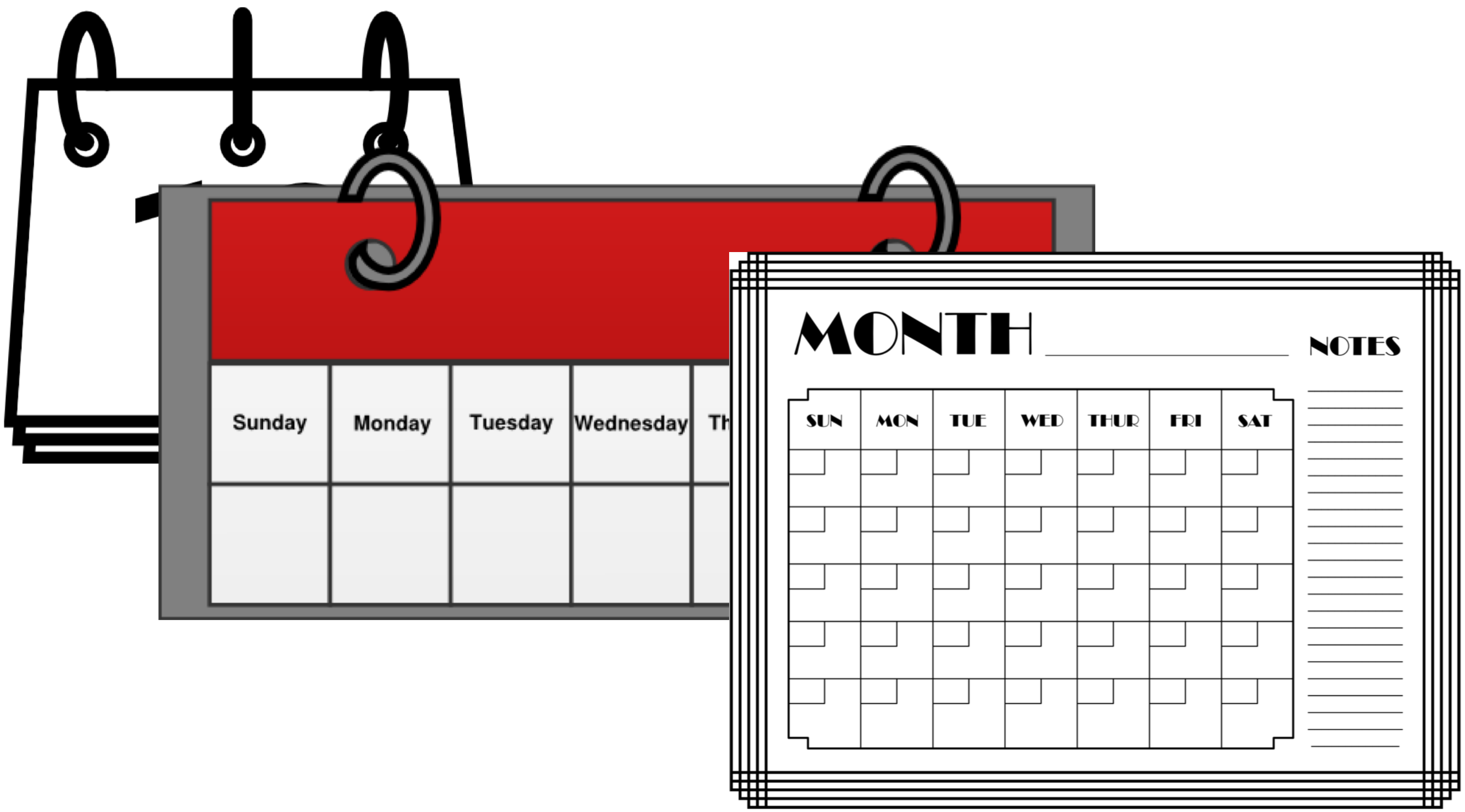
Sledgehammer: Cluster-Fueled Debugging

Andrew Quinn, Jason Flinn, and Michael Cafarella

University of Michigan

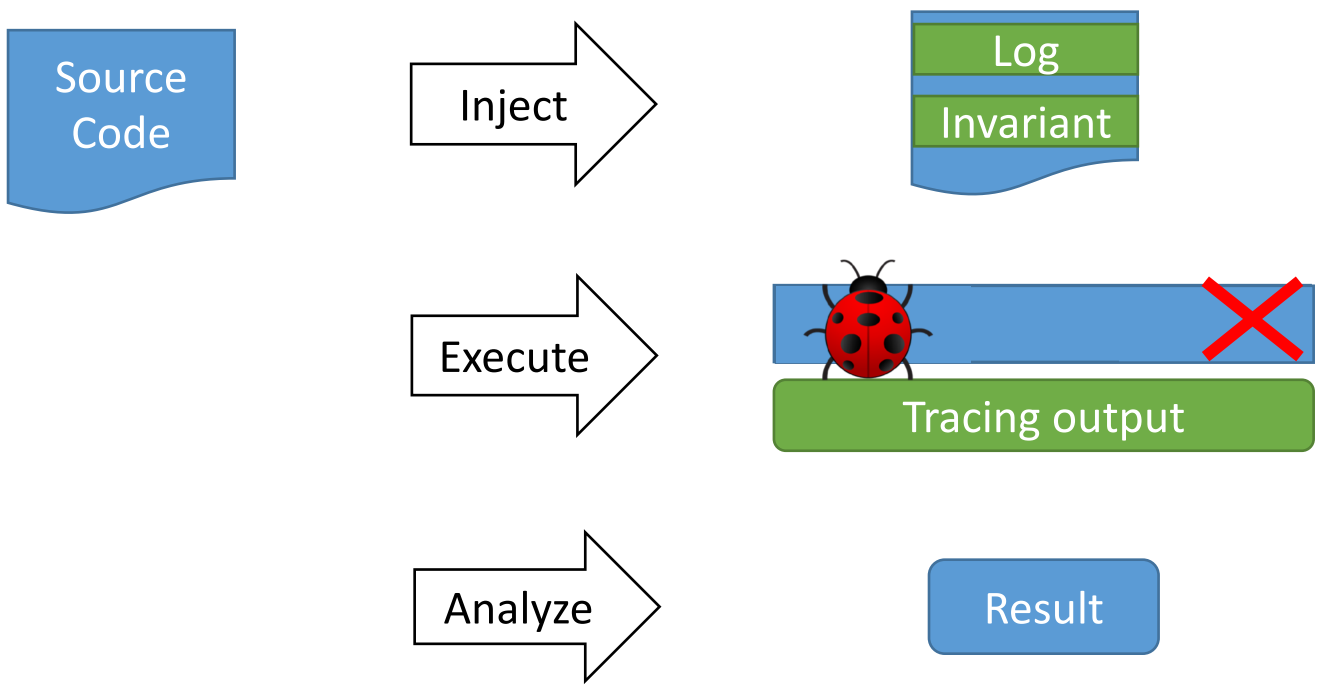
Debugging

- What was your most challenging bug?



