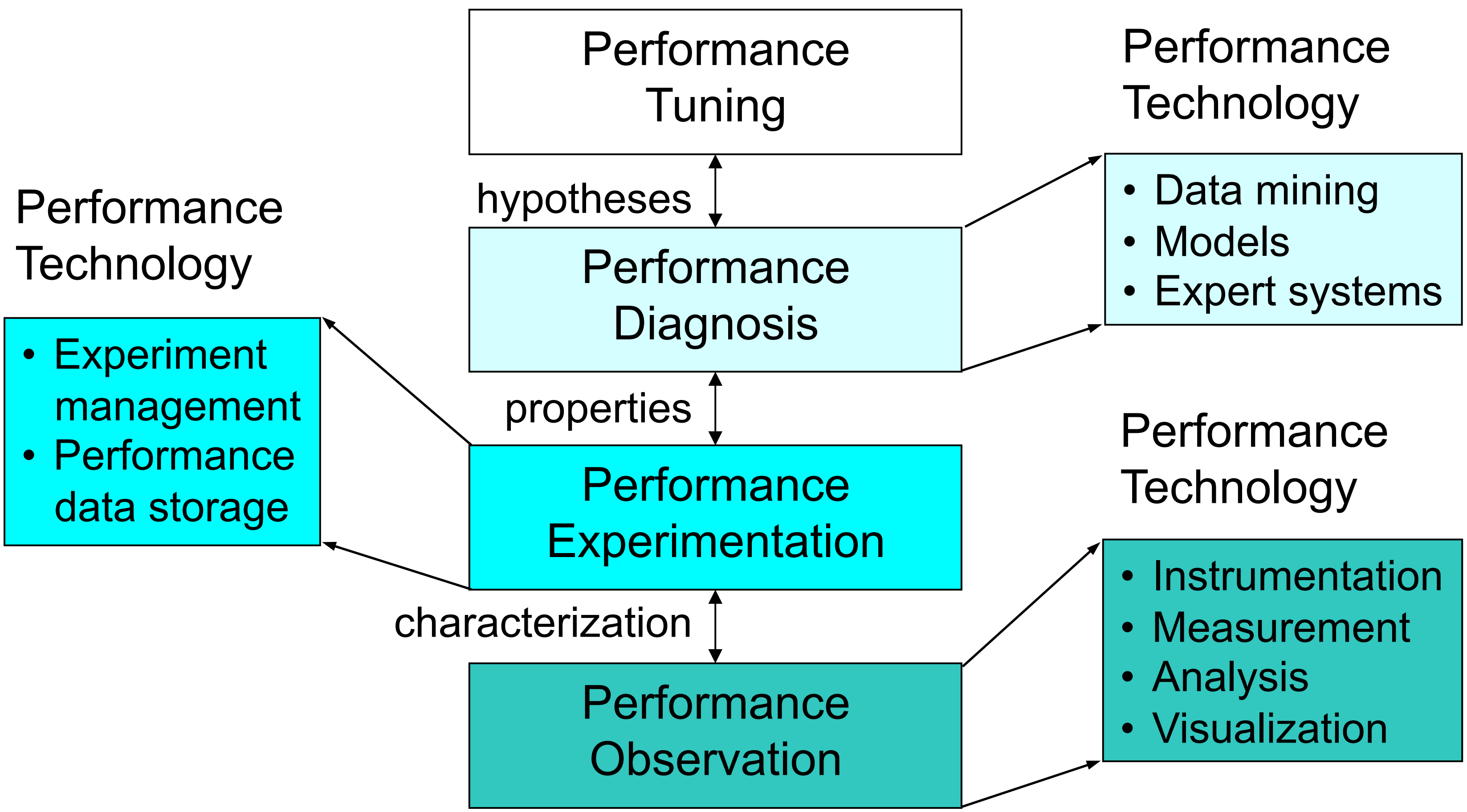
Virtual Institute – High Productivity Supercomputing

