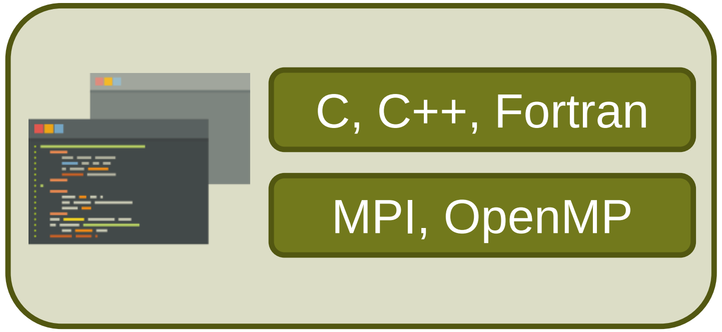
perf-taint: Taint Analysis for Automatic Many-Parameter Performance Modeling

Marcin Copik, Torsten Hoefler (advisor)

Denver, 20th November 2019

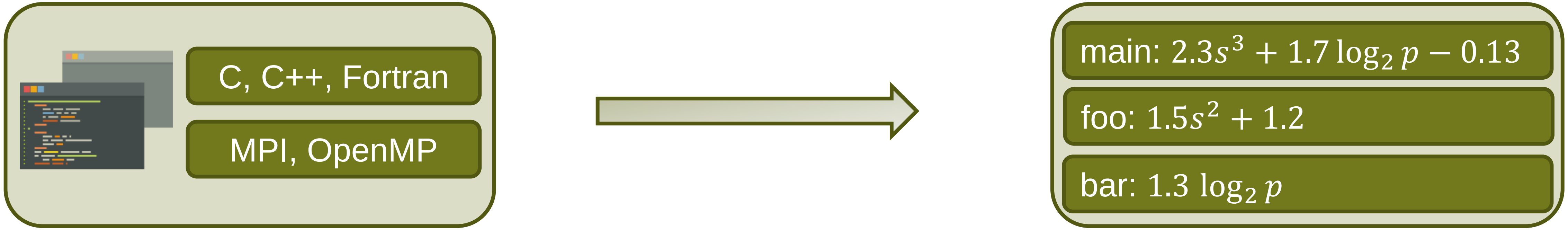
Performance Modeling: state of the art



C, C++, Fortran

MPI, OpenMP

Performance Modeling: state of the art



C, C++, Fortran

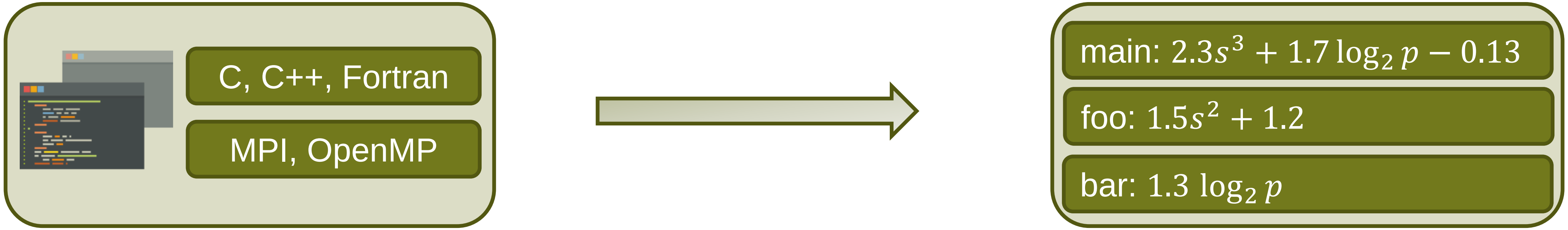
MPI, OpenMP

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

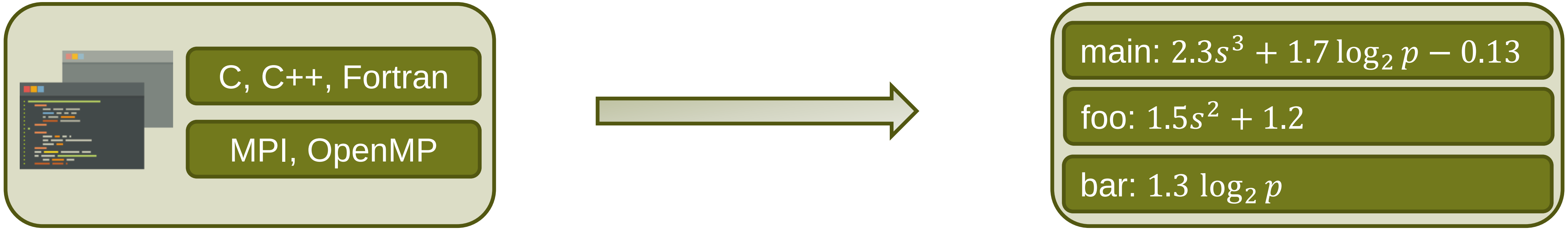
foo: $1.5s^2 + 1.2$

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: $1.5s^2 + 1.2$

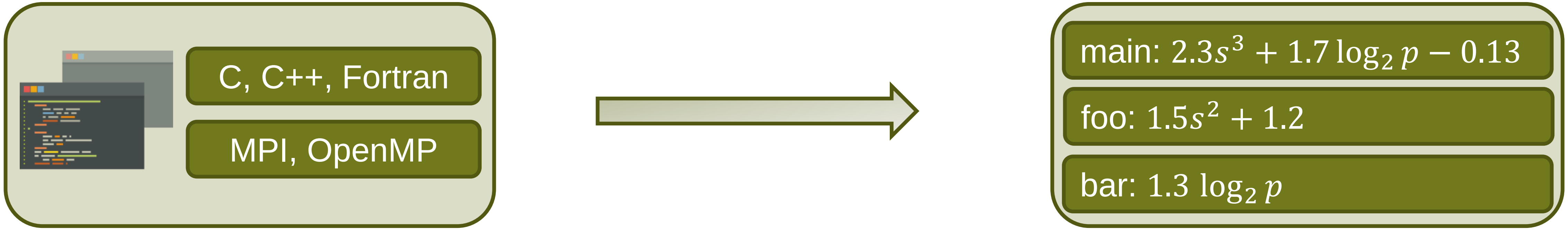
bar: $1.3\log_2 p$

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]

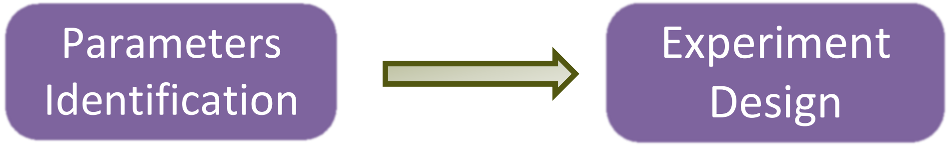
[3] A. Calotoiu, *"Lightweight Requirements Engineering for Exascale Co-design",* **CLUSTER '18**

Challenges in Automatic Performance Modeling

Parameters Identification

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

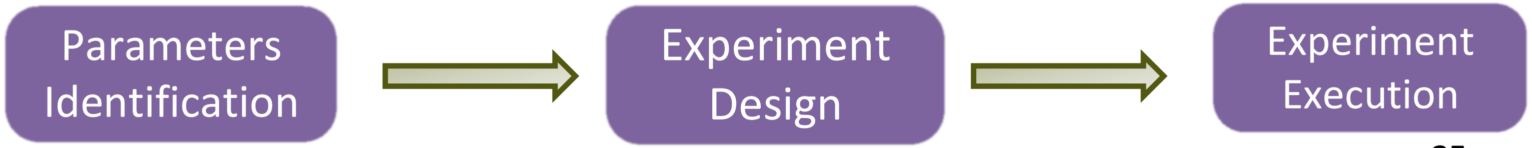
Challenges in Automatic Performance Modeling



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



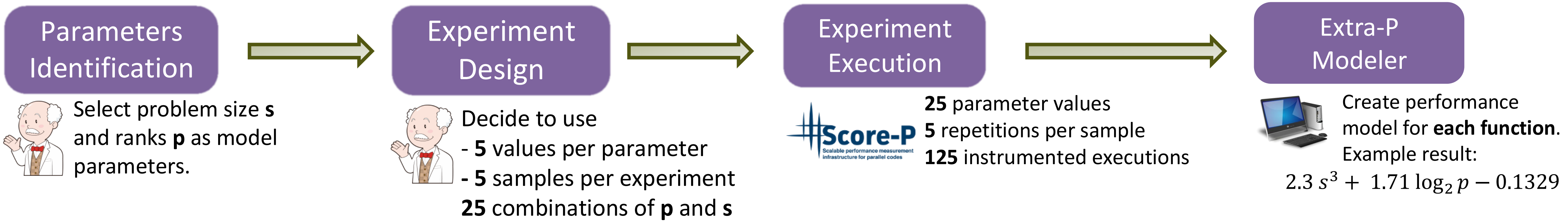
Select problem size **s** and ranks **p** as model parameters.

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

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
Scalable performance measurement infrastructure for parallel codes

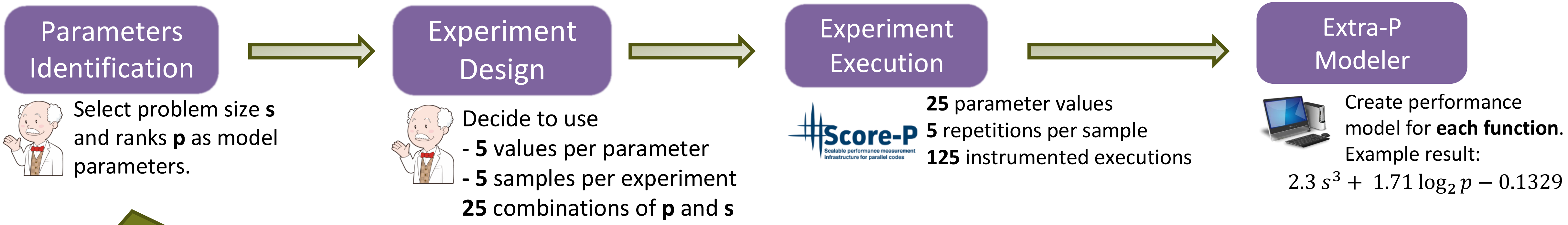
25 parameter values
5 repetitions per sample
125 instrumented executions

Extra-P Modeler

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

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

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
Scalable performance measurement infrastructure for parallel codes

25 parameter values
5 repetitions per sample
125 instrumented executions

Extra-P Modeler

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

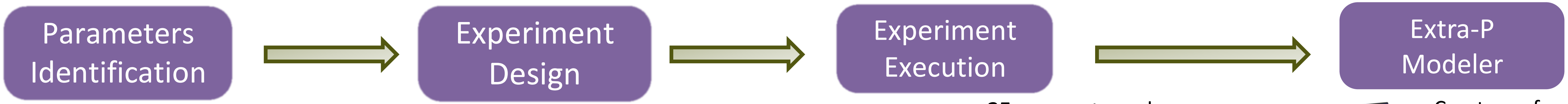
$$2.3\ s^3 + 1.71 \log_2 p - 0.1329$$

#1: Which parameters are relevant?

```
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.

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

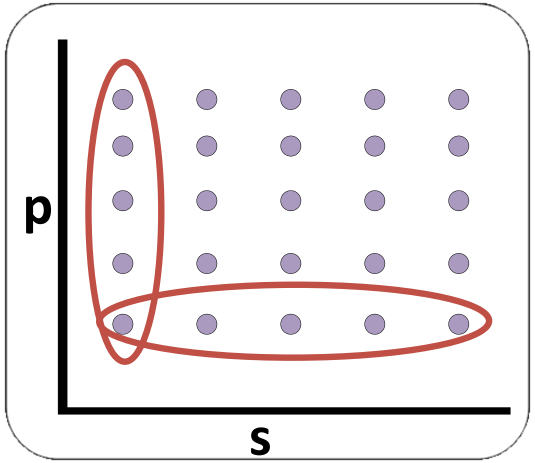
25 parameter values
5 repetitions per sample
125 instrumented executions

Extra-P Modeler

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

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

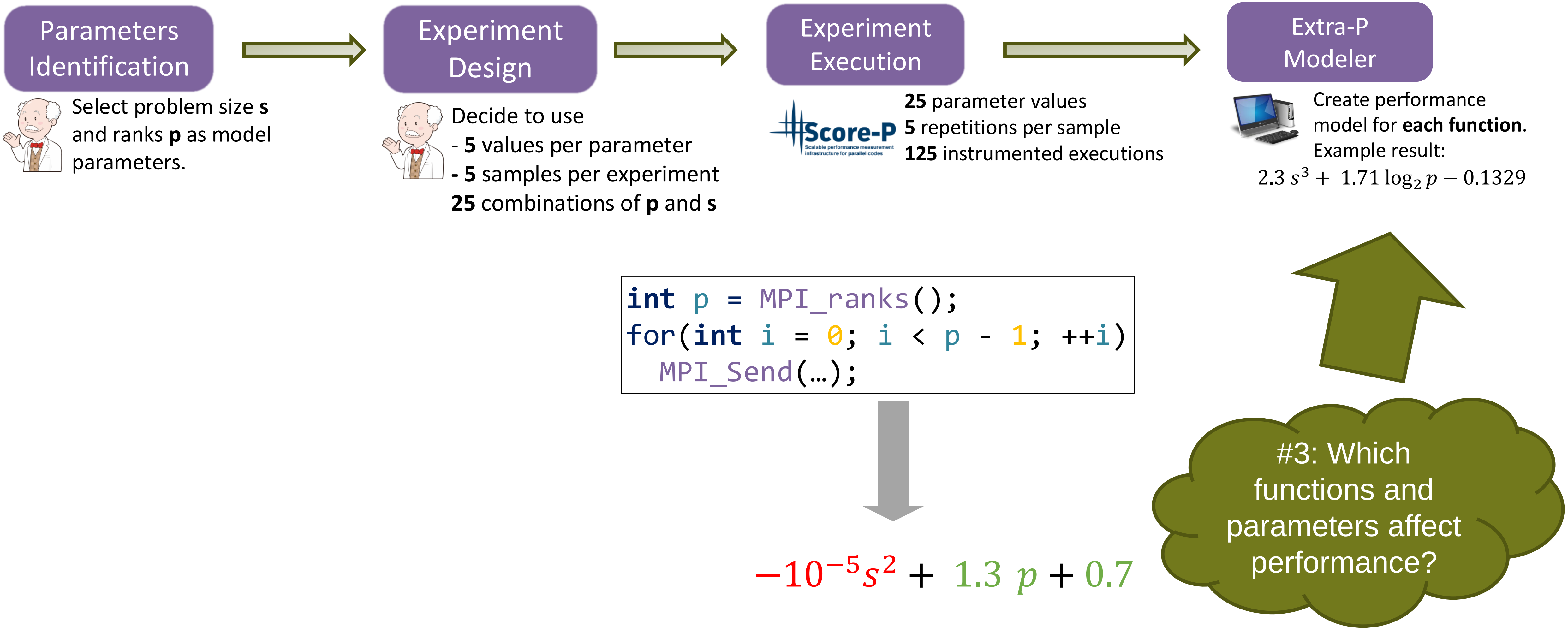
#2: How parameters interact with each other?



$p \times s$
25 experiments

$p + s$
9 experiments

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

Score-P
Scalable performance measurement infrastructure for parallel codes

Extra-P Modeler

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

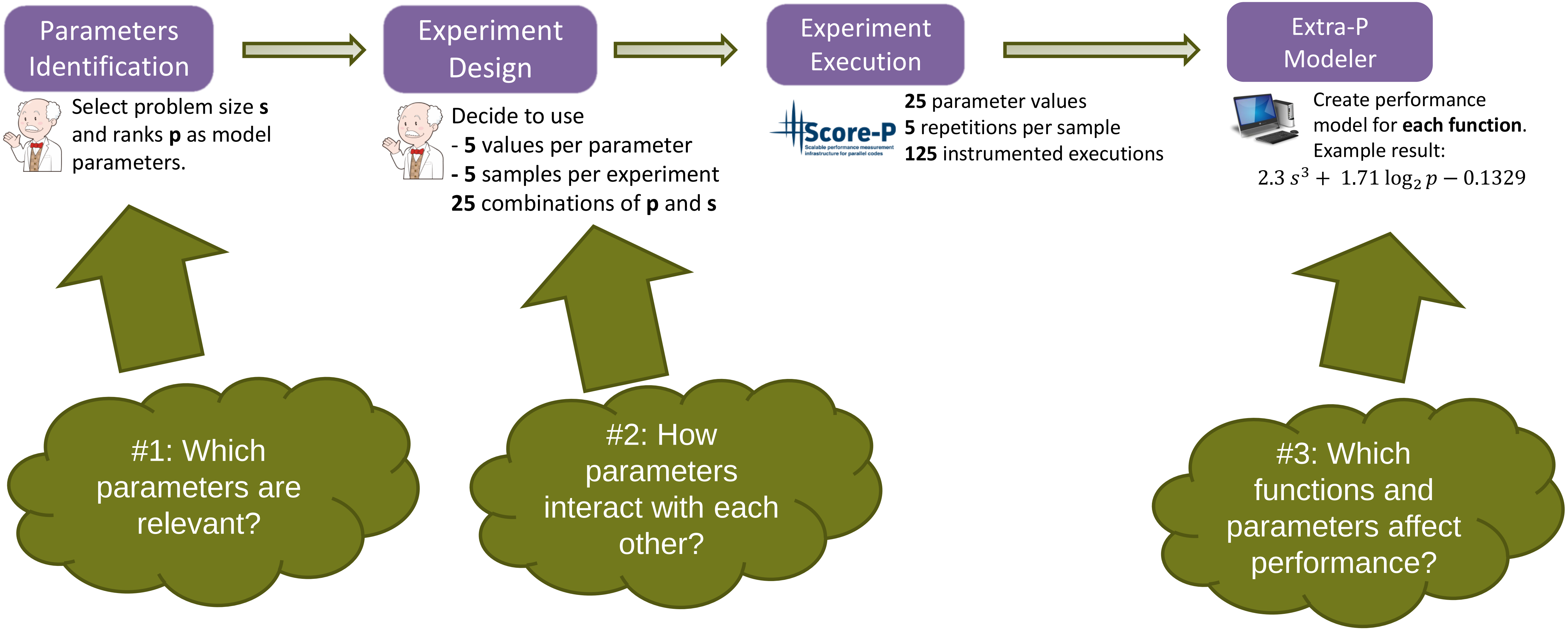
$$2.3\, s^3 + 1.71 \log_2 p - 0.1329$$

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

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

#3: Which functions and parameters affect performance?

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:

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

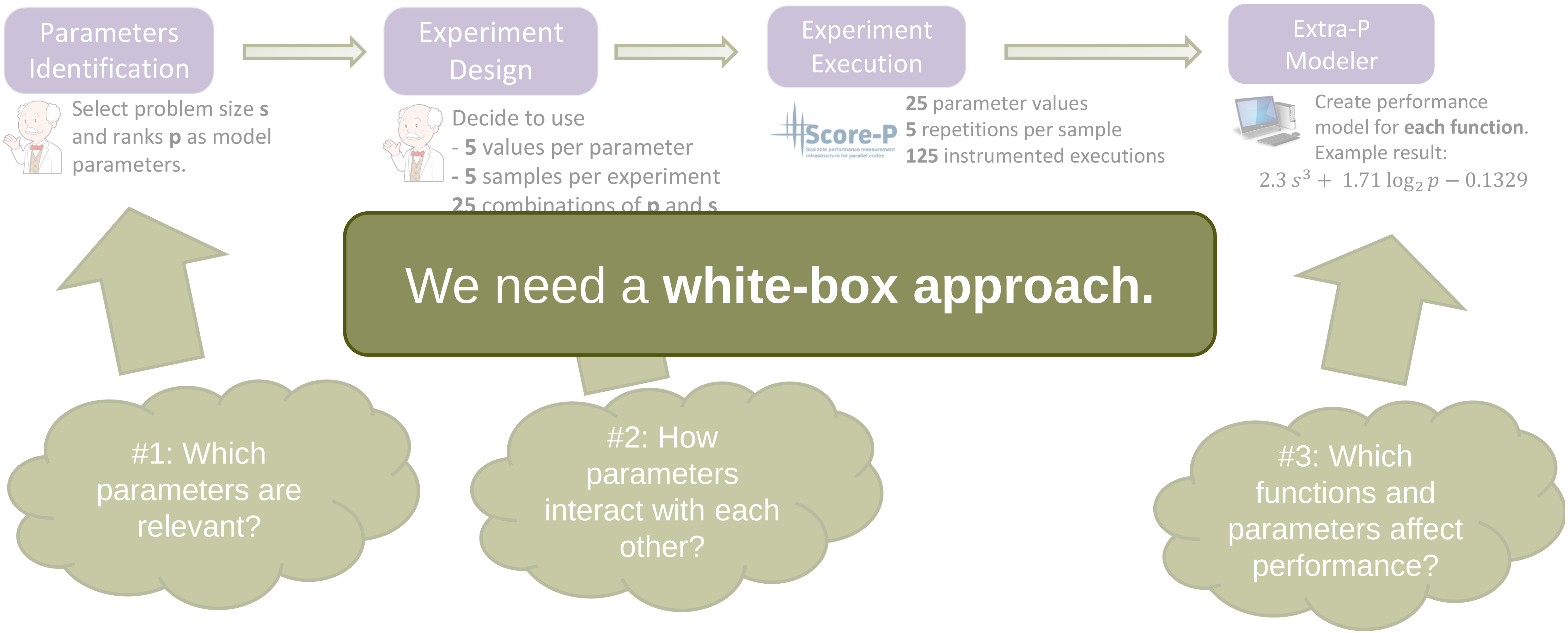
#1: Which parameters are relevant?

#2: How parameters interact with each other?

#3: Which functions and parameters affect performance?

Score-P
Scalable performance measurement infrastructure for parallel codes

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
Scalable performance measurement infrastructure for parallel codes

25 parameter values
5 repetitions per sample
125 instrumented executions

Extra-P Modeler

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

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

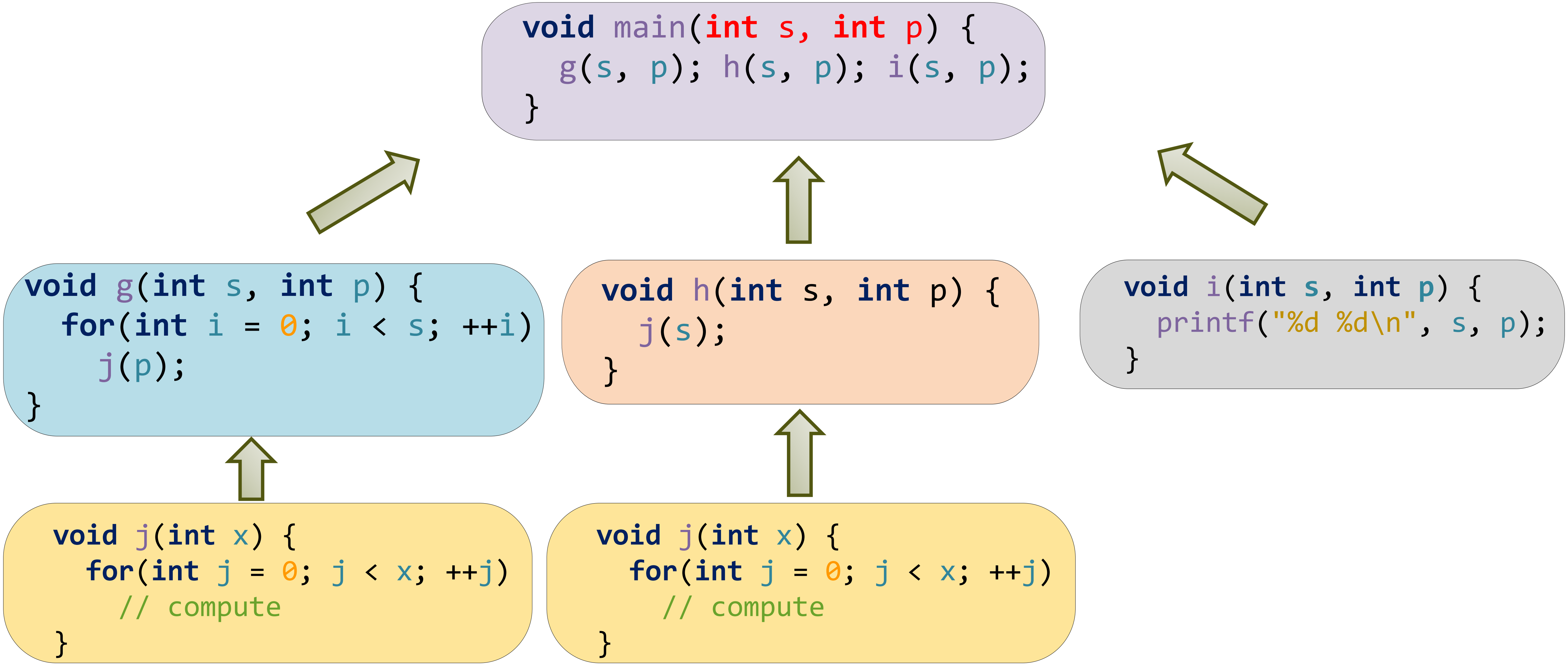
We need a **white-box approach.**

#1: Which parameters are relevant?

#2: How parameters interact with each other?

#3: Which functions and parameters affect performance?

What is important in our program?



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

