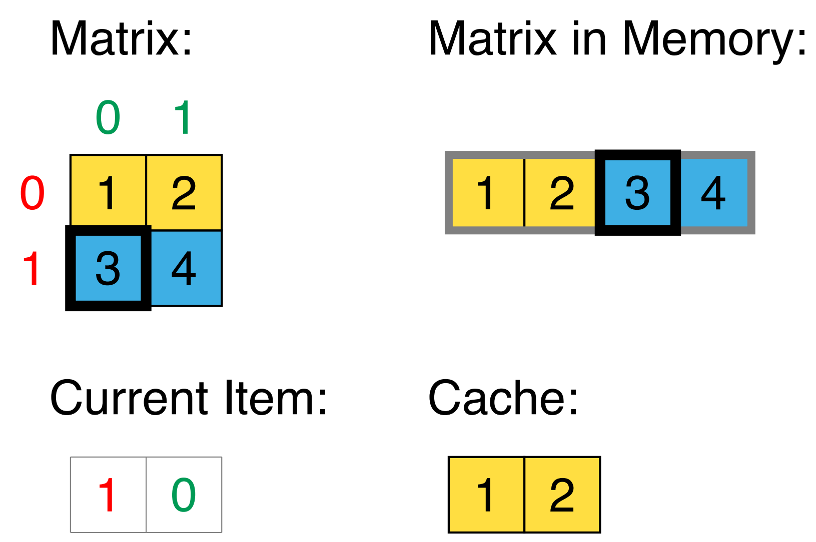


Number of cache misses: **1**

Sum of Matrix Elements

Listing 1: Matrix Summation

```
let matrix = Matrix::random(2, 2);
let mut sum = 0;
for column in 0..2 {
  for row in 0..2 {
    sum += matrix.get(row, column);
  }
}
sum
```

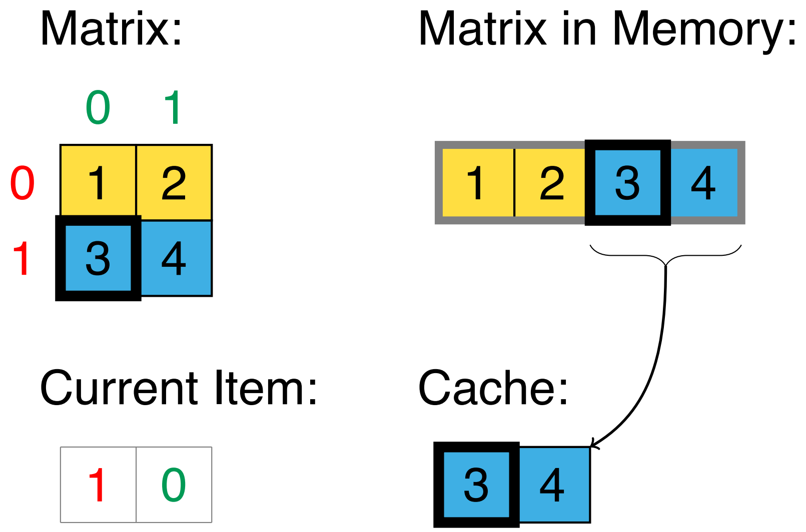


Number of cache misses: **1**

Sum of Matrix Elements

Listing 1: Matrix Summation

```
let matrix = Matrix::random(2, 2);
let mut sum = 0;
for column in 0..2 {
  for row in 0..2 {
    sum += matrix.get(row, column);
  }
}
sum
```

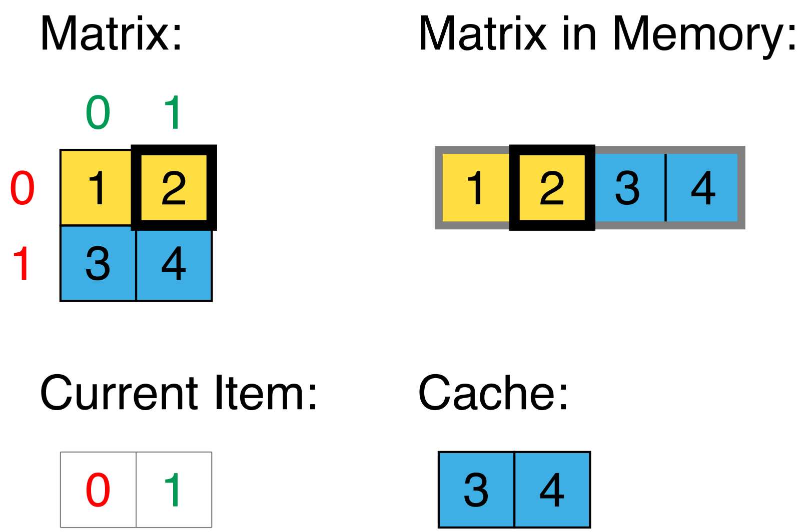


Number of cache misses: **2**

Sum of Matrix Elements

Listing 1: Matrix Summation

```
let matrix = Matrix::random(2, 2);
let mut sum = 0;
for column in 0..2 {
  for row in 0..2 {
    sum += matrix.get(row, column);
  }
}
sum
```

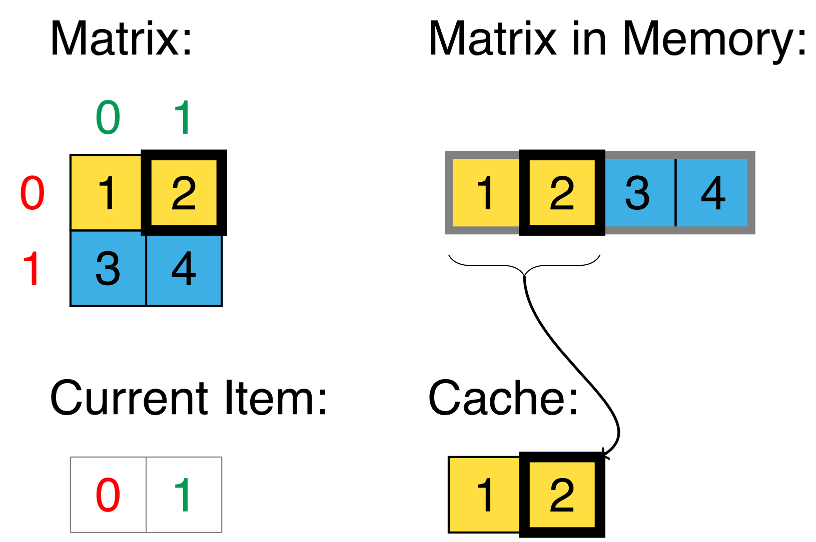


Number of cache misses: **2**

Sum of Matrix Elements

Listing 1: Matrix Summation

```
let matrix = Matrix::random(2, 2);
let mut sum = 0;
for column in 0..2 {
  for row in 0..2 {
    sum += matrix.get(row, column);
  }
}
sum
```



Number of cache misses: **3**

Sum of Matrix Elements

Listing 1: Matrix Summation

```
let matrix = Matrix::random(2, 2);
let mut sum = 0;
for column in 0..2 {
  for row in 0..2 {
    sum += matrix.get(row, column);
  }
}
sum
```

Matrix:

	0	1
0	1	2
1	3	4

Matrix in Memory:

1	2	3	4

Current Item:

1	1

Cache:

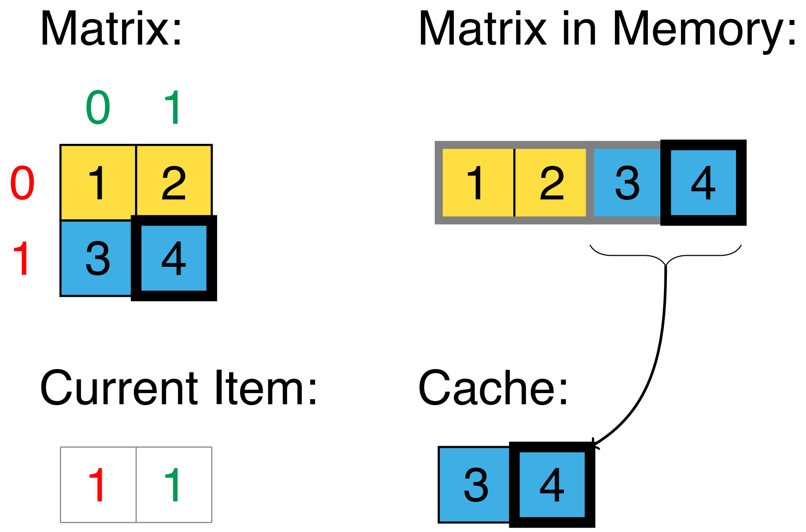
1	2

Number of cache misses: **3**

Sum of Matrix Elements

Listing 1: Matrix Summation

```
let matrix = Matrix::random(2, 2);
let mut sum = 0;
for column in 0..2 {
  for row in 0..2 {
    sum += matrix.get(row, column);
  }
}
sum
```

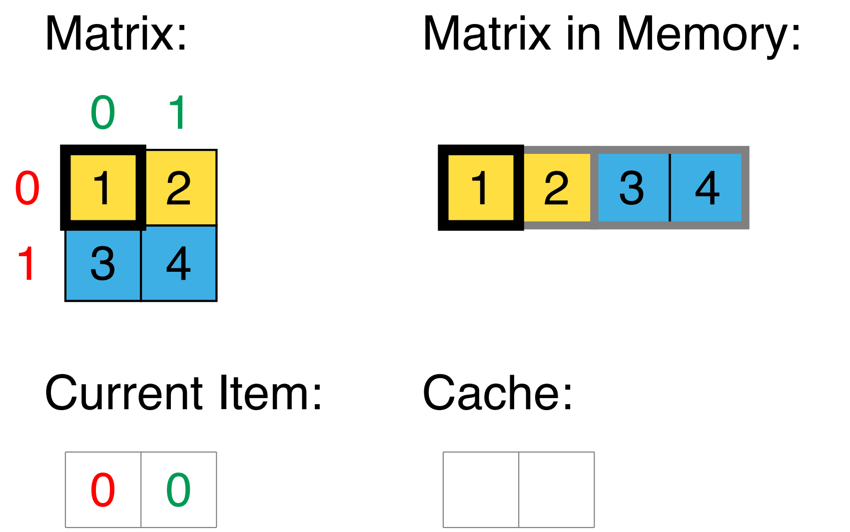


Number of cache misses: **4**

Sum of Matrix Elements - Reordered Loops

Listing 2: Matrix Summation

```
let matrix = Matrix::random(2, 2);
let mut sum = 0;
for row in 0..2 {
  for column in 0..2 {
    sum += matrix.get(row, column);
  }
}
sum
```

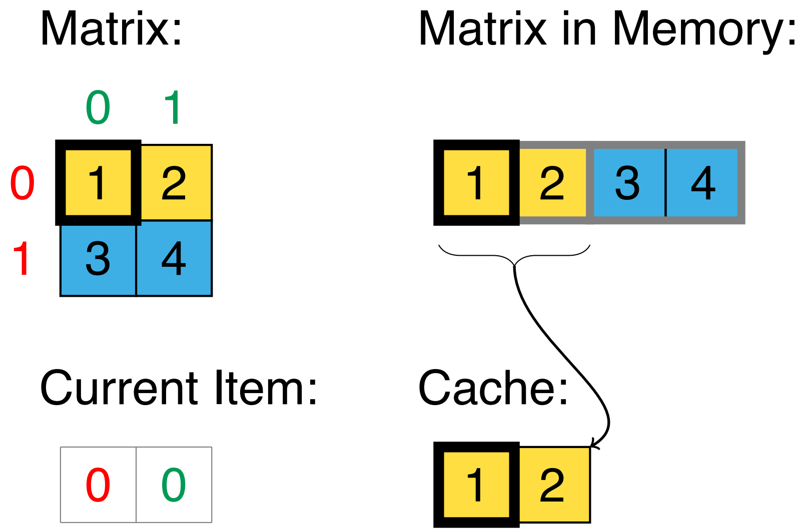


Number of cache misses: **0**

Sum of Matrix Elements - Reordered Loops

Listing 2: Matrix Summation

```
let matrix = Matrix::random(2, 2);
let mut sum = 0;
for row in 0..2 {
  for column in 0..2 {
    sum += matrix.get(row, column);
  }
}
sum
```

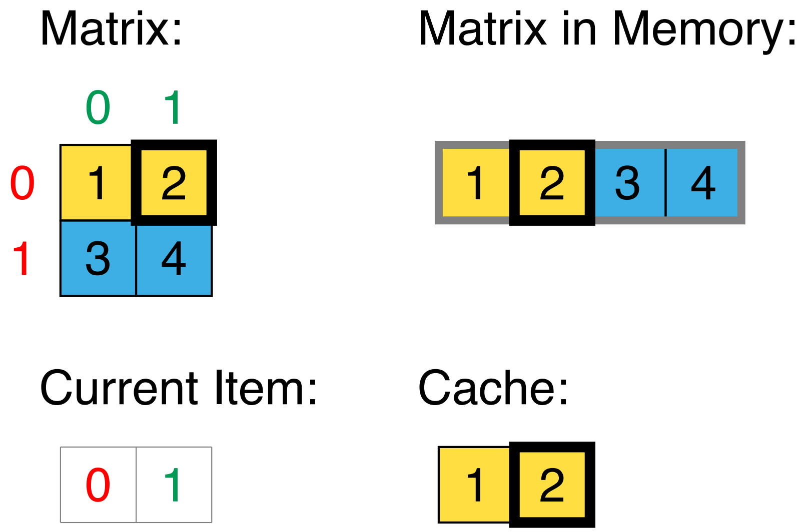


Number of cache misses: **1**

Sum of Matrix Elements - Reordered Loops

Listing 2: Matrix Summation

```
let matrix = Matrix::random(2, 2);
let mut sum = 0;
for row in 0..2 {
  for column in 0..2 {
    sum += matrix.get(row, column);
  }
}
sum
```

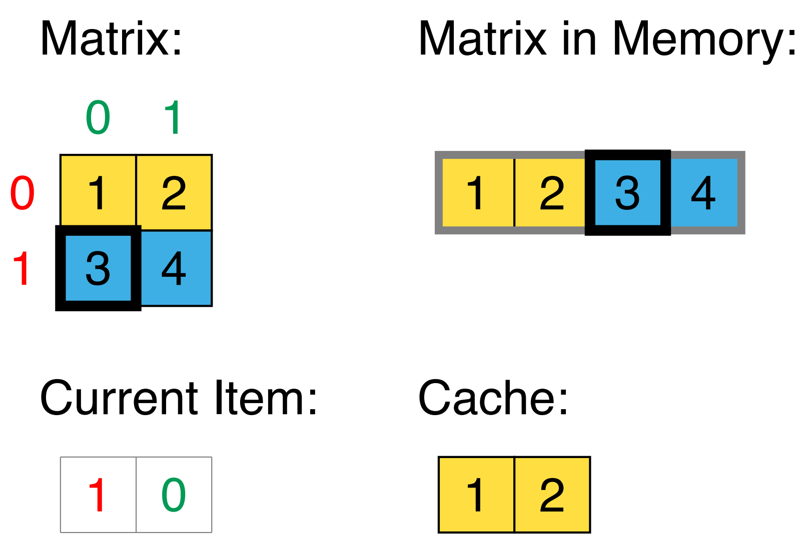


Number of cache misses: **1**

Sum of Matrix Elements - Reordered Loops

Listing 2: Matrix Summation

```
let matrix = Matrix::random(2, 2);
let mut sum = 0;
for row in 0..2 {
  for column in 0..2 {
    sum += matrix.get(row, column);
  }
}
sum
```