Tracing Tools

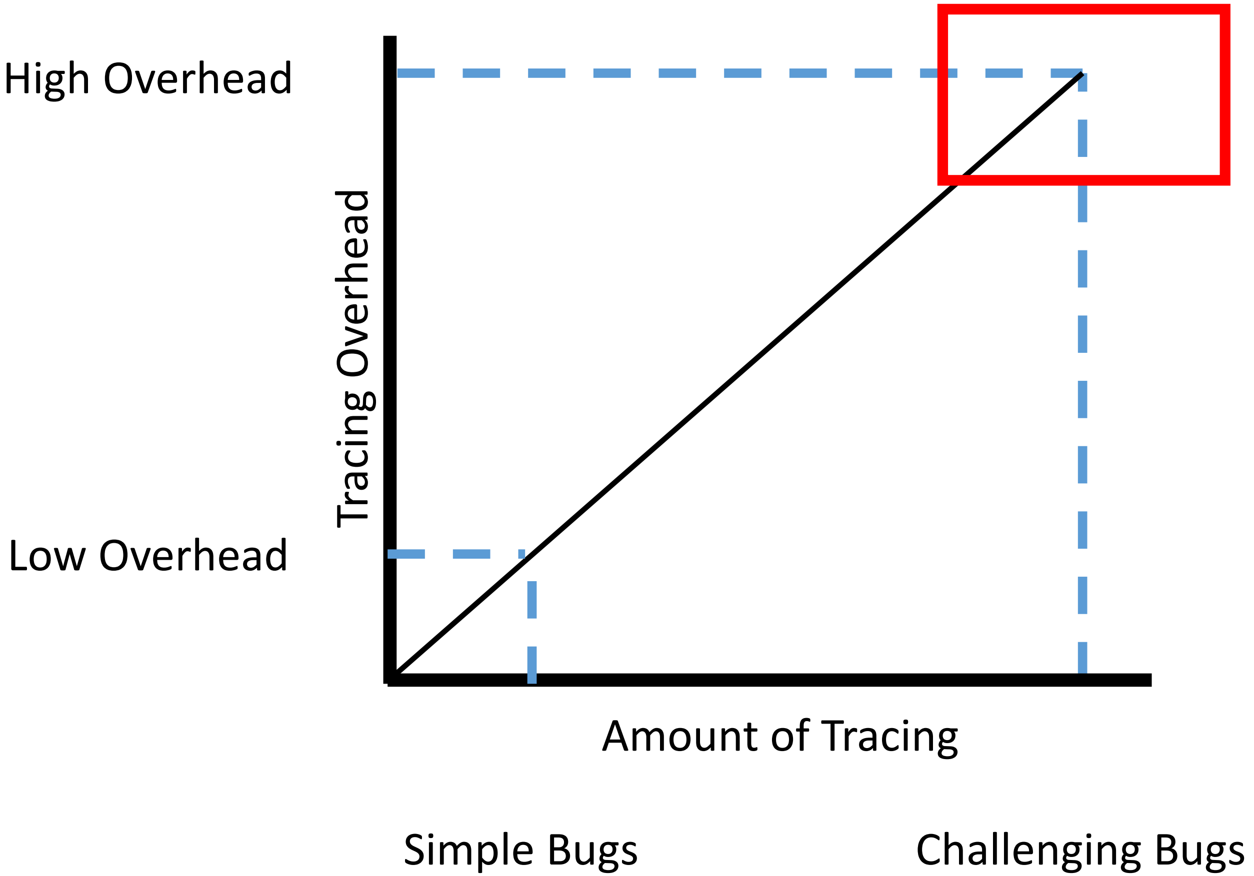
- Inject logic into software to track program state [1][2]
- Re-create failing execution and analyze tracing output



[1] "Where is the Bug and How Is It Fixed? An Experiment with Practitioners" (FSE '17)
[2] "Characterizing Logging Practices in Open-Source Software" (ICSE '12)

