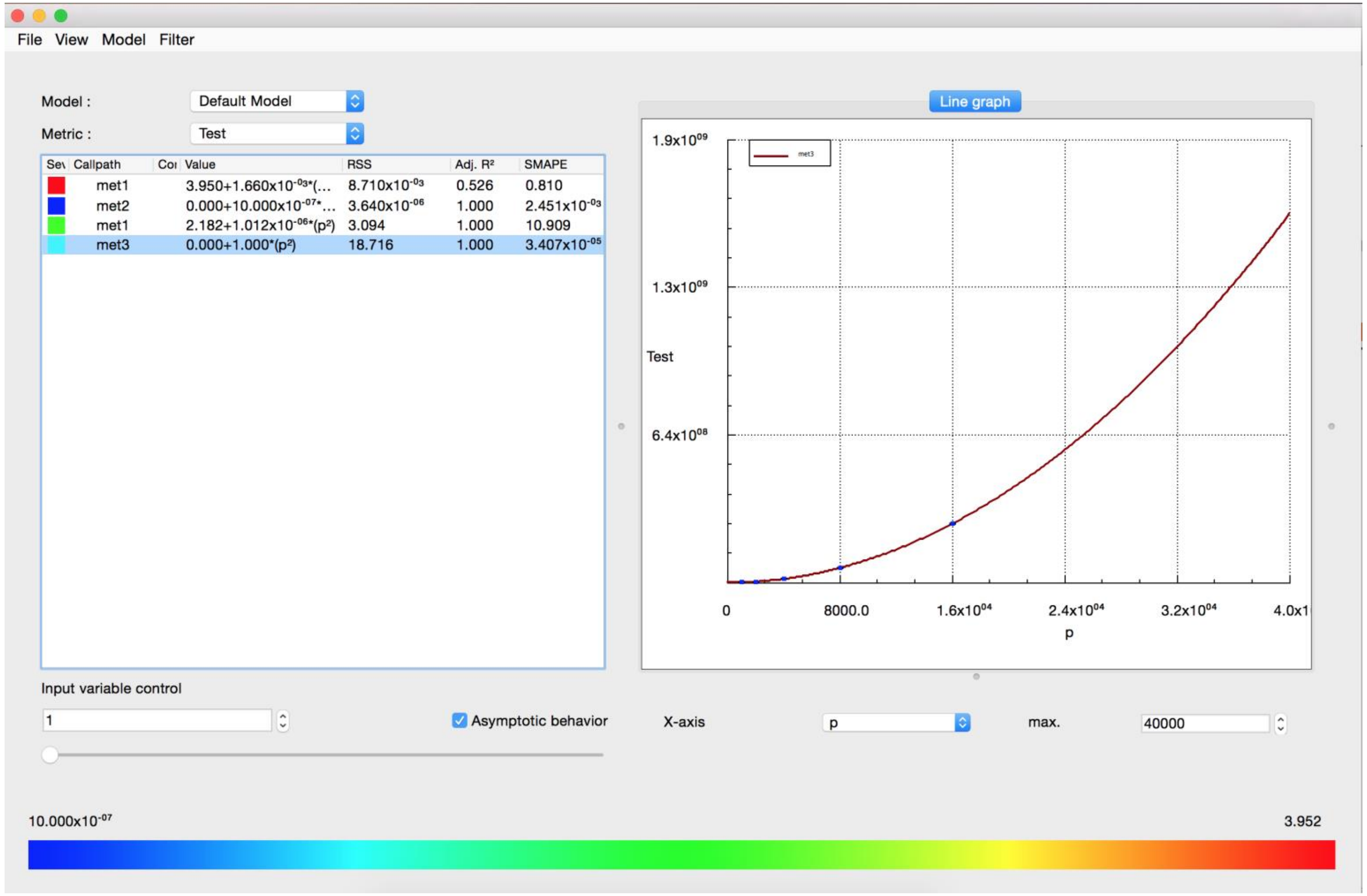
perf-taint: Taint Analysis for Automatic Many-Parameter Performance Modeling

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

SPPEXA

Dresden, 23th October 2019

Extra-P



Performance Modeling

- Mostly automatic
- Parametric models of functions
- Black-box approach

But how much work is still performed by the user?

Challenges in Automatic Performance Modeling

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

Challenges in Automatic Performance Modeling

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 and **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:

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

#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 → Experiment Design → Experiment Execution → Models Generation

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

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

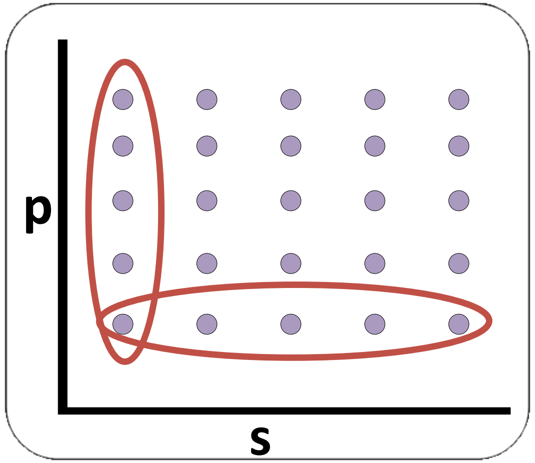
Score-P

25 parameter values
5 repetitions per sample
125 instrumented executions

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

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

#2: How parameters interact with each other?