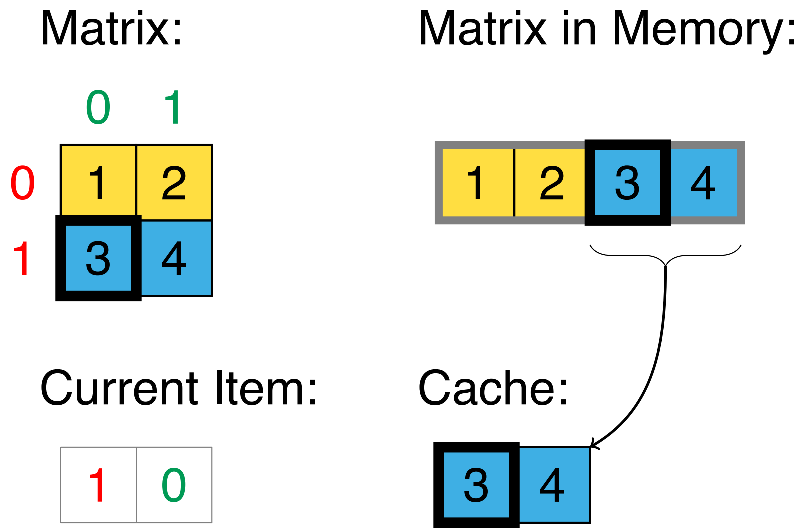


Number of cache misses: **1**

Sum of Matrix Elements - Reordered Loops

Listing 2: Matrix Summation

```
let matrix = Matrix::random(2, 2);
let mut sum = 0;
for row in 0..2 {
  for column in 0..2 {
    sum += matrix.get(row, column);
  }
}
sum
```

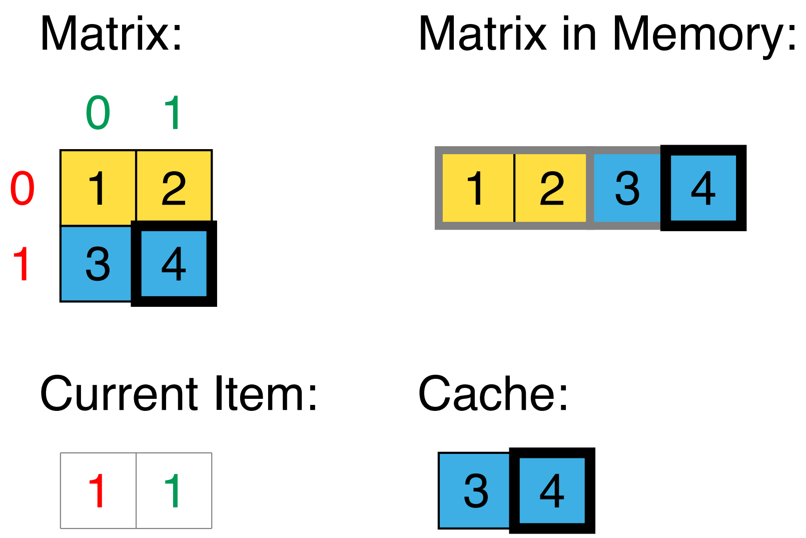


Number of cache misses: **2**

Sum of Matrix Elements - Reordered Loops

Listing 2: Matrix Summation

```
let matrix = Matrix::random(2, 2);
let mut sum = 0;
for row in 0..2 {
  for column in 0..2 {
    sum += matrix.get(row, column);
  }
}
sum
```



Number of cache misses: **2**

Outline

- Memory-Related Performance Problems
  - Data Locality
  - Processor-Memory Performance Gap
- Overview of the Optimization Workflow
- Data Gathering Approaches
- Visualization Techniques
- Specific Optimization Tool
- Conclusion

Memory-Related Performance Problems | Data Locality

$$t_{avg} = p \cdot t_c + (1 - p) \cdot t_m \tag{1}$$

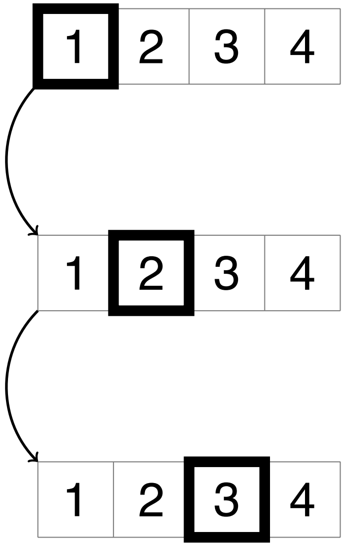
t_{avg} : average access time
p : cache hit percentage
t_c : cache access time
t_m : memory access time

$$t_c \ll t_m \tag{2}$$

Memory-Related Performance Problems I Data Locality

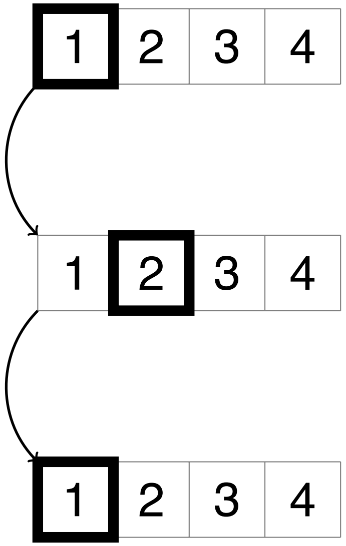
Spacial Locality

- Data that is referenced spatially close together is likely to be accessed in the near future

