DiffTrace: Efficient Whole-Program Trace Analysis and Diffing for Debugging

Saeed Taheri, Ian Briggs, Martin Burtscher, Ganesh Gopalakrishnan

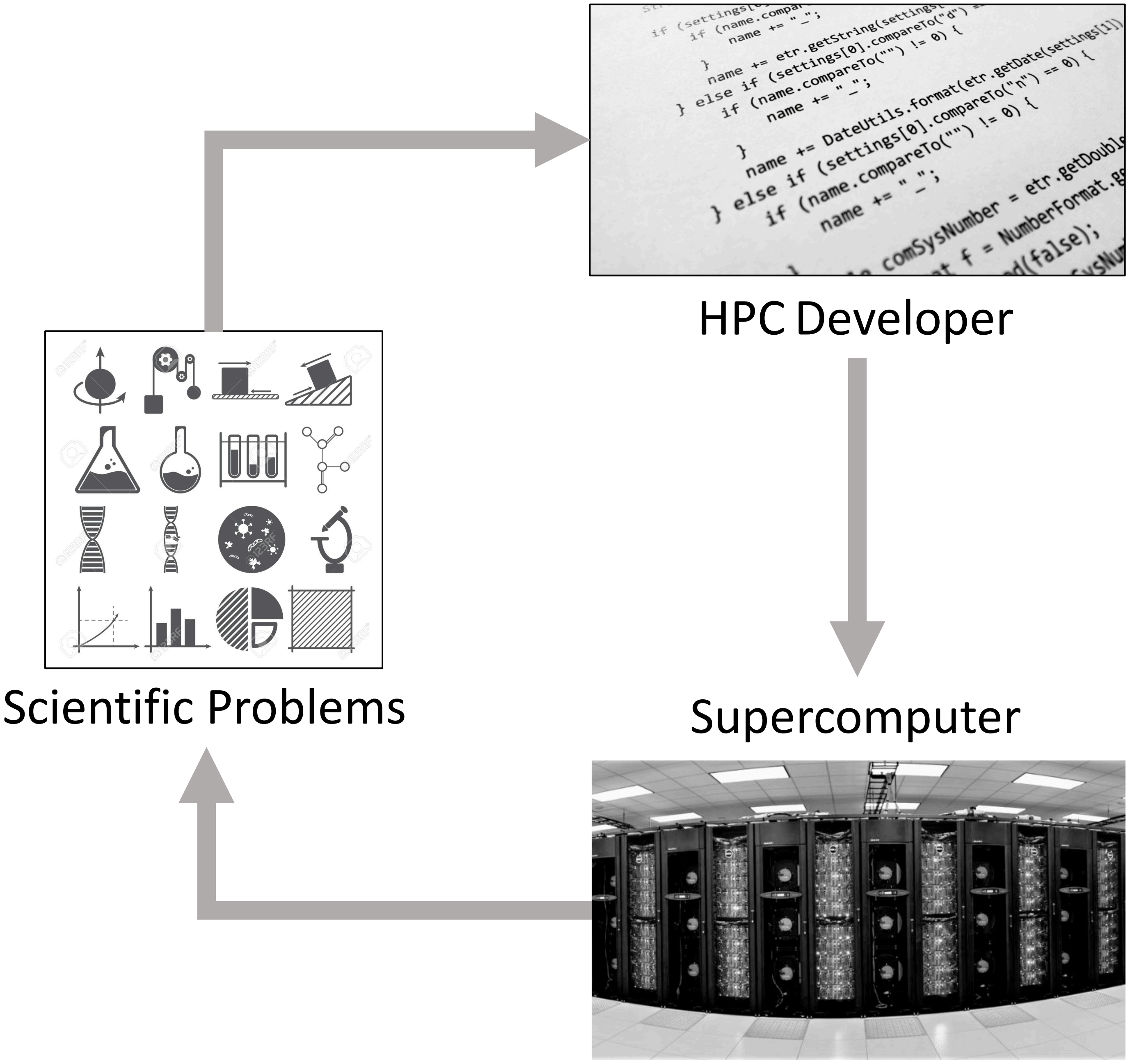
School of Computing, University of Utah

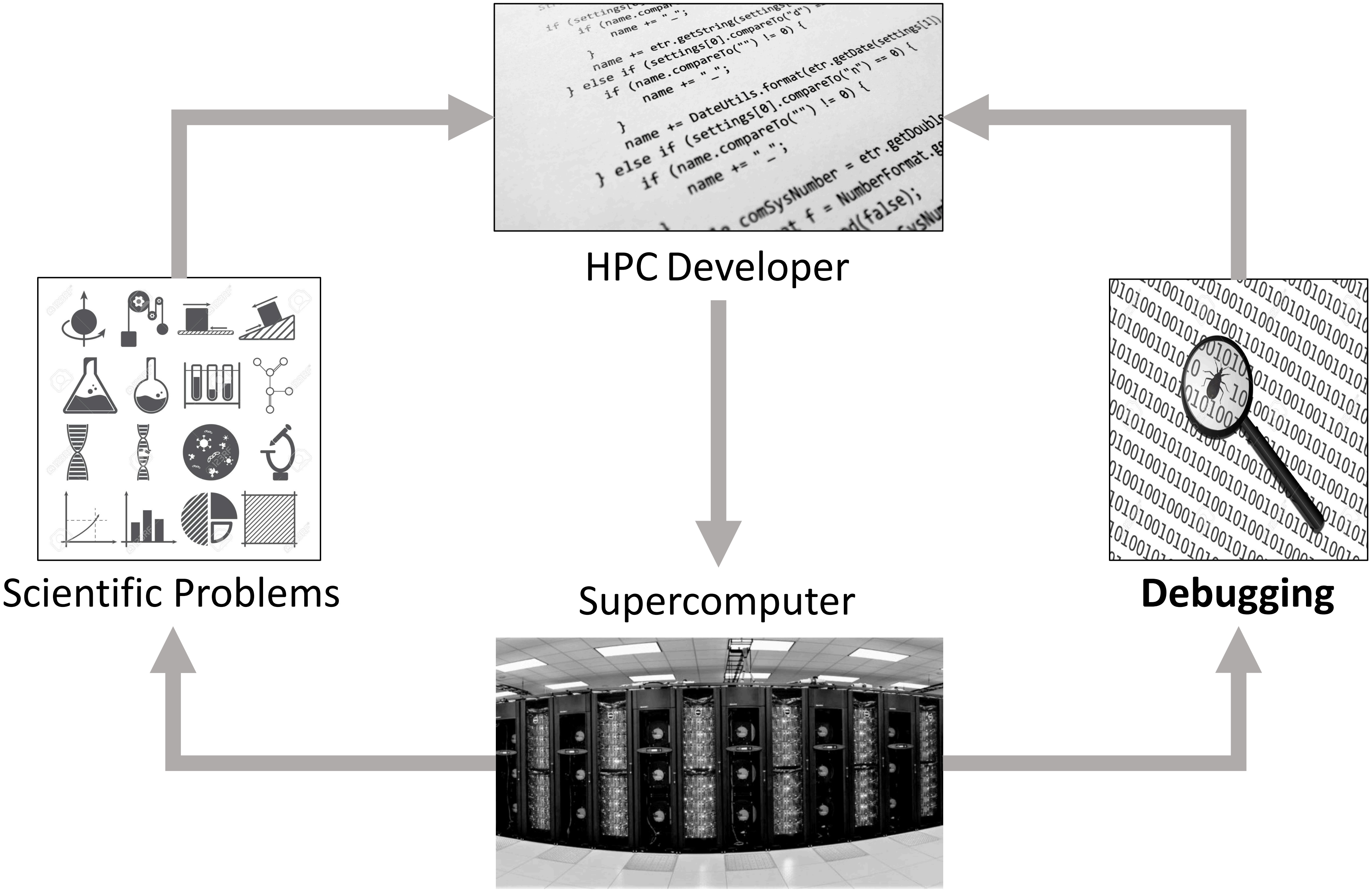
Department of Computer Science, Texas State University

THE UNIVERSITY OF UTAH®

TEXAS STATE UNIVERSITY®

The rising STAR *of Texas*


HPC Developer
Supercomputer
Scientific Problems


HPC Developer
Scientific Problems
Supercomputer
Debugging

HPC Debugging is Challenging…

- Hierarchy of parallelism
- Heterogeneity of compilers & libraries
- Complex and large code bases
- Debugging iterations are **expensive**
  - Resources (time, CPU cycles, energy, etc.)
  - Reproducibility limitations