Introduction to Parallel Performance Analysis and Engineering

Shirley Moore
shirley@eecs.utk.edu

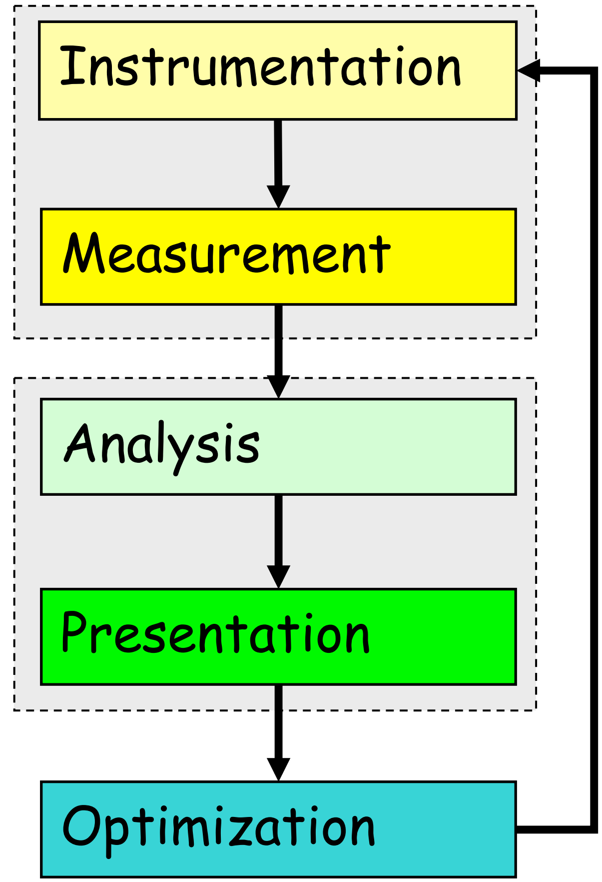
8th VI-HPS Tuning Workshop
5-9 September 2011

- Optimization process
- Effective use of performance technology



Performance Optimization Cycle

- Design experiment
- Collect performance data
- Calculate metrics
- Analyze results
- Visualize results
- Identify bottlenecks and causes
- Tune performance



- Parallel code performance is influenced by both sequential and parallel factors
- Sequential factors
  - Computation
  - Cache and memory use
  - Input/output
- Parallel factors
  - Thread / process interactions
  - Communication and synchronization

- Understanding performance requires observation of performance properties.
- Performance tools and methodologies are primarily distinguished by what observations are made and how.
  - How application program is instrumented
  - What performance data are obtained
- Tools and methods cover broad range.

- Observability depends on measurement
- A metric represents a type of measured data
  - Count, time, hardware counters
- A measurement records performance data
  - Associated with application program static or dynamic execution portions
- Derived metrics are computed
  - Rates (e.g., flops)
- Metrics and measurements dictated by model or experiment

Execution Time

- Wallclock time
  - Based on realtime clock
- Virtual process time
  - Time when process is executing
    - User time and system time
  - Does not include time when process is stalled
- Parallel execution time
  - Runs whenever any parallel part is executing
  - Global time basis

