



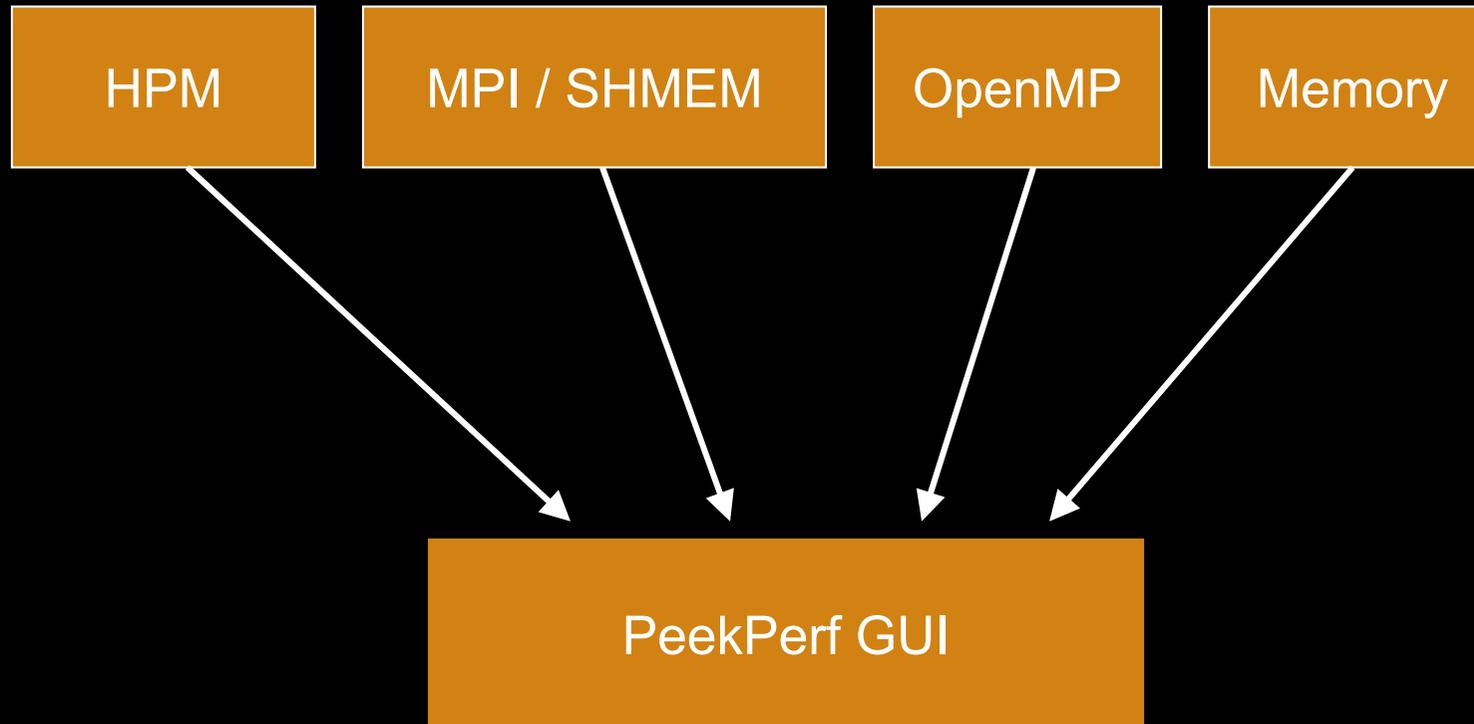
# The IBM High Performance Computing Toolkit

I-Hsin Chung  
IBM T.J. Watson Research Center  
[ihchung@us.ibm.com](mailto:ihchung@us.ibm.com)

## IBM HPC Toolkit Highlights:

- A common application performance analysis environment across **all IBM HPC servers**
- Common framework for performance analysis of **communication, memory, CPU, shared-memory, I/O**
- A **flexible** framework in which new analysis tools can be easily plugged in
- Operate on the **binary** and yet provide reports in terms of source-level symbols
- Full source code **traceback** capability
- **Dynamically** activate/deactivate data collection and change what information to collect
- **Query** and “what-if” capabilities
- Support **automation** of the performance tuning process

# OLD vs NEW HPC Toolkit (PERCS/HPCS Initiative)

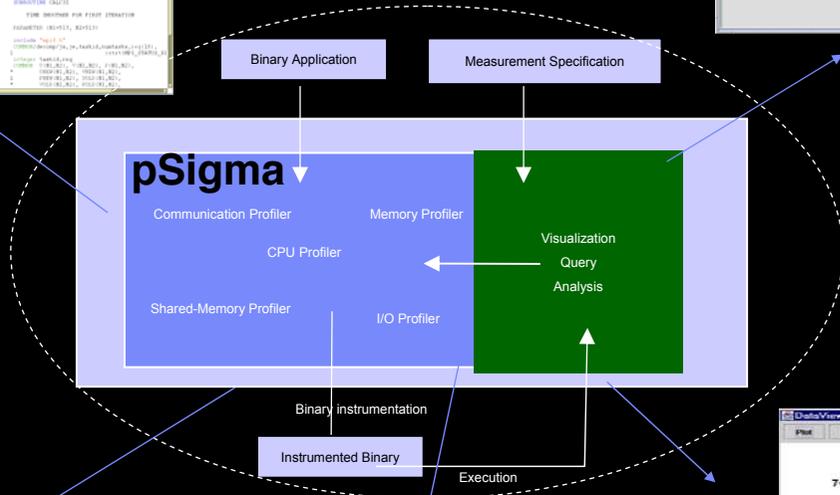
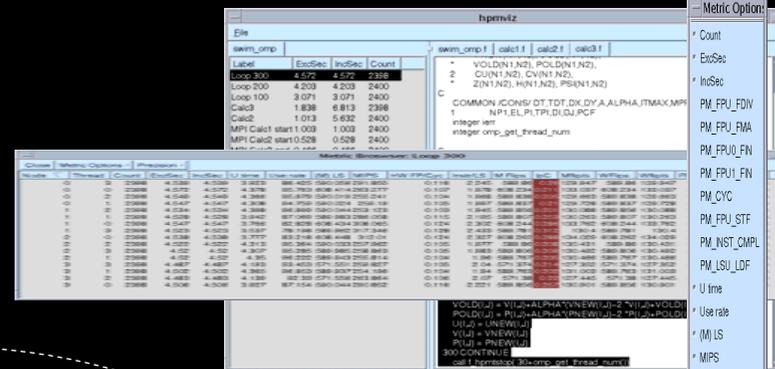
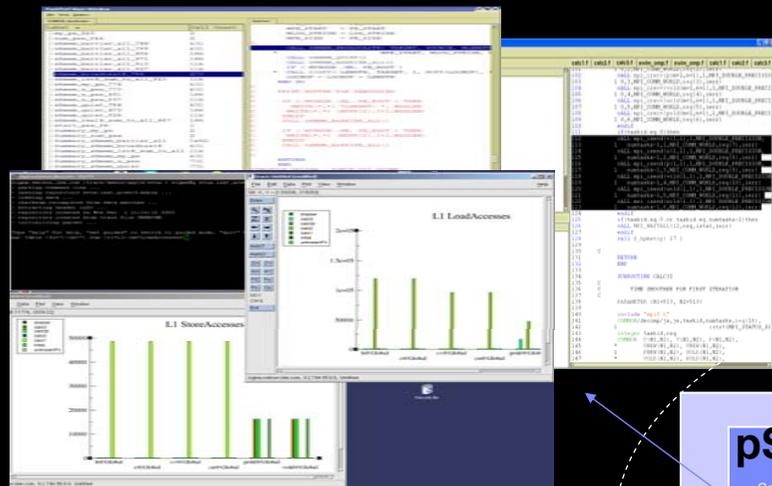


“OLD”: A Collection of tools unified by a GUI for data analysis

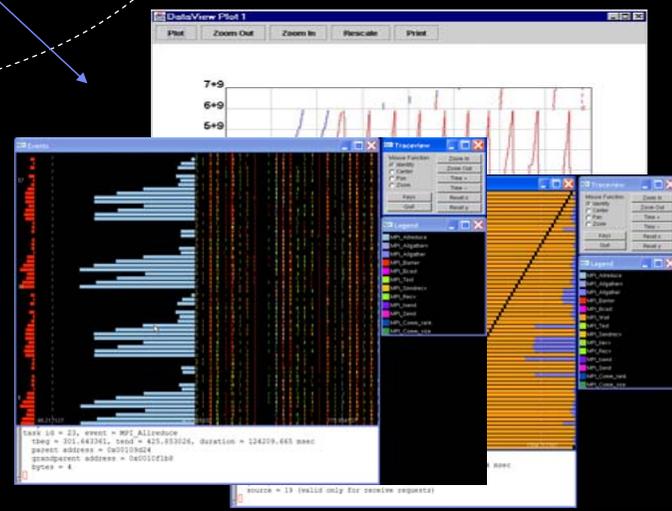
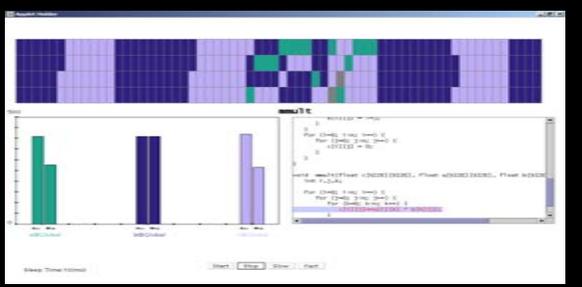
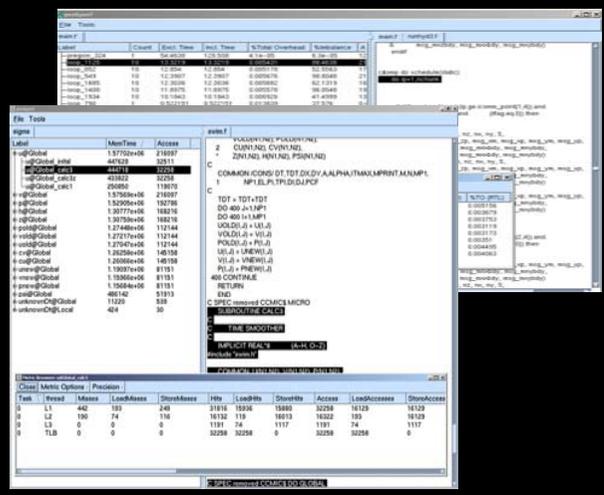
## Disadvantages:

- Lack of standard in the **instrumentation**: source code instrumentation (HPM), DPCL instrumentation (OpenMP), binary instrumentation (Memory)
- **Source code modifications** and recompilations often needed
- Lack of standard in **traceback** mechanism
- No **dynamic** capabilities
- Not easily **extensible**
- No support for **iterative tuning**
- Tuning process difficult to **automate**

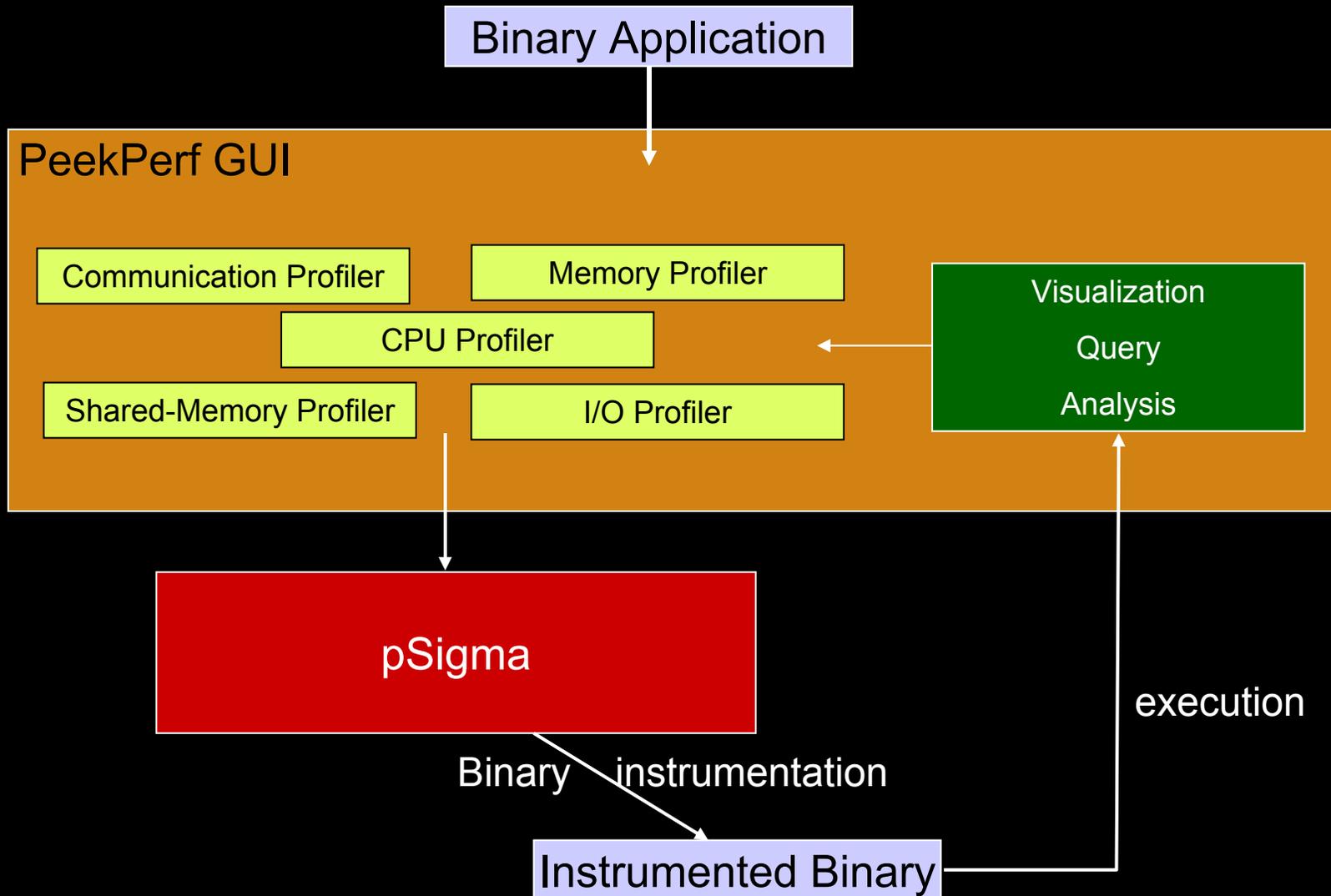
# IBM High Performance Computing Toolkit (HPCS)



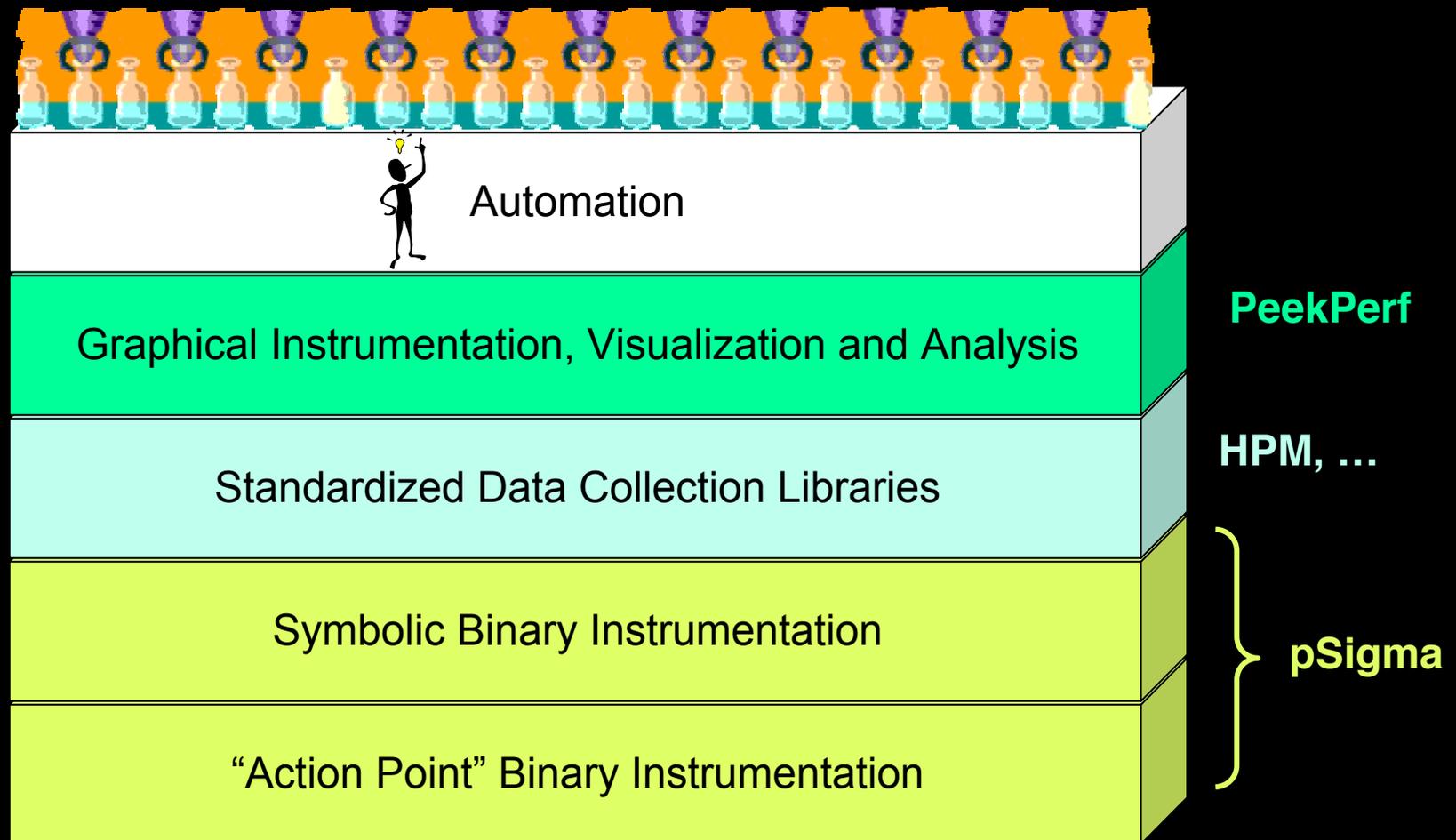
- Metric Options:
- Count
  - ExcSec
  - InstSec
  - PM\_FPU\_FDV
  - PM\_FPU\_FMA
  - PM\_FPU\_FN
  - PM\_FPU\_FN
  - PM\_CYC
  - PM\_FPU\_STF
  - PM\_INST\_CMPL
  - PM\_LSU\_LDF
  - U time
  - Userate
  - (M)LS
  - MIPS
  - HW FPUcyc
  - InstrLS
  - M Flips
  - lcc
  - Mlipsis
  - WFlips
  - Wlipsis
  - FMA %
  - Comp Int.