```
void g(int s, int p) {
  for(int i = 0; i < s; ++i)
    j(p);
}
```

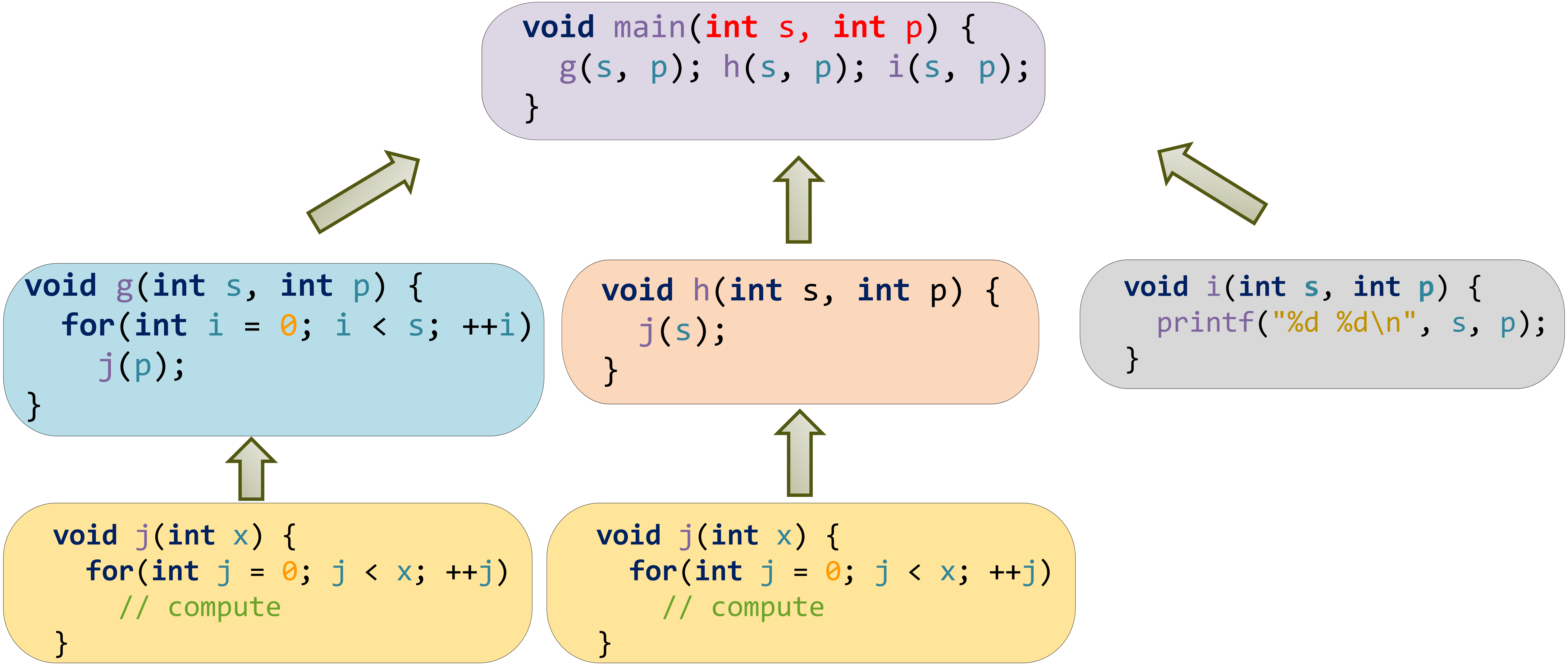
```
void h(int s, int p) {
  j(s);
}
```

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

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

```
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);
}
```

```
void g(int s, int p) {
  for(int i = 0; i < s; ++i)
    j(p);
}
```

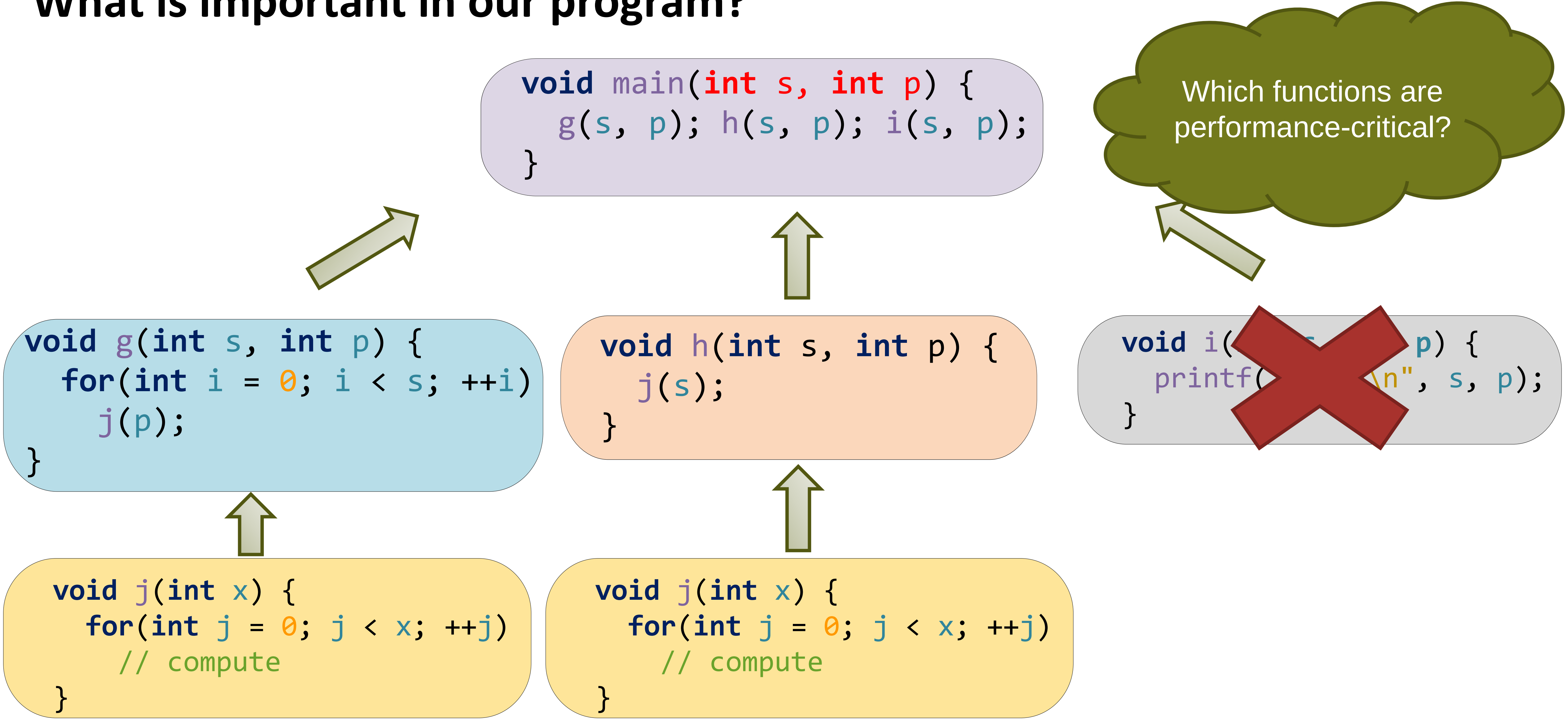
```
void h(int s, int p) {
  j(s);
}
```

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

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

```
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);
}
```

```
void g(int s, int p) {
  for(int i = 0; i < s; ++i)
    j(p);
}
```

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

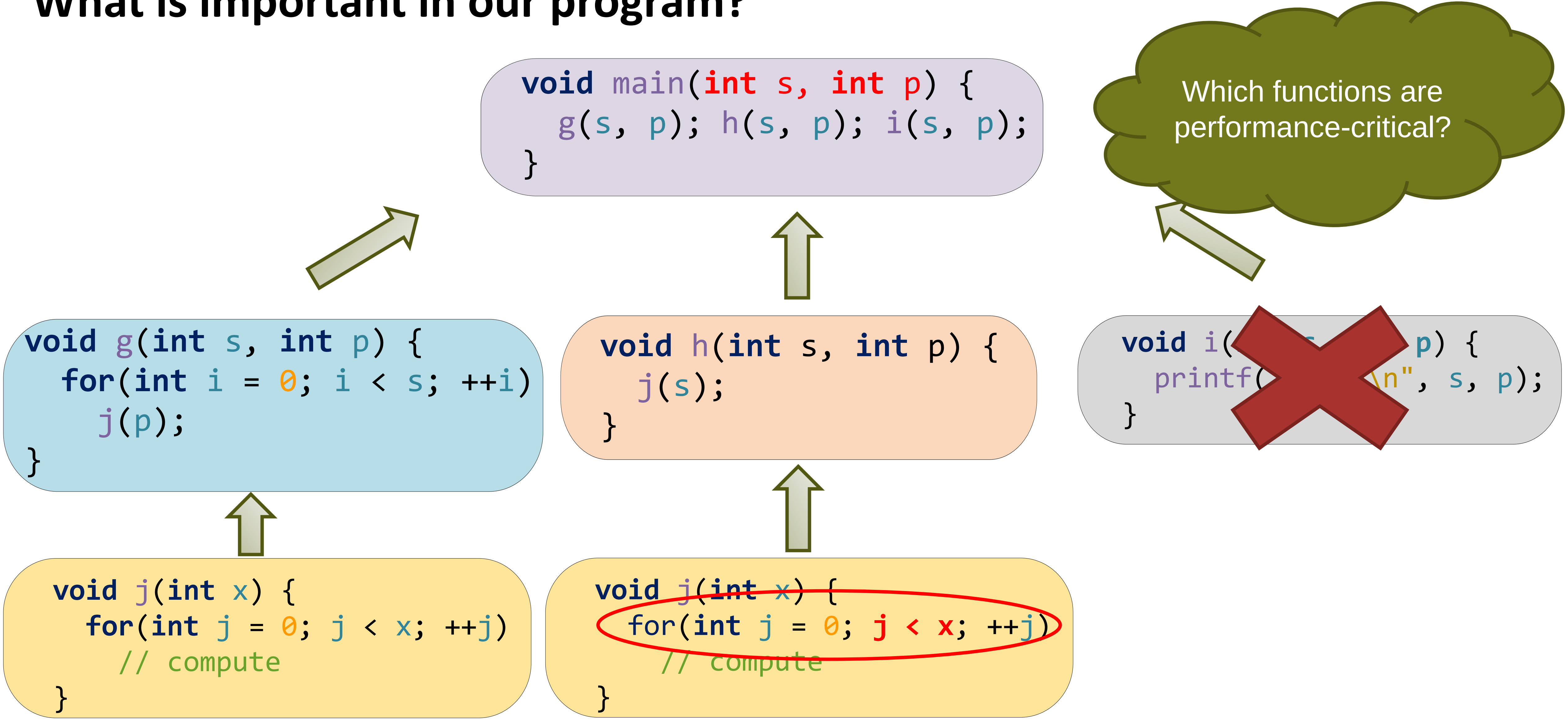
```
void i(          p) {
  printf(       \n", s, p);
}
```

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

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

Which functions are performance-critical?

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, int p) {
  for(int i = 0; i < s; ++i)
    j(p);
}
```

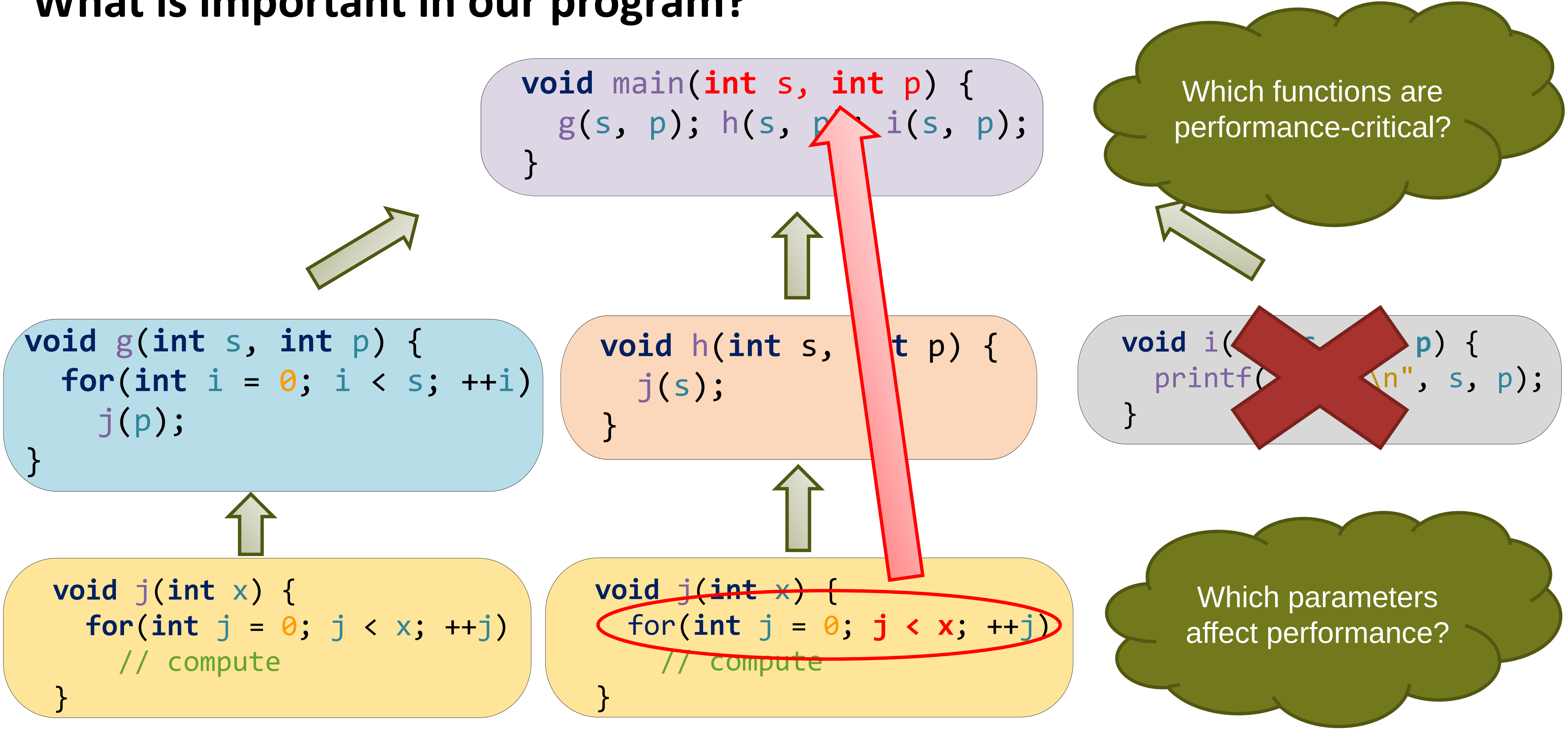
```
void h(int s, int p) {
  j(s);
}
```

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

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

```
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);
}
```

```
void g(int s, int p) {
  for(int i = 0; i < s; ++i)
    j(p);
}
```

```
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
}
```

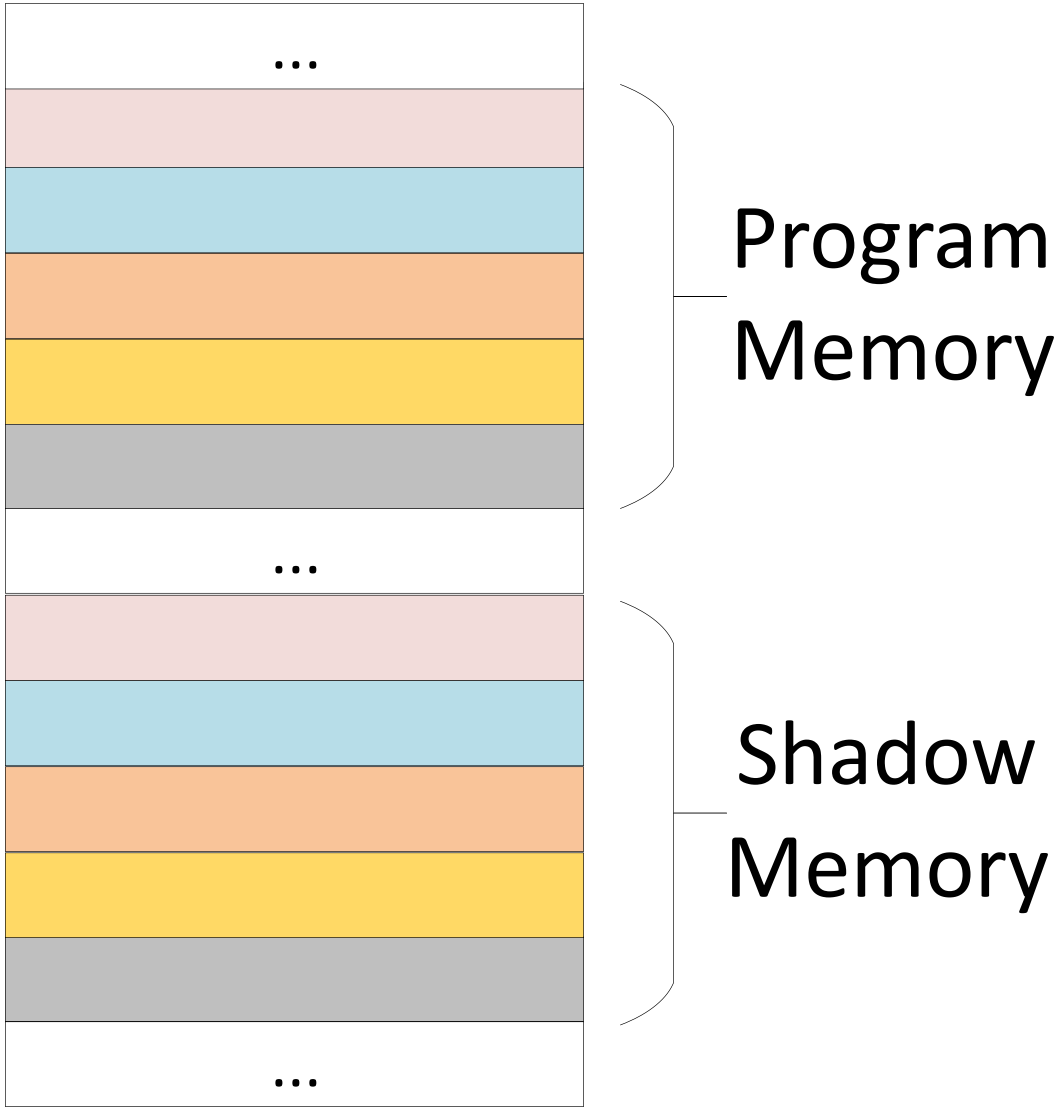
```
void j(int x) {
  for(int j = 0; j < x; ++j)
    // compute
}
```

Which functions are performance-critical?

Which parameters affect performance?

