Perspective: A Sensible Approach to Speculative Automatic Parallelization

Sotiris Apostolakis, Ziyang Xu, Greg Chan, Simone Campanoni[†], and David I. August

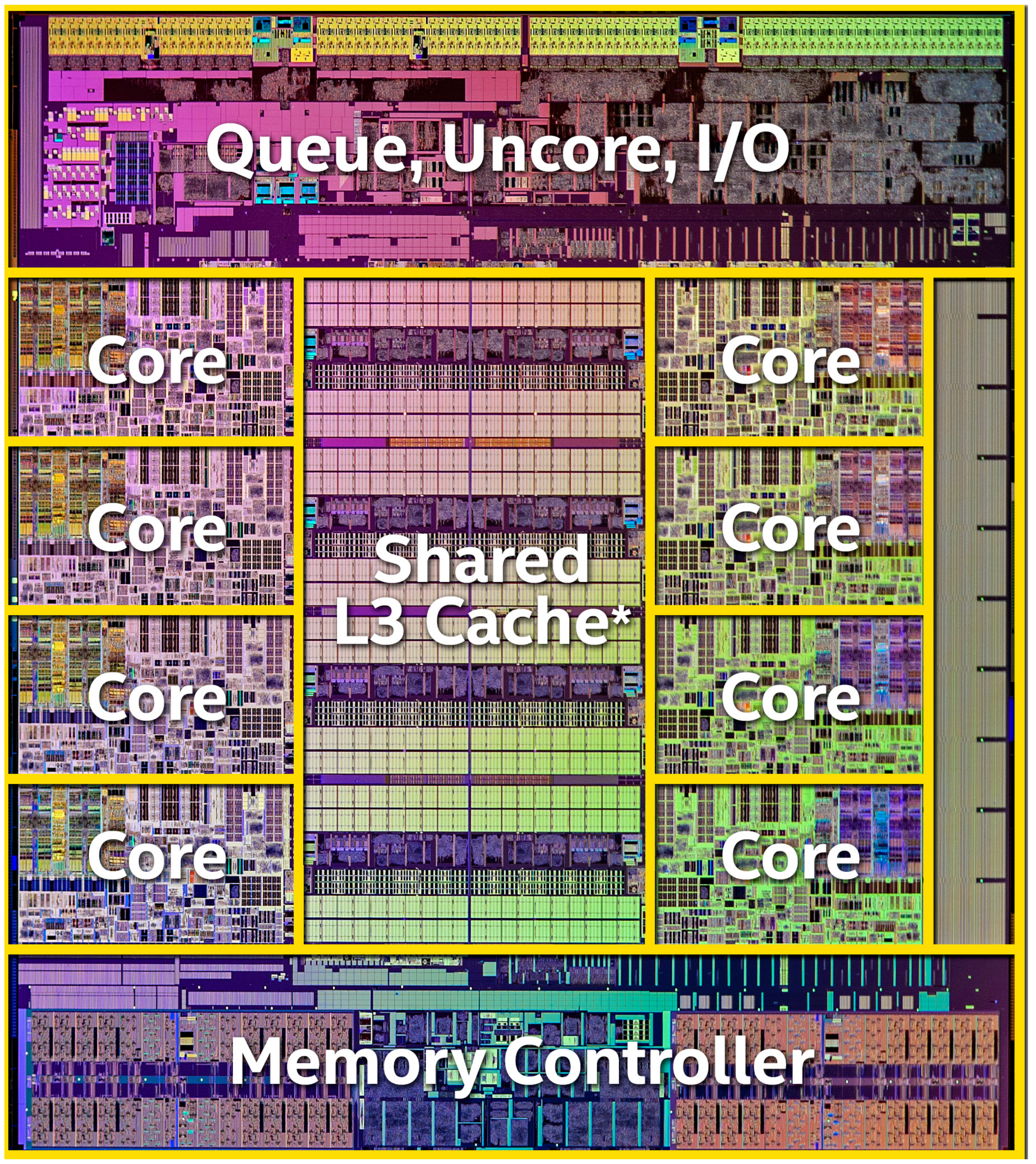
ASPLOS 2020

PRINCETON UNIVERSITY

[†]Northwestern University

Why Automatic Parallelization?

Why Automatic Parallelization?

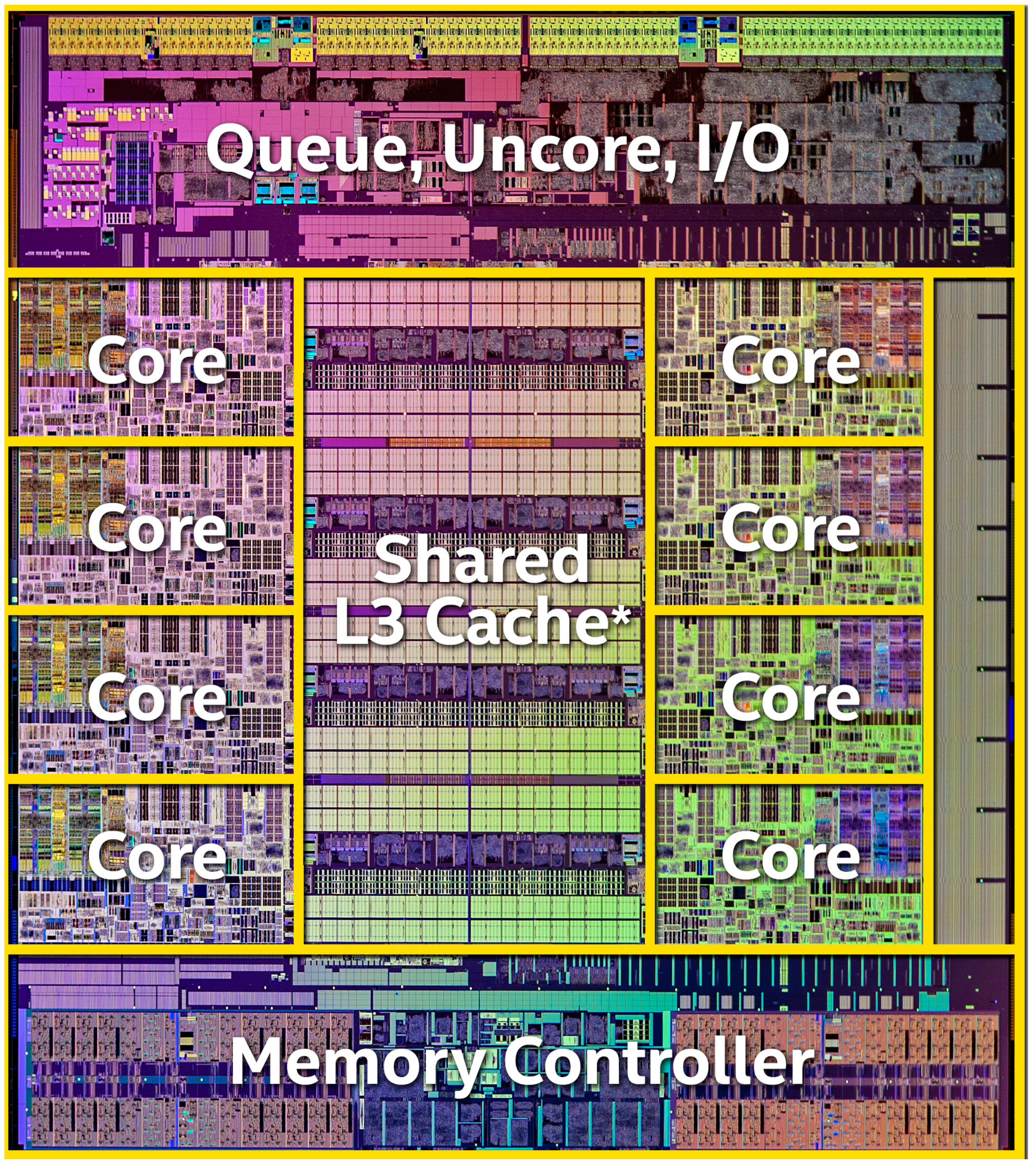


Multicore systems are grossly underutilized [1,2]

[1] L. A. Barroso, U. Holzle. The Datacenter as a Computer, 2013

[2] C. Delimitrou and C. Kozyrakis. Quasar: Resource-Efficient and QoS-Aware Cluster Management, ASPLOS 2014

Why Automatic Parallelization?

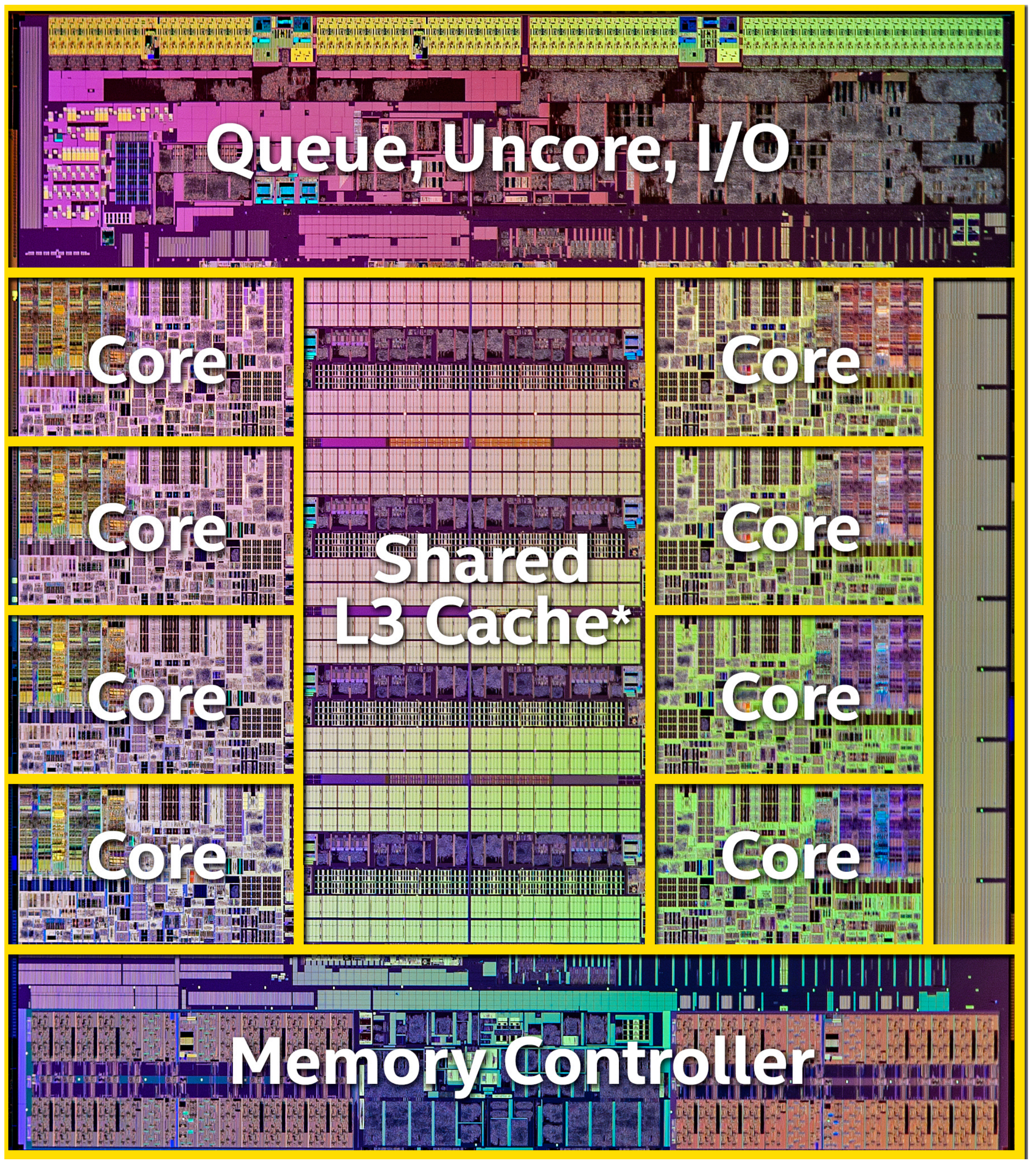


Multicore systems are grossly underutilized [1,2]

Extraction of parallelism fine-grained enough for multicore is notoriously hard [3]

[1] L. A. Barroso, U. Holzle. The Datacenter as a Computer, 2013
[2] C. Delimitrou and C. Kozyrakis. Quasar: Resource-Efficient and QoS-Aware Cluster Management, ASPLOS 2014
[3] P. Prabhu et al., A survey of the practice of computational science, SC ’11

Why Automatic Parallelization?



Multicore systems are grossly underutilized [1,2]

Extraction of parallelism fine-grained enough for multicore is notoriously hard [3]

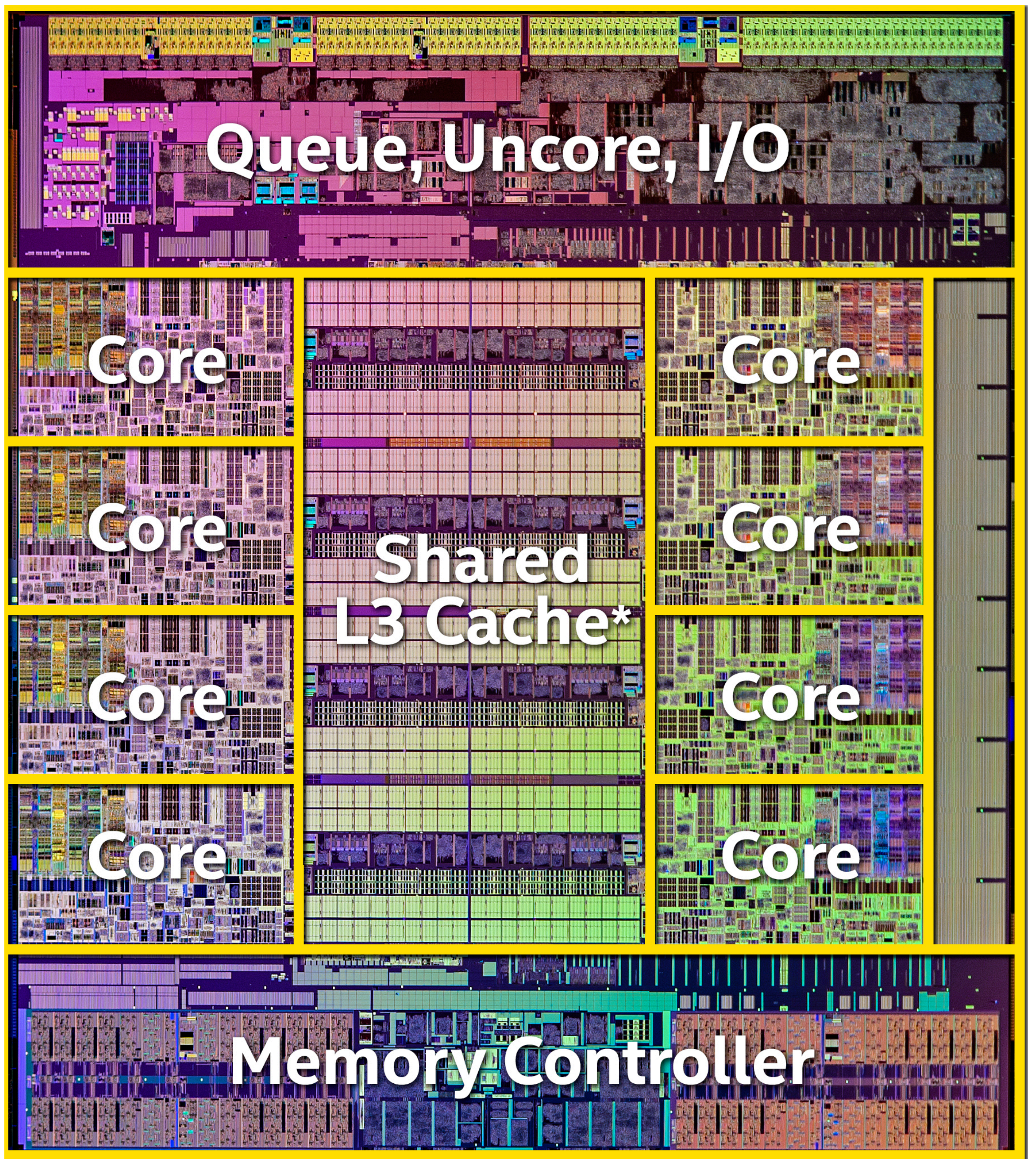
Programmers are mostly limited to coarse-grained parallelism (CGP)

[1] L. A. Barroso, U. Holzle. The Datacenter as a Computer, 2013

[2] C. Delimitrou and C. Kozyrakis. Quasar: Resource-Efficient and QoS-Aware Cluster Management, ASPLOS 2014

[3] P. Prabhu et al., A survey of the practice of computational science, SC '11

Why Automatic Parallelization?



Multicore systems are grossly underutilized [1,2]

Extraction of parallelism fine-grained enough for multicore is notoriously hard [3]

Programmers are mostly limited to coarse-grained parallelism (CGP)

CGP is ill-suited for multicore as it tends to stress multicore's shared resources

[1] L. A. Barroso, U. Holzle. The Datacenter as a Computer, 2013
[2] C. Delimitrou and C. Kozyrakis. Quasar: Resource-Efficient and QoS-Aware Cluster Management, ASPLOS 2014
[3] P. Prabhu et al., A survey of the practice of computational science, SC '11

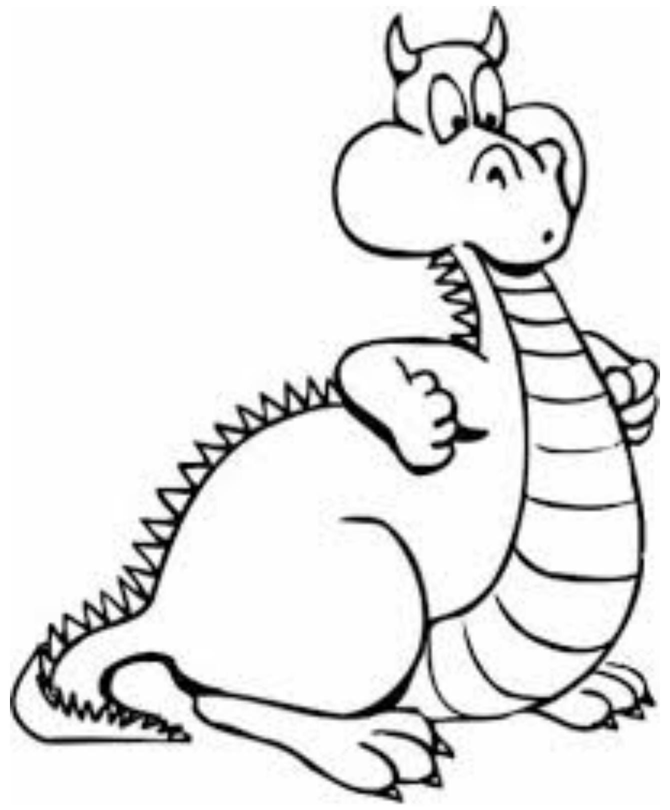
The Potential of Automatic Parallelization:

enable efficient use of multicore systems

The Potential of Automatic Parallelization: enable efficient use of multicore systems

Programmer

Sequential program



Parallelizing compiler

Parallel executable

Why Speculative Automatic Parallelization?