# Structure of the New HPC toolkit



# HPC Toolkit Software Stack



# Action Point Binary Instrumentation

Basic ability to modify a binary executable so that the execution of the program will be intercepted at given points and one or more associated actions can be executed.

shallow:	0x7c0802a6	mflr	r0	
0x100003fc:	0x7d800026	mfcrr	r12	
0x10000400:	0x48000781	bl	.Ssavef28	
0x10000404:	0x80620050	lwz	r3,80(rtoc)	—————→ <i>Execute Action 1</i>
0x10000408:	0xbe41ffa8	stmw	r18,-88(sp)	
0x1000040c:	0x91810004	stw	r12,4(sp)	
0x10000410:	0x90010008	stw	r0,8(sp)	
0x10000414:	0x607f0000	ori	r31,r3,0	—————→ <i>Execute Actions 2, 3, 4</i>
0x10000418:	0x9421fed0	stwu	sp,-304(sp)	
0x1000041c:	0x48000791	bl	.mpi_init	—————→ <i>Execute Actions 5, 6</i>
0x10000420:	0x80410014	lwz	rtoc,20(sp)	
0x10000424:	0x3bc00000	li	r30,0	
0x10000428:	0x83a20058	lwz	r29,88(rtoc)	
0x1000042c:	0x93c10044	stw	r30,68(sp)	
0x10000430:	0x38610044	addi	r3,sp,68	
0x10000434:	0x389d0008	addi	r4,r29,8	
0x10000438:	0x80a20050	lwz	r5,80(rtoc)	
0x1000043c:	0x48000799	bl	.mpi_comm_rank	

# Action Point Binary Instrumentation

Example: intercept all memory operations

shallow:	0x7c0802a6	mflr	r0	
0x100003fc:	0x7d800026	mfcrr	r12	
0x10000400:	0x48000781	bl	.Ssavef28	
0x10000404:	0x80620050	lww	r3,80(rtoc)	
0x10000408:	0xbe41ffa8	stmw	r18,-88(sp)	
0x1000040c:	0x91810004	stw	r12,4(sp)	
0x10000410:	0x90010008	stw	r0,8(sp)	
0x10000414:	0x607f0000	ori	r31,r3,0	
0x10000418:	0x9421fed0	stwu	sp,-304(sp)	
0x1000041c:	0x48000791	bl	.mpi_init	
0x10000420:	0x80410014	lww	rtoc,20(sp)	
0x10000424:	0x3bc00000	li	r30,0	
0x10000428:	0x83a20058	lww	r29,88(rtoc)	
0x1000042c:	0x93c10044	stw	r30,68(sp)	
0x10000430:	0x38610044	addi	r3,sp,68	
0x10000434:	0x389d0008	addi	r4,r29,8	
0x10000438:	0x80a20050	lww	r5,80(rtoc)	
0x1000043c:	0x48000799	bl	.mpi_comm_rank	

*Execute Action 1*

# Actions

Only two basic actions:

- Invoke a sequence of functions
- Activate/Deactivate the instrumentation

Action point = (position, [condition], action)

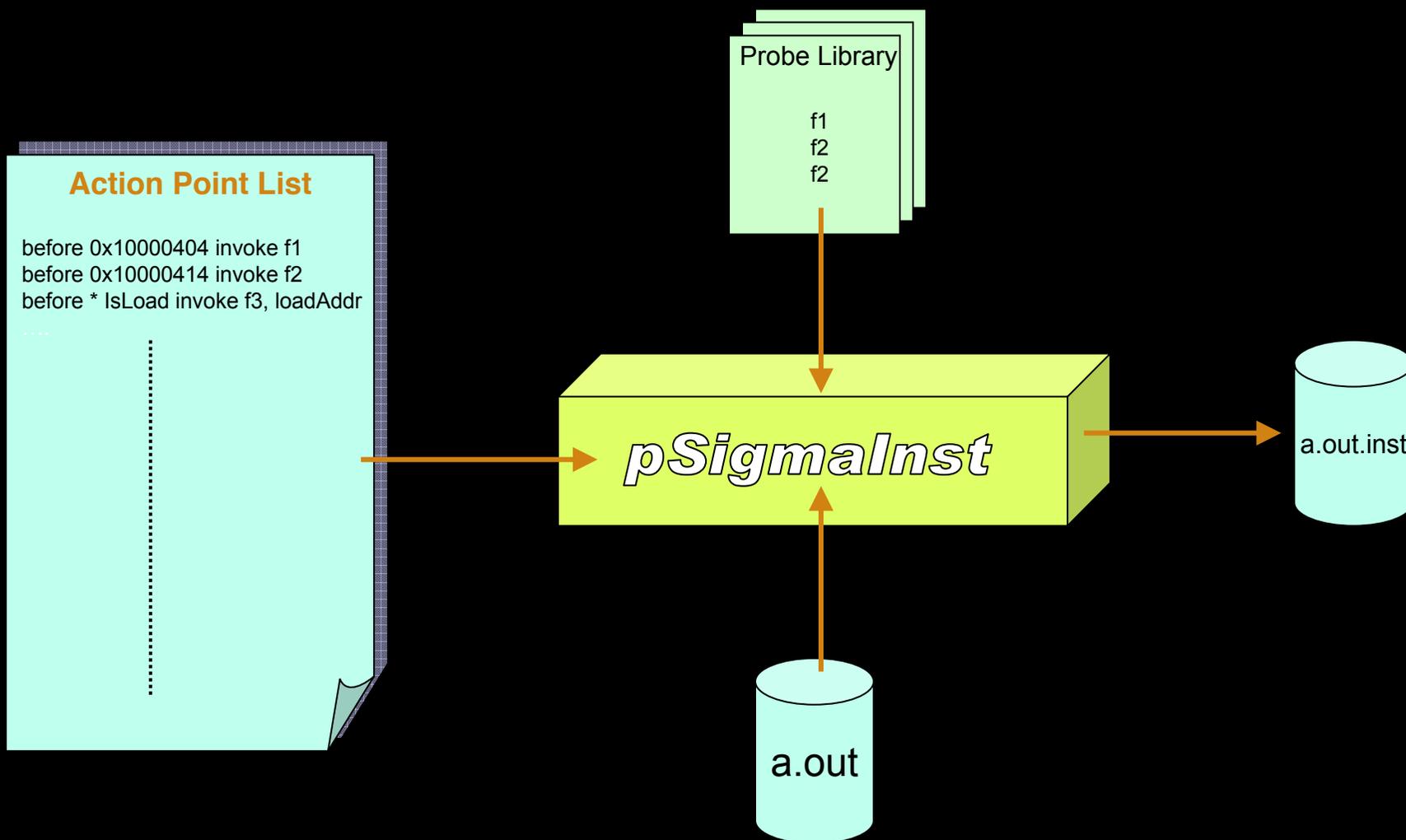
Position = [before | after | replace] instruction

Condition = IsLoad, IsStore, IsBranch, ExCount > N, ...

Action = Activate | Deactivate | invoke FunctionList

FunctionList = (func1, arg1, ...), (func2, arg1, ...), (func3, arg1, ...)

# Action Point Binary Instrumentation



# Action Point Instrumentation Features

- Low-overhead instrumentation
- Support single-threaded and multiple-threaded applications
- Programming Language and Compiler Independent
- Fine-grained (individual instructions)
- Dynamically activate/deactivate through events or signals

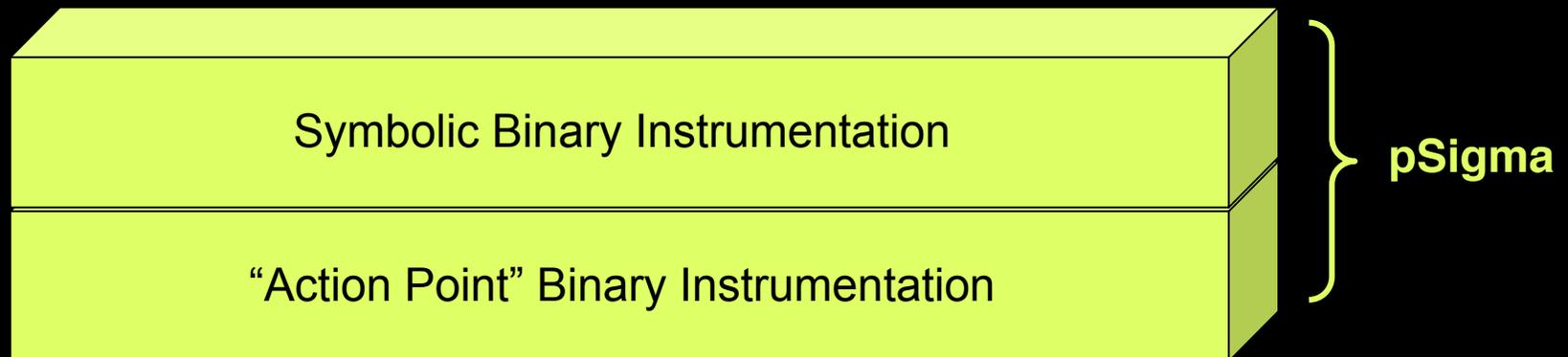
# Problems with Action Point Binary Instrumentation

- **Not performance-oriented:**
  - When the probe is invoked, no information on the state of the system or on the effect of the operation is provided.
  - The description of the rules that should trigger the probe cannot use performance metrics (“invoke the probe on every L1 miss”)
- **Not symbolic:**
  - If a probe function is invoked on a memory operation, there is no information about which data structure/function the mem op refers to.
  - The description of the rules that should trigger the probe cannot use symbolic names in the source program (“invoke the probe every time the array A is touched”)



