Extracting Clean Performance Models from Tainted Programs

Marcin Copik, Alexandru Calotoiu, Tobias Grosser, Nicolas Wicki, Felix Wolf, Torsten Hoefler

SIAM PP 2022
25th February 2021

SPCL
THE UNIVERSITY OF EDINBURGH
TECHNISCHE UNIVERSITÄT DARMSTADT

Performance Modeling: state of the art

C, C++, Fortran

MPI, OpenMP

Performance Modeling: state of the art

C, C++, Fortran

MPI, OpenMP

main: $2.3s^3 + 1.7\log_2 p - 0.13$

foo: $1.5s^2 + 1.2$

bar: $1.3\log_2 p$

Performance Modeling: state of the art

C, C++, Fortran

MPI, OpenMP

main: $2.3s^3 + 1.7 \log_2 p - 0.13$

foo: $1.5s^2 + 1.2$

bar: $1.3 \log_2 p$

Scalability bugs [1]

[1] A. Calotoiu, *"Using Automated Performance Modeling to Find Scalability Bugs in Complex Codes"*, **SC '13**

Performance Modeling: state of the art

C, C++, Fortran

MPI, OpenMP

main: $2.3s^3 + 1.7\log_2 p - 0.13$

foo: $1.5s^2 + 1.2$

bar: $1.3\log_2 p$

Scalability bugs [1]

Performance validation [2]

[2] S. Shudler, "*Exascaling Your Library: Will Your Implementation Meet Your Expectations?*", **ICS '15**

Performance Modeling: state of the art

C, C++, Fortran

MPI, OpenMP

main: $2.3s^3 + 1.7 \log_2 p - 0.13$

foo: $1.5s^2 + 1.2$

bar: $1.3 \log_2 p$

Scalability bugs [1]

Performance validation [2]

Exascale system design [3]

[3] A. Calotoiu, *"Lightweight Requirements Engineering for Exascale Co-design"*, **CLUSTER '18**

Challenges in Automatic Performance Modeling

Parameters Identification

Select problem size **s** and ranks **p** as model parameters.

Challenges in Automatic Performance Modeling

Parameters Identification → **Experiment Design**

Select problem size **s** and ranks **p** as model parameters.

Decide to use
- **5** values per parameter
- **5** samples per experiment

25 combinations of **p** and **s**

Challenges in Automatic Performance Modeling

Parameters Identification → Experiment Design → Experiment Execution

Parameters Identification

Select problem size **s** and ranks **p** as model parameters.

Experiment Design

Decide to use
- **5** values per parameter
- **5** samples per experiment

25 combinations of **p** and **s**

Experiment Execution

Score-P
Scalable performance measurement infrastructure for parallel codes

25 parameter values
5 repetitions per sample
125 instrumented executions

Challenges in Automatic Performance Modeling

Parameters Identification → **Experiment Design** → **Experiment Execution** → **Extra-P Modeler**

Select problem size **s** and ranks **p** as model parameters.

Decide to use
- **5** values per parameter
- **5** samples per experiment

25 combinations of **p** and **s**

Score-P

25 parameter values
5 repetitions per sample
125 instrumented executions

Create performance model for **each function**.
Example result:

$$2.3\, s^3 + 1.71 \log_2 p - 0.1329$$

Challenges in Automatic Performance Modeling

Parameters Identification → **Experiment Design** → **Experiment Execution** → **Extra-P Modeler**

Select problem size **s** and ranks **p** as model parameters.

Decide to use
- **5** values per parameter
- **5** samples per experiment

25 combinations of **p** and **s**

25 parameter values
5 repetitions per sample
125 instrumented executions

Create performance model for **each function**.
Example result:
$2.3\, s^3 + 1.71 \log_2 p - 0.1329$

#1: Which parameters are relevant? [1]

```
int nx, ny, nz, nt;
int node_geometry[4];
int nflavors, propinterval;
int warms, trajecs, steps;
int niter, nrestart, prec_pbp;
```

A **subset** of all *su3_rmd* parameters.

[1] T. Hoefler *"Performance Modeling for Systematic Performance Tuning"*, **SC '11**

Challenges in Automatic Performance Modeling

Parameters Identification → **Experiment Design** → **Experiment Execution** → **Extra-P Modeler**

Select problem size **s** and ranks **p** as model parameters.

Decide to use
- **5** values per parameter
- **5** samples per experiment

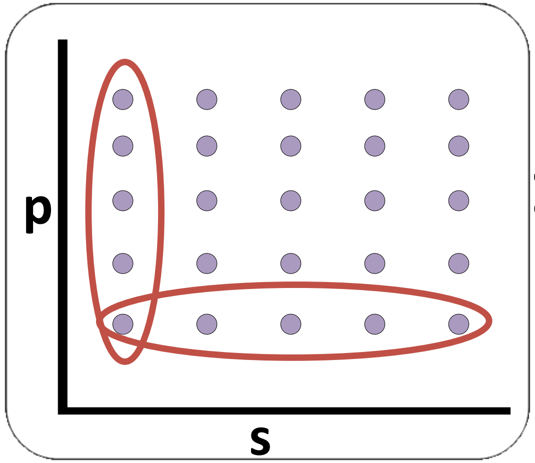
25 combinations of **p** and **s**

Score-P
Scalable performance measurement infrastructure for parallel codes

25 parameter values
5 repetitions per sample
125 instrumented executions

Create performance model for **each function**.
Example result:
$2.3\ s^3 + 1.71 \log_2 p - 0.1329$

#2: How do parameters interact with each other? [2]



$p \times s$
10+ experiments
$p + s$
9 experiments

[2] M. Ritter, *"Performance Modeling at a Discount"*, **IPDPS '20**

Challenges in Automatic Performance Modeling

Parameters Identification → **Experiment Design** → **Experiment Execution** → **Extra-P Modeler**

Select problem size **s** and ranks **p** as model parameters.

Decide to use
- **5** values per parameter
- **5** samples per experiment

25 combinations of **p** and **s**

Score-P

25 parameter values
5 repetitions per sample
125 instrumented executions

Create performance model for **each function**.
Example result:
$2.3\,s^3 + 1.71\log_2 p - 0.1329$

#3: Which functions are worth being instrumented?

Challenges in Automatic Performance Modeling

Parameters Identification → **Experiment Design** → **Experiment Execution** → **Extra-P Modeler**

Select problem size **s** and ranks **p** as model parameters.

Decide to use
- **5** values per parameter
- **5** samples per experiment

25 combinations of **p** and **s**

Score-P

25 parameter values
5 repetitions per sample
125 instrumented executions

Create performance model for **each function**.
Example result:

$$2.3\, s^3 + 1.71 \log_2 p - 0.1329$$

```
int p = MPI_ranks();
for(int i = 0; i < p - 1; ++i)
  MPI_Send(…);
```

$$-10^{-5} s^2 + 1.3\, p + 0.7$$

#4: Which functions and parameters affect performance?

Challenges in Automatic Performance Modeling

Parameters Identification → **Experiment Design** → **Experiment Execution** → **Extra-P Modeler**

Select problem size **s** and ranks **p** as model parameters.

Decide to use
- **5** values per parameter
- **5** samples per experiment

25 combinations of **p** and **s**

Score-P

25 parameter values
5 repetitions per sample
125 instrumented executions

Create performance model for **each function**.
Example result:

$$2.3\, s^3 + 1.71 \log_2 p - 0.1329$$

```
int p = MPI_ranks();
for(int i = 0; i < p - 1; ++i)
  MPI_Send(…);
```

$$-10^{-5}s^2 + 1.3\, p + 0.7$$

#5: Does the model represent application behavior or hardware effects?

Challenges in Automatic Performance Modeling

Parameters Identification → **Experiment Design** → **Experiment Execution** → **Extra-P Modeler**

Select problem size **s** and ranks **p** as model parameters.

Decide to use
- **5** values per parameter
- **5** samples per experiment

25 combinations of **p** and **s**

Score-P

25 parameter values
5 repetitions per sample
125 instrumented executions

Create performance model for **each function**.
Example result:

$$2.3\ s^3 + 1.71 \log_2 p - 0.1329$$

#1: Which parameters are relevant? [1]

#2: How do parameters interact with each other? [2]

#3: Which functions are worth being instrumented?

#4: Which functions and parameters affect performance?

#5: Does the model represent application behavior or hardware effects?

Challenges in Automatic Performance Modeling

Parameters Identification → **Experiment Design** → **Experiment Execution** → **Extra-P Modeler**

Select problem size **s** and ranks **p** as model parameters.

Decide to use
- **5** values per parameter
- **5** samples per experiment

25 combinations of **p** and **s**

Score-P

25 parameter values
5 repetitions per sample
125 instrumented executions

Create performance model for **each function**.
Example result:

$$2.3\ s^3 + 1.71 \log_2 p - 0.1329$$

#1: Which parameters are relevant? [1]

other? [2]

We need a **white-box approach.**

#4: Which functions and parameters affect performance?

#5: Does the model represent application behavior or hardware effects?

What is important in our program?

```c
void main(int s, int p) {
  g(s, p); h(s, p); i(s, p);
}
```

```c
void g(int s) {
  MPI_Send(&s, 1, MPI_INT);
}
```

```c
void h(int s, int p) {
  j(s);
}
```

```c
void i(int s, int p) {
  printf("%d %d\n", s, p);
}
```

```c
void j(int x) {
  for(int j = 0; j < x; ++j)
    // compute
}
```

What is important in our program?

```
void main(int s, int p) {
  g(s, p); h(s, p); i(s, p);
}
```

Which functions are performance-critical?

```
void g(int s) {
  MPI_Send(&s, 1, MPI_INT);
}
```

```
void h(int s, int p) {
  j(s);
}
```

```
void i(      p) {
  printf(      \n", s, p);
}
```

```
void j(int x) {
  for(int j = 0; j < x; ++j)
    // compute
}
```

What is important in our program?

```
void main(int s, int p) {
  g(s, p); h(s, p); i(s, p);
}
```

Which functions are performance-critical?

```
void g(int s) {
  MPI_Send(&s, 1, MPI_INT);
}
```

```
void h(int s, int p) {
  j(s);
}
```

```
void i(     p) {
  printf(     \n", s, p);
}
```

```
void j(int x) {
  for(int j = 0; j < x; ++j)
    // compute
}
```

What is important in our program?

```
void main(int s, int p) {
  g(s, p); h(s, p); i(s, p);
}
```

Which functions are performance-critical?

```
void g(int s) {
  MPI_Send(&s, 1, MPI_INT);
}
```

```
void h(int s, int p) {
  j(s);
}
```

```
void i(              p) {
  printf(          \n", s, p);
}
```

```
void j(int x) {
  for(int j = 0; j < x; ++j)
    // compute
}
```

What is important in our program?

```
void main(int s, int p) {
  g(s, p); h(s, p); i(s, p);
}
```

Which functions are performance-critical?

```
void g(int s) {
  MPI_Send(&s, 1, MPI_INT);
}
```

```
void h(int s, int p) {
  j(s);
}
```

```
void i(      p) {
  printf(      \n", s, p);
}
```

```
void j(int x) {
  for(int j = 0; j < x; ++j)
    // compute
}
```

What is important in our program?

```
void main(int s, int p) {
  g(s, p); h(s, p); i(s, p);
}
```

Which functions are performance-critical?

```
void g(int s) {
  MPI_Send(&s, 1, MPI_INT);
}
```

```
void h(int s, int p) {
  j(s);
}
```

```
void i( ... p) {
  printf( ... \n", s, p);
}
```

```
void j(int x) {
  for(int j = 0; j < x; ++j)
    // compute
}
```

What is important in our program?

```
void main(int s, int p) {
  g(s, p); h(s, p); i(s, p);
}
```

Which functions are performance-critical?

```
void g(int s) {
  MPI_Send(&s, 1, MPI_INT);
}
```

```
void h(int s, int p) {
  j(s);
}
```

```
void i(    p) {
  printf(    \n", s, p);
}
```

```
void j(int x) {
  for(int j = 0; j < x; ++j)
    // compute
}
```

What is important in our program?

```
void main(int s, int p) {
  g(s, p); h(s, p); i(s, p);
}
```

Which functions are performance-critical?

```
void g(int s) {
  MPI_Send(&s, 1, MPI_INT);
}
```

```
void h(int s, int p) {
  j(s);
}
```

```
void i(      p) {
  printf(     \n", s, p);
}
```

```
void j(int x) {
  for(int j = 0; j < x; ++j)
    // compute
}
```

Which parameters affect performance?

Analysis Requirements

The analysis must…

Analysis Requirements

The analysis must…

- …detect functions which performance does not change.

Analysis Requirements

The analysis must…

- …detect functions which performance does not change.
- …detect which parameters affected non-constant loops.

Analysis Requirements

The analysis must…

- …detect functions which performance does not change.
- …detect which parameters affected non-constant loops.
- …support inter-procedural parameter dependencies.

Analysis Requirements

The analysis must…

- …detect functions which performance does not change.
- …detect which parameters affected non-constant loops.
- …support inter-procedural parameter dependencies.
- …be memory agnostic.

Analysis Requirements

The analysis must…

- …detect functions which performance does not change.
- …detect which parameters affected non-constant loops.
- …support inter-procedural parameter dependencies.
- …be memory agnostic.
- …support parallelism patterns introduced by MPI.

Analysis Requirements

The analysis must…

- …detect functions which performance does not change.
- …detect which parameters affected non-constant loops.
- …support inter-procedural parameter dependencies.
- …be memory agnostic.
- …support parallelism patterns introduced by MPI.
- …require minor code changes.

Analysis Requirements

The analysis must…

- …detect functions which performance does not change.
- …detect which parameters affected non-constant loops.
- …support inter-procedural parameter dependencies.
- …be memory agnostic.
- …support parallelism patterns introduced by MPI.
- …require minor code changes.

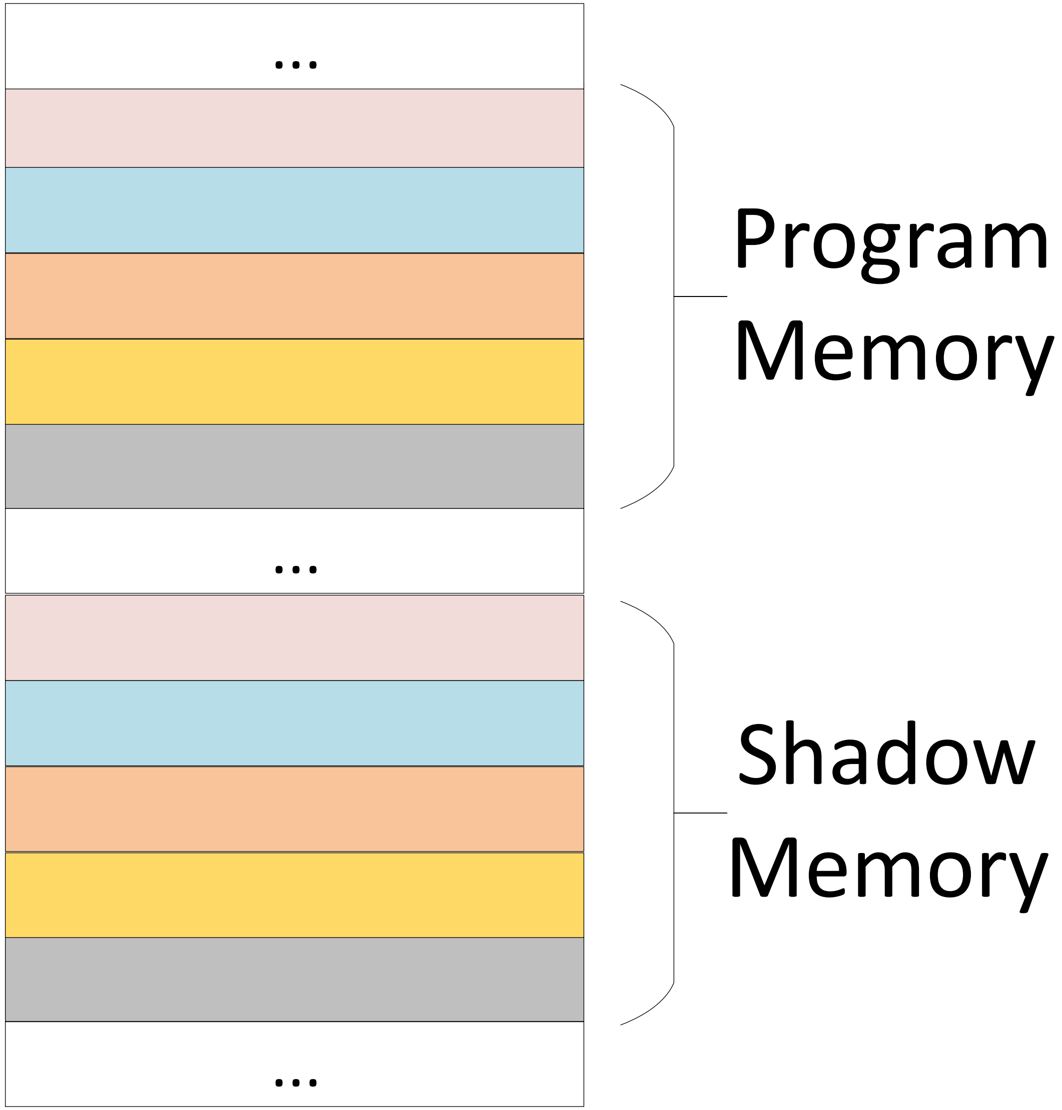
Answer? Taint Analysis.