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

Challenges in Automatic Performance Modeling

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 and **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:

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

```
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 → **Experiment Design** → **Experiment Execution** → **Models Generation**

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

Decide to use **5** values per parameter and **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:

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

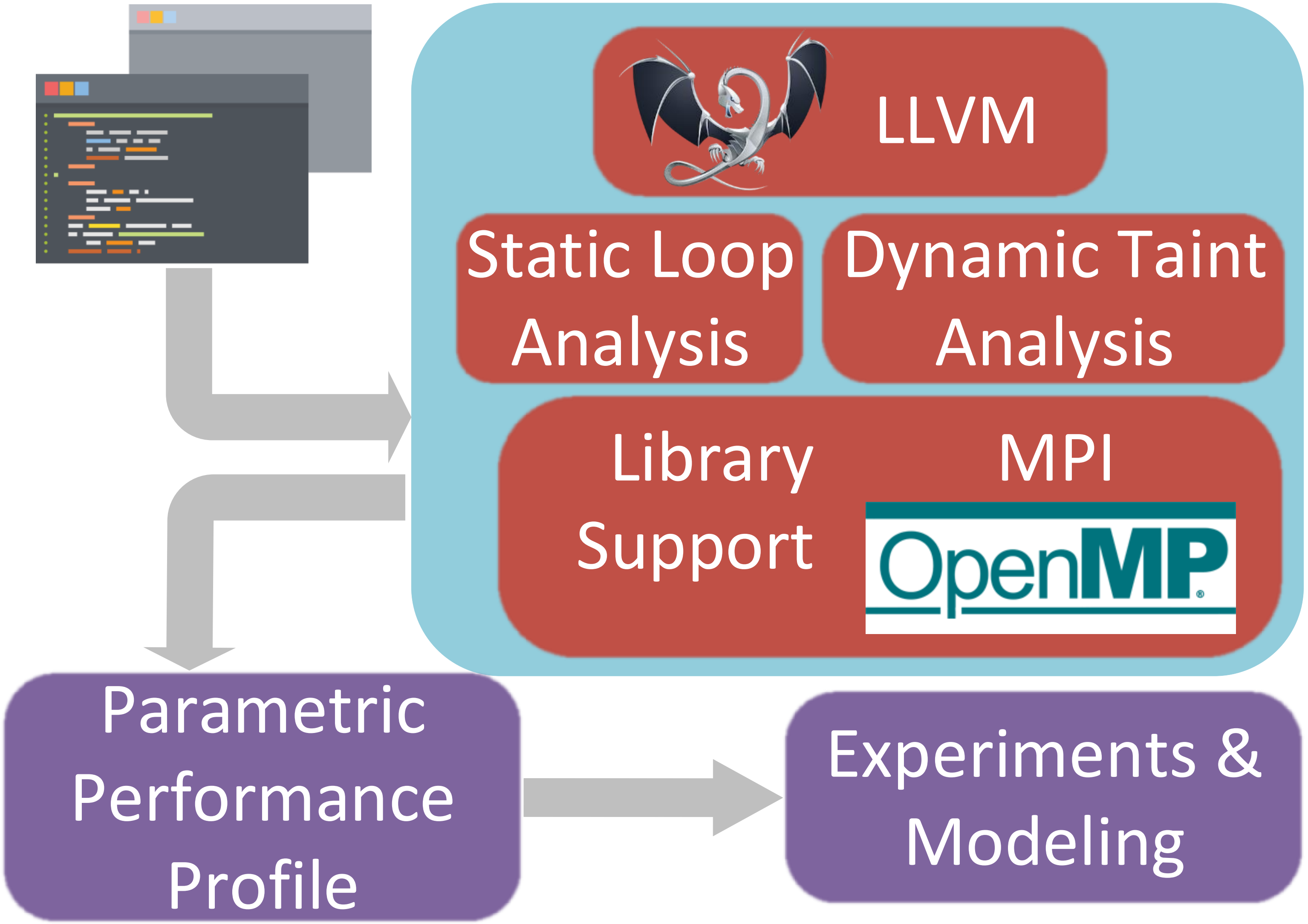
#1: Which parameters are relevant?

#2: How parameters interact with each other?

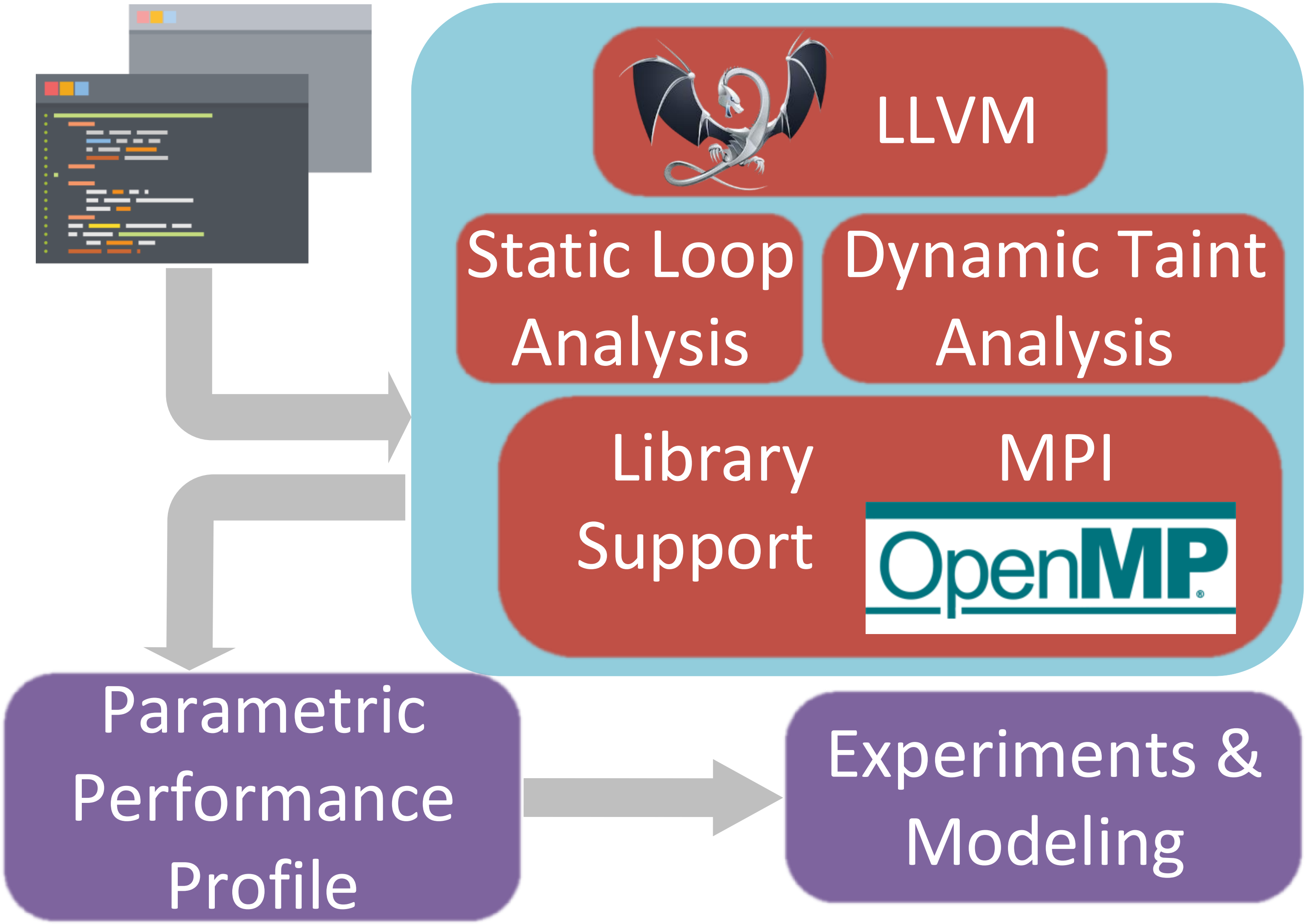
#3: Which functions and parameters affect performance?

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

Hybrid Taint Analysis



Hybrid Taint Analysis



Why do we need a dynamic analysis? Static techniques are often limited by:

- Theoretical
- Inter-procedural dataflow
- Data-dependent control-flow
- Overapproximations, e.g. in alias analysis
- High-level abstractions

What if we try to model an application built in dynamically typed language (Julia, Python)?

Parametric Performance Profile

