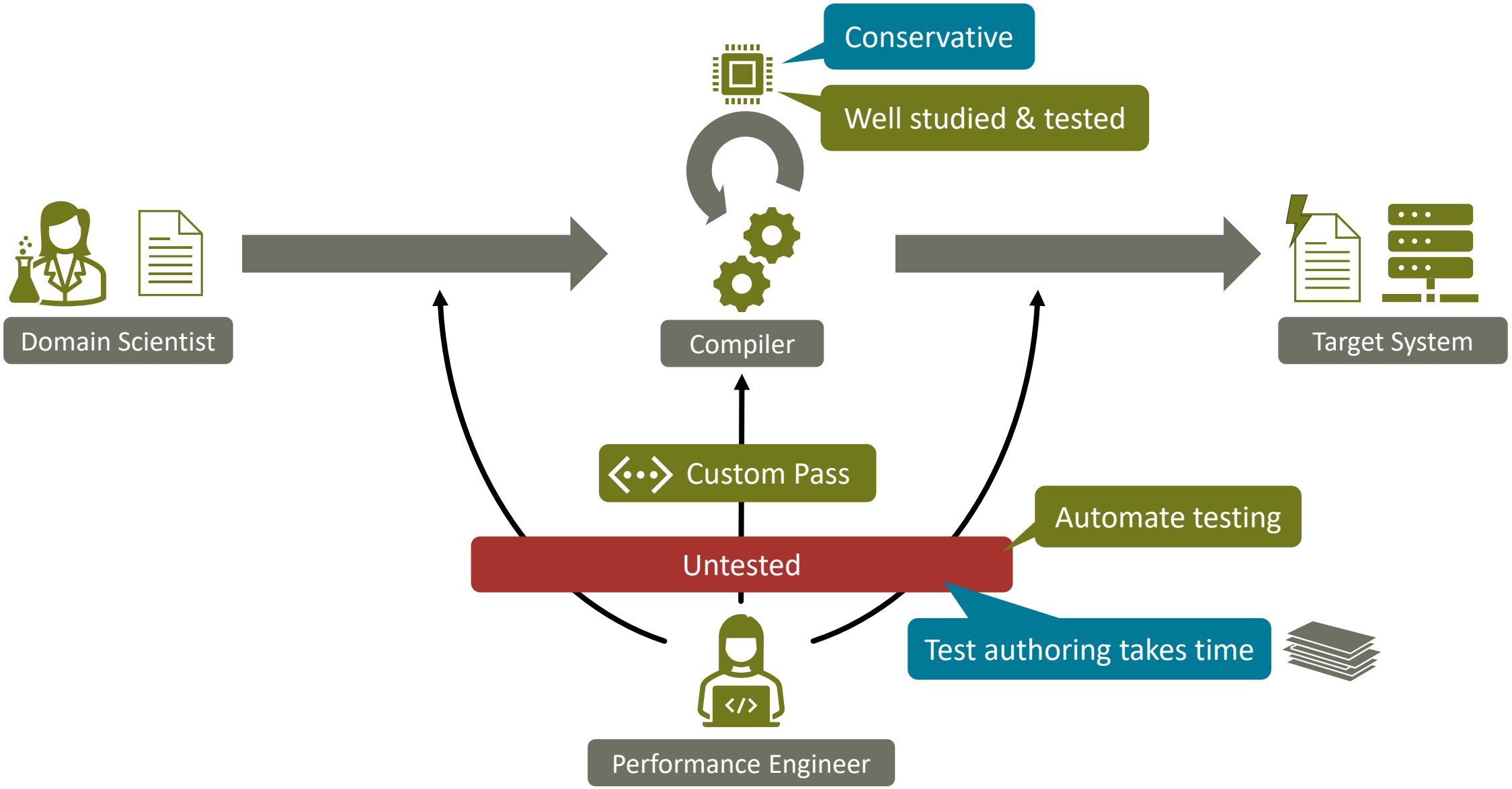
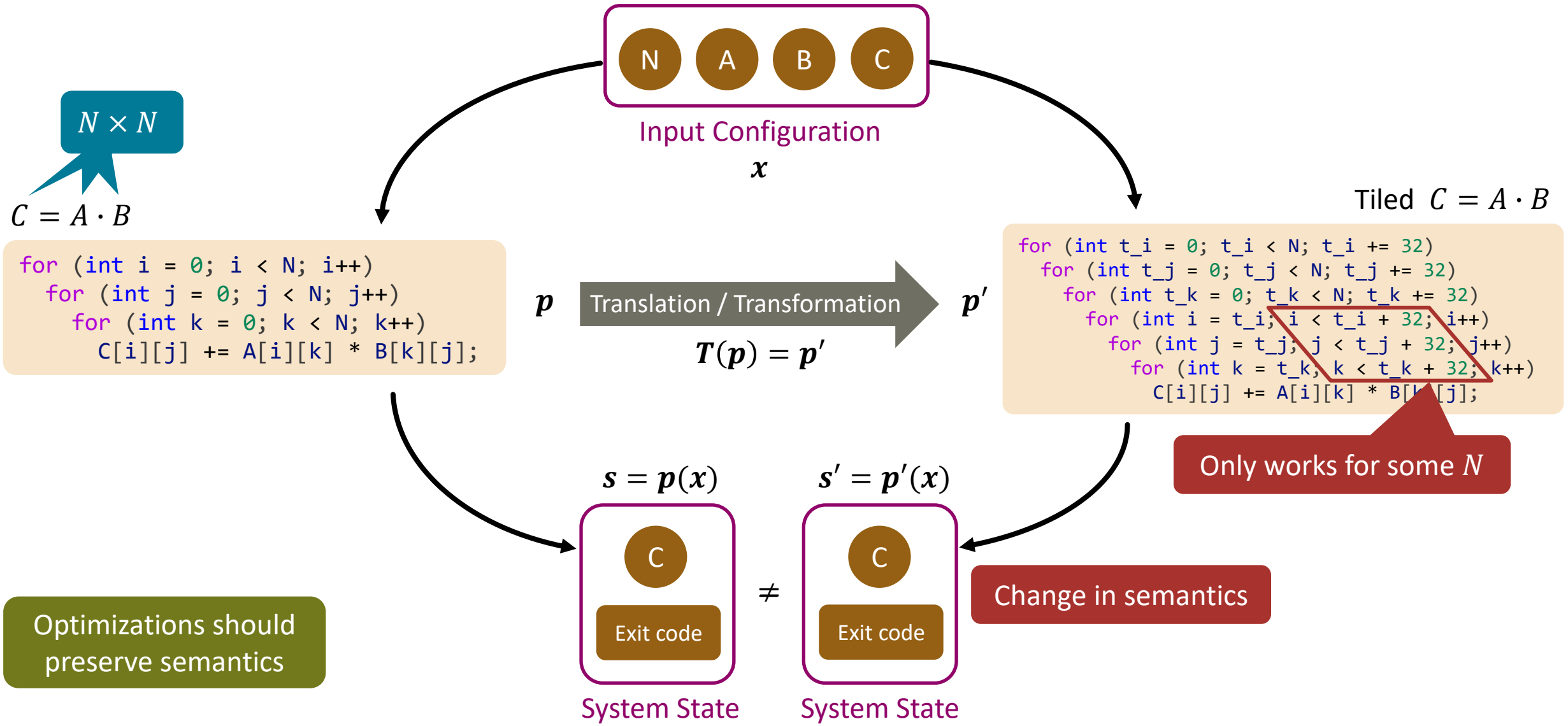
Philipp Schaad, Timo Schneider, Tal Ben-Nun, Alexandru Calotoiu, Alexandros Nikolaos Ziogas, Torsten Hoefler

