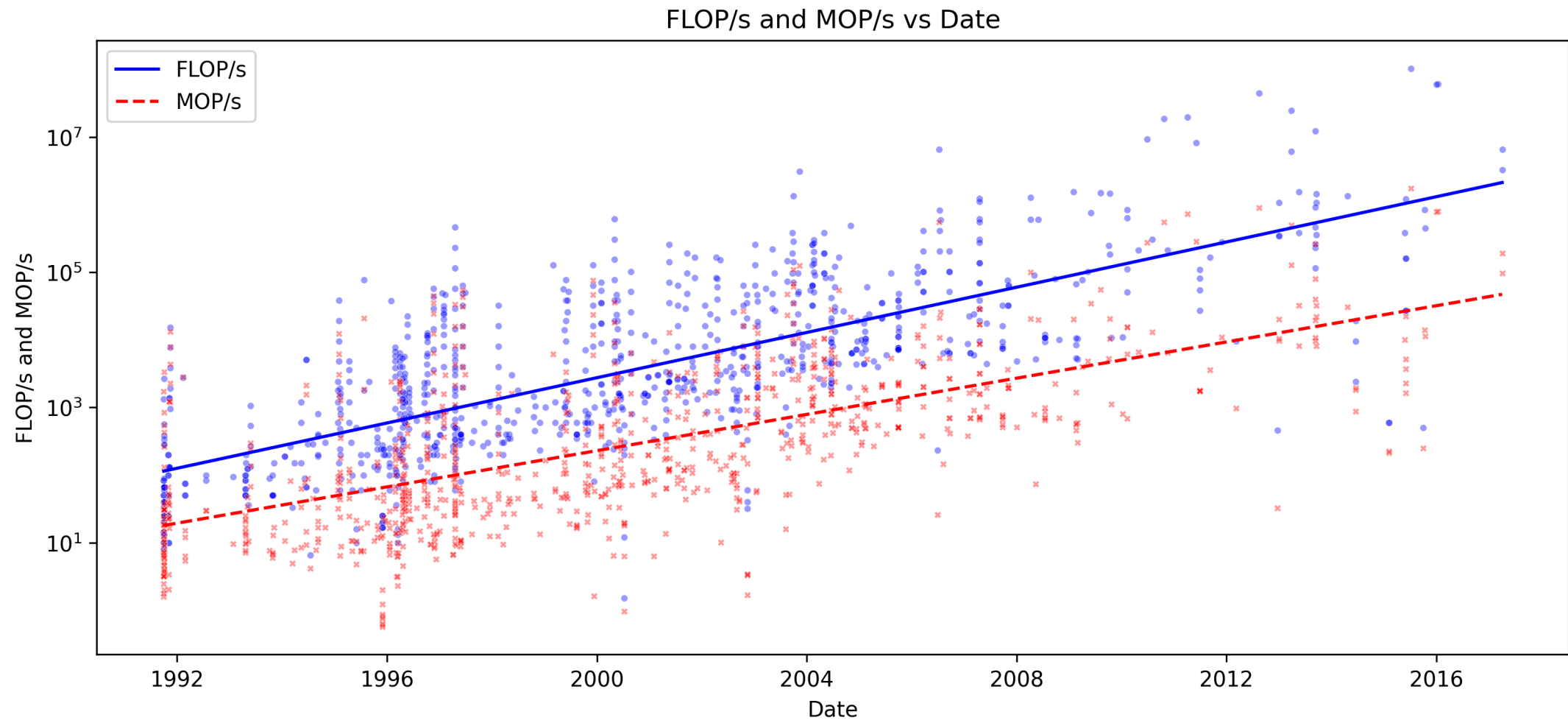


# Visualization of Data Movements and Accesses

Til Mohr

# Memory-Related Performance Problems | Processor-Memory Performance Gap



Data: <https://www.cs.virginia.edu/stream/>

## Memory-Related Performance Problems | Data Locality

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

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

$$t_c \ll t_m \quad (2)$$

# Memory-Related Performance Problems I Data Locality

Listing 1: Matrix Summation

```
1 let matrix = Matrix::random(2, 2);
2 let mut sum = 0;
3 for column in 0..2 {
4     for row in 0..2 {
5         sum += matrix.get(row, column);
6     }
7 }
8 sum
```

Matrix:

	0	1
0	1	2
1	3	4

# Memory-Related Performance Problems I Data Locality

Listing 1: Matrix Summation

```
1 let matrix = Matrix::random(2, 2);
2 let mut sum = 0;
3 for column in 0..2 {
4     for row in 0..2 {
5         sum += matrix.get(row, column);
6     }
7 }
8 sum
```

Matrix:

	0	1
0	1	2
1	3	4

Matrix in Memory:

1	2	3	4
---	---	---	---

# Memory-Related Performance Problems I Data Locality

Listing 1: Matrix Summation

```
1 let matrix = Matrix::random(2, 2);
2 let mut sum = 0;
3 for column in 0..2 {
4     for row in 0..2 {
5         sum += matrix.get(row, column);
6     }
7 }
8 sum
```

Matrix:

	0	1
0	1	2
1	3	4

Current Item:

0	0
---	---

Matrix in Memory:

1	2	3	4
---	---	---	---

Cache:

--	--

Number of cache misses: 0

# Memory-Related Performance Problems I Data Locality

Listing 1: Matrix Summation

```
1 let matrix = Matrix::random(2, 2);
2 let mut sum = 0;
3 for column in 0..2 {
4     for row in 0..2 {
5         sum += matrix.get(row, column);
6     }
7 }
8 sum
```

Matrix:

	0	1
0	1	2
1	3	4

Current Item:

0	0
---	---

Matrix in Memory:

1	2	3	4
---	---	---	---

Cache:

1	2
---	---

Number of cache misses: 1

# Memory-Related Performance Problems I Data Locality

Listing 1: Matrix Summation

```
1 let matrix = Matrix::random(2, 2);
2 let mut sum = 0;
3 for column in 0..2 {
4     for row in 0..2 {
5         sum += matrix.get(row, column);
6     }
7 }
8 sum
```

Matrix:

	0	1
0	1	2
1	3	4

Current Item:

1	0
---	---

Matrix in Memory:

1	2	3	4
---	---	---	---

Cache:

1	2
---	---

Number of cache misses: 1



# Memory-Related Performance Problems I Data Locality

Listing 1: Matrix Summation

```
1 let matrix = Matrix::random(2, 2);
2 let mut sum = 0;
3 for column in 0..2 {
4     for row in 0..2 {
5         sum += matrix.get(row, column);
6     }
7 }
8 sum
```