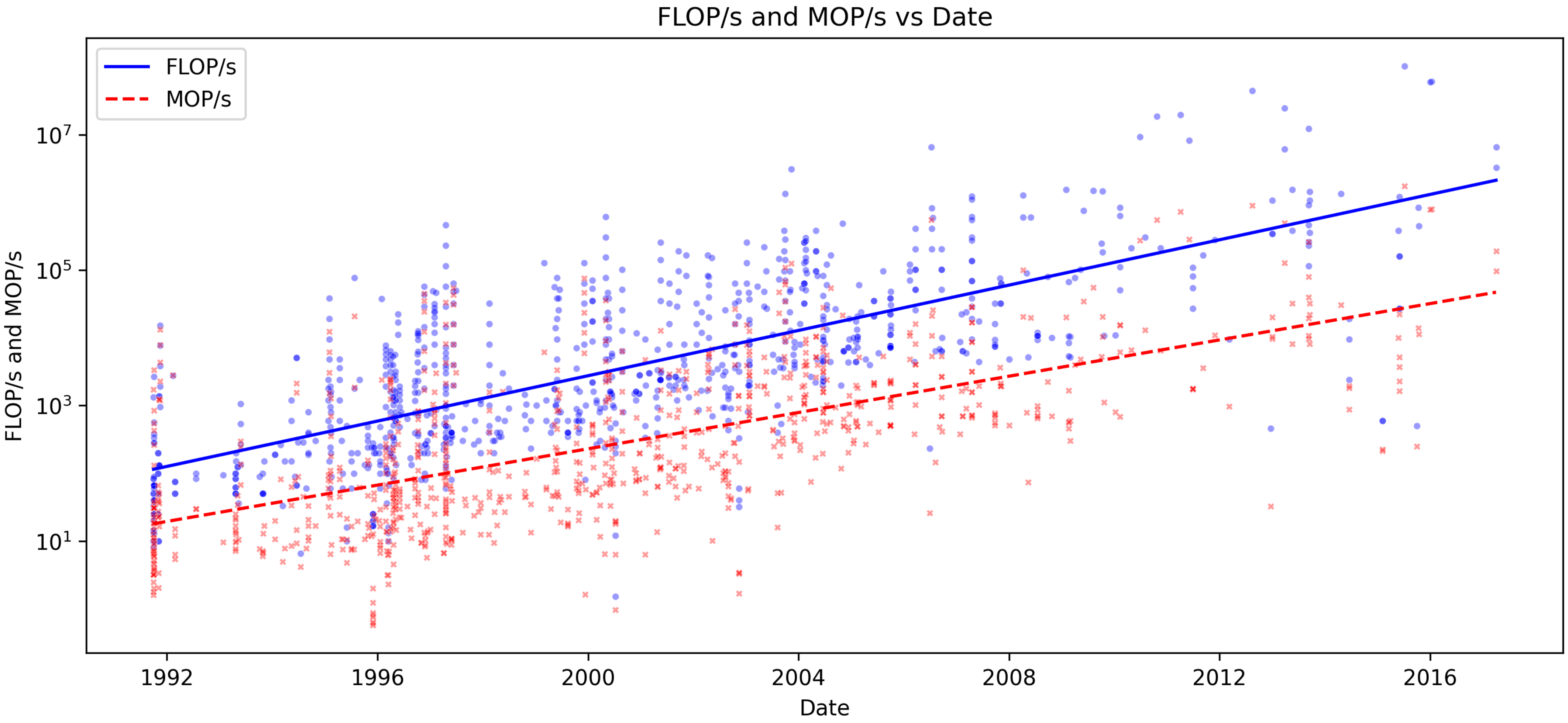


Temporal Locality

- Data that is referenced in the near past is likely to be accessed in the near future



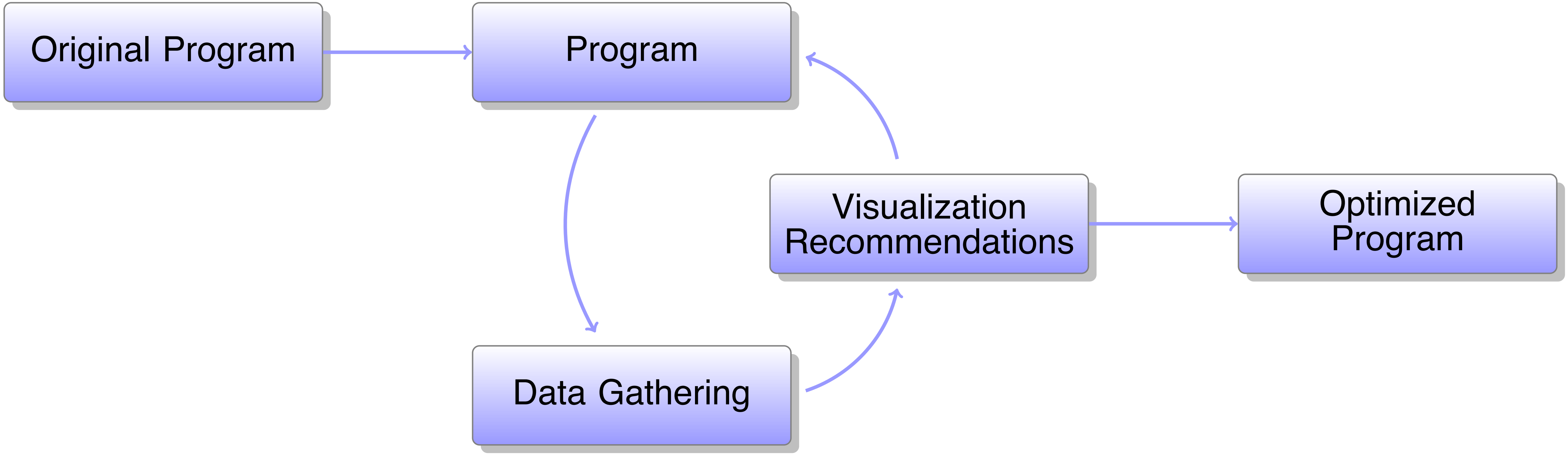
Memory-Related Performance Problems I Processor-Memory Performance Gap



Data: *https://www.cs.virginia.edu/stream/*

Overview of the Optimization Workflow

Visualization-Guided Optimization



Data Gathering Approaches

Goal: Acquire Memory-Related Data for Visualization

- Data Accesses
  - Memory locations / variables
  - Frequencies
- Data access patterns
  - Nested loops
- Cache performance
  - Hit/miss rates
  - Utilization
  - Amount of data transfer in between different cache levels and main memory

Data Gathering Approaches I Dynamic Analyis

Run Program and Capture Memory-Related Information

- Hardware counters
  - Counts cache hits/misses
- Tracing / profiling
- Store source code references alongside memory-related information

☺ Very accurate
- Real program data
- Actual physical hardware

☹ Can be very slow

☹ Possible large overhead for very granular data

☹ Cannot easily analyze just parts of the program

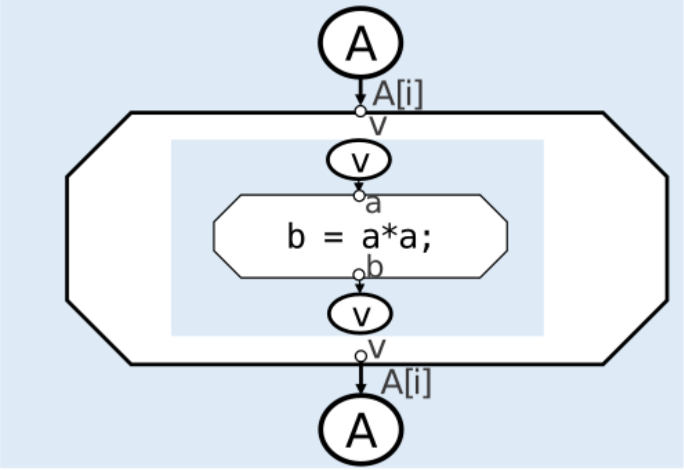
Analyze the Programs Source Code for Data Accesses

- Extract any data access information purely from the source code
- Compile the program into a Data-Flow Oriented IR
- Statistics gathered by analyzing the IR
  - Algorithmic intensity
  - Volume of data circulating in the program

```
void square(double &v) {
    v = v * v;
}

// ...

square(A[i]);
```



Source: *Alexandru Calotoiu et al. "Lifting C semantics for dataflow optimization". In: Proceedings of the 36th ACM International Conference on Supercomputing. 2022, pp. 1-13.*

Analyze the Programs Source Code for Data Accesses

- ☺ Very fast
- ☺ Provides holistic view of the program and its performance

- ☹ Very abstract analysis
  - Memory layout of data is not considered
  - Hardware architecture unknown
  - No information about real-world cache performance

Data Gathering Approaches I Cache Simulation

Imitate the Programs Memory Accesses on a Simulated Cache Hierarchy

- Replicate actual hardware through software
  - Cache hierarchy (size, associativity, etc.)
  - Cache replacement policies
  - Cache coherence protocols
- Simulate the programs memory-wise on the simulated hardware
  - Memory (de-)allocations
  - Data accesses