```
void f(int a, int b) {
  taint_variables(a, b);
  g(a, b); h(a, b); i(a, b);
}
```

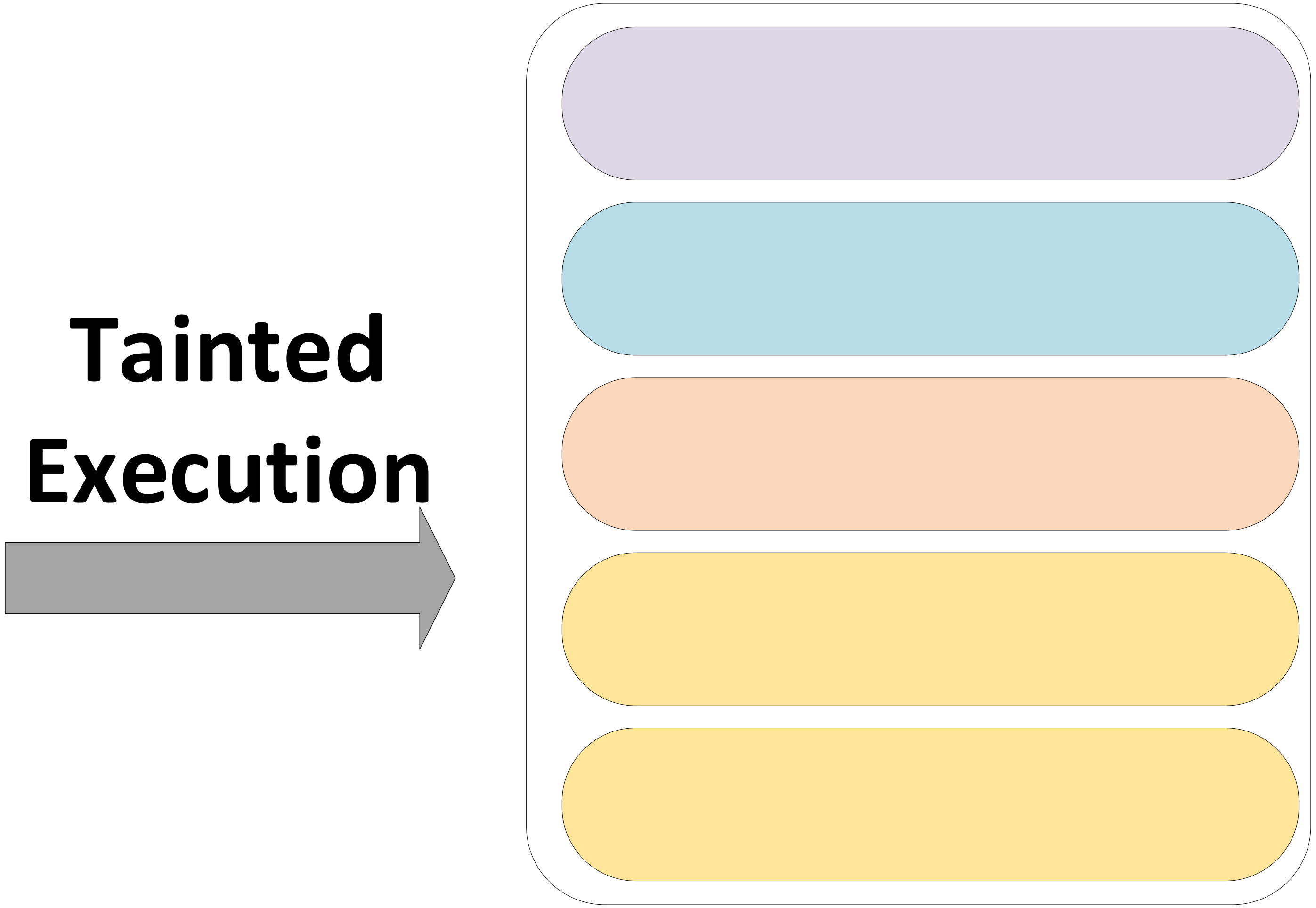
```
void g(int a, int b) {
  for(int i = 0; i < a; ++i)
    j(b);
}
```

```
void h(int a, int b) {
  j(a);
}
```

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

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

Tainted Execution



Parametric Performance Profile

```c
void f(int a, int b) {
  taint_variables(a, b);
  g(a, b); h(a, b); i(a, b);
}
```