**DIFFICULT TO DEVELOP PERFORMANCE TOOLS**

# HPC Toolkit Software Stack

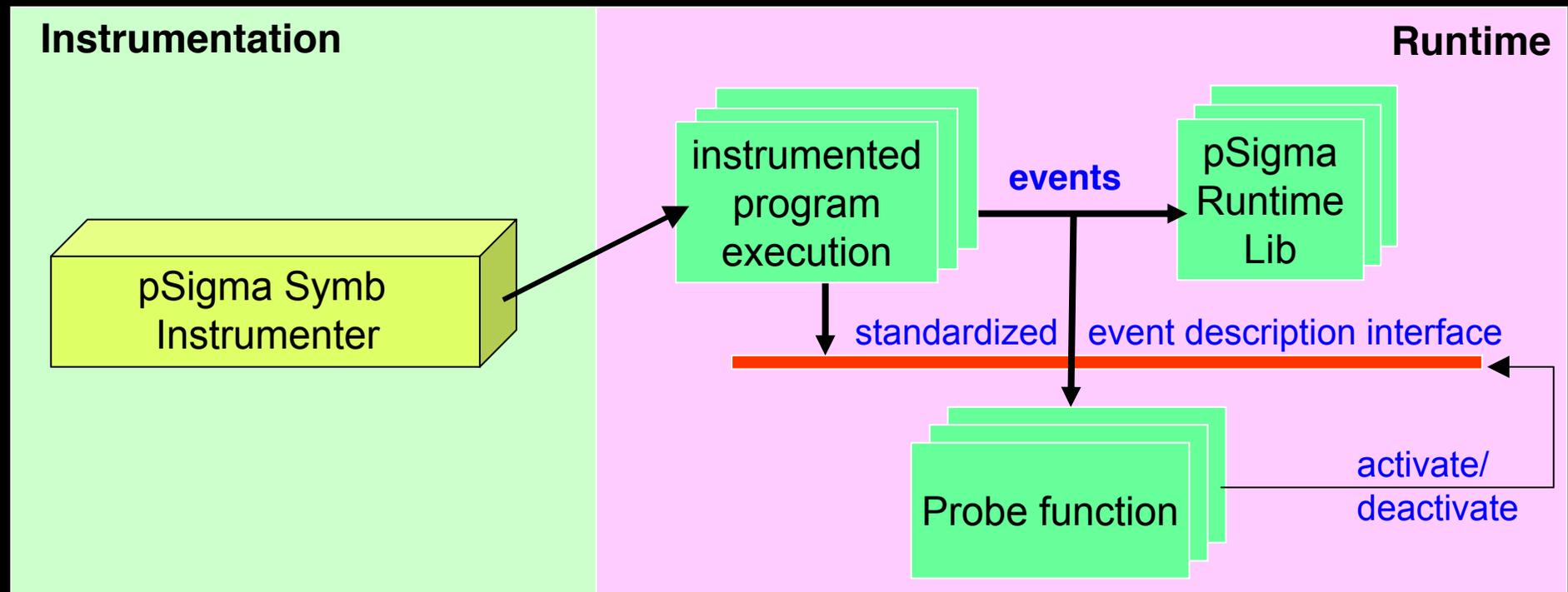


# Symbolic Binary Instrumentation

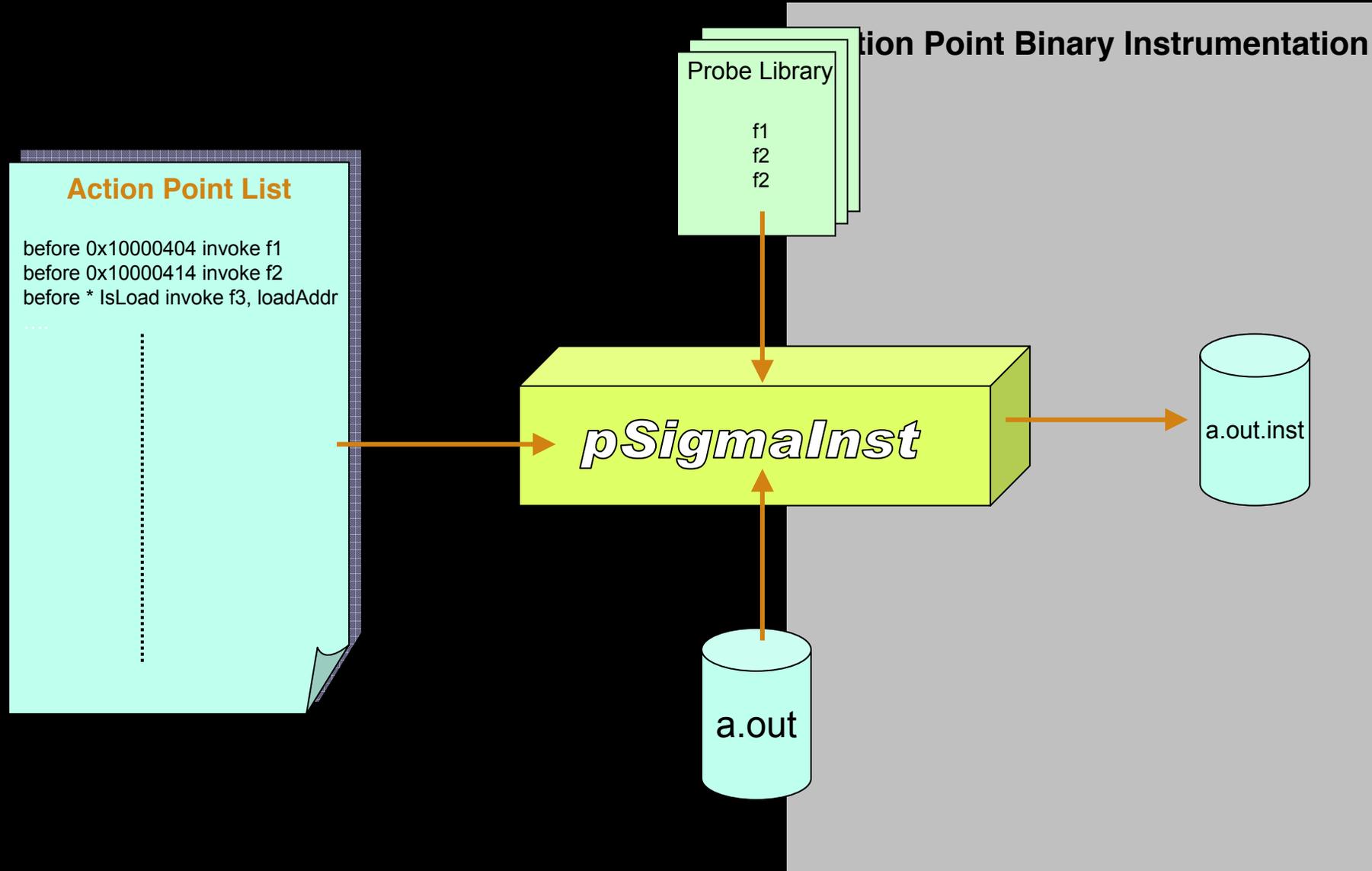
- **Symbolic mapping engine** translates symbolic commands to low-level instructions
- **Builtin cache simulator** allows the use of memory performance metrics in events
  - Instrumentation driven by directives, each specifying an **event** and the corresponding **action**
  - Events can use **symbolic names** and **performance metrics**
  - An event is either a **predicate on the current operation** (eg. operation is a load, or was an L1 miss) or a **predicate on the state of the system** (L1 miss count exceeded a constant)
  - Events can be combined using **logical operators**
  - Events can be qualified by a **context**, if the event is to be considered only for certain data structures/functions.

# Examples: Symbolic Events

- on funcEntry in F do call myHandler
- on (Load for A) or (Store for A) do call myHandler
- on (ExInstructions == 1000) do call myHandler
- on L1Miss for A in F do call myHandler
- on L1Miss for A in F and (L1Misses for A in F > 1000) do call myHandler