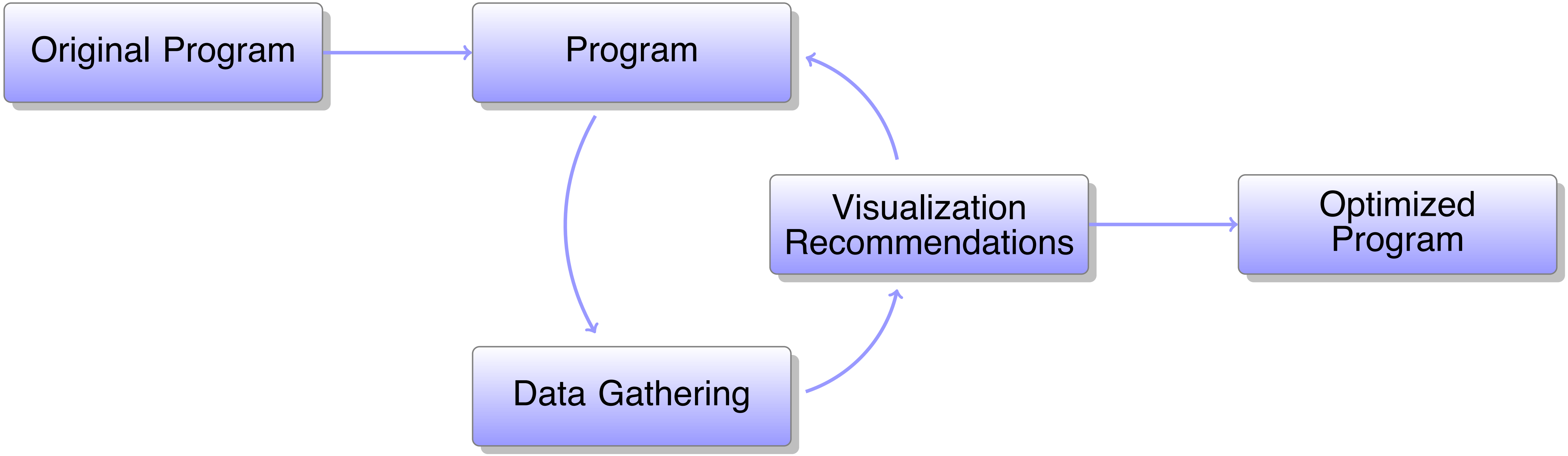
☺ Very detailed
- Insights about the memory-layout of data
- Enables step-by-step analysis of the caches