FuzzyFlow: Leveraging Dataflow To Find and Squash Program Optimization Bugs

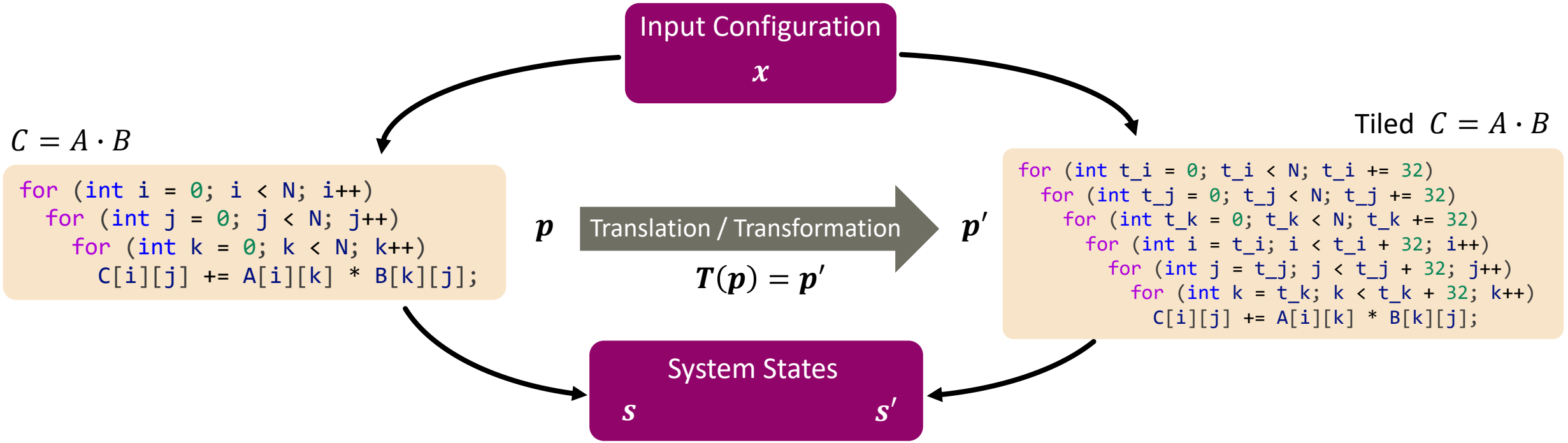
Performance Optimization in HPC



Formalizing Optimizations



Automated Optimization Testing



Formal Validation

Proof of T

- Specification time consuming
- Input-dependent correctness

Differential Testing

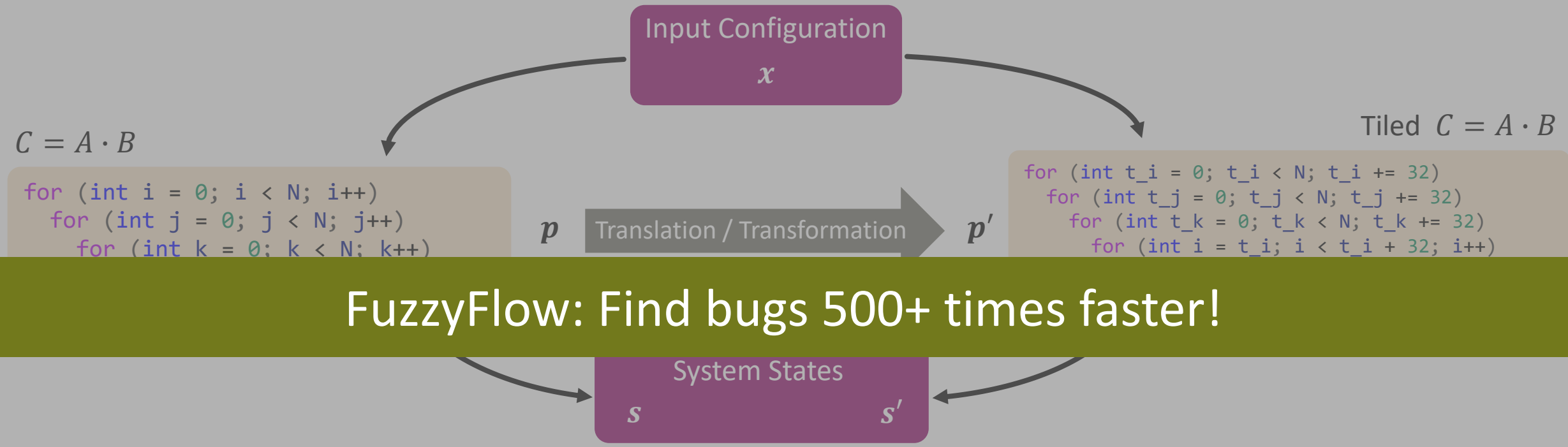
Symbolic execution

- Missing source (e.g.: intrinsics)
- State space explosion
- Floating point arithmetic

Fuzzing

- Probabilistic testing
- Execution takes time

Automated Optimization Testing



FuzzyFlow: Find bugs 500+ times faster!

Formal Validation

Proof of T

- Specification time consuming
- Input-dependent correctness

Differential Testing

Symbolic execution

- Missing source (e.g.: intrinsics)
- State space explosion
- Floating point arithmetic

Fuzzing

- Probabilistic testing
- Execution takes time

How Do We Speed Things Up?

Execution takes time | Probabilistic testing

$$R = ((A \cdot B) \cdot C) \cdot D$$

```
for (int i = 0; i < N; i++)
  for (int j = 0; j < N; j++)
    for (int k = 0; k < N; k++)
      U[i][j] += A[i][k] * B[k][j];
for (int i = 0; i < N; i++)
  for (int j = 0; j < N; j++)
    for (int k = 0; k < N; k++)
      V[i][j] += U[i][k] * C[k][j];
for (int i = 0; i < N; i++)
  for (int j = 0; j < N; j++)
    for (int k = 0; k < N; k++)
      R[i][j] += V[i][k] * D[k][j];
```

p

Loop Tiling

$T(p) = p'$

p'

```
for (int i = 0; i < N; i++)
  for (int j = 0; j < N; j++)
    for (int k = 0; k < N; k++)
      U[i][j] += A[i][k] * B[k][j];
for (int t_i = 0; t_i < N; t_i += 32)
  for (int t_j = 0; t_j < N; t_j += 32)
    for (int t_k = 0; t_k < N; t_k += 32)
      for (int i = t_i; i < t_i + 32; i++)
        for (int j = t_j; j < t_j + 32; j++)
          for (int k = t_k; k < t_k + 32; k++)
            V[i][j] += U[i][k] * C[k][j];
for (int i = 0; i < N; i++)
  for (int j = 0; j < N; j++)
    for (int k = 0; k < N; k++)
      R[i][j] += V[i][k] * D[k][j];
```

Differential Fuzzing

Program Cutout

```
for (int i = 0; i < N; i++)
  for (int j = 0; j < N; j++)
    for (int k = 0; k < N; k++)
      V[i][j] += U[i][k] * C[k][j];
```

$\subseteq p$

c

Loop Tiling

$T(c) = c'$

c'

```
for (int t_i = 0; t_i < N; t_i += 32)
  for (int t_j = 0; t_j < N; t_j += 32)
    for (int t_k = 0; t_k < N; t_k += 32)
      for (int i = t_i; i < t_i + 32; i++)
        for (int j = t_j; j < t_j + 32; j++)
          for (int k = t_k; k < t_k + 32; k++)
            V[i][j] += U[i][k] * C[k][j];
```

Extracting A Program Cutout

Execution takes time | Probabilistic testing

$$R = ((A \cdot B) \cdot C) \cdot D$$

```
for (int i = 0; i < N; i++)
  for (int j = 0; j < N; j++)
    for (int k = 0; k < N; k++)
      U[i][j] += A[i][k] * B[k][j];
for (int i = 0; i < N; i++)
  for (int j = 0; j < N; j++)
    for (int k = 0; k < N; k++)
      V[i][j] += U[i][k] * C[k][j];
for (int i = 0; i < N; i++)
  for (int j = 0; j < N; j++)
    for (int k = 0; k < N; k++)
      R[i][j] += V[i][k] * D[k][j];
```