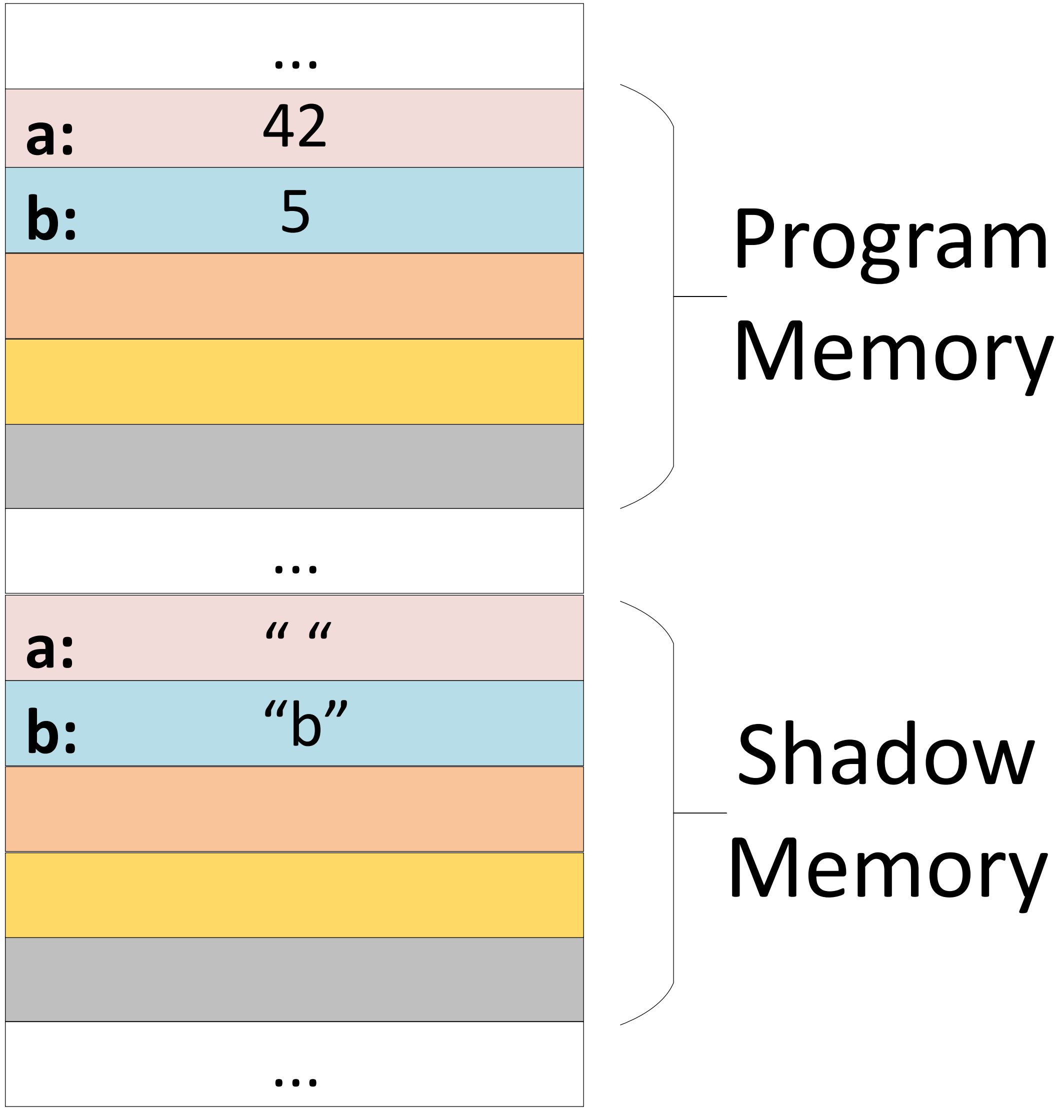
Taint Analysis: track parameters propagation

```
int a = 42;
int b = omp_get_num_threads();
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;
```



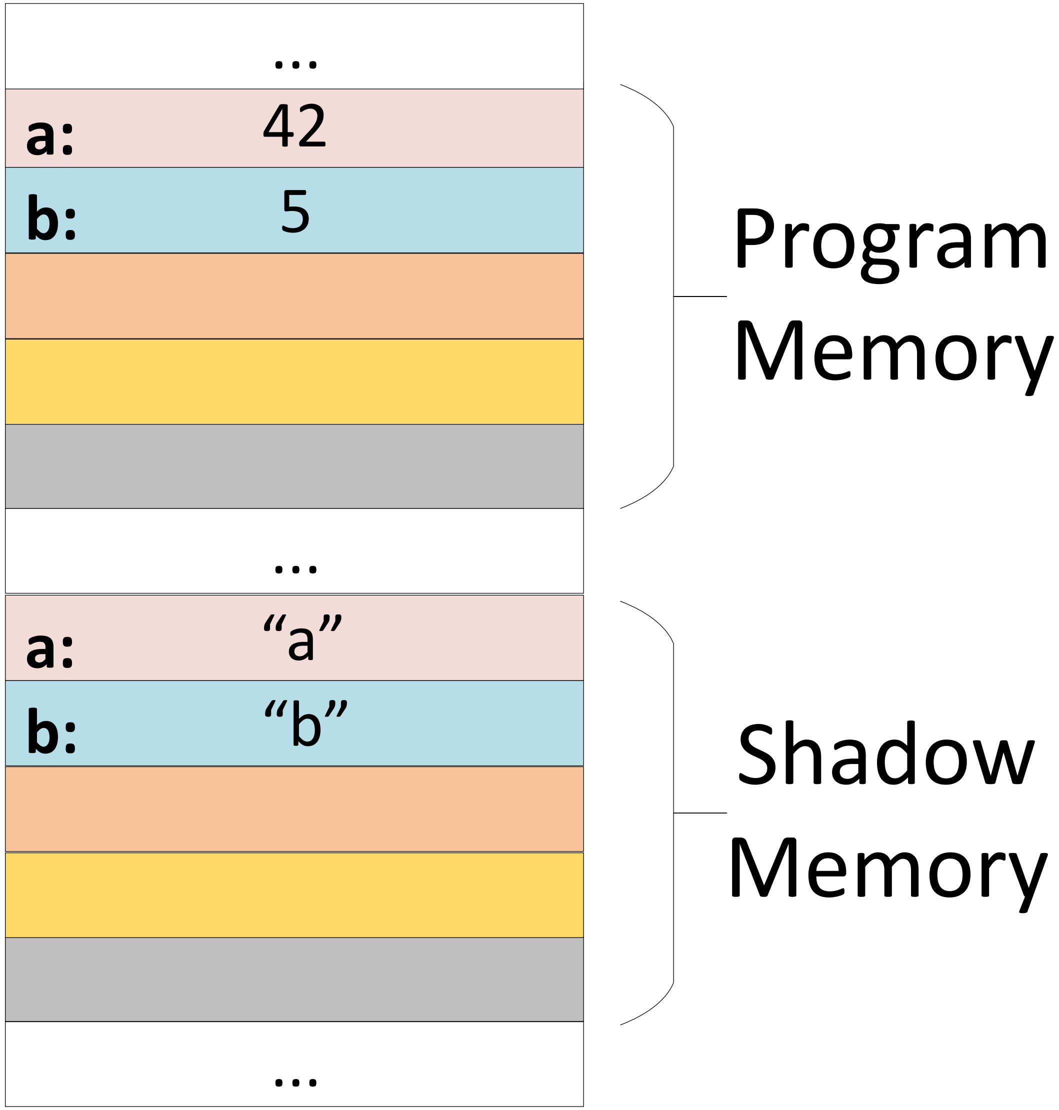
Taint Analysis: track parameters propagation

```
int a = 42;
int b = omp_get_num_threads();
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;
```



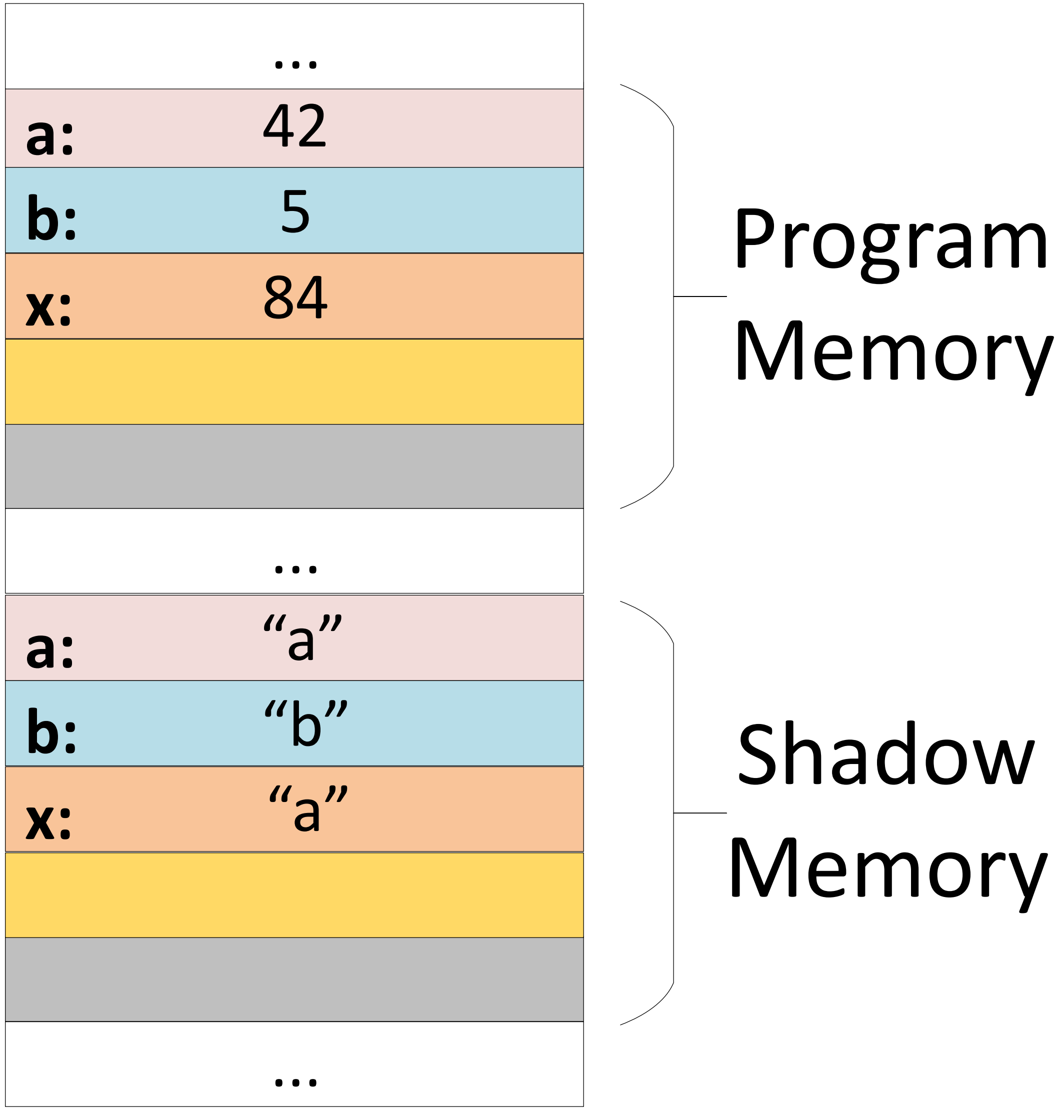
Taint Analysis: track parameters propagation

```
int a = 42;
int b = omp_get_num_threads();
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

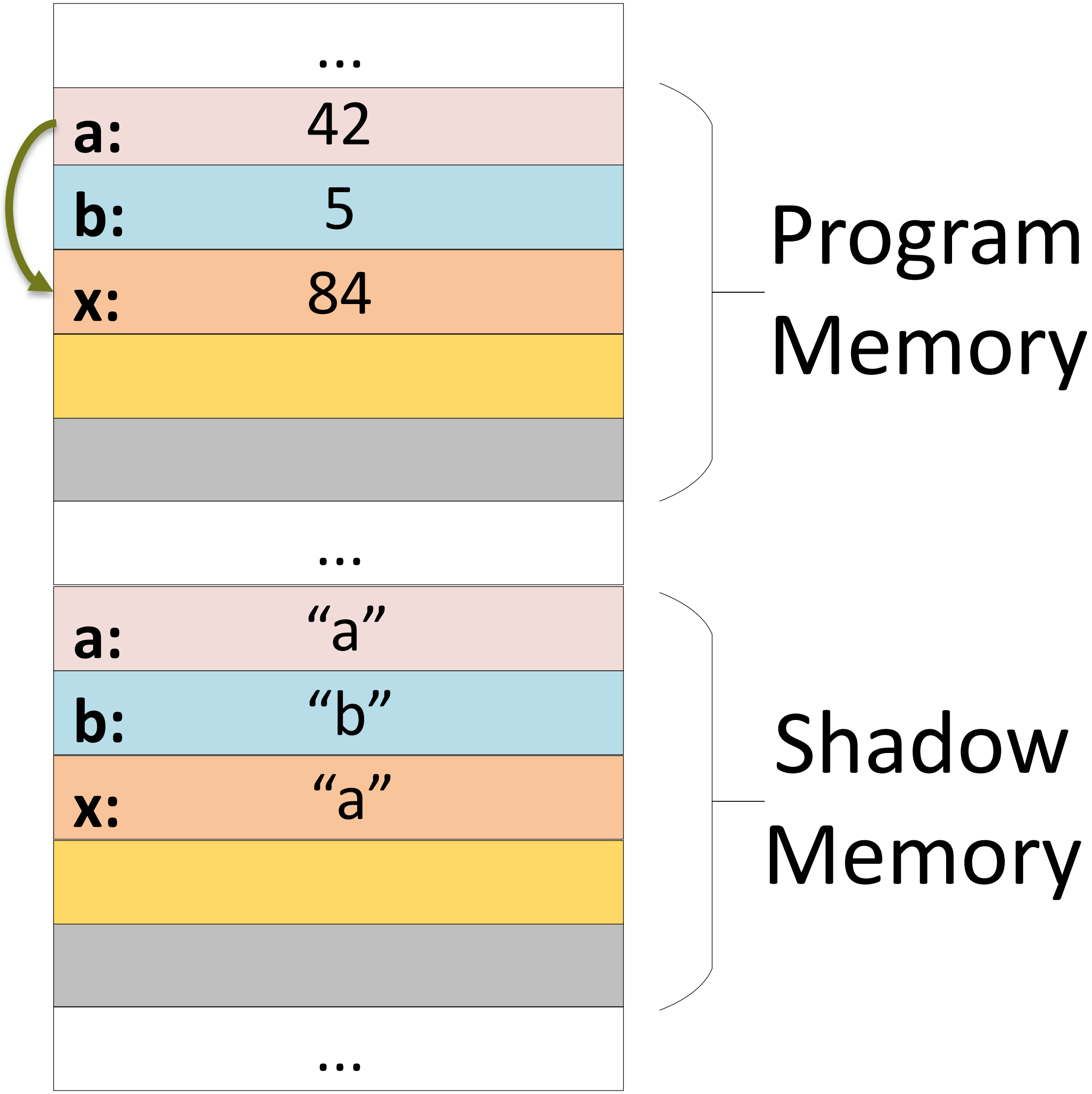
```
int a = 42;
int b = omp_get_num_threads();
taint_variable(a);

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

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



…	
a:	42
b:	5
x:	84
…	
a:	“a”
b:	“b”
x:	“a”
…	

Taint Analysis: track parameters propagation

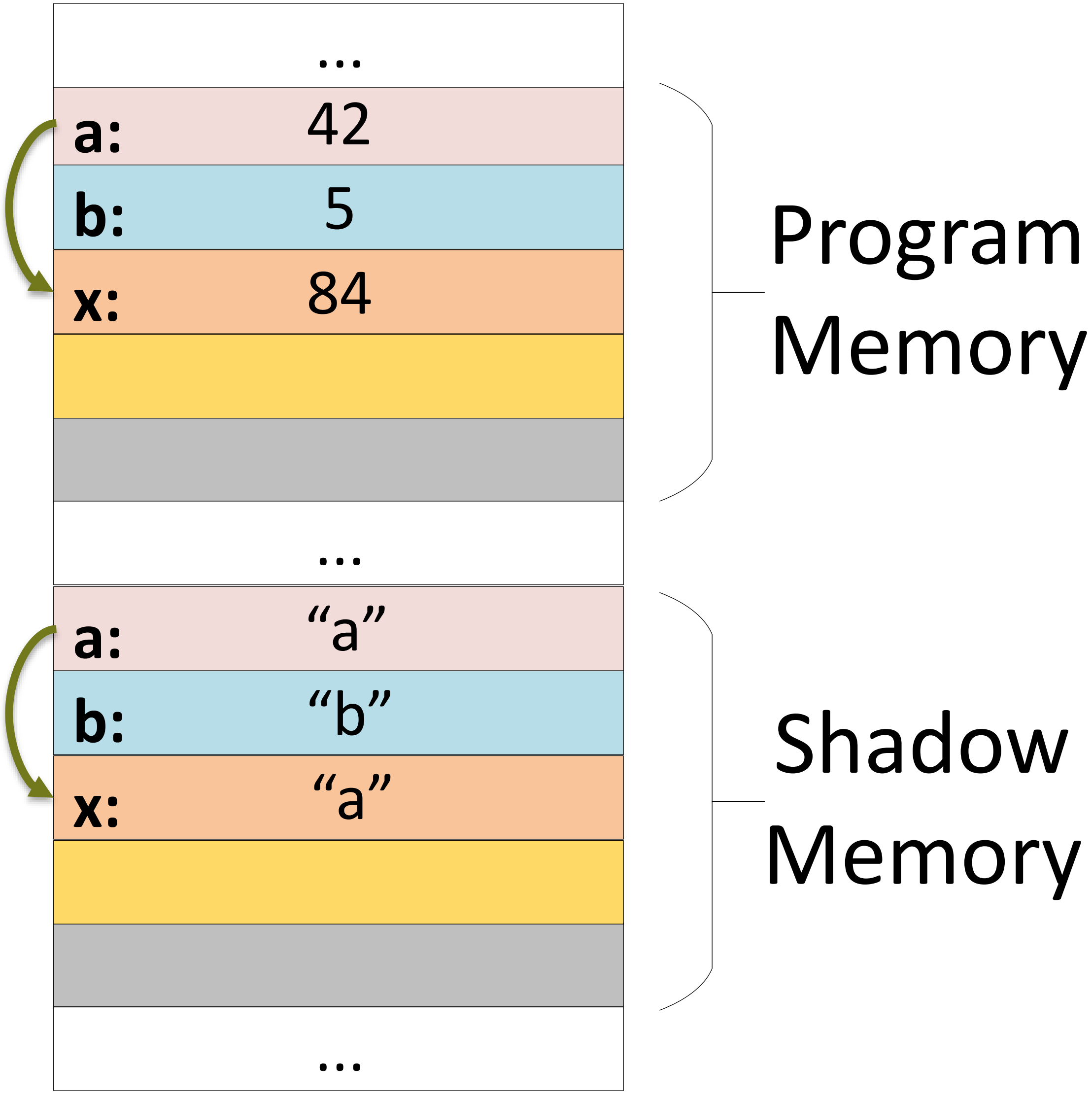
```
int a = 42;
int b = omp_get_num_threads();
taint_variable(a);

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

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



…
a: 42
b: 5
x: 84
…
a: "a"
b: "b"
x: "a"
…

Taint Analysis: track parameters propagation