```c
void g(int a, int b) {
  for(int i = 0; i < a; ++i)
    j(b);
}
```

```c
void h(int a, int b) {
  j(a);
}
```

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

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

Tainted Execution

$(f, g) \rightarrow \{b\}$

$(f, h) \rightarrow \{a\}$

Parametric Performance Profile

```
void f(int a, int b) {
  taint_variables(a, b);
  g(a, b); h(a, b); i(a, b);
}
```

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

```
void h(int a, int b) {
  j(a);
}
```

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

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

Tainted Execution →

$(f) \rightarrow \{a\}$

$(f, g) \rightarrow \{b\}$

$(f, h) \rightarrow \{a\}$

Parametric Performance Profile

```
void f(int a, int b) {
  taint_variables(a, b);
  g(a, b); h(a, b); i(a, b);
}
```

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

```
void h(int a, int b) {
  j(a);
}
```

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

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

Tainted Execution →

$(f) \rightarrow \{a, b, a \times b\}$

$(f) \rightarrow \{a\}$

$(f, g) \rightarrow \{b\}$

$(f, h) \rightarrow \{a\}$

Parametric Performance Profile

```
void f(int a, int b) {
  taint_variables(a, b);
  g(a, b); h(a, b); i(a, b);
}
```

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

```
void h(int a, int b) {
  j(a);
}
```

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

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

Tainted Execution

$$() \rightarrow \{\{a, b, a \times b\} + \{a\}\}$$

$$(f) \rightarrow \{a, b, a \times b\}$$

$$(f) \rightarrow \{a\}$$

$$(f, g) \rightarrow \{b\}$$

$$(f, h) \rightarrow \{a\}$$

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 and **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:

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

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 and **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:

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

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

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 and **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"

How do we apply this knowledge?

Parameters Identification → Experiment Design → Experiment Execution → Models Generation

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

Experiment Design: Decide to use **5** values per parameter and **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

Models Generation: 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"

"Only 10% of functions are relevant, skip instrumenting the rest."

How do we apply this knowledge?

Parameters Identification → **Experiment Design** → **Experiment Execution** → **Models Generation**

Parameters Identification

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

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

Experiment Design

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

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

Experiment Execution

Score-P
Scalable performance measurement infrastructure for parallel codes

25 parameter values
5 repetitions per sample
125 instrumented executions

"Only 10% of functions are relevant, skip instrumenting the rest."

Models Generation

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

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

Skip irrelevant functions and prune models with false dependencies.

Faster experiments with selective instrumentation…

…. and better models.

LULESH, *CalcHourglassControlForElems*
computation kernel with complexity $O(s^3)$

✗

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

✓

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

…. and better models.

MILC su3_rmd, *do_gather* communication

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

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

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

Summary

We achieved:

- Better understanding of modeled applications
- Faster and cheaper experiments
- Faster modeling
- More accurate models

We're working on:

- Control-flow taint propagation (minority of all cases)
- Recursive functions
- Taint propagation through MPI messages

Summary

We achieved:

- Better understanding of modeled applications
- Faster and cheaper experiments
- Faster modeling
- More accurate models

We're working on:

- Control-flow taint propagation (minor cases)
- Recursive functions
- Taint propagation through MPI messages

Questions?

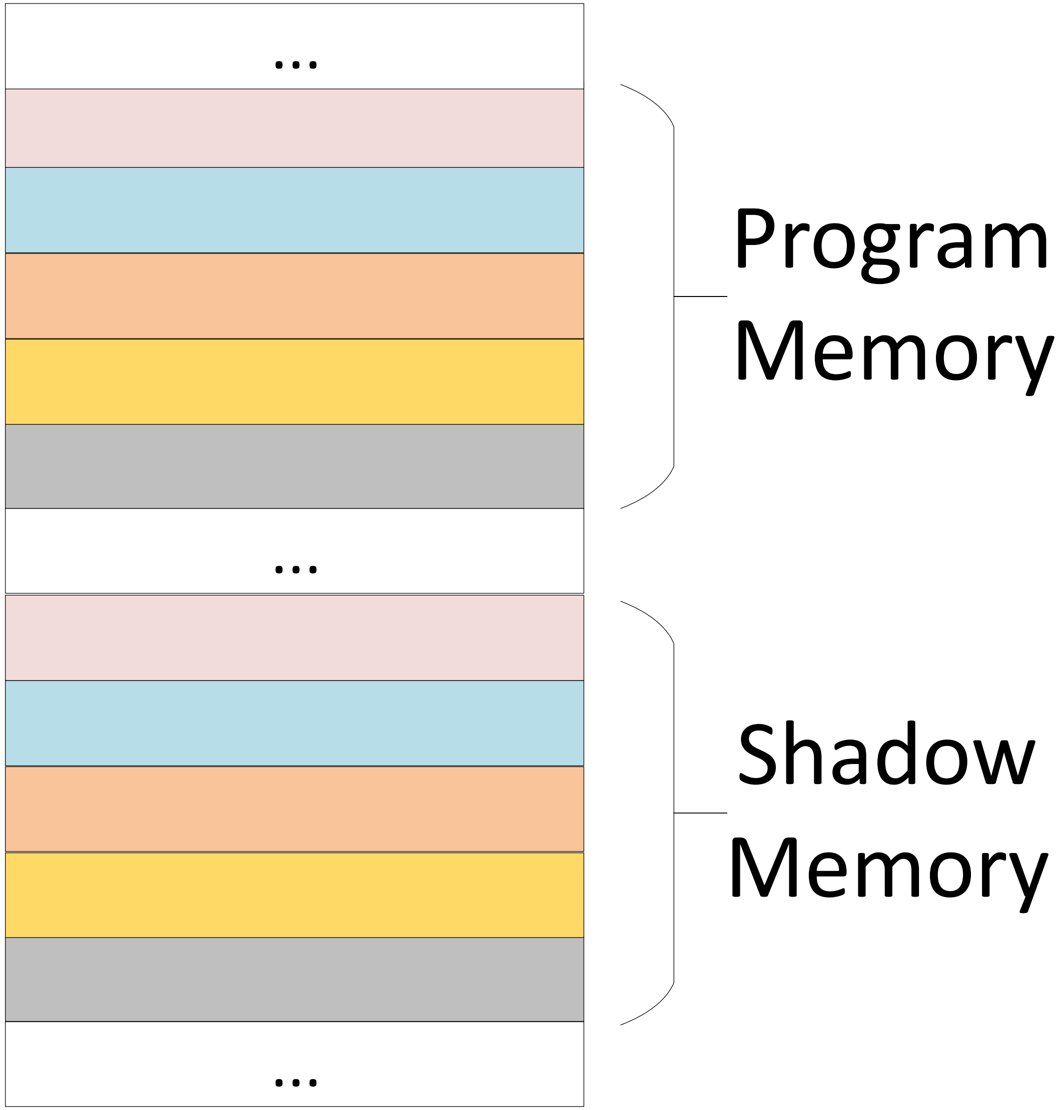
marcin.copik@inf.ethz.ch
mcopik.github.io

Taint Analysis: track parameters propagation