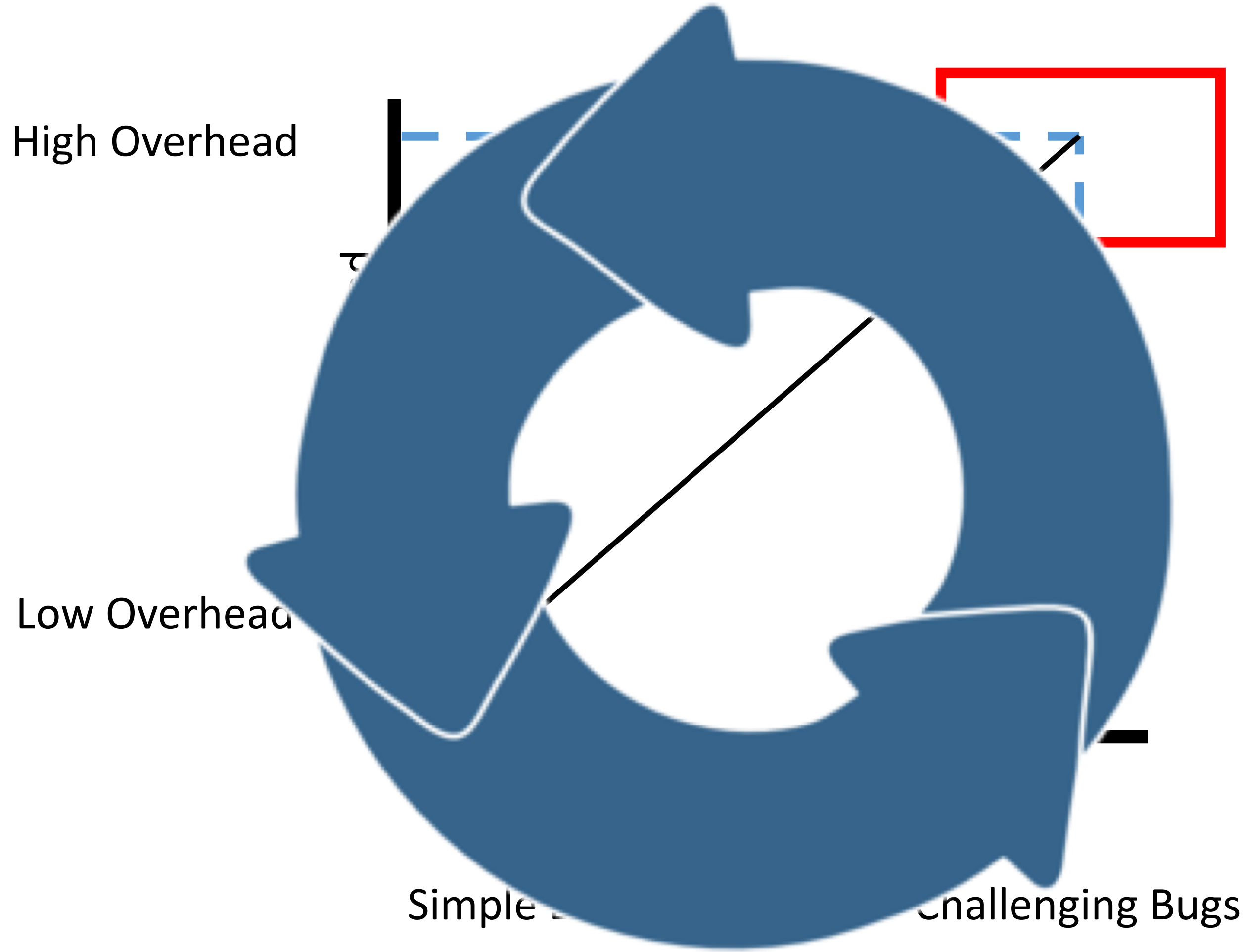
Tracing Tools

- Tradeoff between amount of tracing and overhead



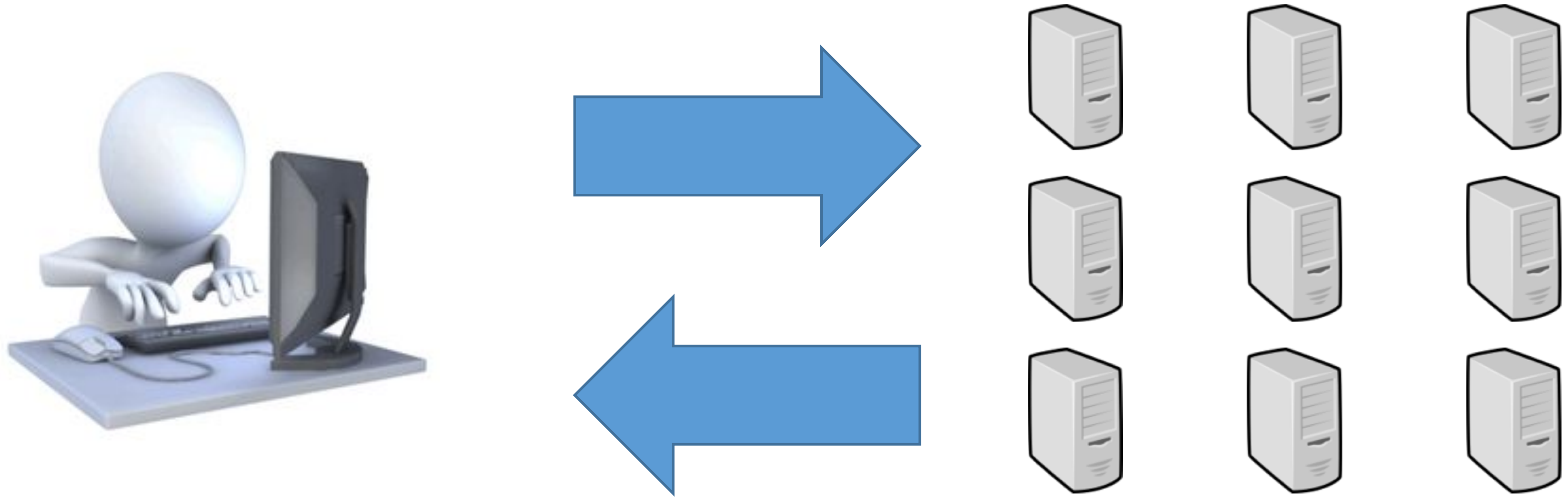
Tracing Tools

- Tradeoff between amount of tracing and overhead



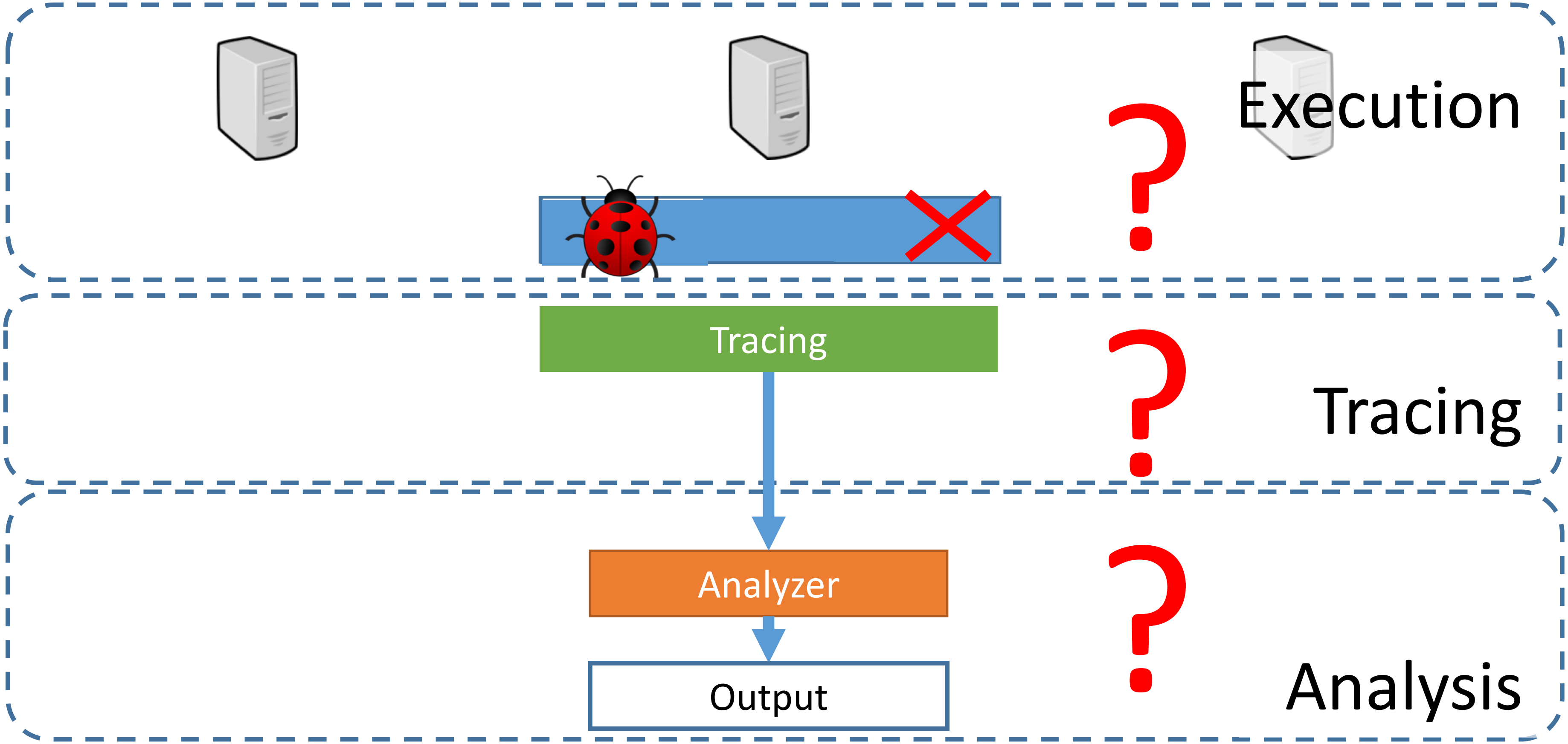