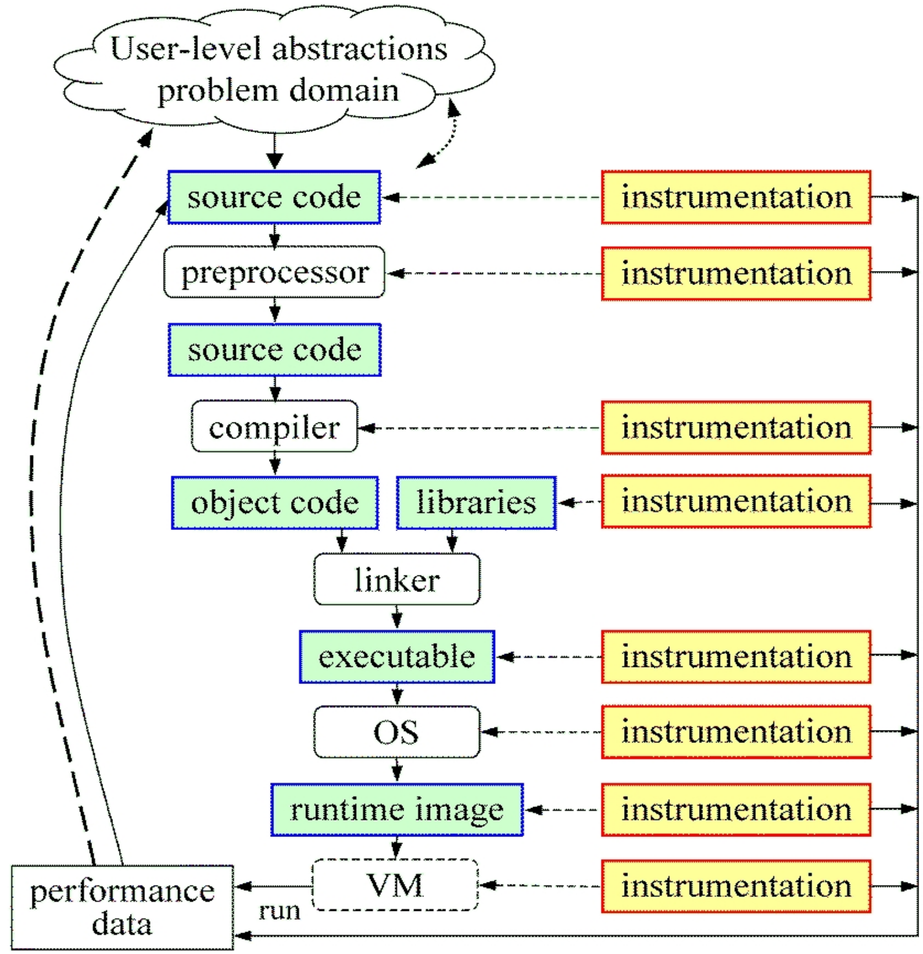
- Execution *actions* exposed as *events*
  - In general, actions reflect some execution state
    - presence at a code location or change in data
    - occurrence in parallelism context (thread of execution)
  - Events encode actions for observation
- Observation is *direct*
  - Direct instrumentation of program code (probes)
  - Instrumentation invokes performance measurement
  - Event measurement = performance data + context
- Performance experiment
  - Actual events + performance measurements

- Program code instrumentation is not used
- Performance is observed indirectly
  – Execution is interrupted
    - can be triggered by different events
  – Execution state is queried (sampled)
    - different performance data measured
  – *Event-based sampling* (EBS)
- Performance attribution is inferred
  – Determined by execution context (state)
  – Observation resolution determined by interrupt period
  – Performance data associated with context for period

- Events defined by instrumentation access
- Instrumentation levels
  - Source code
  - Object code
  - Runtime system
  - Library code
  - Executable code
  - Operating system
- Different levels provide different information
- Different tools needed for each level
- Levels can have different granularity

- Static instrumentation
  - Program instrumented prior to execution
- Dynamic instrumentation
  - Program instrumented at runtime
- Manual and automatic mechanisms
- Tools required for automatic support
  - Source time: preprocessor, translator, compiler
  - Link time: wrapper library, preload
  - Execution time: binary rewrite, dynamic
- Advantages / disadvantages

- Associate performance data with high-level semantic abstractions
- Abstract events at user-level provide semantic context



Indirect Observation: Events/Triggers

- Events are actions external to program code
  - Timer countdown, HW counter overflow, …
  - Consequence of program execution
  - Event frequency determined by:
    - Type, setup, number enabled (exposed)
- Triggers used to invoke measurement tool
  - Traps when events occur (interrupt)
  - Associated with events
  - May add differentiation to events

- When events trigger, execution context determined at time of trap (interrupt)
  - Access to PC from interrupt frame
  - Access to information about process/thread
  - Possible access to call stack
    - requires call stack unwinder
- Assumption is that the context was the same during the preceding period
  - Between successive triggers
  - Statistical approximation valid for long running programs