```c
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 != 42)
  z = 6;
```

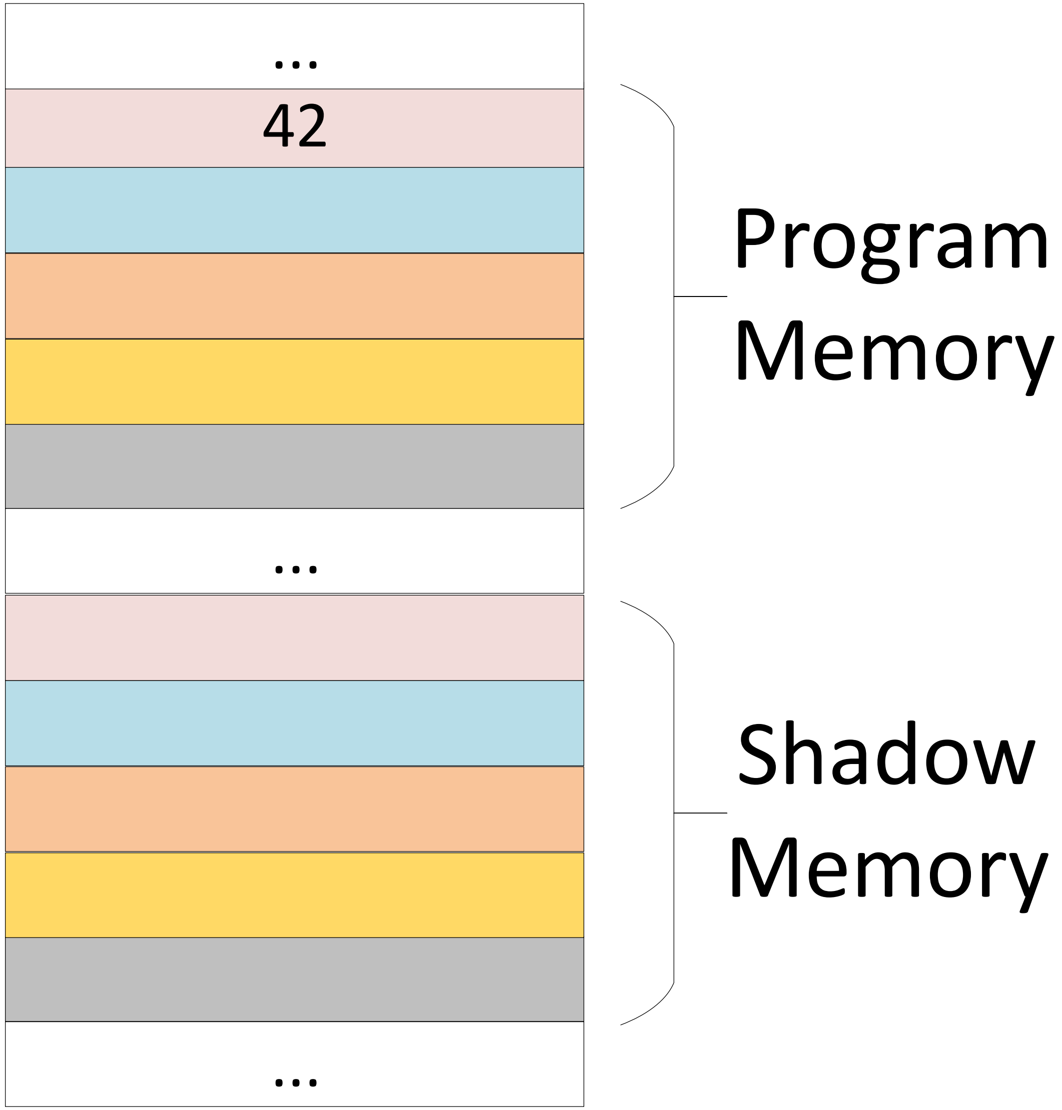


Taint Analysis: track parameters propagation