Debugging Approaches

Existing Approaches
Iteratively • Guess the potential bug • Pick the right debugger • Instrumentation / Re-compile • Re-execute • Gather limited data for specific bug • Analyze data

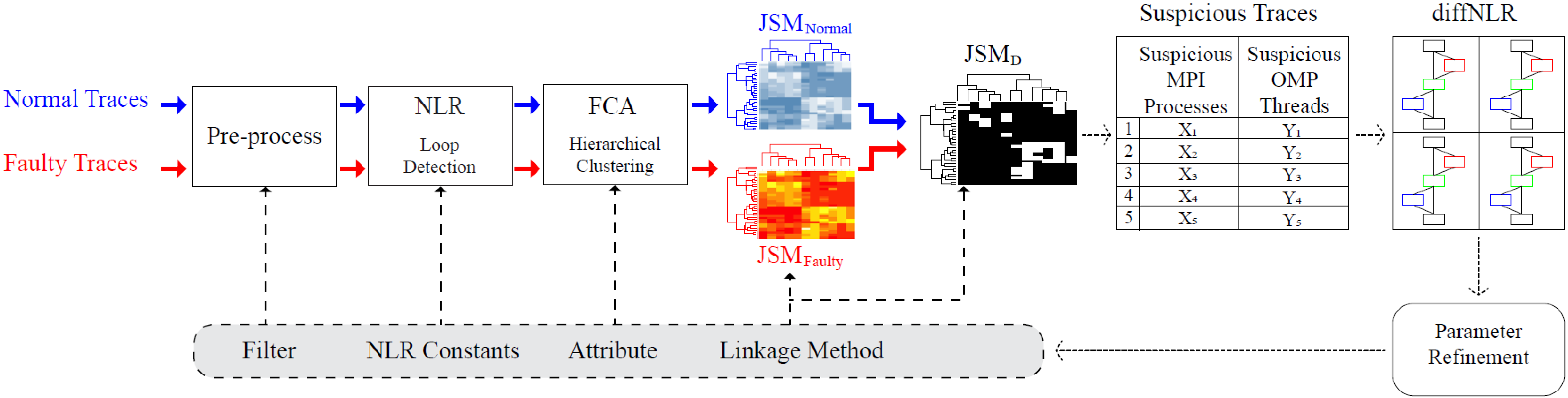
Debugging Approaches

Existing Approaches	**DiffTrace Approach**
Iteratively • Guess the potential bug • Pick the right debugger • Instrumentation / Re-compile • Re-execute • Gather limited data for specific bug • Analyze data	Collect one standard set of data Iteratively (offline): • Intelligently summarize data • **Compare** w/ expected behavior • Detect outliers • Visualize points of **differences**

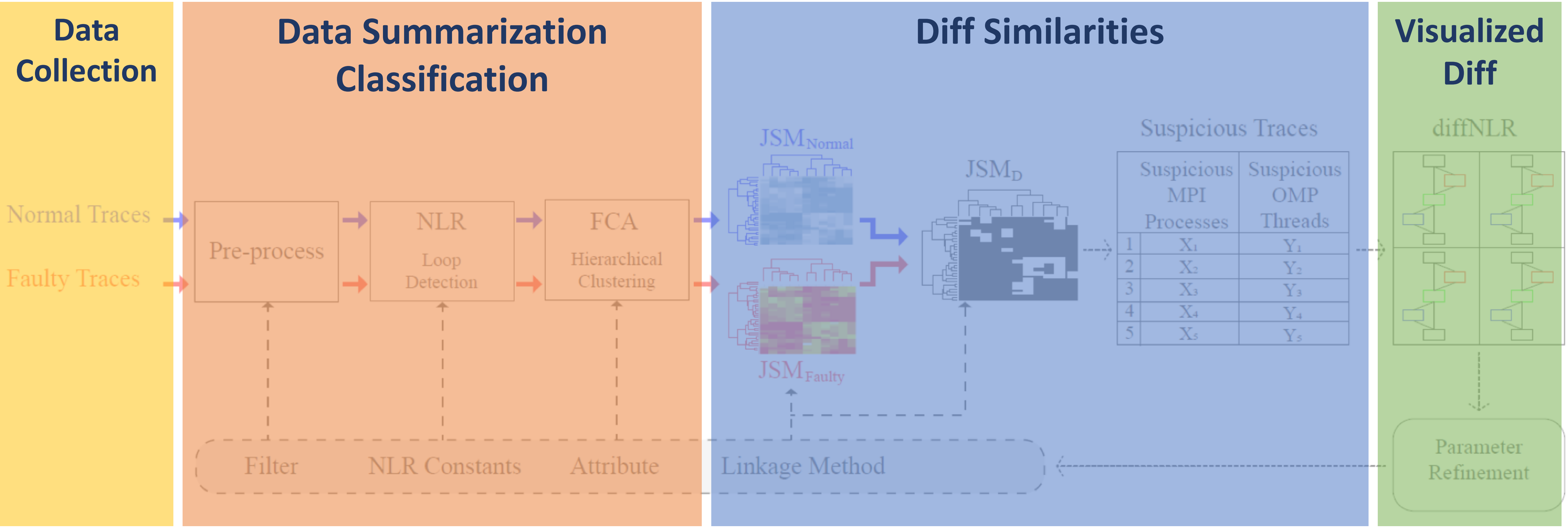
Parallel/HPC Debuggers

- Relative Debugging [DeRose’15]
- Delta Debugging [Choi’02]
- Structural Clustering [Weber’16]
- STAT [Arnold’07]
- AutomaDeD [Laguna’11]