Cluster-Fueled Debugging

- Accelerate complex debugging queries
- Shared cluster => on-demand debugging tool



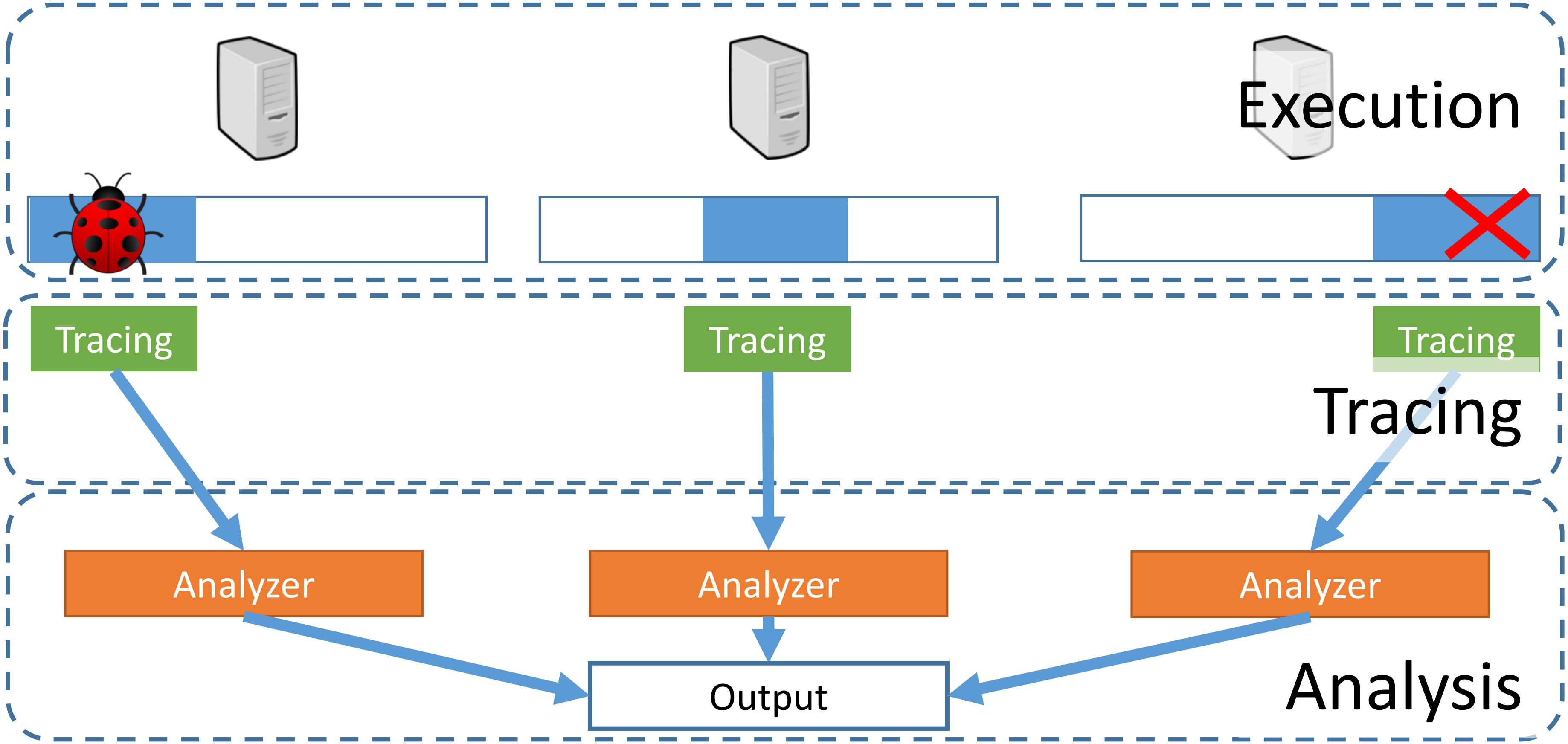
Tracing Tools

- But, the work of these tools are sequential!