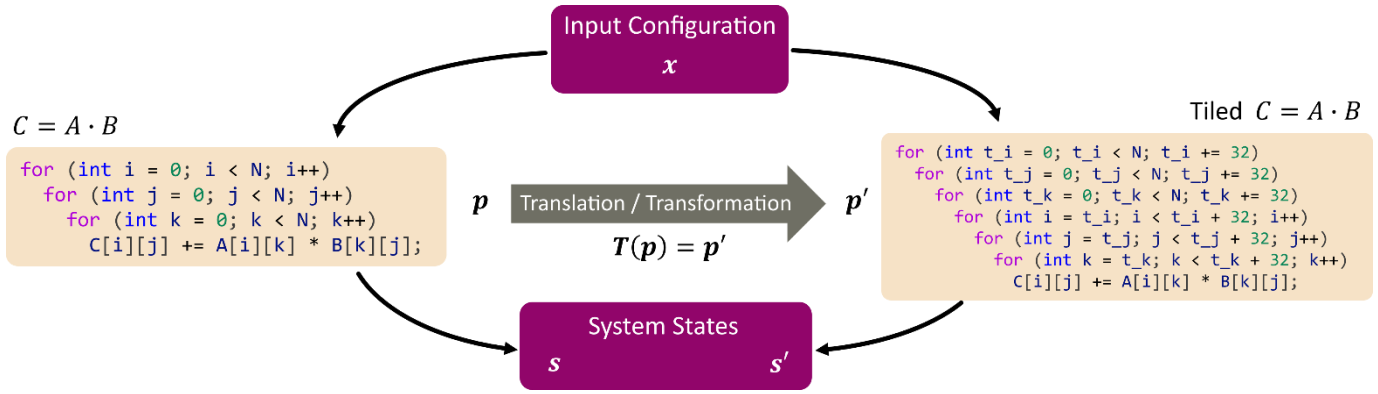
p

?

Program Cutout

```
for (int i = 0; i < N; i++)
  for (int j = 0; j < N; j++)
    for (int k = 0; k < N; k++)
      V[i][j] += U[i][k] * C[k][j];
```

c



System state: { }

Input Configuration: { }

Extracting A Program Cutout

Execution takes time | Probabilistic testing

$$R = ((A \cdot B) \cdot C) \cdot D$$

```
for (int i = 0; i < N; i++)
  for (int j = 0; j < N; j++)
    for (int k = 0; k < N; k++)
      U[i][j] += A[i][k] * B[k][j];
for (int i = 0; i < N; i++)
  for (int j = 0; j < N; j++)
    for (int k = 0; k < N; k++)
      V[i][j] += U[i][k] * C[k][j];
for (int i = 0; i < N; i++)
  for (int j = 0; j < N; j++)
    for (int k = 0; k < N; k++)
      R[i][j] += V[i][k] * D[k][j];
```

c influenced by contents of V, U, C, and N

p

?

Only contents of V influence remainder of p

Program Cutout

```
for (int i = 0; i < N; i++)
  for (int j = 0; j < N; j++)
    for (int k = 0; k < N; k++)
      V[i][j] += U[i][k] * C[k][j];
```

c

Need: Side-Effect Analysis

- Scalar
- Memory
- Sub-Region

Pointer aliasing

Overlapping memory

System state: $\{ V \}$

Input Configuration: $\{ V, U, C, N \}$

Hard to find for arbitrary changes

Extracting A Program Cutout

Execution takes time | Probabilistic testing

$$R = ((A \cdot B) \cdot C) \cdot D$$

```
for (int i = 0; i < N; i++)
  for (int j = 0; j < N; j++)
    for (int k = 0; k < N; k++)
      U[i][j] += A[i][k] * B[k][j];
for (int i = 0; i < N; i++)
  for (int j = 0; j < N; j++)
    for (int k = 0; k < N; k++)
      V[i][j] += U[i][k] * C[k][j];
for (int i = 0; i < N; i++)
  for (int j = 0; j < N; j++)
    for (int k = 0; k < N; k++)
      R[i][j] += V[i][k] * D[k][j];
```

p

Need: Side-Effect Analysis

- Scalar
- Memory
- Sub-Region

Pointer aliasing

Overlapping memory

Need: Generalization

- Inputs
- Sizes

Program Cutout

```
for (int i = 0; i < N; i++)
  for (int j = 0; j < N; j++)
    for (int k = 0; k < N; k++)
      V[i][j] += U[i][k] * C[k][j];
```

Input dependent correctness

c → Loop Tiling → c'

$$T(c) = c'$$

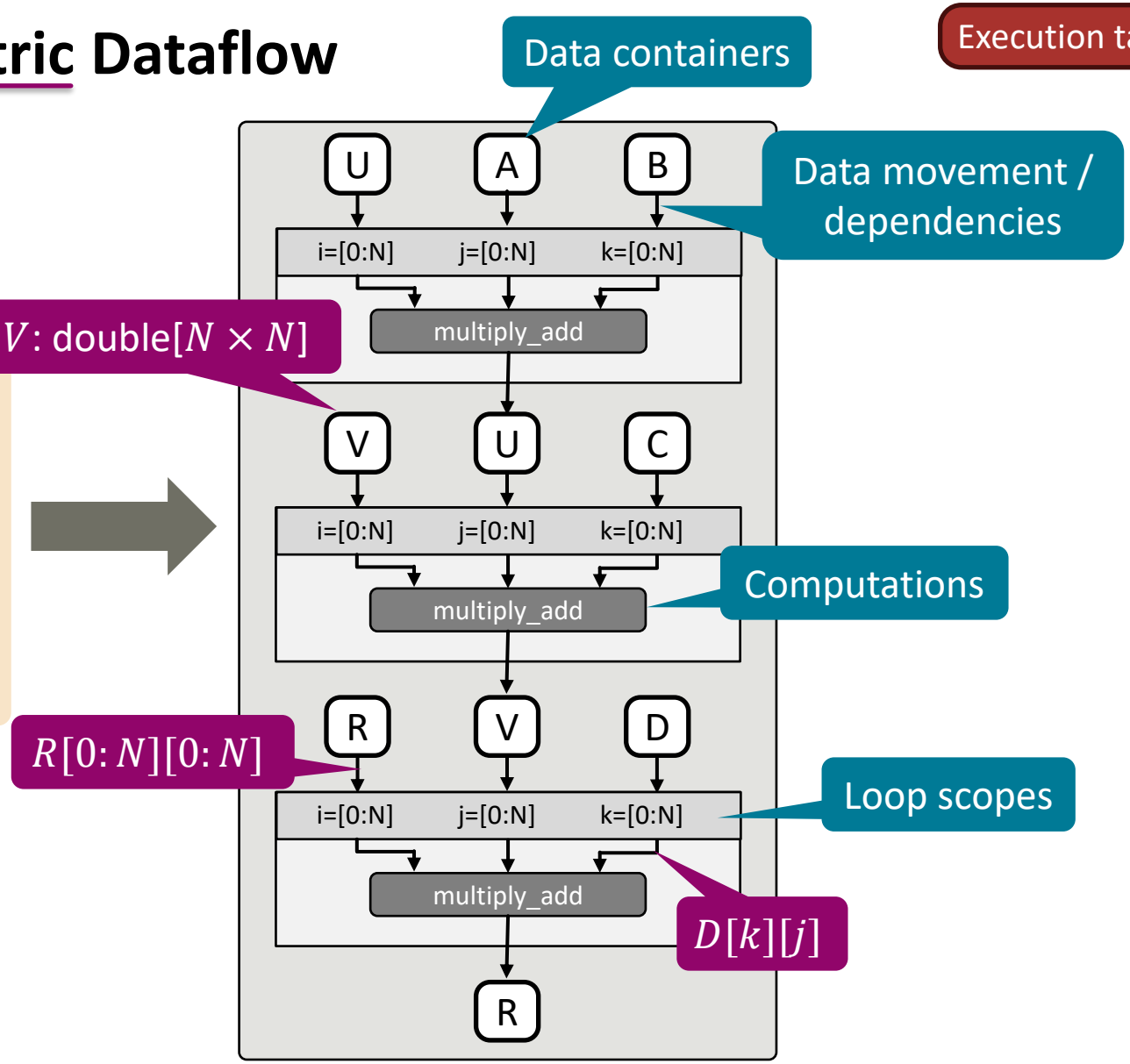
No relation between N and sizes

```
for (int t_i = 0; t_i < N; t_i += 32)
  for (int t_j = 0; t_j < N; t_j += 32)
    for (int t_k = 0; t_k < N; t_k += 32)
      for (int i = t_i; i < t_i + 32; i++)
        for (int j = t_j; j < t_j + 32; j++)
          for (int k = t_k; k < t_k + 32; k++)
            V[i][j] += U[i][k] * C[k][j];
```