Matrix:

	0	1
0	1	2
1	3	4

Current Item:

1	0
---	---

Matrix in Memory:

1	2	3	4
---	---	---	---

Cache:

3	4
---	---

Number of cache misses: 2

# Memory-Related Performance Problems I Data Locality

Listing 1: Matrix Summation

```
1 let matrix = Matrix::random(2, 2);
2 let mut sum = 0;
3 for column in 0..2 {
4     for row in 0..2 {
5         sum += matrix.get(row, column);
6     }
7 }
8 sum
```

Matrix:

	0	1
0	1	2
1	3	4

Current Item:

0	1
---	---

Matrix in Memory:

1	2	3	4
---	---	---	---

Cache:

3	4
---	---

Number of cache misses: 2

# Memory-Related Performance Problems I Data Locality

Listing 1: Matrix Summation

```
1 let matrix = Matrix::random(2, 2);
2 let mut sum = 0;
3 for column in 0..2 {
4     for row in 0..2 {
5         sum += matrix.get(row, column);
6     }
7 }
8 sum
```

Matrix:

	0	1
0	1	2
1	3	4

Current Item:

0	1
---	---

Matrix in Memory:

1	2	3	4
---	---	---	---

Cache:

1	2
---	---

Number of cache misses: 3

# Memory-Related Performance Problems I Data Locality

Listing 1: Matrix Summation

```
1 let matrix = Matrix::random(2, 2);
2 let mut sum = 0;
3 for column in 0..2 {
4     for row in 0..2 {
5         sum += matrix.get(row, column);
6     }
7 }
8 sum
```

Matrix:

	0	1
0	1	2
1	3	4

Current Item:

1	1
---	---

Matrix in Memory:

1	2	3	4
---	---	---	---

Cache:

1	2
---	---

Number of cache misses: 3

# Memory-Related Performance Problems I Data Locality

Listing 1: Matrix Summation

```
1 let matrix = Matrix::random(2, 2);
2 let mut sum = 0;
3 for column in 0..2 {
4     for row in 0..2 {
5         sum += matrix.get(row, column);
6     }
7 }
8 sum
```

Matrix:

	0	1
0	1	2
1	3	4

Current Item:

1	1
---	---

Matrix in Memory:

1	2	3	4
---	---	---	---

Cache:

3	4
---	---

Number of cache misses: 4

# Memory-Related Performance Problems I Data Locality

Listing 2: Reordered Matrix Summation

```
1 let matrix = Matrix::random(2, 2);
2 let mut sum = 0;
3 for row in 0..2 {
4     for column in 0..2 {
5         sum += matrix.get(row, column);
6     }
7 }
8 sum
```

Matrix:

	0	1
0	1	2
1	3	4

Current Item:

0	0
---	---

Matrix in Memory:

1	2	3	4
---	---	---	---

Cache:

--	--

Number of cache misses: 0

# Memory-Related Performance Problems I Data Locality

Listing 2: Reordered Matrix Summation

```
1 let matrix = Matrix::random(2, 2);
2 let mut sum = 0;
3 for row in 0..2 {
4     for column in 0..2 {
5         sum += matrix.get(row, column);
6     }
7 }
8 sum
```

Matrix:

	0	1
0	1	2
1	3	4

Current Item:

0	0
---	---

Matrix in Memory:

1	2	3	4
---	---	---	---

Cache:

1	2
---	---

Number of cache misses: 1

# Memory-Related Performance Problems I Data Locality

Listing 2: Reordered Matrix Summation

```
1 let matrix = Matrix::random(2, 2);
2 let mut sum = 0;
3 for row in 0..2 {
4     for column in 0..2 {
5         sum += matrix.get(row, column);
6     }
7 }
8 sum
```

Matrix:

	0	1
0	1	2
1	3	4

Matrix in Memory:

1	2	3	4
---	---	---	---

Current Item:

0	1
---	---

Cache:

1	2
---	---

Number of cache misses: 1



# Memory-Related Performance Problems I Data Locality

Listing 2: Reordered Matrix Summation

```
1 let matrix = Matrix::random(2, 2);
2 let mut sum = 0;
3 for row in 0..2 {
4     for column in 0..2 {
5         sum += matrix.get(row, column);
6     }
7 }
8 sum
```

Matrix:

	0	1
0	1	2
1	3	4

Current Item:

1	0
---	---

Matrix in Memory:

1	2	3	4
---	---	---	---

Cache:

1	2
---	---

Number of cache misses: 1

# Memory-Related Performance Problems I Data Locality

Listing 2: Reordered Matrix Summation

```
1 let matrix = Matrix::random(2, 2);
2 let mut sum = 0;
3 for row in 0..2 {
4     for column in 0..2 {
5         sum += matrix.get(row, column);
6     }
7 }
8 sum
```

Matrix:

	0	1
0	1	2
1	3	4

Current Item:

1	0
---	---

Matrix in Memory:

1	2	3	4
---	---	---	---

Cache:

3	4
---	---

Number of cache misses: 2

# Memory-Related Performance Problems I Data Locality

Listing 2: Reordered Matrix Summation

```
1 let matrix = Matrix::random(2, 2);
2 let mut sum = 0;
3 for row in 0..2 {
4     for column in 0..2 {
5         sum += matrix.get(row, column);
6     }
7 }
8 sum
```

Matrix:

	0	1
0	1	2
1	3	4

Current Item:

1	1
---	---

Matrix in Memory:

1	2	3	4
---	---	---	---

Cache:

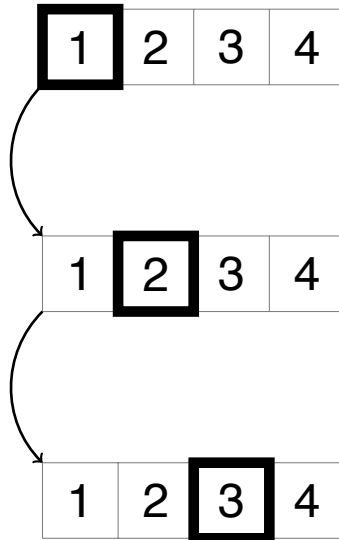
3	4
---	---

Number of cache misses: 2

# Memory-Related Performance Problems I Data Locality

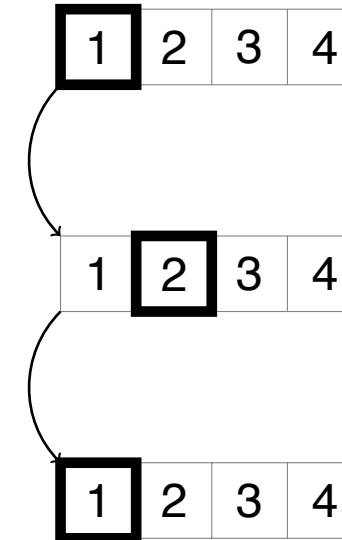
## Spatial Locality

- 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



# How to optimize a program for data locality?

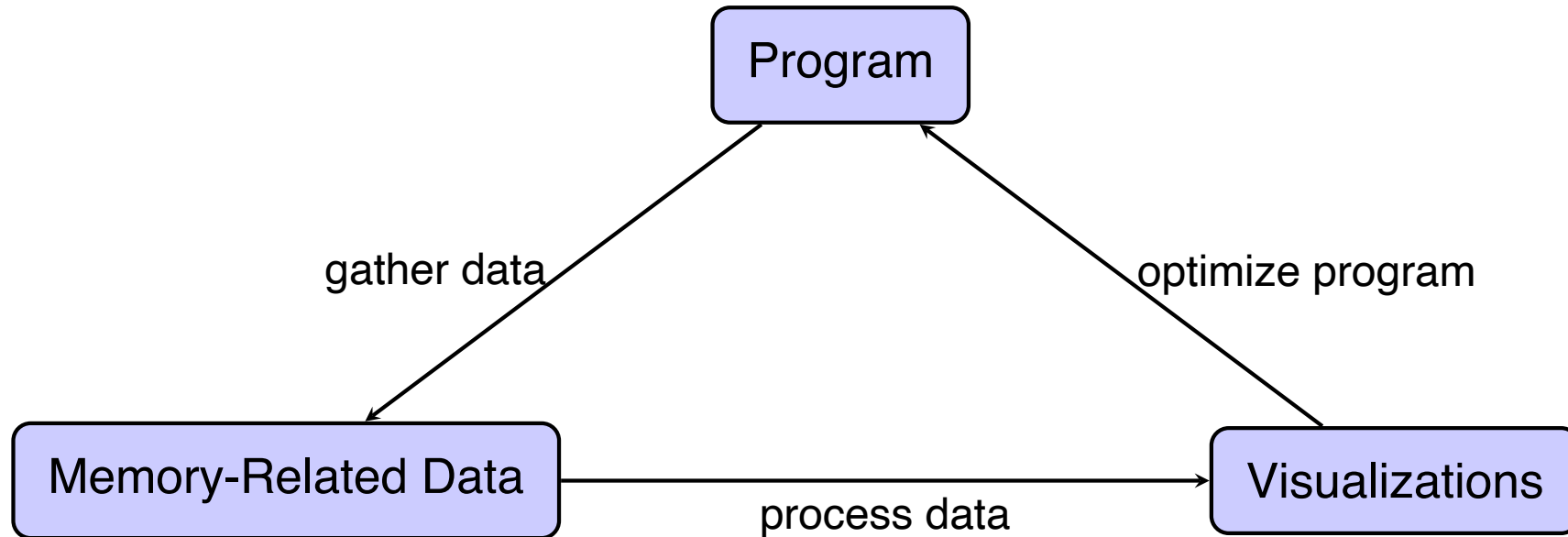
How to optimize a program for data locality?  
→ **Visualization-guided optimization**

# Outline

---

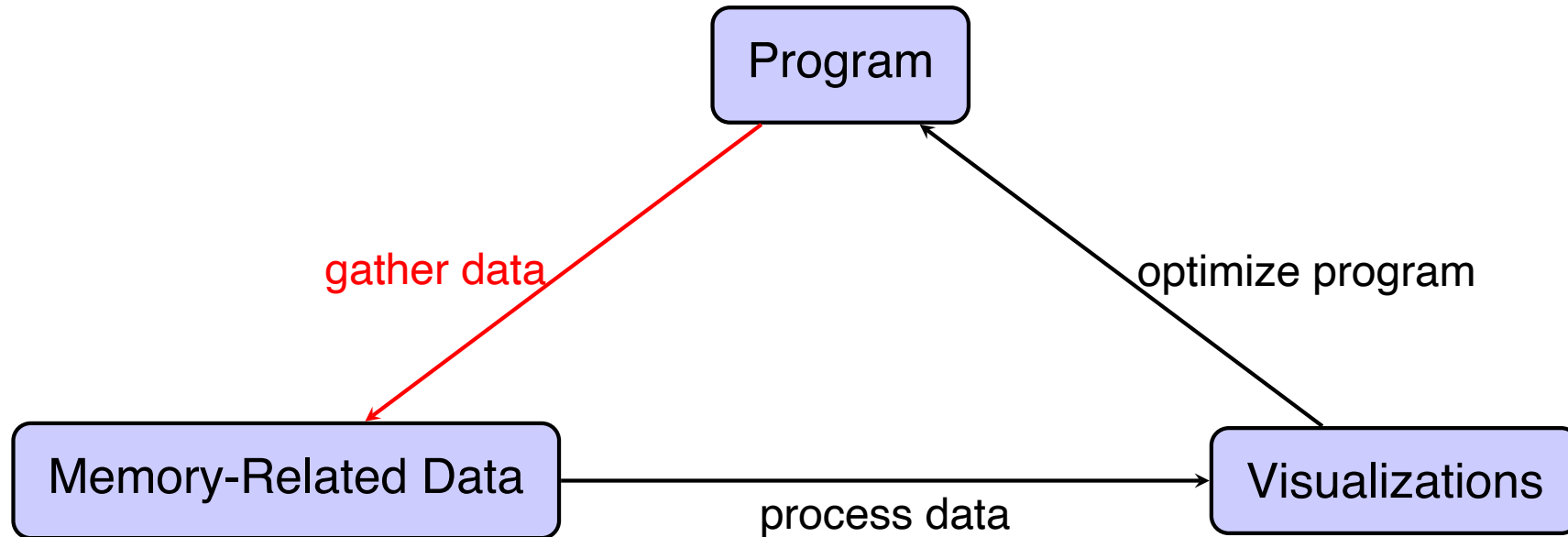
- Overview of the Optimization Workflow
- Data Gathering Approaches
- Visualization Techniques
- Specific Optimization Tool
- Conclusion