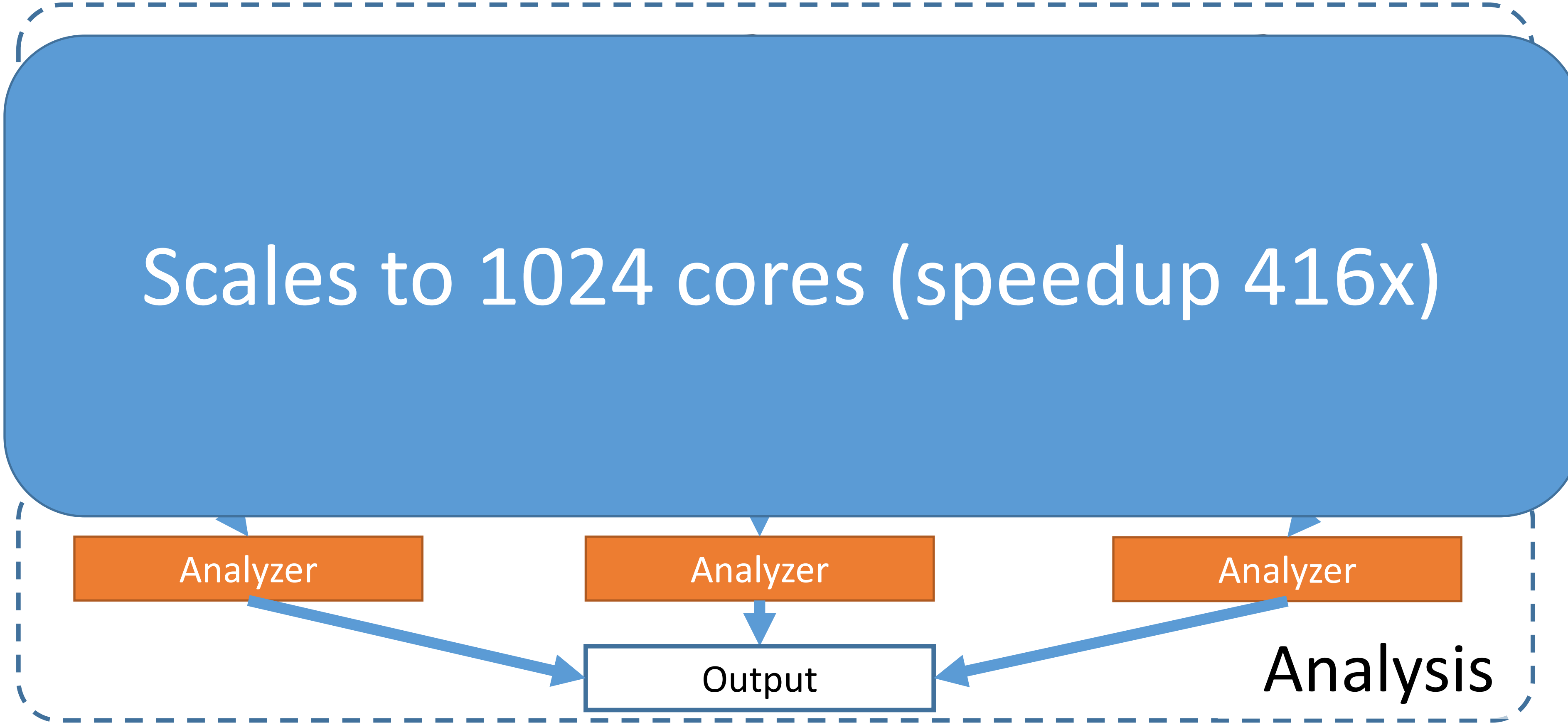
Sledgehammer

- Replay-Based: queries run over past execution



Sledgehammer

- Replay-Based: queries run over past execution



Outline

- Vision of Cluster-Fueled Debugging
- Debugging Scenario
- Parallelizing a Query
- Evaluation

Scenario - memcached

Developer

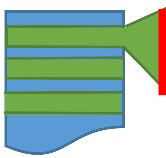
“corrupt cache”

What’s the root cause?
What line of code?

Scenario – Today's Tools

Developer

"corrupt cache"



```
checkCache();
```

Instrumentation

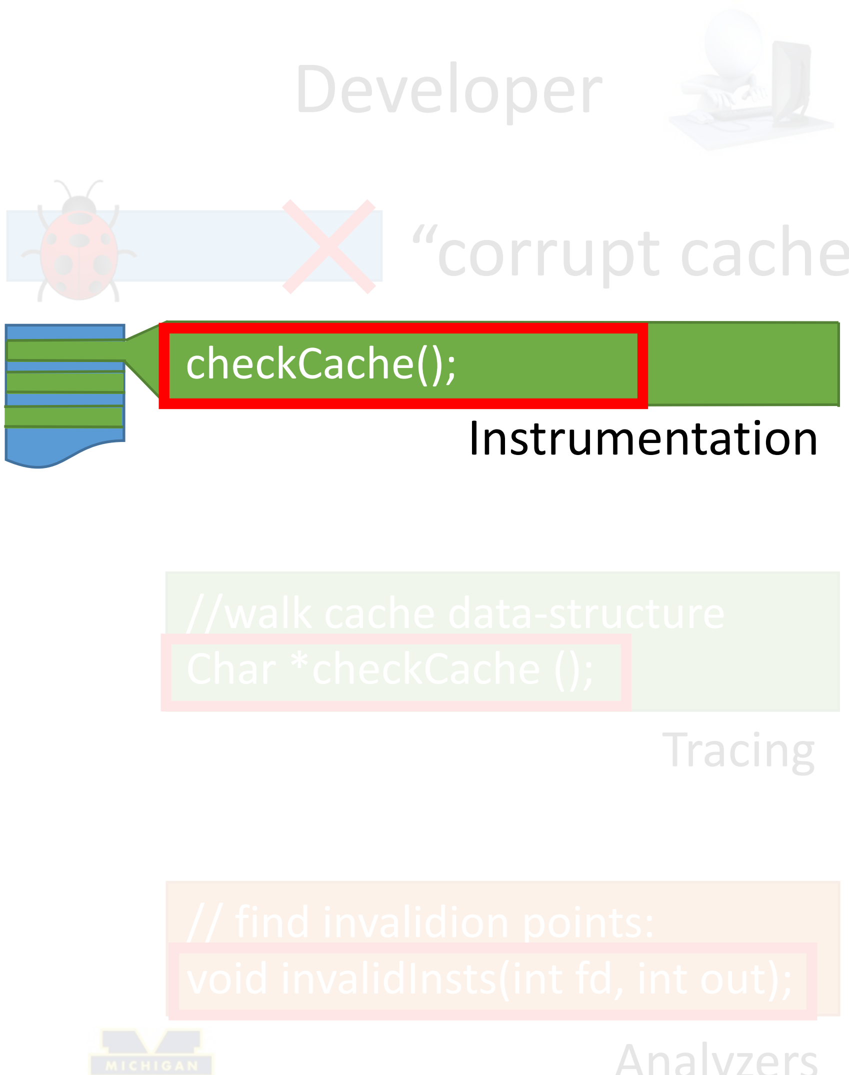
```
//walk cache data-structure
Char *checkCache ();
```