Leveraging Parametric Dataflow

Execution takes time | Probabilistic testing

```
for (int i = 0; i < N; i++)
  for (int j = 0; j < N; j++)
    for (int k = 0; k < N; k++)
      U[i][j] += A[i][k] * B[k][j];
for (int i = 0; i < N; i++)
  for (int j = 0; j < N; j++)
    for (int k = 0; k < N; k++)
      V[i][j] += U[i][k] * C[k][j];
for (int i = 0; i < N; i++)
  for (int j = 0; j < N; j++)
    for (int k = 0; k < N; k++)
      R[i][j] += V[i][k] * D[k][j];
```

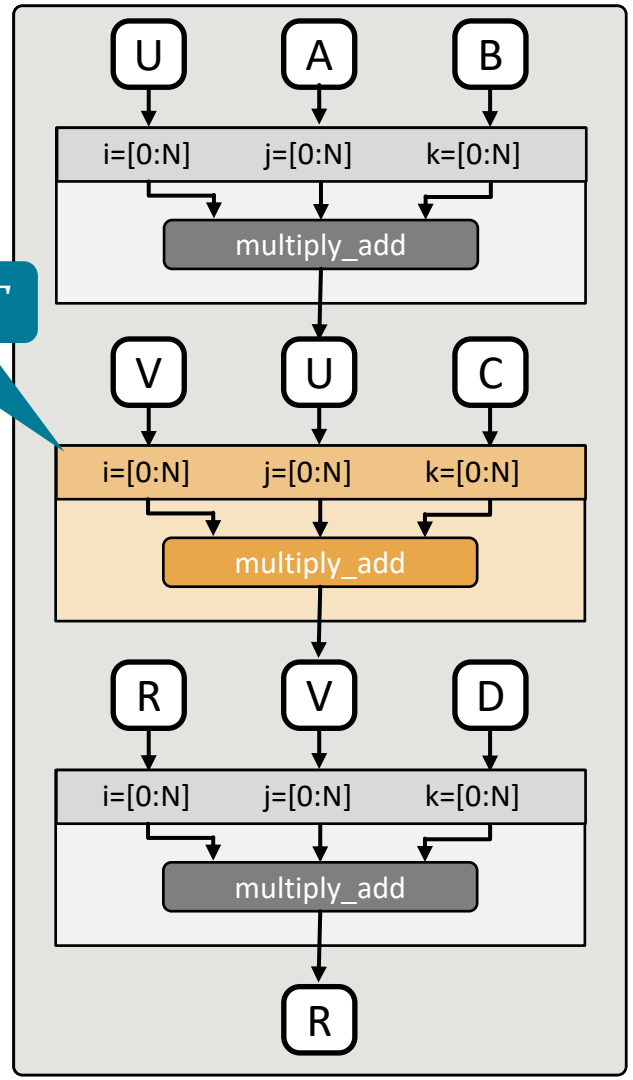


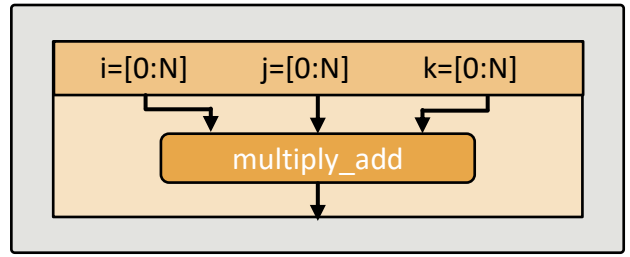
Leveraging Parametric Dataflow

Execution takes time | Probabilistic testing

```
for (int i = 0; i < N; i++)
  for (int j = 0; j < N; j++)
    for (int k = 0; k < N; k++)
      U[i][j] += A[i][k] * B[k][j];
for (int i = 0; i < N; i++)
  for (int j = 0; j < N; j++)
    for (int k = 0; k < N; k++)
      V[i][j] += U[i][k] * C[k][j];
for (int i = 0; i < N; i++)
  for (int j = 0; j < N; j++)
    for (int k = 0; k < N; k++)
      R[i][j] += V[i][k] * D[k][j];
```