Taint Analysis: track parameters propagation

```
int a = 42;
int b; MPI_Comm_size(&b, comm);
taint_variable(a);

// Data-flow propagation
int x = 2 * a;
int y = modulo(a, b);

// Control-flow propagation
int z = 10;
if(a != 43)
  z = 6;
```

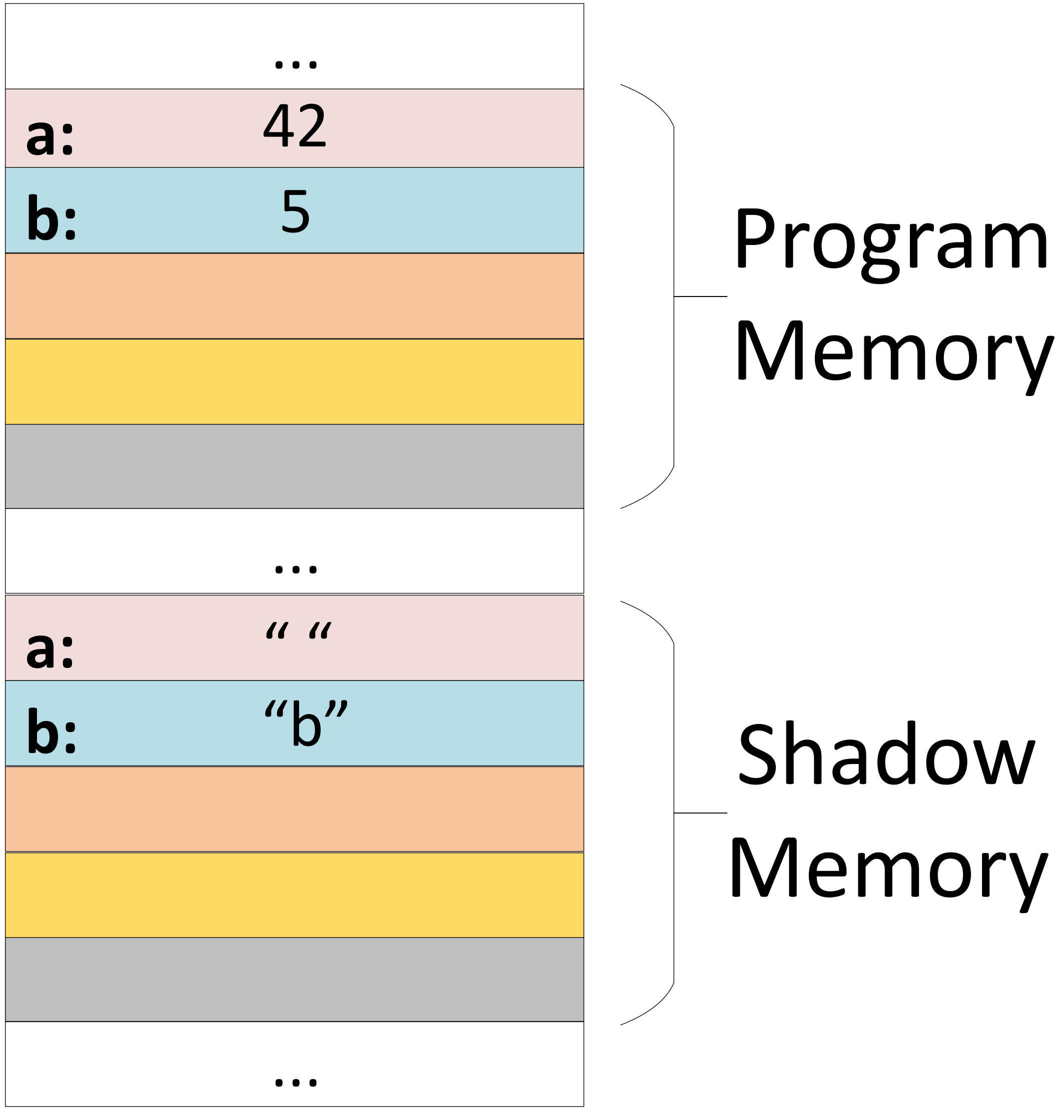


Taint Analysis: track parameters propagation

```
int a = 42;
int b; MPI_Comm_size(&b, comm);
taint_variable(a);

// Data-flow propagation
int x = 2 * a;
int y = modulo(a, b);

// Control-flow propagation
int z = 10;
if(a != 43)
  z = 6;
```

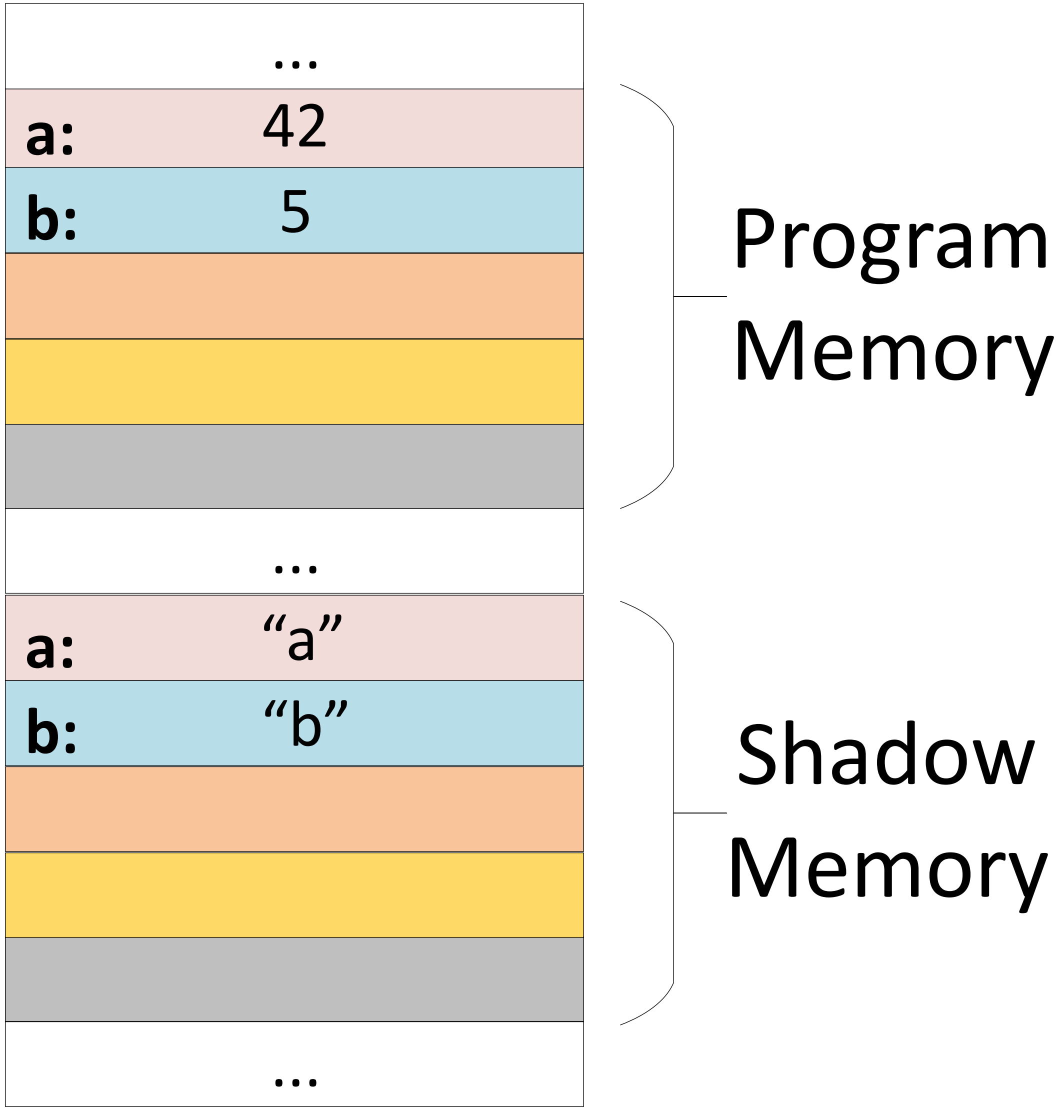


Taint Analysis: track parameters propagation

```
int a = 42;
int b; MPI_Comm_size(&b, comm);
taint_variable(a);

// Data-flow propagation
int x = 2 * a;
int y = modulo(a, b);

// Control-flow propagation
int z = 10;
if(a != 43)
  z = 6;
```

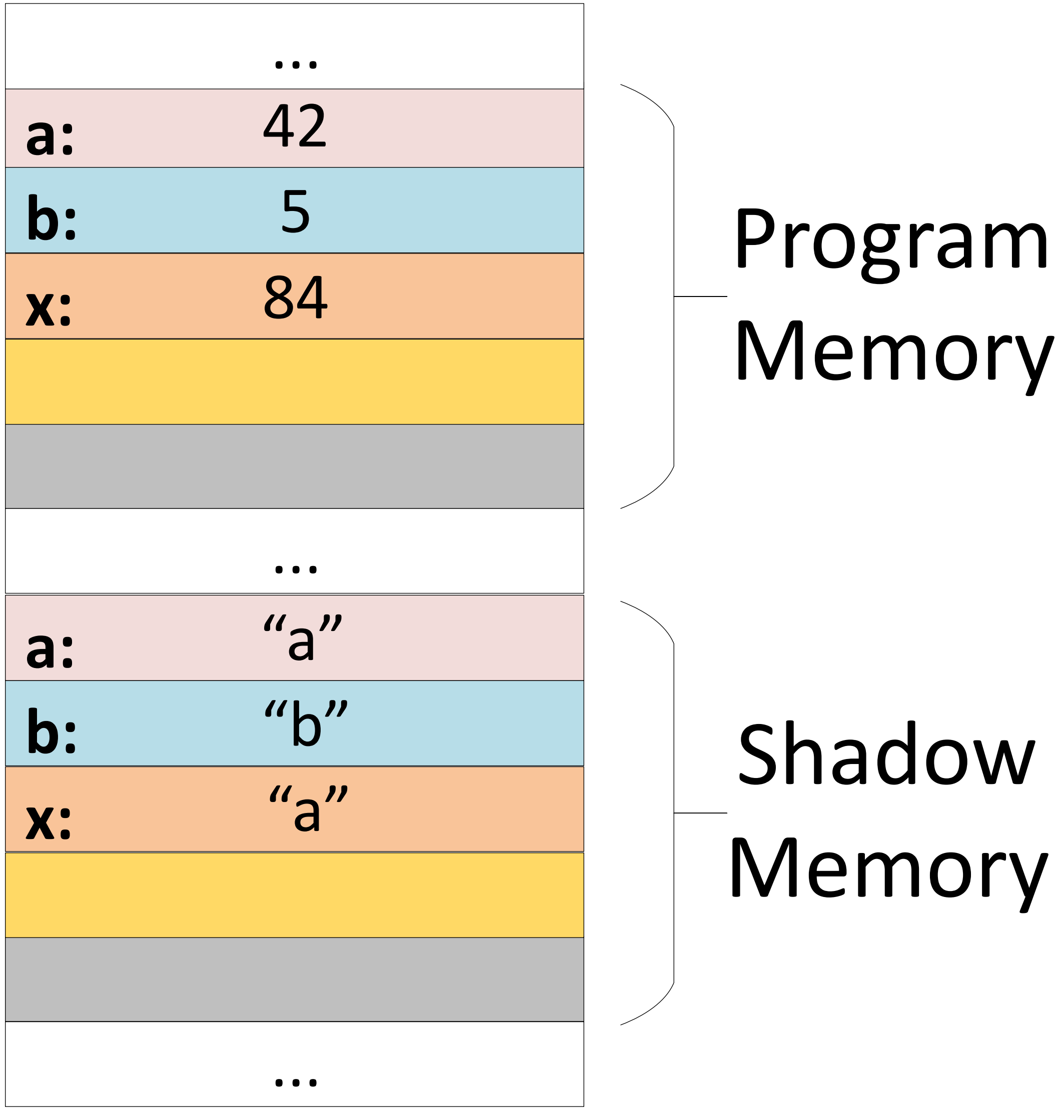


Taint Analysis: track parameters propagation

```
int a = 42;
int b; MPI_Comm_size(&b, comm);
taint_variable(a);

// Data-flow propagation
int x = 2 * a;
int y = modulo(a, b);

// Control-flow propagation
int z = 10;
if(a != 43)
  z = 6;
```

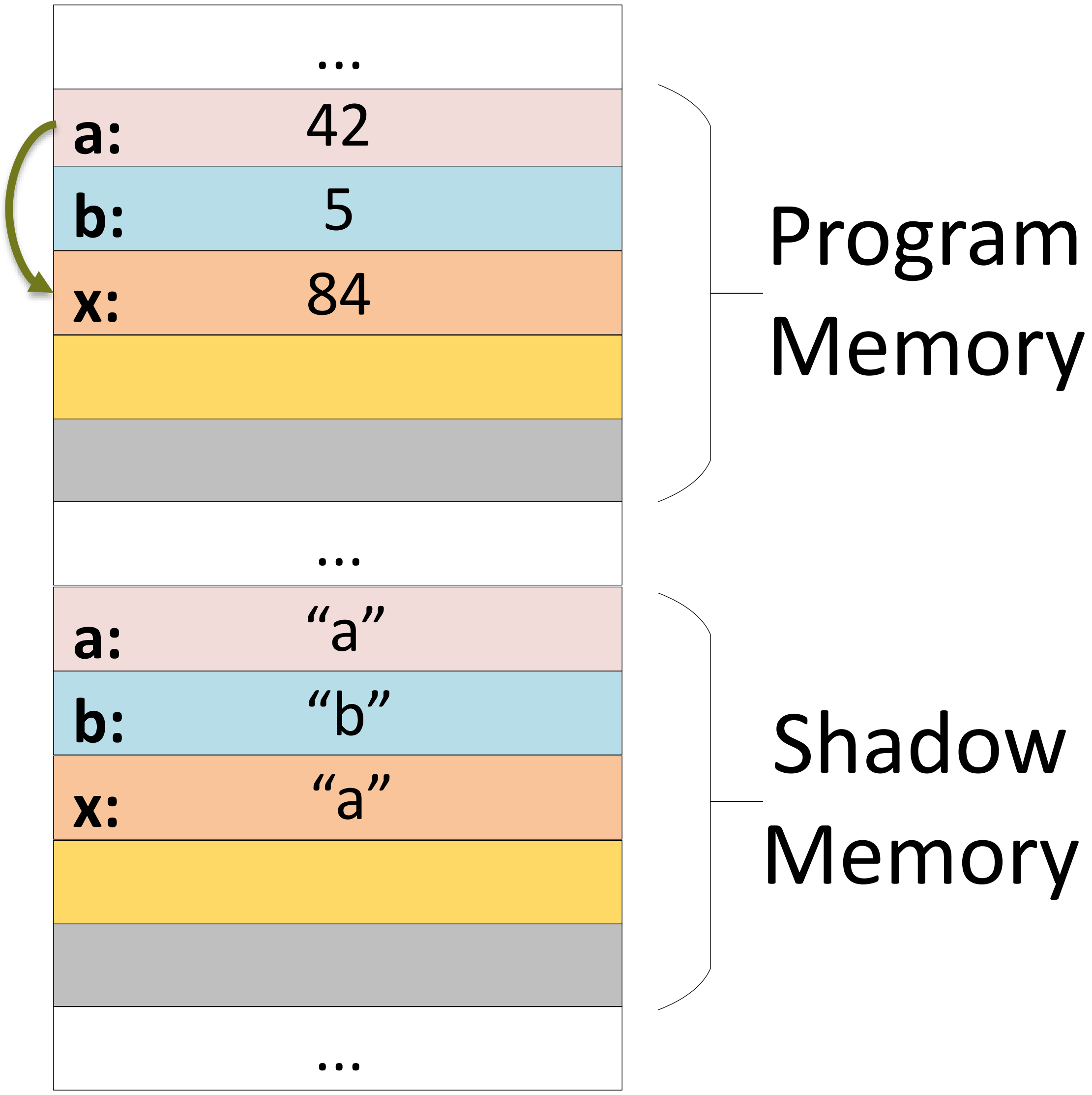


Taint Analysis: track parameters propagation

```
int a = 42;
int b; MPI_Comm_size(&b, comm);
taint_variable(a);

// Data-flow propagation
int x = 2 * a;
int y = modulo(a, b);

// Control-flow propagation
int z = 10;
if(a != 43)
  z = 6;
```