## Visualization-Guided Optimization





## Visualization-Guided Optimization



## 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

## Run Program and Capture Memory-Related Information

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

## 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

## Data Gathering Approaches I Static Analysis

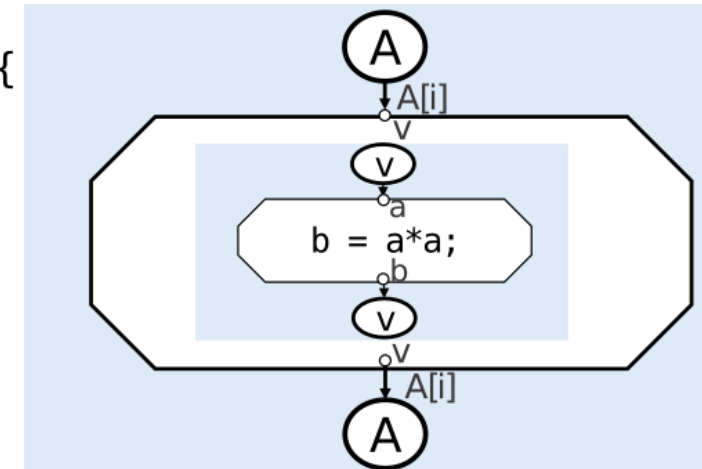
### Analyze the Programs Source Code for Data Accesses

- 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]);
```



## Data Gathering Approaches I Static Analysis

### 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

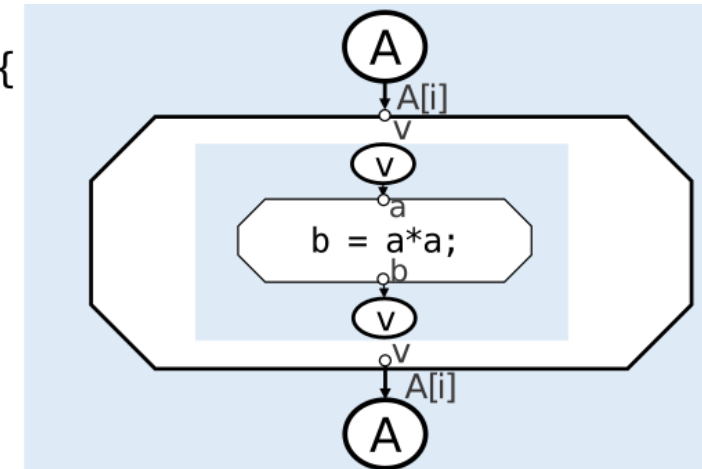
😊 Very fast

😊 Provides holistic view of the program and its performance

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

// ...

square(A[i]);
```



☹️ Very abstract analysis

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

## 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

## 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

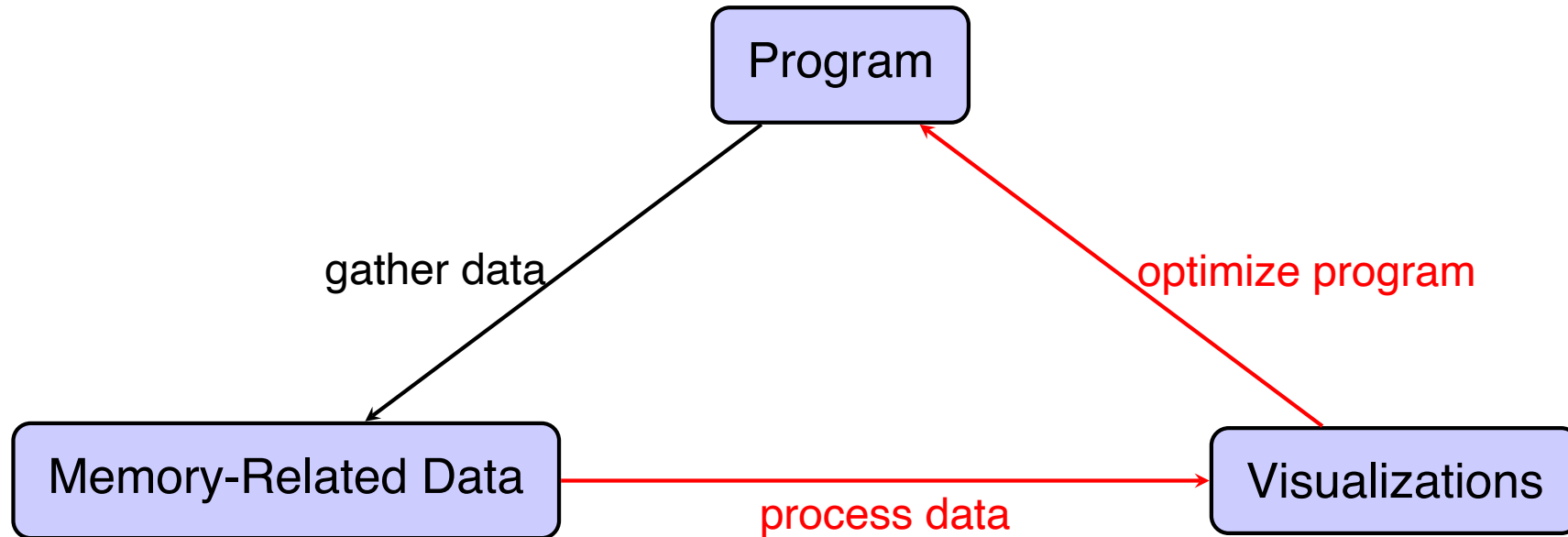
- Insights about the memory-layout of data
- Enables step-by-step analysis of the caches