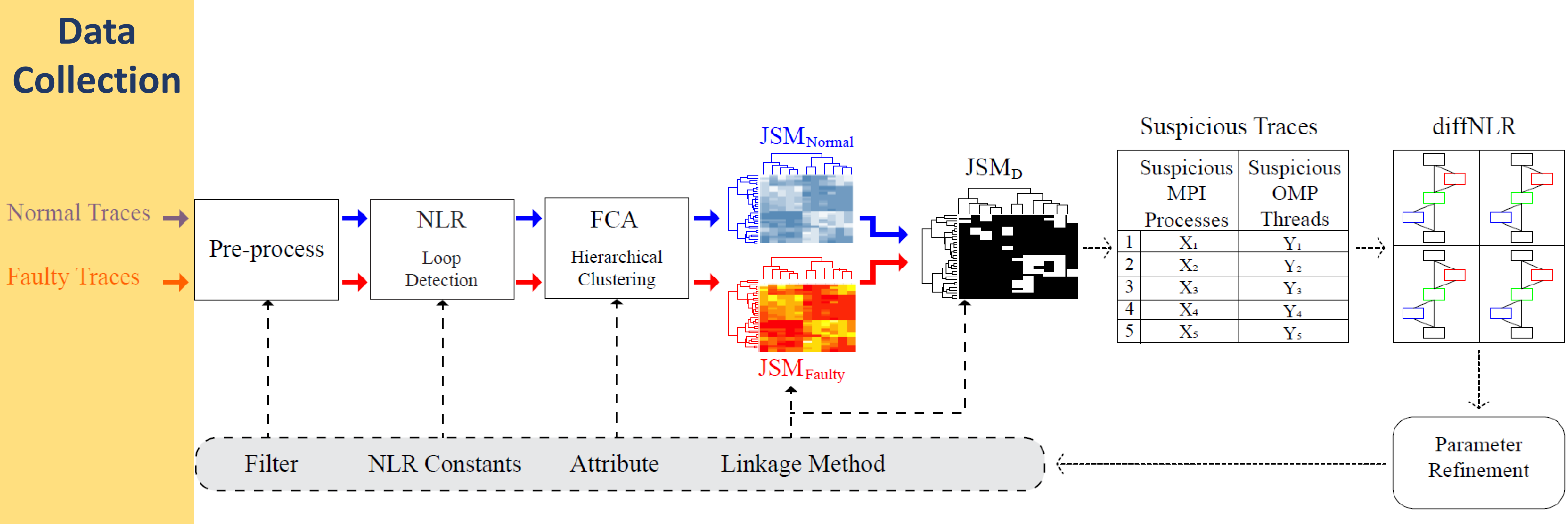
DiffTrace Overview



DiffTrace Overview

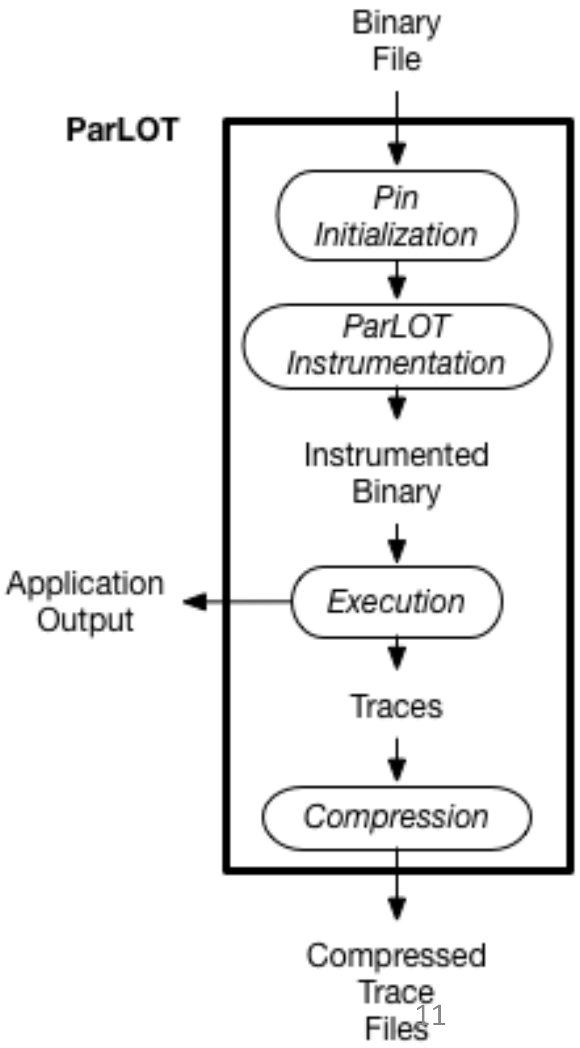


DiffTrace Overview



ParLOT [ESPT'18]

- Instruments binary using Intel **PIN** API
- Captures **function calls/returns** (main/all image)
- **Compress** traces incrementally on-the-fly
  - Avg. compression ratio: **1117.1**
  - Avg. required bandwidth: **7.8 KB/S**
  - Avg. overhead on exec. time: **1.94**
- Enables offline analysis of the whole program



A Walk-through Example

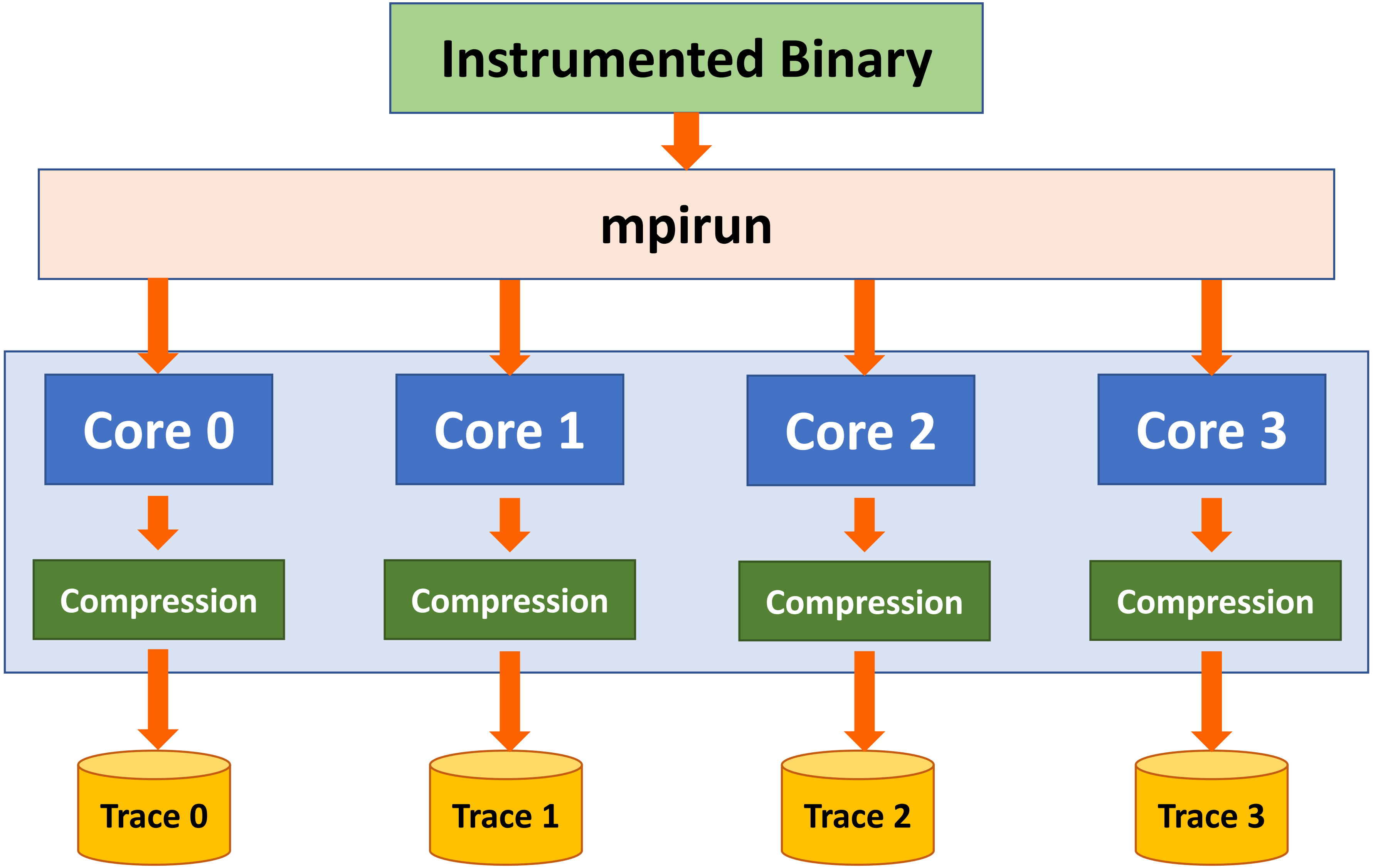
Main Function

```
int main(){
 int rank,cp;
 MPI_Init()
 MPI_Comm_rank(..., &rank);
 MPI_Comm_size(..., &cp);
 // initialize data to sort
 int *data[data_size];
 ...
 oddEvenSort(rank, cp);
 ...
 MPI_Finalize();
}
```

oddEvenSort()

```
oddEvenSort(rank, cp){
 ...
 for (int i=0; i < cp; i++)
 {
  int ptr = findPtr(i, rank);
  ...
  if (rank % 2 == 0) {
   MPI_Send(..., ptr, ...);
   MPI_Recv(..., ptr, ...);
  } else {
   MPI_Recv(..., ptr, ...);
   MPI_Send(..., ptr, ...);
  }
  ...
 }
}
```

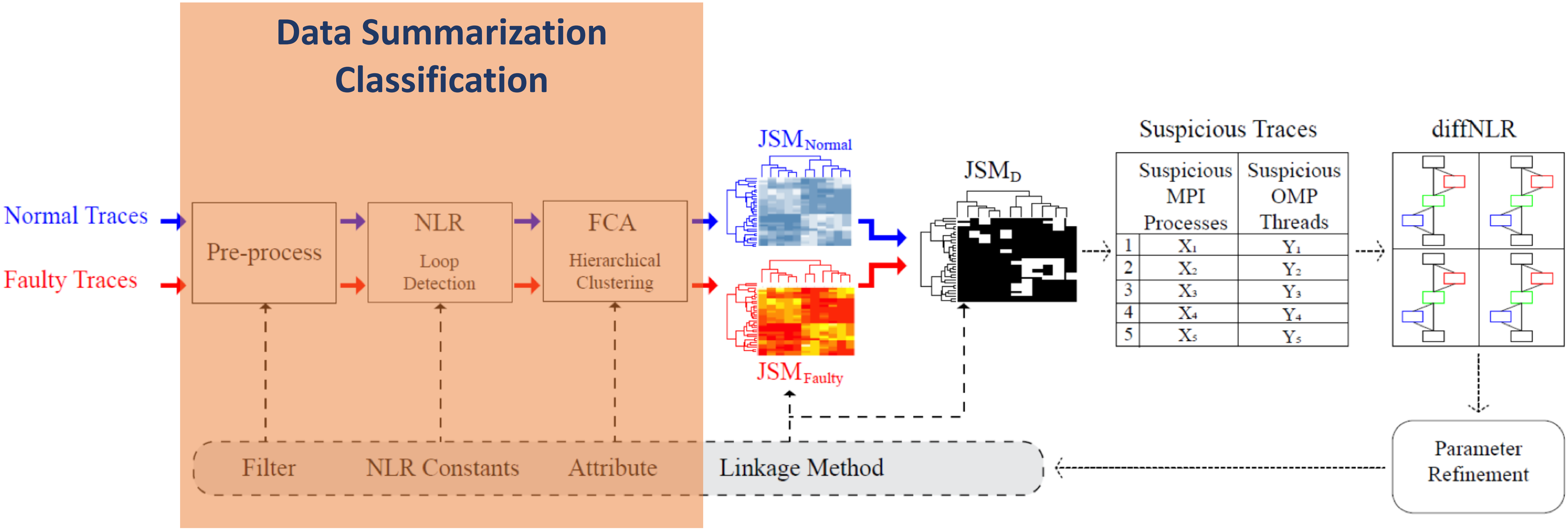
Tracing (mpirun -np 4 pin -t parlot.so -- ./oddeven)