```
int a = 42;
int b = omp_get_num_threads();
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;
```



…

a: 42

b: 5

x: 84

…

a: "a"

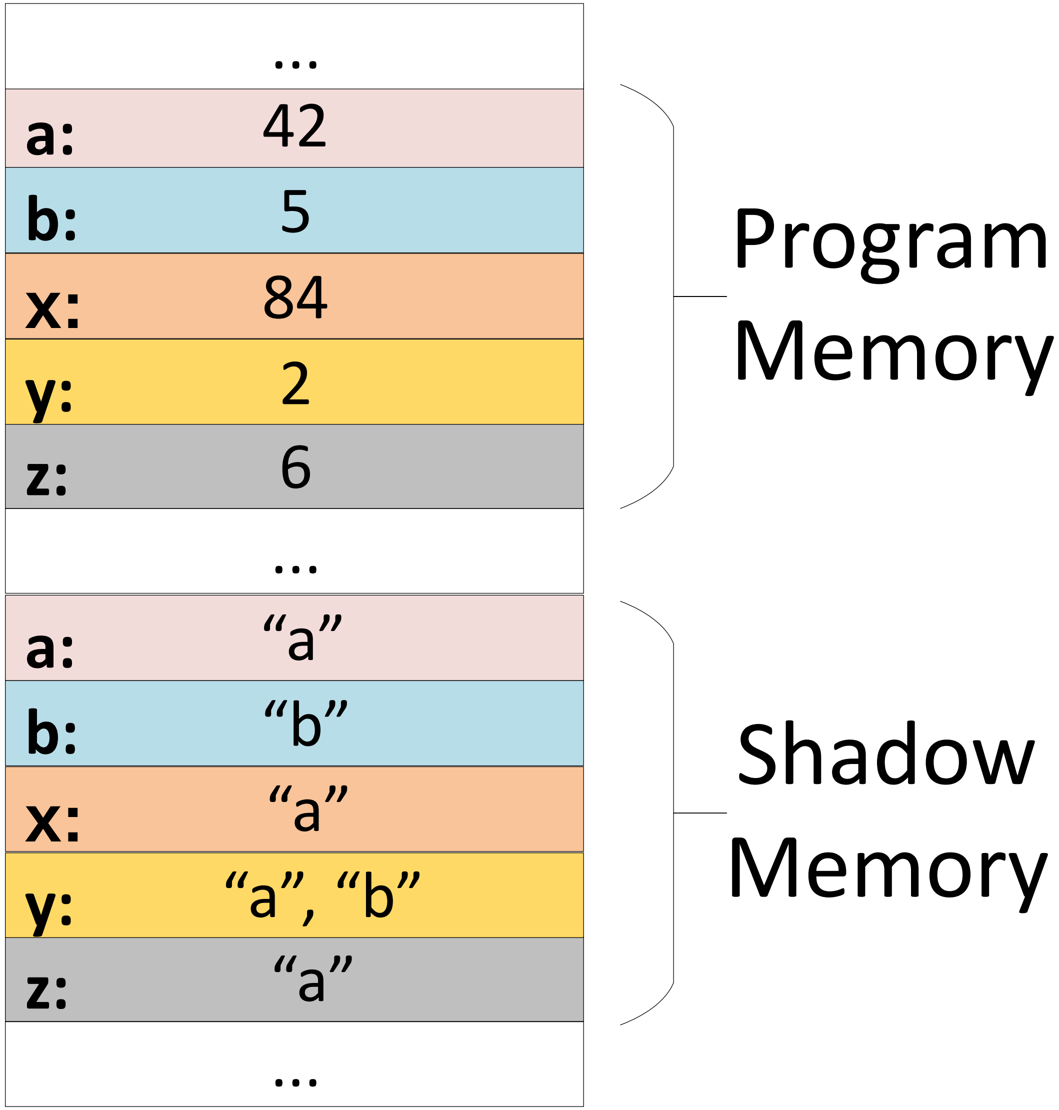
b: "b"

x: "a"

…

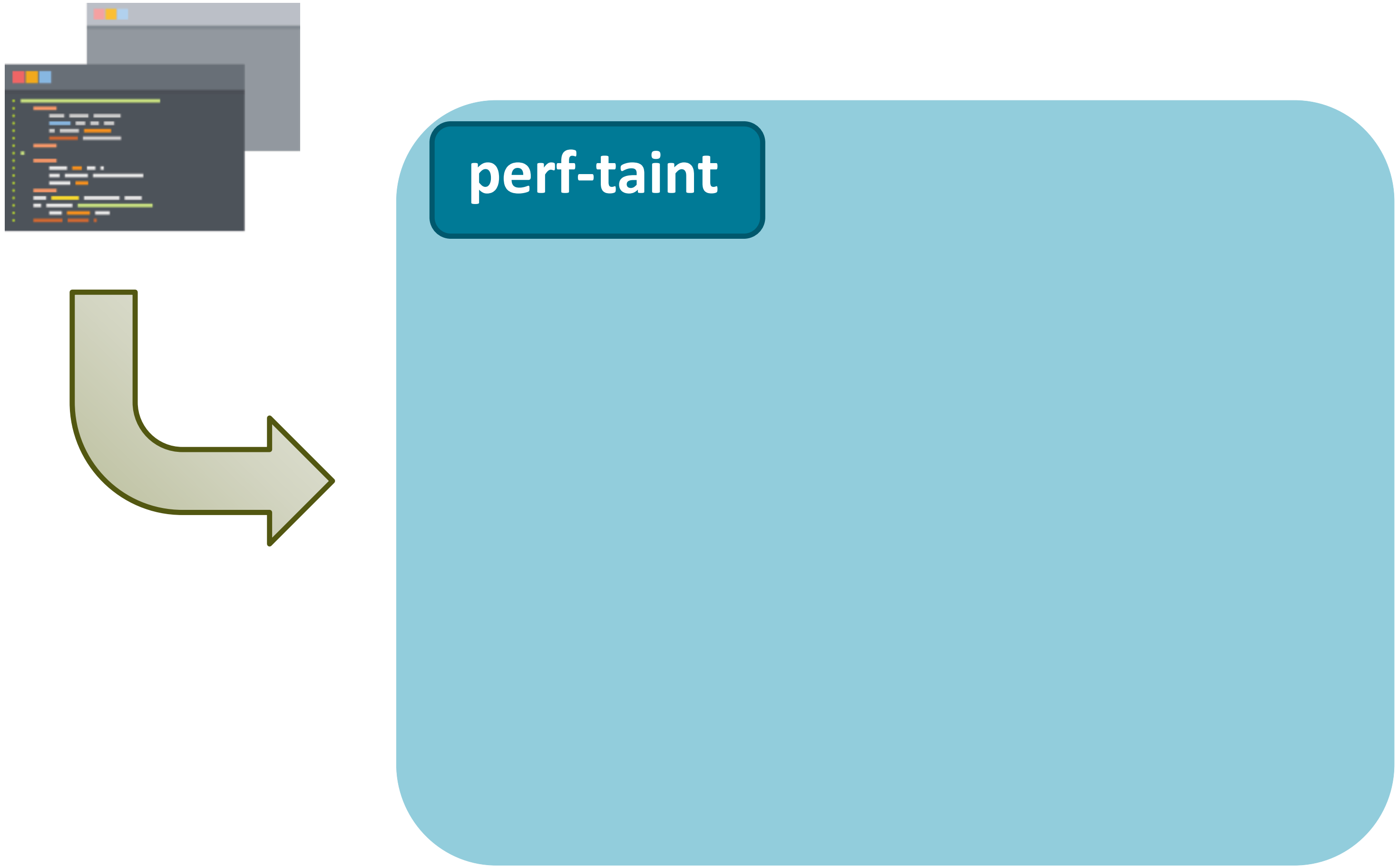
Taint Analysis: track parameters propagation