😊 Allows to analyze only parts of the program

☹ Requires precise parameterization



## Visualization-Guided Optimization



## 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

hpcviewer: GTC scalability analysis

main.F90    poisson.f90

```

1 subroutine poisson(iflag)
2   use global_parameters
3   use field_array
4   use particle_decomp
5   implicit none
6
7   integer iflag,i,it,ij,j,k,n,iteration,mring,mindex,mtest,ierr
8   integer,dimension(:,,:),allocatable :: nindex
9   integer,dimension(:,,:),allocatable :: indexp
10  real(wp),dimension(:,,:),allocatable :: ring
11  real(wp) gamma,tmp,prms,perr(mgrid)
12  real(wp) ptilde(mgrid),phitmp(mgrid),dentmp(mgrid)
13
14  integer :: ipartd,nzeta,izeta1,izeta2
15  real(wp),dimension(:,),allocatable :: sendbuf,recvbuf
16
17  save nindex,indexp,ring

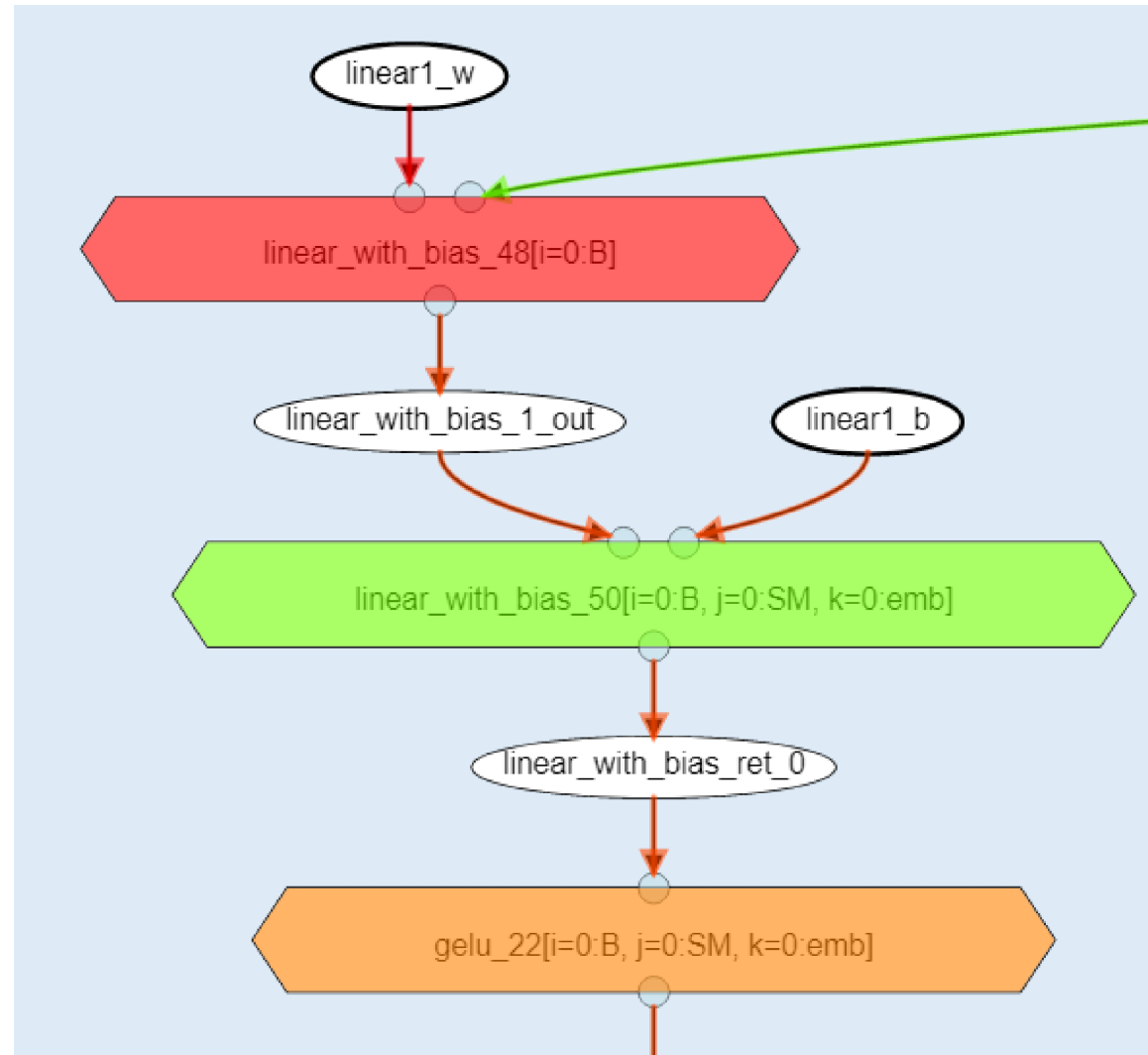
```