Modified by T

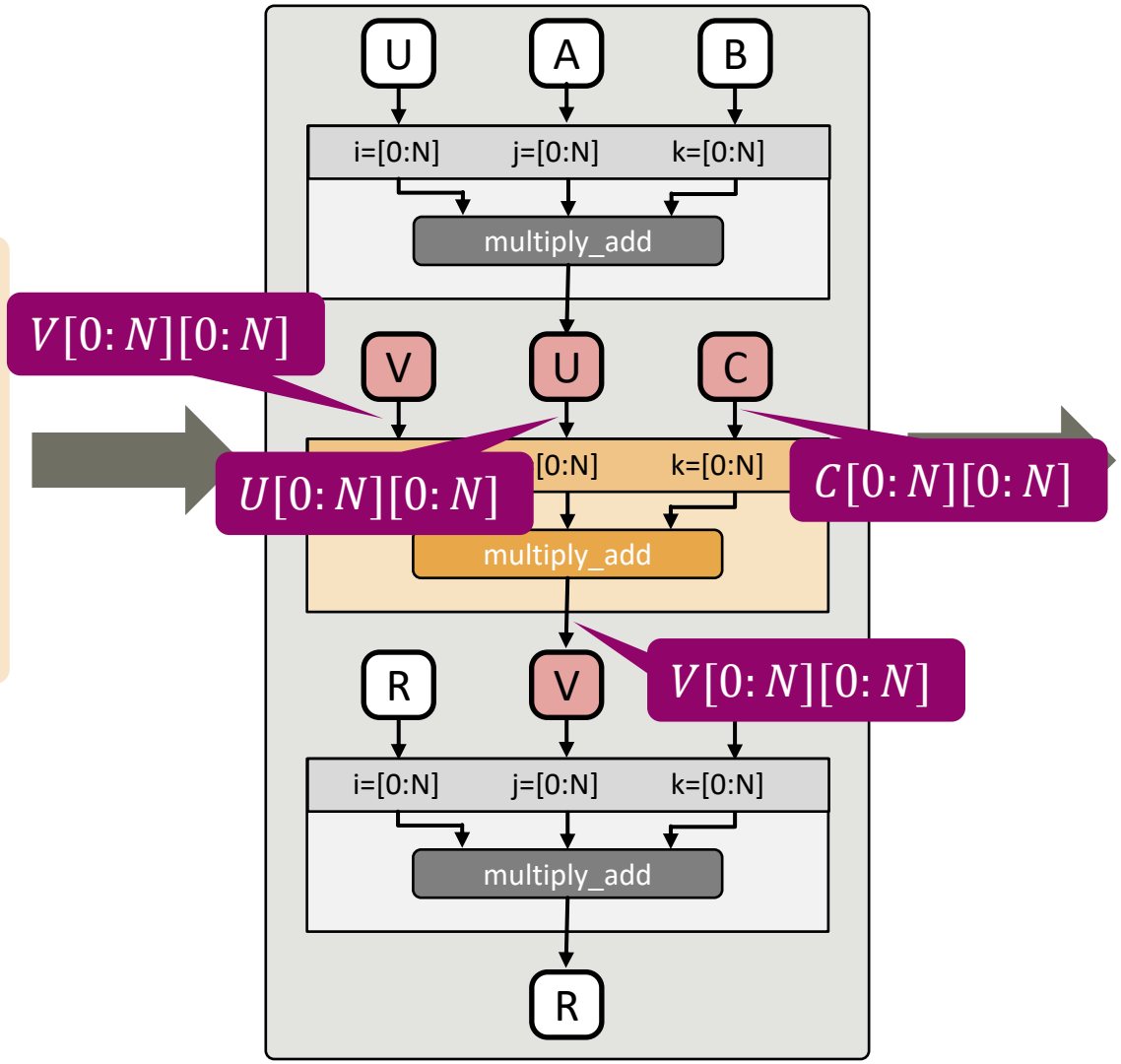


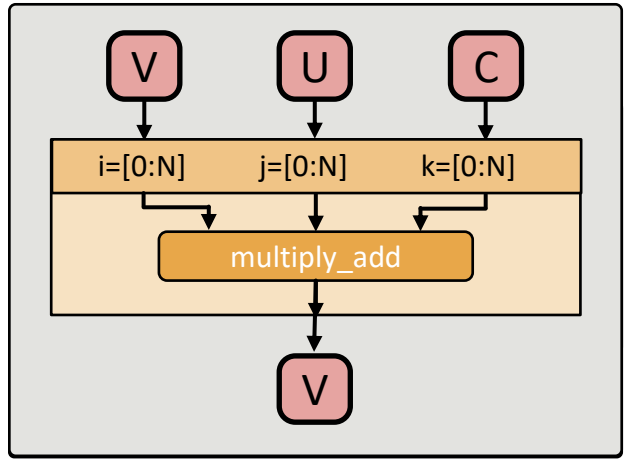


Leveraging Parametric Dataflow

Execution takes time | Probabilistic testing

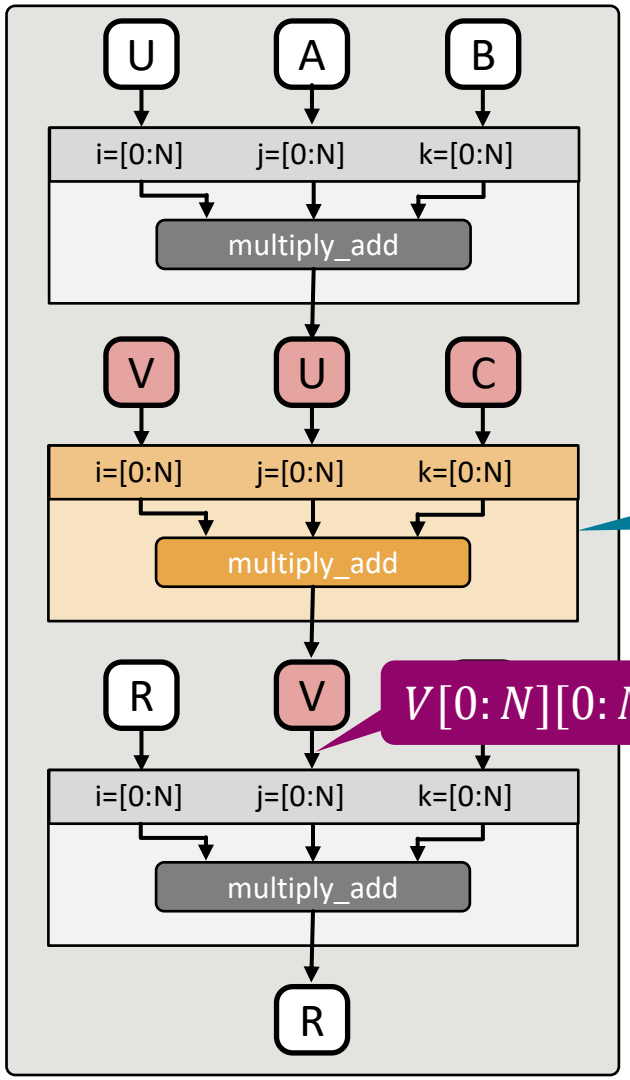
```
for (int i = 0; i < N; i++)
  for (int j = 0; j < N; j++)
    for (int k = 0; k < N; k++)
      U[i][j] += A[i][k] * B[k][j];
for (int i = 0; i < N; i++)
  for (int j = 0; j < N; j++)
    for (int k = 0; k < N; k++)
      V[i][j] += U[i][k] * C[k][j];
for (int i = 0; i < N; i++)
  for (int j = 0; j < N; j++)
    for (int k = 0; k < N; k++)
      R[i][j] += V[i][k] * D[k][j];
```