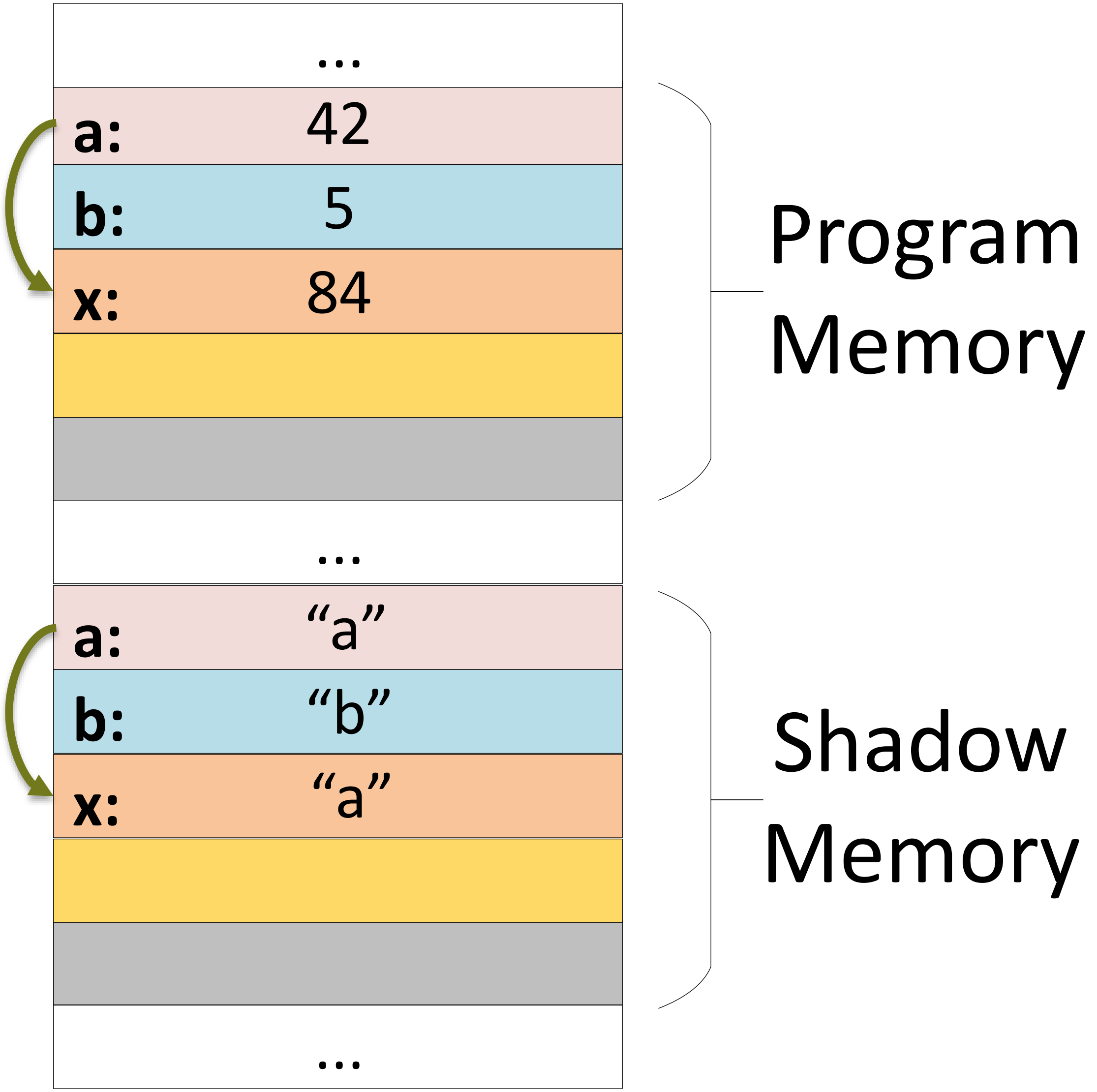


Taint Analysis: track parameters propagation

```
int a = 42;
int b; MPI_Comm_size(&b, comm);
taint_variable(a);

// Data-flow propagation
int x = 2 * a;
int y = modulo(a, b);

// Control-flow propagation
int z = 10;
if(a != 43)
   z = 6;
```

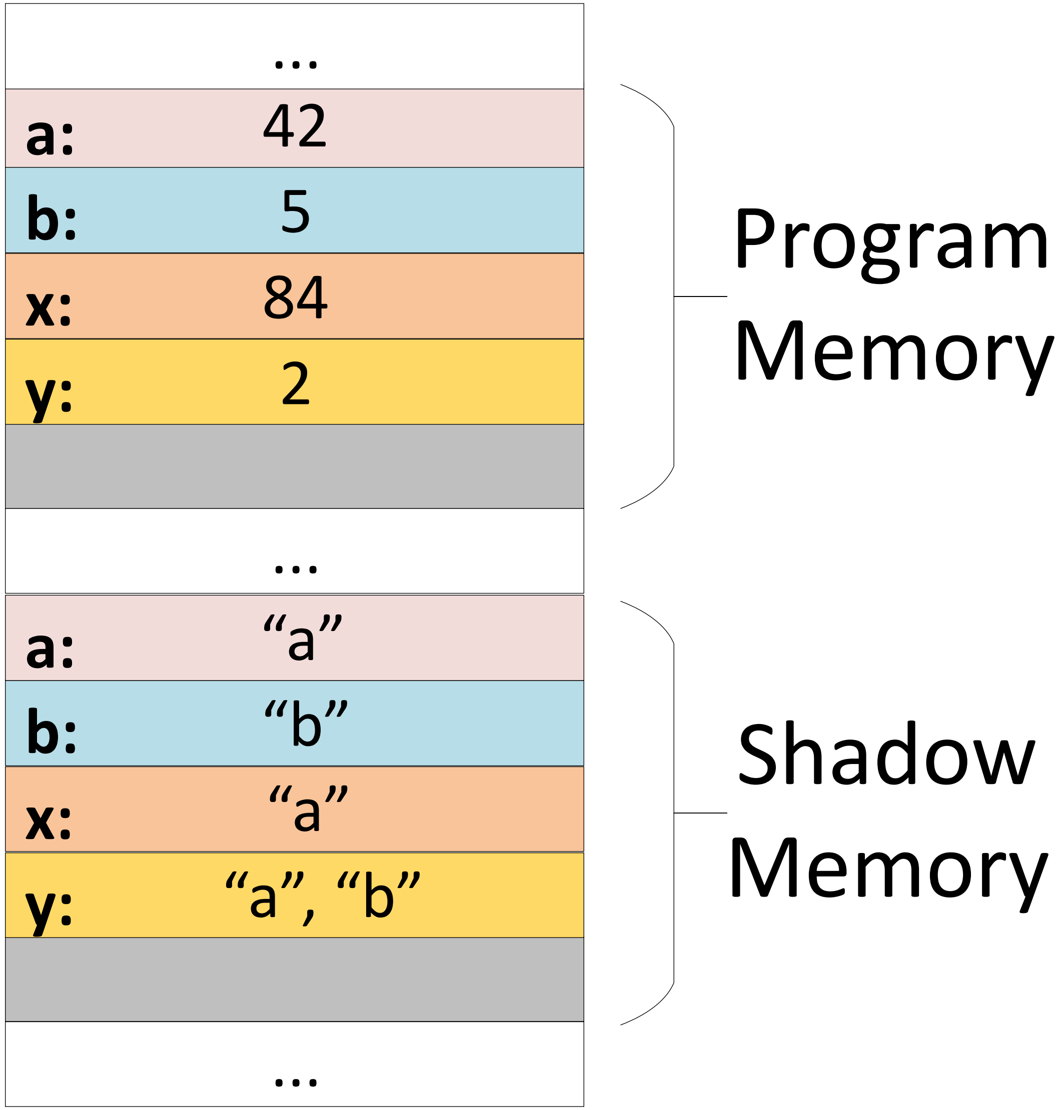


Taint Analysis: track parameters propagation

```
int a = 42;
int b; MPI_Comm_size(&b, comm);
taint_variable(a);

// Data-flow propagation
int x = 2 * a;
int y = modulo(a, b);

// Control-flow propagation
int z = 10;
if(a != 43)
  z = 6;
```

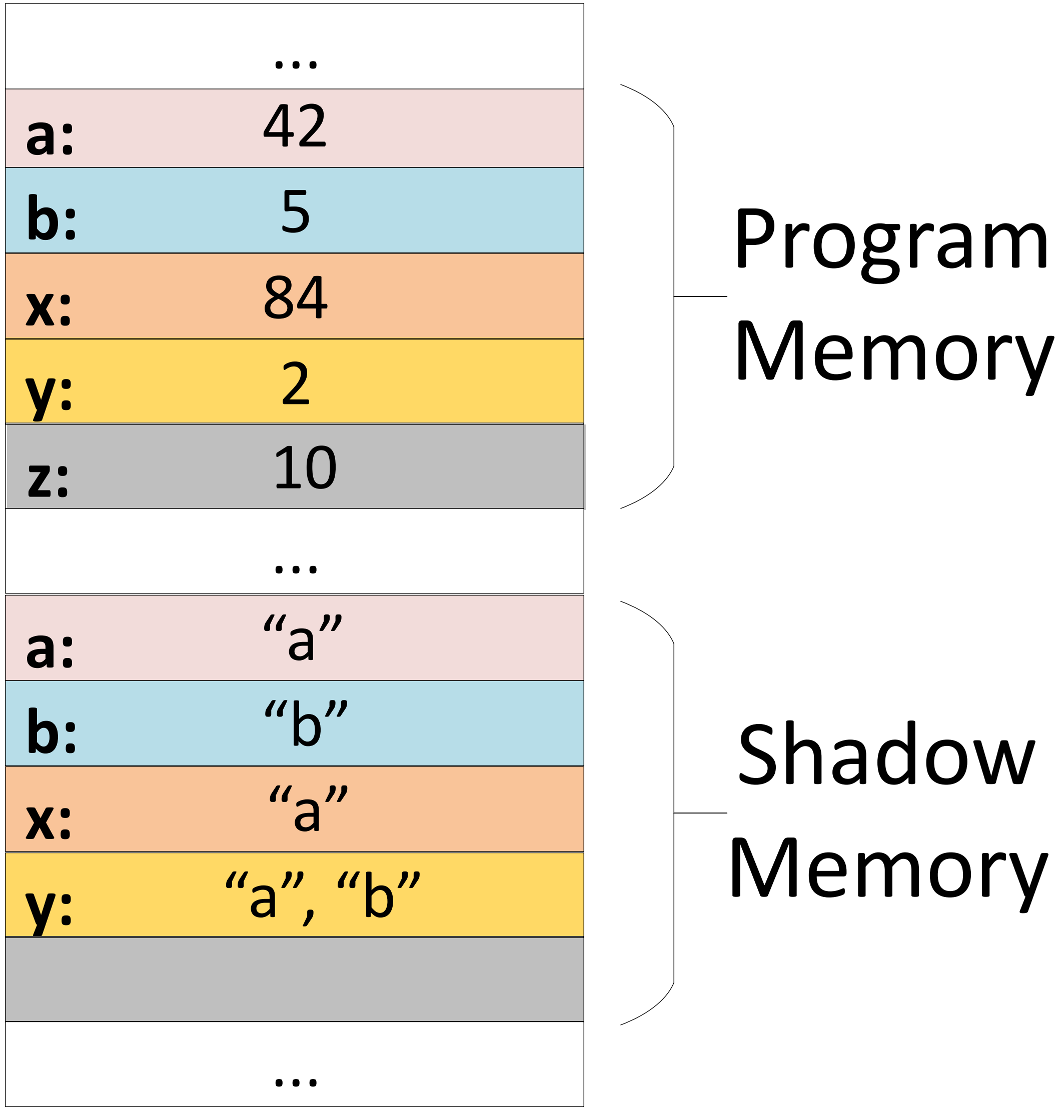


Taint Analysis: track parameters propagation

```
int a = 42;
int b; MPI_Comm_size(&b, comm);
taint_variable(a);

// Data-flow propagation
int x = 2 * a;
int y = modulo(a, b);

// Control-flow propagation
int z = 10;
if(a != 43)
  z = 6;
```



Taint Analysis: track parameters propagation

```
int a = 42;
int b; MPI_Comm_size(&b, comm);
taint_variable(a);

// Data-flow propagation
int x = 2 * a;
int y = modulo(a, b);

// Control-flow propagation
int z = 10;
if(a != 43)
  z = 6;
```