Tracing (mpirun -np 4 pin -t parlot.so -- ./oddeven)

T_0	T_1	T_2	T_3
…	…	…	…
main	main	main	main
MPI_Init	MPI_Init	MPI_Init	MPI_Init
MPI_Comm_rank	MPI_Comm_rank	MPI_Comm_rank	MPI_Comm_rank
MPI_Comm_size	MPI_Comm_size	MPI_Comm_size	MPI_Comm_size
…	…	…	…
oddEvenSort	oddEvenSort	oddEvenSort	oddEvenSort
…	…	…	…
findPtr	findPtr	findPtr	findPtr
MPI_Send	MPI_Recv	MPI_Send	MPI_Recv
MPI_Recv	MPI_Send	MPI_Recv	MPI_Send
…	…	…	…
MPI_Finalize	MPI_Finalize	MPI_Finalize	MPI_Finalize

DiffTrace Overview



Data Pre-processing

Filter Class	Sub-class	Example
Primary	**Returns**	Filter out Returns
	PLT	Procedure Linkage Table
MPI	**MPI ALL**	Functions start with "MPI_"
	MPI Collectives	MPI_Barrier
	MPI Send/Recv	MPI_Send, MPI_Isend
	MPI Library	Inner MPI library
OMP	**OMP ALL**	Functions start with “GOMP_”
	OMP Critical	OMP_Critical_Start
	OMP Mutex	OMP_Mutex
Secondary	Memory	Memcpy
	Network	TCP
	Poll	Poll, yield
	String	strlen
Advanced	Custom	Any source-code function

Loop Summarization

- Programs are (nested) **loops!**
- Loops reflect as sequences of **repetitive** patterns
- Why detecting/summarizing loops?
  - Easy-to-read representation of long traces
  - Reveal unfinished or broken loops due to a fault

Nested Loop Recognition (NLR)

Adapted from NLR algorithm [Ketterlin'14]

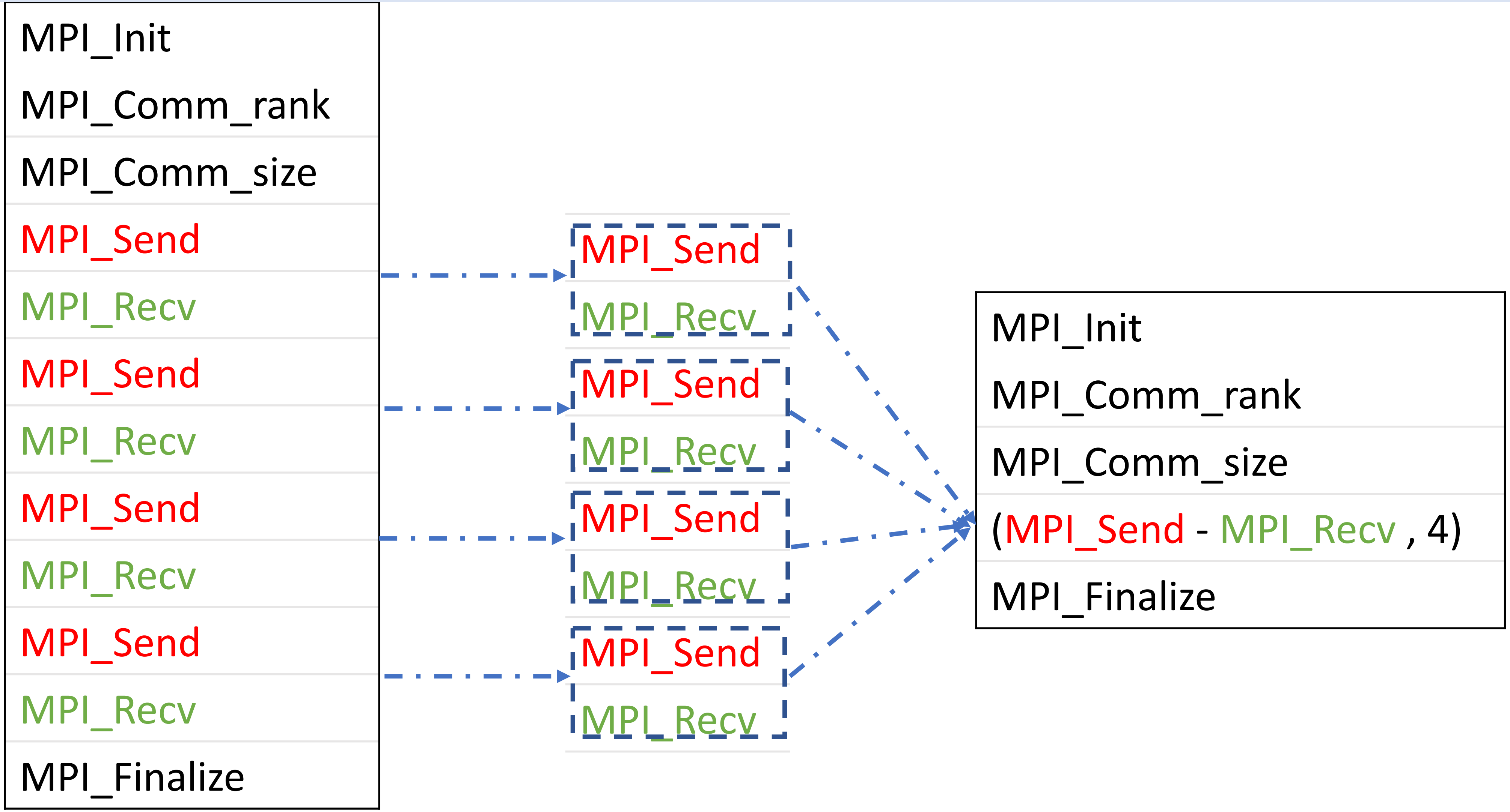
Convert each trace to its equivalent NLR (Nested Loop Representation)

Push **elements** of the trace to a **stack** one by one

On each push:

- Recursively examine the upper elements of the stack to
  - **Form** the loop structure from elements or
  - **Extend** the existing loop structure

Trace to NLR


MPI_Init
MPI_Comm_rank
MPI_Comm_size
MPI_Send
MPI_Recv
MPI_Send
MPI_Recv
MPI_Send
MPI_Recv
MPI_Send
MPI_Recv
MPI_Finalize
MPI_Send
MPI_Recv
MPI_Send
MPI_Recv
MPI_Send
MPI_Recv
MPI_Send
MPI_Recv
MPI_Init
MPI_Comm_rank
MPI_Comm_size
(MPI_Send - MPI_Recv , 4)
MPI_Finalize

Trace to NLR

Loop Table	
L0	MPI_Send - MPI_Recv
L1	MPI_Recv - MPI_Send

T_0	T_1	T_2	T_3
MPI_Init	MPI_Init	MPI_Init	MPI_Init
MPI_Comm_rank	MPI_Comm_rank	MPI_Comm_rank	MPI_Comm_rank
MPI_Comm_size	MPI_Comm_size	MPI_Comm_size	MPI_Comm_size
L0 ^ 2	**L1 ^ 4**	**L0 ^ 4**	**L1 ^ 2**
MPI_Finalize	MPI_Finalize	MPI_Finalize	MPI_Finalize