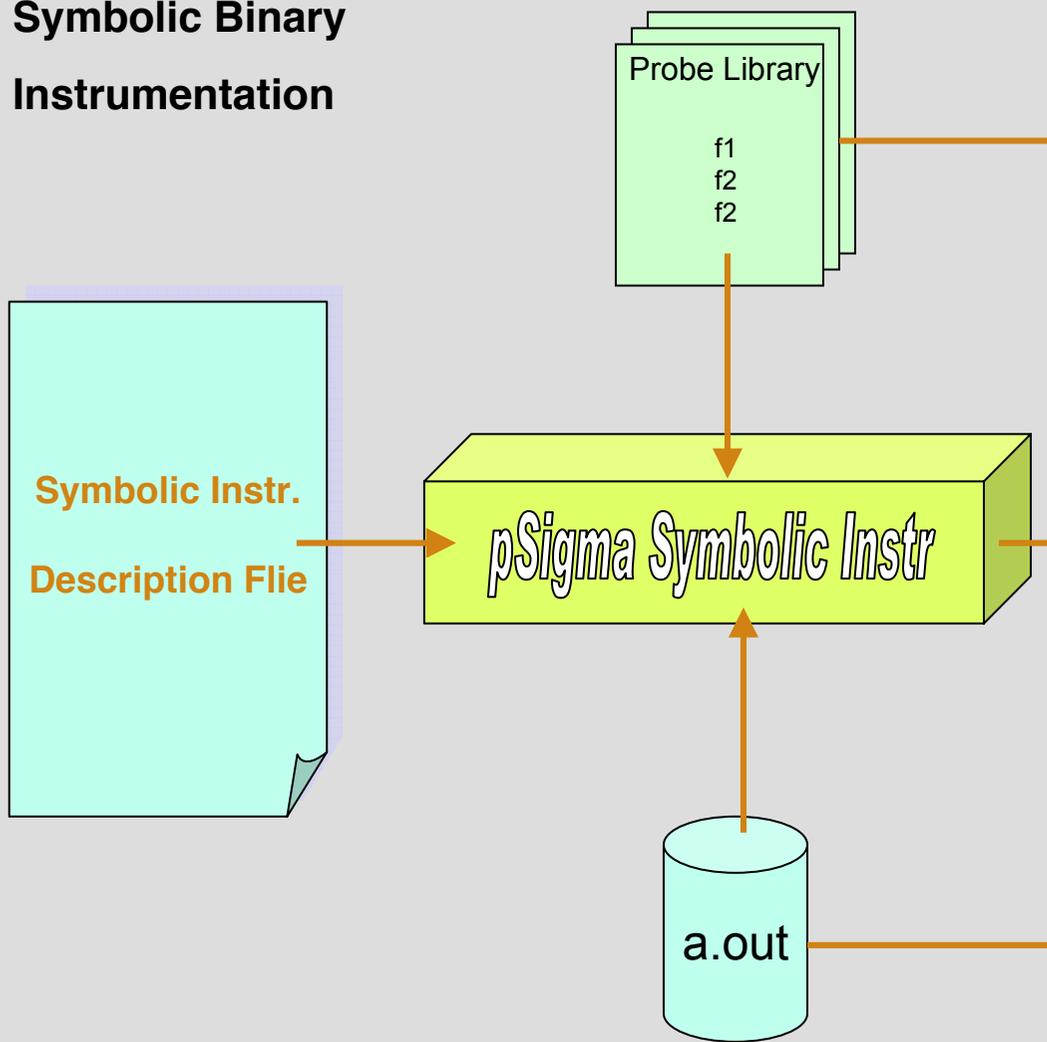


# Symbolic Binary Instrumentation

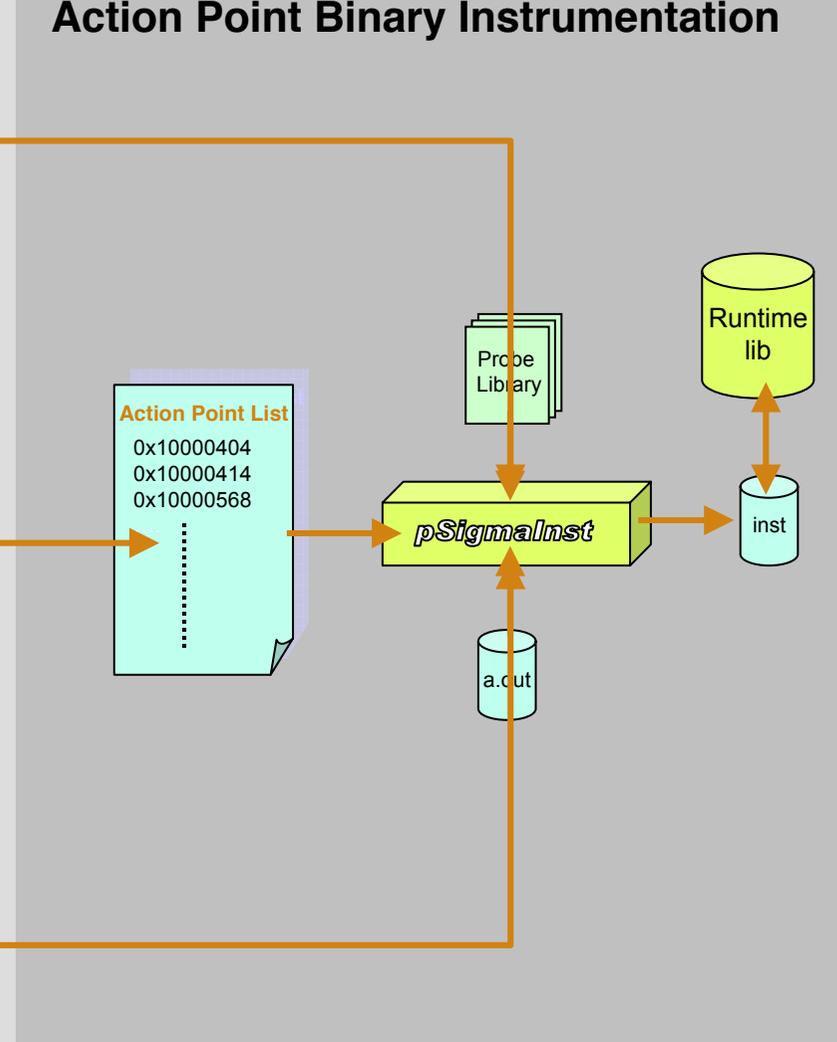


# Symbolic Binary Instrumentation

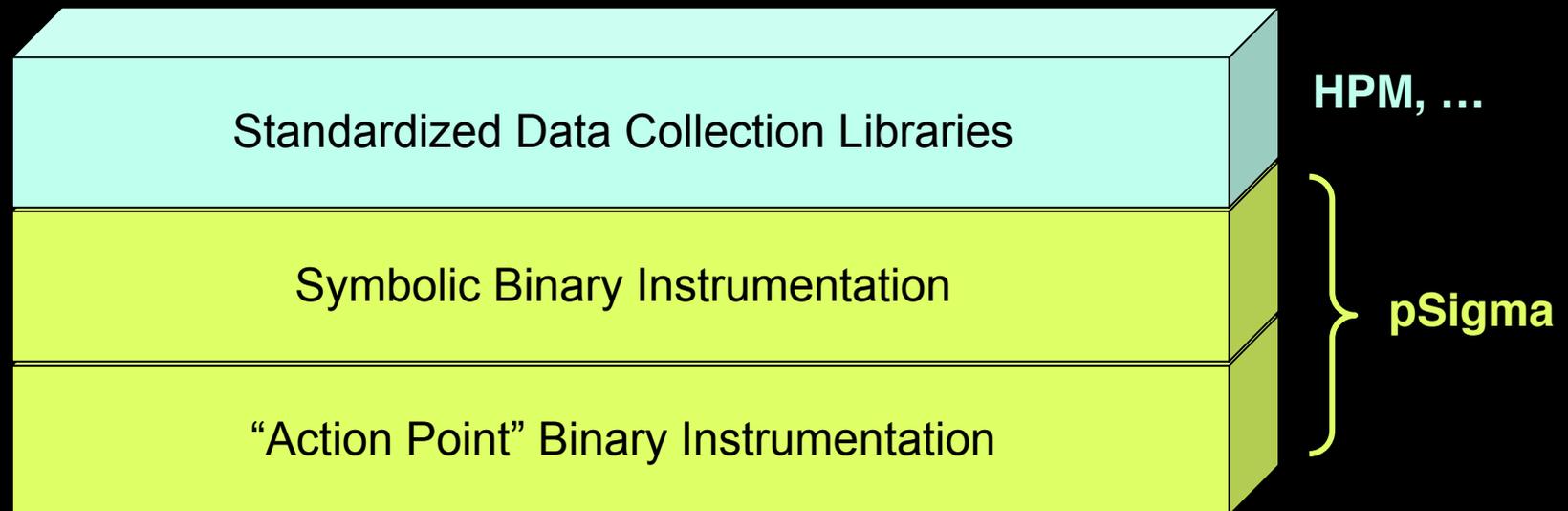
## Symbolic Binary Instrumentation



## Action Point Binary Instrumentation

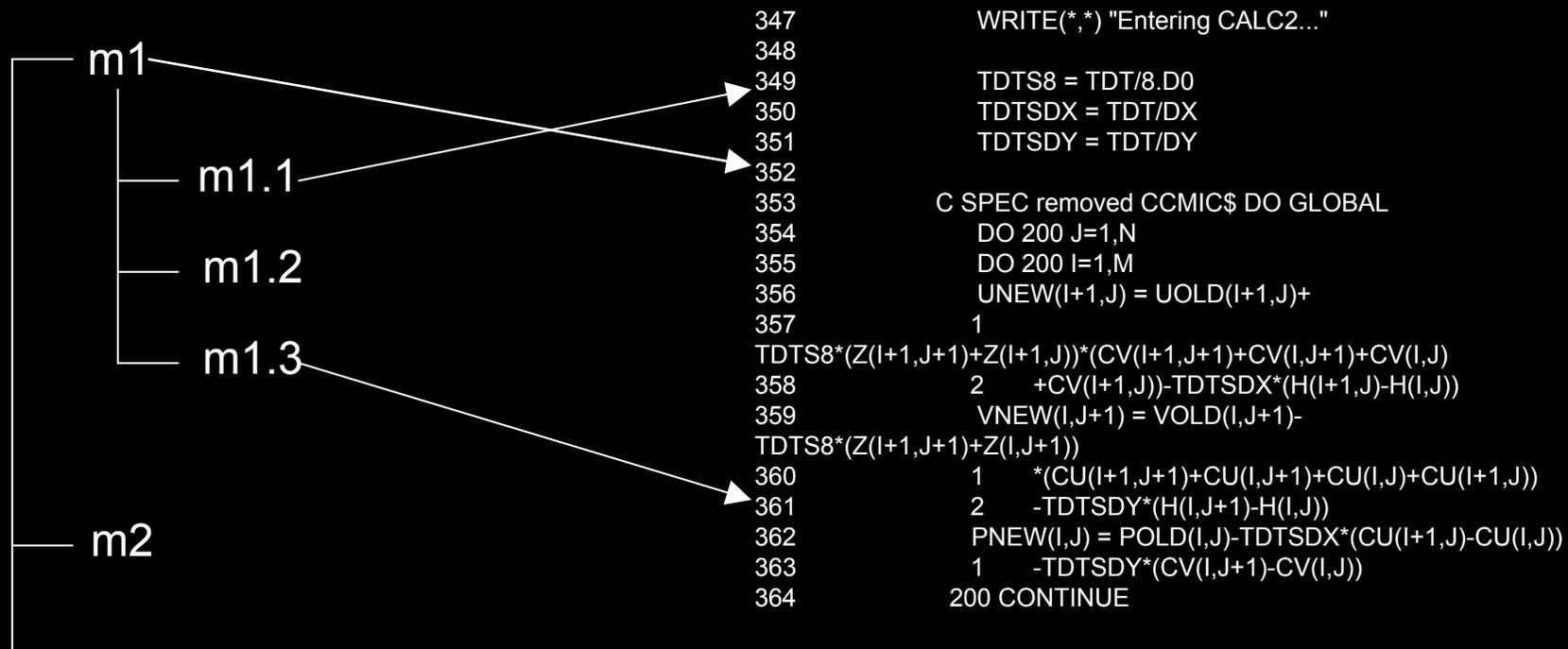


# HPC Toolkit Software Stack



# Standardized Data Collection Library

- Standardized interface for probe libraries:
  - Specify data collection points through **pSigma symbolic language**
  - Output data in a **standardized XML format** which describes:
    - Collected metrics as a **tree**
    - **Mapping** of the metrics to the source code



# PeekPerf: Standardized Data Visualization

The screenshot displays the IBM ACTC PeekPerf interface. The main window shows a tree view of performance metrics for various tasks. The 'calc2' task is selected and highlighted in black. Below this, a 'Metric Browser' window provides a detailed view of cache-related metrics for 'calc2'. To the right, a source code window shows the Fortran code for 'swim.f', with lines 364-369 highlighted in black.