```
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 != 42)
  z = 6;
```

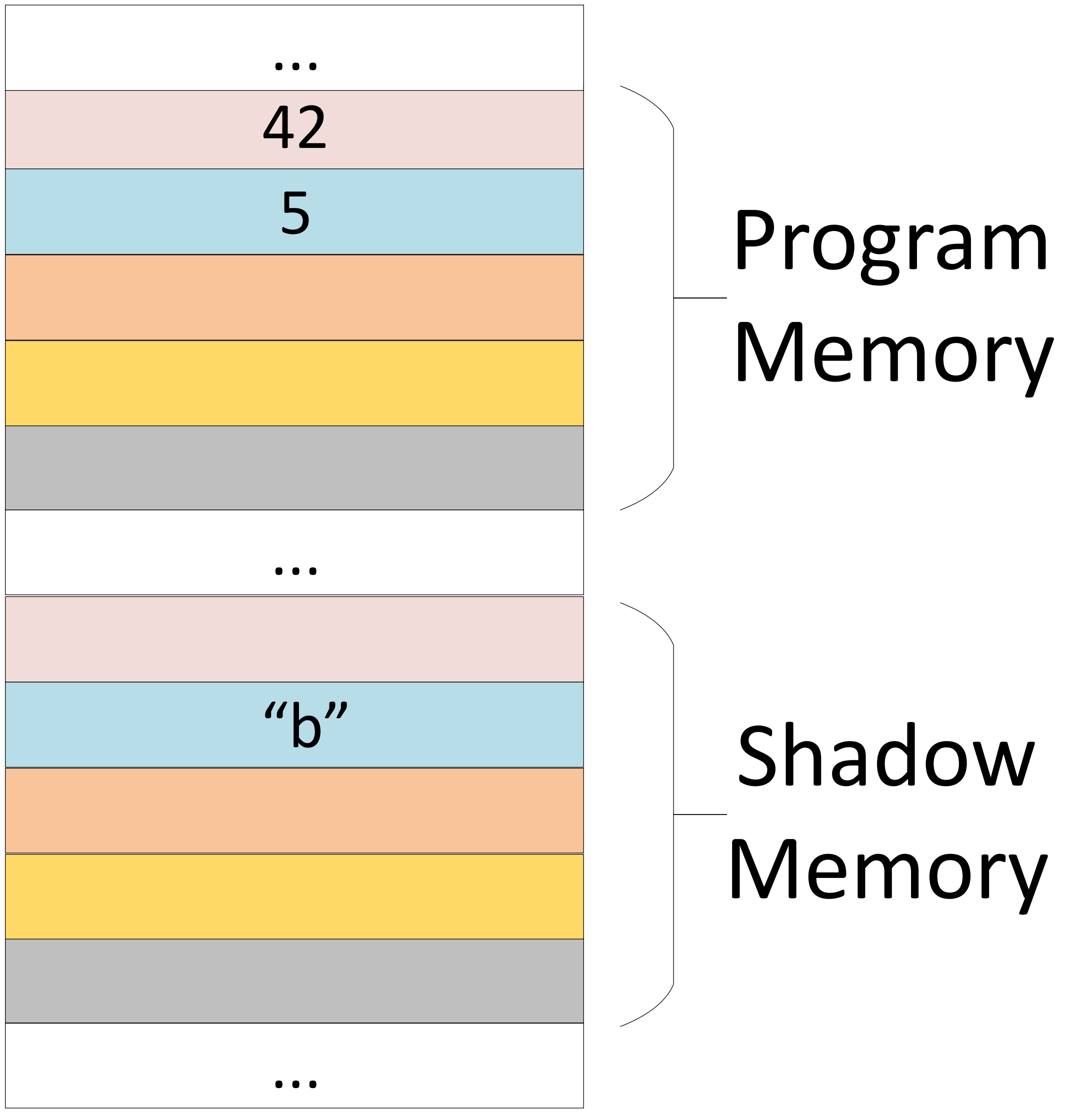


Taint Analysis: track parameters propagation

```
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 != 42)
  z = 6;
```

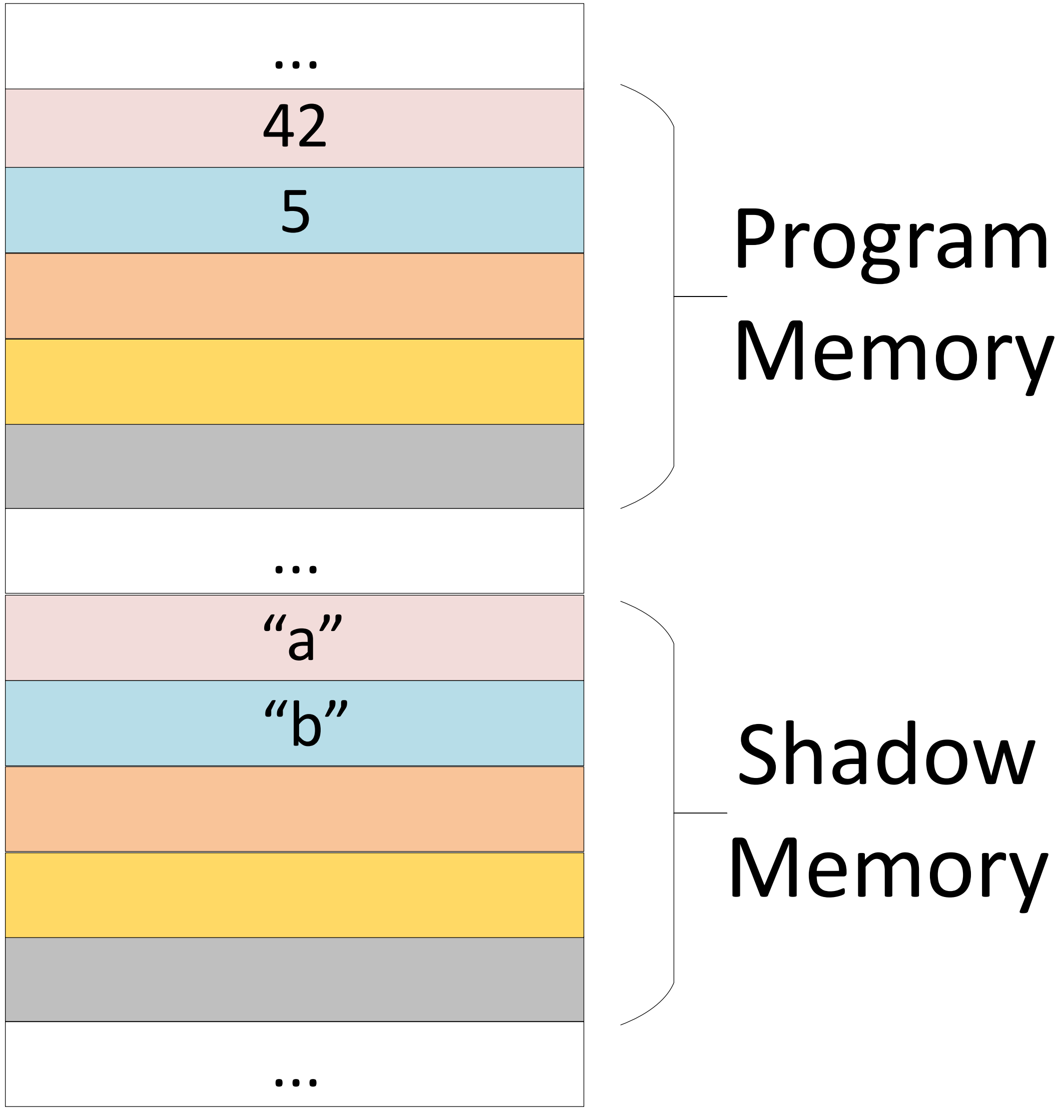


Taint Analysis: track parameters propagation