Calling Context View    Callers View    Flat View

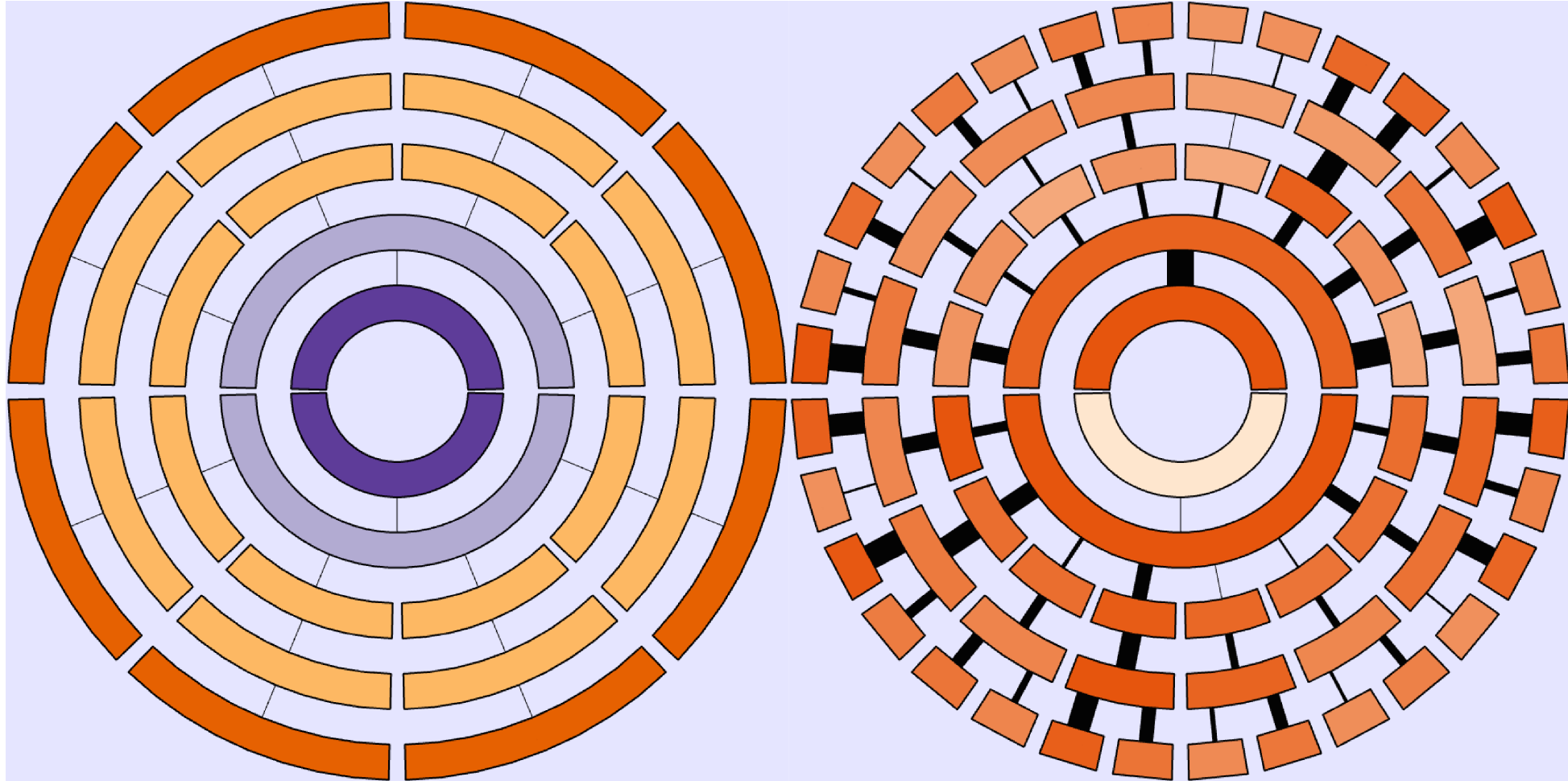
↑ ↓ 🔥 f(x) 🔍

Scope	MPI 0/32 (us) (I)	MPI 0/32 (us) (E)	MPI 0/64 (us) (I)	MPI 0/64 (us) (E)	MPI 1/64 (us) (I)	MPI 1/64 (us) (E)	Percent Excess Work(I)
Experiment Aggregate Metrics	9.49e+08 100 %	9.49e+08 100 %	5.20e+08 100 %	5.20e+08 100 %	5.16e+08 100 %	5.16e+08 100 %	9.13e+00
▼ viutil_spinandwaitcq	3.24e+07 3.4%	2.89e+07 3.0%	2.87e+07 5.5%	2.65e+07 5.1%	2.95e+07 5.7%	2.73e+07 5.3%	2.62e+00
▼ MPID_DeviceCheck	3.24e+07 3.4%	2.89e+07 3.0%	2.87e+07 5.5%	2.65e+07 5.1%	2.95e+07 5.7%	2.73e+07 5.3%	2.62e+00
▼ MPID_SendComplete	2.40e+07 2.5%	2.14e+07 2.3%	1.91e+07 3.7%	1.76e+07 3.4%	2.00e+07 3.9%	1.85e+07 3.6%	1.56e+00
▼ PMPI_Waitall	2.40e+07 2.5%	2.14e+07 2.3%	1.91e+07 3.7%	1.76e+07 3.4%	2.00e+07 3.9%	1.85e+07 3.6%	1.56e+00
▼ PMPI_Sendrecv	2.40e+07 2.5%	2.14e+07 2.3%	1.91e+07 3.7%	1.76e+07 3.4%	2.00e+07 3.9%	1.85e+07 3.6%	1.56e+00
▶ pmpi_sendrecv_	2.40e+07 2.5%	2.14e+07 2.3%	1.91e+07 3.7%	1.76e+07 3.4%	2.00e+07 3.9%	1.85e+07 3.6%	1.56e+00
▶ MPID_RecvComplete	8.36e+06 0.9%	7.50e+06 0.8%	9.65e+06 1.9%	8.82e+06 1.7%	9.48e+06 1.8%	8.80e+06 1.7%	1.07e+00
▶ poisson	1.66e+07 1.7%	1.64e+07 1.7%	1.70e+07 3.3%	1.68e+07 3.2%	1.71e+07 3.3%	1.69e+07 3.3%	1.83e+00
▶ pushe	6.60e+08 69.5%	4.76e+08 50.1%	3.38e+08 65.0%	2.45e+08 47.2%	3.38e+08 65.5%	2.46e+08 47.6%	1.58e+00
▶ __nanosleep_nocancel	3.00e+04 0.0%	3.00e+04 0.0%	4.78e+06 0.9%	4.78e+06 0.9%	1.47e+06 0.3%	1.47e+06 0.3%	6.56e-01
▶ smooth	7.20e+06 0.8%	5.44e+06 0.6%	7.96e+06 1.5%	5.42e+06 1.0%	7.94e+06 1.5%	5.10e+06 1.0%	5.36e-01
▶ intra_RDMA_barrier	5.06e+06 0.5%	4.28e+06 0.5%	5.15e+06 1.0%	3.82e+06 0.7%	5.20e+06 1.0%	3.69e+06 0.7%	3.39e-01

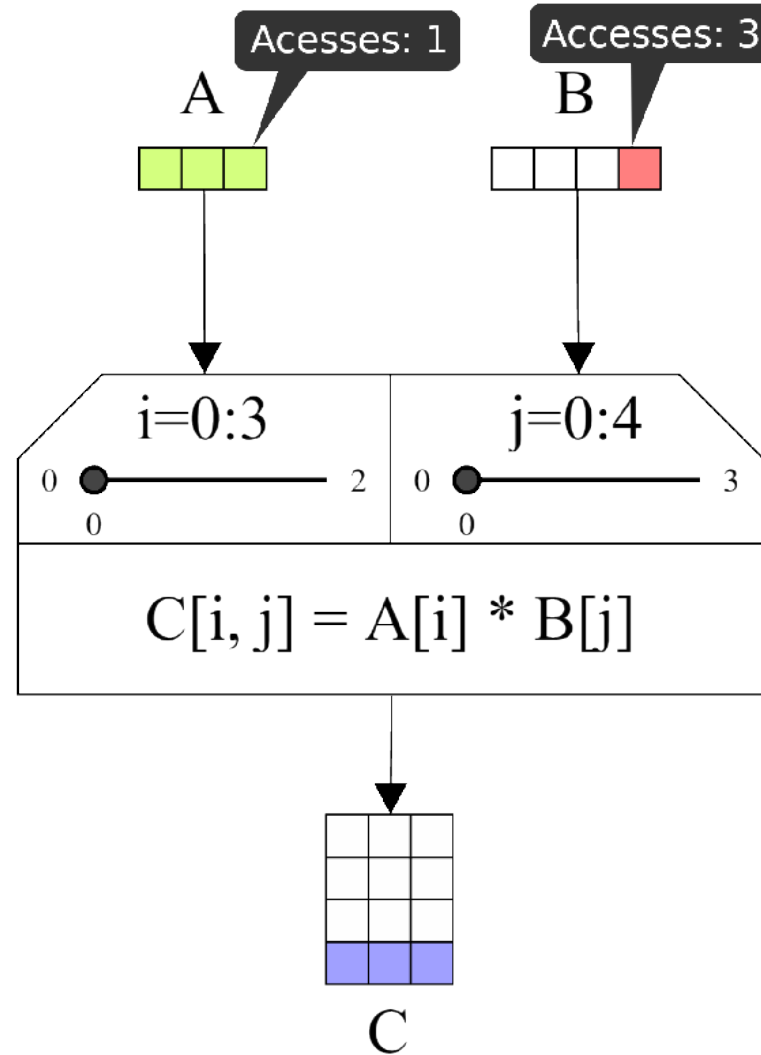
## Visualization Techniques I High-Level