Label	MemTime	Accesses	HitRatio (%)
calc1	2.47503e+06	376425	67.3003
calc1_cu@Global	222048	12669	1.13663
calc1_cv@Global	222748	12669	1.13663
calc1_h@Global	222778	12669	1.11295
calc1_p@Global	454576	107163	77.5809
calc1_u@Global	357670	59535	59.9227
calc1_unknownDt@Global	129806	61659	98.988
calc1_unknownDt@Local	280826	37857	99.4558
calc1_v@Global	361760	59535	59.2357
calc1_z@Global	222820	12669	1.11295
<b>calc2</b>	<b>3.05559e+06</b>	<b>387570</b>	<b>51.9671</b>
calc3	1.1071e+06	106137	42.4056
calc3_p@Global	106496	8192	0
calc3_pnew@Global	95256	7938	0

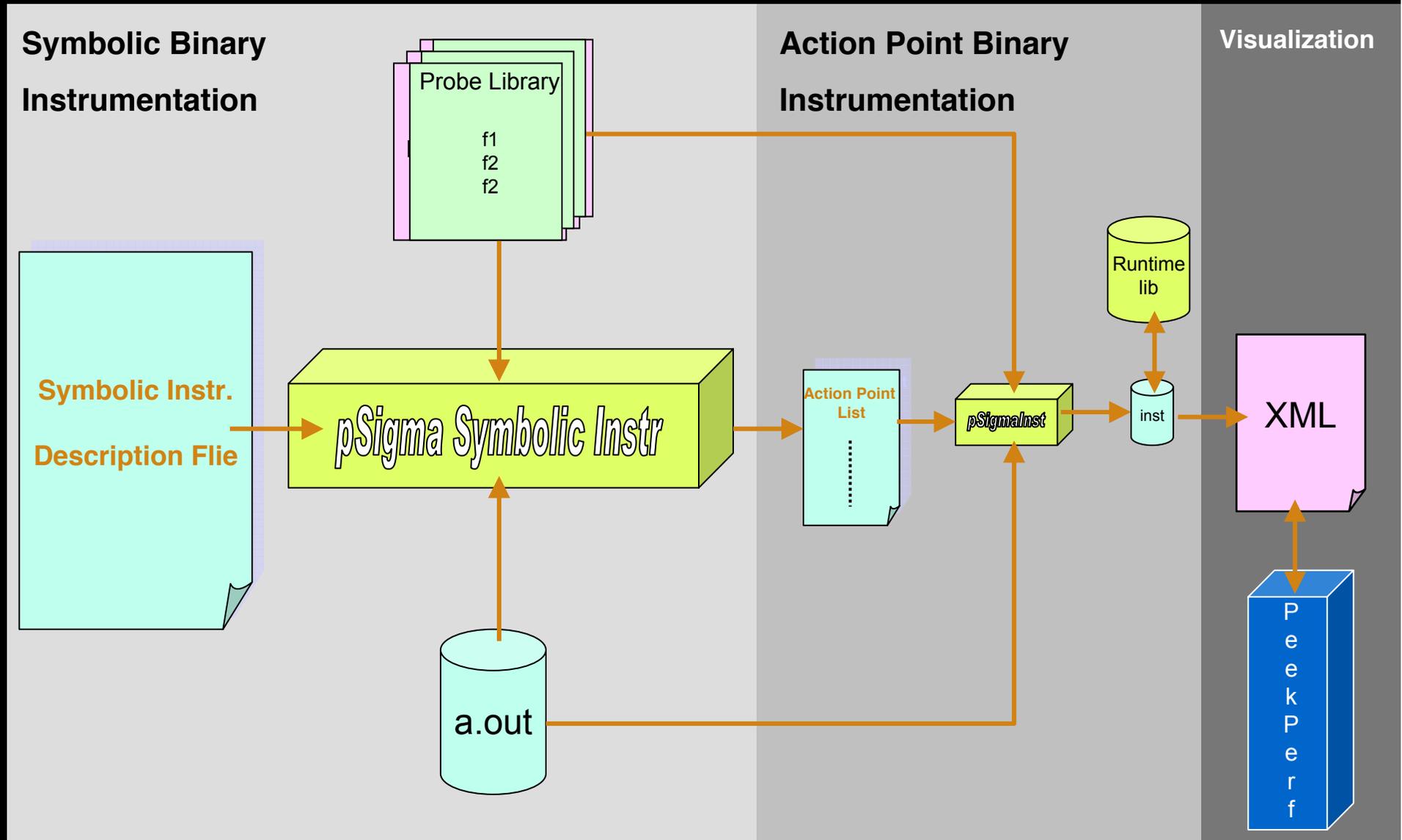
Task	Cache Level	MemTime	Accesses	Hits	Misses	LoadAccesses	LoadHits	LoadMisses	StoreAccesses	StoreMi
0	L1	3.05559e+06	387570	201409	186161	338004	188719	149285	49566	36876
0	L2	0	198851	198083	768	149285	149285	0	49566	768
0	L3	0	768	576	192	0	0	0	768	192
0	TLB	0	387570	387547	23	387570	387547	23	0	0

```

364 200 CONTINUE
365
366 C
367 C PERIODIC CONTINUATION
368 C
369 DO 210 I=1,N
    
```

# Standardized Data Collection Library



## Example Standardized Data Collection Libraries: HPM

### Hardware Counters:

- extra logic inserted in the processor to count specific events
- updated at every cycle
- low-overhead
- processor-specific

- **Cycles**
- **Instructions**
- **Floating point instructions**
- **Integer instructions**
- **Load/stores**
- **Cache misses**
- **TLB misses**
- **Branch taken / not taken**
- **Branch mispredictions**

- **Useful derived metrics**

**IPC - instructions per cycle**

**Float point rate (Mflip/s)**

**Computation intensity**

**Instructions per load/store**

**Load/stores per cache miss**

**Cache hit rate**

**Loads per load miss**

**Stores per store miss**

**Loads per TLB miss**

**Branches mispredicted %**

# HPM Data Visualization

**hpmviz**

---

**File**

swim\_omp | swim\_omp.f | calc1.f | calc2.f | calc3.f

Label	ExcSec	IncSec	Count
Loop 300	4.572	4.572	2398
Loop 200	4.203	4.203	2400
Loop 100	3.071	3.071	2400
Calc3	1.838	6.813	2398
Calc2	1.013	5.632	2400

```

*   VOLD(N1,N2), POLD(N1,N2),
2   CU(N1,N2), CV(N1,N2),
*   Z(N1,N2), H(N1,N2), PSI(N1,N2)
C
COMMON /CONS/ DT,TDT,DX,DY,A,ALPHA,ITMAX,MPRINT,
1   NP1,EL,PI,TPI,DI,DJ,PCF
integer ierr
    
```

- Metric Option:**

  - Count
  - ExcSec
  - IncSec
  - PM\_FPU\_FDIV
  - PM\_FPU\_FMA
  - PM\_FPU0\_FIN
  - PM\_FPU1\_FIN
  - PM\_CYC
  - PM\_FPU\_STF
  - PM\_INST\_CMPL
  - PM\_LSU\_LDF
  - U time
  - Use rate
  - (M) LS
  - MIPS
  - HW FP/Cyc
  - Instr/LS
  - M Flips
  - IpC
  - Mflip/s
  - WFlips
  - Wflip/s
  - FMA %
  - Comp Int.

**Metric Browser: Loop 300**

---

Close | Metric Options | Precision

Node	Thread	Count	ExcSec	IncSec	U time	Use rate	(M) LS	MIPS	HW FP/Cyc	Instr/LS	M Flips	IpC	Mflip/s	WFlips	Wflip/s	FMA %	Comp Int.
0	3	2398	4.539	4.539	3.923	86.425	590.056	291.855	0.116	2.245	589.86	0.26	129.947	589.86	129.947	80.012	1
0	0	2398	4.572	4.572	4.378	95.763	608.414	263.277	0.107	1.978	608.234	0.211	133.037	608.234	133.037	80.011	1
0	2	2398	4.549	4.549	4.366	95.979	590.019	255.241	0.104	1.968	589.838	0.205	129.663	589.838	129.663	80.011	1
0	1	2398	4.547	4.547	4.308	94.759	590.024	259.19	0.105	1.997	589.837	0.21	129.728	589.837	129.728	80.012	1
1	2	2398	4.534	4.534	4.398	96.999	590.044	253.123	0.103	1.945	589.856	0.201	130.088	589.856	130.088	80.012	1
1	1	2398	4.528	4.528	3.942	87.069	589.983	286.058	0.115	2.195	589.807	0.253	130.263	589.807	130.263	80.011	1
1	0	2398	4.547	4.547	3.766	82.828	608.434	308.065	0.124	2.302	608.244	0.286	133.762	608.244	133.762	80.011	1
1	3	2398	4.523	4.523	3.537	78.198	589.962	317.346	0.128	2.433	589.781	0.312	130.4	589.781	130.4	80.011	1
2	0	2398	4.538	4.538	3.777	83.218	608.448	312.01	0.124	2.327	608.262	0.288	134.029	608.262	134.029	80.012	1
2	2	2398	4.522	4.522	4.313	95.364	590.033	257.962	0.105	1.977	589.86	0.208	130.431	589.86	130.431	80.011	1
2	3	2398	4.52	4.52	4.307	95.285	589.985	258.863	0.105	1.983	589.806	0.209	130.492	589.806	130.492	80.011	1
2	1	2398	4.52	4.52	4.35	96.222	589.943	255.814	0.104	1.96	589.767	0.205	130.466	589.767	130.466	80.01	1
3	3	2398	4.487	4.487	4.193	93.453	571.551	259.827	0.105	2.04	571.374	0.214	127.352	571.374	127.352	80.011	1
3	1	2398	4.502	4.502	4.365	96.953	589.937	254.196	0.104	1.94	589.763	0.202	131.003	589.763	131.003	80.01	1
3	2	2398	4.483	4.483	4.139	92.33	571.556	263.864	0.106	2.07	571.38	0.22	127.445	571.38	127.445	80.011	1
3	0	2398	4.506	4.506	3.927	87.154	590.044	290.852	0.116	2.221	589.856	0.257	130.901	589.856	130.901	80.012	1

```

V(I,J) = VNEW(I,J)
P(I,J) = PNEW(I,J)
300 CONTINUE
call f_hpmstop( 30+omp_get_thread_num())
    
```