Leveraging Parametric Dataflow

Execution takes time | Probabilistic testing



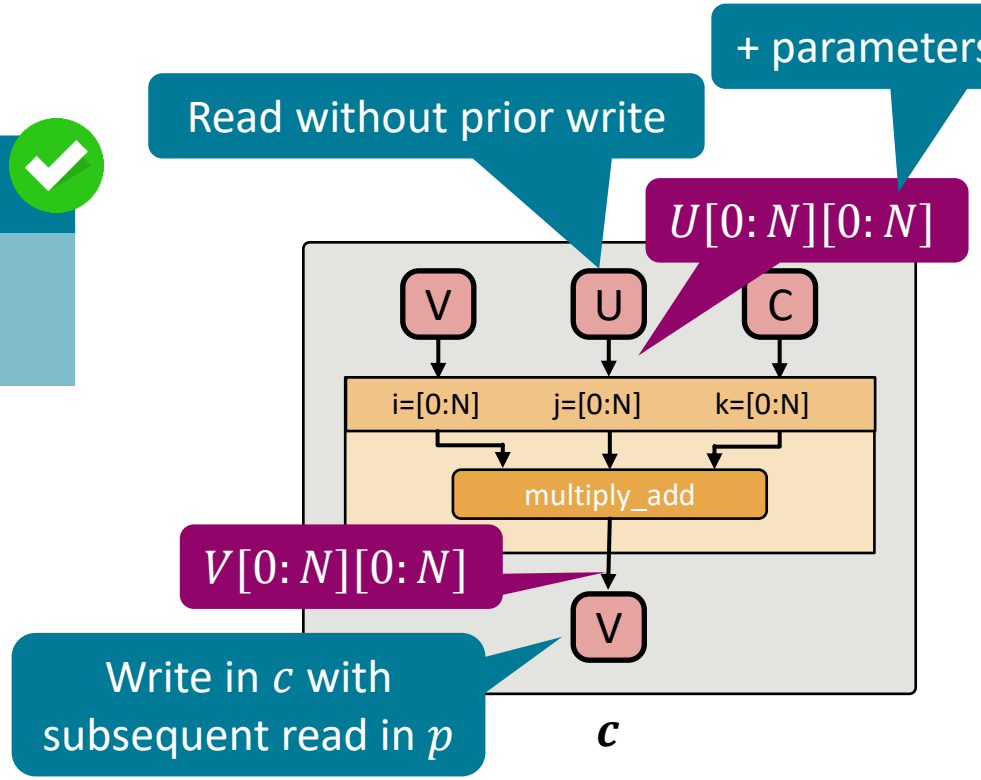
Graph traversal starting at c

Need: Generalization

- Inputs
- Sizes

Need: Side-Effect Analysis

- Scalar
- Memory
- Sub-Region



Input configuration: { V, U, C }

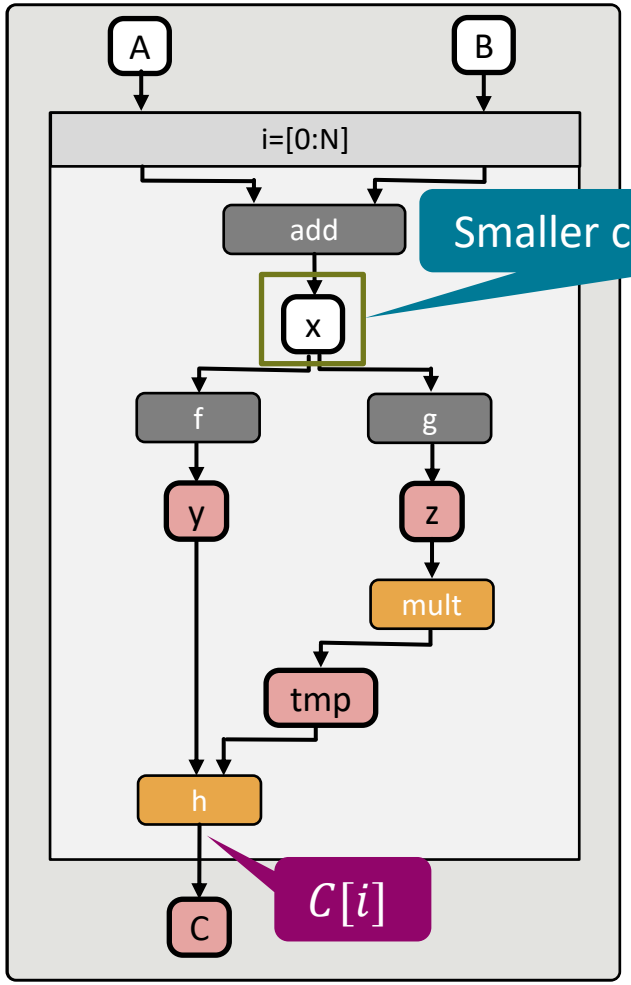
System state: { V }

Reducing Input Configurations

Execution takes time | Probabilistic testing

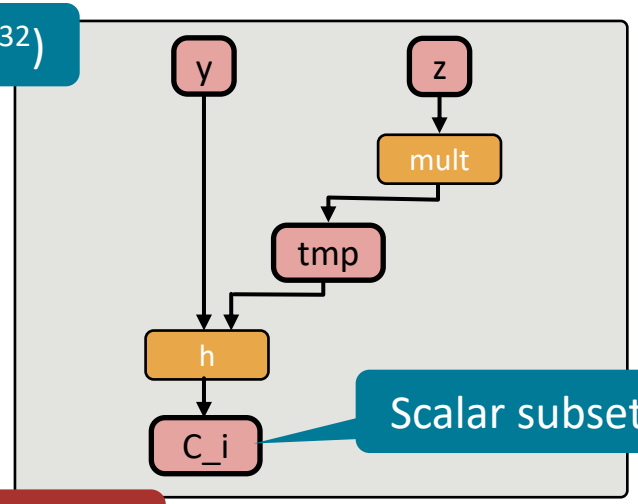
```
for (int i = 0; i < N; i++) {
  int x = A[i] + B[i];
  int y = f(x);
  int z = g(x);
  int tmp = z * 2;
  C[i] = h(y, tmp);
}
```

```
for (int i = 0; i < N; i++) {
  int x = A[i] + B[i];
  int y = f(x);
  int z = g(x);
  C[i] = h(y, z * 2);
}
```



p

Smaller configuration (2^{32})