☺ Allows to analyze only parts of the program

☹ Requires precise parameterization

Overview of the Optimization Workflow

Visualization-Guided Optimization

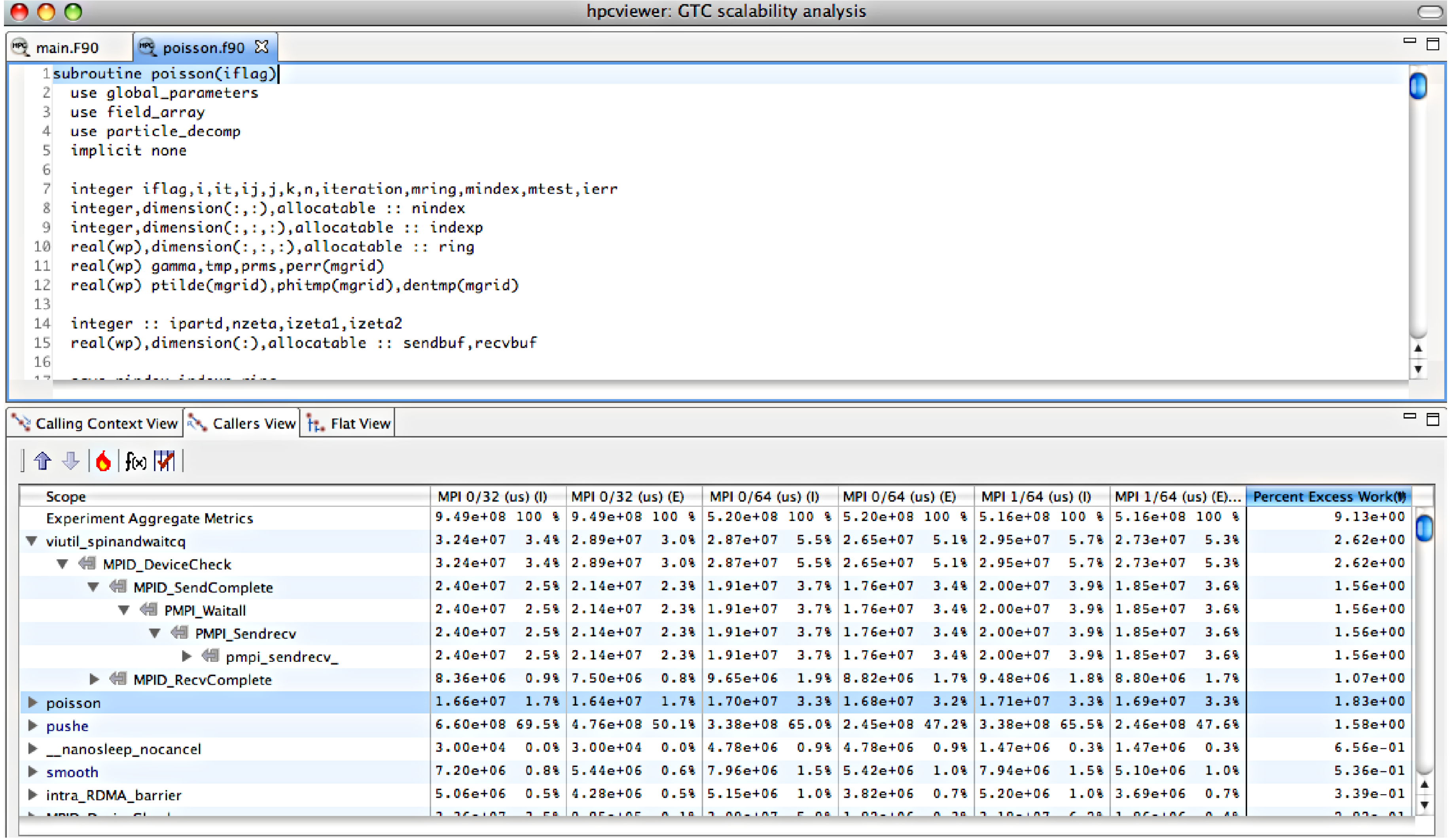


Visualization Techniques

Goal: Display & Explain Bottlenecks

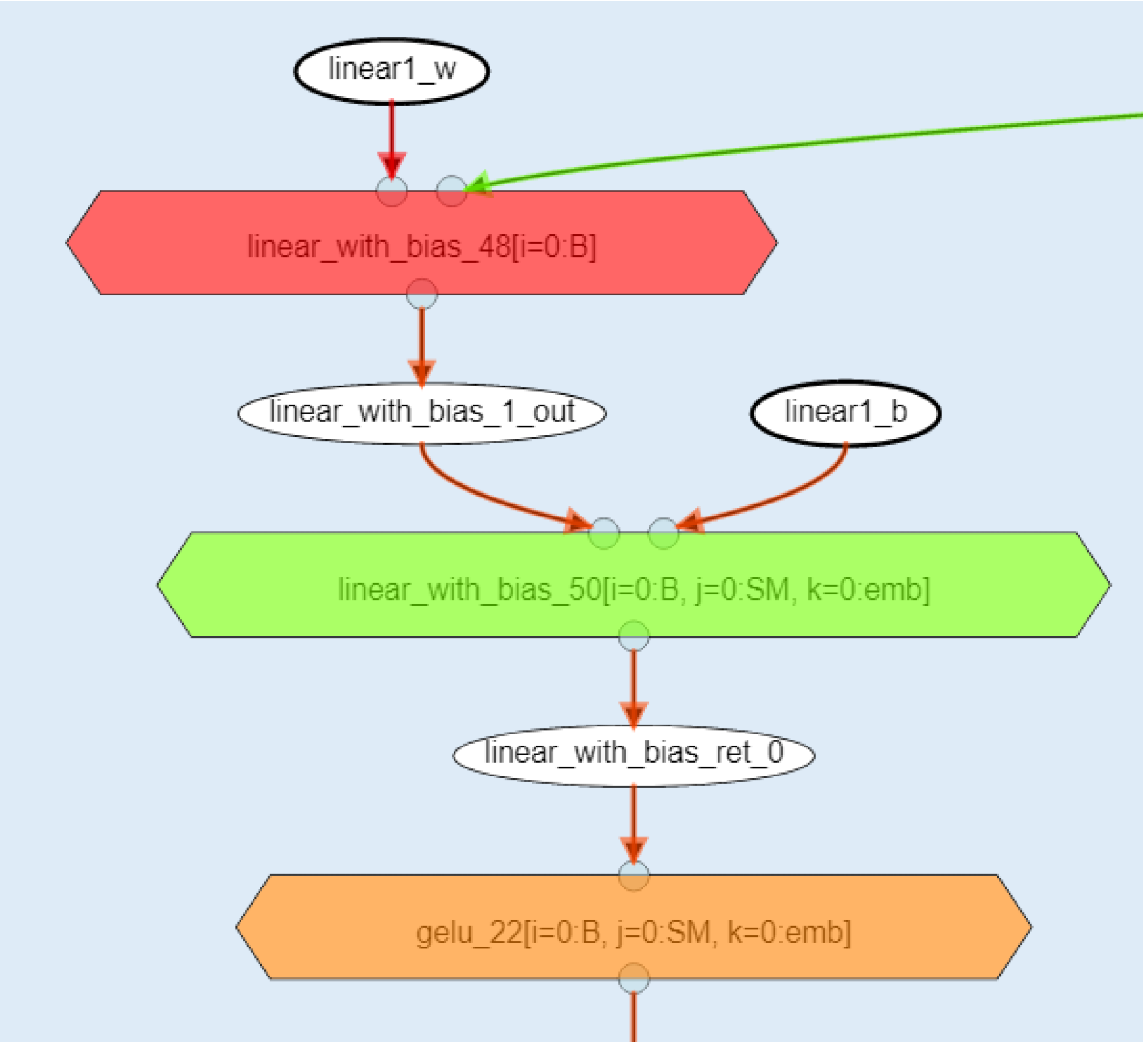
- Balance between intuitiveness and informational value
- Three broad categories:
  - High-level visualizations
  - Intermediate-level visualizations
  - Low-level visualizations

Visualization Techniques I High-Level



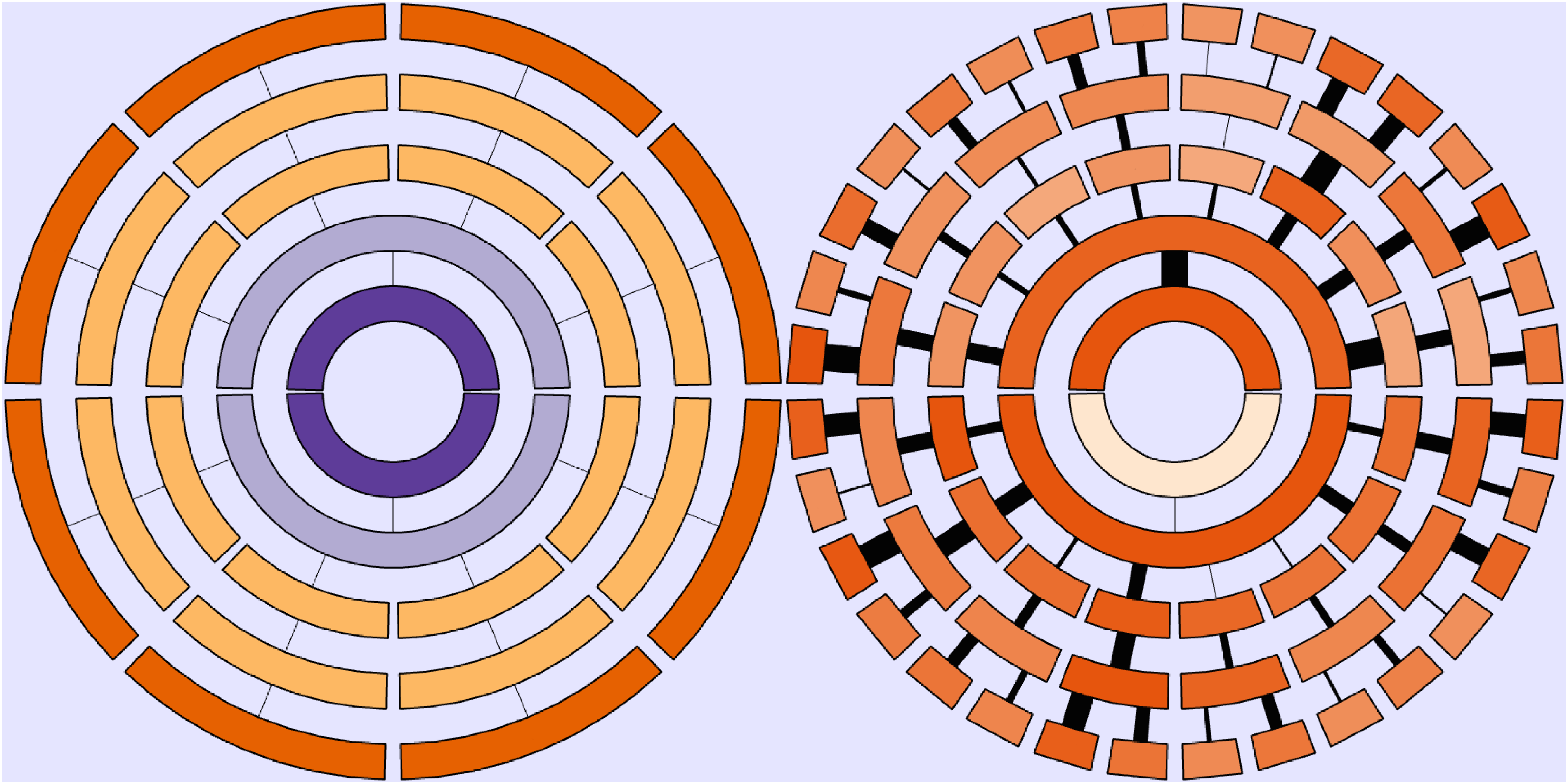
Source: *Laksono Adhianto et al. "HPCToolkit: Tools for performance analysis of optimized parallel programs". In: Concurrency and Computation: Practice and Experience 22.6 (2010), pp. 685-701.*