Direct / Indirect Comparison

- Direct performance observation
  - ☺ Measures performance data exactly
  - ☺ Links performance data with application events
  - 😐 Requires instrumentation of code
  - ☹ Measurement overhead can cause execution intrusion and possibly performance perturbation
- Indirect performance observation
  - ☺ Argued to have less overhead and intrusion
  - ☺ Can observe finer granularity
  - ☺ No code modification required (may need symbols)
  - ☹ Inexact measurement and attribution without hardware support

- When is measurement triggered?
  - External agent (indirect, asynchronous)
    - interrupts, hardware counter overflow, …
  - Internal agent (direct, synchronous)
    - through code modification
- How are measurements made?
  - Profiling
    - summarizes performance data during execution
    - per process / thread and organized with respect to context
  - Tracing
    - trace record with performance data and timestamp
    - per process / thread

Measured Performance

- Counts
- Durations
- Communication costs
- Synchronization costs
- Memory use
- Hardware counts
- System calls

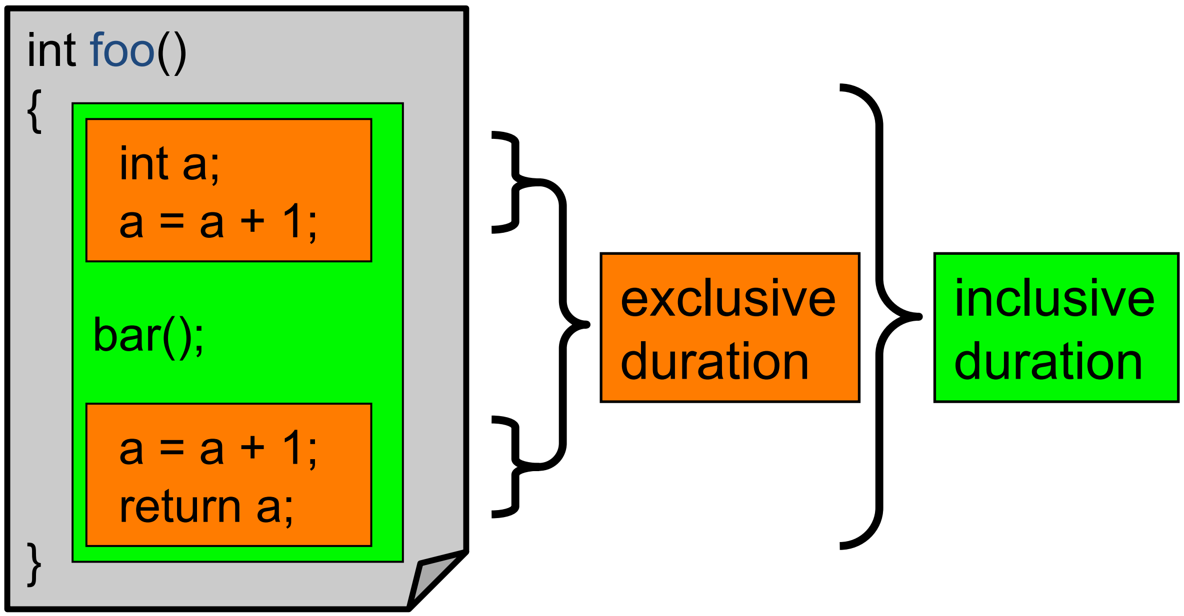
Critical issues

- Accuracy
  - Timing and counting accuracy depends on resolution
  - Any performance measurement generates overhead
    - Execution on performance measurement code
  - Measurement overhead can lead to intrusion
  - Intrusion can cause perturbation
    - alters program behavior
- Granularity
  - How many measurements are made
  - How much overhead per measurement
- Tradeoff (general wisdom)
  - Accuracy is inversely correlated with granularity

Profiling

- Recording of aggregated information
  - Counts, time, …
- … about program and system entities
  - Functions, loops, basic blocks, …
  - Processes, threads
- Methods
  - Event-based sampling (indirect, statistical)
  - Direct measurement (deterministic)

- Performance with respect to code regions
- Exclusive measurements for region only
- Inclusive measurements includes child regions