Why Speculative Automatic Parallelization?

For a long time, memory analysis
limited applicability of automatic parallelization

Why Speculative Automatic Parallelization?

For a long time, memory analysis
limited applicability of automatic parallelization

- undecidable in theory [Landi, LPLS'92]
  For any fixed analysis algorithm, there is a counter-example input for which the algorithm is imprecise.

Why Speculative Automatic Parallelization?

For a long time, memory analysis
limited applicability of automatic parallelization

- undecidable in theory [Landi, LPLS'92]
  For any fixed analysis algorithm, there is a counter-example input for which the algorithm is imprecise.

- insufficiently precise in practice [Hind, PASTE'01]
  especially for languages like C/C++.

Why Speculative Automatic Parallelization?

For a long time, memory analysis
limited applicability of automatic parallelization

- undecidable in theory [Landi, LPLS'92]
  For any fixed analysis algorithm, there is a counter-example input for which the algorithm is imprecise.
- insufficiently precise in practice [Hind, PASTE'01]
  especially for languages like C/C++.
- conservatively respects all possible inputs
  Many real dependences rarely occur in practice.

Why Speculative Automatic Parallelization?

For a long time, memory analysis
limited applicability of automatic parallelization

- undecidable in theory [Landi, LPLS'92]
  For any fixed analysis algorithm, there is a counter-example input for which the algorithm is imprecise.
- insufficiently precise in practice [Hind, PASTE'01]
  especially for languages like C/C++.
- conservatively respects all possible inputs
  Many real dependences rarely occur in practice.

Speculation overcame applicability limitations
by enabling optimization of the expected case

Outline

Why Speculative Automatic Parallelization?

State-of-the-art Approach

Inefficiencies of State-of-the-art

The *Perspective* Approach

Evaluation

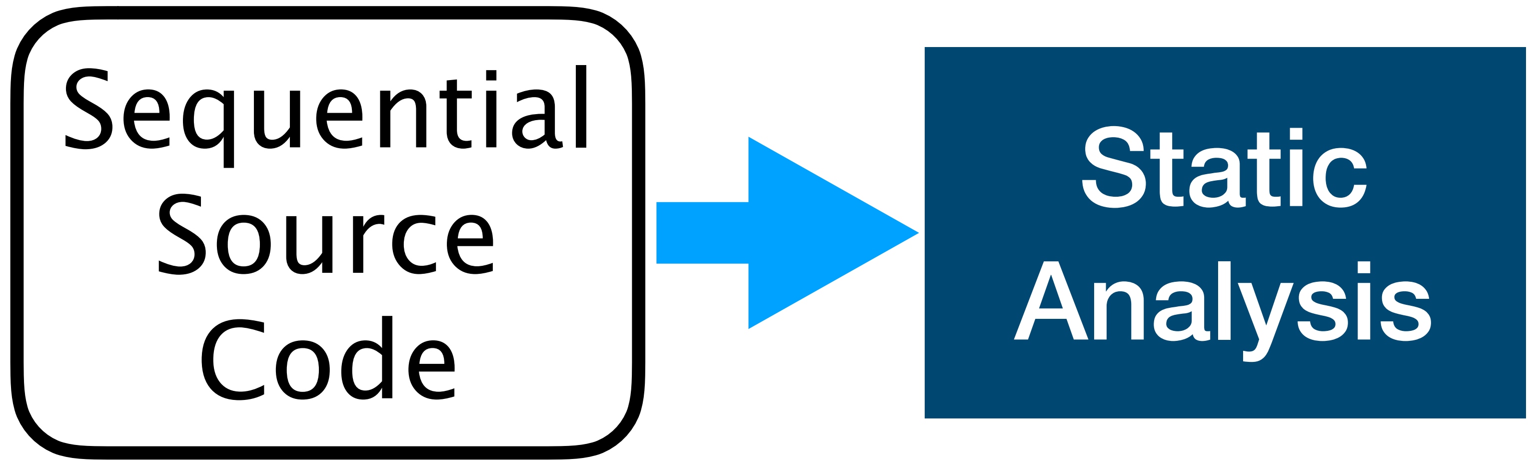
Conclusion

How to automatically parallelize?

State-of-the-art approach

How to automatically parallelize?

State-of-the-art approach

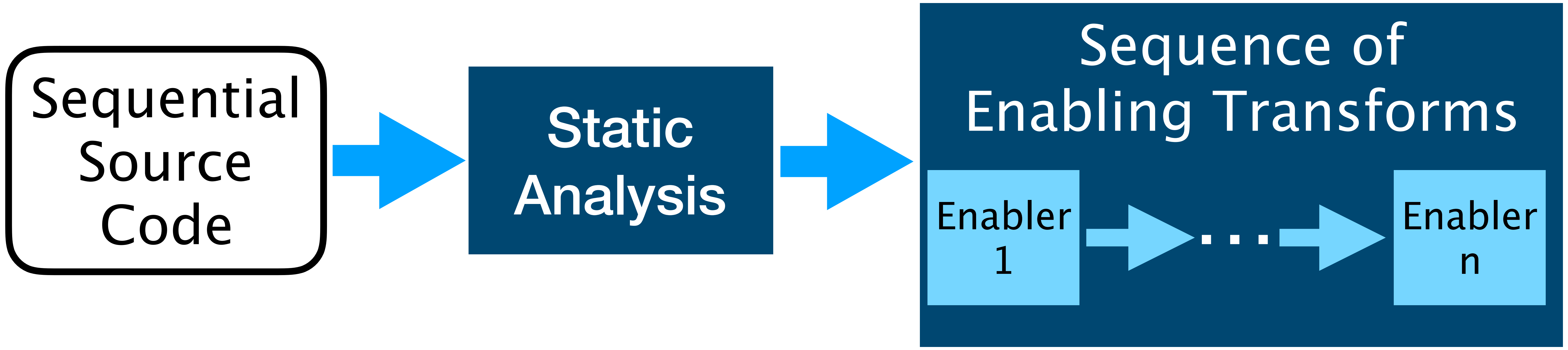


Memory Analysis[1]

[1] Johnson et al., CGO '17

How to automatically parallelize?

State-of-the-art approach



Memory Analysis[1]

(Speculative) Privatization[2,3]
(Speculative) Reduction[2,3]
Memory Speculation[4,5,6]
Control Speculation
Value Prediction

[1] Johnson et al., CGO '17 [2] Tu et al., LCPC '93 [3] Johnson et al., PLDI '12 [4] Mehrara et al., PLDI '09
[5] Tian et al., PLDI '10 [6] Kim et al., CGO '12

How to automatically parallelize?

State-of-the-art approach

Memory Analysis[1]

(Speculative) Privatization[2,3]
(Speculative) Reduction[2,3]
Memory Speculation[4,5,6]
Control Speculation
Value Prediction

DOALL
PS-DSWP[7]
HELIX[8]

[1] Johnson et al., CGO '17
[2] Tu et al., LCPC '93
[3] Johnson et al., PLDI '12
[4] Mehrara et al., PLDI '09
[5] Tian et al., PLDI '10
[6] Kim et al., CGO '12
[7] Raman et al., CGO '08
[8] Campanoni et al., CGO '12

How to automatically parallelize?

State-of-the-art approach

Memory Analysis[1]

(Speculative) Privatization[2,3]
(Speculative) Reduction[2,3]
Memory Speculation[4,5,6]
Control Speculation
Value Prediction

DOALL
PS-DSWP[7]
HELIX[8]

[1] Johnson et al., CGO '17
[2] Tu et al., LCPC '93
[3] Johnson et al., PLDI '12
[4] Mehrara et al., PLDI '09
[5] Tian et al., PLDI '10
[6] Kim et al., CGO '12
[7] Raman et al., CGO '08
[8] Campanoni et al., CGO '12

How to automatically parallelize?

State-of-the-art approach

Memory Analysis[1]

(Speculative) Privatization[2,3]
(Speculative) Reduction[2,3]
Memory Speculation[4,5,6]
Control Speculation
Value Prediction

DOALL
PS-DSWP[7]
HELIX[8]

1 Johnson et al., CGO '17
2 Tu et al., LCPC '93
3 Johnson et al., PLDI '12
4 Mehrara et al., PLDI '09
5 Tian et al., PLDI '10
6 Kim et al., CGO '12
7 Raman et al., CGO '08
8 Campanoni et al., CGO '12

How to automatically parallelize?

State-of-the-art approach

costs often negate parallelization benefits

Memory Analysis[1]

(Speculative) Privatization[2,3]
(Speculative) Reduction[2,3]
Memory Speculation[4,5,6]
Control Speculation
Value Prediction

DOALL
PS-DSWP[7]
HELIX[8]

1 Johnson et al., CGO '17
2 Tu et al., LCPC '93
3 Johnson et al., PLDI '12
4 Mehrara et al., PLDI '09
5 Tian et al., PLDI '10
6 Kim et al., CGO '12
7 Raman et al., CGO '08
8 Campanoni et al., CGO '12

The most applicable prior automatic speculative DOALL system is Privateer*

* Nick P. Johnson et al., Speculative Separation for Privatization and Reductions in PLDI ‘12

The most applicable prior automatic speculative DOALL system is Privateer*

two identified inefficiencies

* Nick P. Johnson et al., Speculative Separation for Privatization and Reductions in PLDI '12

The most applicable prior automatic speculative DOALL system is Privateer*

two identified inefficiencies

Excessive use of memory speculation

Very expensive to validate due to costly communication and bookkeeping for each speculated dependence

* Nick P. Johnson et al., Speculative Separation for Privatization and Reductions in PLDI '12

The most applicable prior automatic speculative DOALL system is Privateer*

two identified inefficiencies

Excessive use of memory speculation

Very expensive to validate due to costly communication and bookkeeping for each speculated dependence

Expensive speculative privatization

Monitor large write sets to correctly merge private memory states of parallel workers

* Nick P. Johnson et al., Speculative Separation for Privatization and Reductions in PLDI ‘12

Inefficiencies of state-of-the-art:

Overuse of expensive-to-validate memory speculation

```
   for (i=0; i<N; ++i) {
     …
     if (observed_always_true)
i1:    *ptr = …
     …
i2:  … = … + *ptr
   }
```

Simplified example from the dijkstra benchmark (MiBench)

- branch condition cannot be statically proven true
- **ptr** is not modified within the loop

Inefficiencies of state-of-the-art:

Overuse of expensive-to-validate memory speculation

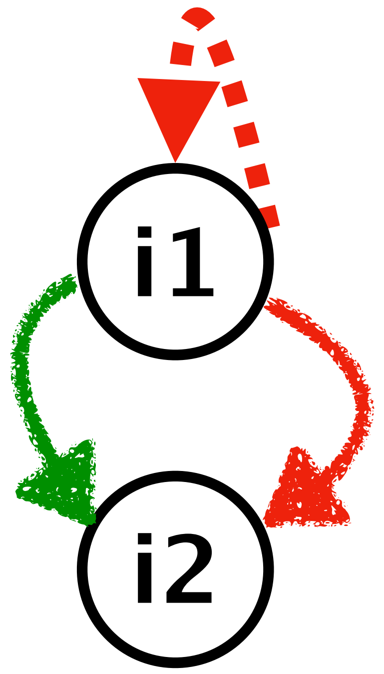
```
   for (i=0; i<N; ++i) {
     …
     if (observed_always_true)
i1:    *ptr = …
     …
i2:  … = … + *ptr
   }
```

Simplified example from the dijkstra benchmark (MiBench)

- branch condition cannot be statically proven true
- **ptr** is not modified within the loop

Program Dependence Graph (PDG)

Cross-iter RAW Dep
Cross-iter WAW Dep
Intra-iter RAW Dep



Inefficiencies of state-of-the-art:

Overuse of expensive-to-validate memory speculation

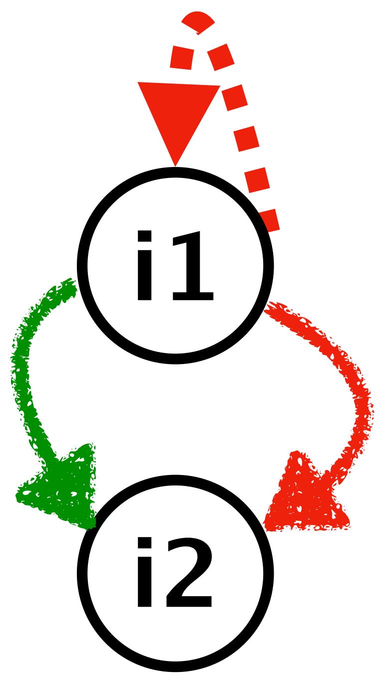
```
   for (i=0; i<N; ++i) {
     …
     if (observed_always_true)
i1:    *ptr = …
     …
i2:  … = … + *ptr
   }
```

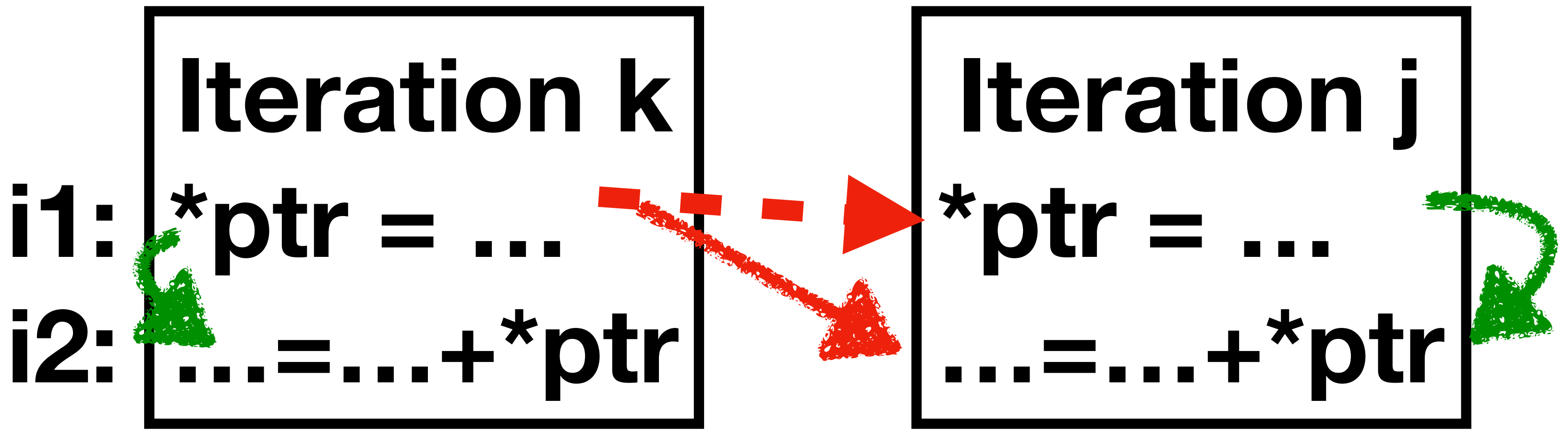
Simplified example from the dijkstra benchmark (MiBench)

- branch condition cannot be statically proven true
- **ptr** is not modified within the loop

Program Dependence Graph (PDG)

Cross-iter RAW Dep
Cross-iter WAW Dep
Intra-iter RAW Dep





Inefficiencies of state-of-the-art:

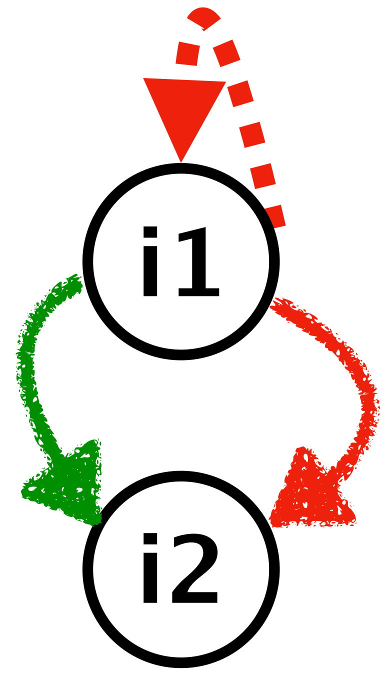
Overuse of expensive-to-validate memory speculation

```
   for (i=0; i<N; ++i) {
     …
     if (observed_always_true)
i1:     *ptr = …
     …
i2:  … = … + *ptr
   }
```

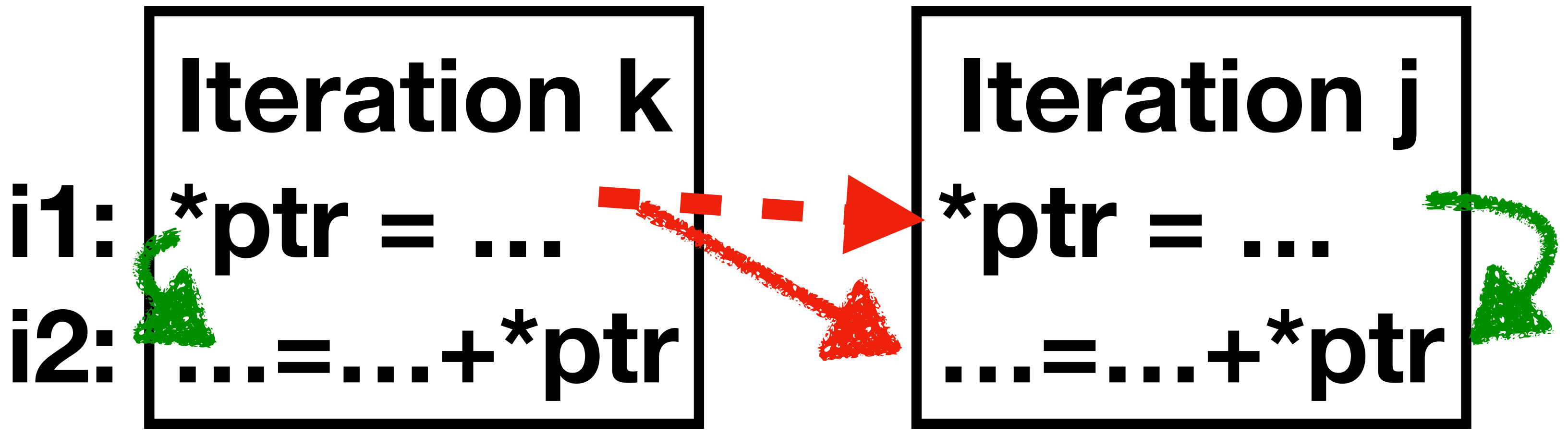
Simplified example from the dijkstra benchmark (MiBench)