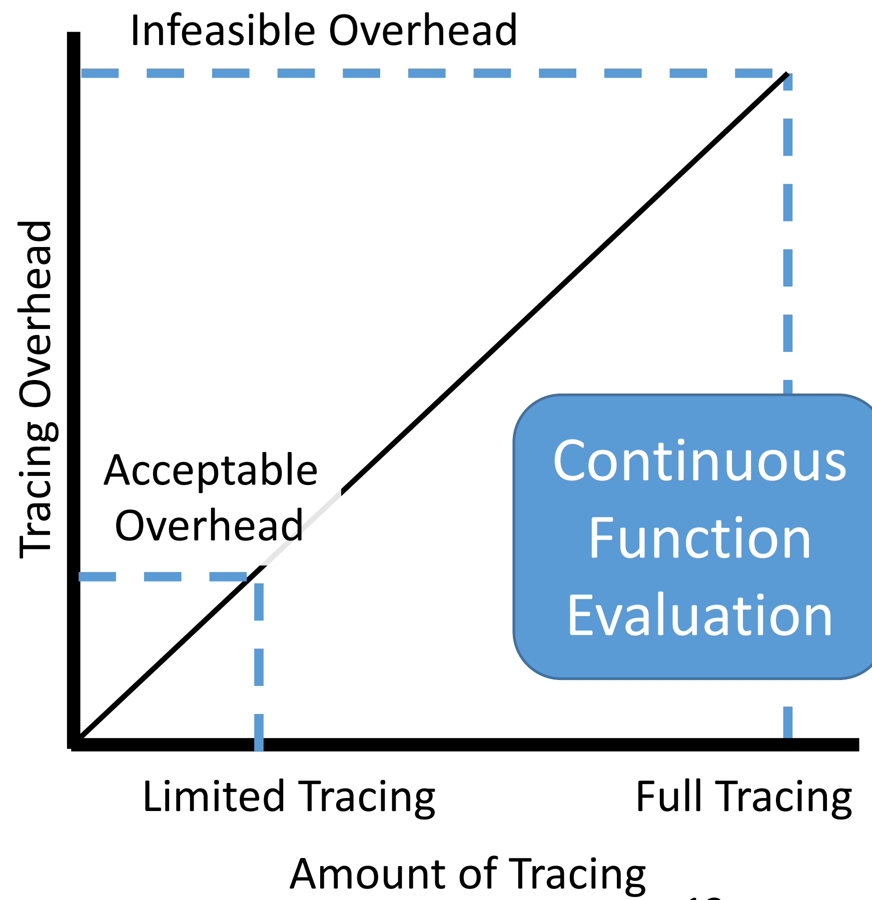
Tracing

```
// find invalidion points:
void invalidInsts(int fd, int out);
```

Analyzers

Scenario – Today’s Tools

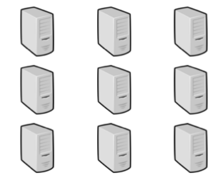




Scenario - Sledgehammer

Developer

Sledgehammer



"corrupt cache"

SH_Cont(checkCache)

Annotations

(New Debugging Tool) Continuous Function Evaluation: execute tracing after each instruction

//walk cache data-structure
Char *checkCache ();

Tracing

CFE Log:
inst1, valid
inst2, invalid
...

// find invalidion points:
void invalidInsts(int fd, int out);

Analyzers

Scenario - Sledgehammer

Developer

Sledgehammer

“corrupt cache”

SH_Cont(checkCache)

Annotations

inst1
inst2
…

Tracing

// find invalidion points:
void invalidInsts(int fd, int out);

Analyzers

(New Debugging Tool)
Continuous Function Evaluation:
execute tracing after each instruction

CFE Log:
inst1, valid
inst2, invalid
…

Scenario - Sledgehammer

Developer

Sledgehammer

“corrupt cac

SH_Cont(checkCache)

//wa cac

Char

Does the lock protect the cache?

(New Debugging Tool)

Function

after

ion

Log:
inst1, valid
inst2, invalid
…

// find invalidion points:
void invalidInsts(int fd, int out);

Analyzers

Scenario - memcached

Developer

Sledgehammer

“corrupt cache”

SH_Cont(checkCache)
SH_Hook(lock, lockAcquire)
SH_Hook(unlock, lockRelease)

Annotations

//walk cache data-structure
Char *checkCache ();
void lockAcquire(mutex *m);
void lockRelease(mutex *m);

Tracing

// find invalid and unlocked:
void badUpdate(int fd, int out);

Analyzers

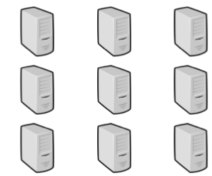
(New Debugging Tool)
Continuous Function Evaluation:
Track tracing output after every instruction

(Cutting-edge Tool)
Retro-Logging:
Inject new logging into past execution

Introvirt SOSP ‘05

Scenario - Sledgehammer

Developer 

Sledgehammer 

"corrupt cache"

SH_Cont(checkCache)
SH_Hook(lock, lockAcquire)
SH_Hook(unlock, lockRelease)

Inst1 (initialization)
Inst2 (BUG)

void lockAcquire(mutex *m);
void lockRelease(mutex *m);

Tracing

// find invalid and unlocked:
void badUpdate(int fd, int out);

Analyzers

(New Debugging Tool)
Continuous Function Evaluation:
Track tracing output after every instruction