- Static call graph
  - Shows all parent-child calling relationships in a program
- Dynamic call graph
  - Reflects actual execution time calling relationships
- Flat profile
  - Performance metrics for when event is active
  - Exclusive and inclusive
- Callpath profile
  - Performance metrics for calling path (event chain)
  - Differentiate performance with respect to program execution state
  - Exclusive and inclusive

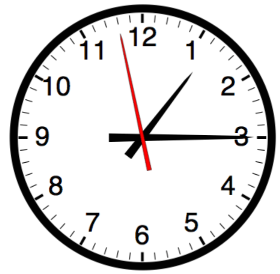
Trace File Generation

Process A:

```
void master {
  trace(ENTER, 1);
  ...
  trace(SEND, B);
  send(B, tag, buf);
  ...
  trace(EXIT, 1);
}
```

Process B:

```
void worker {
 trace(ENTER, 2);
 ...
 recv(A, tag, buf);
 trace(RECV, A);
 ...
 trace(EXIT, 2);
}
```



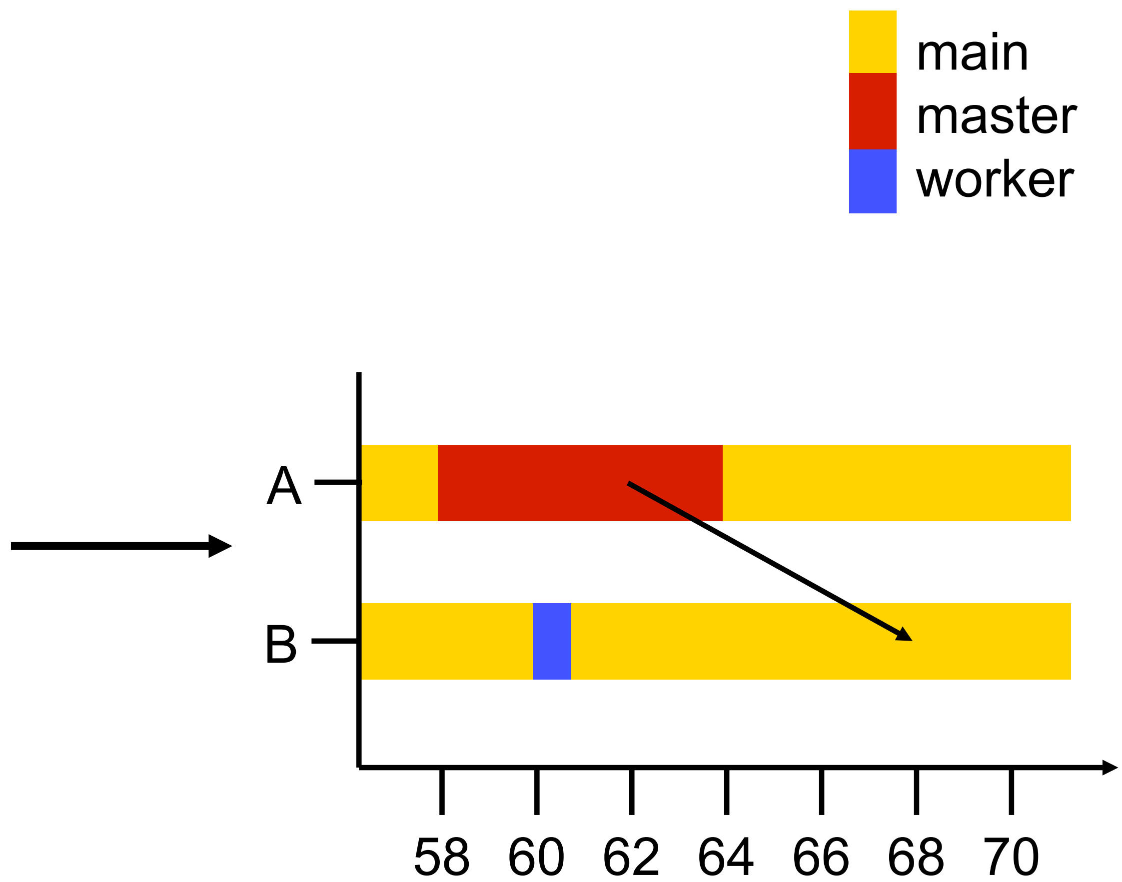
MONITOR

1	master
2	worker
3	...