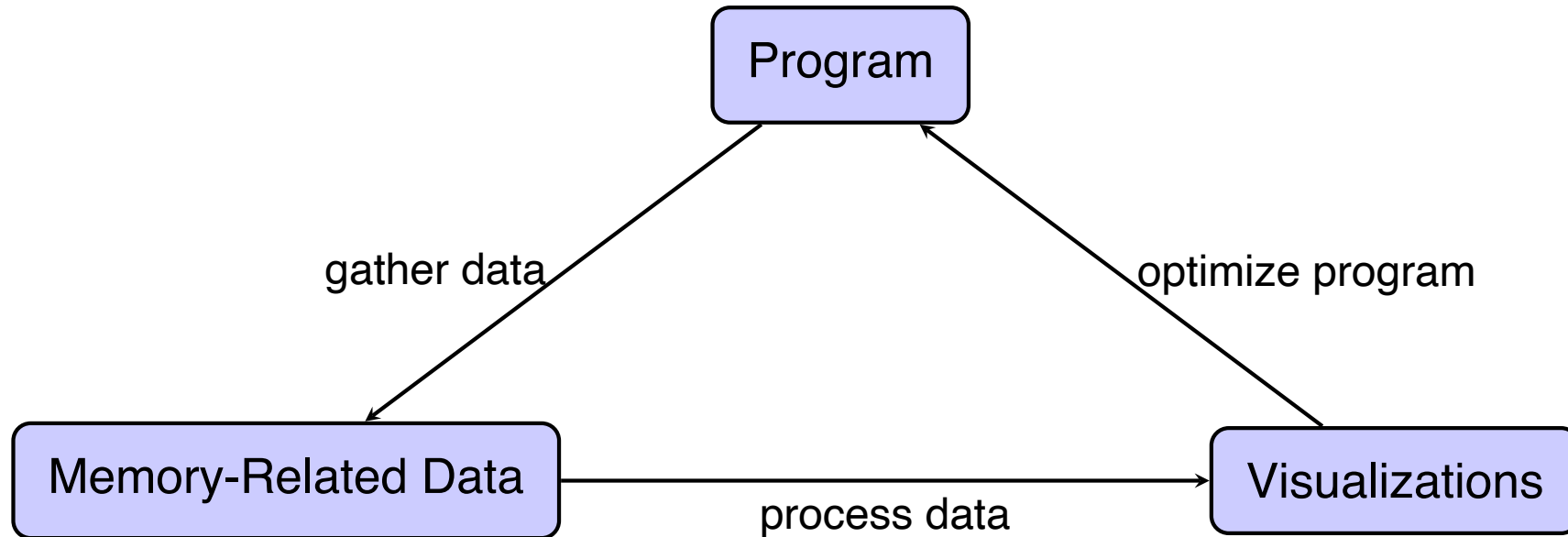
## Visualization Techniques | Intermediate-Level