- branch condition cannot be statically proven true
- **ptr** is not modified within the loop

Program Dependence Graph (PDG)



Cross-iter RAW Dep
Cross-iter WAW Dep
Intra-iter RAW Dep



DOALL parallelization applicability criterion:
No cross-iteration dependences

Inefficiencies of state-of-the-art:

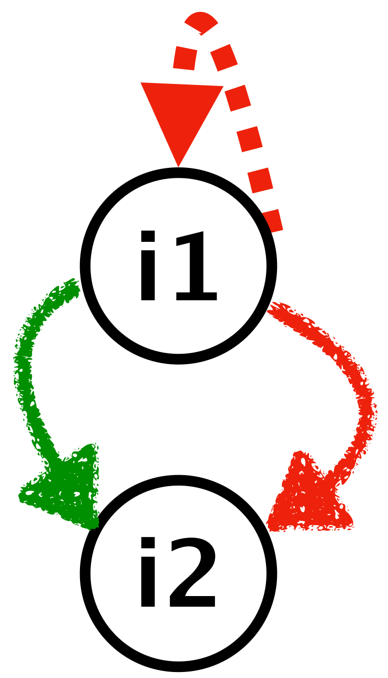
Overuse of expensive-to-validate memory speculation

```
   for (i=0; i<N; ++i) {
     …
     if (observed_always_true)
i1:    *ptr = …
     …
i2:  … = … + *ptr
   }
```

Simplified example from the dijkstra benchmark (MiBench)

- branch condition cannot be statically proven true
- **ptr** is not modified within the loop

Relaxing Program Dependence Graph (PDG)



Inefficiencies of state-of-the-art:

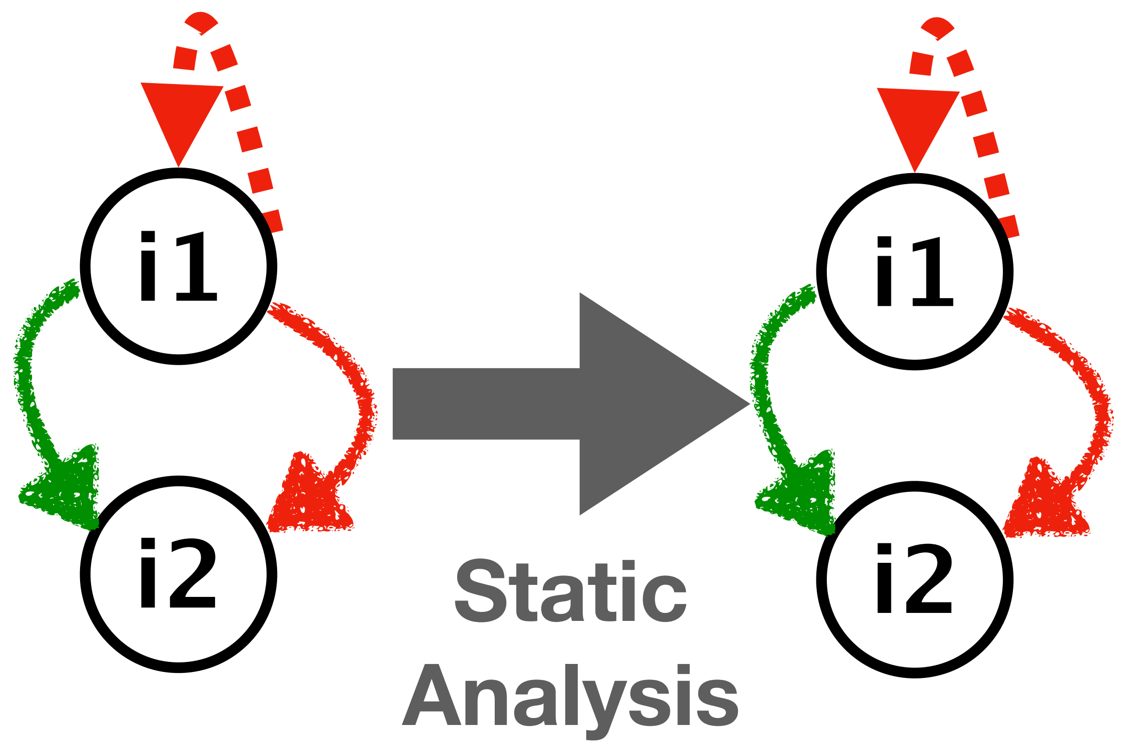
Overuse of expensive-to-validate memory speculation

```
   for (i=0; i<N; ++i) {
     …
     if (observed_always_true)
i1:    *ptr = …
     …
i2:  … = … + *ptr
   }
```

Simplified example from the dijkstra benchmark (MiBench)

- branch condition cannot be statically proven true
- **ptr** is not modified within the loop

Relaxing Program Dependence Graph (PDG)



Inefficiencies of state-of-the-art:

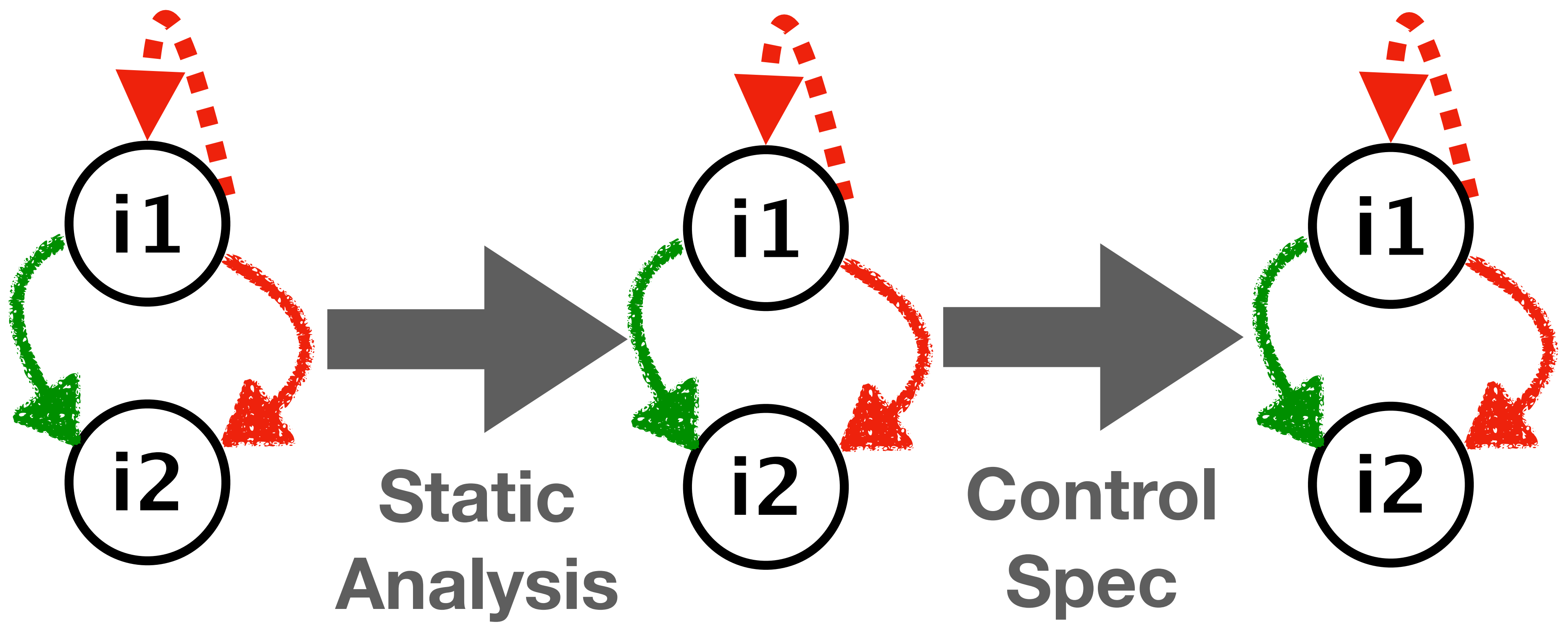
Overuse of expensive-to-validate memory speculation

```
   for (i=0; i<N; ++i) {
     …
     if (observed_always_true)
i1:    *ptr = …
     …
i2:  … = … + *ptr
   }
```

Simplified example from the dijkstra benchmark (MiBench)

- branch condition cannot be statically proven true
- **ptr** is not modified within the loop

Relaxing Program Dependence Graph (PDG)



Cross-iter RAW Dep
Cross-iter WAW Dep
Intra-iter RAW Dep

Inefficiencies of state-of-the-art:

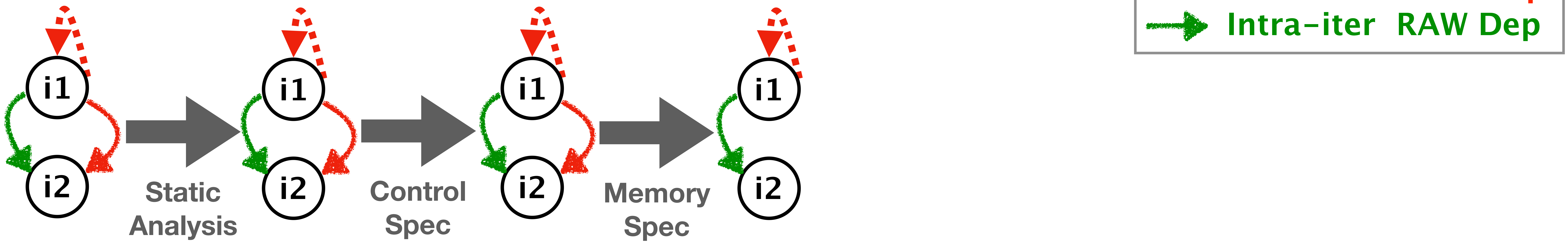
Overuse of expensive-to-validate memory speculation

```
   for (i=0; i<N; ++i) {
     …
     if (observed_always_true)
i1:    *ptr = …
     …
i2:  … = … + *ptr
   }
```

Simplified example from the dijkstra benchmark (MiBench)

- branch condition cannot be statically proven true
- **ptr** is not modified within the loop

Relaxing Program Dependence Graph (PDG)



Inefficiencies of state-of-the-art:

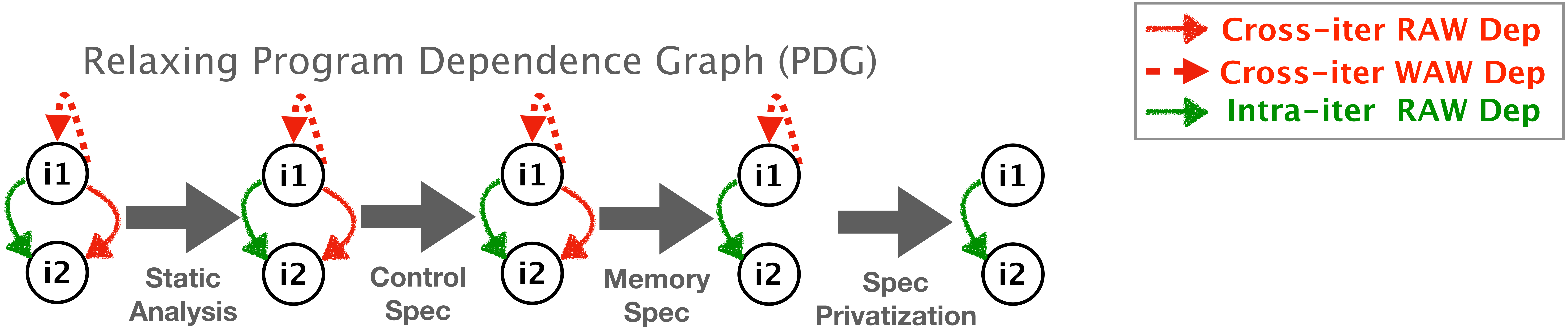
Overuse of expensive-to-validate memory speculation

```
   for (i=0; i<N; ++i) {
     …
     if (observed_always_true)
i1:    *ptr = …
     …
i2:  … = … + *ptr
   }
```

Simplified example from the dijkstra benchmark (MiBench)

- branch condition cannot be statically proven true
- **ptr** is not modified within the loop

Relaxing Program Dependence Graph (PDG)



Inefficiencies of state-of-the-art:

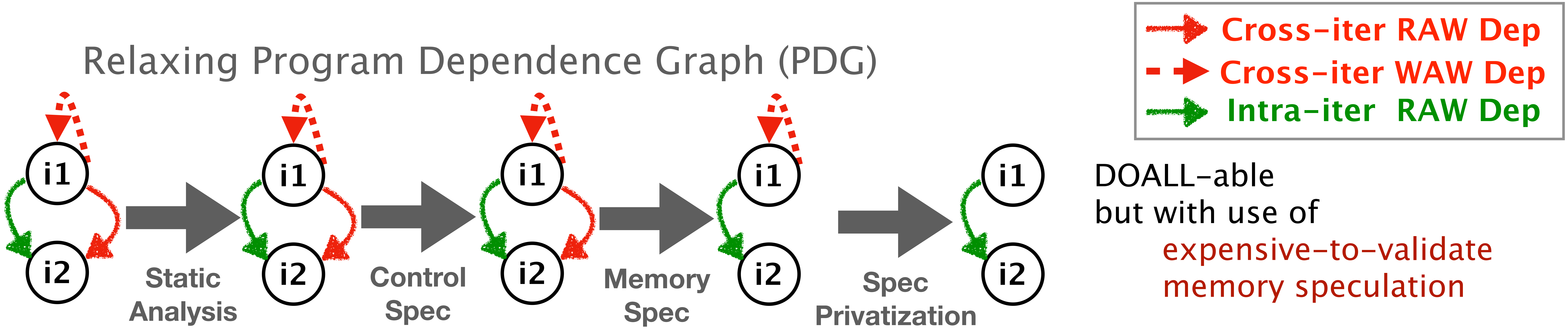
Overuse of expensive-to-validate memory speculation

```
   for (i=0; i<N; ++i) {
     …
     if (observed_always_true)
i1:    *ptr = …
     …
i2:  … = … + *ptr
   }
```

Simplified example from
the dijkstra benchmark (MiBench)

- branch condition cannot be statically proven true
- **ptr** is not modified within the loop

Relaxing Program Dependence Graph (PDG)



DOALL-able
but with use of
expensive-to-validate
memory speculation

Inefficiencies of state-of-the-art:

Overuse of expensive-to-validate memory speculation

```
  for (i=0; i<N; ++i) {
    …
    if (observed_always_true) {
      spec_write(ptr)
i1:   *ptr = …
    }
    …
    spec_read(ptr)
i2: … = … + *ptr
  }
```

Time

Worker 1 **Worker 2** **Validator**

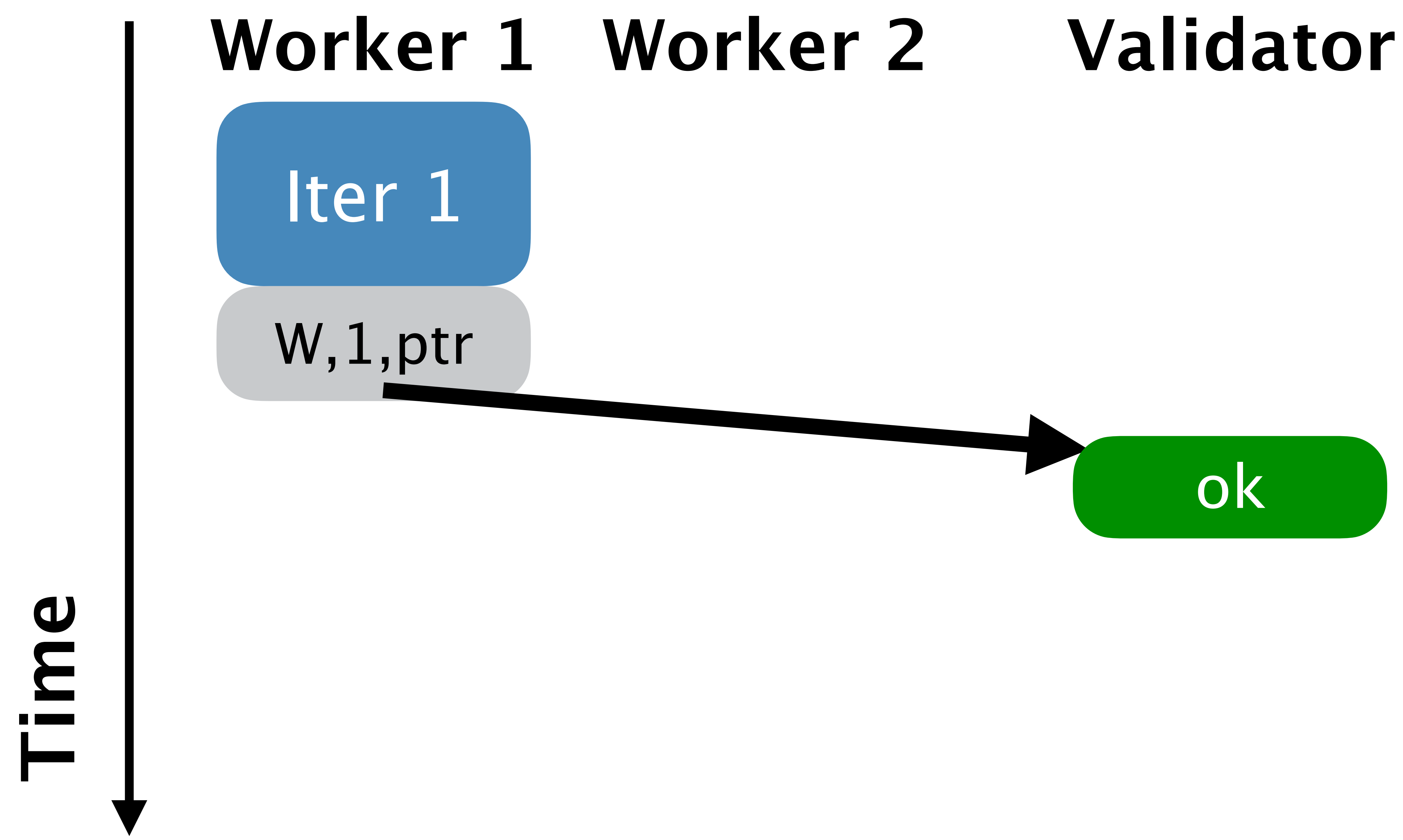
Monitoring Overhead

Inefficiencies of state-of-the-art:

Overuse of expensive-to-validate memory speculation

```
   for (i=0; i<N; ++i) {
     …
     if (observed_always_true) {
       spec_write(ptr)
i1:    *ptr = …
     }
     …
     spec_read(ptr)
i2:  … = … + *ptr
   }
```

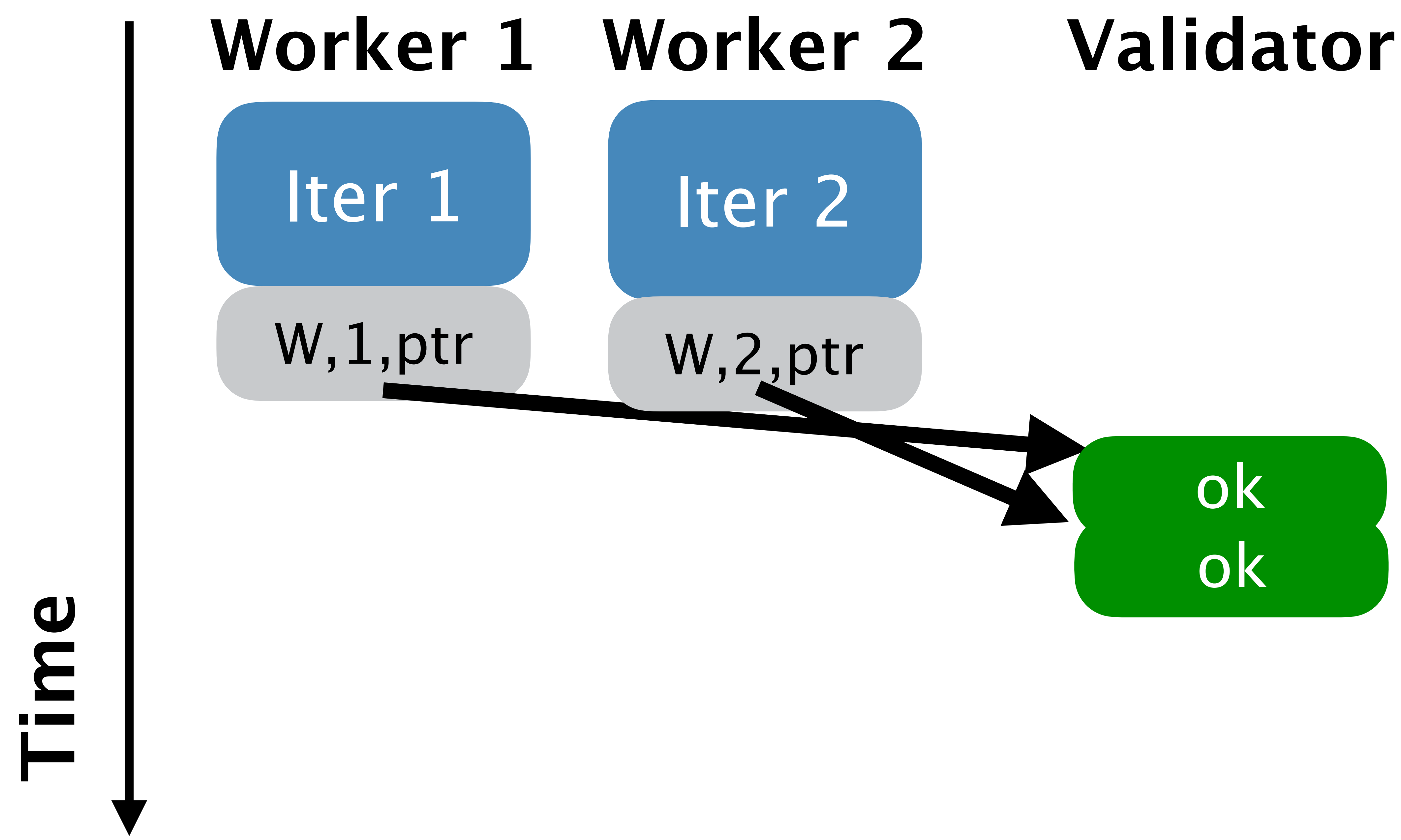
Monitoring Overhead



Inefficiencies of state-of-the-art:

Overuse of expensive-to-validate memory speculation

```
   for (i=0; i<N; ++i) {
     …
     if (observed_always_true) {
       spec_write(ptr)
i1:    *ptr = …
     }
     …
     spec_read(ptr)
i2:  … = … + *ptr
   }
```



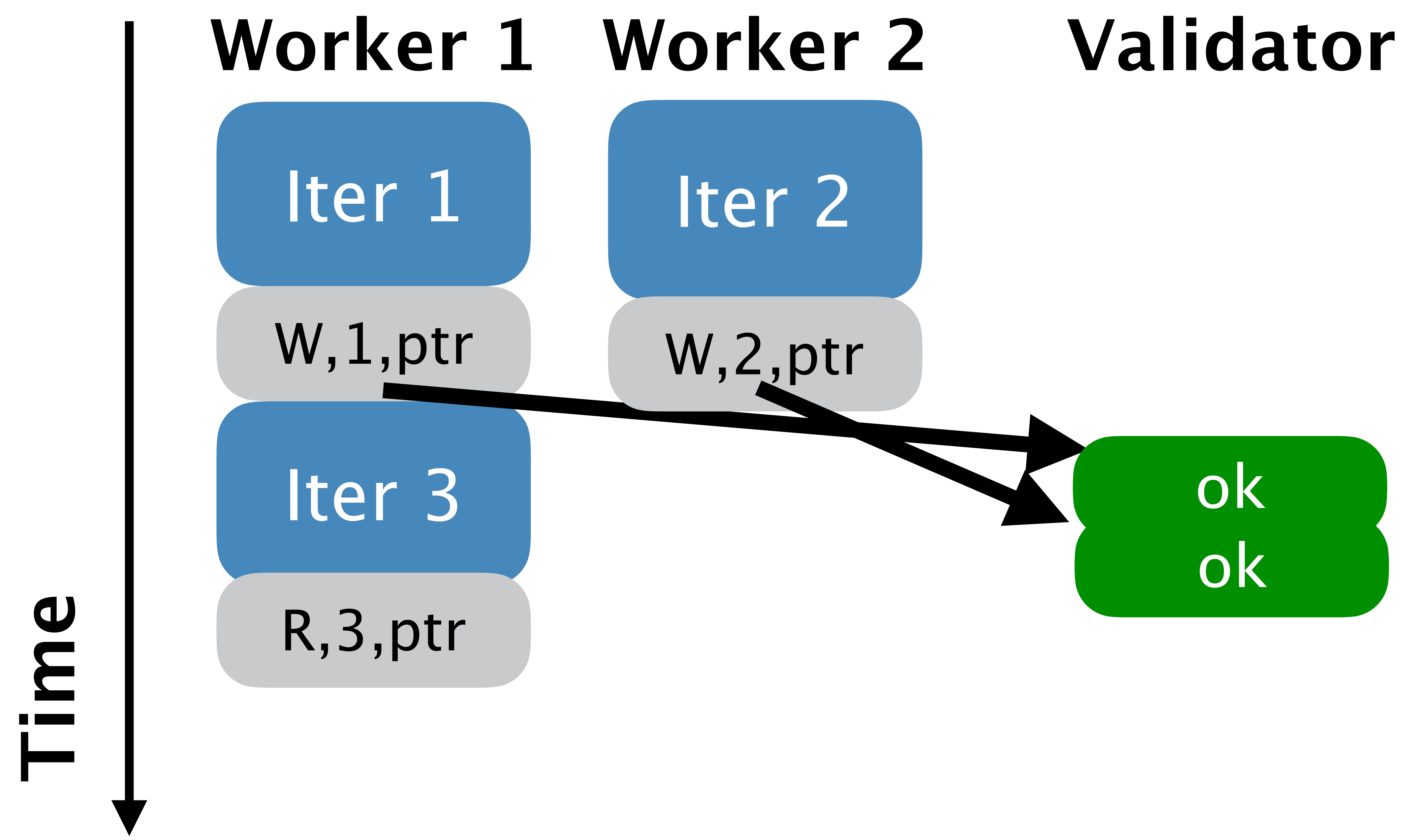
Monitoring Overhead

Inefficiencies of state-of-the-art:

Overuse of expensive-to-validate memory speculation

```
   for (i=0; i<N; ++i) {
      …
      if (observed_always_true) {
         spec_write(ptr)
i1:      *ptr = …
      }
      …
      spec_read(ptr)
i2:   … = … + *ptr
   }
```

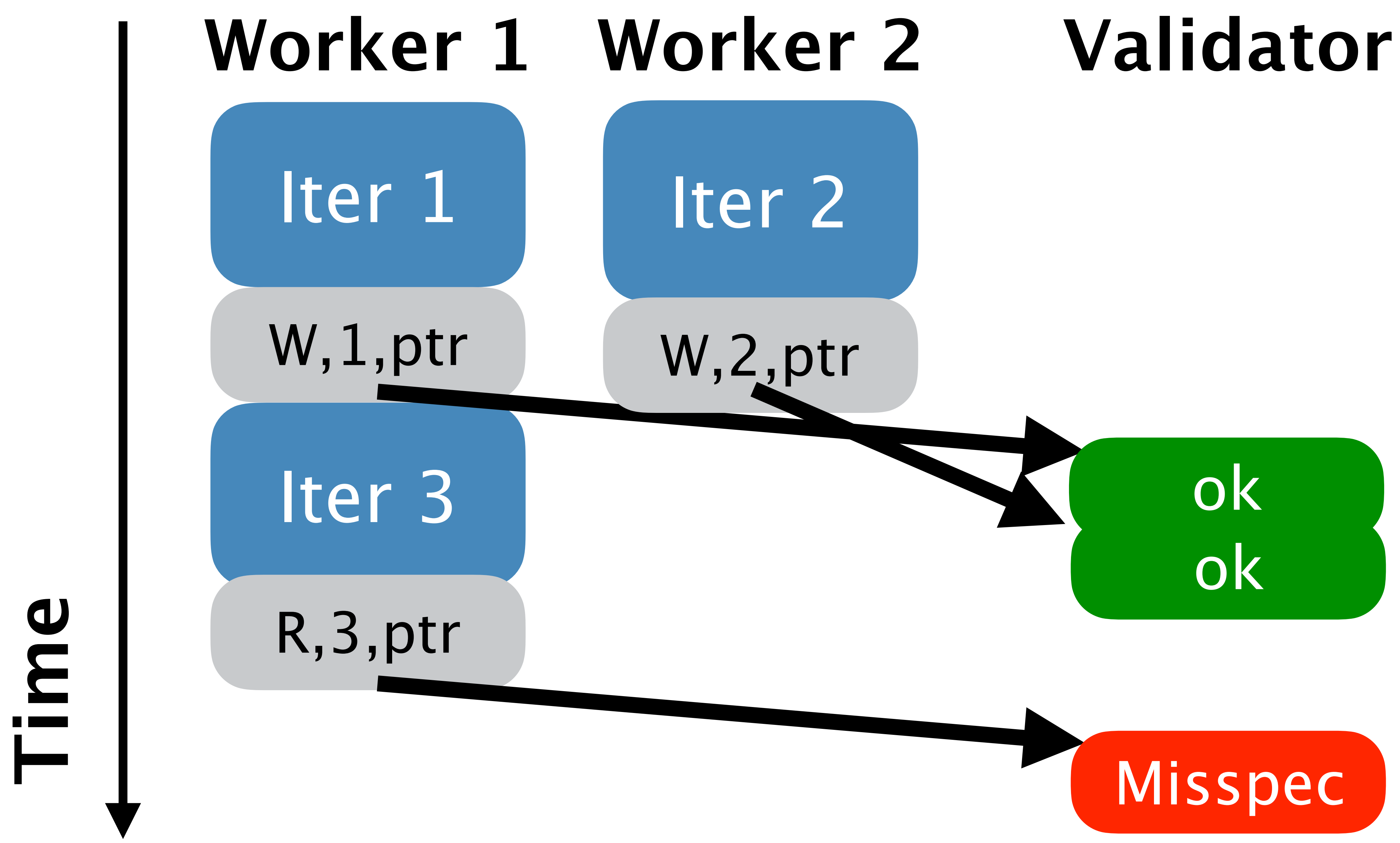
Monitoring Overhead



Inefficiencies of state-of-the-art:

Overuse of expensive-to-validate memory speculation

```
   for (i=0; i<N; ++i) {
     …
     if (observed_always_true) {
       spec_write(ptr)
i1:    *ptr = …
     }
     …
     spec_read(ptr)
i2:  … = … + *ptr
   }
```



Monitoring Overhead

Inefficiencies of state-of-the-art:

Expensive speculative privatization

```
   for (i=0; i<N; ++i) {
     …
     if (observed_always_true)
i1:    *ptr = …
     …
i2:  … = … + *ptr
   }
```

Assumptions

- branch condition **statically proven** true
- **ptr** is not modified within the loop

Inefficiencies of state-of-the-art:

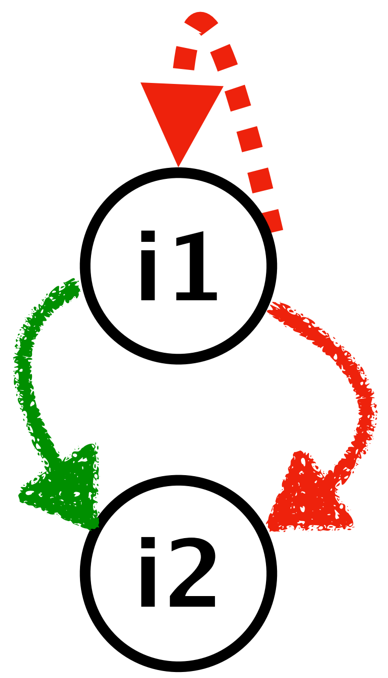
Expensive speculative privatization

```
   for (i=0; i<N; ++i) {
     …
     if (observed_always_true)
i1:    *ptr = …
     …
i2:  … = … + *ptr
   }
```

Assumptions

- branch condition **statically proven** true
- **ptr** is not modified within the loop

Relaxing Program Dependence Graph (PDG)



Cross-iter RAW Dep
Cross-iter WAW Dep
Intra-iter RAW Dep

Inefficiencies of state-of-the-art:

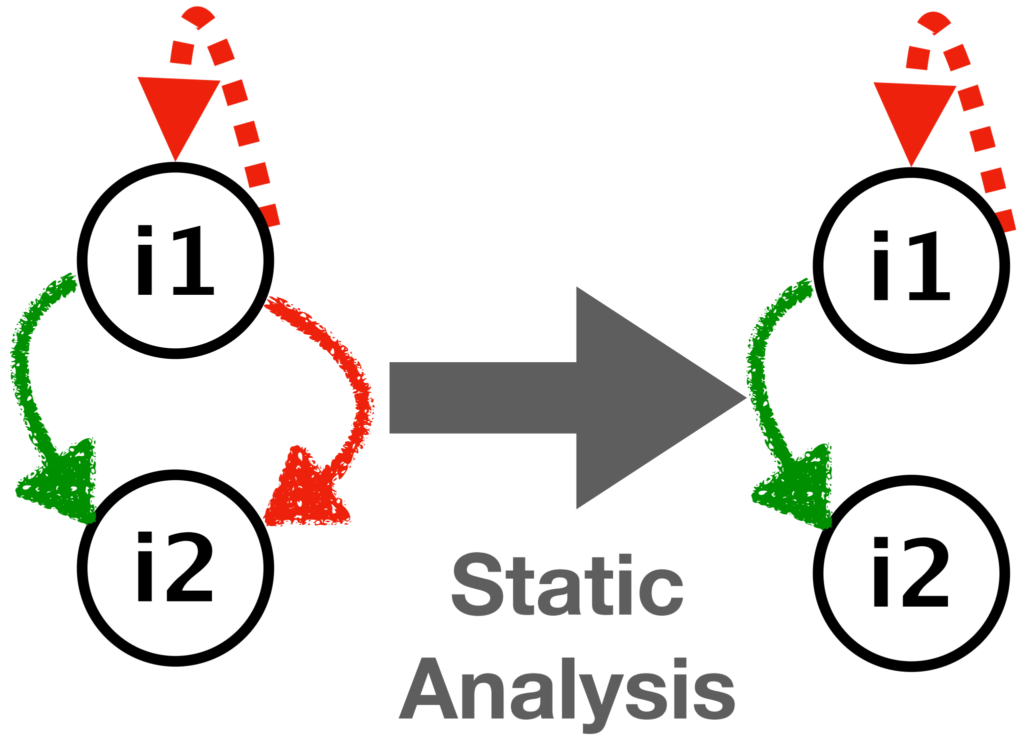
Expensive speculative privatization

```
   for (i=0; i<N; ++i) {
     …
     if (observed_always_true)
i1:    *ptr = …
     …
i2:  … = … + *ptr
   }
```

Assumptions

- branch condition **statically proven** true
- **ptr** is not modified within the loop

Relaxing Program Dependence Graph (PDG)



Inefficiencies of state-of-the-art:

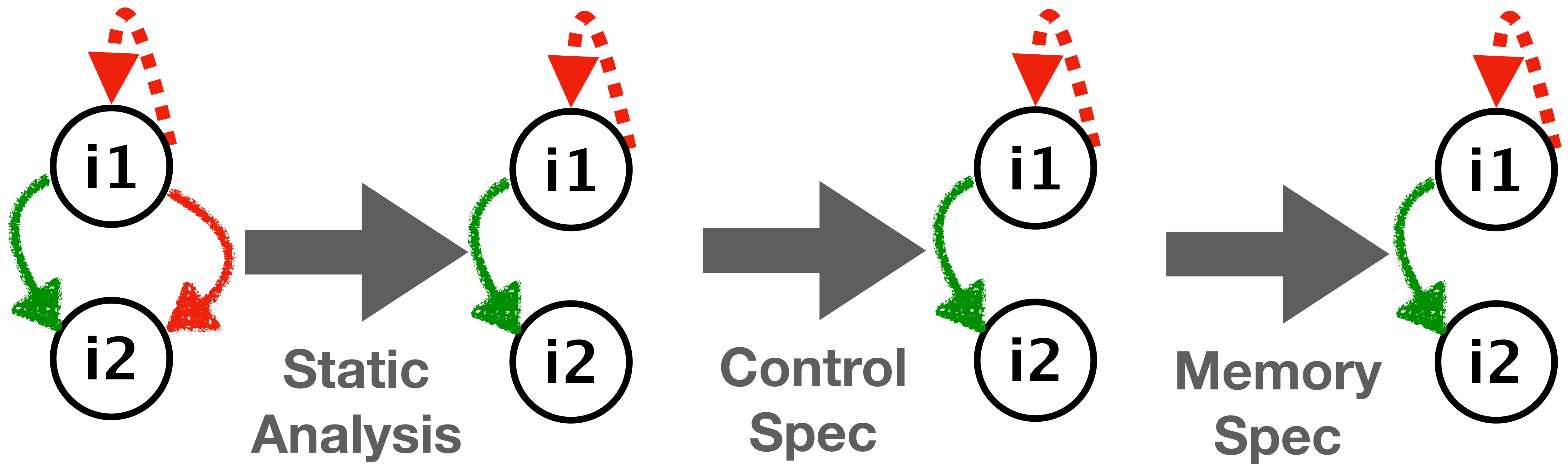
Expensive speculative privatization

```
   for (i=0; i<N; ++i) {
     …
     if (observed_always_true)
i1:     *ptr = …
     …
i2:  … = … + *ptr
   }
```

Assumptions

– branch condition **statically proven** true
– **ptr** is not modified within the loop

Relaxing Program Dependence Graph (PDG)



Inefficiencies of state-of-the-art:

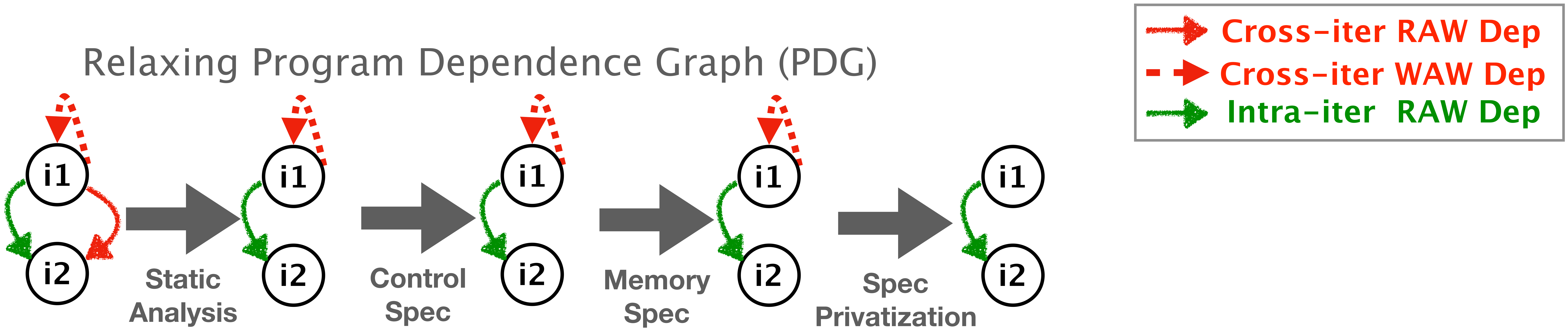
Expensive speculative privatization

```
   for (i=0; i<N; ++i) {
     …
     if (observed_always_true)
i1:      *ptr = …
     …
i2:  … = … + *ptr
   }
```

Assumptions

- branch condition **statically proven** true
- **ptr** is not modified within the loop

Relaxing Program Dependence Graph (PDG)



Inefficiencies of state-of-the-art:

Expensive speculative privatization

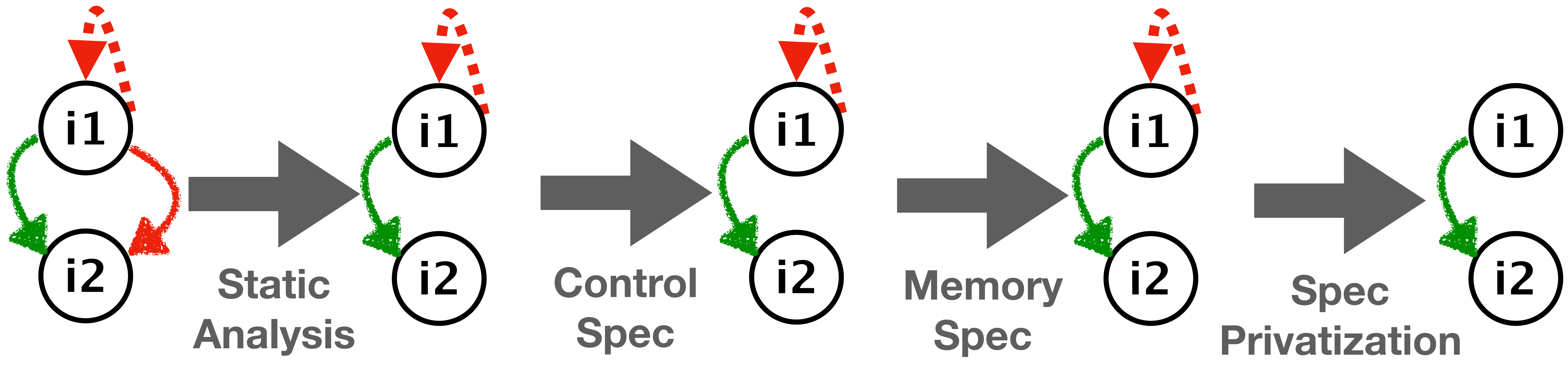
```
   for (i=0; i<N; ++i) {
     …
     if (observed_always_true)
i1:     *ptr = …
     …
i2:  … = … + *ptr
   }
```

Assumptions

- branch condition **statically proven** true
- **ptr** is not modified within the loop

Cross-iter RAW Dep
Cross-iter WAW Dep
Intra-iter RAW Dep

Relaxing Program Dependence Graph (PDG)



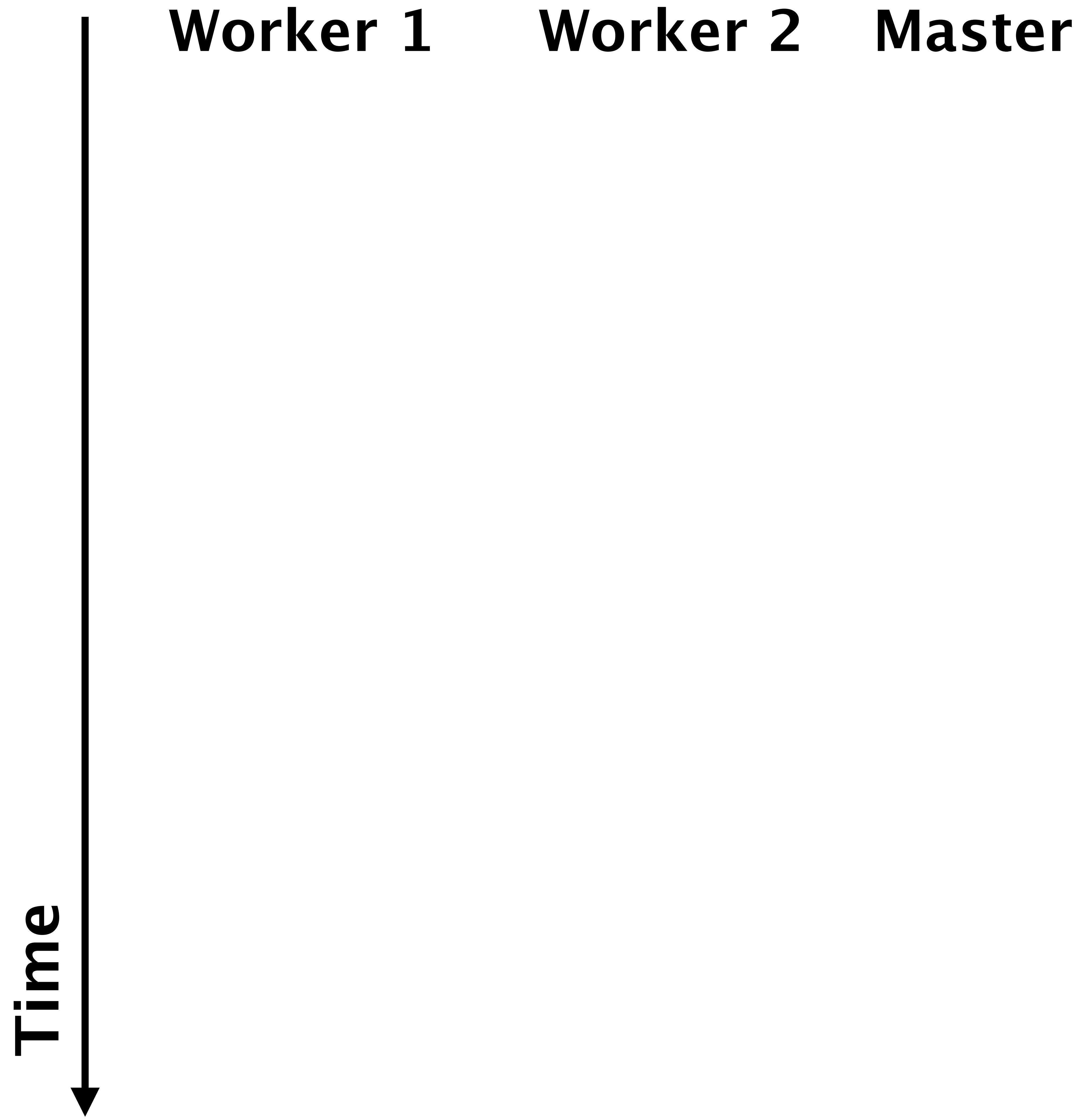
DOALL-able but
expensive
write monitoring
used for live-out state

Inefficiencies of state-of-the-art:

Expensive speculative privatization

```
   for (i=0; i<N; ++i) {
     …
     if (observed_always_true) {
i1:    spec_write(ptr)
       *ptr = …
     }
    …
i2: … = … + *ptr
   }
```

Monitoring Overhead

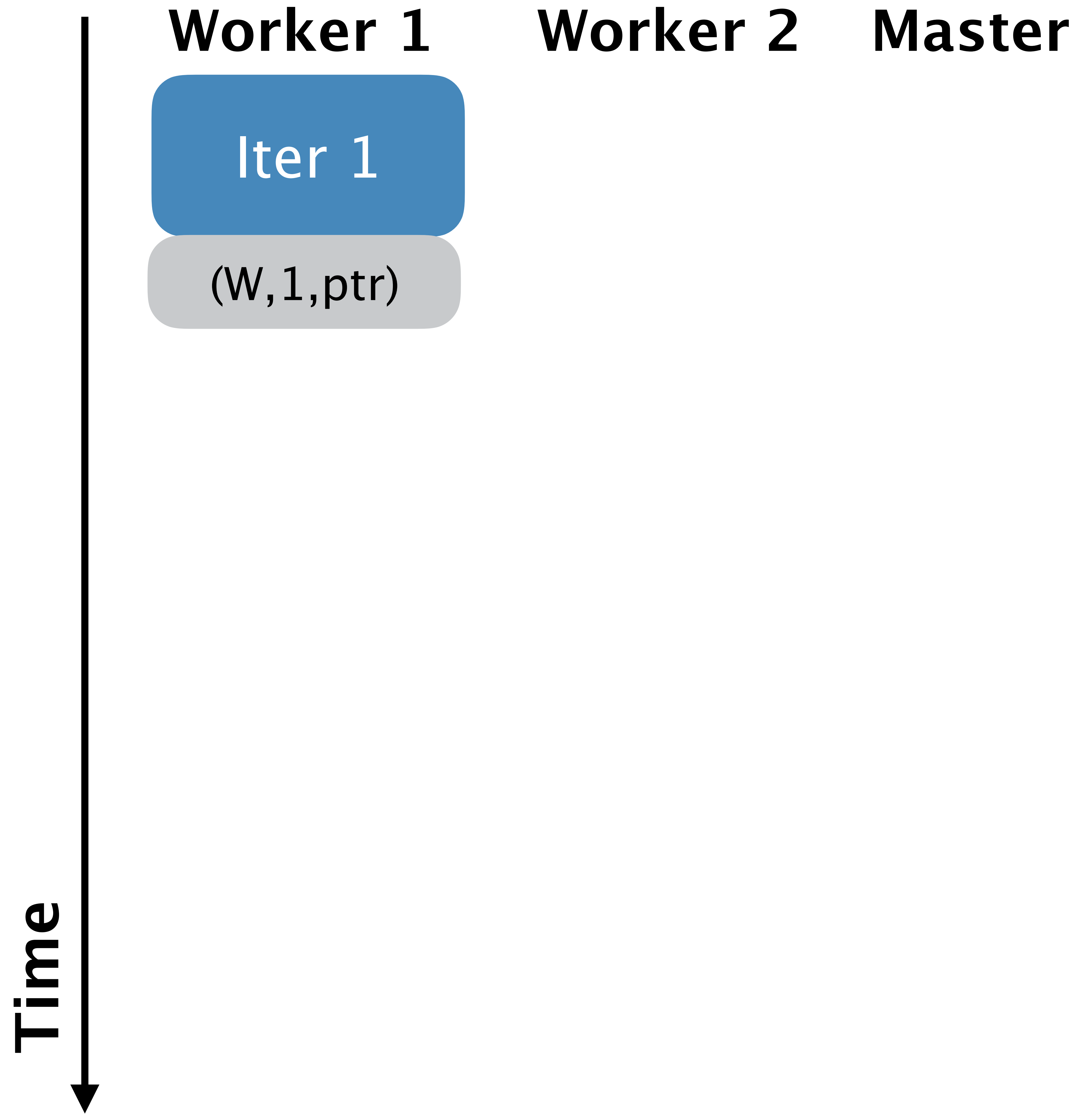


Inefficiencies of state-of-the-art:

Expensive speculative privatization

```
   for (i=0; i<N; ++i) {
     …
     if (observed_always_true) {
i1:    spec_write(ptr)
       *ptr = …
     }
    …
i2: … = … + *ptr
   }
```

Monitoring Overhead

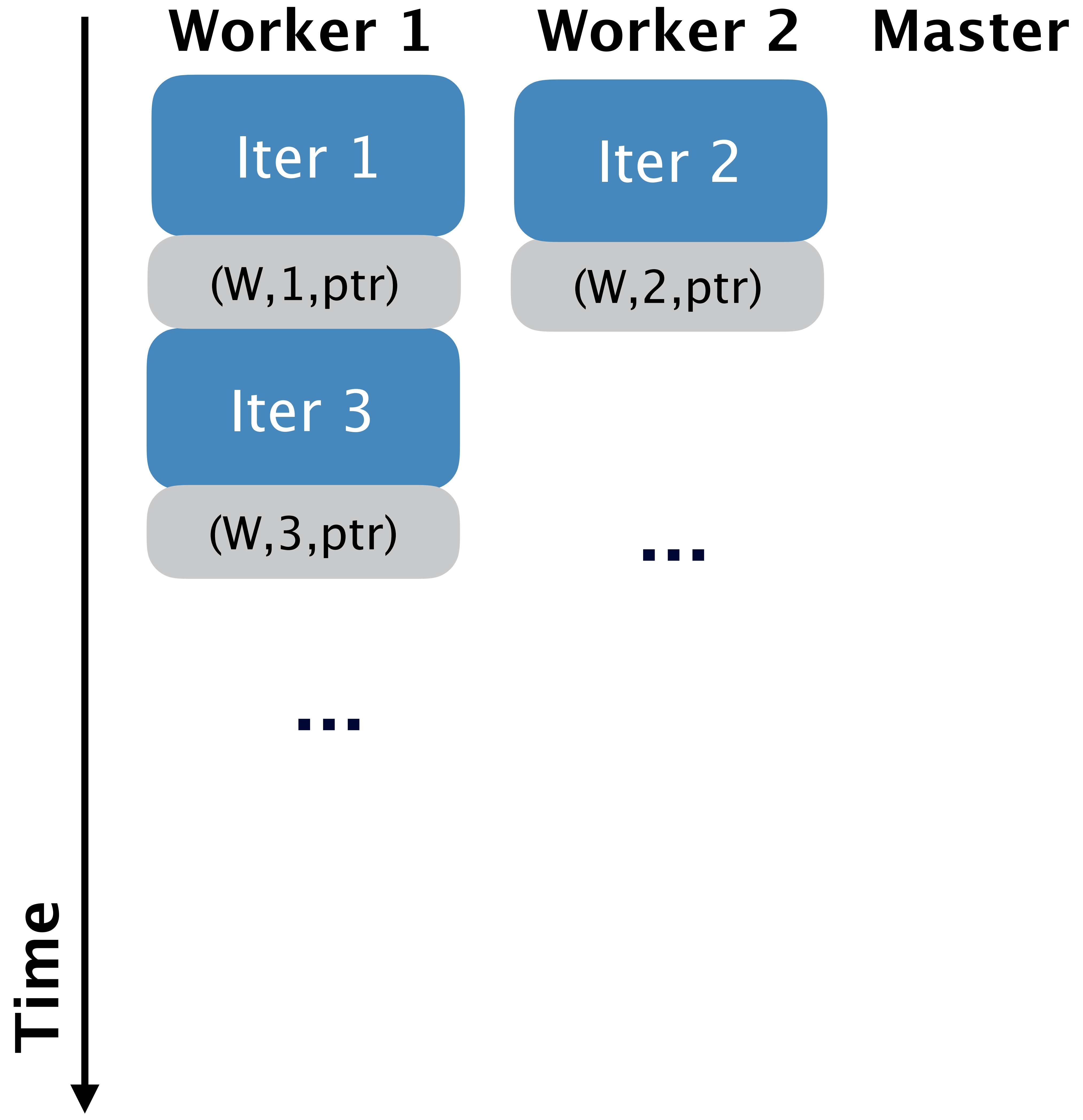


Inefficiencies of state-of-the-art:

Expensive speculative privatization

```
   for (i=0; i<N; ++i) {
     …
     if (observed_always_true) {
i1:     spec_write(ptr)
        *ptr = …
     }
    …
i2: … = … + *ptr
   }
```