## Example Standardized Data Collection Libraries: MP\_Profiler

### Profiling tool for MPI applications

- Implements wrappers around MPI calls using the PMPI interface
  - start timer
  - call pmpi equivalent function
  - stop timer
- Captures both “summary” and trace data for MPI calls with source code traceback
- No changes to source code, but MUST compile with -g
- ~1.7 microsecond overhead per MPI call
- Does not synchronize MPI calls

# MP\_Profiler Data Visualization

PeekPerf Main Window

File Tools Options

GAMESS MPI\_Application

gamsess.f ddif

Label	Call Count [Ma
MPI_Allreduce_807	16
MPI_Allreduce_868	38
MPI_Barrier_1496	1
MPI_Barrier_248	1
MPI_Barrier_765	4
MPI_Bcast_924	285
MPI_Comm_rank_973	5
MPI_Comm_size_972	5
MPI_Iprobe_1160	
MPI_Recv_1027	
MPI_Recv_1135	
MPI_Ssend_1002	
Summary_MPI_Allr	
Summary_MPI_Barr	
Summary_MPI_Bcas	
Summary_MPI_Comm	
Summary_MPI_Comm	
Summary_MPI_Ipro	
Summary_MPI_Recv	
Summary_MPI_Sser	

```

OF COMPUTE
C PROCESSES ONLY, NOT THE TOTAL NUMBER OF
PROCESSES.
C
-----
C
      IMPLICIT NONE
      INTEGER DDI_NP, DDI_ME
C
      INCLUDE 'mpif.h'
    
```

Metric Browser: MPI\_Bcast\_924

Close Metric Options Precision

Task	Message Size	WallClock	Count	Call Count [Max]	WallClock [Max]	Transferred Bytes
0	(2) 5 ... 16	0.049632	133	133	0.049632	1064
0	(3) 17 ... 64	0.000144	2	2	0.000144	64
0	(4) 65 ... 256	0.001124	16	16	0.001124	1408
0	(5) 257 ... 1K	0.030647	163	163	0.030647	47408
0	(6) 1K ... 4K	0.011084	134	134	0.011084	198472
0	(7) 4K ... 16K	0.024244	285	285	0.024244	1.69084e+06
0	(8) 16K ... 64K	0.001328	16	16	0.001328	227200
0	(9) 64K ... 256K	0.078051	121	121	0.078051	1.09938e+07
1	(2) 5 ... 16	0.185036	133	133	0.185036	1064
1	(3) 17 ... 64	0.000183	2	2	0.000183	64
1	(4) 65 ... 256	0.009773	16	16	0.009773	1408
1	(5) 257 ... 1K	0.018897	163	163	0.018897	47408
1	(6) 1K ... 4K	0.019423	134	134	0.019423	198472
1	(7) 4K ... 16K	0.251916	285	285	0.251916	1.69084e+06
1	(8) 16K ... 64K	0.406296	16	16	0.406296	227200
1	(9) 64K ... 256K	0.087307	121	121	0.087307	1.09938e+07

## Example Standardized Data Collection Libraries: SiGMA

- Memory profiler based on a simulator
- Control-Centric and Data-Centric
- Experiment with memory performance models
- Ask “what-if” questions regarding data and code rearrangements
- Provide feedback on design of new memory architectures
- Identify performance bottlenecks due to the memory hierarchy and data layout

## SiGMA Memory Profile:

- Execute functional cache simulation and provide **memory profile**
- IBM architectures **prefetching** implemented
- Write-Back/Write-Through caches, **replacement policies** etc

Provide counters such as hits, misses, cold misses for

**each cache level**

**each function**

**each data structure**

**each data structure within each function**

Output sorted by the SIGMA **memtime**:

$$\text{SUM}( \text{LoadHits}(i) * \text{LoadLat}(i) + \text{StoreHits}(i) * \text{StoreLat}(i) ) + \\ \# \text{TLBmisses} * \text{Lat}(\text{TLBmiss})$$

**memtime should track wall time for memory bound applications**

# SiGMA Data Visualization

peekperf
\_ | 5 | X

File Tools

sigma

Label	MemTime /	Access
u@Global	1.57702e+06	216097
u@Global_inital	447628	32511
u@Global_calc3	444718	32258
u@Global_calc3z	433822	32258
u@Global_calc1	250850	119070
v@Global	1.57569e+06	216097
p@Global	1.52905e+06	192786
h@Global	1.30777e+06	168216
z@Global	1.30759e+06	168216
pold@Global	1.27448e+06	112144
vold@Global	1.27217e+06	112144
uold@Global	1.27047e+06	112144
cv@Global	1.26258e+06	145158
cu@Global	1.26066e+06	145158
unew@Global	1.19097e+06	81151
vnew@Global	1.15966e+06	81151
pnew@Global	1.15684e+06	81151
psi@Global	486142	51913
unknownDt@Global	11220	539
unknownDt@Local	424	30

swim.f

```

VOLD(N1,N2), POLD(N1,N2),
2  CU(N1,N2), CV(N1,N2),
*  Z(N1,N2), H(N1,N2), PSI(N1,N2)
C
COMMON /CONS/ DT, TDT,DX,DY,A,ALPHA,ITMAX,MPRINT,M,N,MP1,
1  NP1,EL,PI,TPI,DI,DJ,PCF
C
TDT = TDT+TDT
DO 400 J=1,NP1
DO 400 I=1,MP1
UOLD(I,J) = U(I,J)
VOLD(I,J) = V(I,J)
POLD(I,J) = P(I,J)
U(I,J) = UNEW(I,J)
V(I,J) = VNEW(I,J)
P(I,J) = PNEW(I,J)
400 CONTINUE
RETURN
END
C SPEC removed CCMIC$ MICRO
SUBROUTINE CALC3
C
C TIME SMOOTHER
C
IMPLICIT REAL*8 (A-H, O-Z)
#include "swim.h"
COMMON U(N1,N2) V(N1,N2) P(N1,N2)
                
```

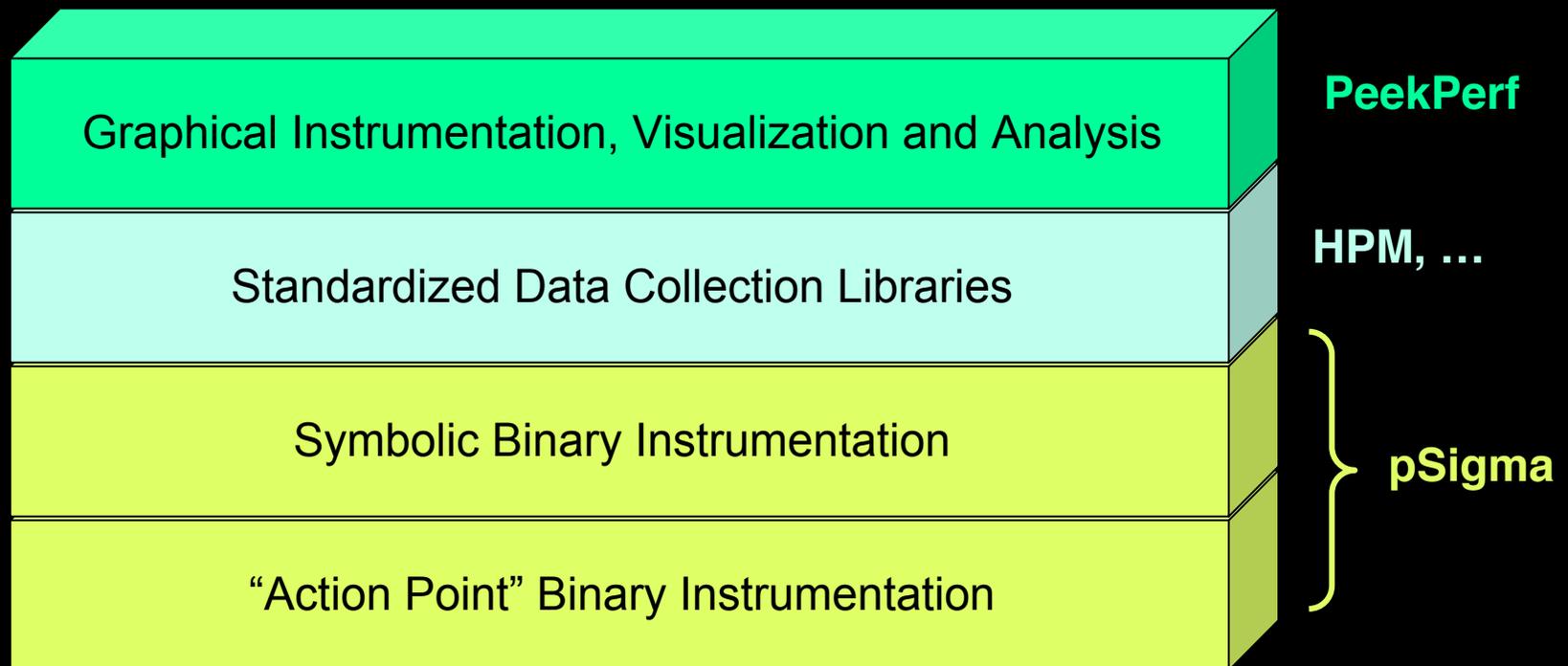
Metric Browser: u@Global\_calc3

Close
Metric Options
Precision

Task	thread	Misses	LoadMisses	StoreMisses	Hits	LoadHits	StoreHits	Access	LoadAccesses	StoreAccess
0	L1	442	193	249	31816	15936	15880	32258	16129	16129
0	L2	190	74	116	16132	119	16013	16322	193	16129
0	L3	0	0	0	1191	74	1117	1191	74	1117
0	TLB	0	0	0	32258	32258	0	32258	32258	0