## Visualization Techniques I Low-Level



## Visualization-Guided Optimization

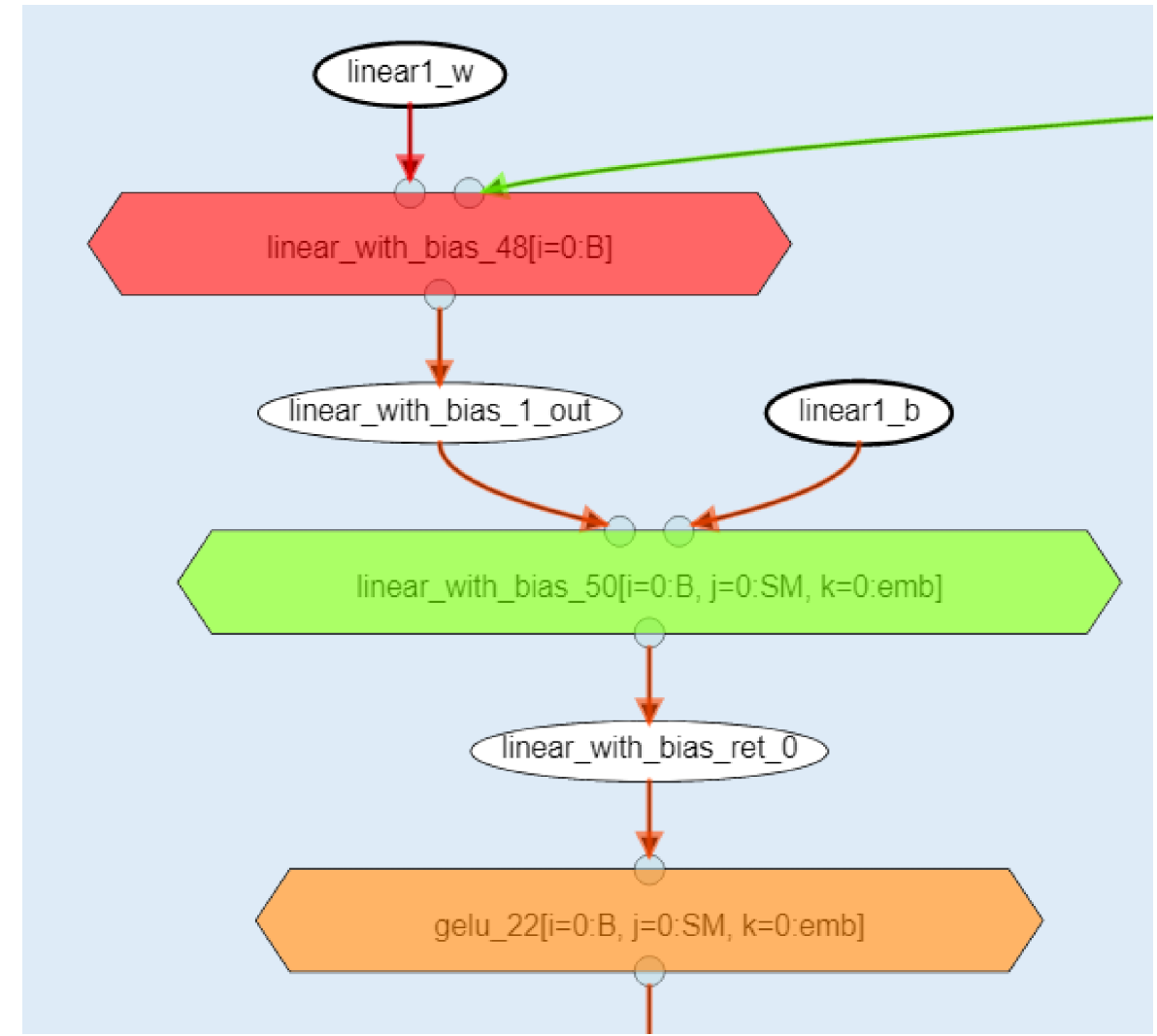


# Boosting Performance Optimization with Interactive Data Movement Visualization

## Two-Tier Program Analysis

### Global Level

- Static analysis of the program
  - High-level visualizations
- Identify regions of interest



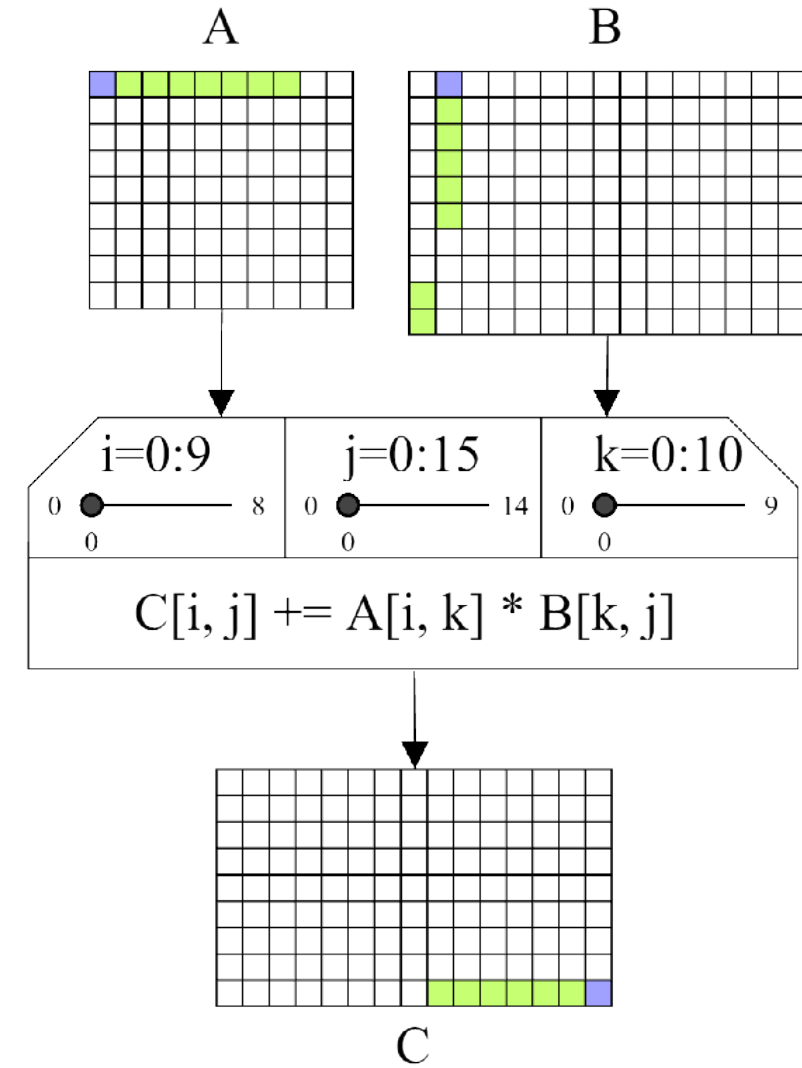


# Boosting Performance Optimization with Interactive Data Movement Visualization

## Two-Tier Program Analysis

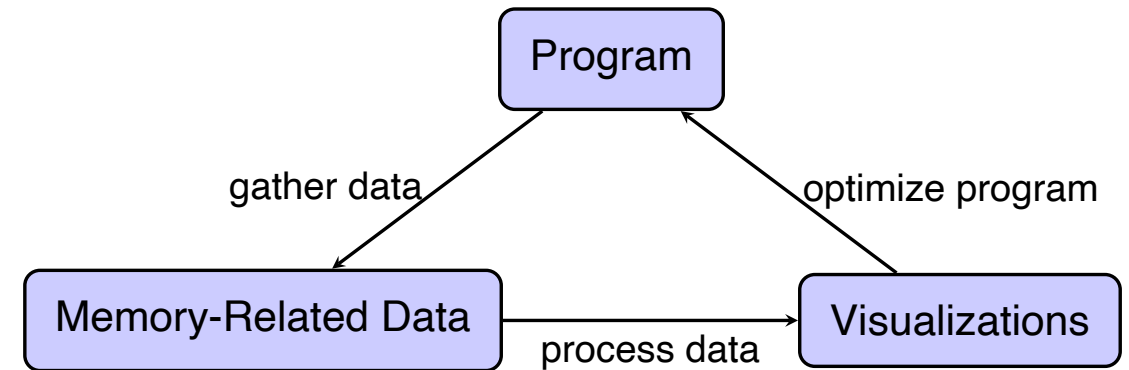
### Local Level

- Cache simulation
  - Low-level visualizations
- Understand the data movements and access patterns



## 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



## References I

- [1] Alexandru Calotoiu et al. “Lifting C semantics for dataflow optimization”. In: *Proceedings of the 36th ACM International Conference on Supercomputing*. 2022, pp. 1-13.
- [2] 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.
- [3] Philipp Schaad. “Boosting Performance Engineering with Visual Interactive Optimization and Analysis”. MA thesis. ETH Zurich, 2021.
- [4] Alfredo Giménez et al. “Memaxes: Visualization and analytics for characterizing complex memory performance behaviors”. In: *IEEE transactions on visualization and computer graphics* 24.7 (2017), pp. 2180-2193.
- [5] Philipp Schaad, Tal Ben-Nun, and Torsten Hoefler. “Boosting performance optimization with interactive data movement visualization”. In: *SC22: International Conference for High Performance Computing, Networking, Storage and Analysis* (2022), pp. 1-16.