Visualization Techniques I High-Level



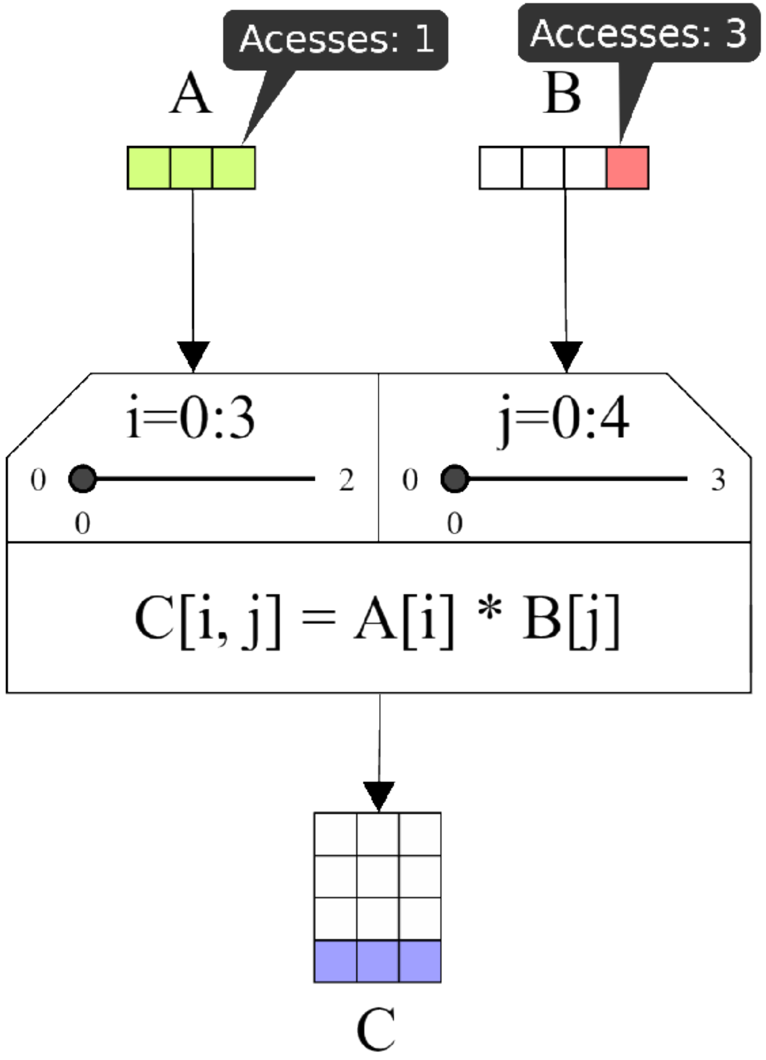
Source: *Philipp Schaad, Tal Ben-Nun, and Torsten Hoefler. "Boosting performance optimization with interactive data movement visualization". In: arXiv preprint arXiv:2207.07433 (2022).*

Visualization Techniques I Intermediate-Level



Source: *Alfredo Giménez et al. “Memaxes: Visualiza- tion and analytics for characterizing complex memory performance behaviors”. In: IEEE transactions on visualization and computer graphics 24.7 (2017), pp. 2180-2193.*

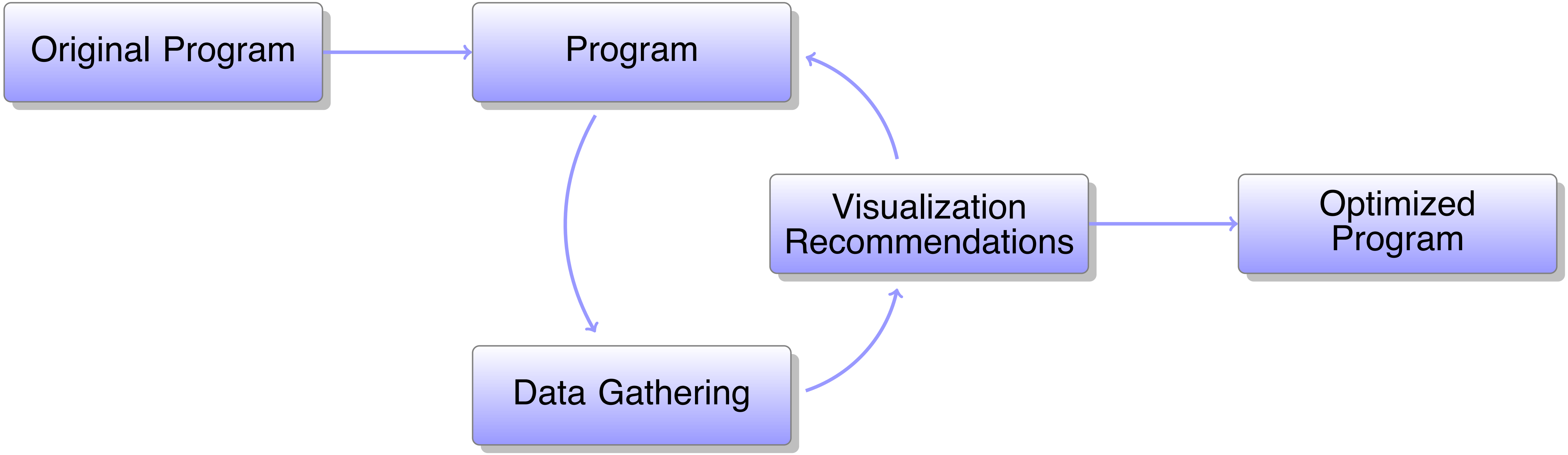
Visualization Techniques | Low-Level



Source: *Philipp Schaad, Tal Ben-Nun, and Torsten Hoefler. "Boosting performance optimization with interactive data movement visualization". In: arXiv preprint arXiv:2207.07433 (2022).*