(Cutting-edge Tool)
Retro-Logging:
Inject new logging into past execution

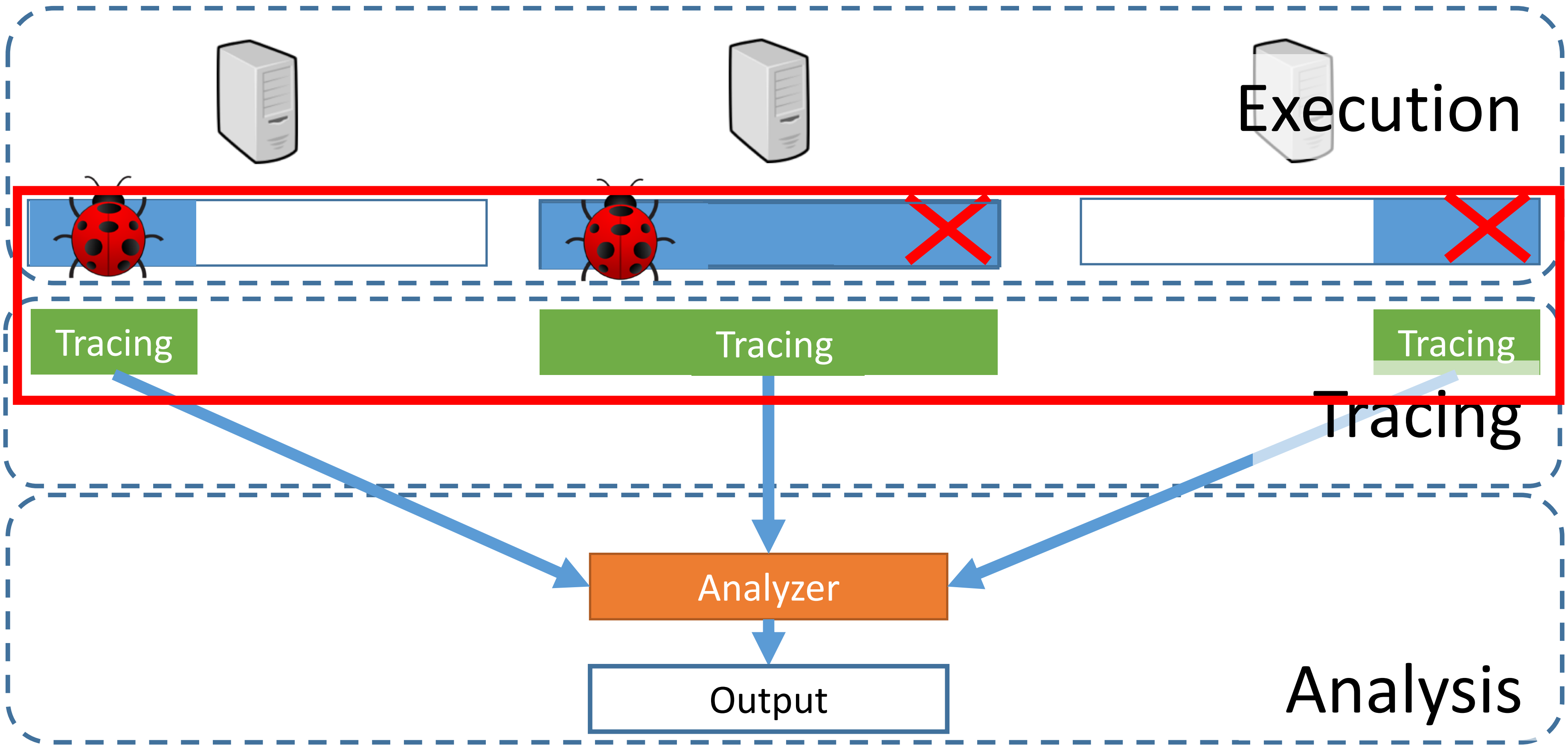
Introvirt SOSP '05

Outline

- Vision of Cluster-Fueled Debugging
- Debugging Scenario
- Parallelizing a Query
- Evaluation