```
int a = 42;
int b = omp_get_num_threads();
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;
```

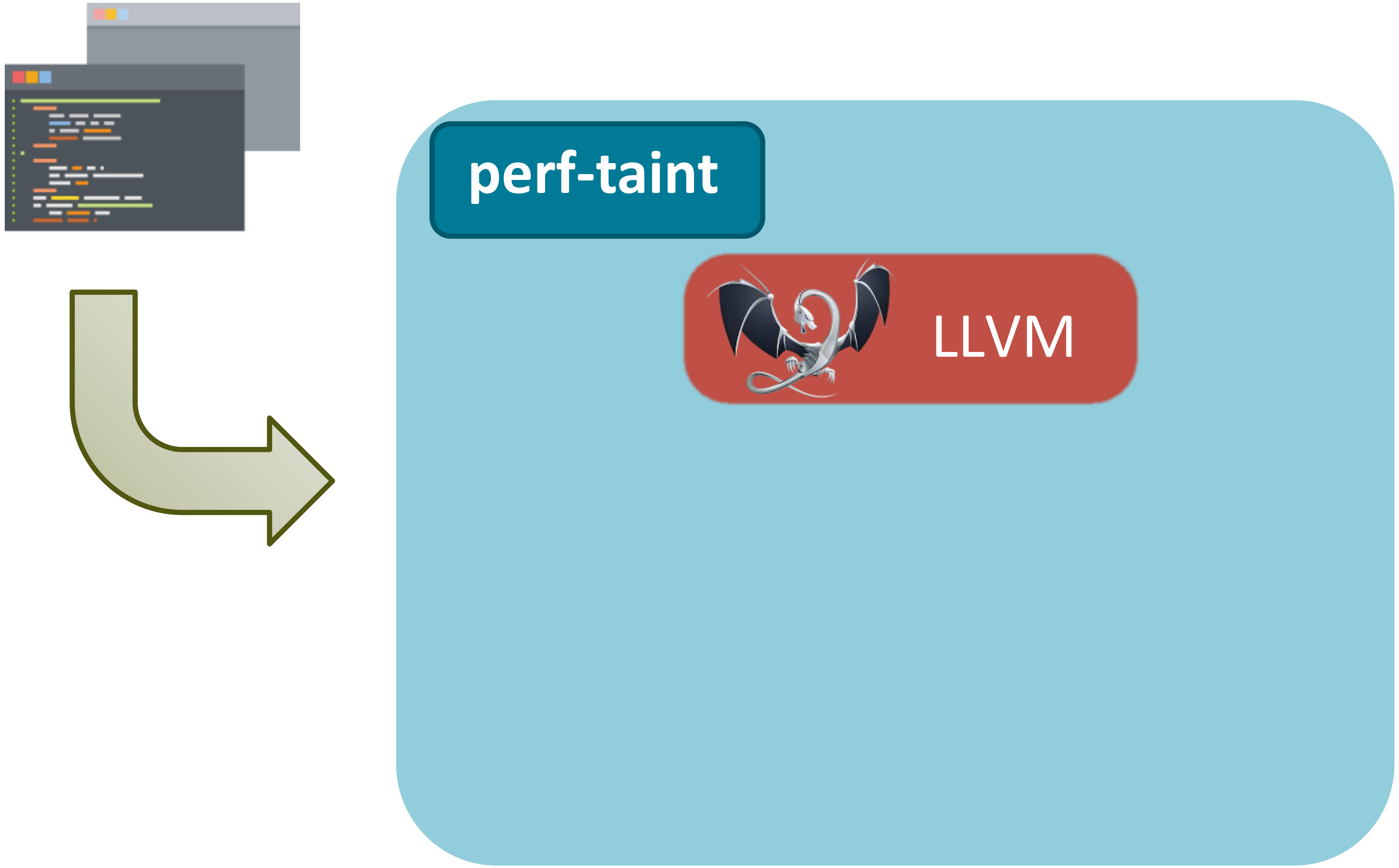


	…
a:	42
b:	5
x:	84
y:	2
z:	6
	…
a:	“a”
b:	“b”
x:	“a”
y:	“a”, “b”
z:	“a”
	…

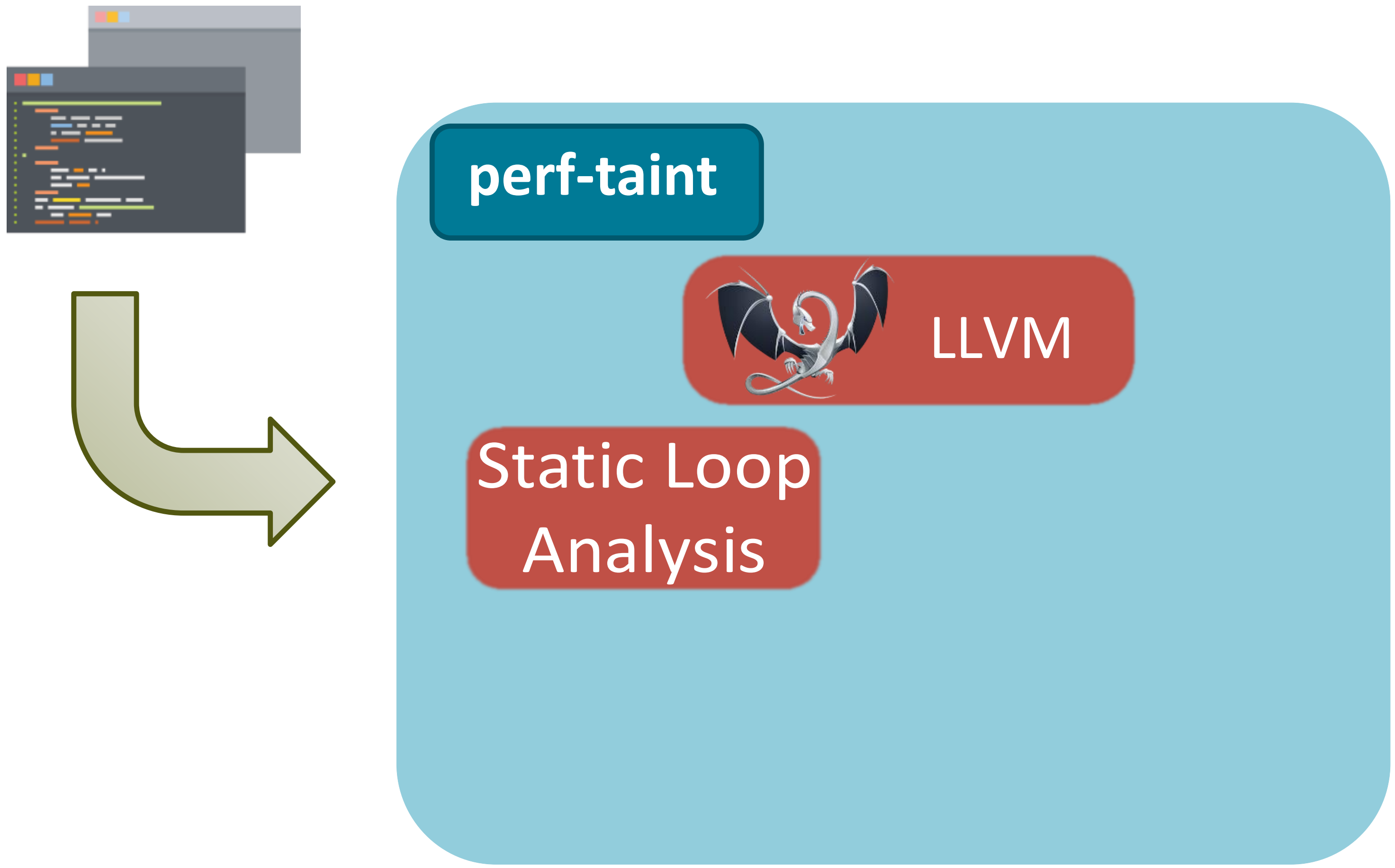
Hybrid Taint Analysis



Hybrid Taint Analysis



Hybrid Taint Analysis

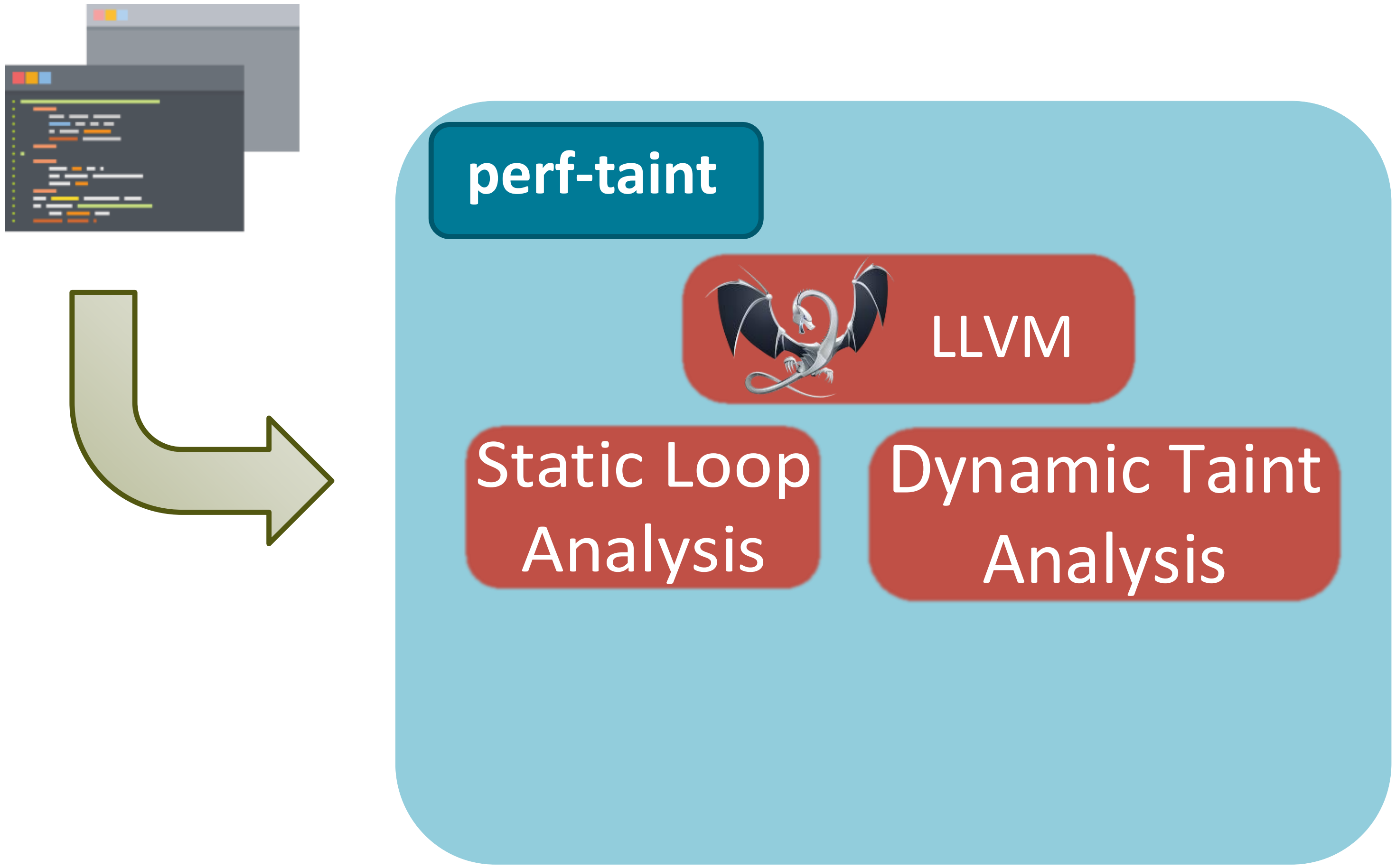


perf-taint

LLVM

Static Loop Analysis

Hybrid Taint Analysis



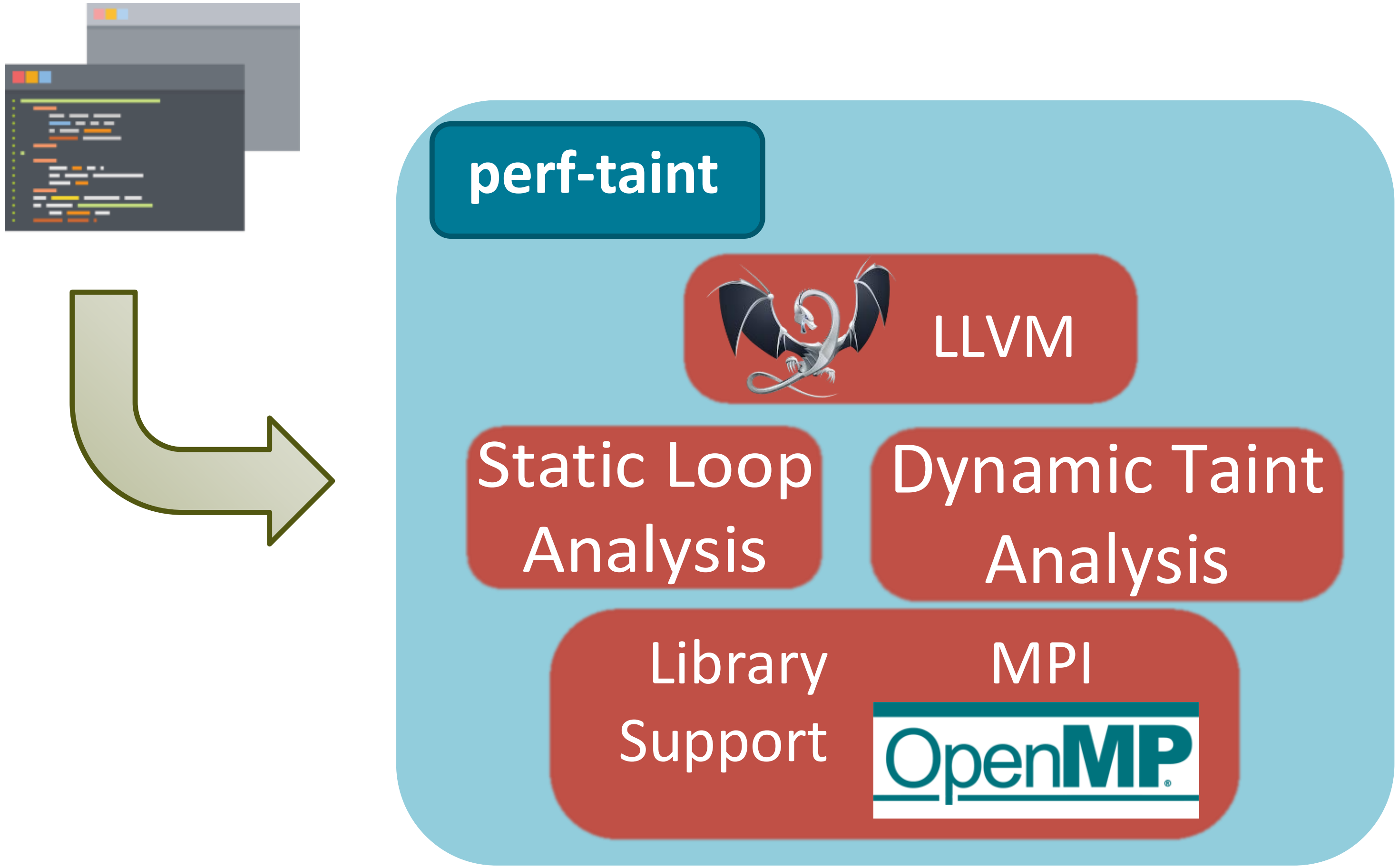
perf-taint

LLVM

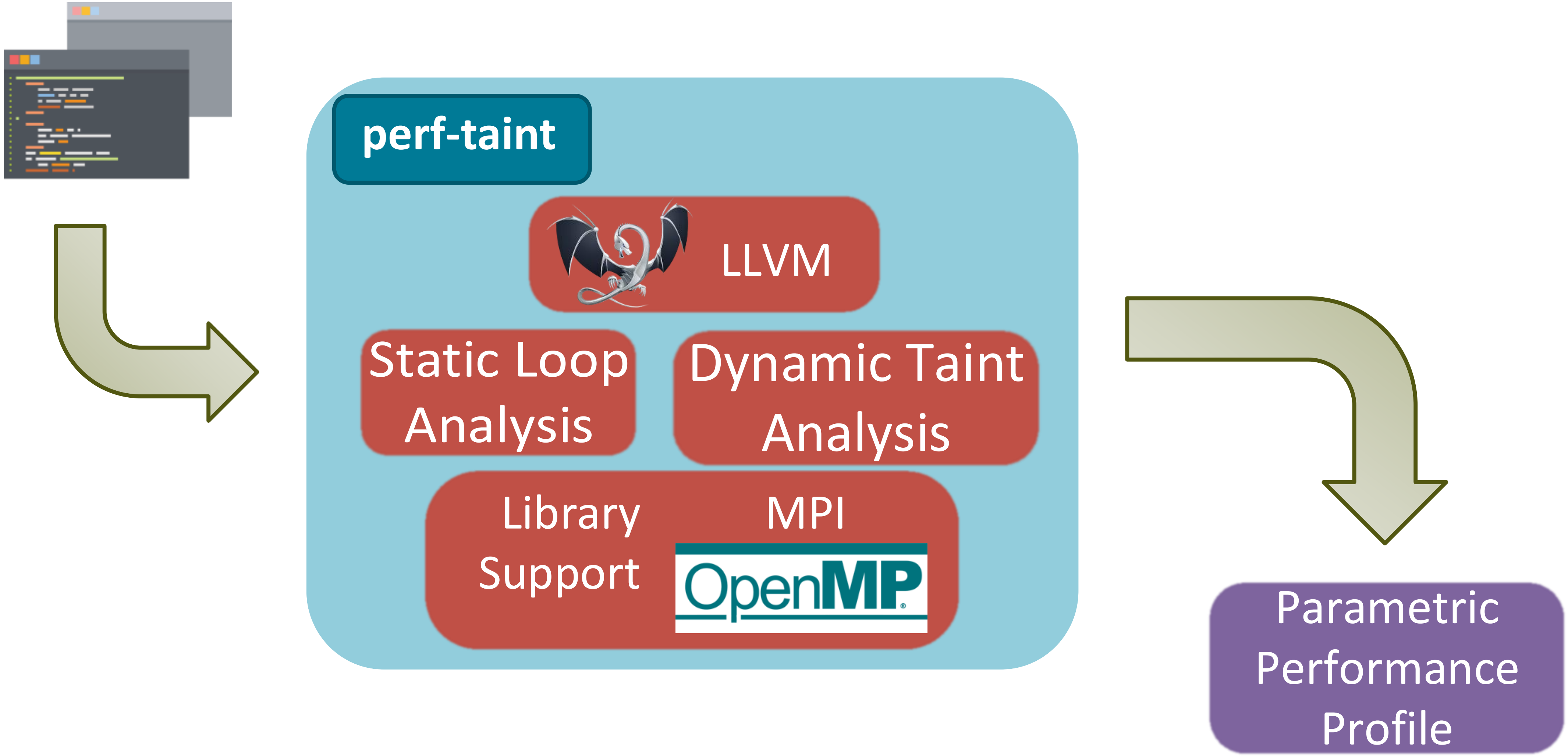
Static Loop Analysis

Dynamic Taint Analysis

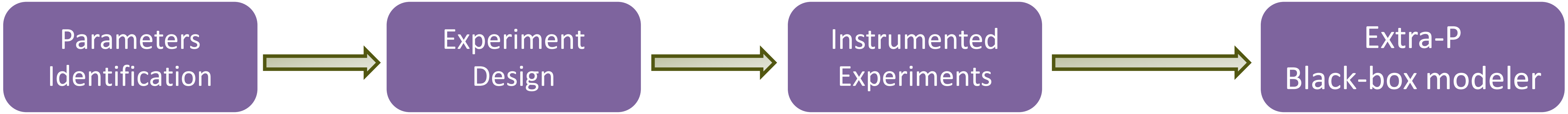
Hybrid Taint Analysis



Hybrid Taint Analysis



How do we apply this knowledge?



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

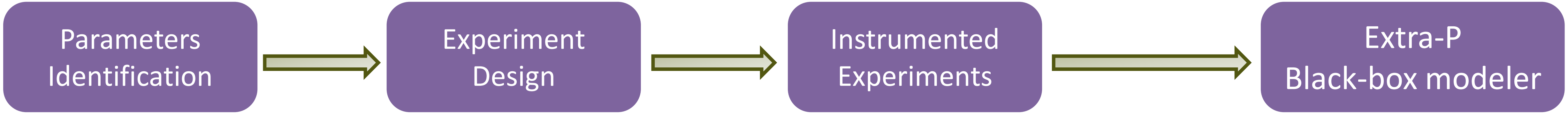
Decide to use
- **5** values per parameter
25 combinations of **p** and **s**

Score-P
Scalable performance measurement infrastructure for parallel codes

25 parameter values
5 sample repetitions
125 instrumented runs

Parametric model for **each function**.

How do we apply this knowledge?



Parameters Identification → Experiment Design → Instrumented Experiments → Extra-P Black-box modeler

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

Decide to use
- **5** values per parameter
25 combinations of **p** and **s**

25 parameter values
5 sample repetitions
125 instrumented runs

Parametric model for **each function.**

“**p** and **s** have the highest impact.”

How do we apply this knowledge?

Parameters Identification → **Experiment Design** → **Instrumented Experiments** → **Extra-P Black-box modeler**

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

Decide to use
- **5** values per parameter
25 combinations of **p** and **s**

Score-P
Scalable performance measurement infrastructure for parallel codes

25 parameter values
5 sample repetitions
125 instrumented runs

Parametric model for **each function.**

“**p** and **s** have the highest impact.”

“**p** and s are not multiplicatively dependent”

How do we apply this knowledge?

Parameters Identification → Experiment Design → Instrumented Experiments → Extra-P Black-box modeler

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

“**p** and **s** have the highest impact.”

Decide to use

- **5** values per parameter

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

“**p** and s are not multiplicatively dependent”

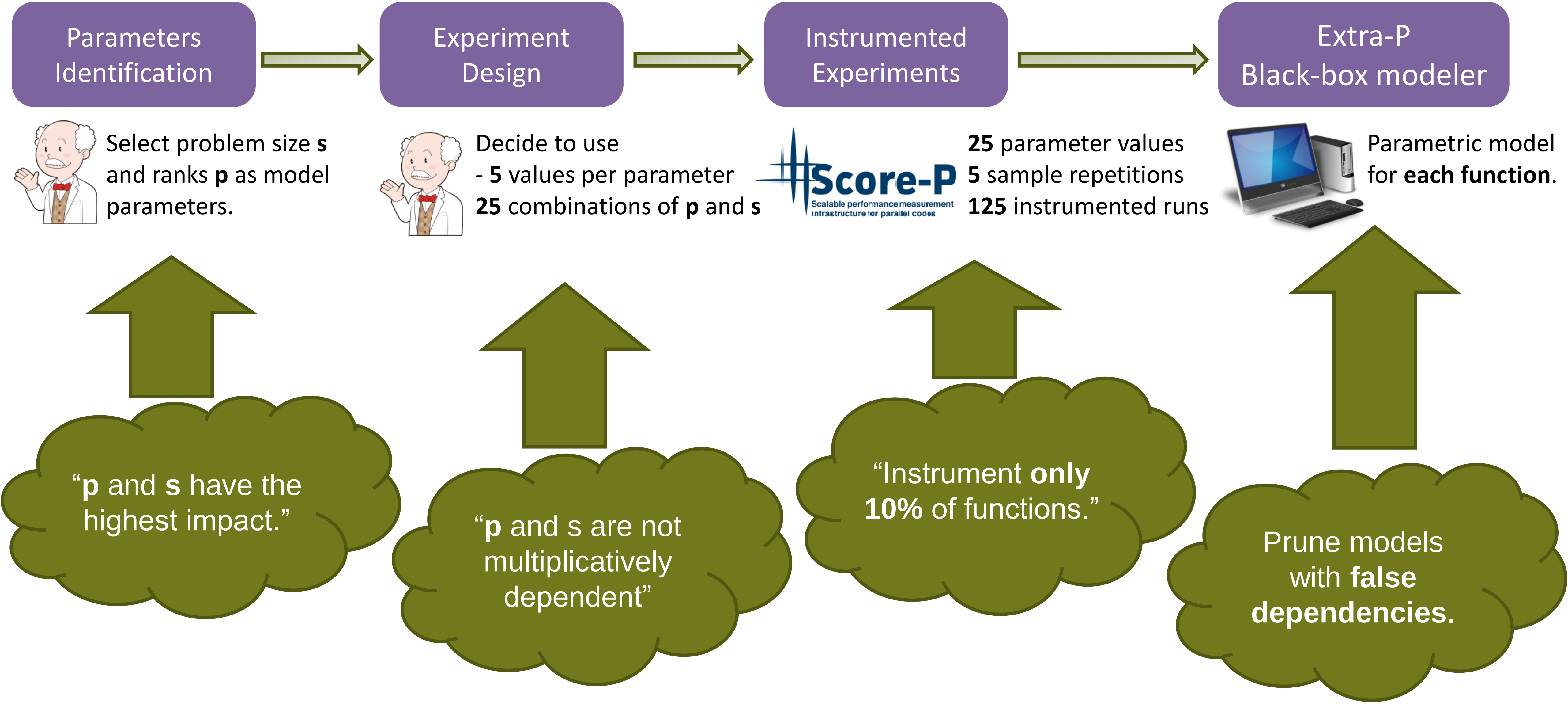
Score-P
Scalable performance measurement infrastructure for parallel codes

25 parameter values
5 sample repetitions
125 instrumented runs

“Instrument **only 10%** of functions.”

Parametric model for **each function**.

How do we apply this knowledge?



Parameters Identification → **Experiment Design** → **Instrumented Experiments** → **Extra-P Black-box modeler**

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

Decide to use
- **5** values per parameter
25 combinations of **p** and **s**

Score-P
Scalable performance measurement infrastructure for parallel codes

25 parameter values
5 sample repetitions
125 instrumented runs

Parametric model for **each function.**

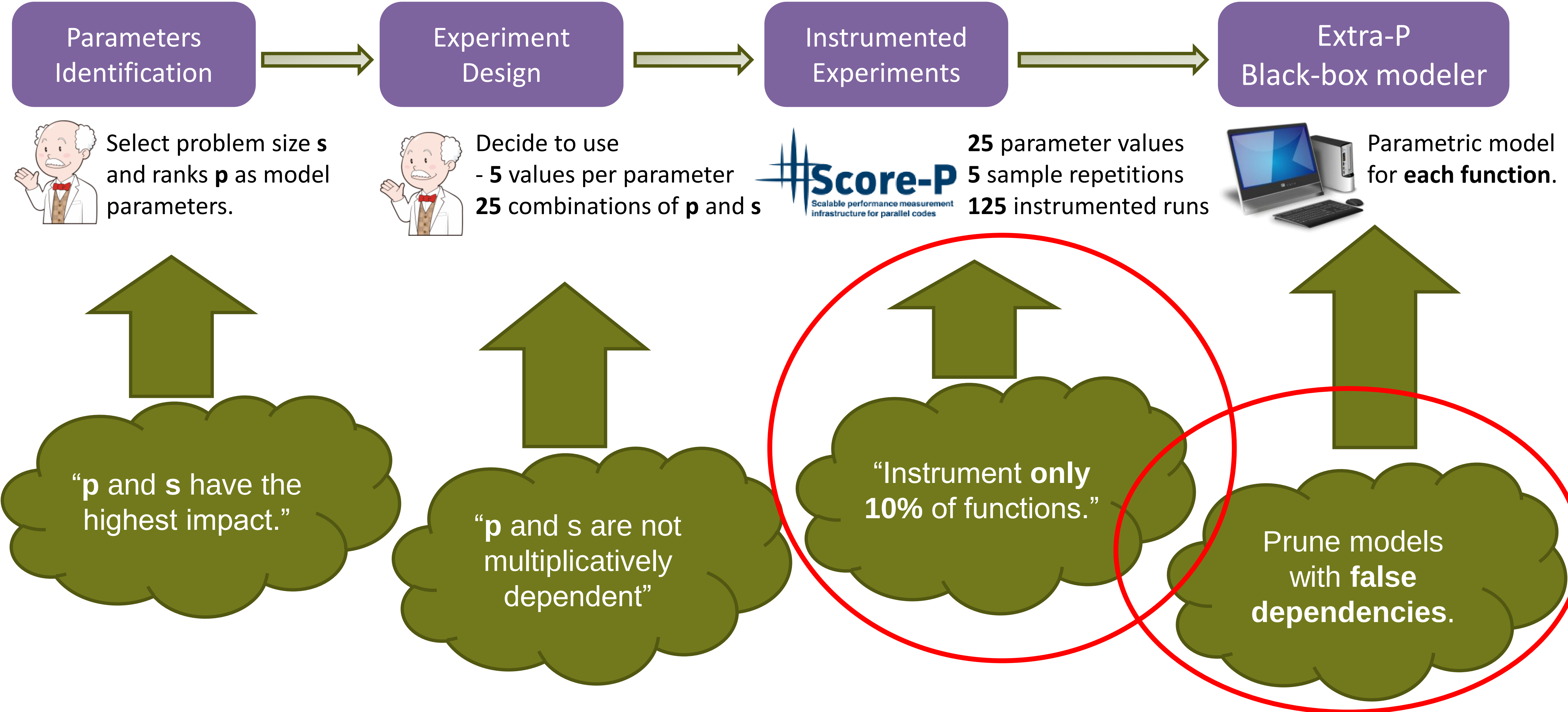
“**p** and **s** have the highest impact.”

“**p** and s are not multiplicatively dependent”

“Instrument **only 10%** of functions.”

Prune models with **false dependencies.**

How do we apply this knowledge?



Parameters Identification

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

Experiment Design

Decide to use
- **5** values per parameter
25 combinations of **p** and **s**

Instrumented Experiments

25 parameter values
5 sample repetitions
125 instrumented runs

Extra-P Black-box modeler

Parametric model for **each function**.

“**p** and **s** have the highest impact.”

“**p** and s are not multiplicatively dependent”

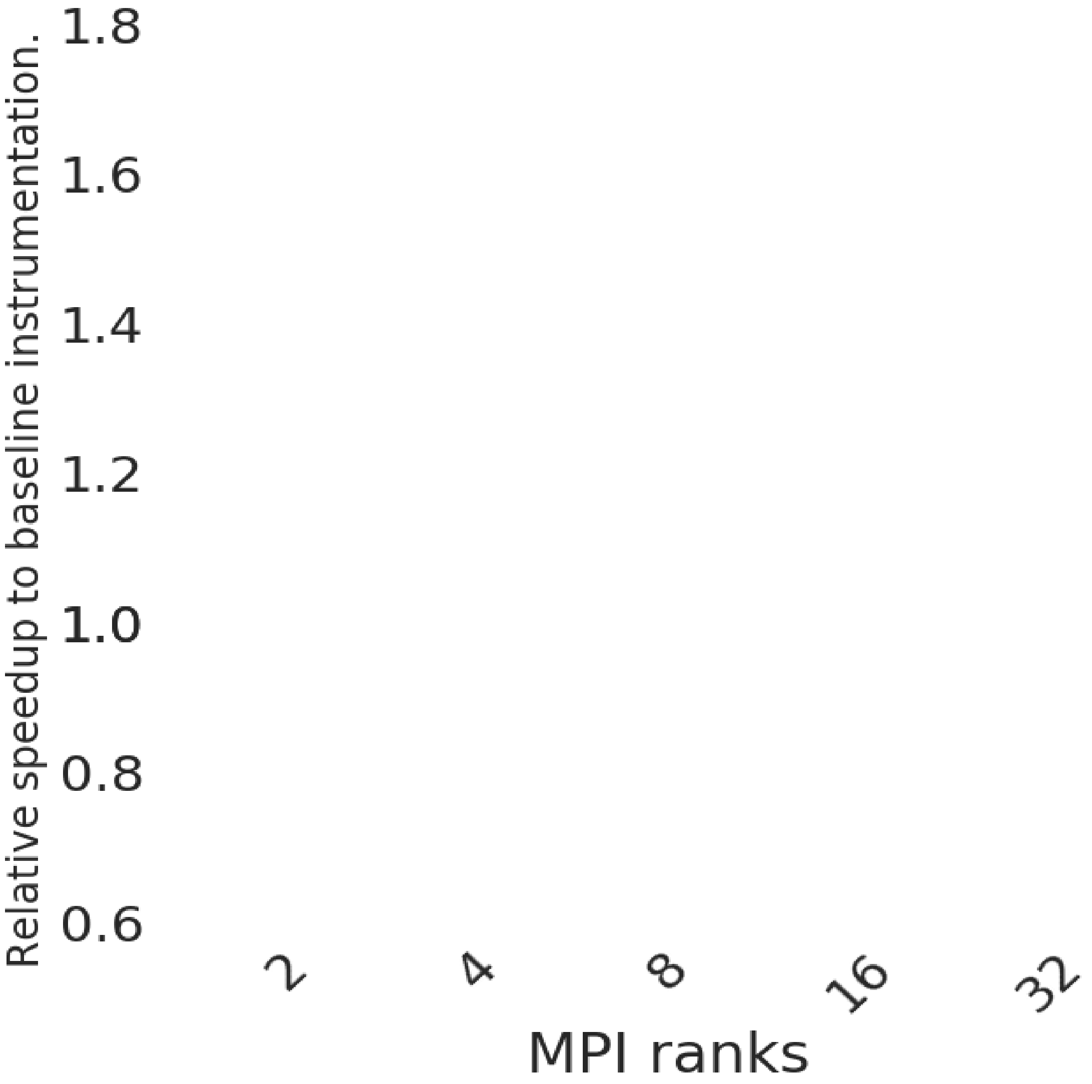
“Instrument **only 10%** of functions.”

Prune models with **false dependencies**.

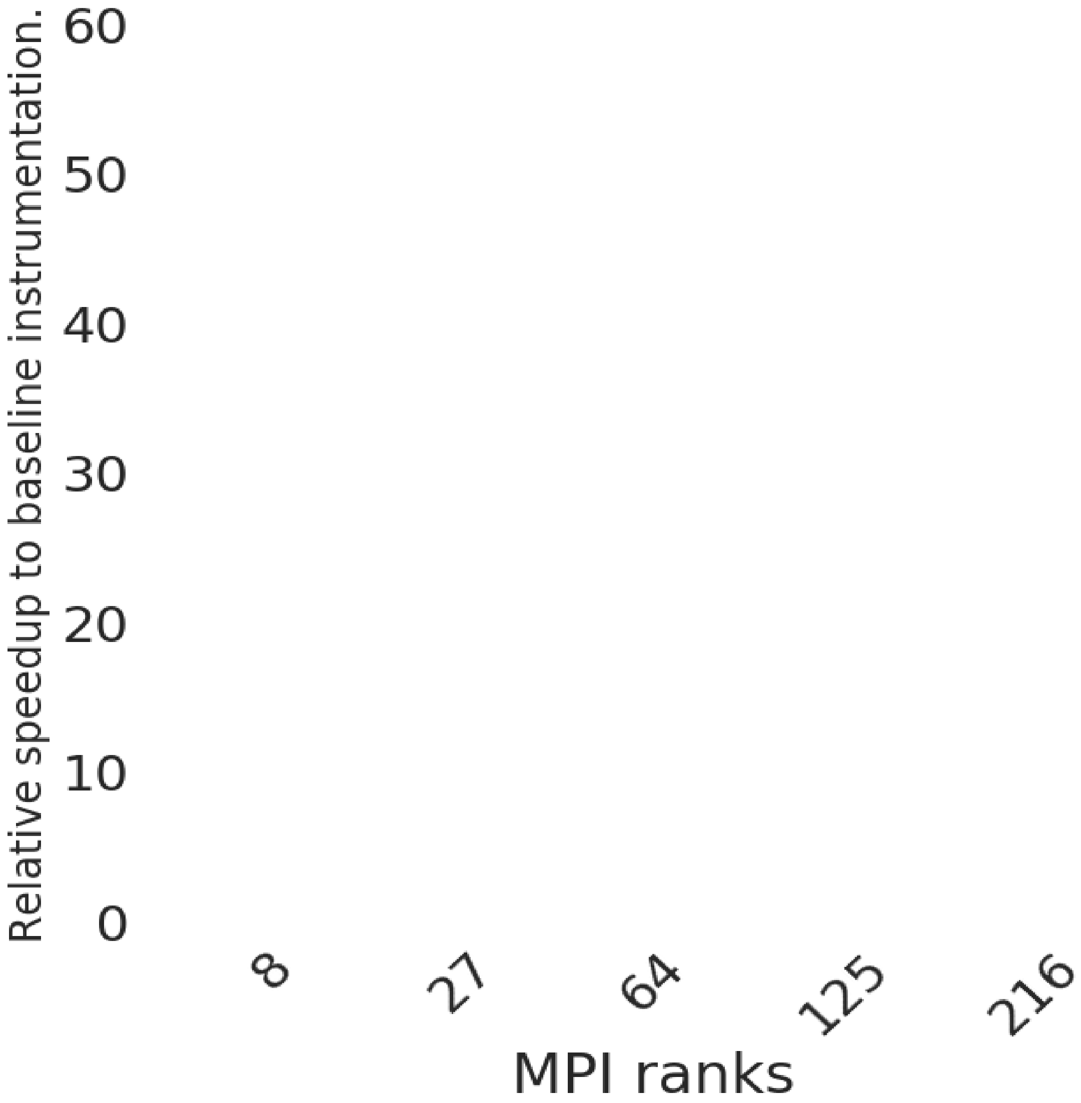
Score-P
Scalable performance measurement infrastructure for parallel codes

Faster experiments with selective instrumentation

MILC su3_rmd (C)

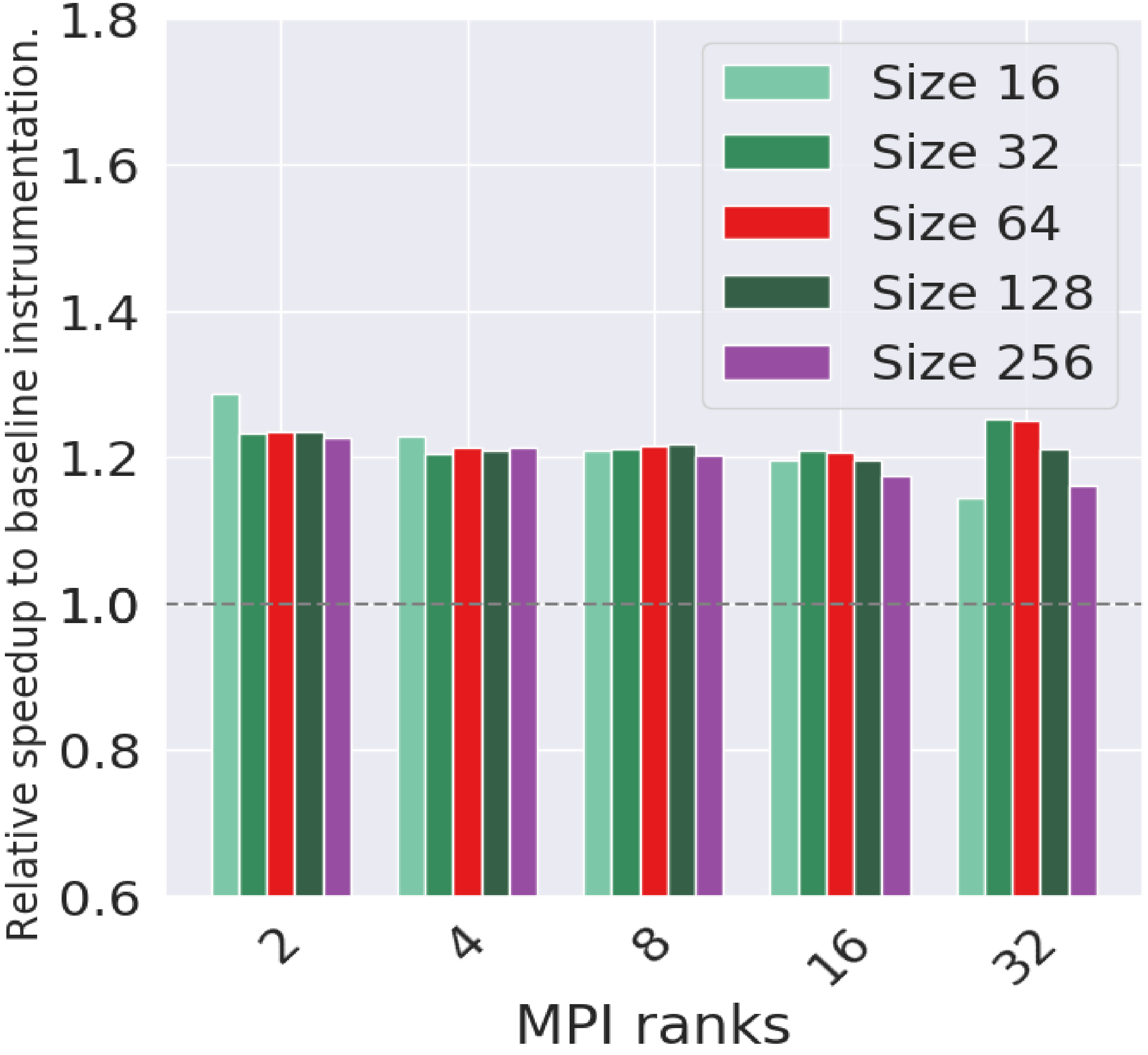


LULESH (C++)

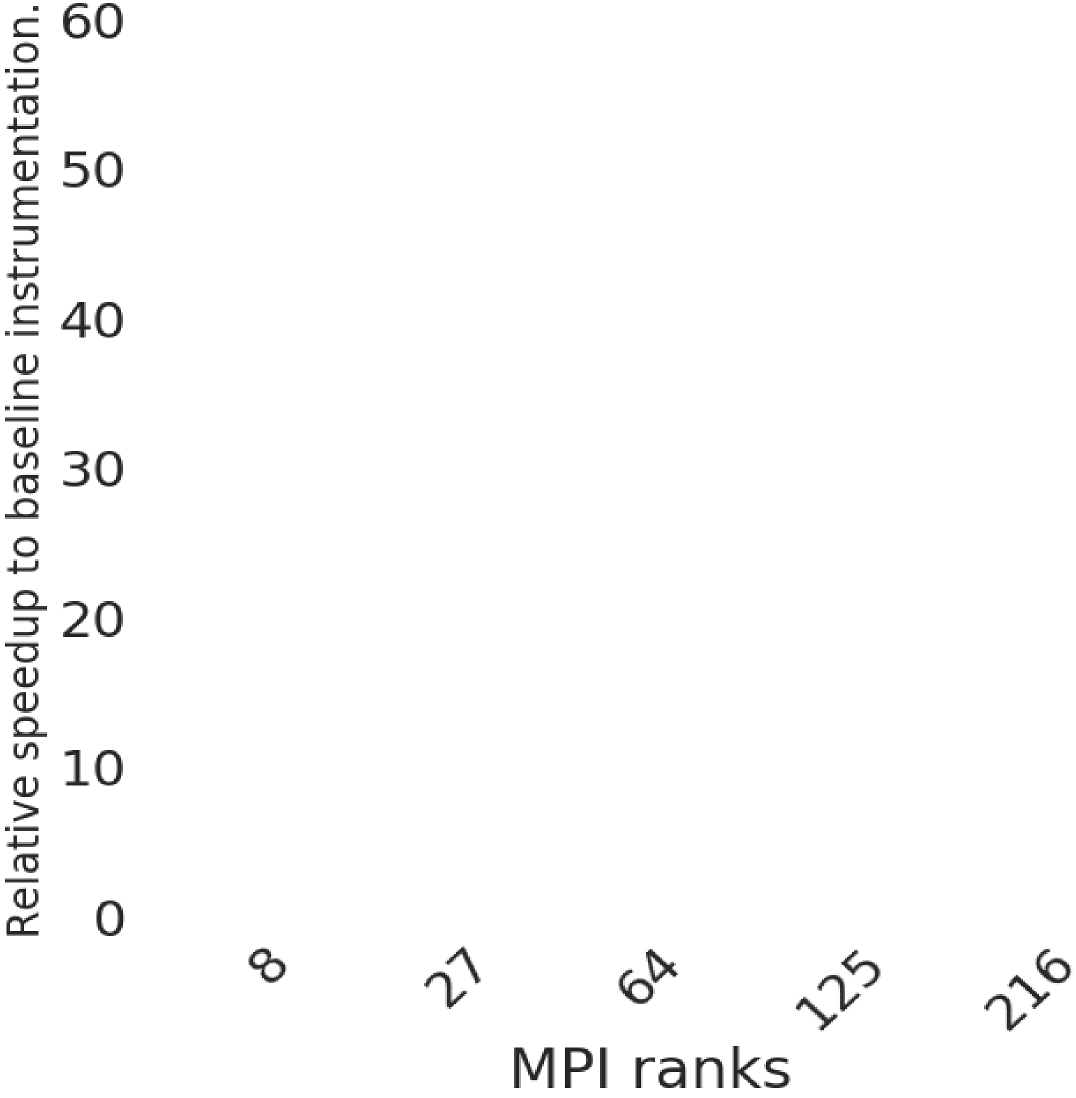


Faster experiments with selective instrumentation

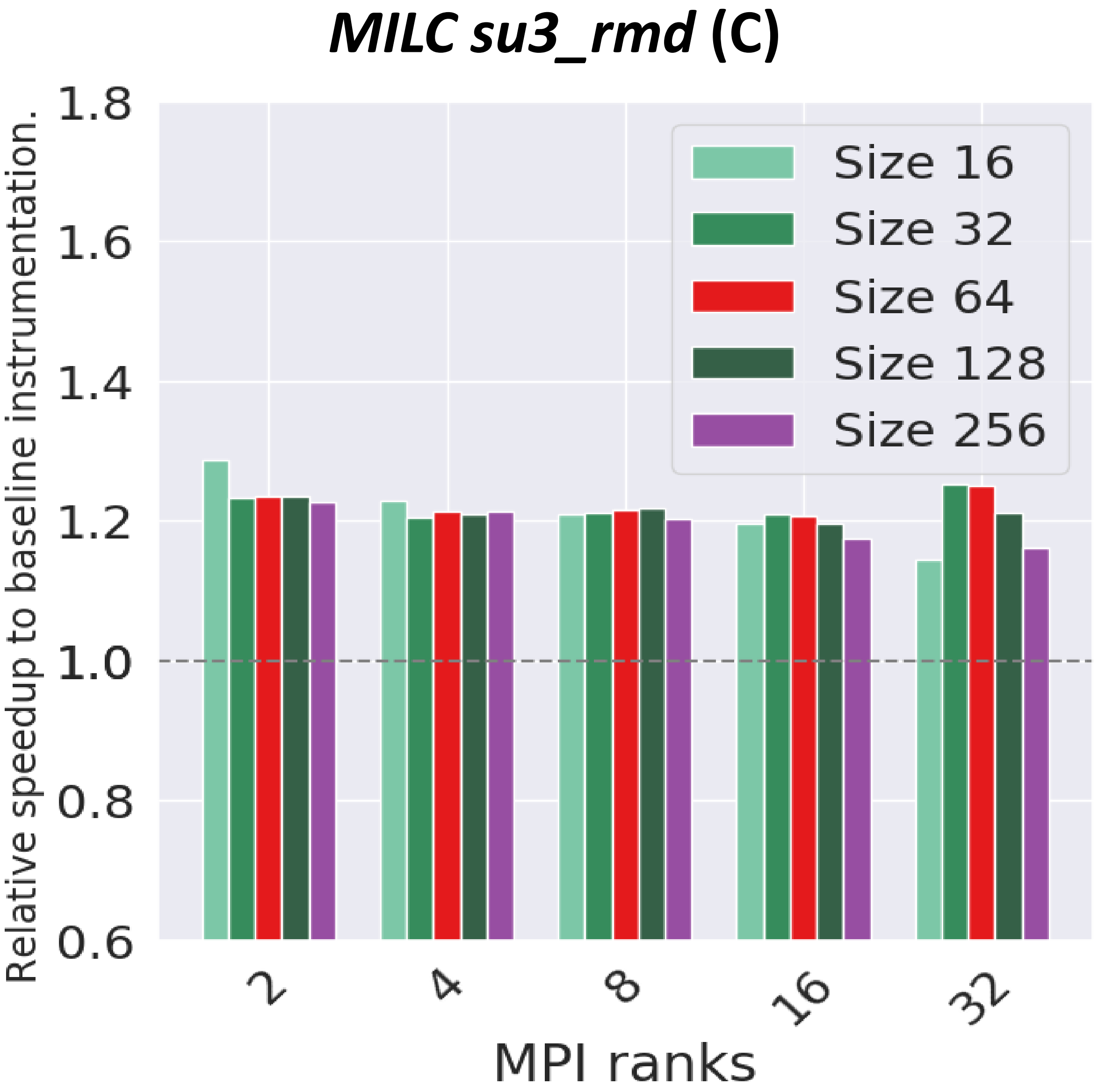
MILC su3_rmd (C)

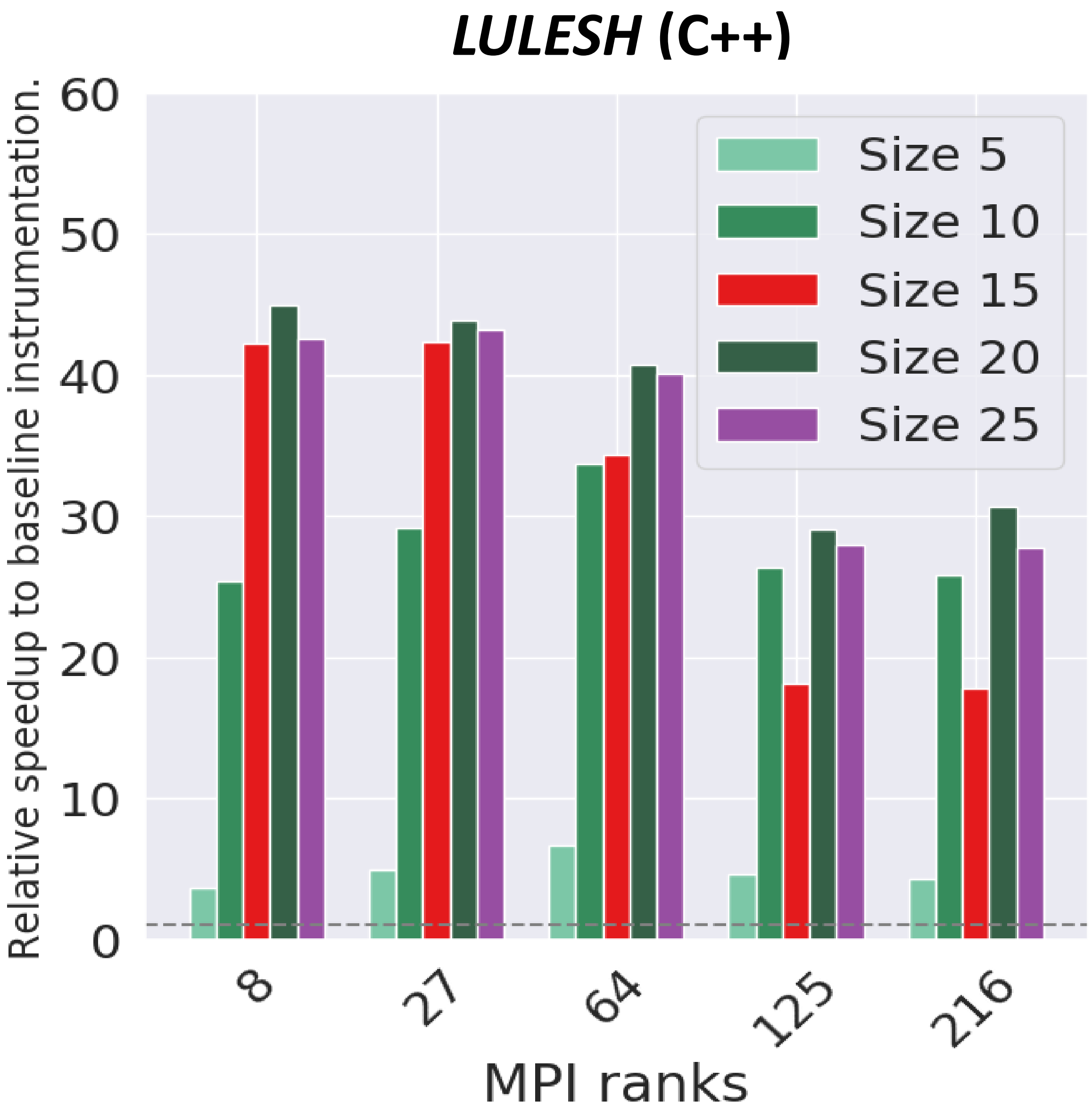


LULESH (C++)



Faster experiments with selective instrumentation





Better models.

LULESH, *CalcHourglassControlForElems* computation kernel
Complexity $O(size^3)$

Better models.

LULESH, *CalcHourglassControlForElems* computation kernel
Complexity $O(size^3)$

✗

$$9.7 \times 10^{-7} s^{2.5} \log_2 s + \mathbf{0.0024 \log_2 p} - 0.016$$

Better models.

LULESH, *CalcHourglassControlForElems* computation kernel
Complexity $O(size^3)$

✗

$$9.7 \times 10^{-7} s^{2.5} \log_2 s + \mathbf{0.0024 \log_2 p} - 0.016$$

✓

$$7.6 \times 10^{-7} s^{2.5} \log_2 s - 0.0025$$

…and better models.

MILC su3_rmd, *do_gather* communication routine

…and better models.

MILC su3_rmd, *do_gather* communication routine

✗

$$8.2 \times 10^{-12} p^{3} \boldsymbol{s}^{\mathbf{0.75}} \log_2 p + 6.2 \times 10^{-6}$$

…and better models.

MILC su3_rmd, *do_gather* communication routine

✗

$$8.2 \times 10^{-12} p^3 \boldsymbol{s}^{\mathbf{0.75}} \log_2 p + 6.2 \times 10^{-6}$$

✓

$$2.2 \times 10^{-12} p^3 \log_2 p + 2.4 \times 10^{-6}$$

…and better models.

MILC su3_rmd, *do_gather* communication routine

✗

$$8.2 \times 10^{-12} p^{3} \boldsymbol{s}^{\mathbf{0.75}} \log_2 p + 6.2 \times 10^{-6}$$

✓

$$2.2 \times 10^{-12} p^{3} \log_2 p + 2.4 \times 10^{-6}$$

Validation	Runtime
s = 2048, p = 1024	0.039 s

…and better models.

MILC su3_rmd, *do_gather* communication routine

✗

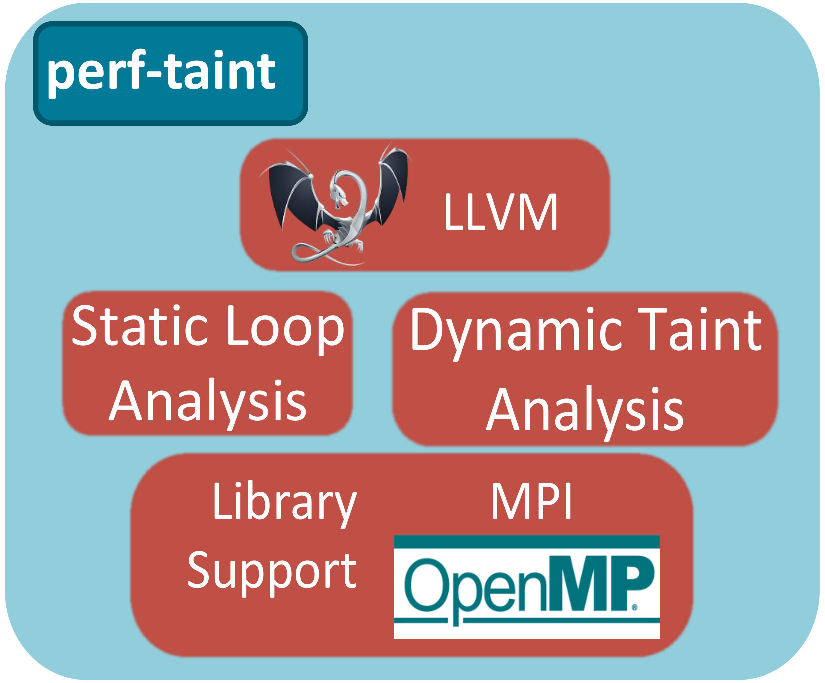
$$8.2 \times 10^{-12} p^3 \boldsymbol{s^{0.75}} \log_2 p + 6.2 \times 10^{-6}$$

✓

$$2.2 \times 10^{-12} p^3 \log_2 p + 2.4 \times 10^{-6}$$

Validation	Runtime	Black-box model	White-box model
s = 2048, p = 1024	0.039 s	26.7 s	0.023 s

Summary



perf-taint

LLVM

Static Loop Analysis

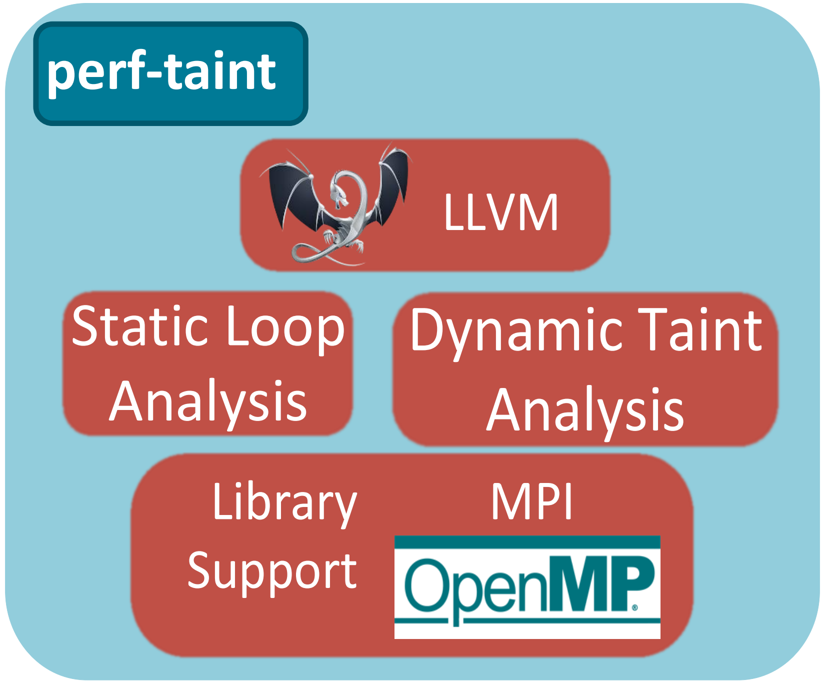
Dynamic Taint Analysis

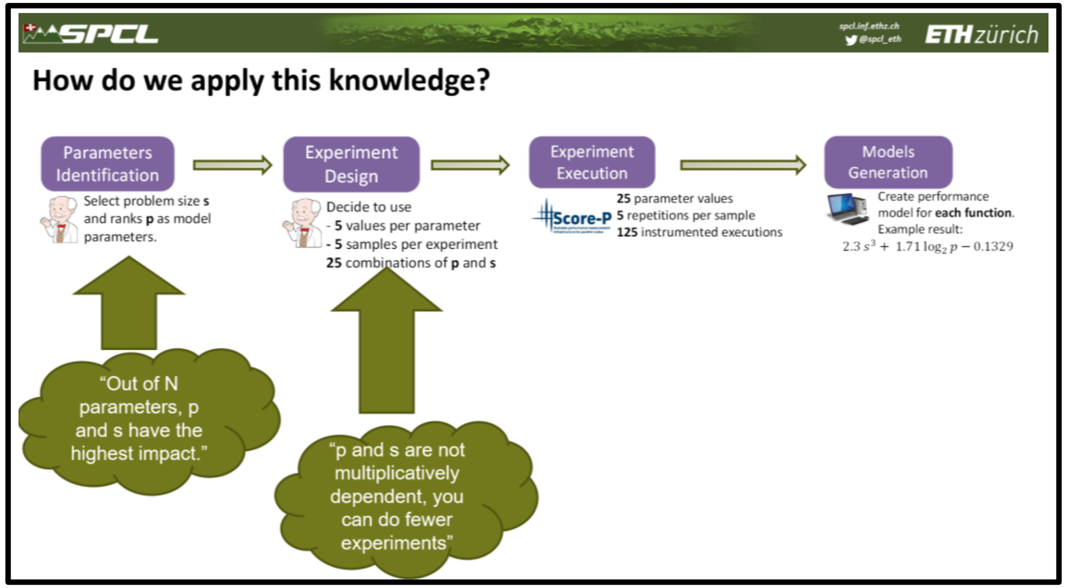
Library Support

MPI

OpenMP

Summary





How do we apply this knowledge?

Parameters Identification → Experiment Design → Experiment Execution → Models Generation

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**

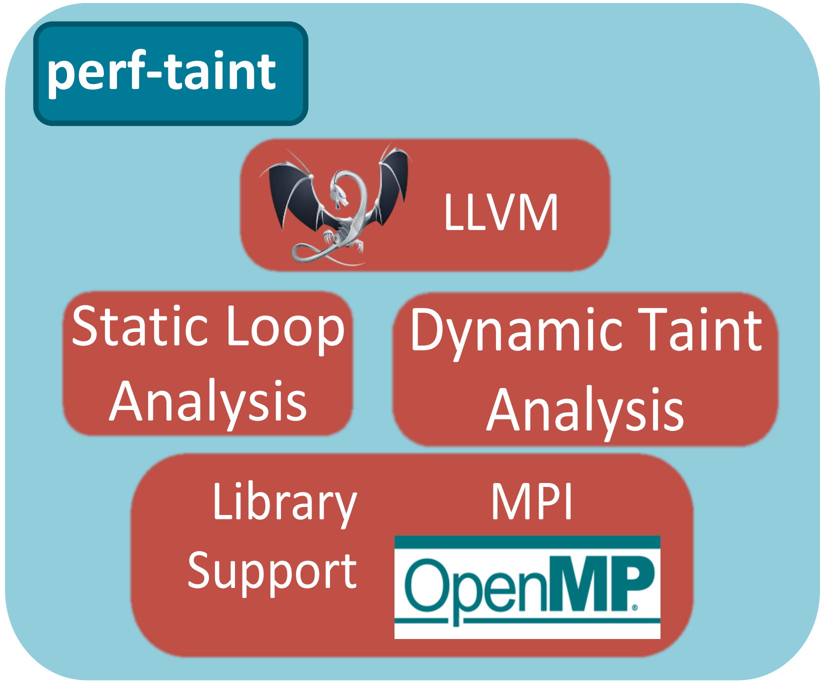
25 parameter values
5 repetitions per sample
125 instrumented executions

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

"Out of N parameters, p and s have the highest impact."

"p and s are not multiplicatively dependent, you can do fewer experiments"

Summary



perf-taint

LLVM

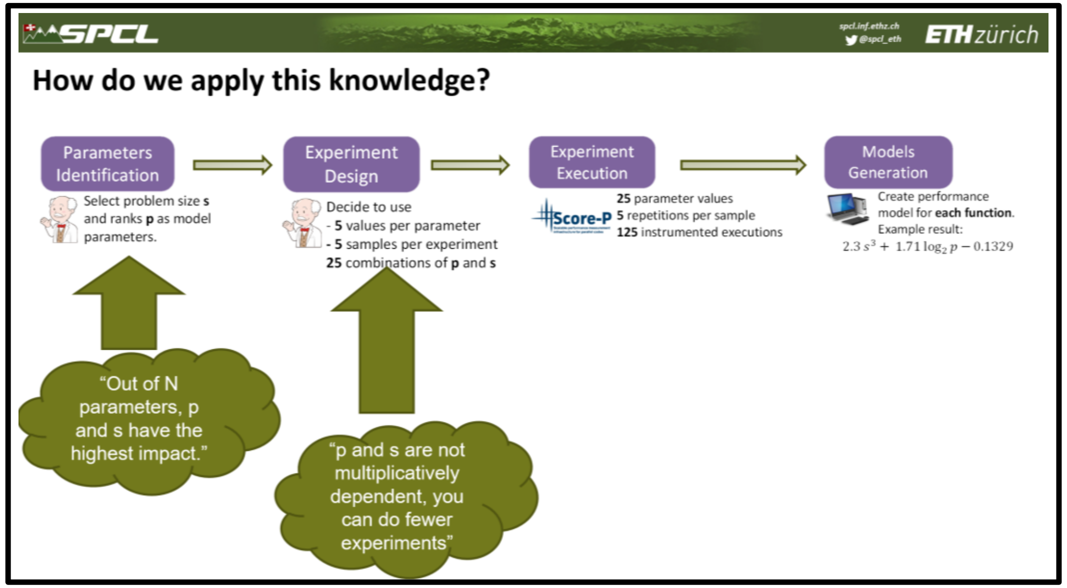
Static Loop Analysis

Dynamic Taint Analysis

Library Support

MPI

OpenMP



How do we apply this knowledge?

Parameters Identification → Experiment Design → Experiment Execution → Models Generation

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

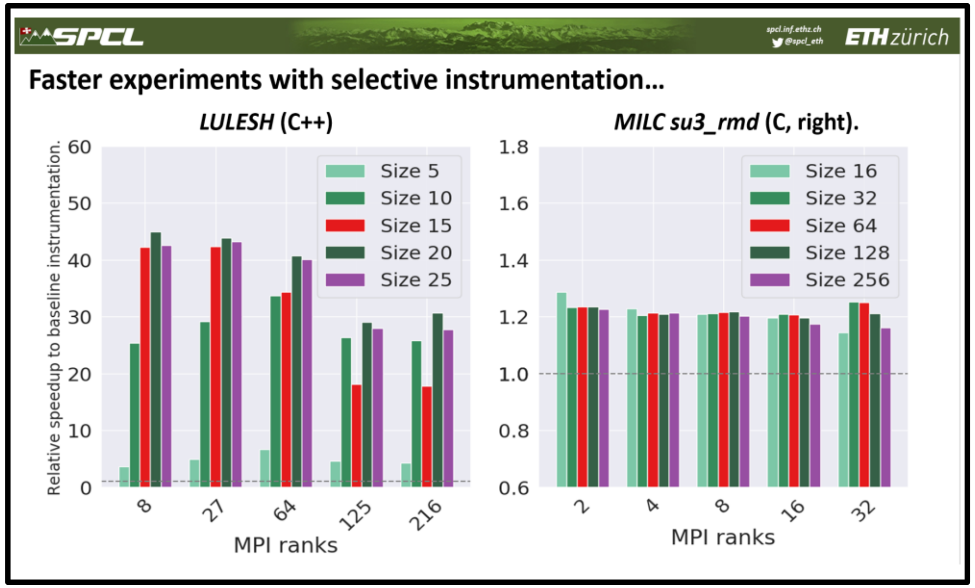
Decide to use
- **5** values per parameter
- **5** samples per experiment
25 combinations of **p** and **s**

Score-P
25 parameter values
5 repetitions per sample
125 instrumented executions

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

"Out of N parameters, p and s have the highest impact."

"p and s are not multiplicatively dependent, you can do fewer experiments"

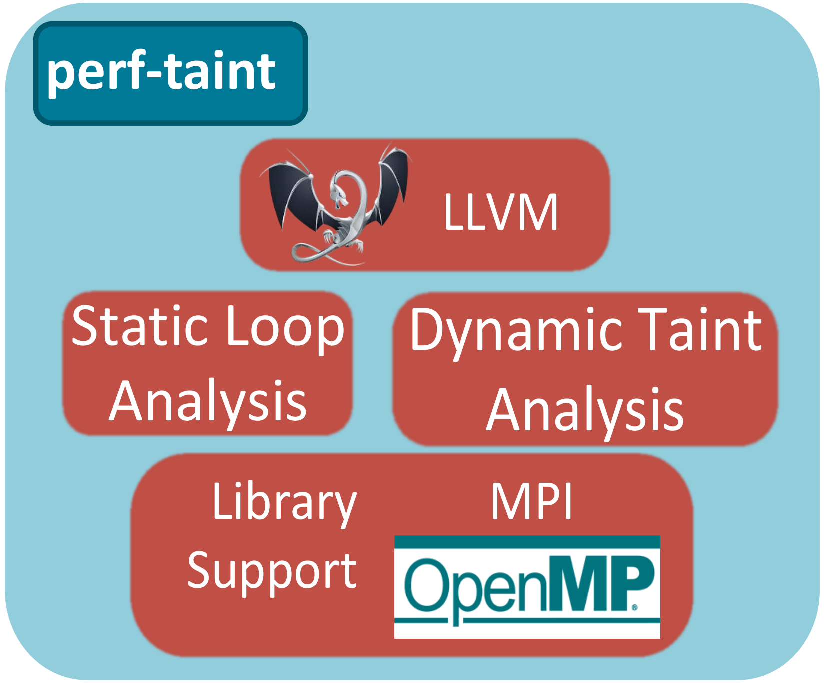


Faster experiments with selective instrumentation...

LULESH (C++)

MILC su3_rmd (C, right).

Summary



perf-taint

LLVM

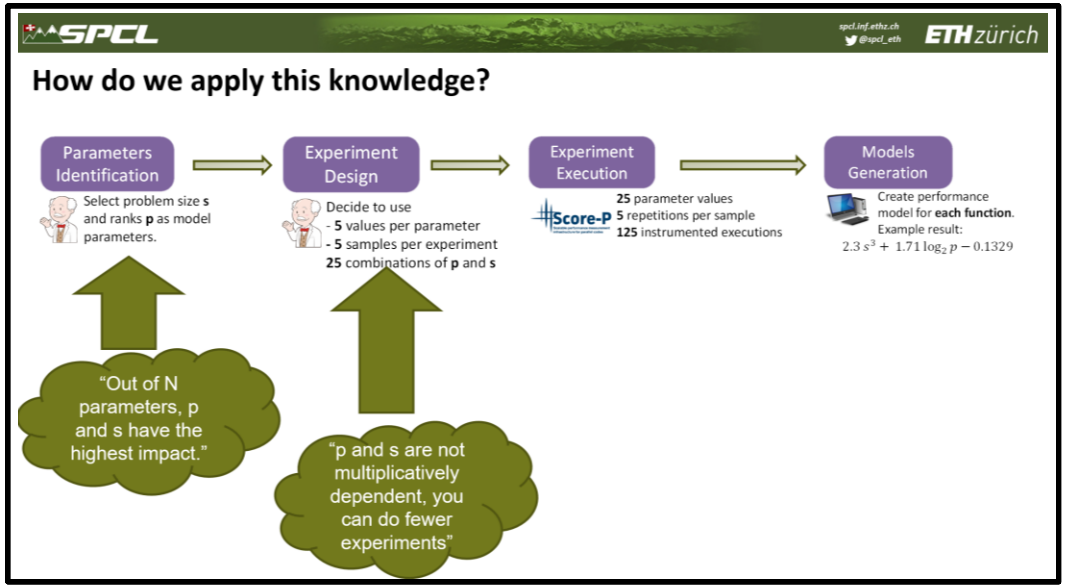
Static Loop Analysis

Dynamic Taint Analysis

Library Support

MPI

OpenMP



How do we apply this knowledge?

Parameters Identification → Experiment Design → Experiment Execution → Models Generation

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**

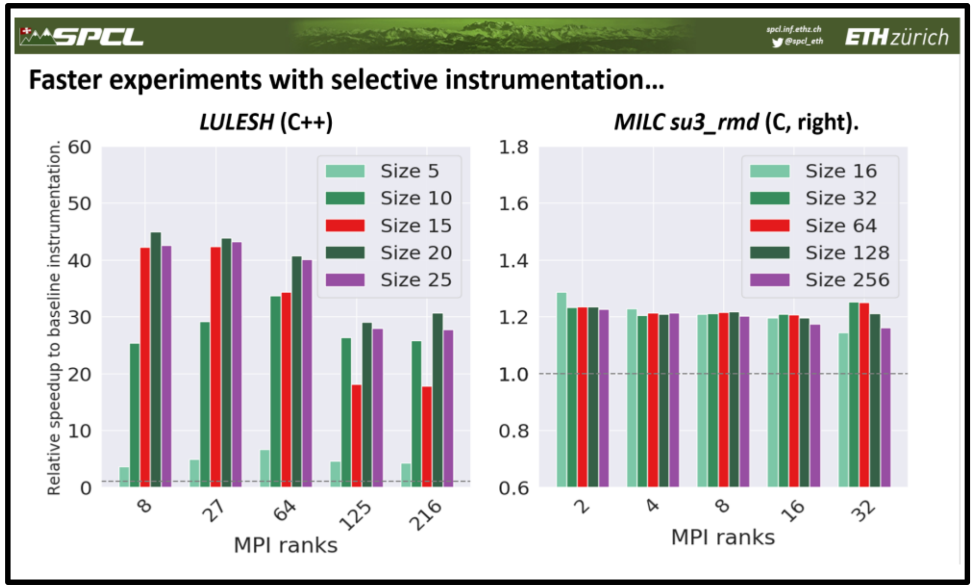
Score-P

25 parameter values
5 repetitions per sample
125 instrumented executions

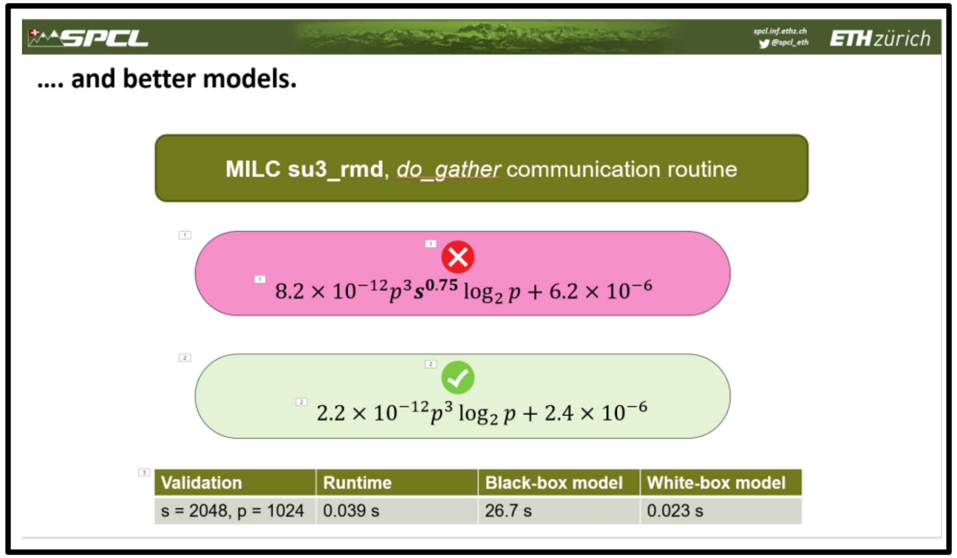
Create performance model for **each function**.
Example result:
$2.3\, s^3 + 1.71 \log_2 p - 0.1329$

“Out of N parameters, p and s have the highest impact.”

“p and s are not multiplicatively dependent, you can do fewer experiments”



Faster experiments with selective instrumentation…



…. and better models.

MILC su3_rmd, *do_gather* communication routine

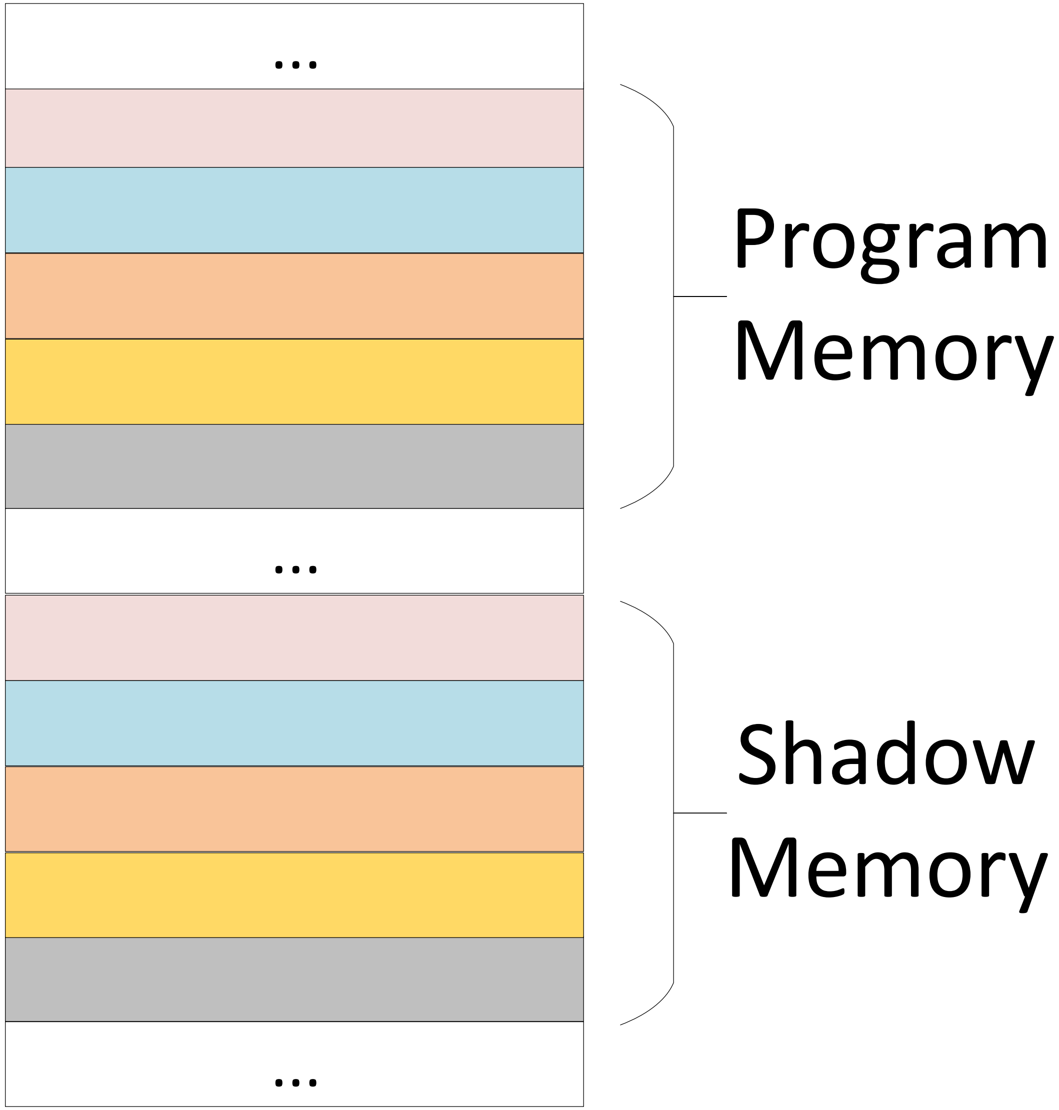
❌ $8.2 \times 10^{-12} p^3 s^{0.75} \log_2 p + 6.2 \times 10^{-6}$

✓ $2.2 \times 10^{-12} p^3 \log_2 p + 2.4 \times 10^{-6}$

Validation	Runtime	Black-box model	White-box model
s = 2048, p = 1024	0.039 s	26.7 s	0.023 s

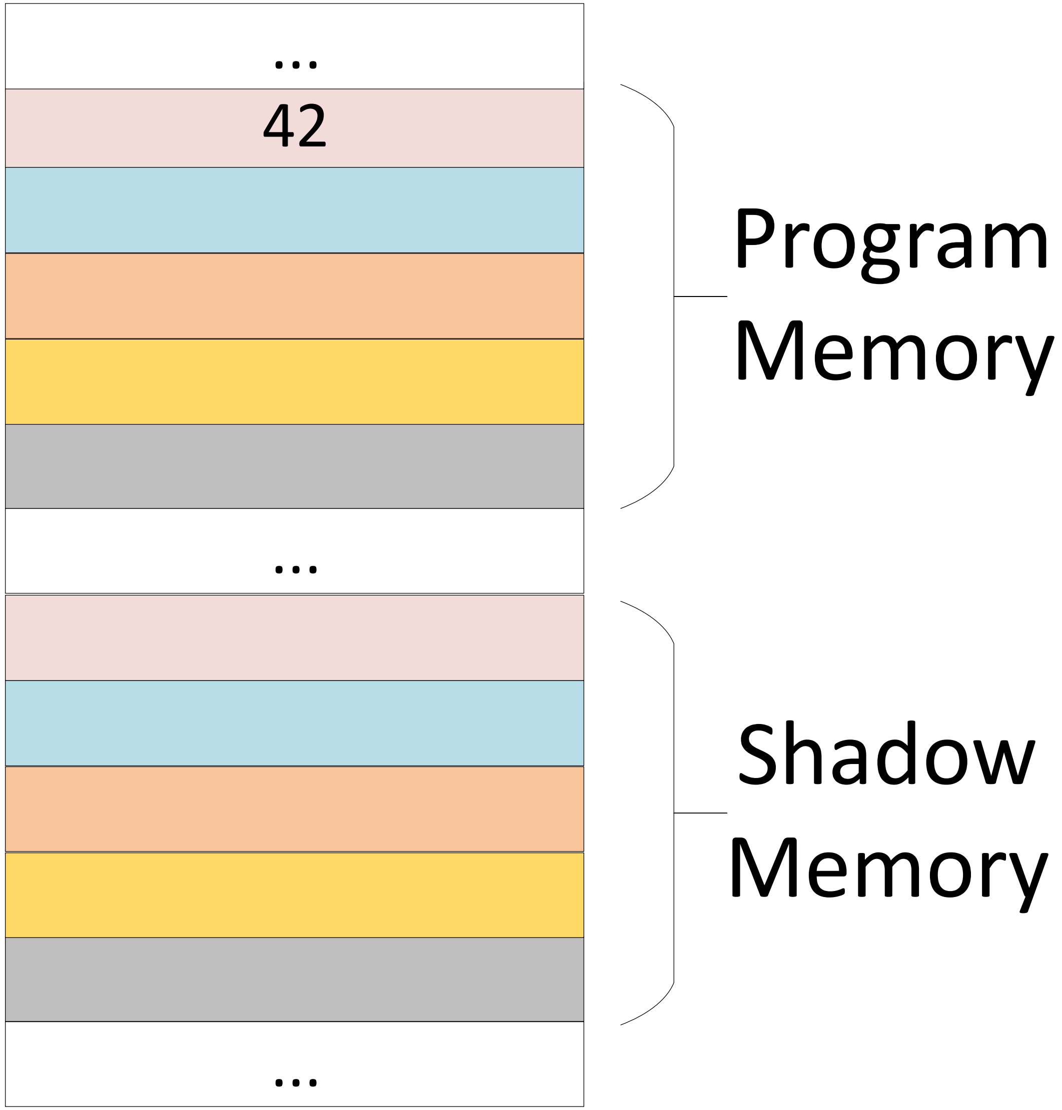
Taint Analysis: track parameters propagation