```
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 != 42)
  z = 6;
```

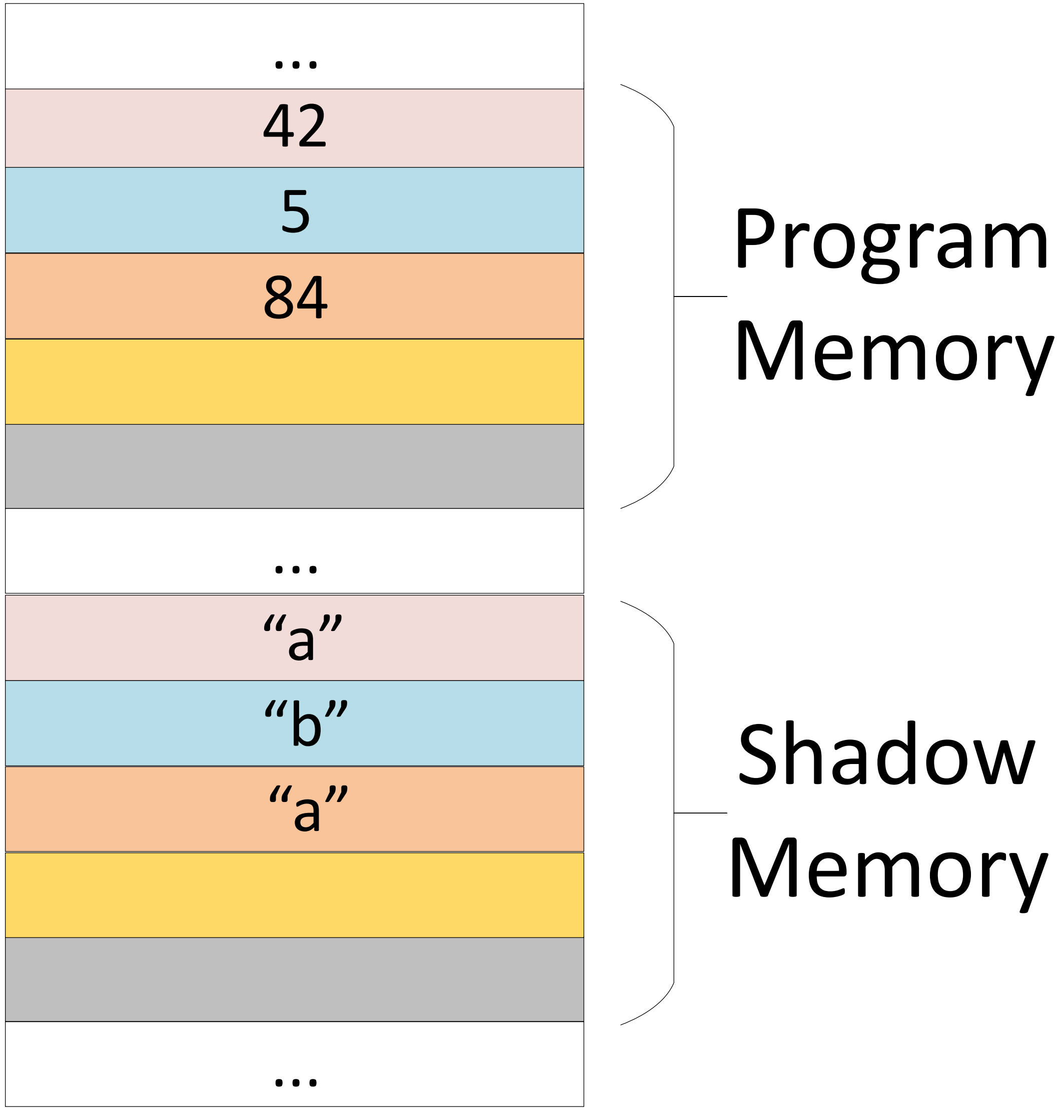


Taint Analysis: track parameters propagation