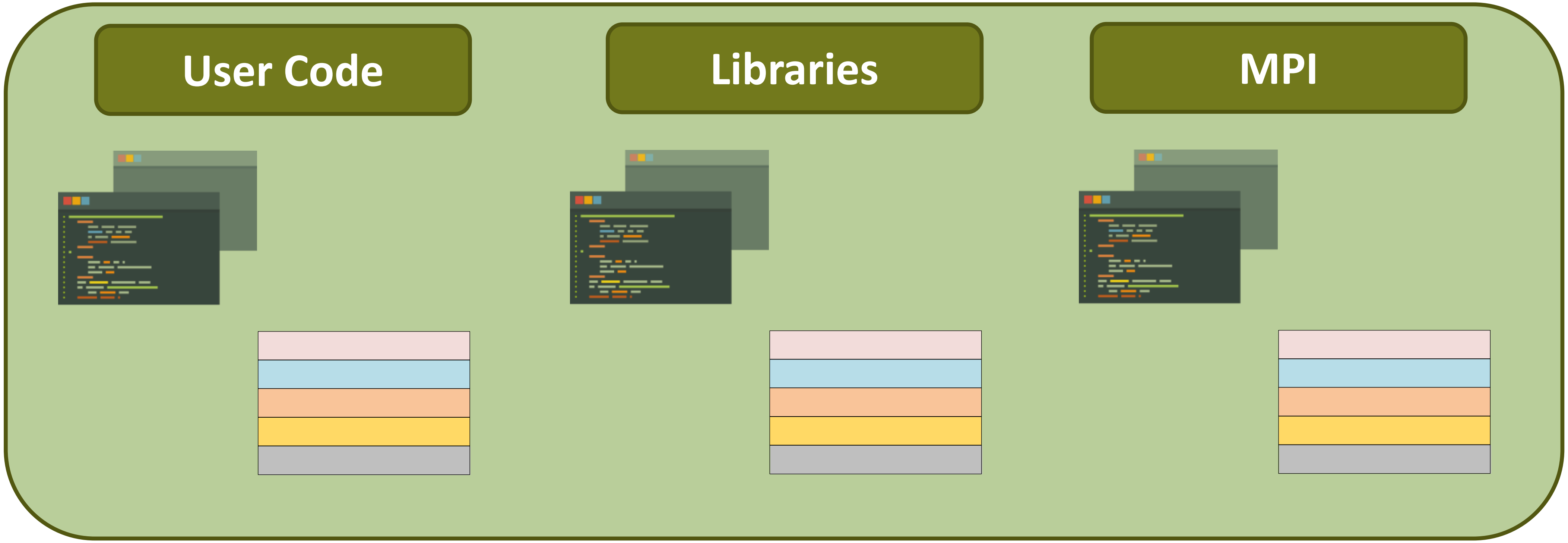
Program Memory

	…
a:	42
b:	5
x:	84
y:	2
z:	6
	…

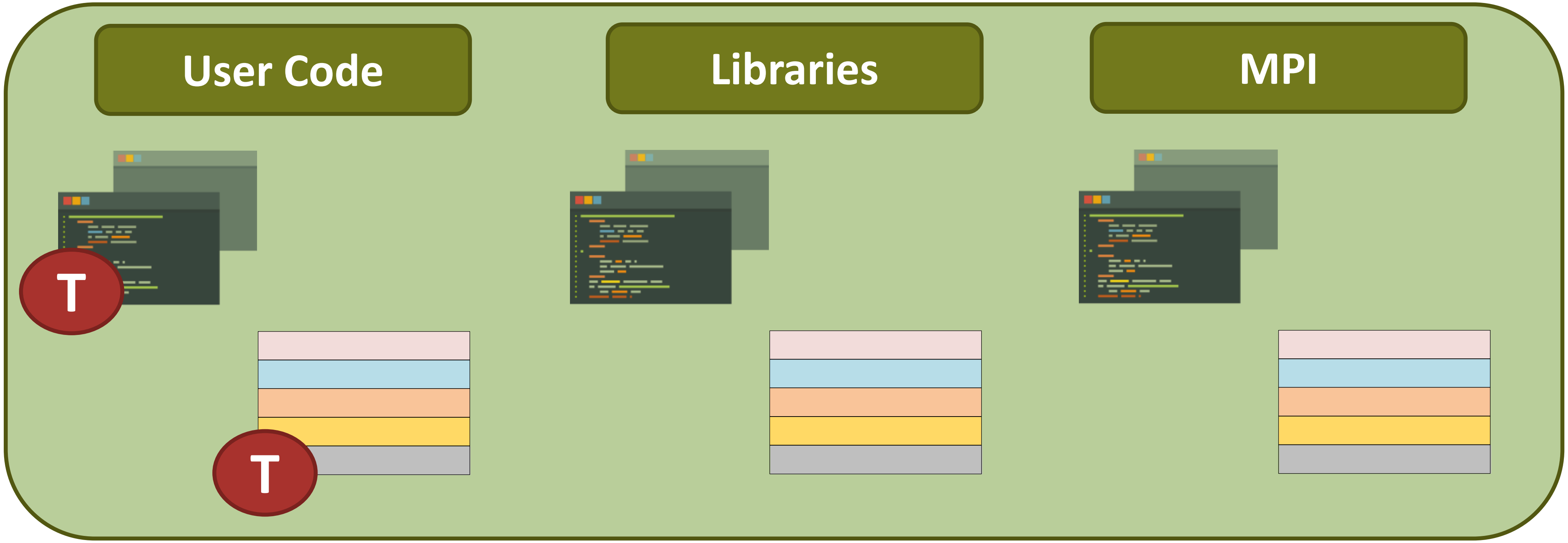
Shadow Memory

a:	“a”
b:	“b”
x:	“a”
y:	“a”, “b”
z:	“a”
	…

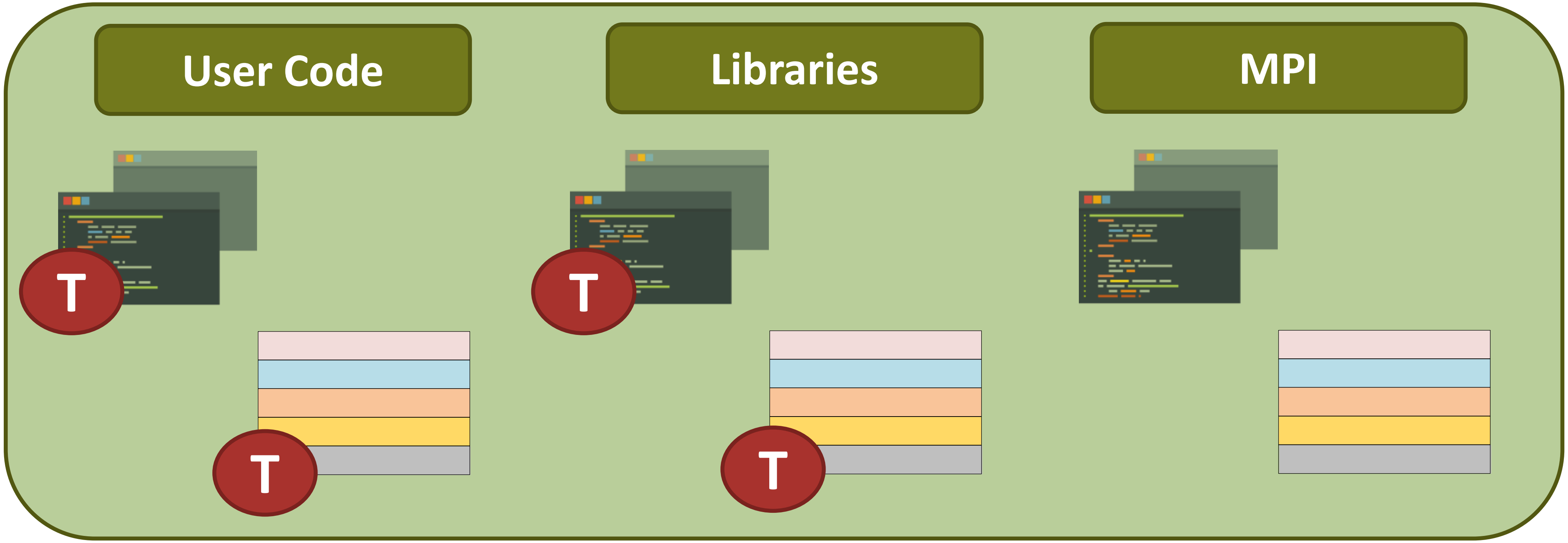
Support for MPI



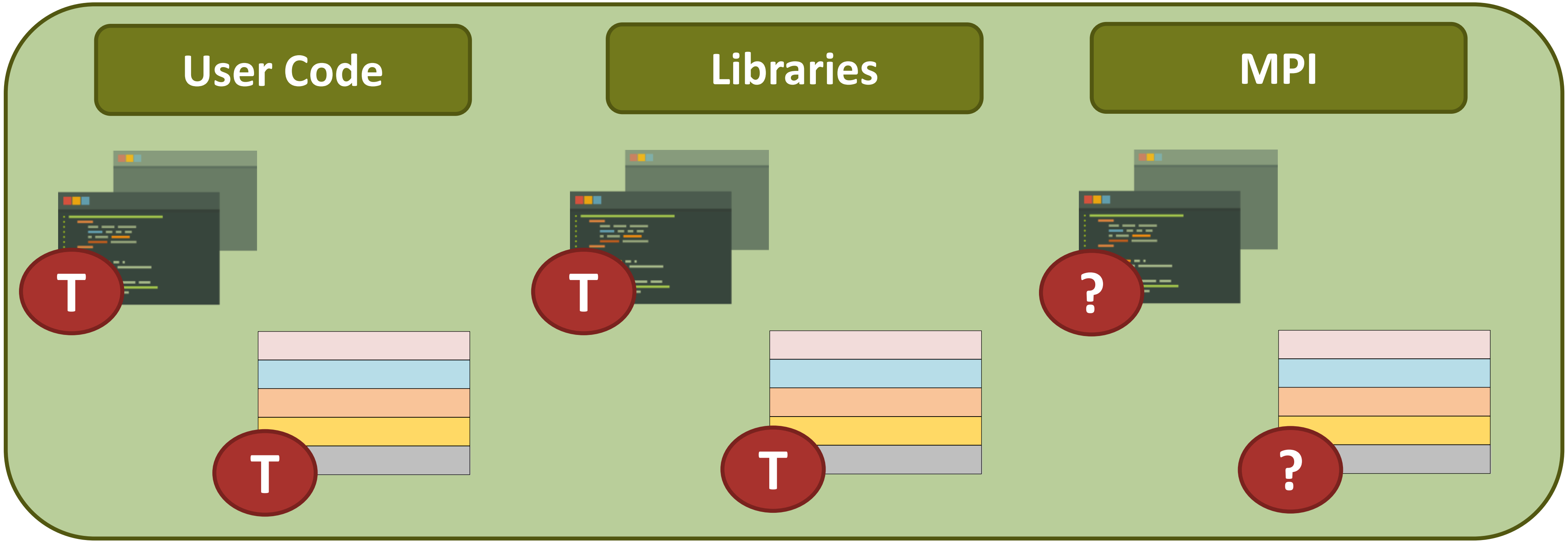
Support for MPI



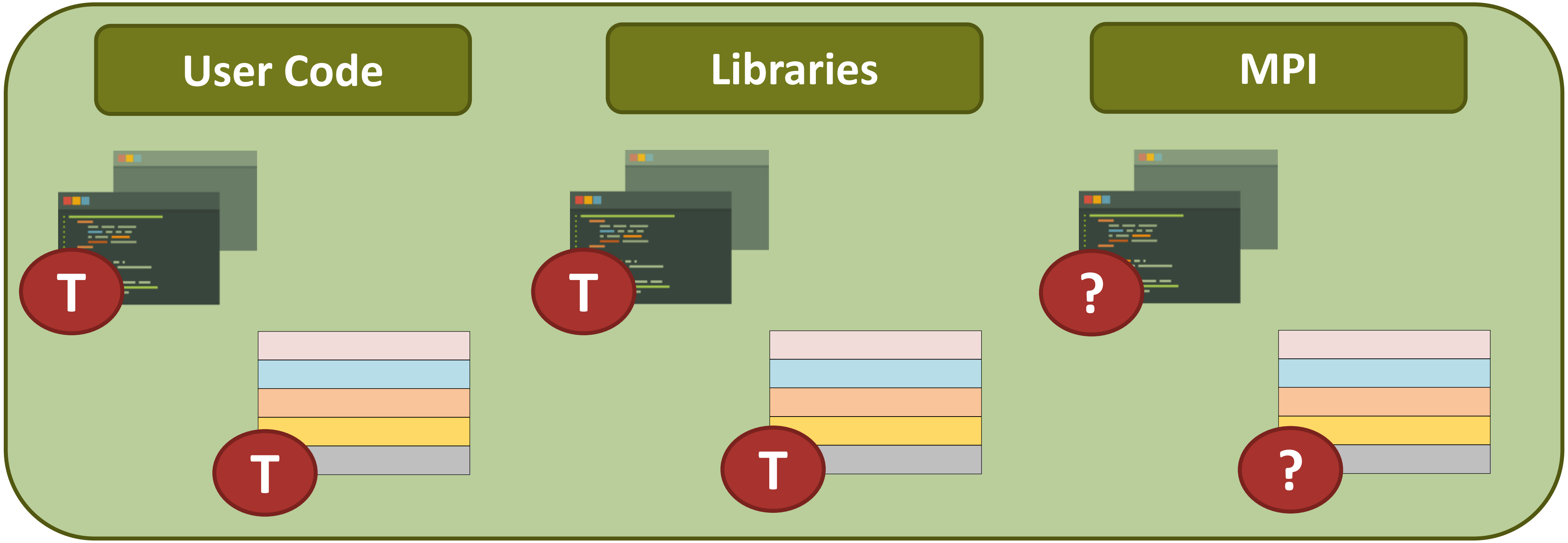
Support for MPI



Support for MPI



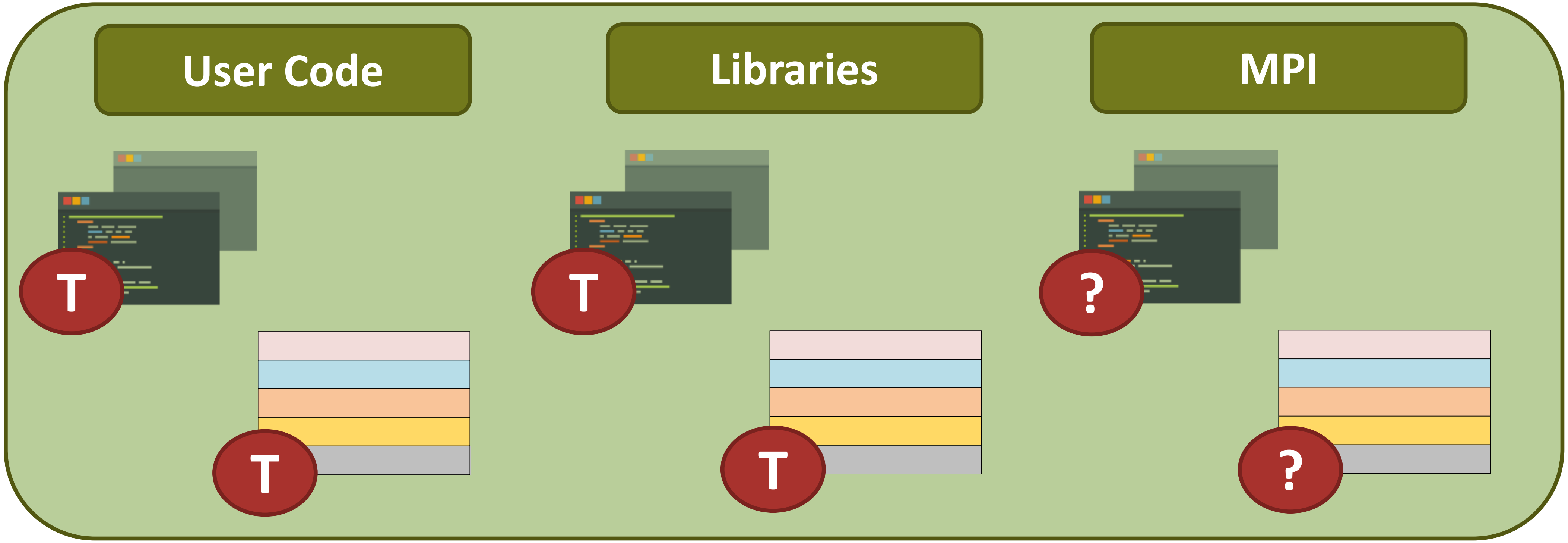
Support for MPI



Compile MPI with taint support?

- ❌ No support for closed implementations.
- ❌ Troublesome deployment on a cluster.

Support for MPI



Compile MPI with taint support?

- ✗ No support for closed implementations.
- ✗ Troublesome deployment on a cluster.

Define taint characteristics of MP

- ✓ No recompilation needed.
- ✓ Works with every implementation.

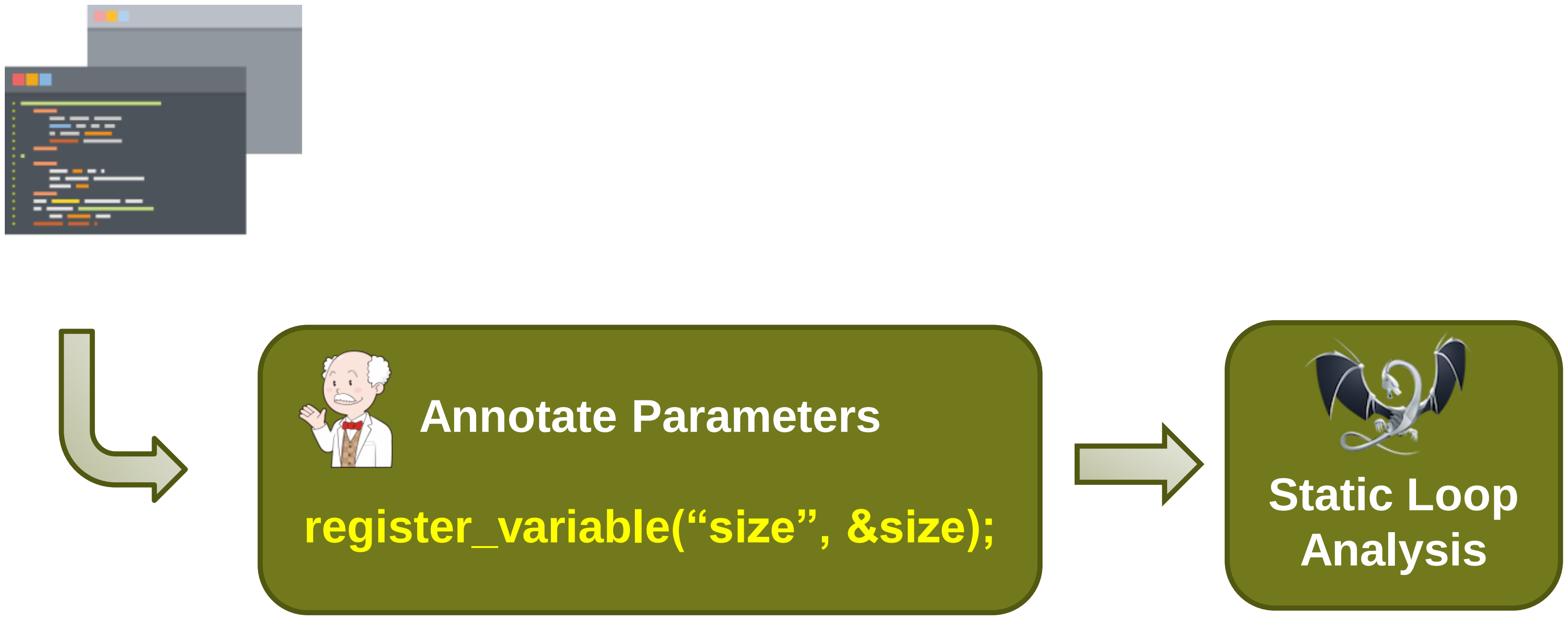
perf-taint: Hybrid Taint Analysis

perf-taint: Hybrid Taint Analysis

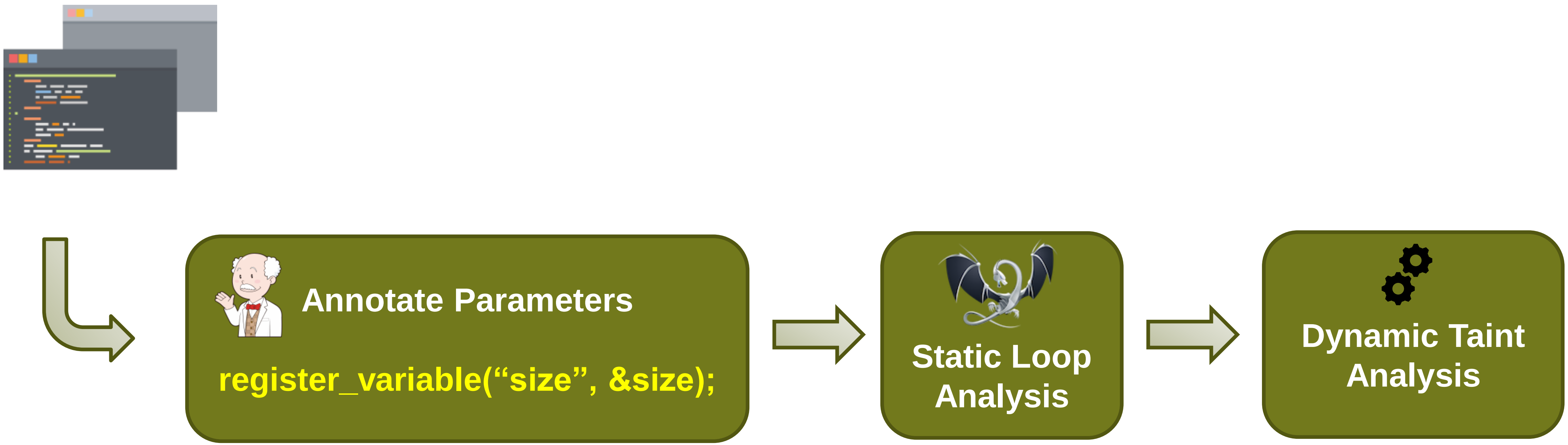
Annotate Parameters

`register_variable(“size”, &size);`

perf-taint: Hybrid Taint Analysis



perf-taint: Hybrid Taint Analysis



perf-taint: Hybrid Taint Analysis

Annotate Parameters

register_variable("size", &size);

→ Static Loop Analysis

→ Dynamic Taint Analysis

↓

(1) Parametric Dependencies
(2) Constant Functions

How do we apply this knowledge?

How do we apply this knowledge?

Parameters Identification

Select problem size **s** and ranks **p** as model parameters.

How do we apply this knowledge?

Parameters Identification

Select problem size **s** and ranks **p** as model parameters.

Expert selects parameters.

How do we apply this knowledge?

Parameters Identification

Select problem size **s** and ranks **p** as model parameters.

Expert selects parameters.

Taint-based coverage selects parameters.

How do we apply this knowledge?

Parameters Identification → **Experiment Design**

Select problem size **s** and ranks **p** as model parameters.

Decide to use
- **5** values per parameter
- **5** samples per experiment

25 combinations of **p** and **s**

Expert selects parameters.

Taint-based coverage selects parameters.

How do we apply this knowledge?