```
int a = 42;
int b = omp_get_num_threads();
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;
```



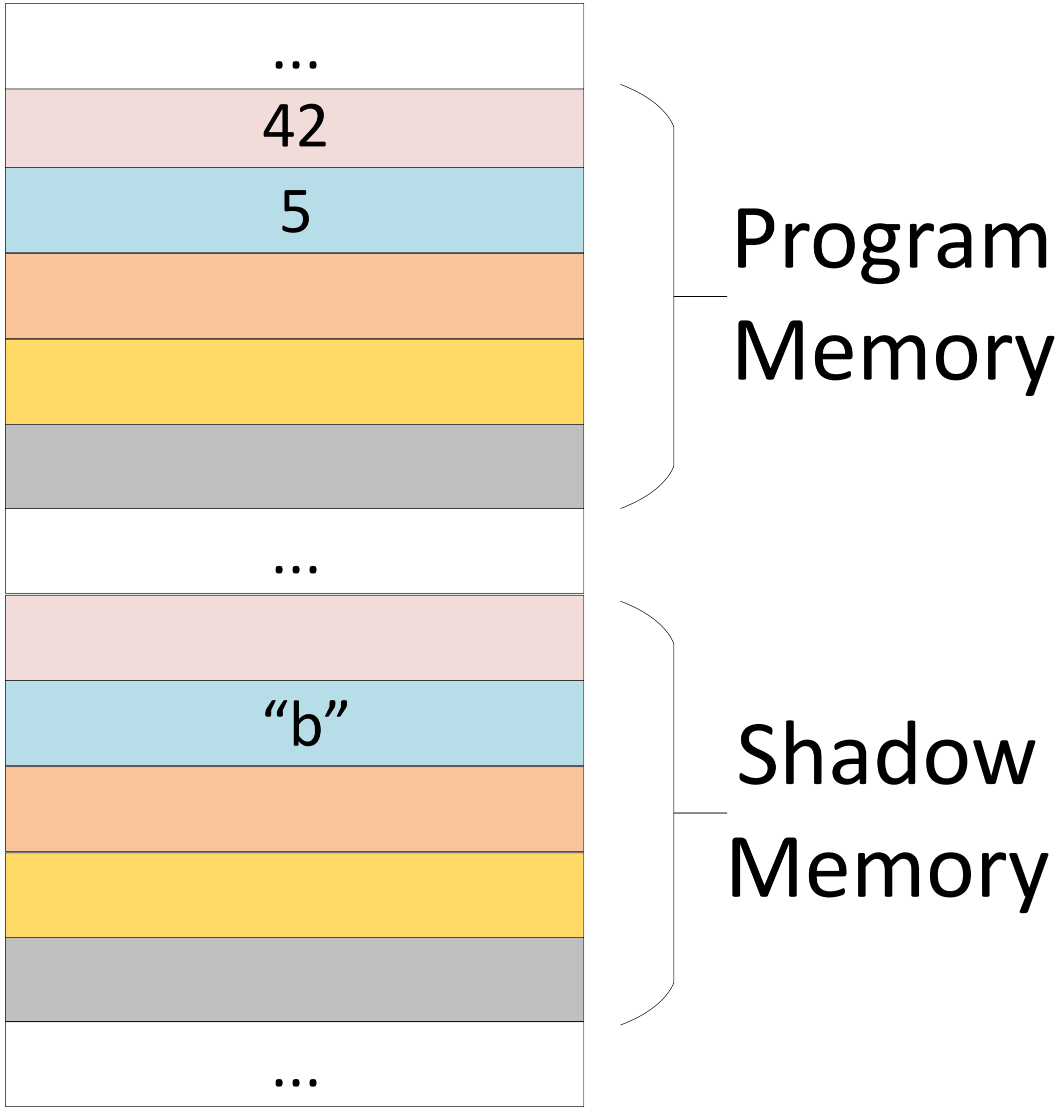
Taint Analysis: track parameters propagation