Hierarchical Clustering via FCA

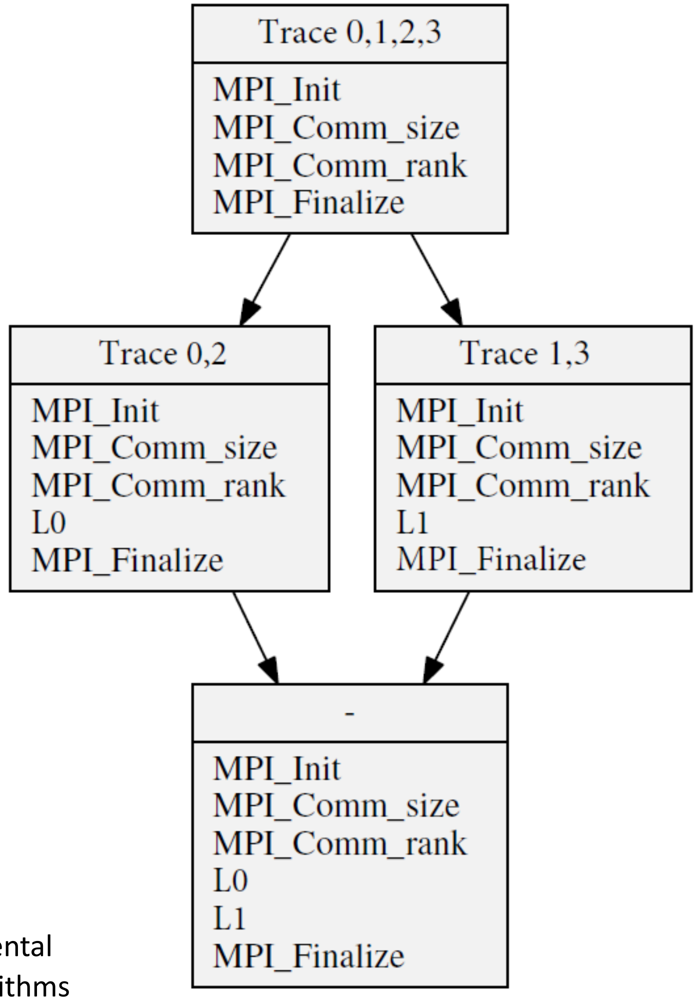
- Few **equivalence classes** of threads/processes in HPC applications
  - Master/worker, Odd/Even, Producer/Consumer
- Clustering based on this property
  - Distinguish between structurally different threads
  - Reduce the search space for bug location
  - Detect mis-behaved traces (i.e., outliers)
- STAT: Prefix trees; AutomaDeD: Markov model
- DiffTrace Approach: **Formal Concept Analysis (FCA)**

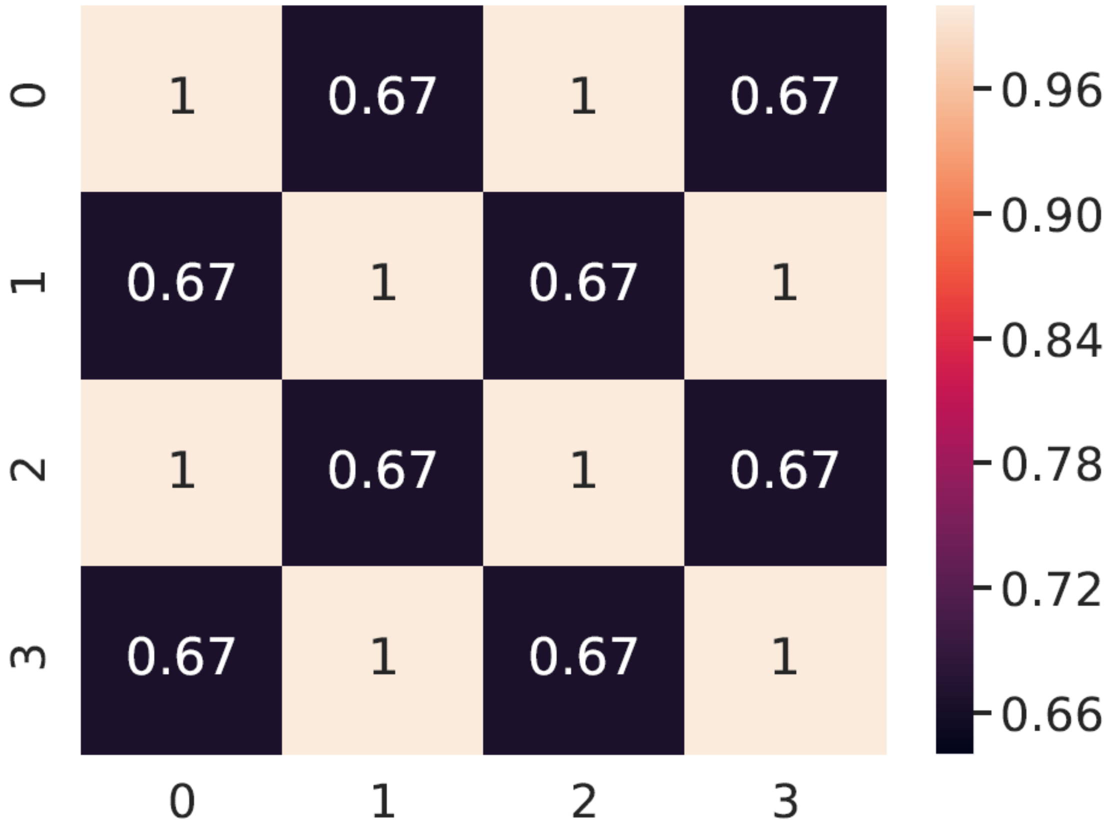
FCA

- *Formal Concept Analysis* (FCA) is a way of deriving a **concept hierarchy** from a collection of **objects** and their **attributes.**

	MPI_Init()	MPI_Comm_Size()	MPI_Comm_Rank()	L0	L1	MPI_Finalize()
Trace 0	×	×	×	×		×
Trace 1	×	×	×		×	×
Trace 2	×	×	×	×		×
Trace 3	×	×	×		×	×

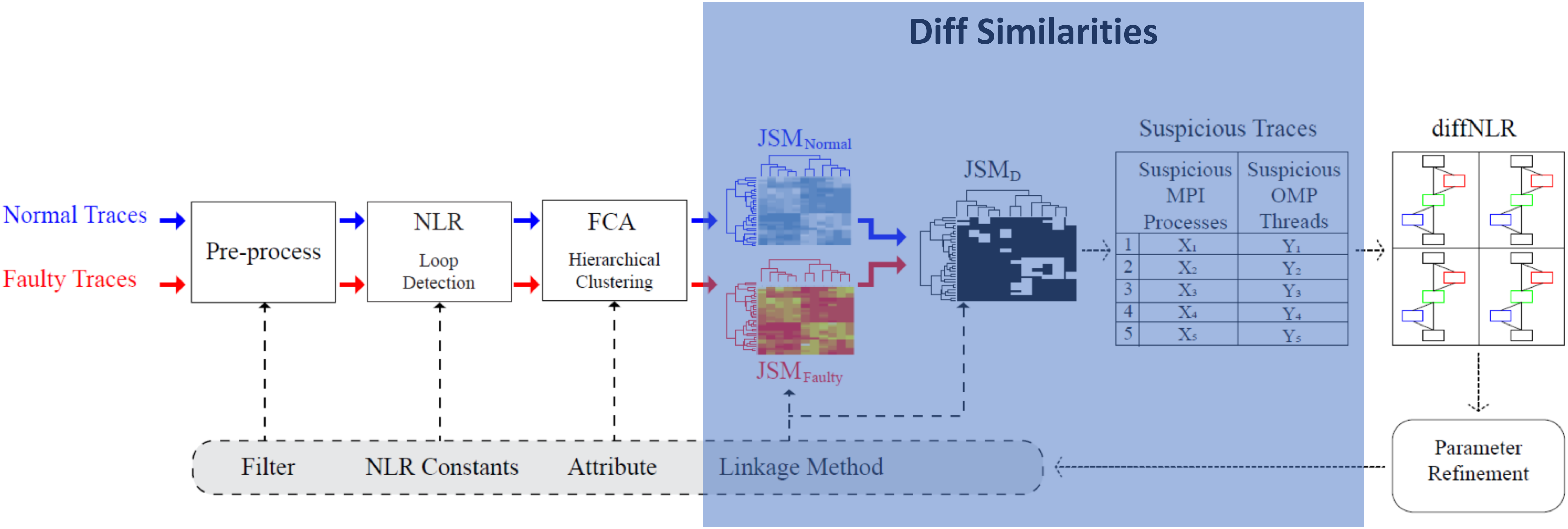
Concept Lattice & JSM (Jaccard Similarity Matrix)