Scalar subset

c

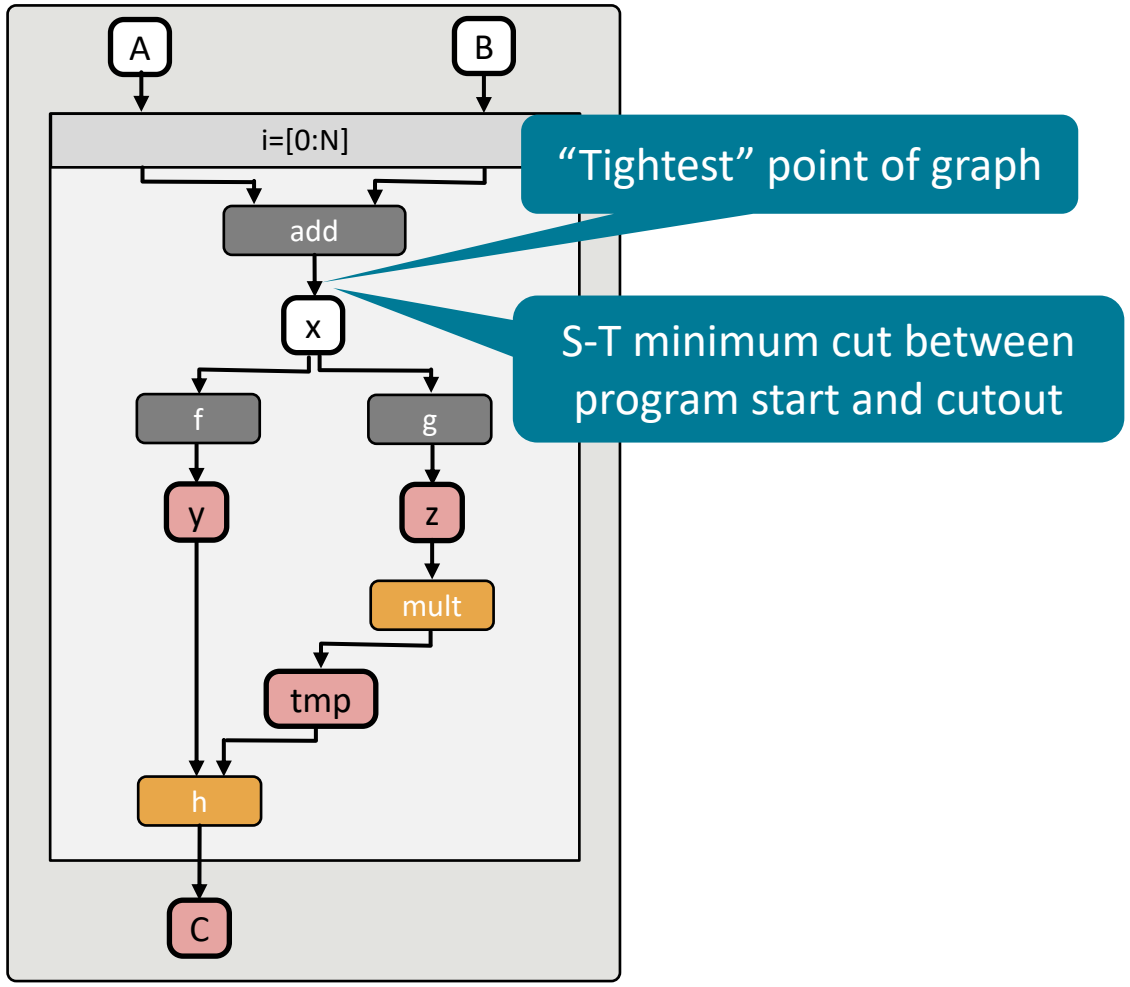
2^{64} possible inputs

Input configuration: { y, z }

System state: { C_i }

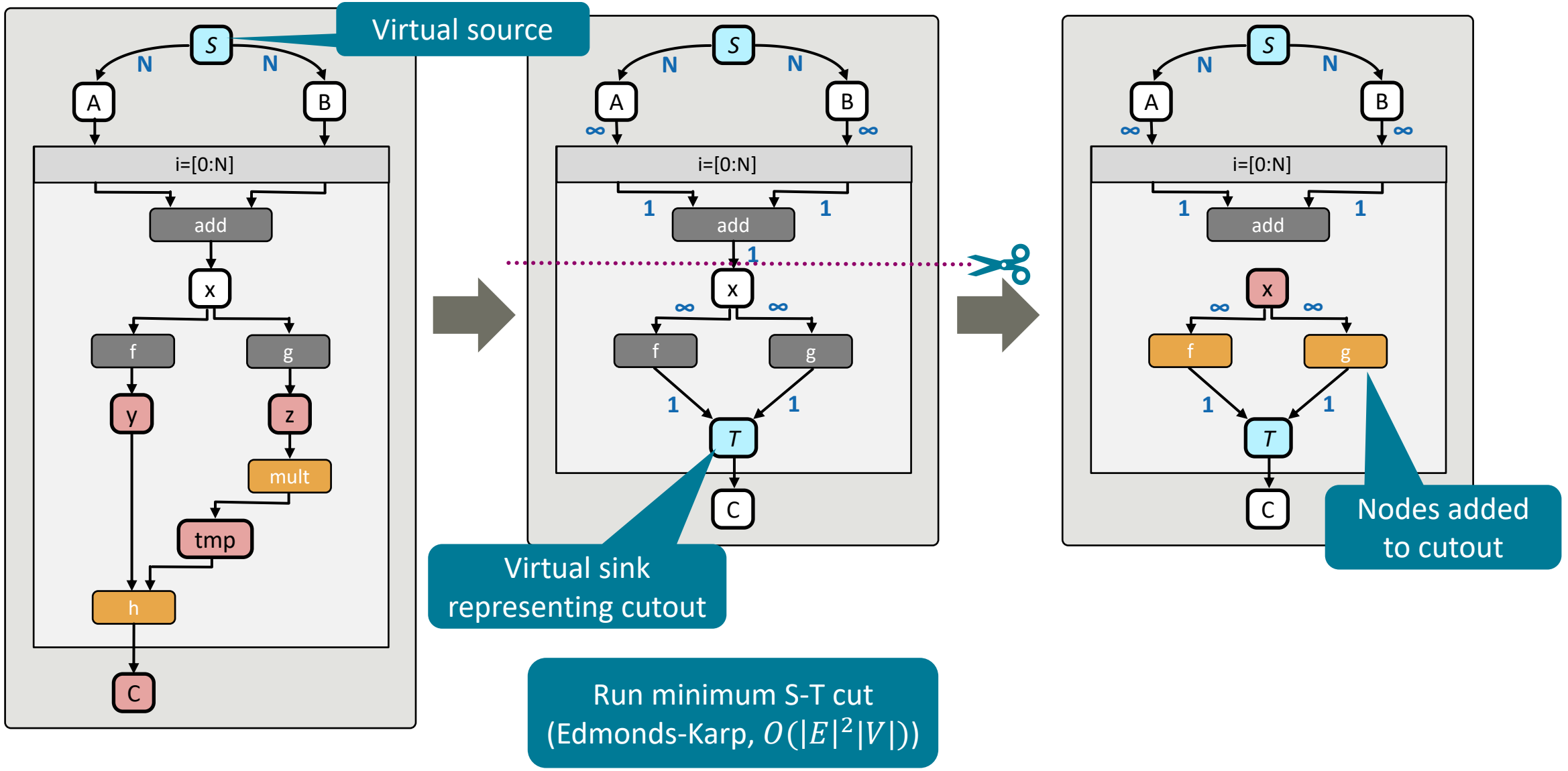
Reducing Input Configurations

Execution takes time | Probabilistic testing



Reducing Input Configurations

Execution takes time | Probabilistic testing



Constraining Inputs

Execution takes time | Probabilistic testing

p

```
void foo(double **C, int i, int j) {
  int tmp = C[j][i] * 2;
  C[i][j] = tmp * 2;
}

void main(double **C, int i) {
  for (int j = 0; j < 20; j++)
    foo(C, i, j);
}
```

$T(p) = p'$

```
void foo(double **C, int i, int j) {
  C[i][j] = C[j][i] * 4;
}

void main(double **C, int i) {
  for (int j = 0; j < 20; j++)
    foo(C, i, j);
}
```

p'

j is further constrained

c

```
int tmp = C[j][i] * 2;
C[i][j] = tmp * 2;
```

C: double[50 × 50]

Input configuration: { C, i, j }

System state: { C }

Used to index data container C

Mostly “uninteresting” crashes

Constrain parameters used to access data

$i \in [0,49],\ j \in [0,19]$

Derive control-flow constraints

User specified constraints

Improved Coverage

Execution takes time | Probabilistic testing

Minimal size of input space

Constrained input choices

Maximized cutout coverage

Most likely never executed

```
if (x == 0)
  foo();
else
  bar();
```

Dataflow Cutout

Generate C++

Sample input configuration

AFL++ [1]

Coverage-guided fuzzing

Instrument code, maximize coverage

[1] Fioraldi et al., AFL++ : Combining Incremental Steps of Fuzzing Research, WOOT'20

FuzzyFlow Process

Given program p and transformation T

1. Obtain dataflow cutout c from p
2. Apply transformation T to cutout c
3. Sample or mutate input configuration x
4. Run c and $c' = T(c)$ on configuration
5. Compare system states $s = c(x)$ and $s' = c'(x)$
6. Counterexample

Uniform sampling over constraints

Coverage-guided mutation

Capture crashes

Results equivalent? Exit codes match?

Repeat r times

Equivalence:
Numerical difference below threshold t_Δ.
$t_\Delta = 0$ → checks for bit-wise equality.

FuzzyFlow – A Proof-Of-Concept

DaCe [1]

Productive Performance Engineering for Weather and Climate Modeling with Python

Lifting C Semantics for Dataflow Optimization

Deinsum: Practically I/O Optimal Multi-Linear Algebra

Alexandros Nikolaos Ziogas
Department of Computer Science
ETH Zurich
Zurich, Switzerland
alexandros.ziogas@inf.ethz.ch

Grzegorz Kwasniewski*
NextSilicon
Tel Aviv, Israel
grzegorz.kwasniewski@nextsilicon.com

Timo Schneider
Department of Computer Science
ETH Zurich
Zurich, Switzerland
timo.schneider@inf.ethz.ch

DATA MOVEMENT IS ALL YOU NEED: A CASE STUDY ON OPTIMIZING TRANSFORMERS

A Case Study on DaCe Portability & Performance for Batched Discrete Fourier Transforms

Måns I. Andersson
Stefano Markidis

A Data-Centric Approach to Extreme-Scale *Ab initio* Dissipative Quantum Transport Simulations

Alexandros Nikolaos Ziogas*, Tal Ben-Nun*, Guillermo Indalecio Fernández†, Timo Schneider*, Mathieu Luisier†, and Torsten Hoefler*
*Scalable Parallel Computing Laboratory, ETH Zurich, Switzerland
†Integrated Systems Laboratory, ETH Zurich, Switzerland

ABSTRACT

The computational efficiency of a state of the art *ab initio* quantum transport (QT) solver, capable of revealing the coupled electro-thermal properties of atomically-resolved nano-transistors, has been improved by up to two orders of magnitude through a data cen-

2 PERFORMANCE ATTRIBUTES

Performance attribute	Our submission
Category of achievement	Scalability, time-to-solution
Type of method used	Non-linear system of equations

[1] Ben-Nun et al., Stateful Dataflow Multigraphs: A Data-Centric Model for Performance Portability on Heterogeneous Architectures, SC'19

Bug Hunt

NPBench

52 benchmark applications

61 optimizations and passes tested

3,280 test cases extracted

6 DaCe transformations containing bugs found

Changes in semantics (2)

Lead to invalid code (4)

Reducing Input Spaces

BERT Encoder layer

```
// ...
mkl_batched_gemm(X, Y, tmp);
// ...
// ...
// ...
#pragma omp parallel for
for (int i = 0; i < H; i++)
  for (int j = 0; j < B; j++)
    for (int k = 0; k < SM; k++)
      for (int l = 0; l < SM; l++)
        beta[i][j][k][l] = \
          tmp[i][j][k][l] * scale;
// ...
```