Monitoring Overhead

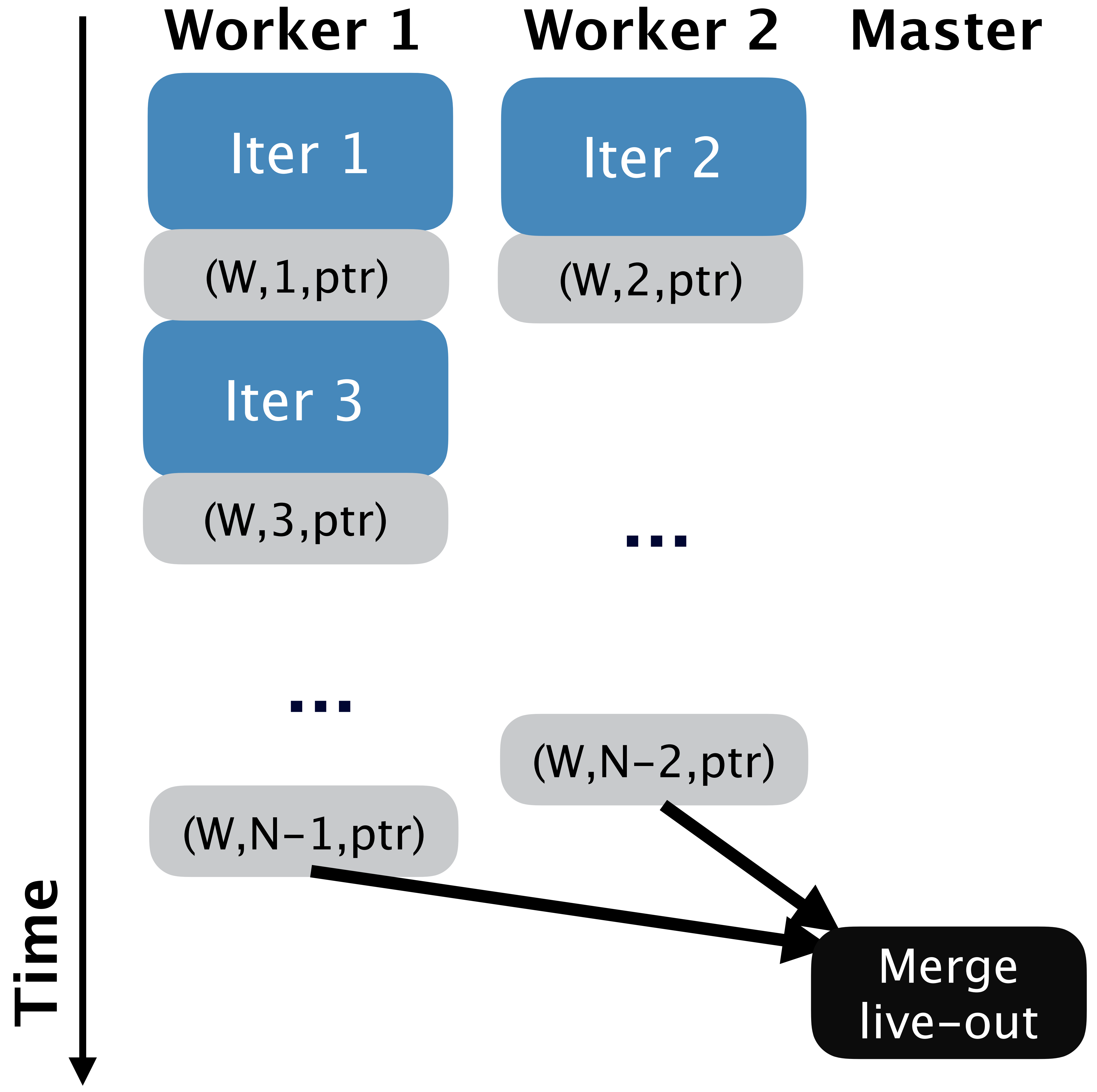


Inefficiencies of state-of-the-art:

Expensive speculative privatization

```
   for (i=0; i<N; ++i) {
     …
     if (observed_always_true) {
i1:     spec_write(ptr)
        *ptr = …
     }
    …
i2: … = … + *ptr
   }
```

Monitoring Overhead



Parallelization of dijkstra benchmark (MiBench) with Privateer*

Required monitoring of
973GB of reads & 649GB of writes
for an input graph of 3K nodes!
$O(N^3)$, where N is # of nodes

* Nick P. Johnson et al., Speculative Separation for Privatization and Reductions in PLDI '12

Outline

Why Speculative Automatic Parallelization?

State-of-the-art Approach

Inefficiencies of State-of-the-art

The *Perspective* Approach

Evaluation

Conclusion

Maintain the applicability of prior speculative automatic parallelization systems without unnecessary overheads

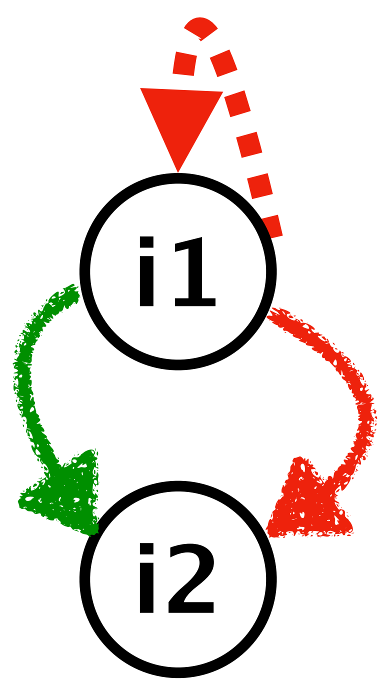
Fully leverage inexpensive speculative assertions to efficiently break dependences

```
    for (i=0; i<N; ++i) {
      …
      if (observed_always_true)
i1:     *ptr = …
      …
i2:   … = … + *ptr
    }
```

Simplified example from the dijkstra benchmark (MiBench)

- branch condition cannot be statically proven true
- **ptr** is not modified within the loop

Program Dependence Graph (PDG)



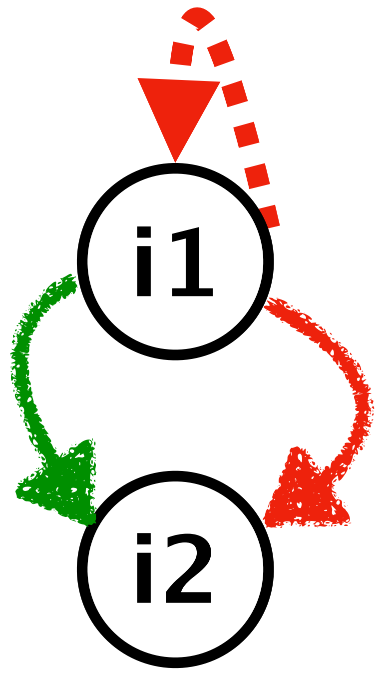
Fully leverage inexpensive speculative assertions to efficiently break dependences

```
   for (i=0; i<N; ++i) {
     …
     if (observed_always_true)
i1:     *ptr = …
     …
i2:  … = … + *ptr
   }
```

Simplified example from
the dijkstra benchmark (MiBench)

- branch condition cannot be statically proven true
- **ptr** is not modified within the loop

Program Dependence Graph (PDG)



Cross-iter RAW Dep
Cross-iter WAW Dep
Intra-iter RAW Dep

Fully leverage inexpensive speculative assertions to efficiently break dependences

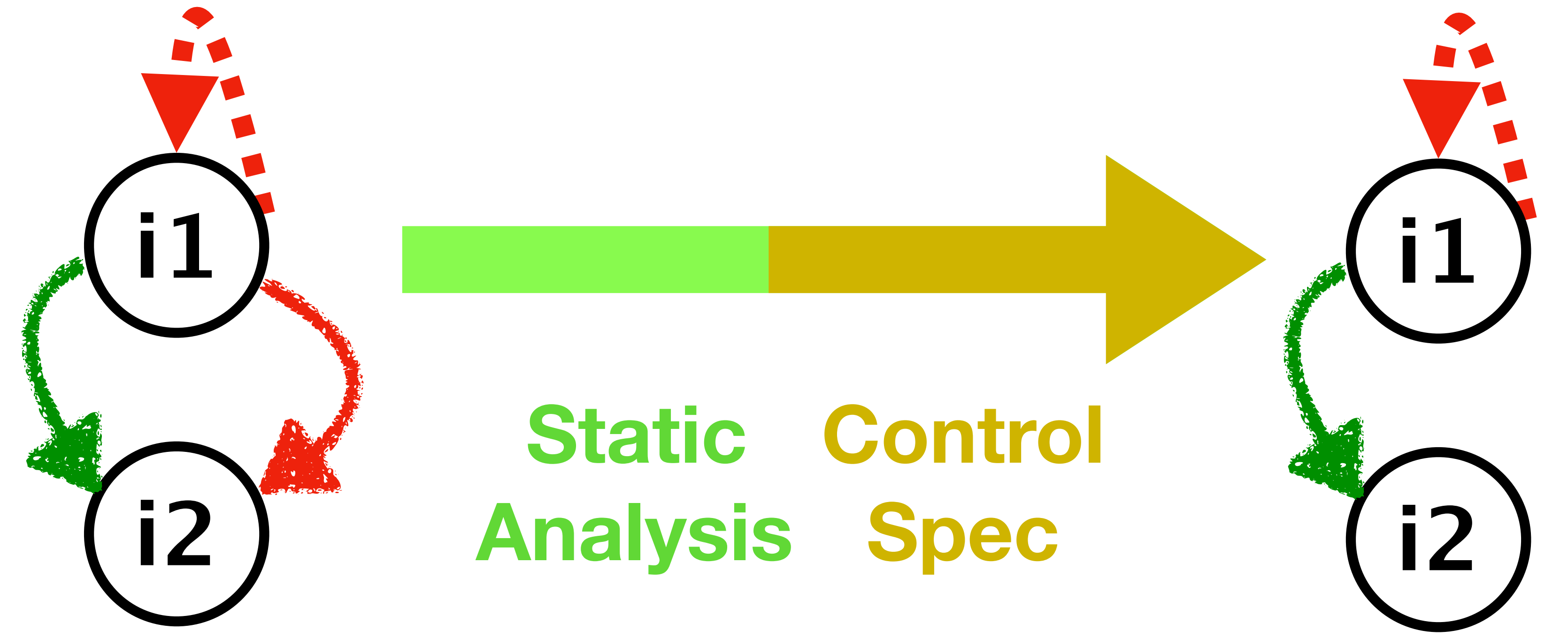
```
   for (i=0; i<N; ++i) {
     …
     if (observed_always_true)
i1:     *ptr = …
     …
i2:  … = … + *ptr
   }
```

Simplified example from
the dijkstra benchmark (MiBench)

- branch condition cannot be statically proven true
- **ptr** is not modified within the loop

Cross-iter RAW Dep
Cross-iter WAW Dep
Intra-iter RAW Dep

Program Dependence Graph (PDG)



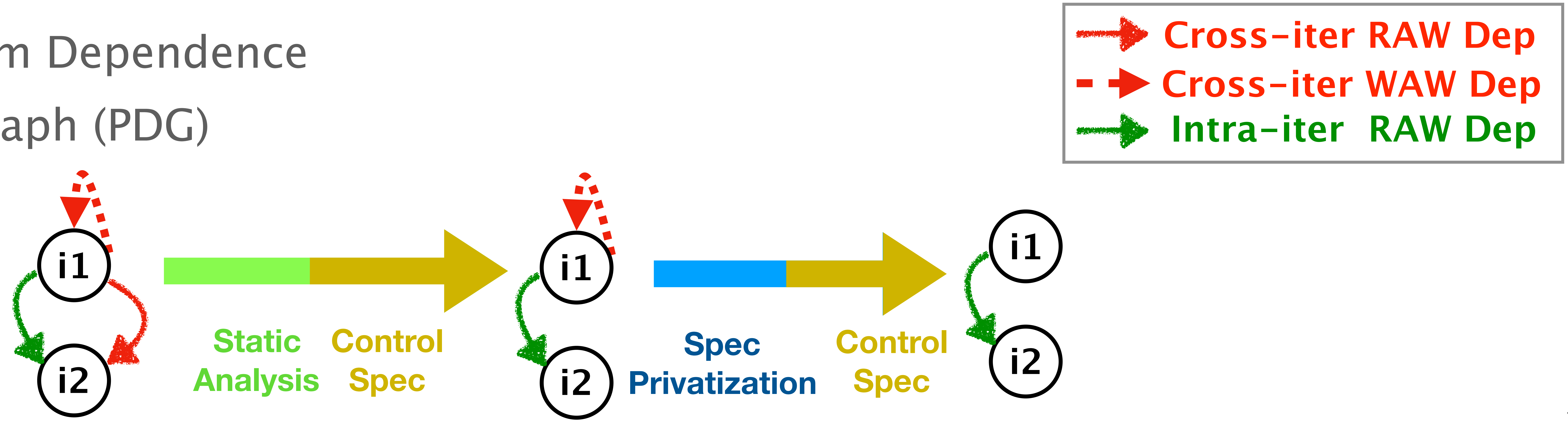
Fully leverage inexpensive speculative assertions to efficiently break dependences

```
   for (i=0; i<N; ++i) {
     …
     if (observed_always_true)
i1:     *ptr = …
     …
i2:  … = … + *ptr
   }
```

Simplified example from
the dijkstra benchmark (MiBench)

- branch condition cannot be statically proven true
- **ptr** is not modified within the loop

Program Dependence Graph (PDG)



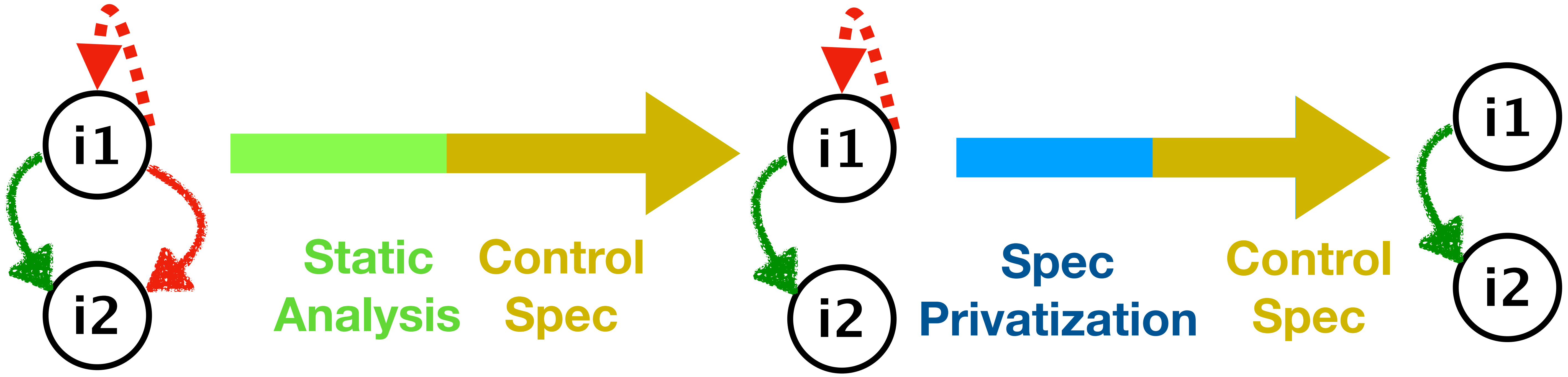
Fully leverage inexpensive speculative assertions to efficiently break dependences

```
   for (i=0; i<N; ++i) {
     …
     if (observed_always_true)
i1:     *ptr = …
     …
i2:  … = … + *ptr
   }
```

Simplified example from the dijkstra benchmark (MiBench)

- branch condition cannot be statically proven true
- **ptr** is not modified within the loop

Program Dependence Graph (PDG)



DOALL-able without monitoring

Inexpensive control speculation check instead of monitoring

```
   for (i=0; i<N; ++i) {
     …
     if (observed_always_true)
i1:    *ptr = …
     else
       misspec()
     …
i2:  … = … + *ptr
   }
```

Control
Spec Check

The *Perspective* Approach

The Perspective Approach

Design goals

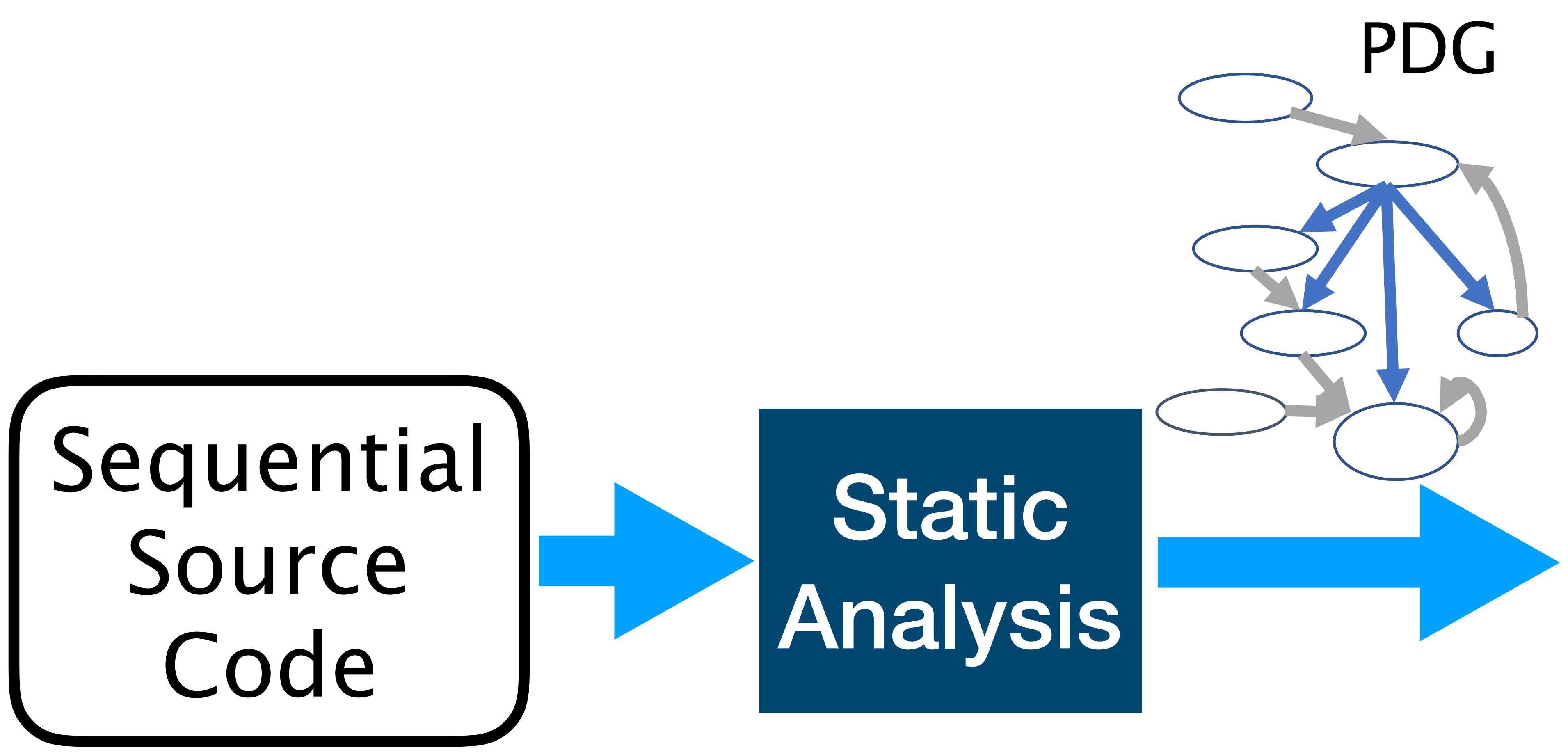
- Increase awareness
- Enable collaboration
- Avoid unnecessary transforms