...			
58	A	ENTER	1
60	B	ENTER	2
62	A	SEND	B
64	A	EXIT	1
68	B	RECV	A
69	B	EXIT	2
...			

1	master
2	worker
3	...

...			
58	A	ENTER	1
60	B	ENTER	2
62	A	SEND	B
64	A	EXIT	1
68	B	RECV	A
69	B	EXIT	2
...			



Trace Formats

- Different tools produce different formats
  - Differ by event types supported
  - Differ by ASCII and binary representations
    - Vampir Trace Format (VTF)
    - KOJAK (EPILOG)
    - Jumpshot (SLOG-2)
    - Paraver
- Open Trace Format (OTF)
  - Supports interoperation between tracing tools

- Profiling
  - ☺ Finite, bounded performance data size
  - ☺ Applicable to both direct and indirect methods
  - 😐 Loses time dimension (not entirely)
  - ☹ Lacks ability to fully describe process interaction
- Tracing
  - ☺ Temporal and spatial dimension to performance data
  - ☺ Capture parallel dynamics and process interaction
  - 😐 Some inconsistencies with indirect methods
  - ☹ Unbounded performance data size (large)
  - ☹ Complex event buffering and clock synchronization

Performance Analysis Questions

- How does performance vary with different compilers?
- Is poor performance correlated with certain OS features?
- Has a recent change caused unanticipated performance?
- How does performance vary with MPI variants?
- Why is one application version faster than another?
- What is the reason for the observed scaling behavior?
- Did two runs exhibit similar performance?
- How are performance data related to application events?
- Which machines will run my code the fastest and why?
- Which benchmarks predict my code performance best?

- Performance diagnosis and optimization involves multiple performance experiments
- Support for common performance data management tasks augments tool use
  - Performance experiment data and metadata storage
  - Performance database and query
- What type of performance data should be stored?
  - Parallel profiles or parallel traces
  - Storage size will dictate
  - Experiment metadata helps in meta analysis tasks
- Serves tool integration objectives

- Integration of metadata with each parallel profile
  - Separate information from performance data
- Three ways to incorporate metadata
  - Measured hardware/system information
    - CPU speed, memory in GB, MPI node IDs, …
  - Application instrumentation (application-specific)
    - Application parameters, input data, domain decomposition
    - Capture arbitrary name/value pair and save with experiment
  - Data management tools can read additional metadata
    - Compiler flags, submission scripts, input files, …
    - Before or after execution
- Enhances analysis capabilities