Parameters Identification → **Experiment Design**

Select problem size **s** and ranks **p** as model parameters.

Decide to use
- **5** values per parameter
- **5** samples per experiment

25 combinations of **p** and **s**

Expert selects parameters.

Use complex heuristics.

Taint-based coverage selects parameters.

How do we apply this knowledge?

Parameters Identification → **Experiment Design**

Select problem size **s** and ranks **p** as model parameters.

Decide to use
- **5** values per parameter
- **5** samples per experiment

25 combinations of **p** and **s**

Expert selects parameters.

Use complex heuristics.

Taint-based coverage selects parameters.

Use parameter dependencies.

How do we apply this knowledge?

Parameters Identification → **Experiment Design** → **Experiment Execution**

Select problem size **s** and ranks **p** as model parameters.

Decide to use
- **5** values per parameter
- **5** samples per experiment

25 combinations of **p** and **s**

Score-P
Scalable performance measurement infrastructure for parallel codes

25 parameter values
5 repetitions per sample
125 instrumented executions

Expert selects parameters.

Use complex heuristics.

Taint-based coverage selects parameters.

Use parameter dependencies.

How do we apply this knowledge?

Parameters Identification → Experiment Design → Experiment Execution

Parameters Identification

Select problem size **s** and ranks **p** as model parameters.

Experiment Design

Decide to use
- **5** values per parameter
- **5** samples per experiment

25 combinations of **p** and **s**

Experiment Execution

Score-P: Scalable performance measurement infrastructure for parallel codes

25 parameter values
5 repetitions per sample
125 instrumented executions

Expert selects parameters.

Use complex heuristics.

Instrument all functions.

Taint-based coverage selects parameters.

Use parameter dependencies.

How do we apply this knowledge?

Parameters Identification → Experiment Design → Experiment Execution

Parameters Identification

Select problem size **s** and ranks **p** as model parameters.

Experiment Design

Decide to use
- **5** values per parameter
- **5** samples per experiment

25 combinations of **p** and **s**

Experiment Execution

25 parameter values
5 repetitions per sample
125 instrumented executions

Expert selects parameters.

Use complex heuristics.

Instrument all functions.

Taint-based coverage selects parameters.

Use parameter dependencies.

Skip irrelevant functions.

How do we apply this knowledge?

Parameters Identification → **Experiment Design** → **Experiment Execution** → **Extra-P Modeler**

Parameters Identification

Select problem size **s** and ranks **p** as model parameters.

Experiment Design

Decide to use
- **5** values per parameter
- **5** samples per experiment

25 combinations of **p** and **s**

Experiment Execution

Score-P

25 parameter values
5 repetitions per sample
125 instrumented executions

Extra-P Modeler

Create performance model for **each function**.
Example result:

$$2.3\ s^3 + 1.71 \log_2 p - 0.1329$$

Expert selects parameters.

Use complex heuristics.

Instrument all functions.

Taint-based coverage selects parameters.

Use parameter dependencies.

Skip irrelevant functions.

How do we apply this knowledge?

Parameters Identification → **Experiment Design** → **Experiment Execution** → **Extra-P Modeler**

Select problem size **s** and ranks **p** as model parameters.

Decide to use
- **5** values per parameter
- **5** samples per experiment

25 combinations of **p** and **s**

Score-P

25 parameter values
5 repetitions per sample
125 instrumented executions

Create performance model for **each function**.
Example result:

$$2.3\ s^3 + 1.71 \log_2 p - 0.1329$$

Expert selects parameters.

Use complex heuristics.

Instrument all functions.

Model all functions.

Taint-based coverage selects parameters.

Use parameter dependencies.

Skip irrelevant functions.

How do we apply this knowledge?

Parameters Identification → **Experiment Design** → **Experiment Execution** → **Extra-P Modeler**

Parameters Identification: Select problem size **s** and ranks **p** as model parameters.

Experiment Design: Decide to use
- **5** values per parameter
- **5** samples per experiment

25 combinations of **p** and **s**

Experiment Execution (Score-P): **25** parameter values
5 repetitions per sample
125 instrumented executions

Extra-P Modeler: Create performance model for **each function**. Example result:

$$2.3\, s^3 + 1.71 \log_2 p - 0.1329$$

Expert selects parameters.	**Use complex heuristics.**	**Instrument all functions.**	**Model all functions.**
Taint-based coverage selects parameters.	**Use parameter dependencies.**	**Skip irrelevant functions.**	**Model only relevant functions.**

Case Studies

Cost

Fewer Experiments

Cheaper Experiments

Case Studies

Cost

Fewer Experiments

Cheaper Experiments

Quality

Less Intrusion

More Noise Resilience

Case Studies

Cost
Fewer Experiments
Cheaper Experiments

Quality
Less Intrusion
More Noise Resilience

Validity
Experiment Design
Hardware Contention

Case Studies

Case Studies

Piz Daint, 21 nodes

- Intel Xeon E5-2695 v4 2.1 GHz
- 2 sockets, 18 cores each
- 128 GB Memory
- Cray MPICH 7.7.2

Case Studies

Piz Daint, 21 nodes

- Intel Xeon E5-2695 v4 2.1 GHz
- 2 sockets, 18 cores each
- 128 GB Memory
- Cray MPICH 7.7.2

Skylake Cluster, 2 nodes

- Intel Xeon 6154 3 GHz
- 36 cores
- 384 GB Memory
- OpenMPI 4.0.3

Case Studies

Piz Daint, 21 nodes

- Intel Xeon E5-2695 v4 2.1 GHz
- 2 sockets, 18 cores each
- 128 GB Memory
- Cray MPICH 7.7.2

Skylake Cluster, 2 nodes

- Intel Xeon 6154 3 GHz
- 36 cores
- 384 GB Memory
- OpenMPI 4.0.3

LULESH

- p: 27, 64, 81, 125, 343, 729
- *size*: 25, 30, 35, 40, 45
- Taint run: $p = 8$, *size* = 5
- Taint overhead: < 5 mins

Case Studies

Piz Daint, 21 nodes
- Intel Xeon E5-2695 v4 2.1 GHz
- 2 sockets, 18 cores each
- 128 GB Memory
- Cray MPICH 7.7.2

Skylake Cluster, 2 nodes
- Intel Xeon 6154 3 GHz
- 36 cores
- 384 GB Memory
- OpenMPI 4.0.3

LULESH
- *p*: 27, 64, 81, 125, 343, 729
- *size*: 25, 30, 35, 40, 45
- Taint run: *p* = 8, *size* = 5
- Taint overhead: < 5 mins

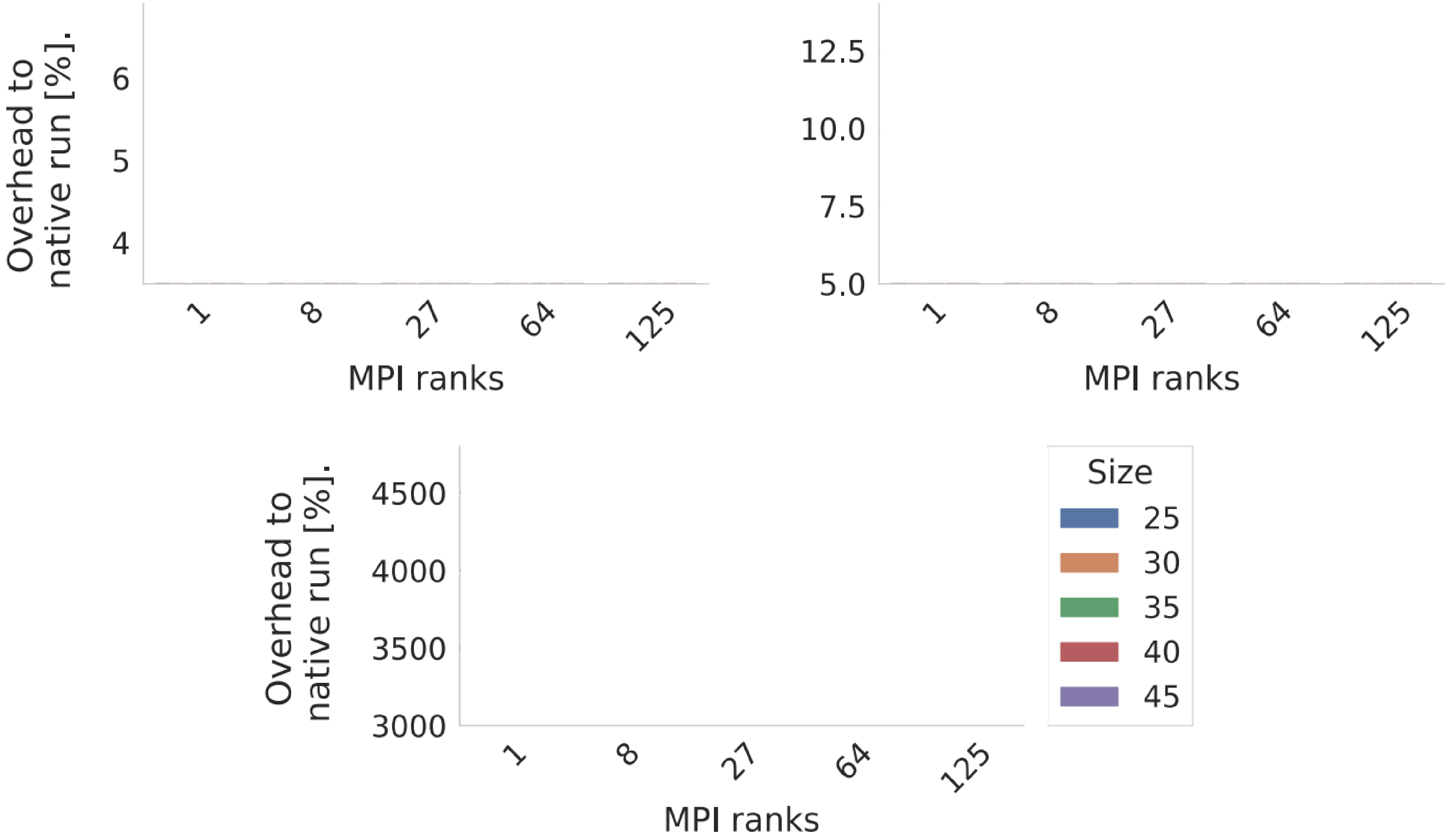
MILC su3_rmd
- *p*: 4, 8, 16, 32, 64
- *size*: 32, 64, 128, 256, 512
- Taint run: *p* = 32, *size* = 128
- Taint overhead: ~1 hr

Parameter Pruning

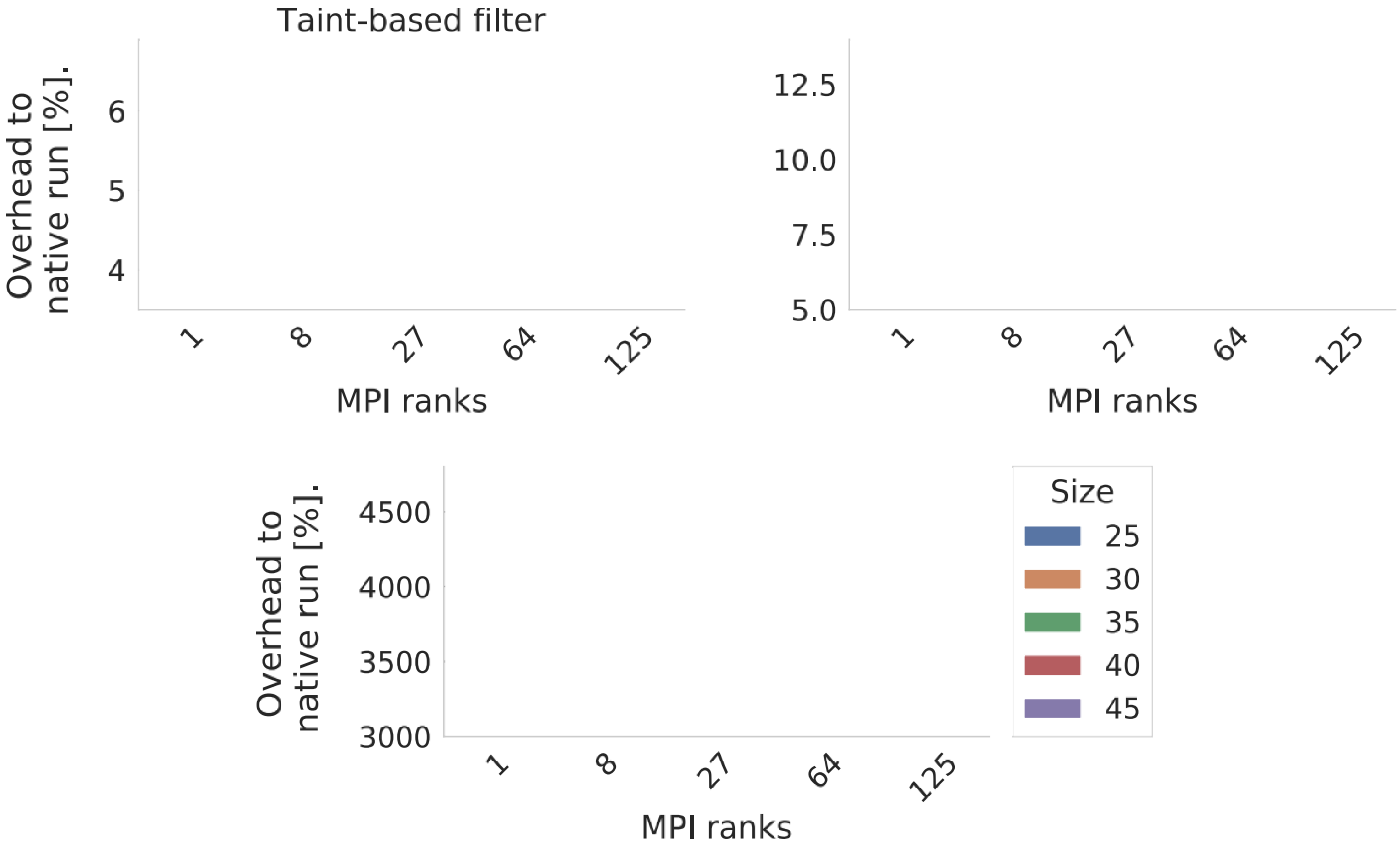
LULESH	Total	Comm	p	size	regions	iters	balance	cost	p, size
Functions [count]	349	2 + 7	2	40	13	4	9	2	40
Loops [count]	275	-	2	78	27	4	20	2	78

MILC	Total	Comm	p	size	trajecs	warms steps	nrest. niter	mass, beta, nfl.	u0	p, size
Functions [count]	629	13 + 8	54	53	12	9	6	1	4	56
Loops [count]	874	-	187	161	39	31	15	1	7	196