[R. Godin et al, “Incremental concept formation algorithms based on galois (concept) lattices”]

DiffTrace Overview



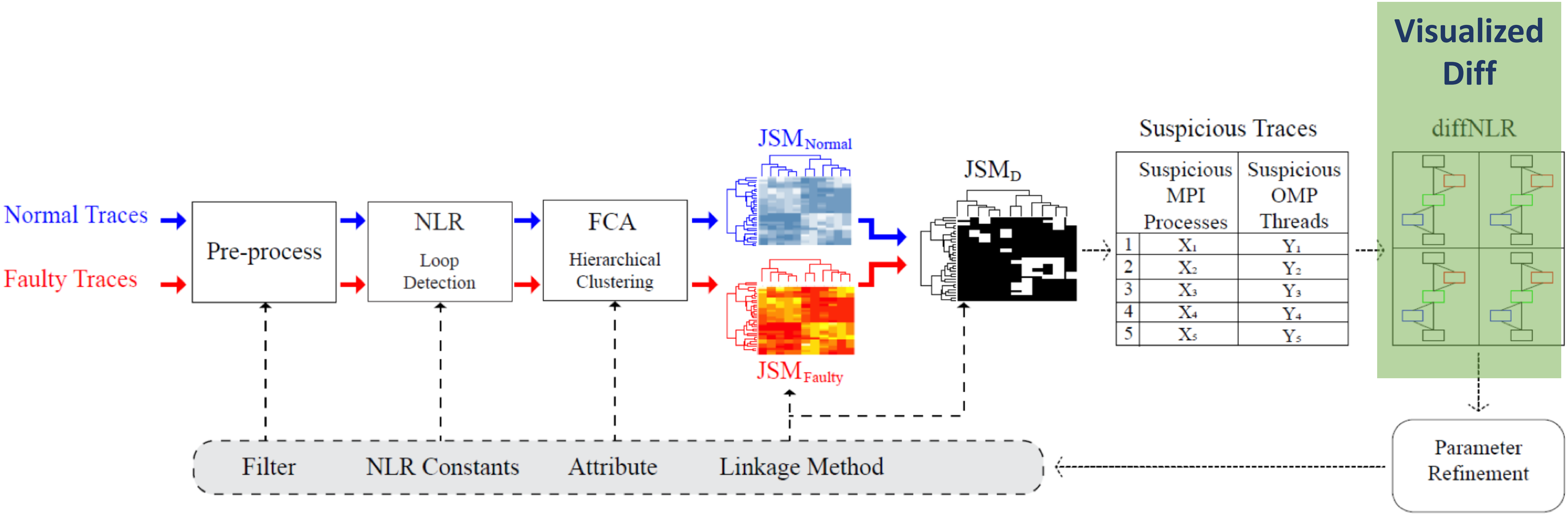
Diff Similarity Matrices

- Compute how **similarity** relations among traces of a normal execution **changes** when faults are introduced.
- $JSM_D = |JSM_{faulty} - JSM_{normal}|$
- Hierarchical clustering based on JSM_D:

Reveals the traces that have changed the most w.r.t their similarity with other traces

- Ranking metric: **B-score –** The distance between two clusterings

DiffTrace Overview



Potential Bug

oddEvenSort()

```
oddEvenSort(rank, cp){
 ...
 for (int i=0; i < cp; i++)
 {
  int ptr = findPtr(i, rank);
  ...
  if (rank % 2 == 0) {
   MPI_Send(..., ptr, ...);
   MPI_Recv(..., ptr, ...);
  } else {
   MPI_Recv(..., ptr, ...);
   MPI_Send(..., ptr, ...);
  }
  ...
 }
}
```

oddEvenSort()

```
oddEvenSort(rank, cp){
 ...
 for (int i=0; i < cp; i++)
 {
  int ptr = findPtr(i, rank);
  ...
  if (rank % 2 == 0) {
   MPI_Send(..., ptr, ...);
   MPI_Recv(..., ptr, ...);
  } else {
   MPI_Send(..., ptr, ...);
   MPI_Recv(..., ptr, ...);
  }
  ...
 }
}
```

Planted Bug

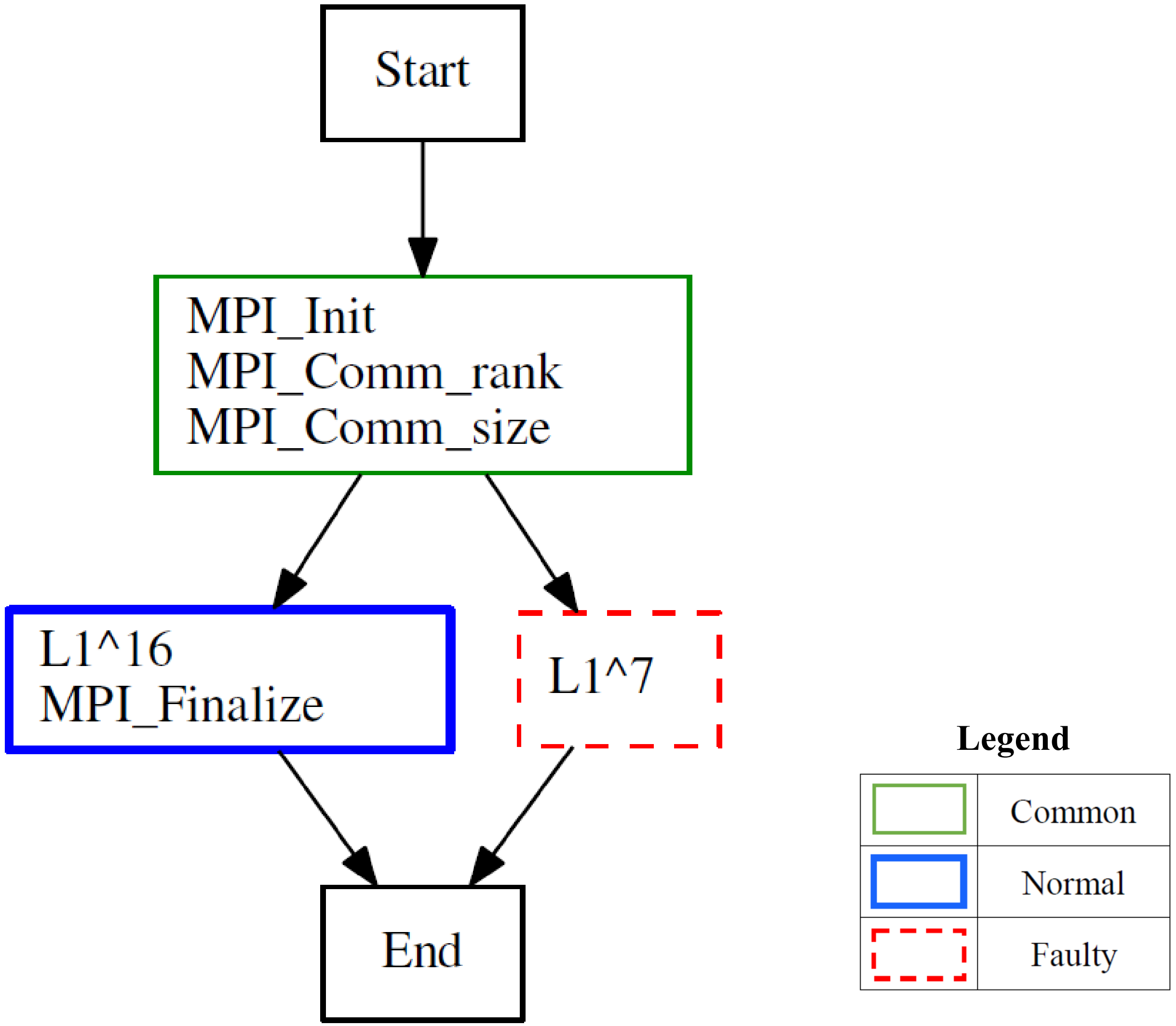
Deadlock:

- Only after 7 iterations
- Only in process #5
- Suggested Rank: **#5**
- diffNLR(5_{normal},5_{faulty})

diffNLR

Deadlock:

- Only after 7 iterations
- Only in process #5
- Suggested Rank: **#5**
- diffNLR(5_{normal},5_{faulty})

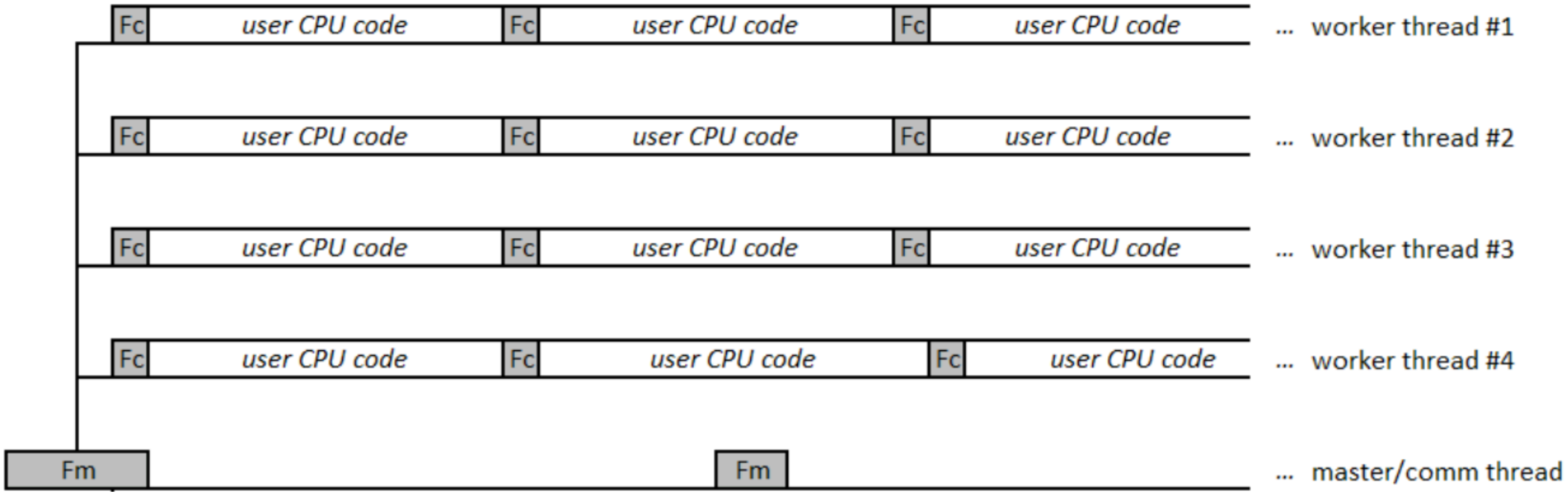


ILCS Case Study

- **ILCS:** A scalable framework for running iterative local searches on HPC platforms. LOC: 276 , Scales up to **32,768** cores

ILCS Case Study

- **ILCS:** A scalable framework for running iterative local searches on HPC platforms. LOC: 276 , Scales up to **32,768** cores
  - Workers: find the local champion
  - Masters: globally reduce local champions



Result #1: Unprotected Memory Access

- Worker thread **#4** of process **#6**
- Omitted the critical section
- Results in data race that might produce corrupted result

Result #1: Unprotected Memory Access

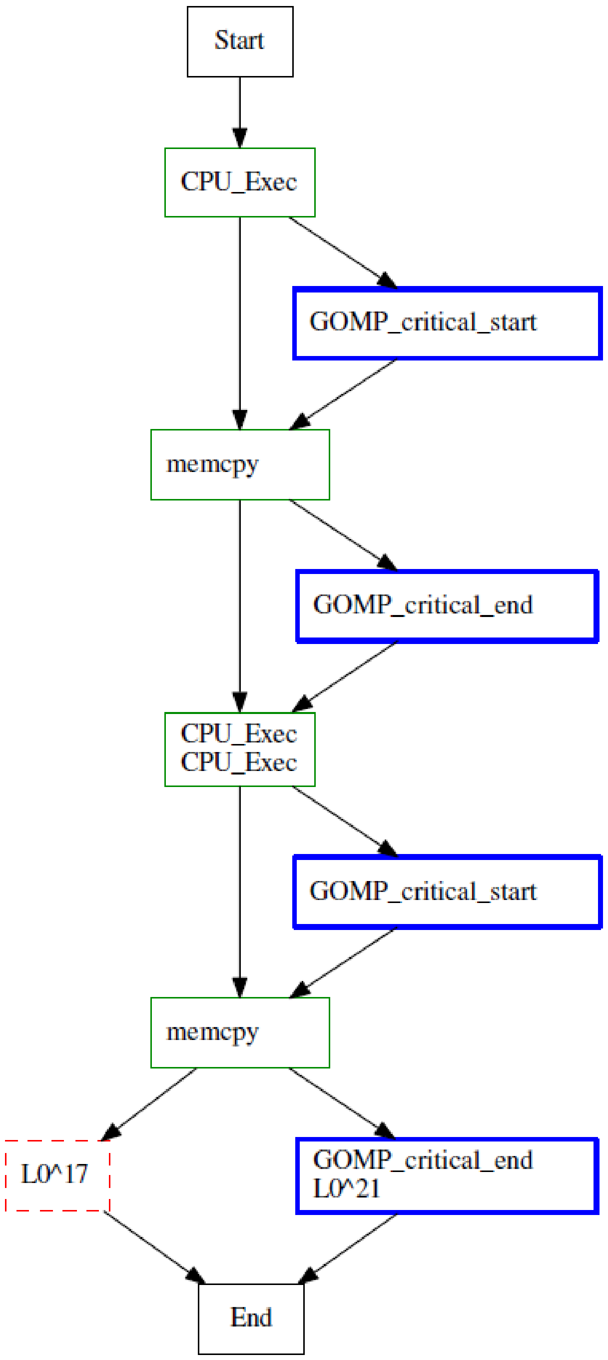
- Worker thread **#4** of process **#6**
- Omitted the critical section
- Results in data race that might produce corrupted result

Filter	Attributes	B-score	Top Processes	Top Threads
11.plt.mem.cust.0K10	doub.noFreq	0.244	7, 3, 4	**6.4**, 7.3, 1.4, 3.3, 3.4, 4.2
11.plt.mem.cust.0K10	doub.log10	0.244	7, 3, 4	**6.4**, 7.3, 1.4, 3.3, 3.4, 4.2
01.plt.mem.cust.0K10	doub.noFreq	0.244	7, 3, 4	**6.4**, 7.3, 1.4, 3.3, 3.4, 4.2
01.plt.mem.cust.0K10	doub.log10	0.244	7, 3, 4	**6.4**, 7.3, 1.4, 3.3, 3.4, 4.2
01.mem.ompcrit.cust.0K10	sing.log10	0.262	3	**6.4**, 7.1, 3.3, 4.1, 5.1, 6.1
01.mem.ompcrit.cust.0K10	sing.noFreq	0.262	3	**6.4**, 7.1, 3.3, 4.1, 5.1, 6.1
11.mem.ompcrit.cust.0K10	sing.log10	0.262	3	**6.4**, 7.1, 3.3, 4.1, 5.1, 6.1
11.mem.ompcrit.cust.0K10	sing.noFreq	0.262	3	**6.4**, 7.1, 3.3, 4.1, 5.1, 6.1
11.plt.mem.cust.0K10	doub.actual	0.273	7	**6.4**, 2.4, 3.4, 4.2, 4.4
01.plt.mem.cust.0K10	doub.actual	0.273	7	**6.4**, 2.4, 3.4, 4.2, 4.4

Result #1: Unprotected Memory Access

- Worker thread **#4** of process **#6**
- Omitted the critical section
- Results in data race that might produce corrupted result