The *Perspective* Approach

- ✦ Planning

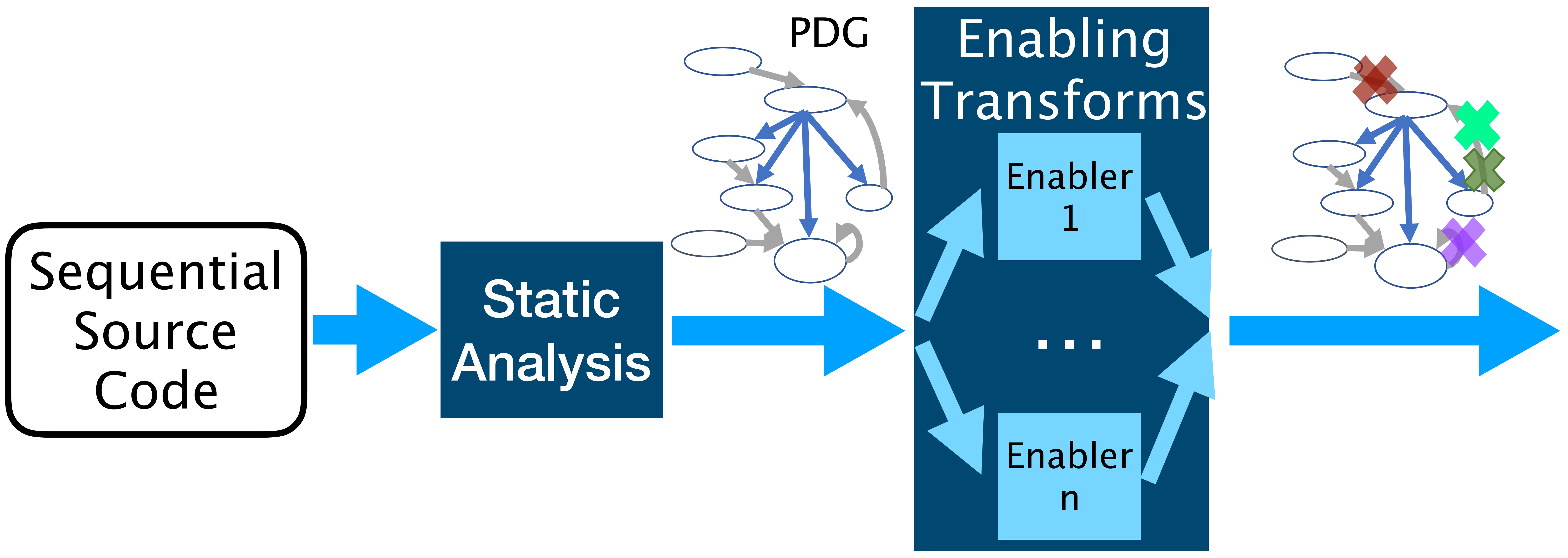
The Perspective Approach

✦ Planning



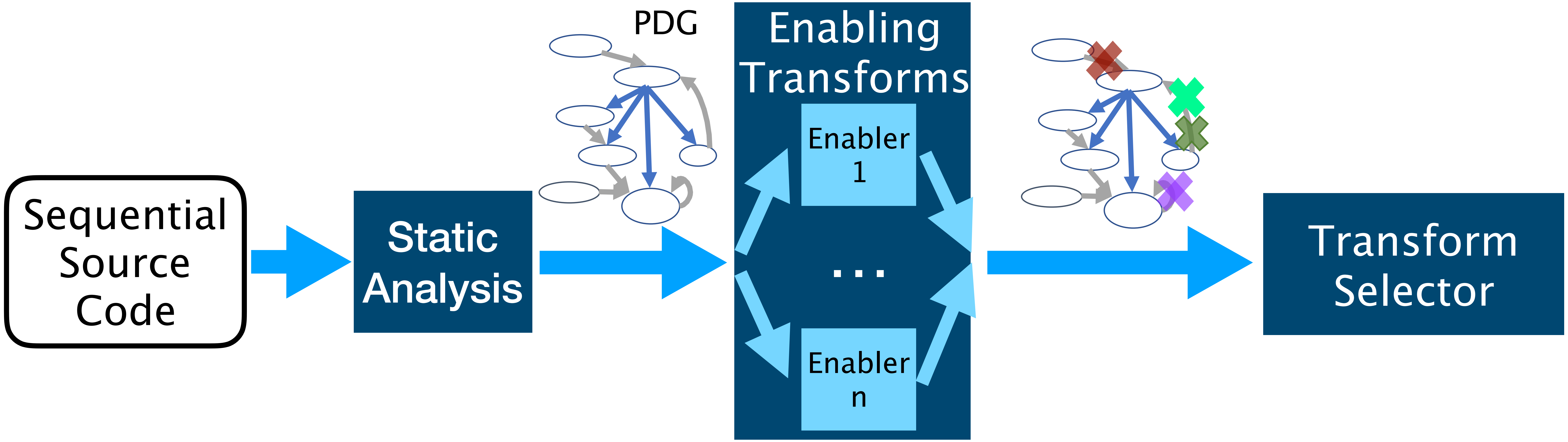
The Perspective Approach

✦ Planning



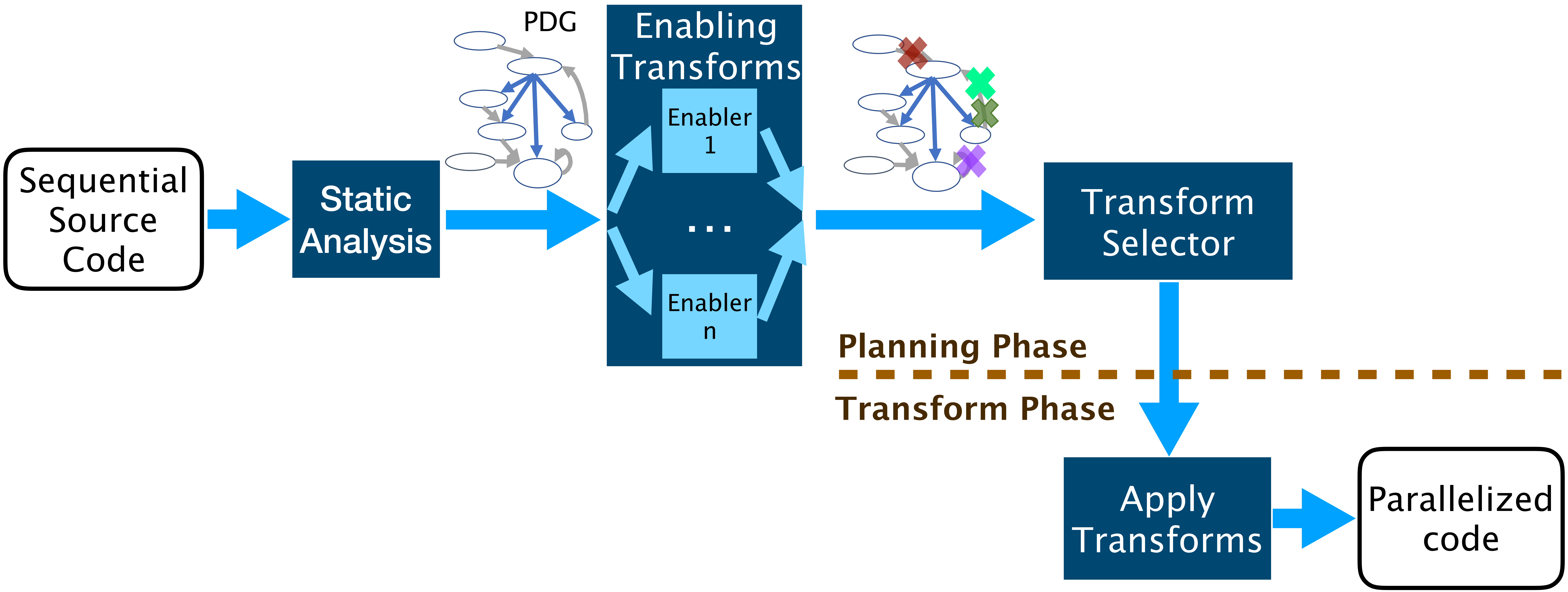
The Perspective Approach

✦ Planning



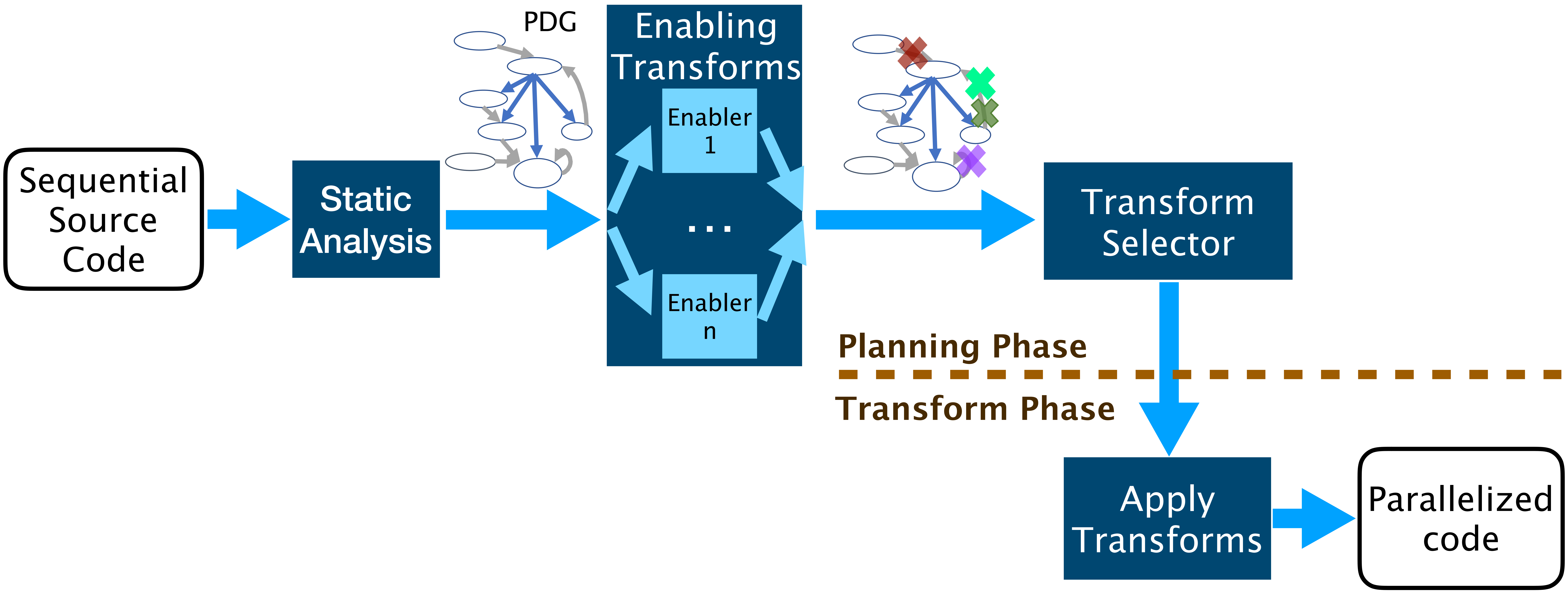
The Perspective Approach

- ✦ Planning



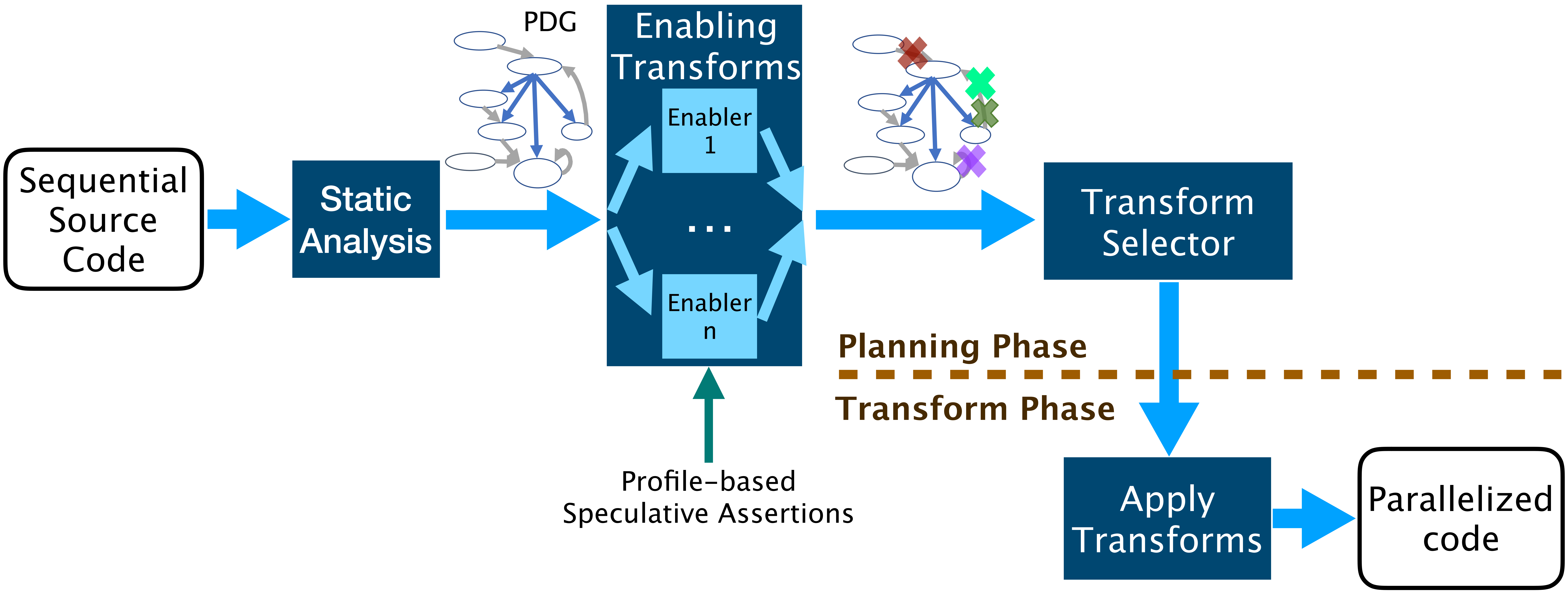
The Perspective Approach

- ✦ Planning
- ✦ Speculation-Aware Memory Analysis



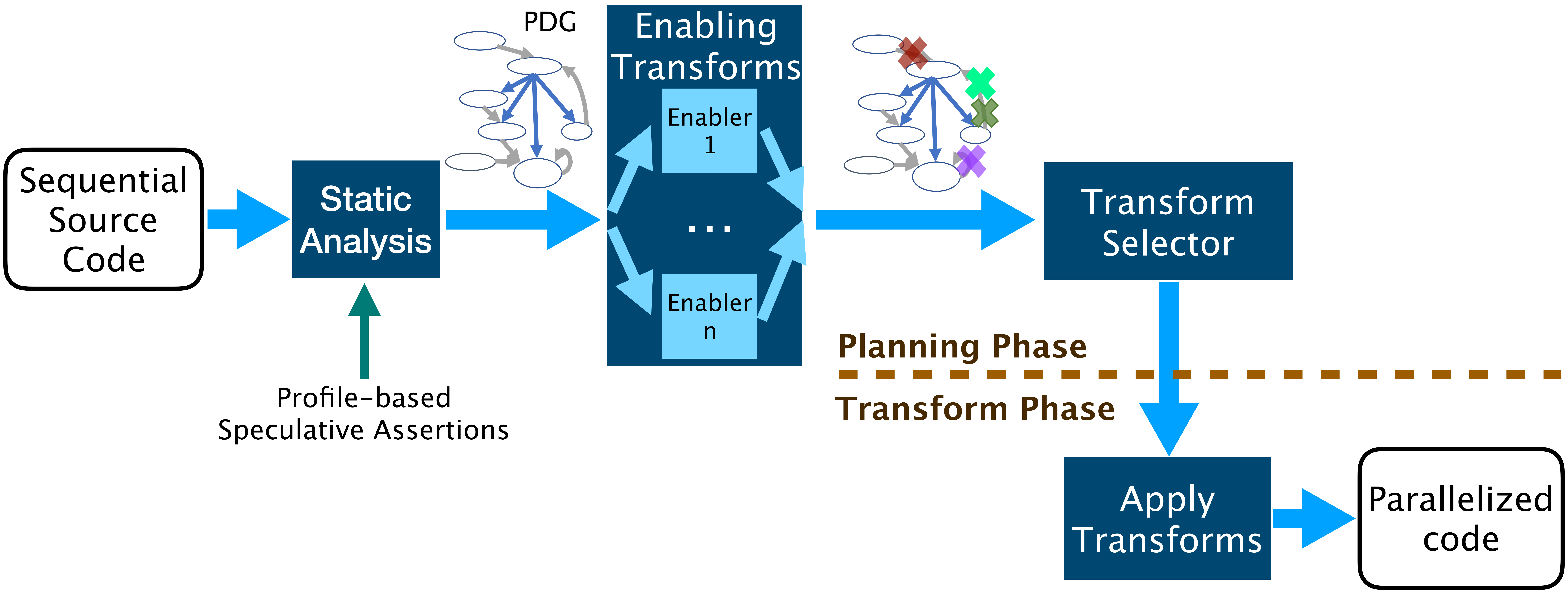
The *Perspective* Approach

- ✦ Planning
- ✦ Speculation-Aware Memory Analysis



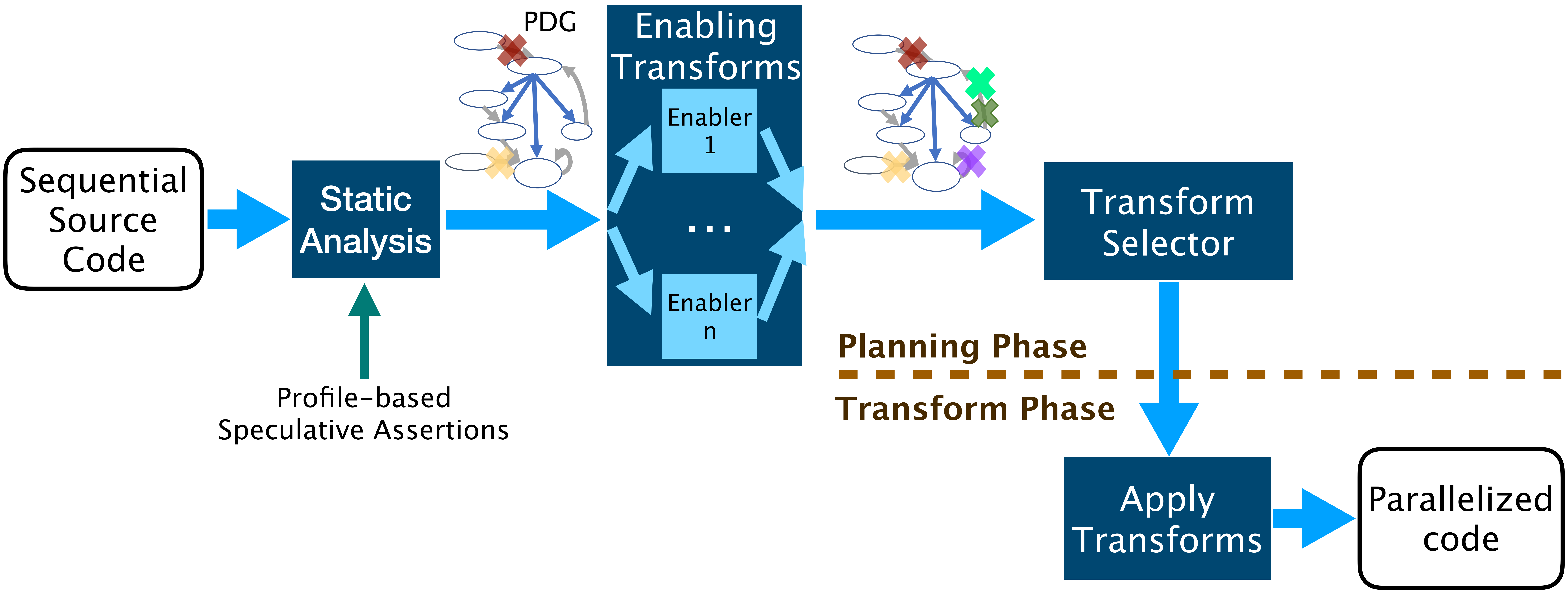
The Perspective Approach

- ✦ Planning
- ✦ Speculation-Aware Memory Analysis



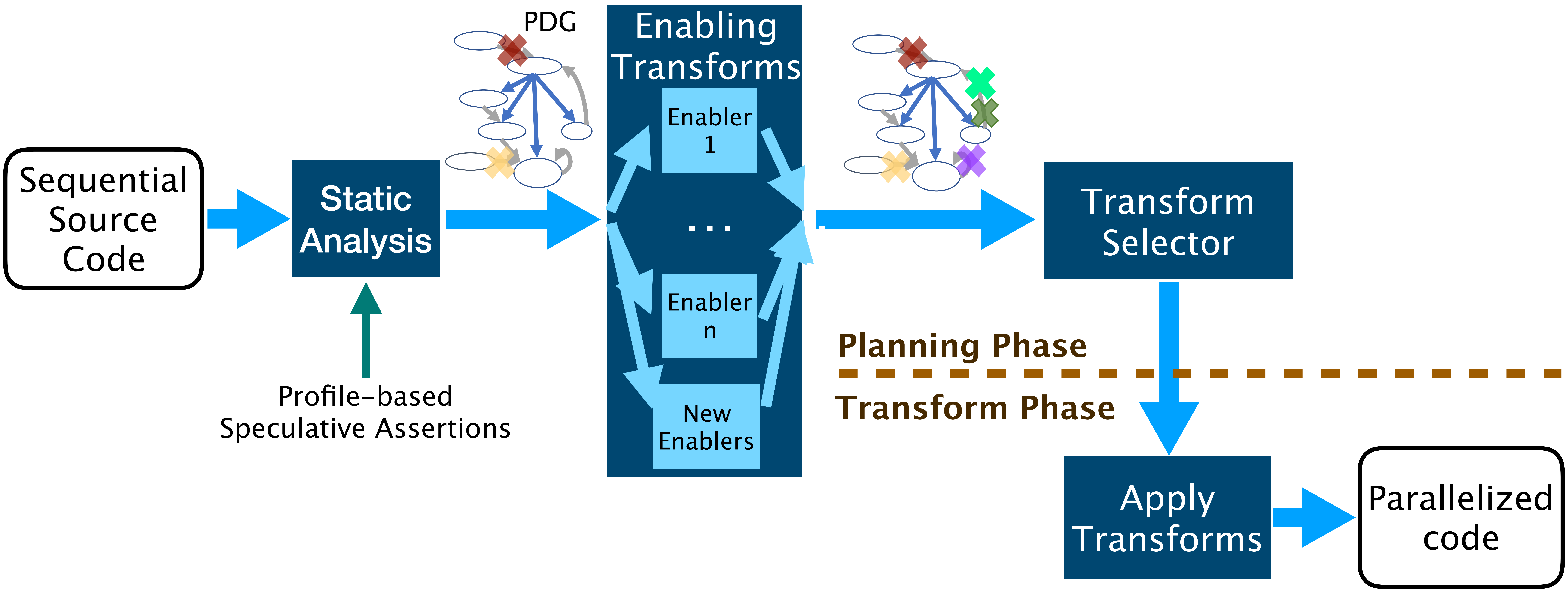
The Perspective Approach

- ✦ Planning
- ✦ Speculation-Aware Memory Analysis



The Perspective Approach

- ✦ Planning
- ✦ Speculation-Aware Memory Analysis
- ✦ New Efficient Enabling Transforms



Revisiting motivating example with Perspective

Parallelization of dijkstra benchmark (MiBench)

No ~~Excessive~~ use of memory speculation

Efficient ~~Expensive~~ privatization

Required monitoring of
~~973GB of reads & 649GB of writes~~!
0B of reads & 4KB of writes

4.8x speedup over Privateer*

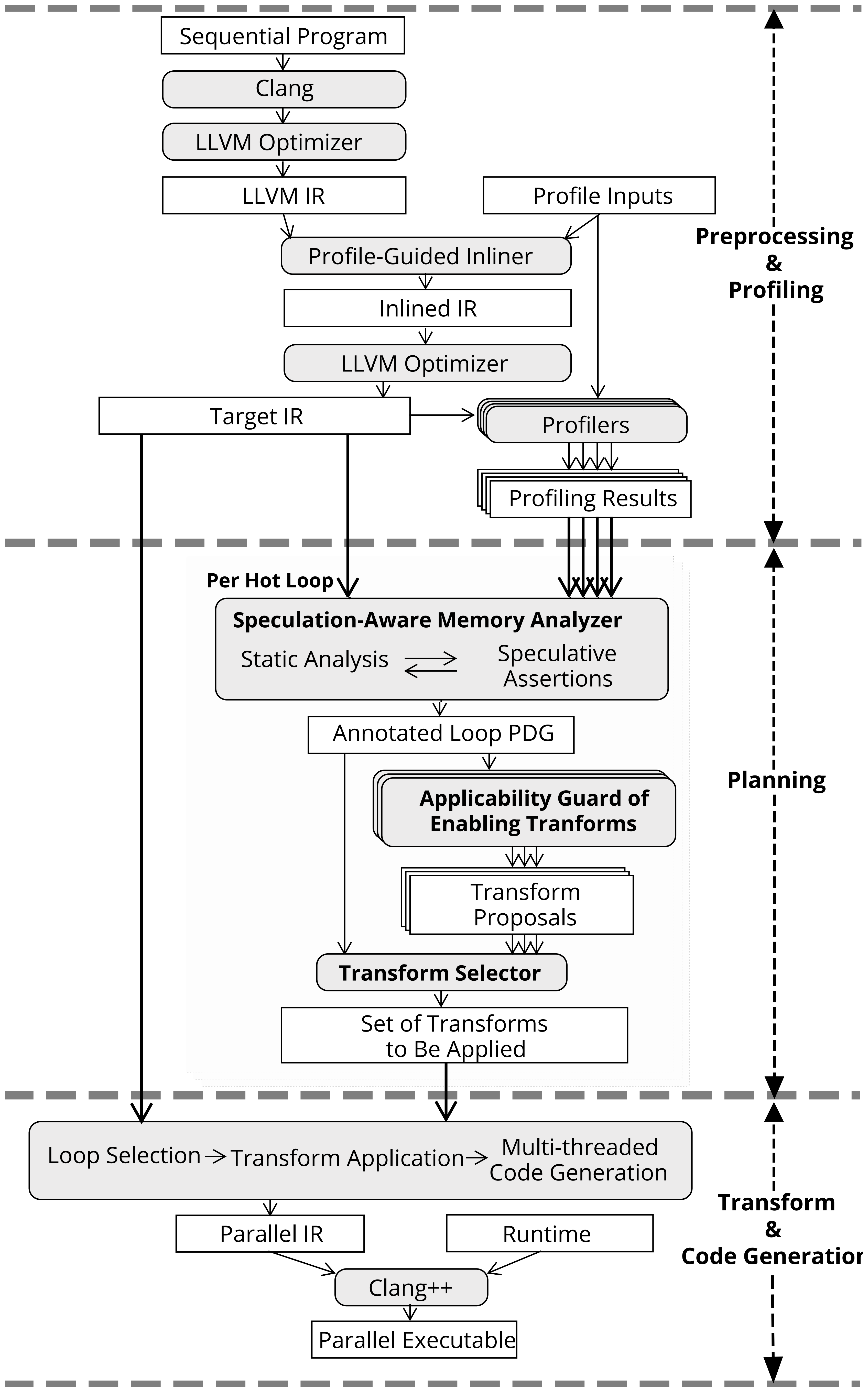
* Nick P. Johnson et al., PLDI '12

Perspective Framework

is implemented on the

LLVM Compiler Infrastructure

~80K loc in C/C++



Perspective's Evaluation Methodology

Perspective's Evaluation Methodology

Platform

Evaluated on a commodity shared-memory machine with 28 cores

Perspective's Evaluation Methodology

Platform

Evaluated on a commodity shared-memory machine with 28 cores

Empirically Evaluated Claim

Maintain the applicability of prior automatic-DOALL systems
while improving their efficiency