```
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 != 42)
  z = 6;
```

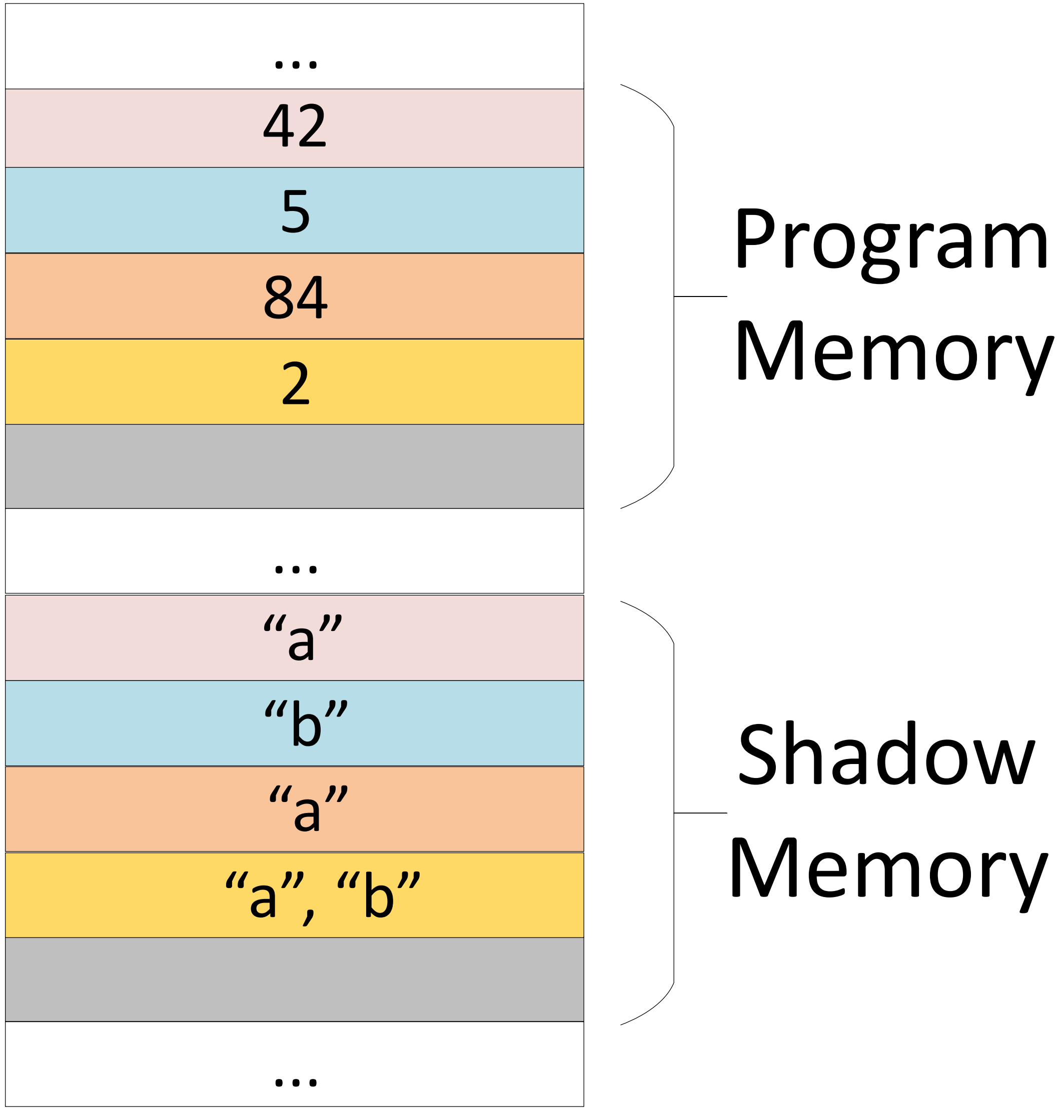


Taint Analysis: track parameters propagation

```
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 != 42)
  z = 6;
```

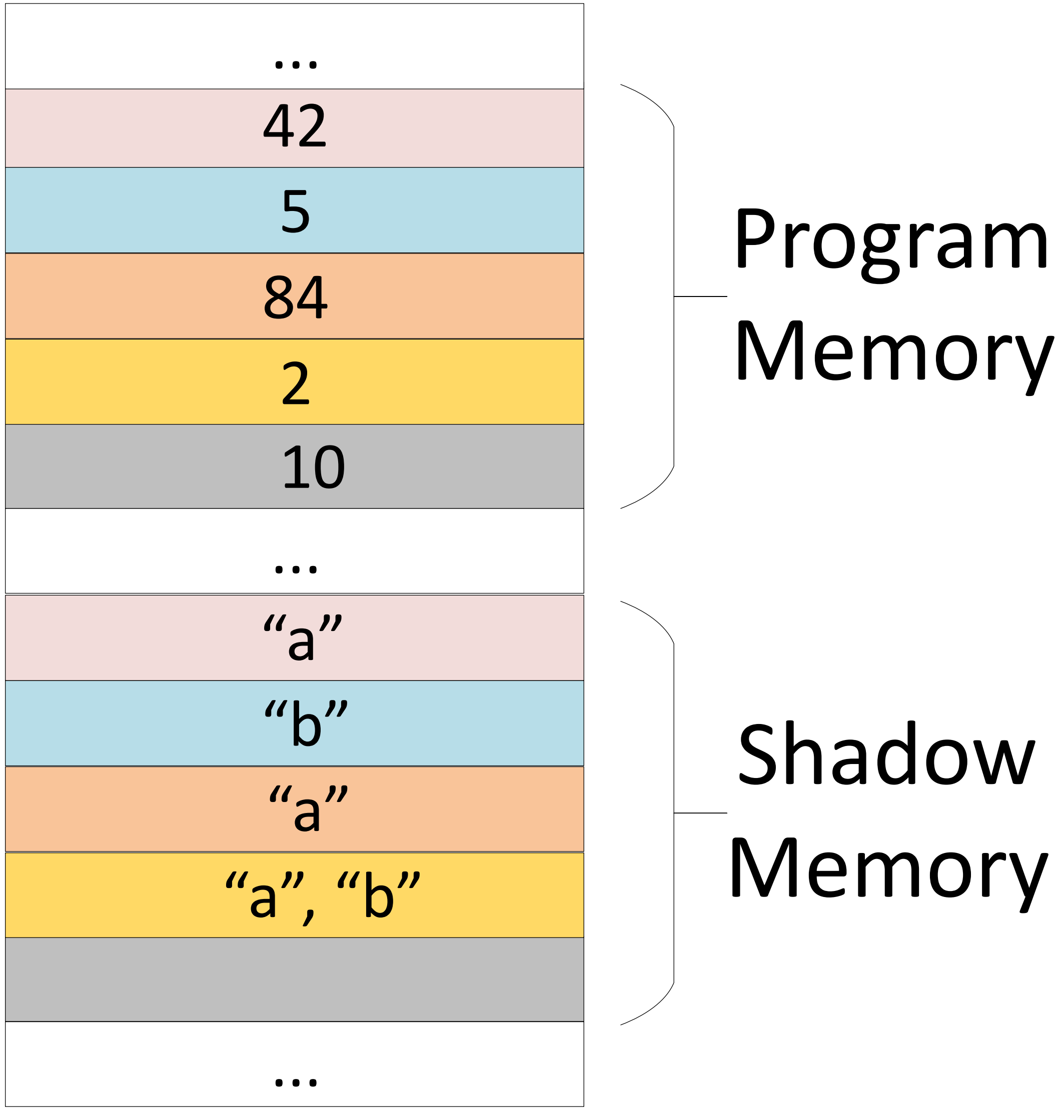


Taint Analysis: track parameters propagation

```
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 != 42)
  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 != 42)
  z = 6;
```