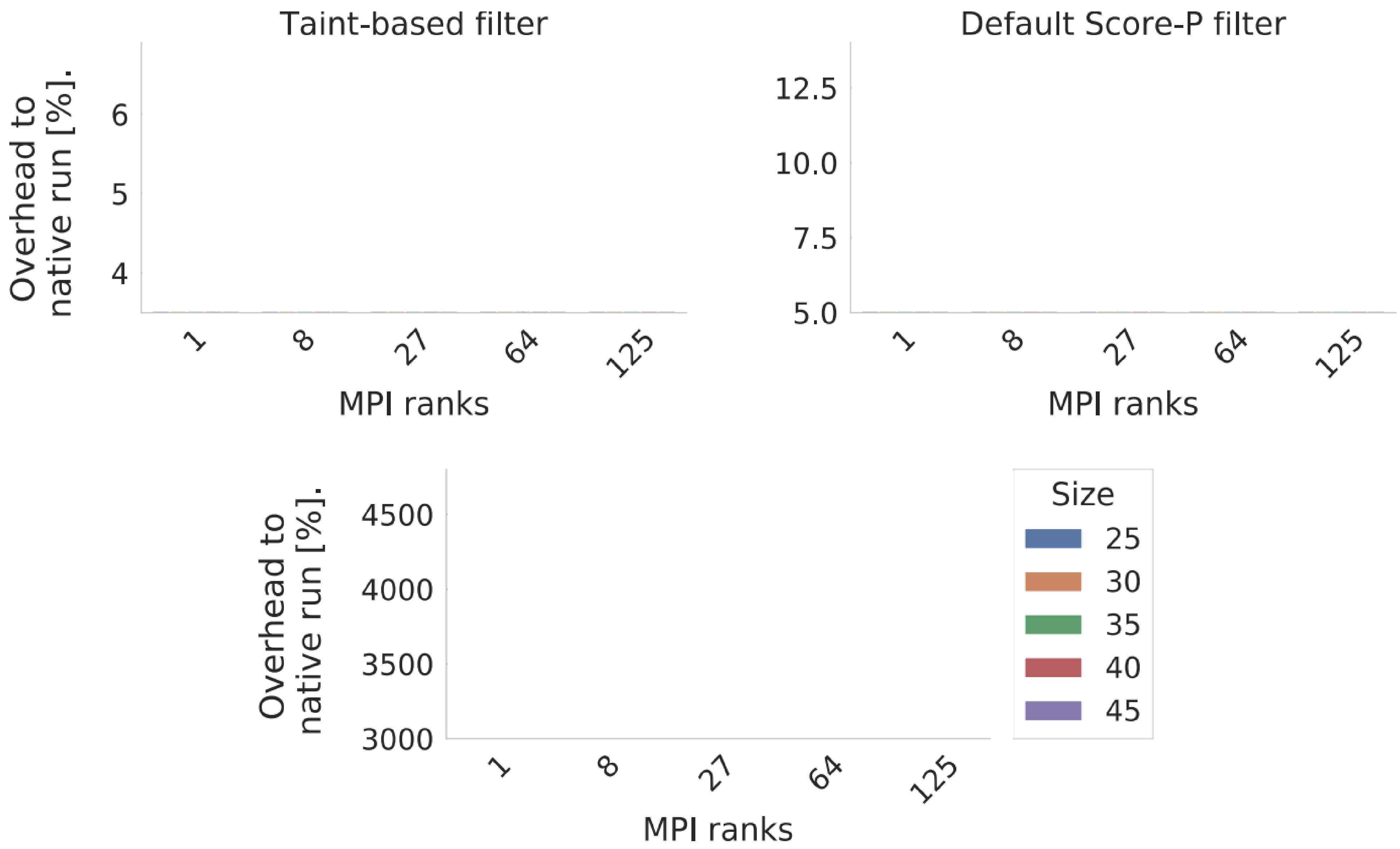
Instrumentation Overhead



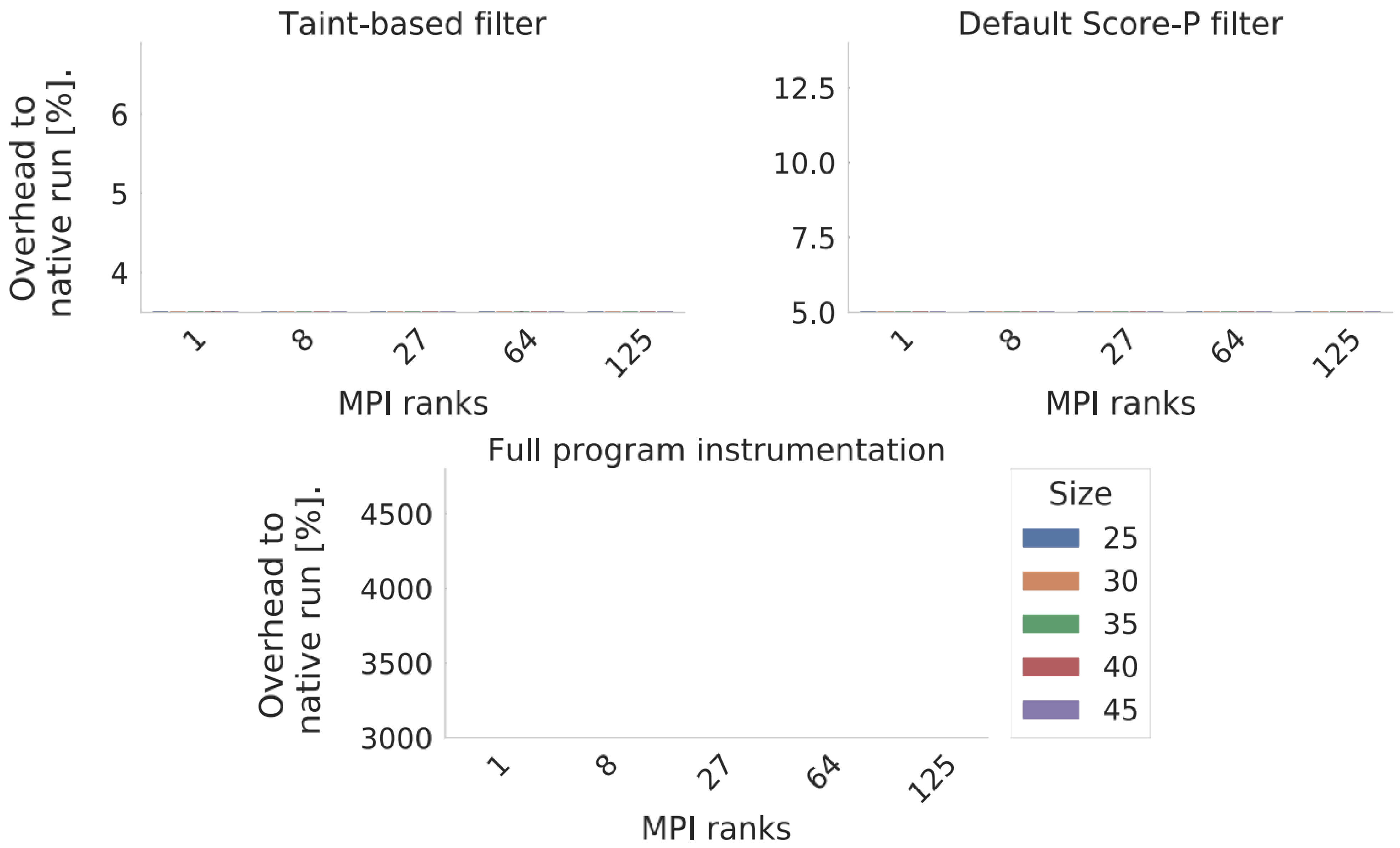
Instrumentation Overhead



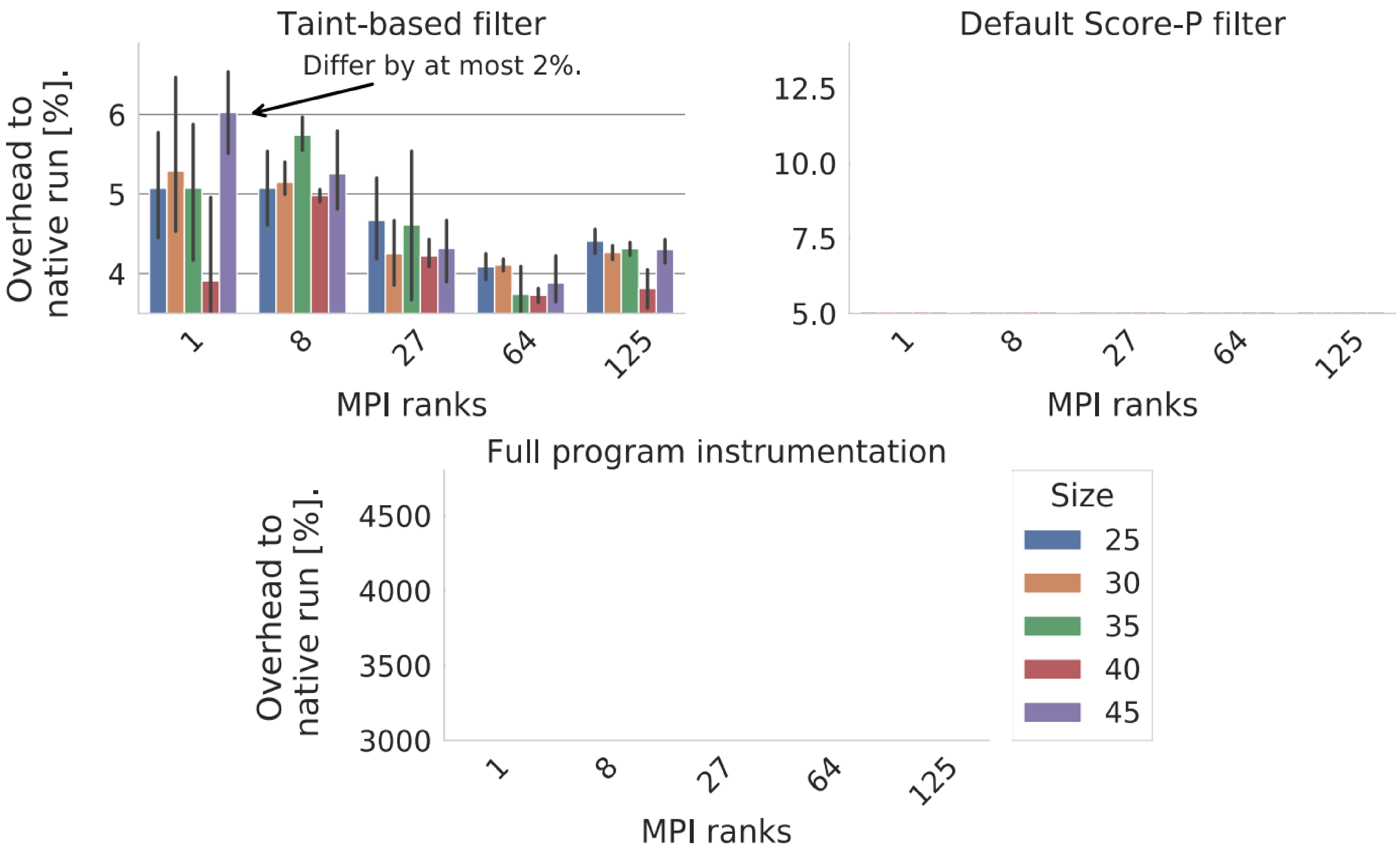
Instrumentation Overhead



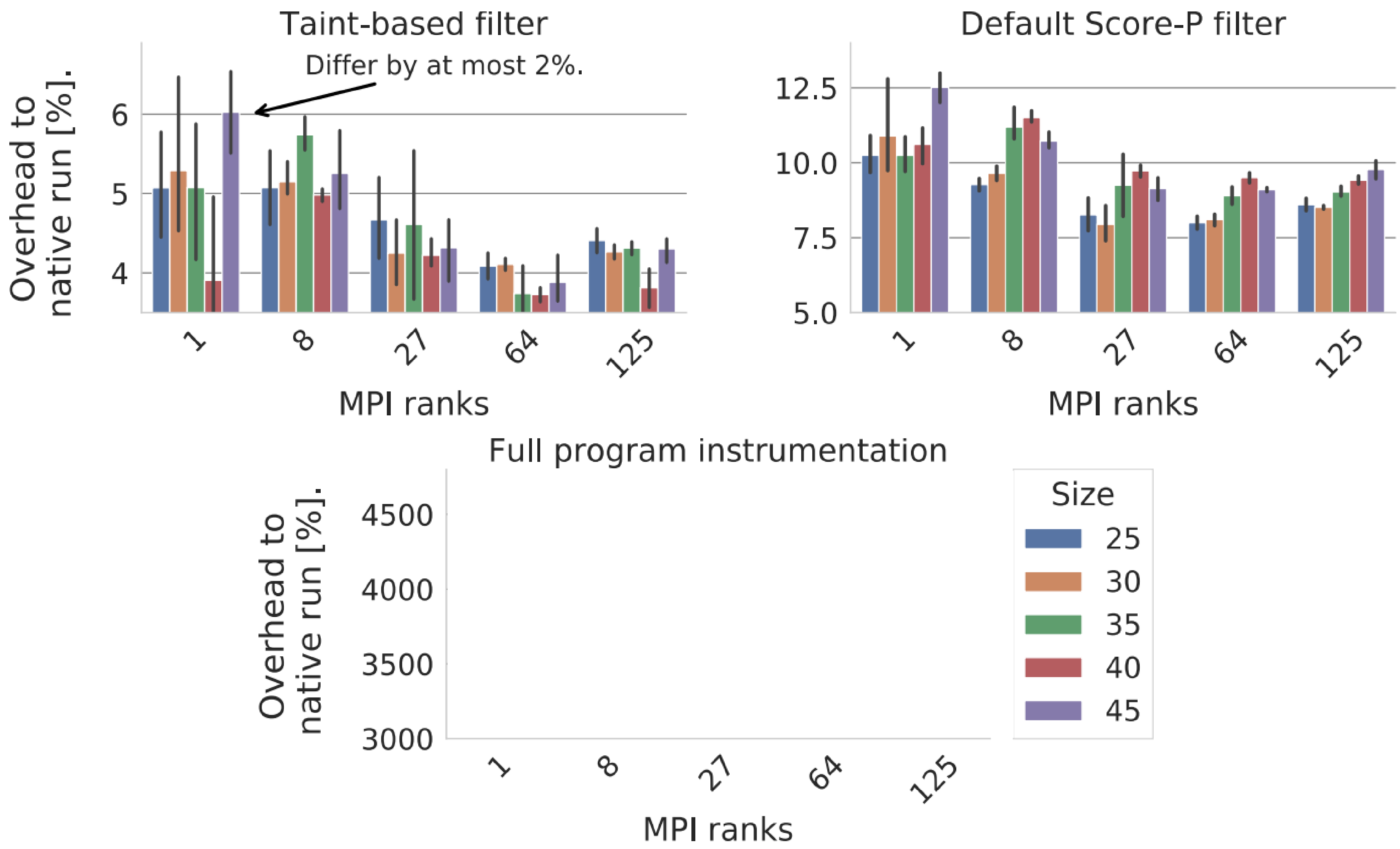
Instrumentation Overhead



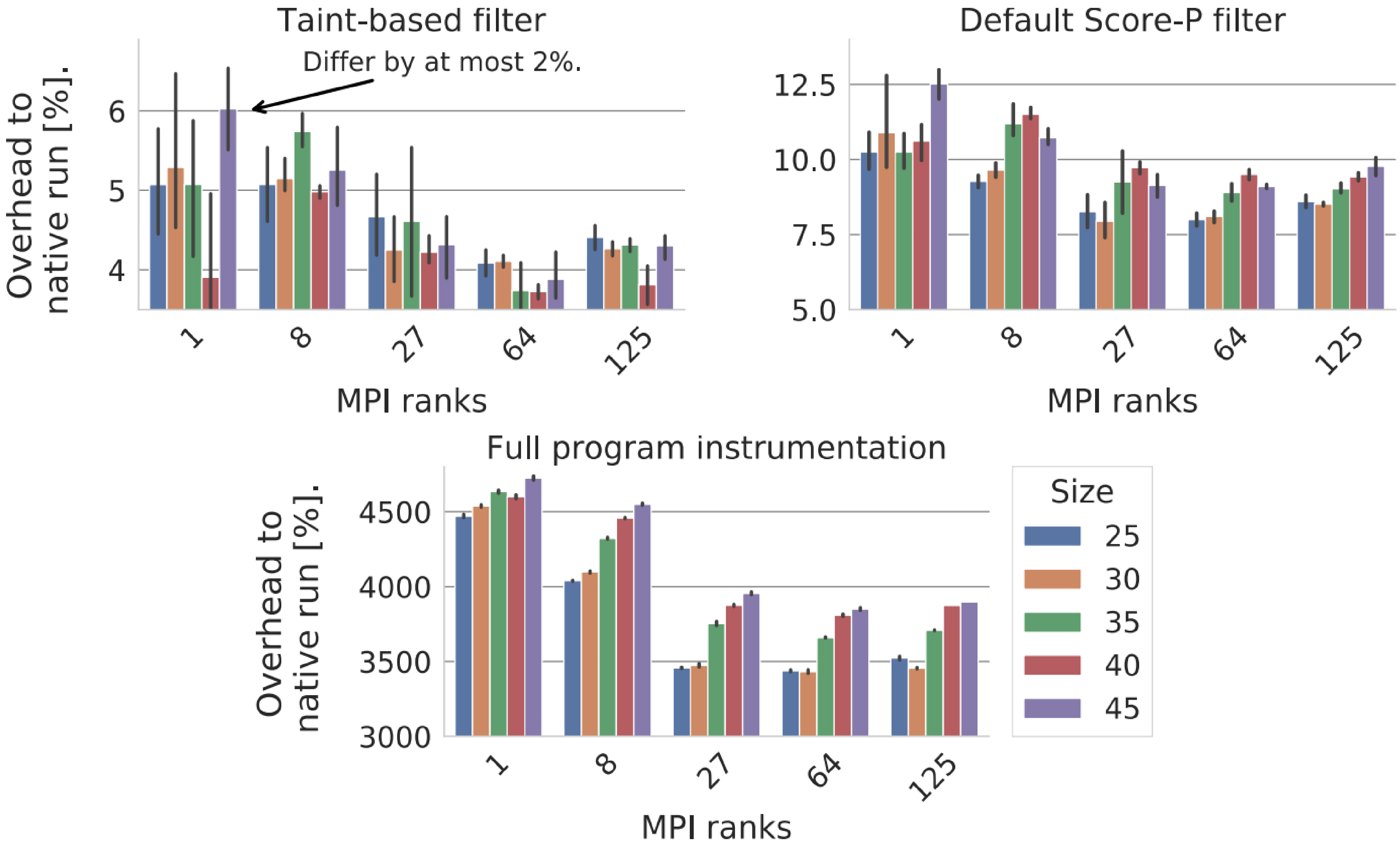
Instrumentation Overhead



Instrumentation Overhead



Instrumentation Overhead



Less Noise

```
int foo(int a, int b, int& result) {
  for(int i = 0 ; i < a; ++i)
       result += b * i;
}
```

$$0.5a + 10^{-3}b$$

Separate **program** from **noise**.

Less Noise

```
int foo(int a, int b, int& result) {
  for(int i = 0 ; i < a; ++i)
       result += b * i;
}
```

$$0.5a + 10^{-3}b$$

Separate **program** from **noise**.

Less Noise

```
int foo(int a, int b, int& result) {
  for(int i = 0 ; i < a; ++i)
      result += b * i;
}
```

$0.5a + 10^{-3}b$

Separate **program** from **noise**.

Less Noise

```
int foo(int a, int b, int& result) {
  for(int i = 0 ; i < a; ++i)
       result += b * i;
}
```

$0.5a + 10^{-3}b$

Separate **program** from **noise**.

LULESH

- 86.2% of functions are constant
- TOP 5 models with perf-taint: parse 14 functions
- Same TOP 5 models with black-box modeling: parse 33 functions

Less Noise

```
int foo(int a, int b, int& result) {
  for(int i = 0 ; i < a; ++i)
      result += b * i;
}
```

$0.5a + 10^{-3}b$

Separate **program** from **noise**.