```
int a = 42;
int b = omp_get_num_threads();
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;
```



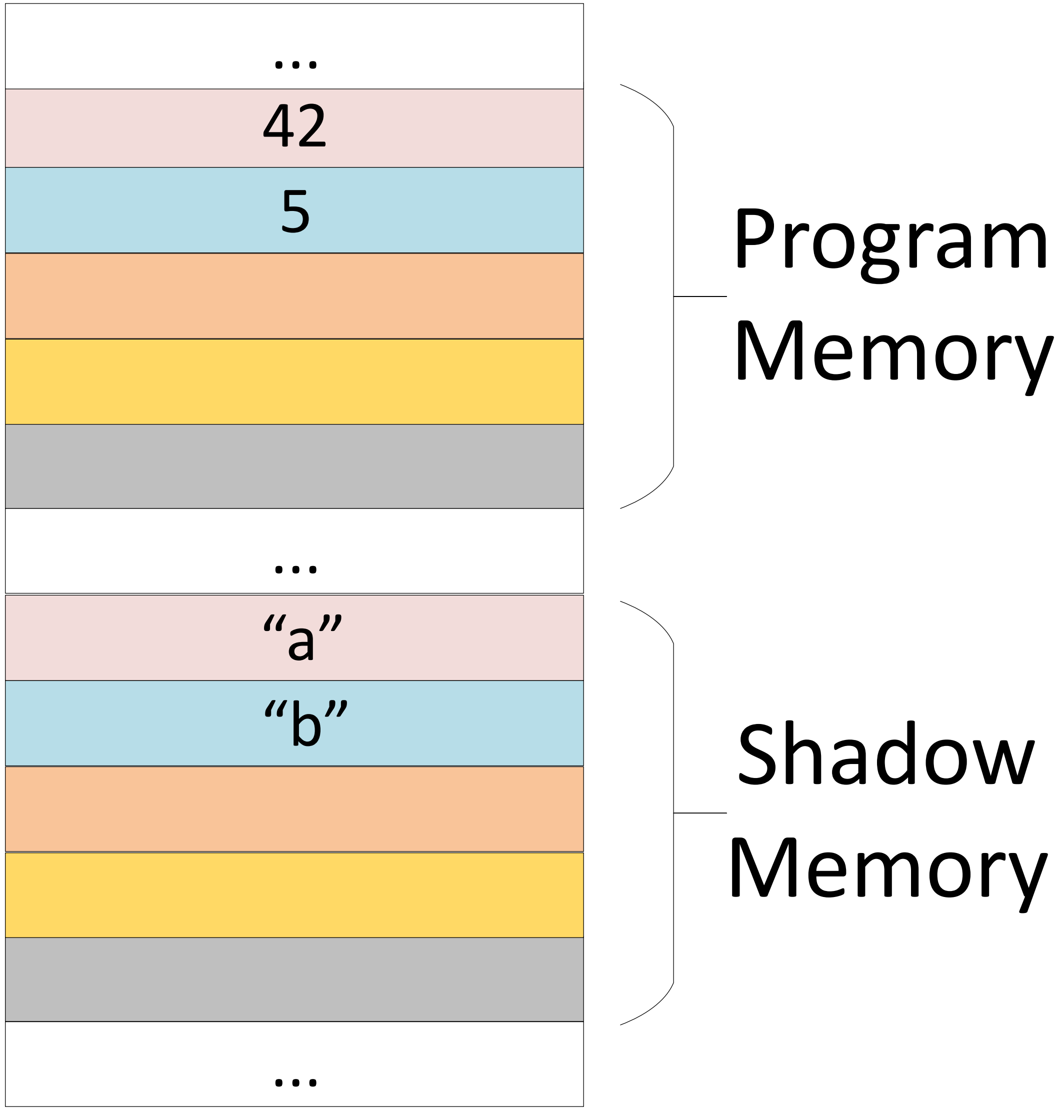
Taint Analysis: track parameters propagation