Filter	Attributes	B-score	Top Processes	Top Threads
11.plt.mem.cust.0K10	doub.noFreq	0.244	7, 3, 4	**6.4**, 7.3, 1.4, 3.3, 3.4, 4.2
11.plt.mem.cust.0K10	doub.log10	0.244	7, 3, 4	**6.4**, 7.3, 1.4, 3.3, 3.4, 4.2
01.plt.mem.cust.0K10	doub.noFreq	0.244	7, 3, 4	**6.4**, 7.3, 1.4, 3.3, 3.4, 4.2
01.plt.mem.cust.0K10	doub.log10	0.244	7, 3, 4	**6.4**, 7.3, 1.4, 3.3, 3.4, 4.2
01.mem.ompcrit.cust.0K10	sing.log10	0.262	3	**6.4**, 7.1, 3.3, 4.1, 5.1, 6.1
01.mem.ompcrit.cust.0K10	sing.noFreq	0.262	3	**6.4**, 7.1, 3.3, 4.1, 5.1, 6.1
11.mem.ompcrit.cust.0K10	sing.log10	0.262	3	**6.4**, 7.1, 3.3, 4.1, 5.1, 6.1
11.mem.ompcrit.cust.0K10	sing.noFreq	0.262	3	**6.4**, 7.1, 3.3, 4.1, 5.1, 6.1
11.plt.mem.cust.0K10	doub.actual	0.273	7	**6.4**, 2.4, 3.4, 4.2, 4.4
01.plt.mem.cust.0K10	doub.actual	0.273	7	**6.4**, 2.4, 3.4, 4.2, 4.4



Result #2: Collective with Wrong Size

- MPI_Allreduce with wrong size: DL
- Process #2

Result #2: Collective with Wrong Size

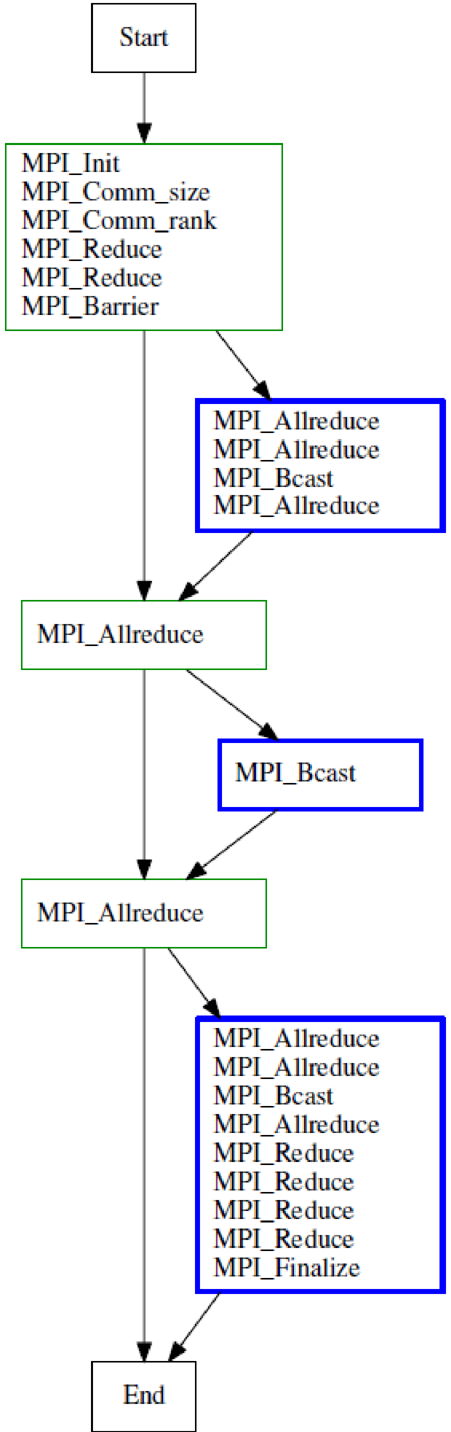
- MPI_Allreduce with wrong size: DL
- Process #2

Filter	Attributes	B-score	Top Processes	Top Threads
11.mpicol.cust.0K10	sing.log10	0.439	0, 7, 2, 4, 5, 6	1.1, 1.3, 3.1, 3.2, 3.4
11.mpicol.cust.0K10	sing.noFreq	0.439	0, 7, 2, 4, 5, 6	1.1, 1.3, 3.1, 3.2, 3.4
11.mpi.cust.0K10	doub.noFreq	0.457	0, 7, 2, 4, 5, 6	1.4, 3.3, 3.4
11.mpi.cust.0K10	doub.actual	0.457	0, 7, 2, 4, 5, 6	1.4, 3.3, 3.4
11.mpiall.cust.0K10	doub.noFreq	0.457	0, 7, 2, 4, 5, 6	1.4, 3.3, 3.4
11.mpiall.cust.0K10	doub.actual	0.457	0, 7, 2, 4, 5, 6	1.4, 3.3, 3.4
11.mpicol.cust.0K10	doub.noFreq	0.457	0, 7, 2, 4, 5, 6	1.4, 3.3, 3.4
11.mpicol.cust.0K10	doub.actual	0.457	0, 7, 2, 4, 5, 6	1.4, 3.3, 3.4
11.mpi.cust.0K10	sing.log10	0.465	0, 7, 2, 4, 5, 6	1.1, 1.3, 3.1, 3.2, 3.4
11.mpi.cust.0K10	sing.noFreq	0.465	0, 7, 2, 4, 5, 6	1.1, 1.3, 3.1, 3.2, 3.4
11.mpiall.cust.0K10	sing.log10	0.465	0, 7, 2, 4, 5, 6	1.1, 1.3, 3.1, 3.2, 3.4
11.mpiall.cust.0K10	sing.noFreq	0.465	0, 7, 2, 4, 5, 6	1.1, 1.3, 3.1, 3.2, 3.4
11.mpi.cust.0K10	doub.noFreq	0.543	0, 7, 2, 4, 5, 6	1.4, 3.3, 3.4
11.mpi.cust.0K10	doub.actual	0.543	0, 7, 2, 4, 5, 6	1.4, 3.3, 3.4

Result #2: Collective with Wrong Size

- MPI_Allreduce with wrong size: DL
- Process #2

Filter	Attributes	B-score	Top Processes	Top Threads
11.mpicol.cust.0K10	sing.log10	0.439	0, 7, 2, 4, 5, 6	1.1, 1.3, 3.1, 3.2, 3.4
11.mpicol.cust.0K10	sing.noFreq	0.439	0, 7, 2, 4, 5, 6	1.1, 1.3, 3.1, 3.2, 3.4
11.mpi.cust.0K10	doub.noFreq	0.457	0, 7, 2, 4, 5, 6	1.4, 3.3, 3.4
11.mpi.cust.0K10	doub.actual	0.457	0, 7, 2, 4, 5, 6	1.4, 3.3, 3.4
11.mpiall.cust.0K10	doub.noFreq	0.457	0, 7, 2, 4, 5, 6	1.4, 3.3, 3.4
11.mpiall.cust.0K10	doub.actual	0.457	0, 7, 2, 4, 5, 6	1.4, 3.3, 3.4
11.mpicol.cust.0K10	doub.noFreq	0.457	0, 7, 2, 4, 5, 6	1.4, 3.3, 3.4
11.mpicol.cust.0K10	doub.actual	0.457	0, 7, 2, 4, 5, 6	1.4, 3.3, 3.4
11.mpi.cust.0K10	sing.log10	0.465	0, 7, 2, 4, 5, 6	1.1, 1.3, 3.1, 3.2, 3.4
11.mpi.cust.0K10	sing.noFreq	0.465	0, 7, 2, 4, 5, 6	1.1, 1.3, 3.1, 3.2, 3.4
11.mpiall.cust.0K10	sing.log10	0.465	0, 7, 2, 4, 5, 6	1.1, 1.3, 3.1, 3.2, 3.4
11.mpiall.cust.0K10	sing.noFreq	0.465	0, 7, 2, 4, 5, 6	1.1, 1.3, 3.1, 3.2, 3.4
11.mpi.cust.0K10	doub.noFreq	0.543	0, 7, 2, 4, 5, 6	1.4, 3.3, 3.4
11.mpi.cust.0K10	doub.actual	0.543	0, 7, 2, 4, 5, 6	1.4, 3.3, 3.4



Summary

- DiffTrace situates HPC debugging around whole program trace diffing
  - Provides user-selectable filters
  - Summarizes loops based on the state-of-the-art algorithms
  - Condenses summarized traces into concept lattices
  - Obtains similarity matrices and hierarchically clusters traces
  - Detects, ranks and highlights most salient differences w.r.t. normal execution
- DiffTrace addresses missing features in existing tools

Future Work

- Optimize DiffTrace components to exploit multi-core CPUs
- Convert ParLOT traces into known formats such as OTF2 to mine temporal properties of functions
- Conduct systematic bug injection to evaluate use of concept lattices and loop structures as features for bug classification (via ML and NN)
- Take up more challenging and real-world examples to evaluate DiffTrace against similar tools, and release it to the community.

Thanks.
Any questions?