LULESH

- 86.2% of functions are constant
- TOP 5 models with perf-taint: parse 14 functions
- Same TOP 5 models with black-box modeling: parse 33 functions

MILC su3_rmd

- 87.7% of functions are constant
- TOP 5 models with perf-taint: parse 32 functions
- Same TOP 5 models with black-box modeling: parse 43 functions

Less Intrusion

```
int bar(int a) {
  instrument();
  return a * a;
}
```

```
int foo(int a, int& res) {
  instrument();
  for(int i = 0 ; i < a; ++i)
      res += bar(i);
}
```

$$1.3a + 10^{-4}\sqrt{a}$$

Separate **program** and **instrumentation**.

Less Intrusion

```
int bar(int a) {
  instrument();
  return a * a;
}
```

```
int foo(int a, int& res) {
  instrument();
  for(int i = 0 ; i < a; ++i)
      res += bar(i);
}
```

$1.3a + 10^{-4}\sqrt{a}$

Separate **program** and **instrumentation**.

Less Intrusion

```
int bar(int a) {
  instrument();
  return a * a;
}
```

```
int foo(int a, int& res) {
 instrument();
  for(int i = 0 ; i < a; ++i)
      res += bar(i);
}
```

$1.3a + 10^{-4}\sqrt{a}$

Separate **program** and **instrumentation**.

Less Intrusion

```
int bar(int a) {
  instrument();
  return a * a;
}
```

```
int foo(int a, int& res) {
  instrument();
  for(int i = 0 ; i < a; ++i)
      res += bar(i);
}
```

$1.3a + 10^{-4}\sqrt{a}$

Separate **program** and **instrumentation**.

$3 * 10^{-3} * p^{0.5} + 10^{-5} * size^{3}$

Less Intrusion

```
int bar(int a) {
  instrument();
  return a * a;
}
```

```
int foo(int a, int& res) {
  instrument();
  for(int i = 0 ; i < a; ++i)
       res += bar(i);
}
```

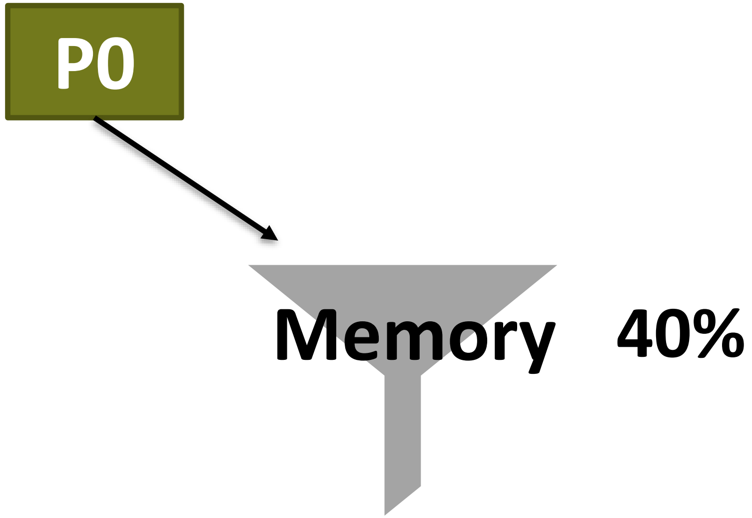
$1.3a + 10^{-4}\sqrt{a}$

Separate **program** and **instrumentation**.

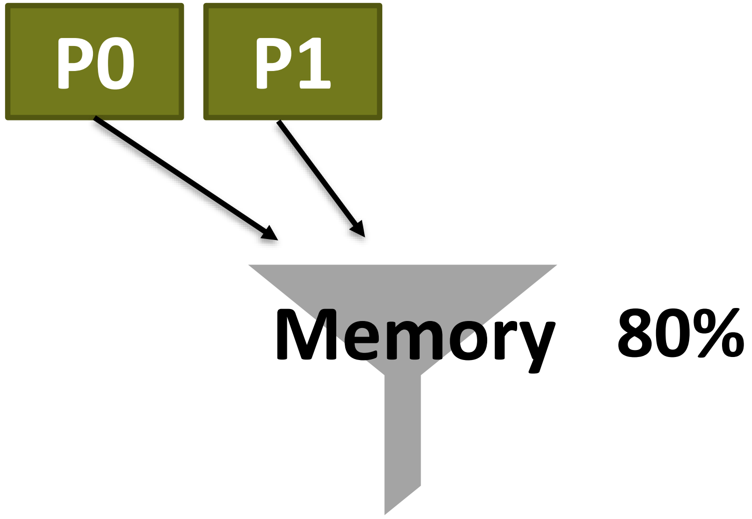
$3 * 10^{-3} * p^{0.5} + 10^{-5} * size^3$

$2.4 * 10^{-8} * p^{0.25} * size^3$

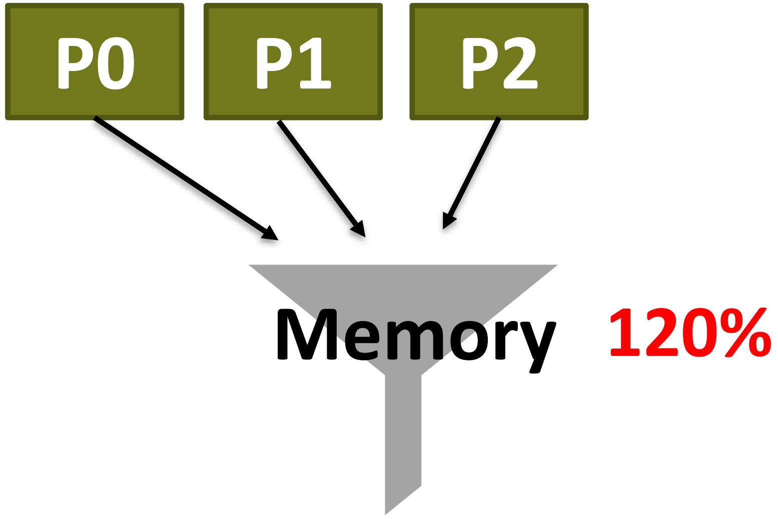
Hardware Contention



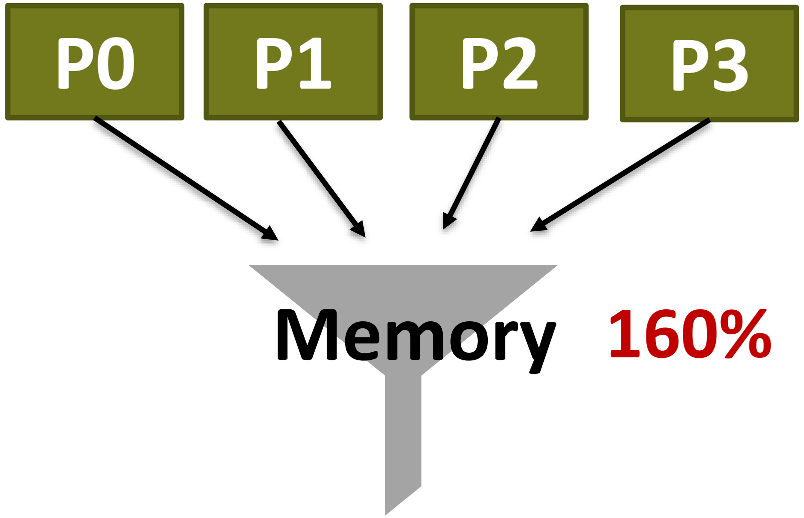
Hardware Contention



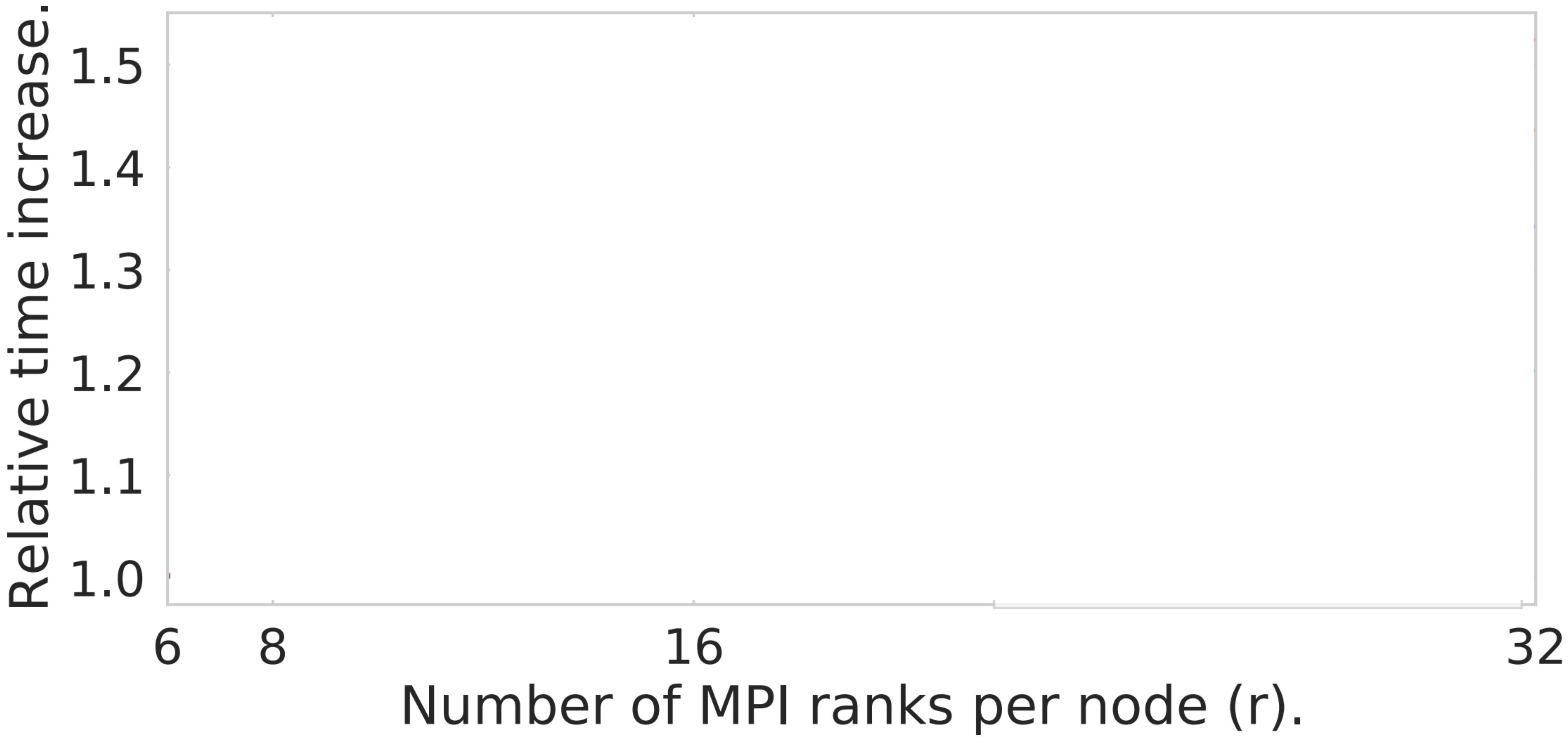
Hardware Contention



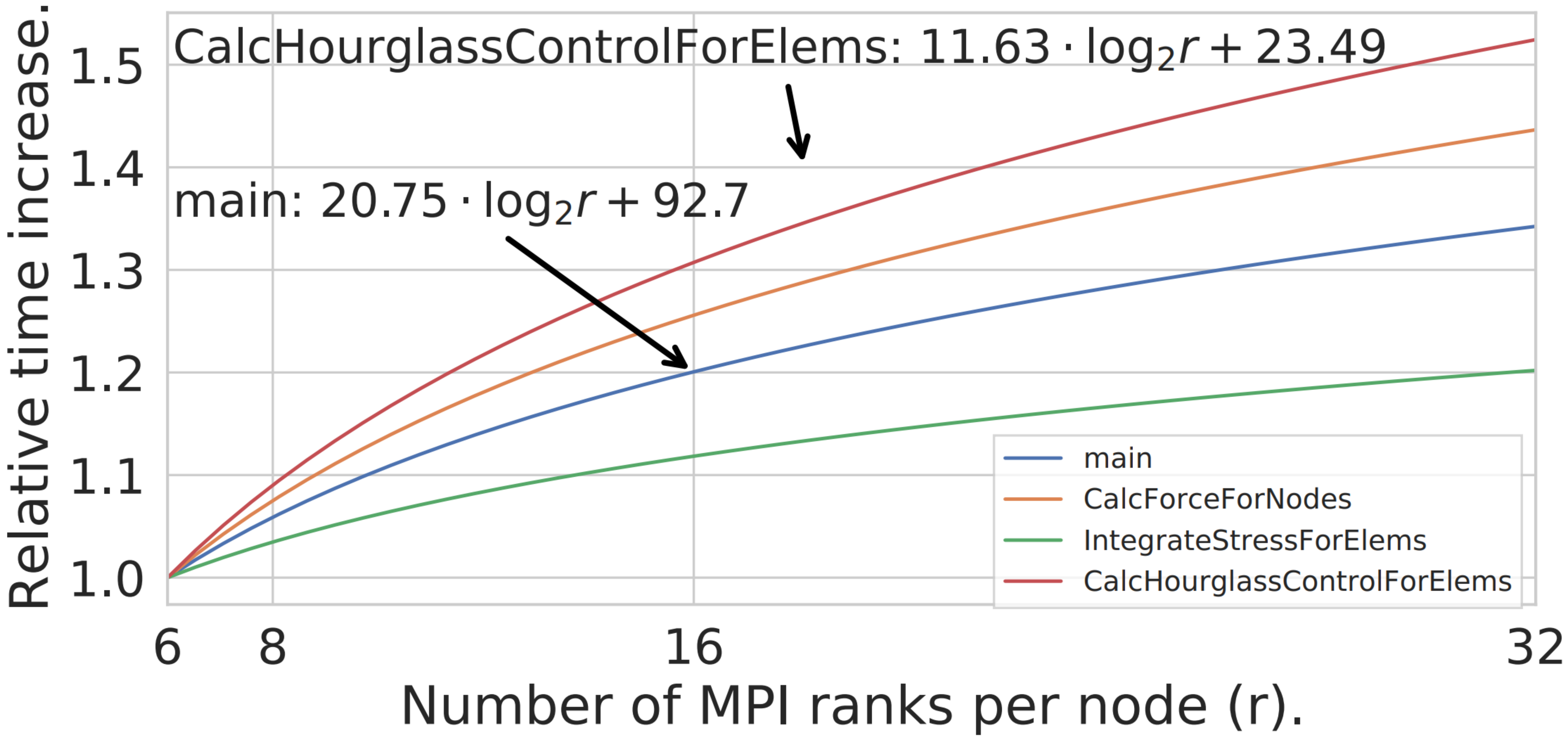
Hardware Contention



Hardware Contention



Hardware Contention



Experiment Design

```
int foo(int a) {
  if(a < 4) kernel_linear(a);
  else      kernel_log(a);
}
```

$$f: \begin{cases} a & a < 4 \\ \log_2 a & a \geq 8 \end{cases}$$

Qualitative change of behavior.
Modeling requires **consistency**.

Experiment Design

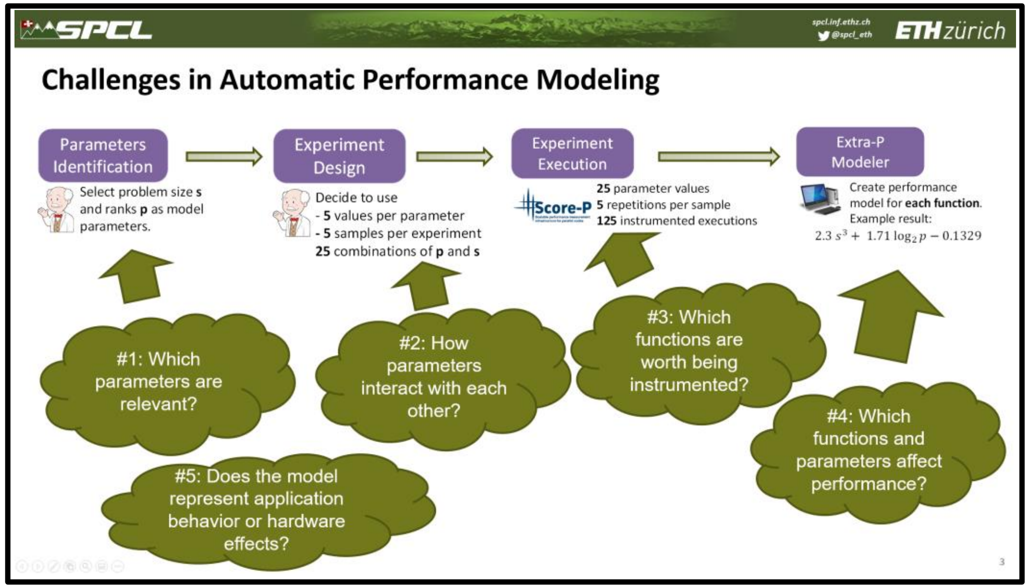
```
int foo(int a) {
   if(a < 4) kernel_linear(a);
   else      kernel_log(a);
}
```

$$f: \begin{cases} a & a < 4 \\ \log_2 a & a \geq 8 \end{cases}$$

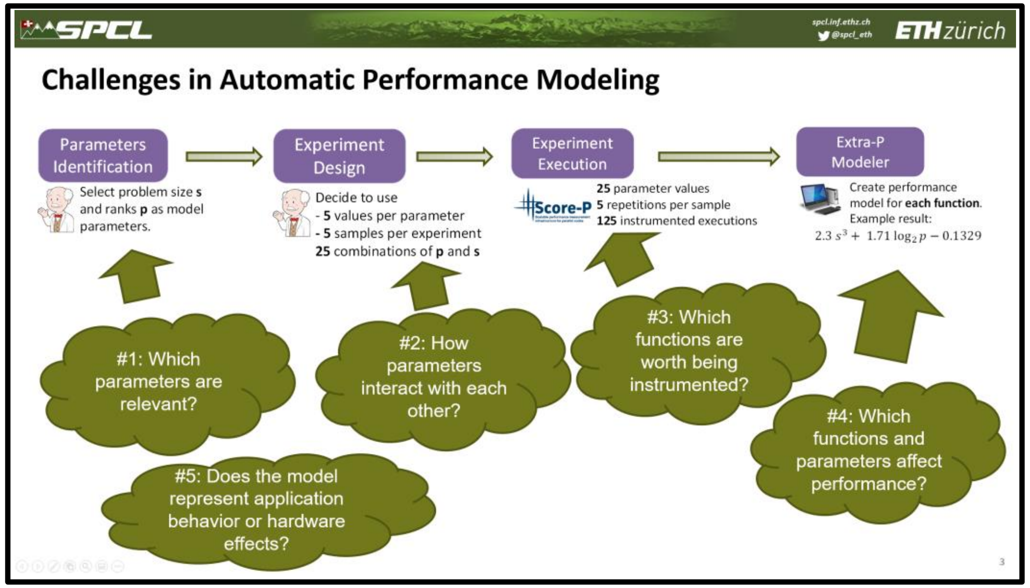
Qualitative change of behavior.
Modeling requires **consistency**.