```
int a = 42;
int b = omp_get_num_threads();
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 = omp_get_num_threads();
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;
```



…

42

5

…

“a”

“b”

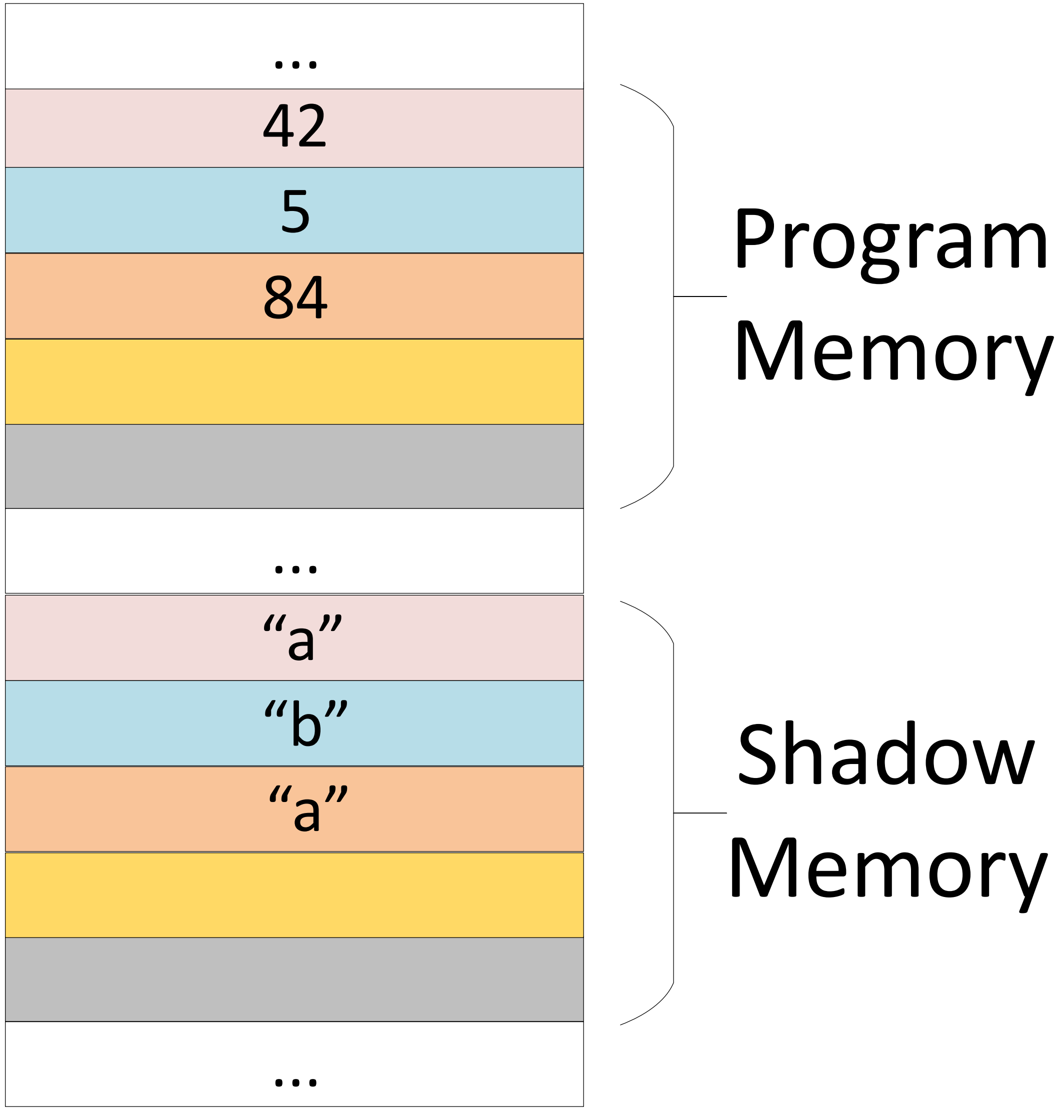
…

Program Memory

Shadow Memory

Taint Analysis: track parameters propagation

```
int a = 42;
int b = omp_get_num_threads();
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;
```



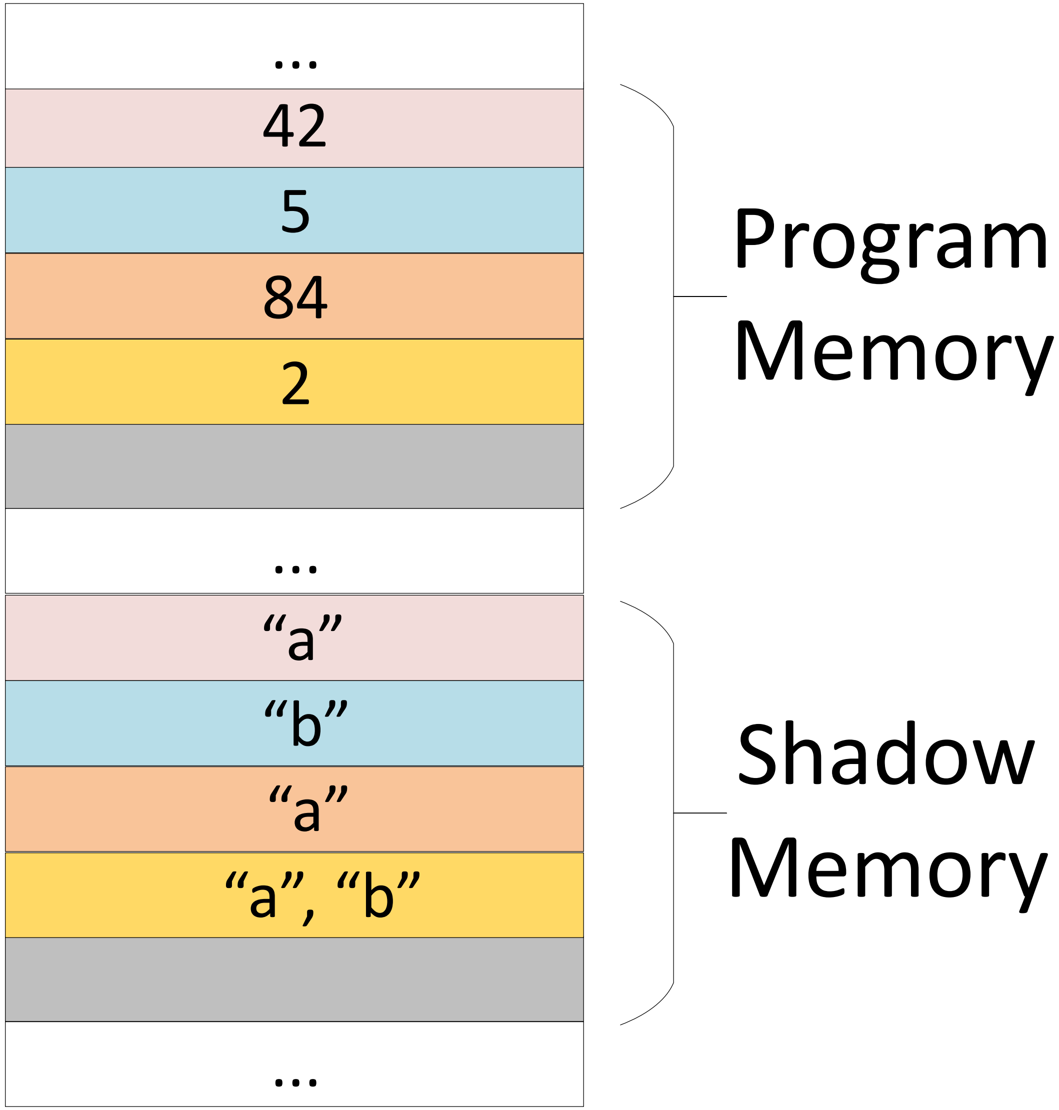
...
42
5
84

...
“a”
“b”
“a”

...

Taint Analysis: track parameters propagation

```
int a = 42;
int b = omp_get_num_threads();
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;
```



…
42
5
84
2

…
“a”
“b”
“a”
“a”, “b”

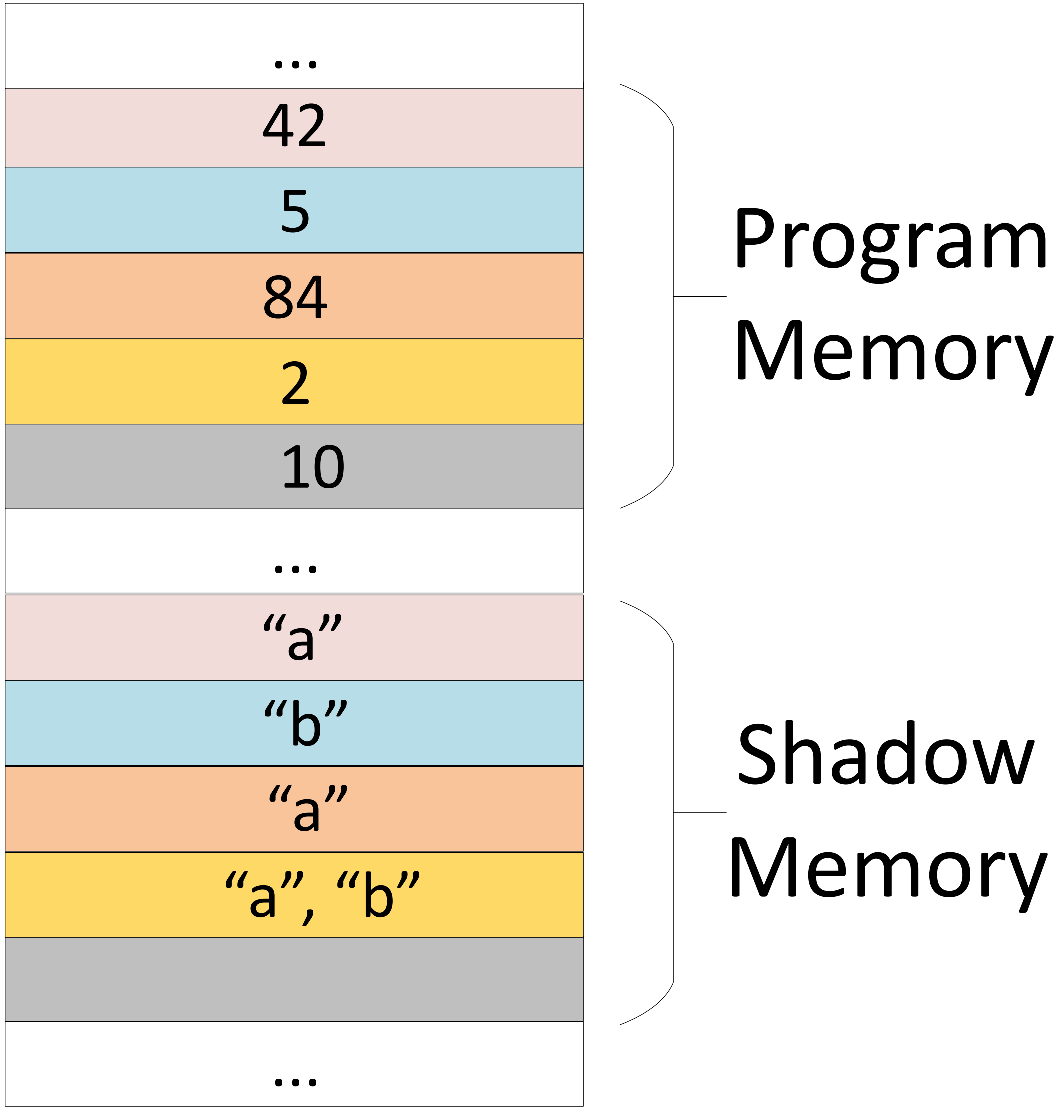
…

Program Memory

Shadow Memory

Taint Analysis: track parameters propagation

```
int a = 42;
int b = omp_get_num_threads();
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;
```

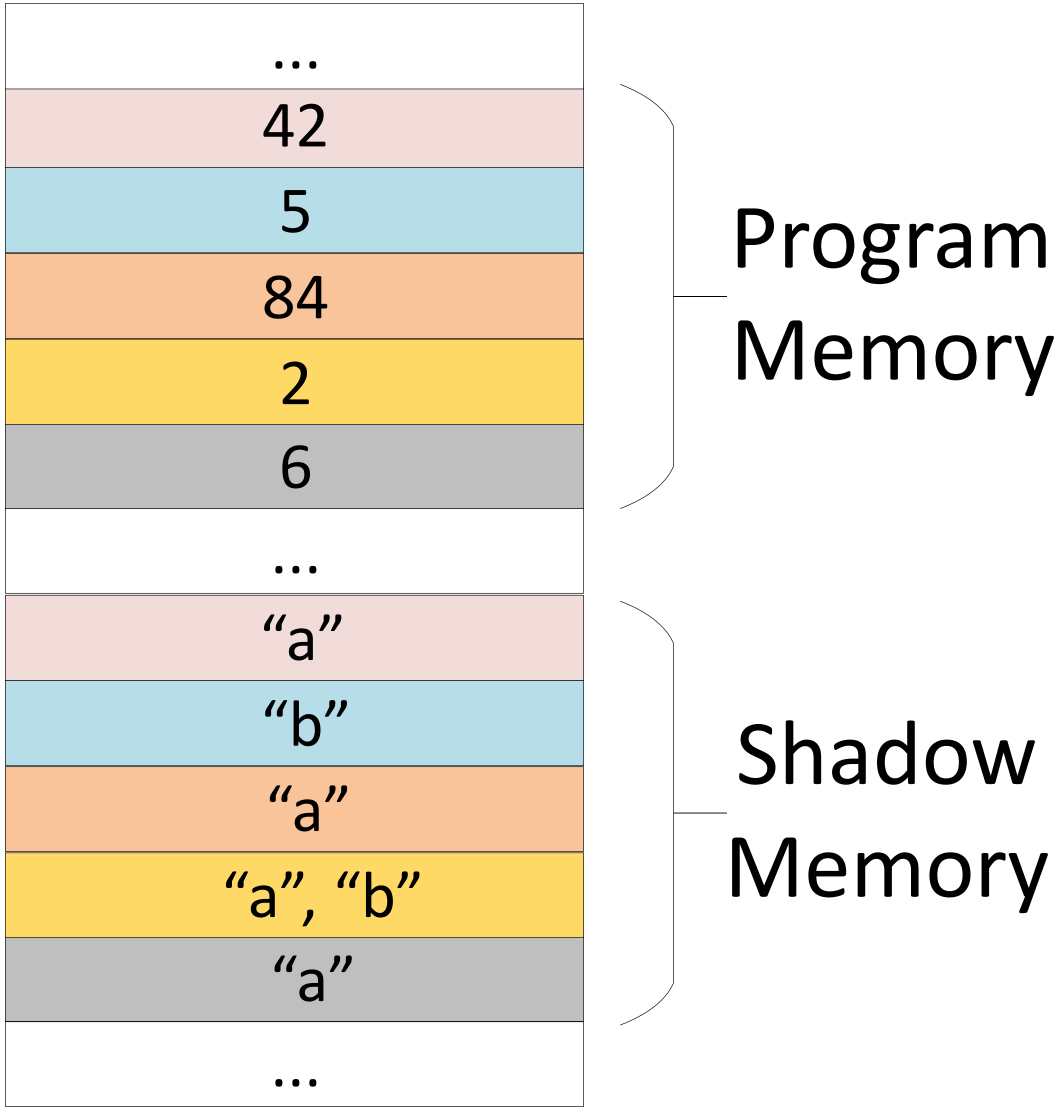


…
42
5
84
2
10
…
“a”
“b”
“a”
“a”, “b”

…

Taint Analysis: track parameters propagation

```
int a = 42;
int b = omp_get_num_threads();
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;
```



…
42
5
84
2
6
…
"a"
"b"
"a"
"a", "b"
"a"
…