Overview of the Optimization Workflow

Visualization-Guided Optimization



Boosting Performance Optimization with Interactive Data Movement Visualization

Two-Tier Program Analysis

Global Level

- Static analysis of the program
- High-level visualizations

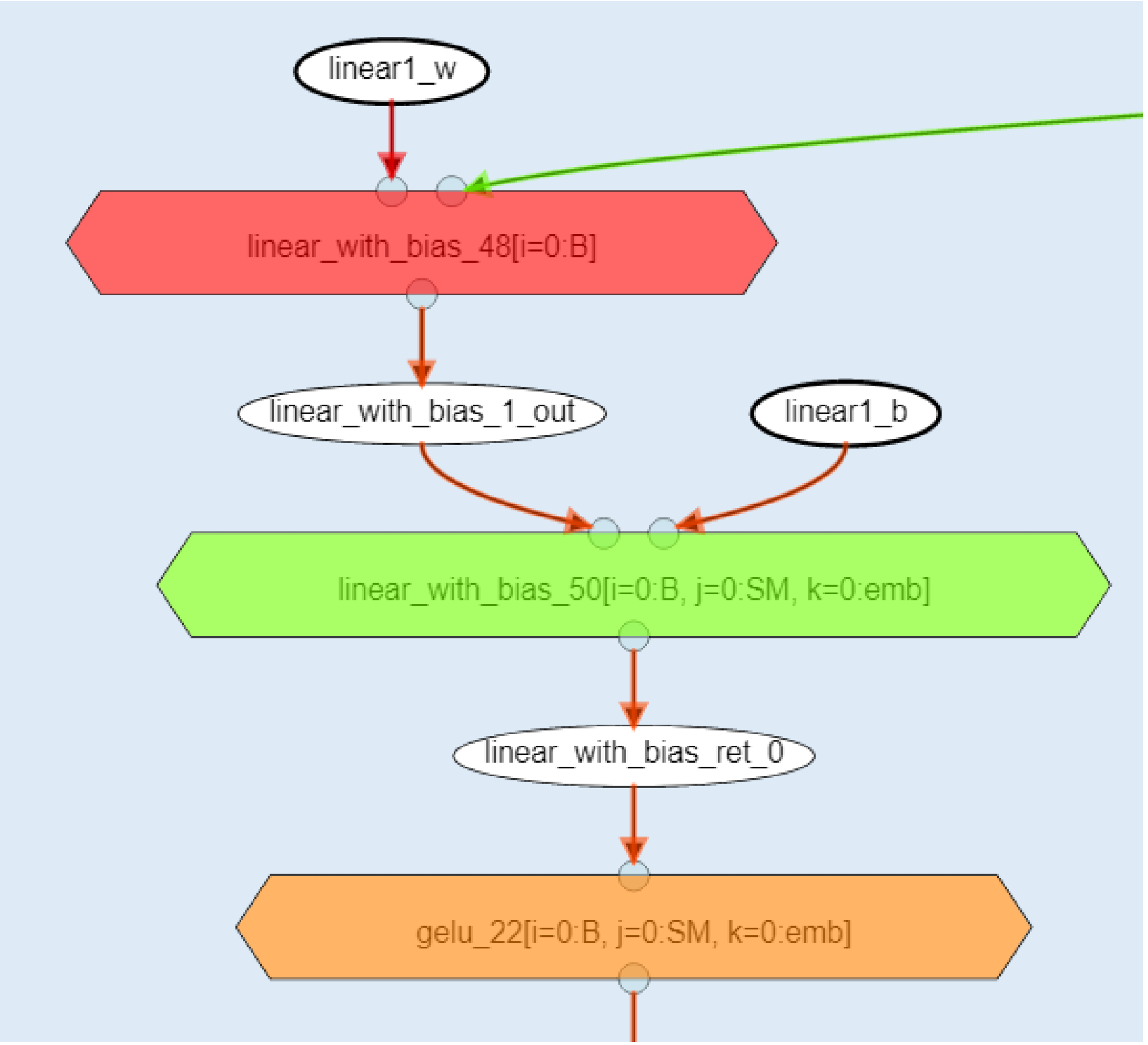
→ Identify regions of interest

Local Level

- Cache simulation
- Low-level visualizations

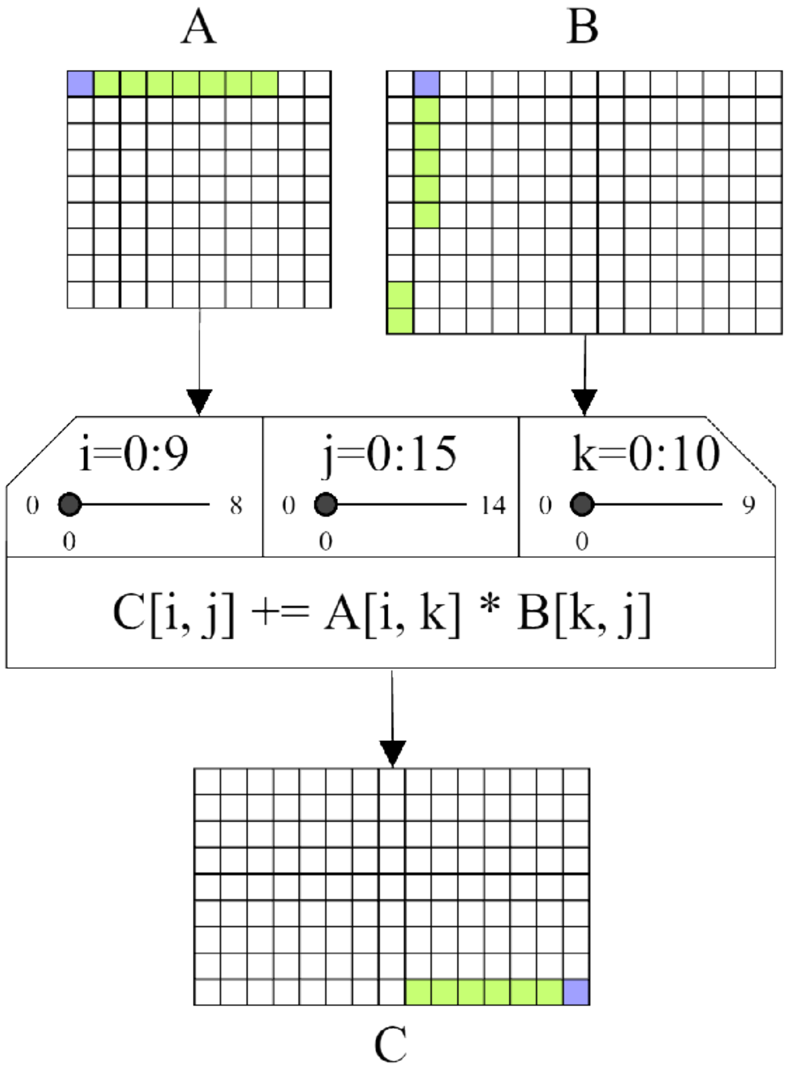
→ Understand the data movements and access patterns

Boosting Performance Optimization with Interactive Data Movement Visualization



Source: *Philipp Schaad, Tal Ben-Nun, and Torsten Hoefler. "Boosting performance optimization with interactive data movement visualization". In: arXiv preprint arXiv:2207.07433 (2022).*

Boosting Performance Optimization with Interactive Data Movement Visualization



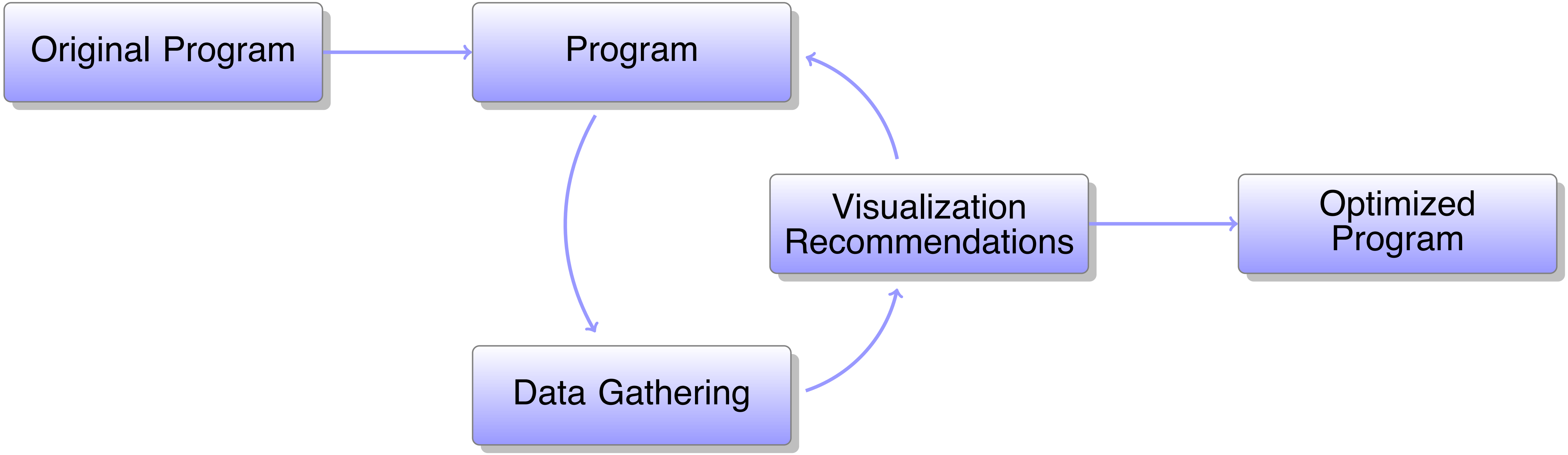
Source: *Philipp Schaad, Tal Ben-Nun, and Torsten Hoefler. "Boosting performance optimization with interactive data movement visualization". In: arXiv preprint arXiv:2207.07433 (2022).*

Conclusion

- Importance of data locality due to processor-memory gap
- Data Gathering Methods:
  - Dynamic Analysis
  - Static Analysis
  - Cache Simulation
- Visualization: High-level to fine-grained insights
- Future goals: Automatic optimization in compilers

Overview of the Optimization Workflow

Visualization-Guided Optimization



References I

- Alexandru Calotoiu et al. "Lifting C semantics for dataflow optimization". In: *Proceedings of the 36th ACM International Conference on Supercomputing*. 2022, pp. 1-13.
- Laksono Adhianto et al. "HPCToolkit: Tools for performance analysis of optimized parallel programs". In: *Concurrency and Computation: Practice and Experience* 22.6 (2010), pp. 685-701.
- Alfredo Giménez et al. "Memaxes: Visualiza- tion and analytics for characterizing complex memory performance behaviors". In: *IEEE transactions on visualization and computer graphics* 24.7 (2017), pp. 2180-2193.
- Philipp Schaad, Tal Ben-Nun, and Torsten Hoefler. "Boosting performance optimization with interactive data movement visualization". In: *arXiv preprint arXiv:2207.07433* (2022).