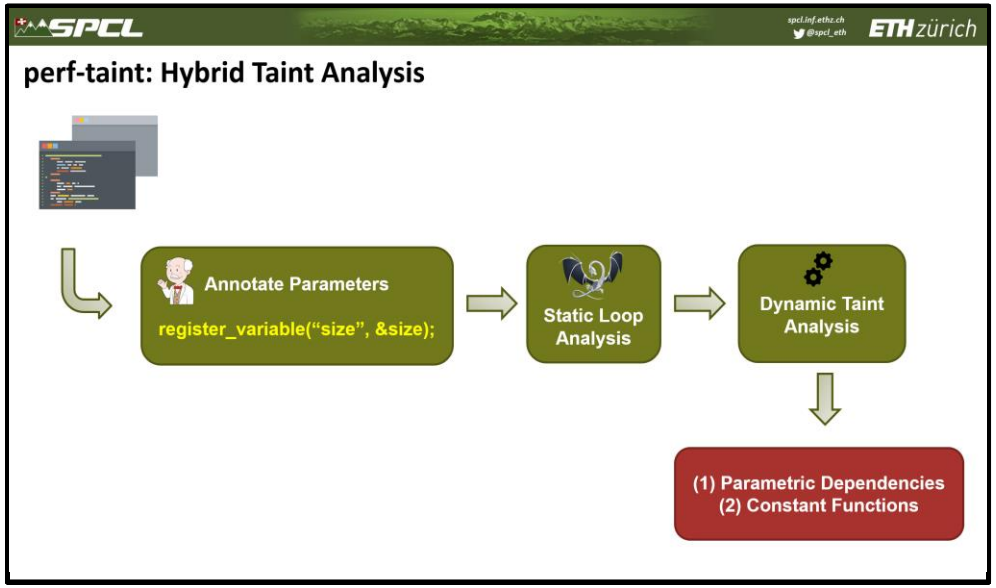
Summary

Summary

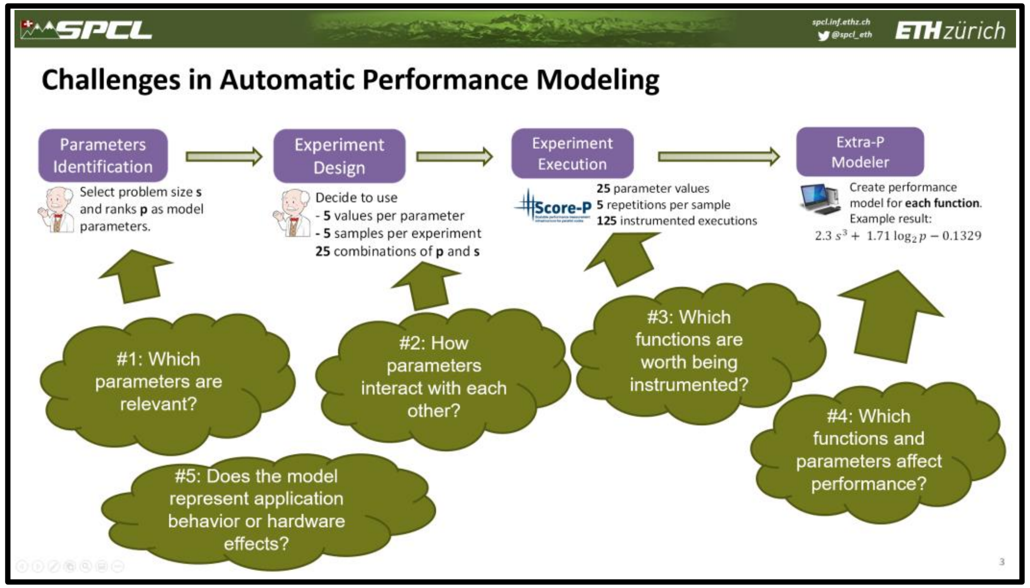

Challenges in Automatic Performance Modeling
Parameters Identification
Select problem size s and ranks p as model parameters.
Experiment Design
Decide to use
- 5 values per parameter
- 5 samples per experiment
25 combinations of p and s
Experiment Execution
Score-P
25 parameter values
5 repetitions per sample
125 instrumented executions
Extra-P Modeler
Create performance model for each function.
Example result:
2.3 s^3 + 1.71 log_2 p − 0.1329
#1: Which parameters are relevant?
#2: How parameters interact with each other?
#3: Which functions are worth being instrumented?
#4: Which functions and parameters affect performance?
#5: Does the model represent application behavior or hardware effects?

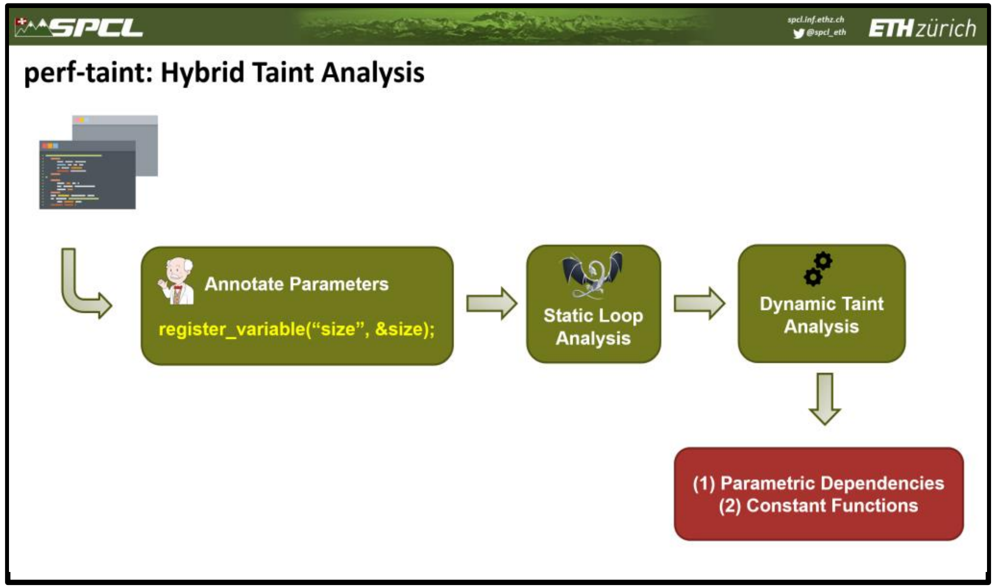
Summary

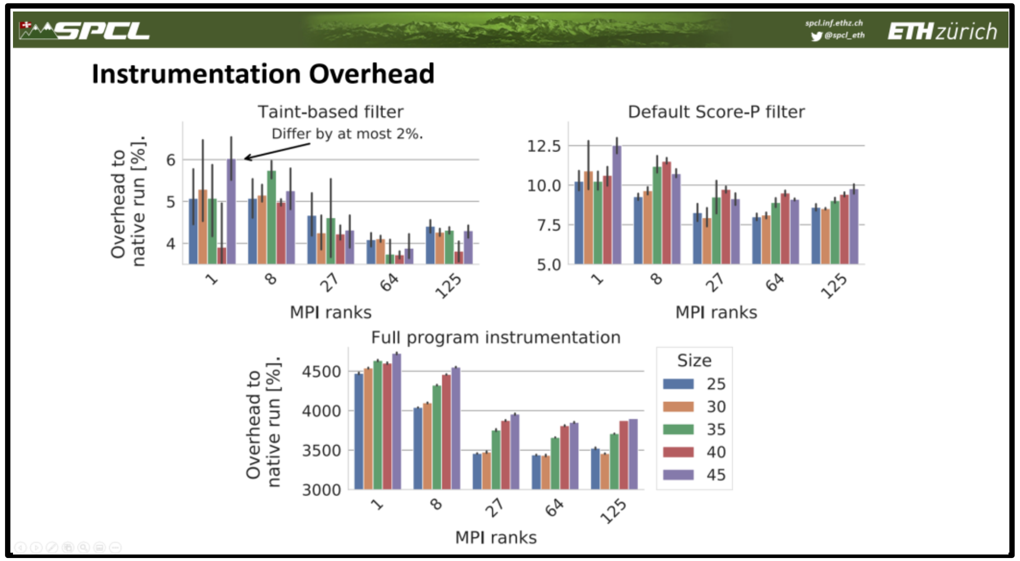

Challenges in Automatic Performance Modeling
Parameters Identification
Select problem size s and ranks p as model parameters.
Experiment Design
Decide to use
- 5 values per parameter
- 5 samples per experiment
25 combinations of p and s
Experiment Execution
Score-P
25 parameter values
5 repetitions per sample
125 instrumented executions
Extra-P Modeler
Create performance model for each function.
Example result:
2.3 s^3 + 1.71 log_2 p − 0.1329
#1: Which parameters are relevant?
#2: How parameters interact with each other?
#3: Which functions are worth being instrumented?
#4: Which functions and parameters affect performance?
#5: Does the model represent application behavior or hardware effects?


perf-taint: Hybrid Taint Analysis
Annotate Parameters
register_variable("size", &size);
Static Loop Analysis
Dynamic Taint Analysis
(1) Parametric Dependencies
(2) Constant Functions

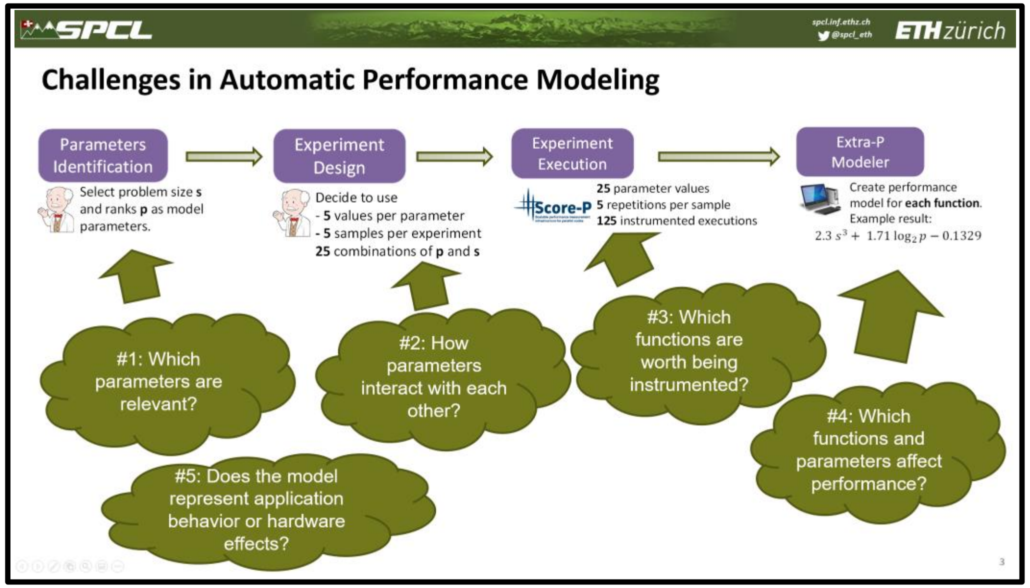
Summary

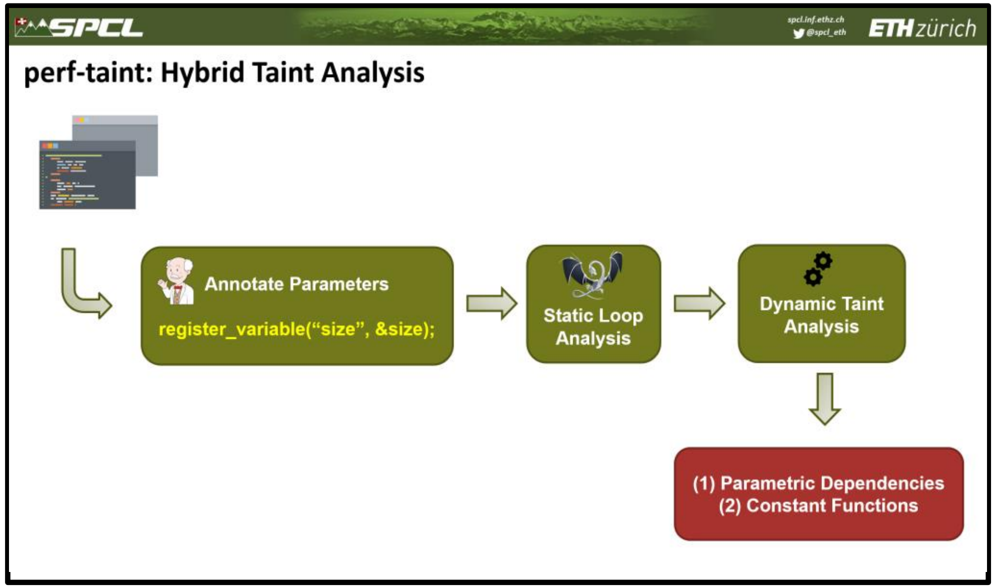

Challenges in Automatic Performance Modeling
Parameters Identification
Select problem size s and ranks p as model parameters.
Experiment Design
Decide to use
- 5 values per parameter
- 5 samples per experiment
25 combinations of p and s
Experiment Execution
25 parameter values
5 repetitions per sample
125 instrumented executions
Extra-P Modeler
Create performance model for each function.
Example result:
$2.3\,s^3 + 1.71\log_2 p - 0.1329$
#1: Which parameters are relevant?
#2: How parameters interact with each other?
#3: Which functions are worth being instrumented?
#4: Which functions and parameters affect performance?
#5: Does the model represent application behavior or hardware effects?

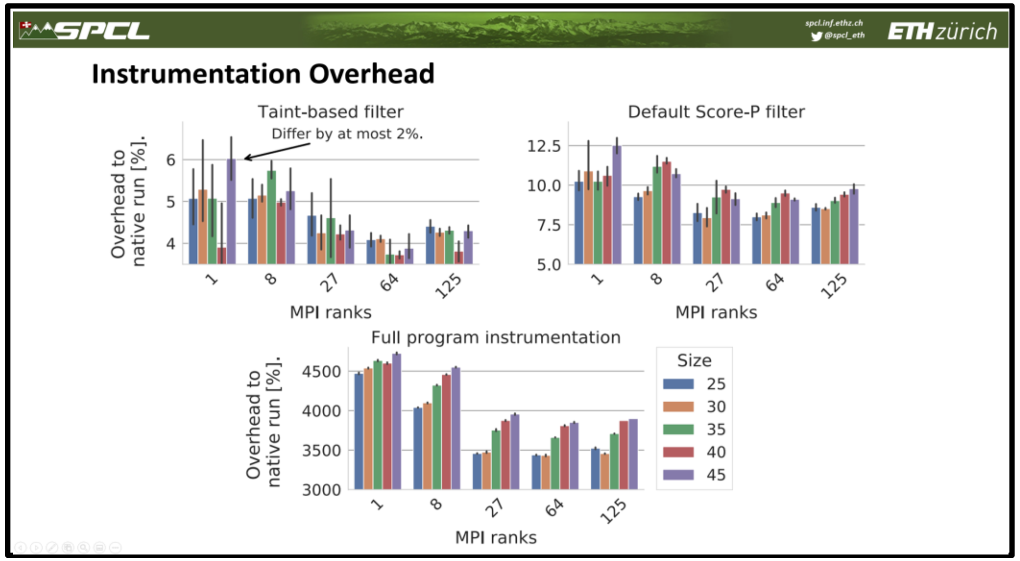

perf-taint: Hybrid Taint Analysis
Annotate Parameters
register_variable("size", &size);
Static Loop Analysis
Dynamic Taint Analysis
(1) Parametric Dependencies
(2) Constant Functions


Instrumentation Overhead
Taint-based filter
Differ by at most 2%.
Default Score-P filter
Full program instrumentation
Overhead to native run [%].
MPI ranks
Size
25
30
35
40
45

Summary







spcl/perf-taint