C SPEC removed CCMIC\$ DO GLOBAL

# HPC Toolkit Software Stack



# PeekPerf GUI for Instrumentation

- Control instrumentation from PeekPerf: one **complete framework** for performance analysis
- Operate on the **source code** but perform modifications on the **binary**
- **Debugger-like** interface
- **Automatically** display collected data
- **Refine** instrumentation (iterative tuning)
- **Comparison** between data and between multiple runs
- Graphics capabilities (tables, charts)
- Query language for “what-if” analysis

# PeekPerf GUI for Instrumentation

The screenshot displays the IBM ACTC PeekPerf Main Window. The interface is divided into several sections:

- File Tools Options Action**: The top menu bar.
- HPM | MPI | SIGMA | DPOMP**: A set of tabs for different instrumentation methods.
- Name**: A tree view showing the hierarchy of MPI operations. The 'MPI\_Isend' operation is highlighted.
- swim\_omp**: A table showing performance metrics for various operations. The table has columns for Label, Count, ExcSec, and IncSec.
- Code Editor**: A window showing the source code of the application, with line numbers 101 through 147 visible. The code includes MPI communication calls and a subroutine named CALC3Z.

Label	Count	ExcSec	IncSec
-ALL	1	0.006	1.062
-Calc1	102	0.006	0.263
-Calc2	102	0.072	0.373
-Calc3	100	0.098	0.379
-Inital	1	0.042	0.042
-Loop 100	102	0.119	0.119
-Loop 200	102	0.178	0.178
-Loop 300	100	0.209	0.209
-MPI Calc1 end	102	0.08	0.08
-MPI Calc1 start	102	0.05	0.05
-MPI Calc2 end	102	0.068	0.068
-MPI Calc2 start	102	0.055	0.055
-MPI in Calc3	100	0.072	0.072
-loop 110	102	0.009	0.009

```

101 1 0,2,MPI_COMM_WORLD,req(2),ierr)
102  CALL mpi_irecv(p(m+1,n+1),1,MPI_DOUBLE_PRECISIO
103  1 0,3,MPI_COMM_WORLD,req(3),ierr)
104  CALL mpi_irecv(vold(m+1,n+1),1,MPI_DOUBLE_PRECI
105  1 0,4,MPI_COMM_WORLD,req(4),ierr)
106  CALL mpi_irecv(uold(m+1,n+1),1,MPI_DOUBLE_PRECI
107  1 0,5,MPI_COMM_WORLD,req(5),ierr)
108  CALL mpi_irecv(pold(m+1,n+1),1,MPI_DOUBLE_PRECI
109  1 0,6,MPI_COMM_WORLD,req(6),ierr)
110  endif
111  if(taskid.eq.0)then
112  CALL mpi_isend(v(1,1),1,MPI_DOUBLE_PRECISION,
113  1 numtasks-1,1,MPI_COMM_WORLD,req(7),ierr)
114  CALL mpi_isend(u(1,1),1,MPI_DOUBLE_PRECISION,
115  1 numtasks-1,2,MPI_COMM_WORLD,req(8),ierr)
116  CALL mpi_isend(p(1,1),1,MPI_DOUBLE_PRECISION,
117  1 numtasks-1,3,MPI_COMM_WORLD,req(9),ierr)
118  CALL mpi_isend(vold(1,1),1,MPI_DOUBLE_PRECISION
119  1 numtasks-1,4,MPI_COMM_WORLD,req(10),ierr)
120  CALL mpi_isend(uold(1,1),1,MPI_DOUBLE_PRECISION
121  1 numtasks-1,5,MPI_COMM_WORLD,req(11),ierr)
122  CALL mpi_isend(pold(1,1),1,MPI DOUBLE PRECISION
123  1 numtasks-1,6,MPI_COMM_WORLD,req(12),ierr)
124  endif
125  if(taskid.eq.0.or.taskid.eq.numtasks-1)then
126  CALL MPI_WAITALL(12,req,istat,ierr)
127  endif
128  call f_hpmstop( 17 )
129
130  C
131
132  RETURN
133  END
134
135  C
136  C      TIME SMOOTHER FOR FIRST ITERATION
137  C
138  PARAMETER (N1=513, N2=513)
139
140  include "mpif.h"
141  COMMON/decomp/js,je,taskid,numtasks,req(16),
142  1 istat(MPI_STATUS_SI
143  integer taskid,req
144  COMMON U(N1,N2), V(N1,N2), P(N1,N2),
145  * UNEW(N1,N2), VNEW(N1,N2),
146  1 PNEW(N1,N2), UOLD(N1,N2),
147  * VOLD(N1,N2), POLD(N1,N2),

```

# PeekPerf: Graphical Instrumentation, Visualization and Analysis

**Instrumentation**

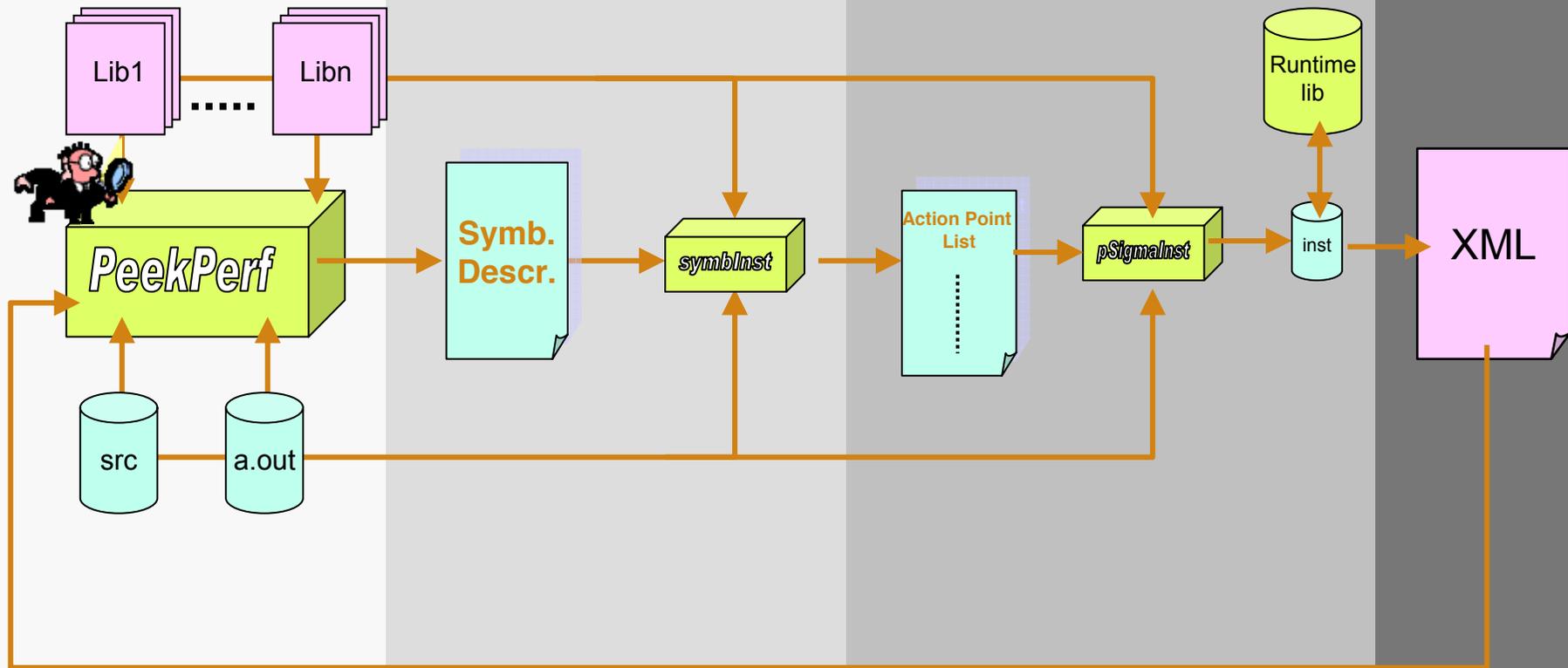
**Visualization**

**Analysis**

**Symbolic Binary Instrumentation**

**Action Point Binary Instrumentation**

**Visualization**



# HPC Toolkit for Productivity

- “End-to-End” Optimization of Application Performance-Tuning Cycle

## **Completely Binary Approach**

Programmable and dynamic, yet without the need for costly and error-prone source code modification.

## **Data Centric Analysis (DCA)**

Performance information is directly related to the application data structures.

Critical to understanding data movement and memory-related performance of shared memory hierarchical systems intended for HPCS.

## **Alternate-Scenario Prediction (ASP)**

Fast and automated path to testing effects of altering data structures and/or code structure, without having to change any source code.

Examples:

- Data-Structure Layout
- Order of a parallel computation, scheduling of threads, etc.

Query language and what-if analysis

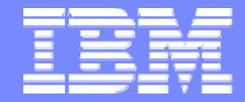
## **User-Controlled Automation (UCA)**

Selective removal of programmer from tuning cycle

Reduces complexity of the system to the application developer.

## Summary:

- The IBM HPC Toolkit provides an **integrated framework** for performance analysis
- Support **iterative analysis** and **automation** of the performance tuning process
- The standardized software layers make it easy to **plug in** new performance analysis tools
- Operates on the **binary** and yet provide reports in terms of source-level symbols
- Provides **multiple layers** that the user can exploit (from low-level instrumentations to high-level performance analysis)
- Full source code **traceback** capability
- **Dynamically** activate/deactivate data collection and change what information to collect



# Questions / Comments