Perspective's Evaluation Methodology

Platform

Evaluated on a commodity shared-memory machine with 28 cores

Empirically Evaluated Claim

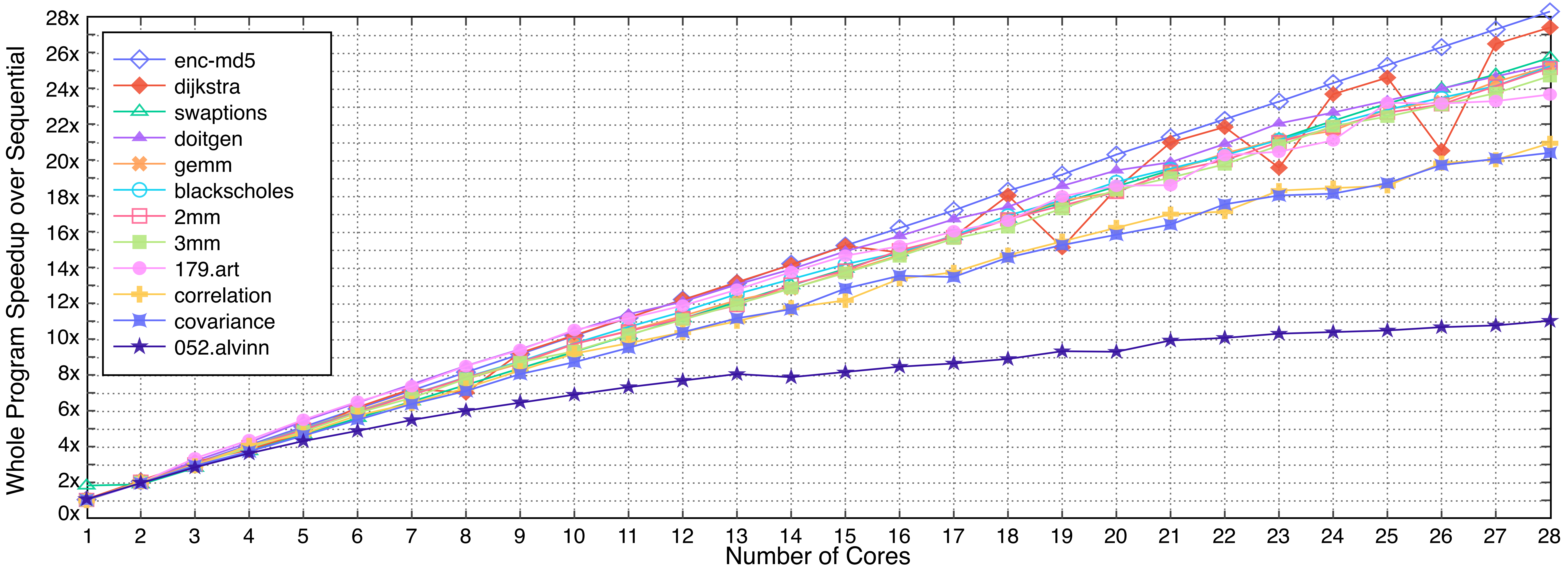
Maintain the applicability of prior automatic-DOALL systems while improving their efficiency

Benchmarks

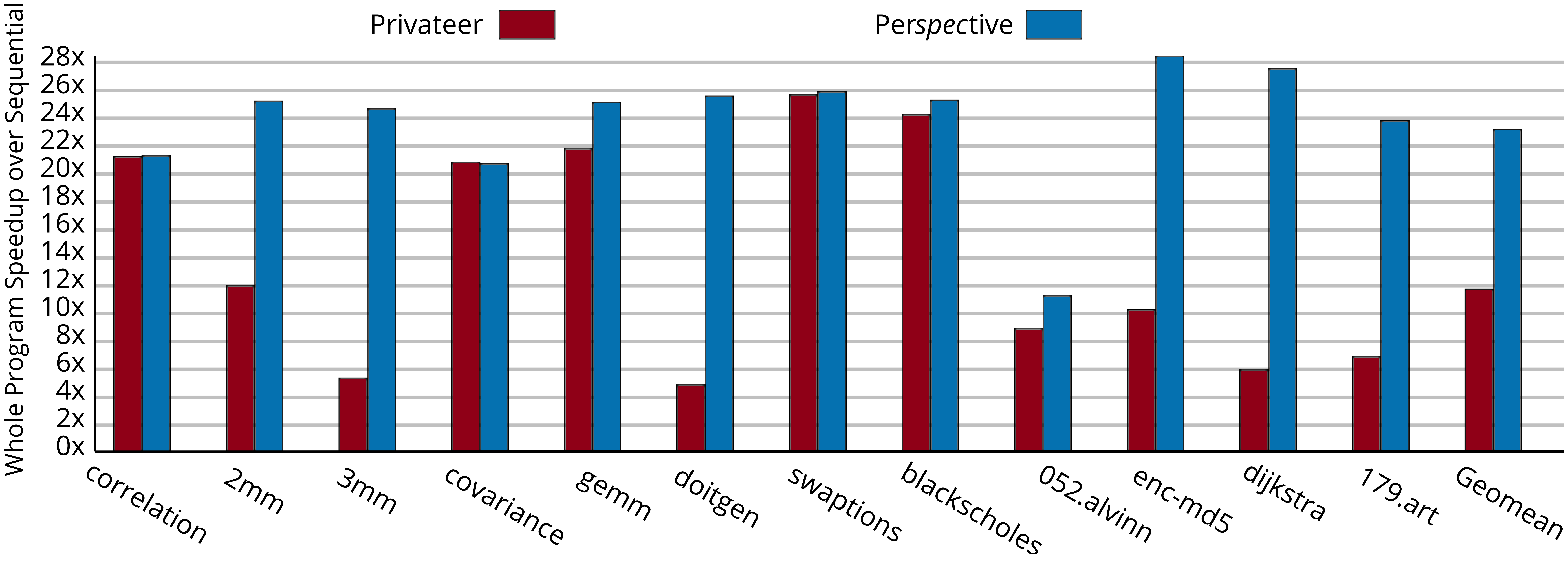
All parallelizable benchmarks from two state-of-the-art automatic DOALL-parallelization papers [1,2].
12 C/C++ benchmarks from SPEC CPU, PARSEC, PolyBench and MiBench.

[1] Johnson et al., PLDI '12 [2] Kim et al., CGO '12

Perspective yields scalable speedup



Perspective doubles performance of Privateer*



* Nick P. Johnson et al., Speculative Separation for Privatization and Reductions in PLDI ‘12

Perspective doubles performance of Privateer thanks to dramatic reduction of monitored reads/writes

Benchmark	Monitored Read Set Size		Monitored Write Set Size	
	Privateer	Perspective	Privateer	Perspective
enc-md5	1.87TB	39.1KB	581GB	43.2KB
052.alvinn	153GB	0B	107GB	10.2MB
179.art	1.6TB	0B	958GB	1.68GB
2mm	1TB	0B	1TB	0B
3mm	3TB	0B	1.5TB	0B
correlation	0B	0B	192MB	192MB
covariance	0B	0B	192GB	192MB
doitgen	2.53TB	0B	2.54TB	0B
gemm	128MB	0B	256MB	0B
blackscholes	0B	0B	37.3GB	336B
swaptions	703KB	0B	165KB	165KB
dijkstra	973GB	0B	649GB	3.61KB

Conclusion

- Per*spec*tive **advances** state-of-the-art by identifying and mitigating **core inefficiencies** of prior speculative automatic parallelization systems.

- Per*spec*tive generates **minimal-cost DOALL-parallelization plans** by combining a planning phase, speculation-aware memory analysis, and efficient speculative privatization.

- Per*spec*tive fully-**automatically** yields scalable speedup (**23.0**× on 28 cores), **double** the performance of state-of-the-art.

Artifact available at: https://doi.org/10.5281/zenodo.3606885