Vectorize parallel loops

```
mkl_batched_gemm(X, Y, tmp);
#pragma omp parallel for
for (int i = 0; i < H; i++)
  for (int j = 0; j < B; j++)
    for (int k = 0; k < SM; k++)
      for (int l = 0; l < SM; l++)
        beta[i][j][k][l] = \
          tmp[i][j][k][l] * scale;
```

528x faster than running whole application (12.1 seconds)

43.7 trials per second on consumer hardware

Input space reduced by 75%

Using $BERT_{LARGE}$

Input configuration: { $X, Y, scale$ }

System state: { $beta$ }

GPU Kernel Extraction

Took engineer 16h to find and debug

Attempt to find bug using FuzzyFlow

ECMWF Cloud Microphysics Scheme (CLOUDSC)

```
// ...
void loop_function() {
  for (int j = 0; j < NPROMA; j++)
    ZPFPLSX[j:j+NPROMA] = comp(ZFALLSINK, ZQXN, ZRDTGDP[j]);
}
// ...
```

Only writes to subset of ZPFPLSX

GPU Kernel Extraction

CUDA copy, allocation, …

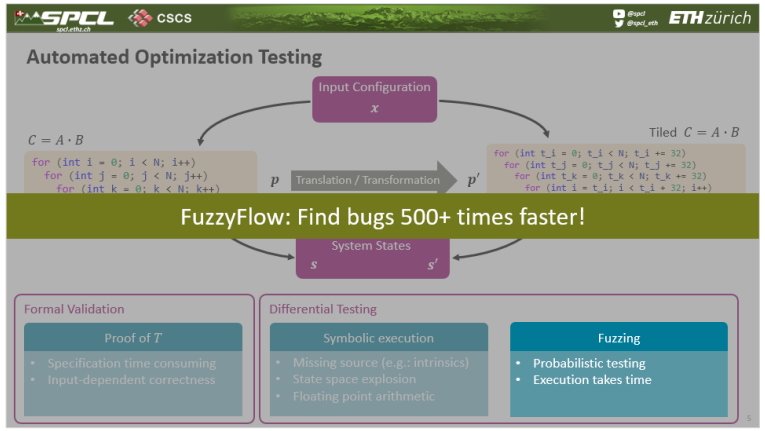
43 seconds on consumer hardware

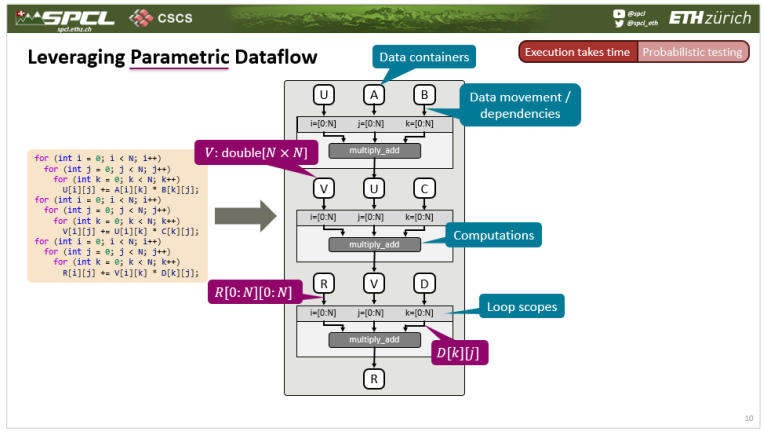
```
// ...
[ CUDA boilerplate ]
cudaLaunchKernel(loop_function, ...);
[ CUDA boilerplate ]
// ...
```

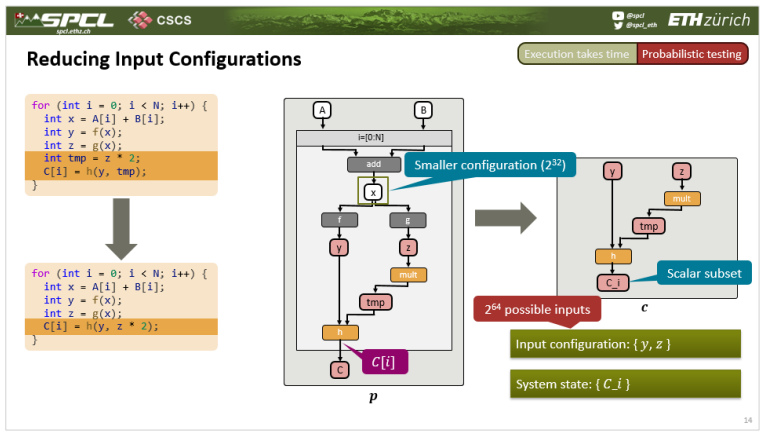
Kernel launch

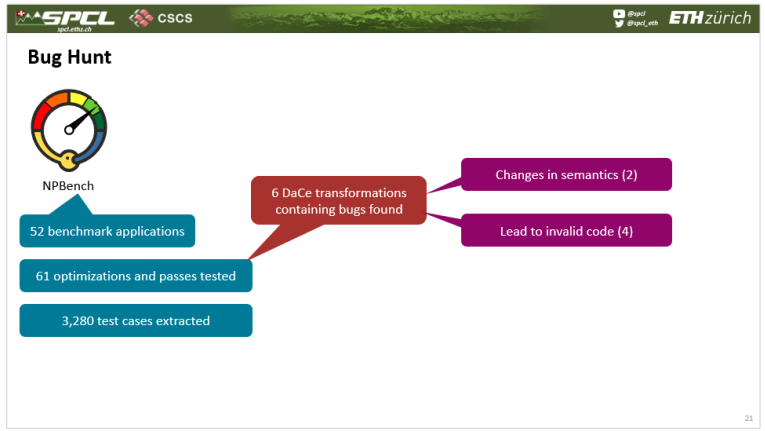
All of ZPFPLSX copied back!

Conclusions









More of SPCL's research:

youtube.com/@spcl — 180+ Talks

twitter.com/spcl_eth — 1.4K+ Followers

github.com/spcl — 3.8K+ Stars

… or spcl.ethz.ch

GitHub

Localized Optimization Testing Requirements

Requirements → Representation ↓	Side Effect Analysis			Generalization	
	Scalar	Memory	Sub-Region	Inputs	Sizes
Abstract Syntax Tree	X	X	X	X	X
SSA-Form	✓	X	X	X	X
Program Dependence Graph	✓	✓	X	X	X
MLIR	✓	✓	✓[1]	✓	X
Parametric Dataflow	✓	✓	✓	✓	✓

[1] Constant sizes only.

From Multi-Node to Single-Node

Distributed Vanilla Graph Attention, SDDMM

```
// ...
MPI_Bcast(H2, LAcols * LHcols, MPI_DOUBLE, 0, comm);
// ...
#pragma omp parallel for
for (int i =  0; i < LAnnz; i++) {
  for (int k = 0; k < LHcols; k++) {
    int H1_idx = LWcols * A_rowidx[i] + k;
    int H2_idx = LWcols * A_colidx[i] + k;
    double tmp = H1[H1_idx] * H2[H2_idx];
    values[i] = tmp + values[i];
  }
}
// ...
MPI_Allreduce(MPI_IN_PLACE, output, LArows * LWcols,
              MPI_DOUBLE, MPI_SUM, comm);
// ...
```

Drops need for communication

SDDM Optimization
(e.g., remove intermediates)

```
int H1_idx = LWcols * A_rowidx[i] + k;
int H2_idx = LWcols * A_colidx[i] + k;
double tmp = H1[H1_idx] * H2[H2_idx];
values[i] = tmp + values[i];
```

Input configuration: $\{ i, k, values, H1, H2, LWcols, A\_rowidx, A\_colidx \}$

System state: $\{ values_i \}$