Performance Data Mining

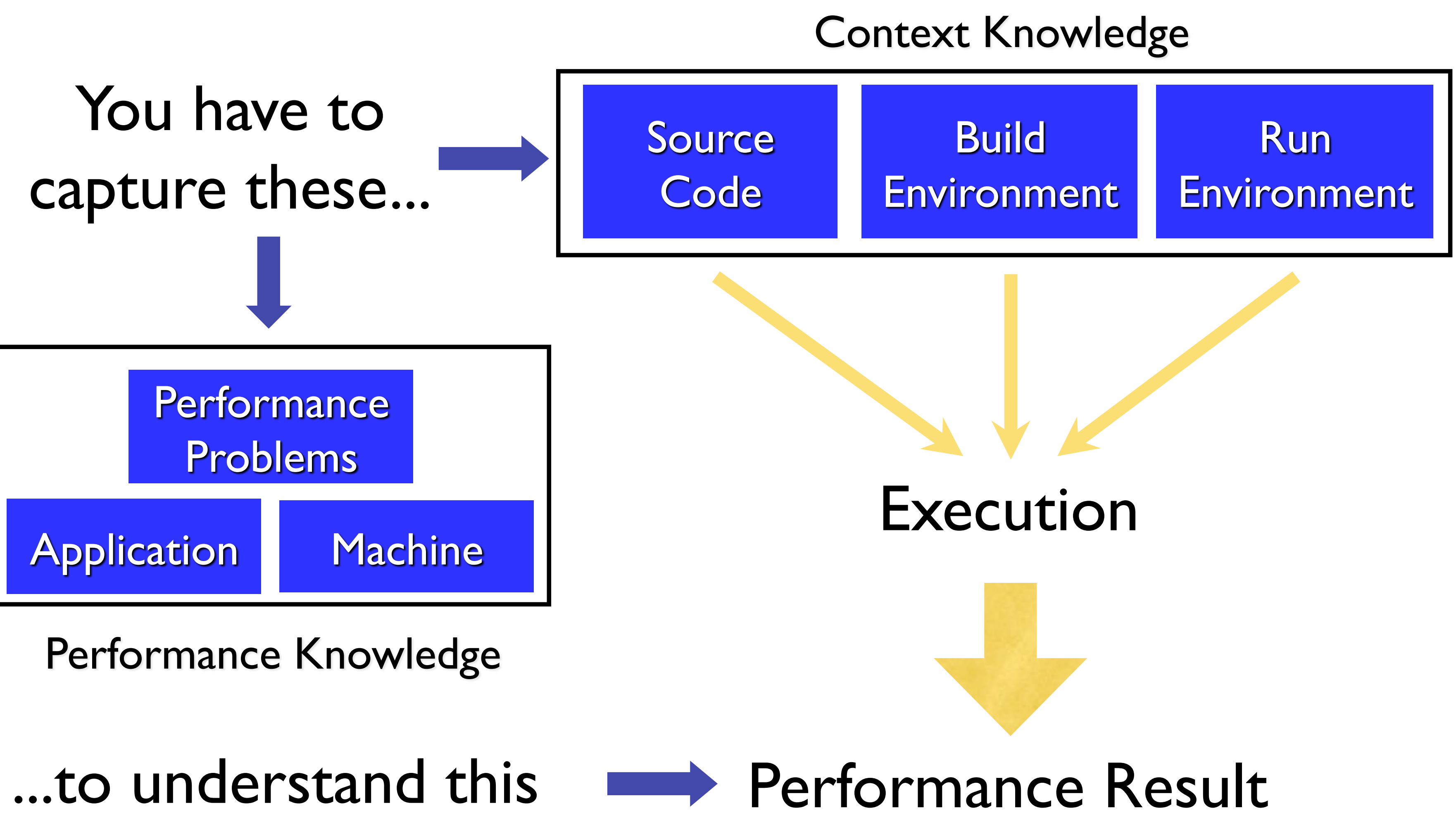
- Conduct parallel performance analysis in a systematic, collaborative and reusable manner
  - Manage performance complexity and automate process
  - Discover performance relationship and properties
  - Multi-experiment performance analysis
- Data mining applied to parallel performance data
  - Comparative, clustering, correlation, characterization, …
  - Large-scale performance data reduction
- Implement extensible analysis framework
  - Abstraction / automation of data mining operations
  - Interface to existing analysis and data mining tools

How to explain performance?

- Should not just redescribe performance results
- Should explain performance phenomena
  - What are the causes for performance observed?
  - What are the factors and how do they interrelate?
  - Performance analytics, forensics, and decision support
- Add *knowledge* to do more intelligent things
  - Automated analysis needs informed feedback
  - Performance model generation requires interpretation
- Performance knowledge discovery framework
  - Integrating meta-information
  - Knowledge-based performance problem solving

Metadata and Knowledge Role

VI-HPS



- Performance characterization
  - Identify major performance contributors
  - Identify sources of performance inefficiency
  - Utilize timing and hardware measures
- Performance diagnosis (Performance Debugging)
  - Look for conditions of performance problems
  - Determine if conditions are met and their severity
  - What and where are the performance bottlenecks
- Performance tuning
  - Focus on dominant performance contributors
  - Eliminate main performance bottlenecks