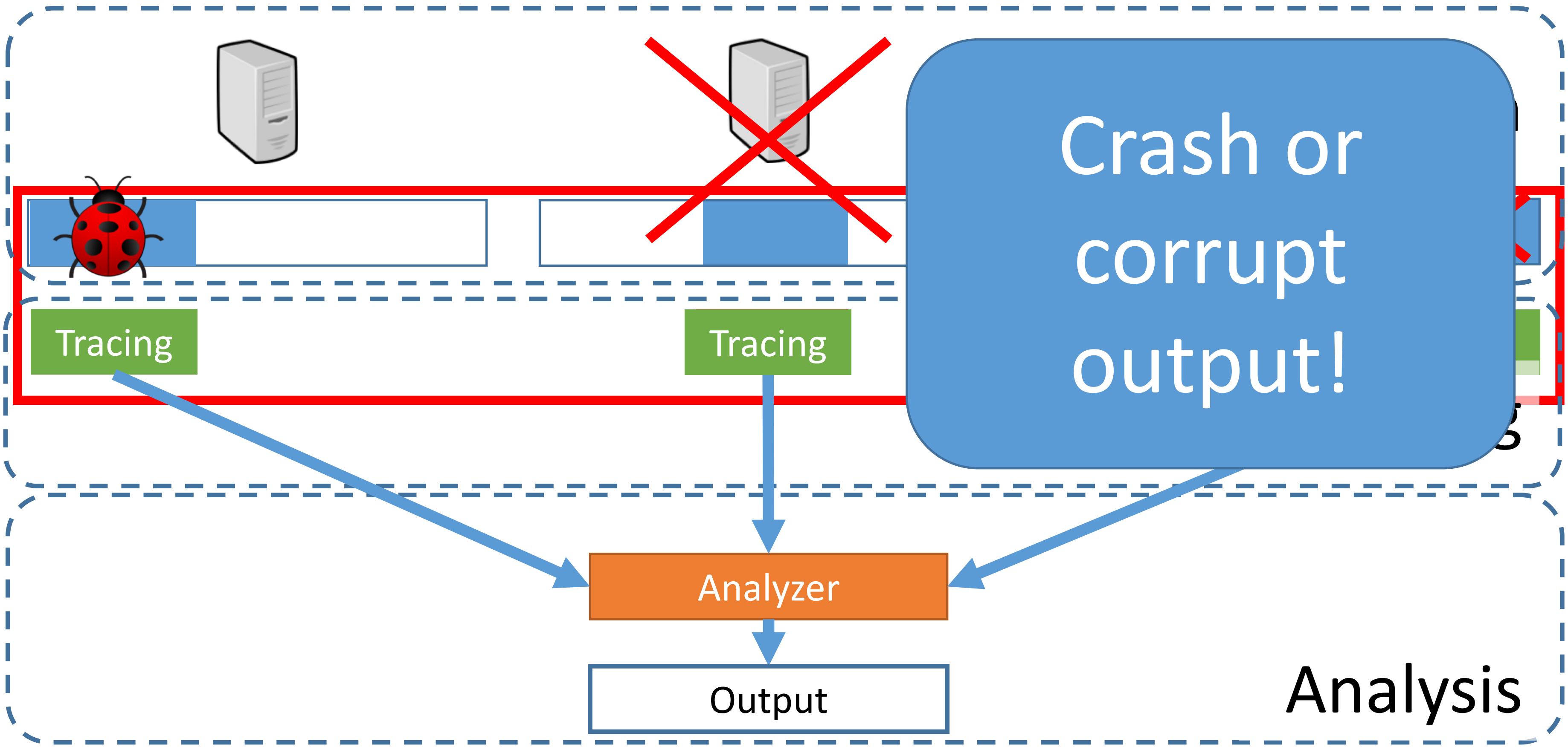
Parallelizing a Query

- Epoch parallelism – leverage deterministic replay



Parallelizing a Query

- Unconstrained tracing code causes divergences



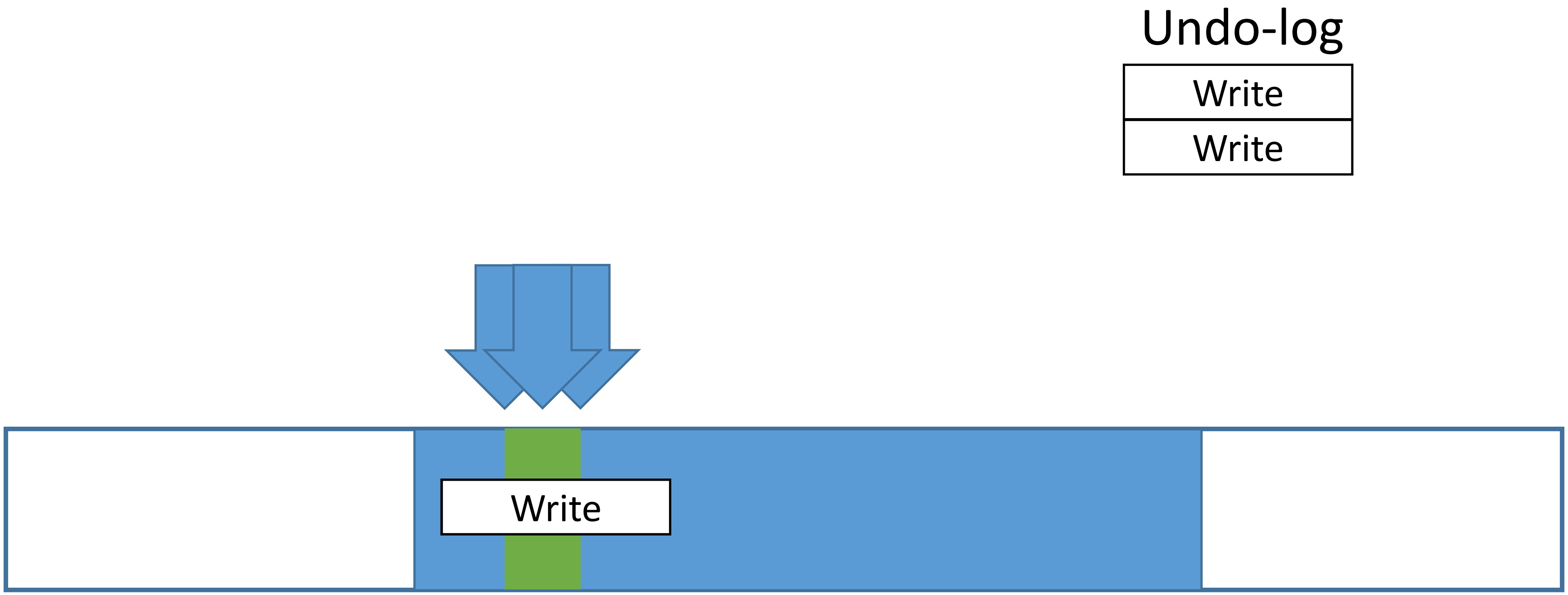
Tracing Isolation

- Prevent tracing code from updating replay state
- Requires scalability and lightweight technique

	Heavyweight	Lightweight
Poor Scalability		Pin / Valgrind
Good Scalability	Checkpointing (Introvirt SOSP '05)	Compiler-based Isolation

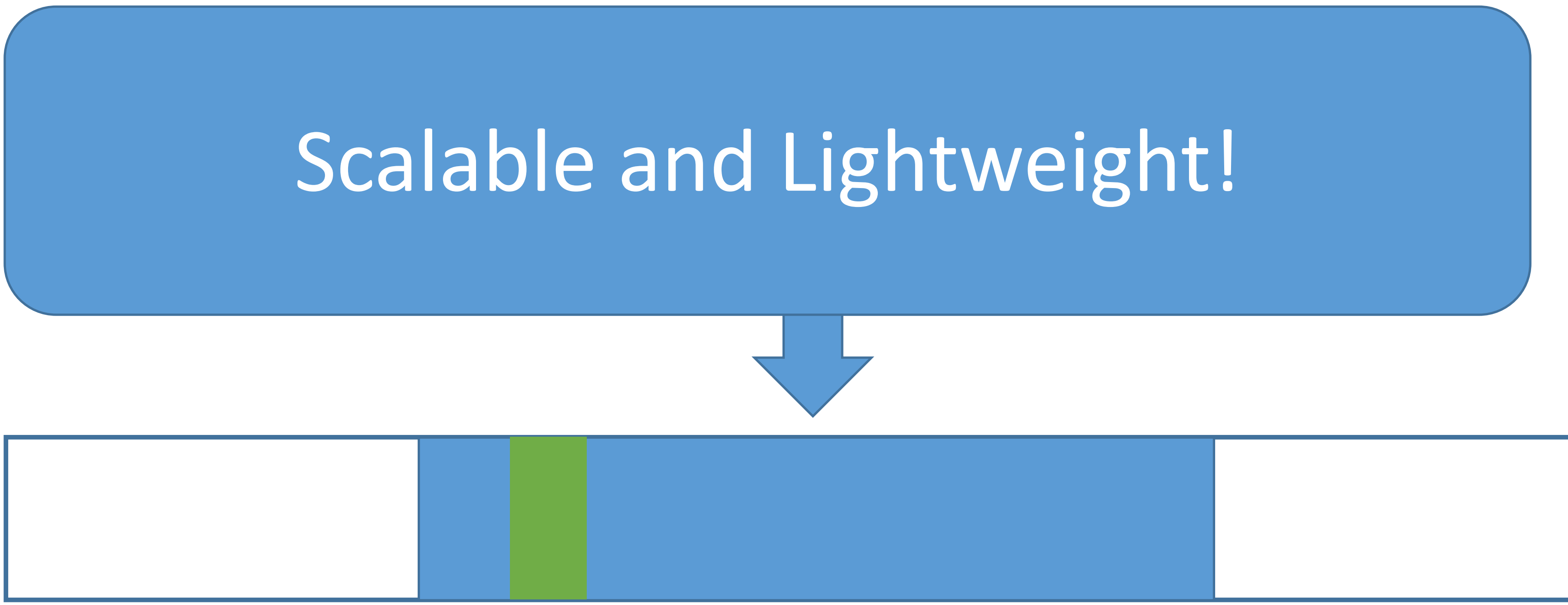
Compiler-based Isolation

- Instrument tracing code to log updates
- Revert updates using log



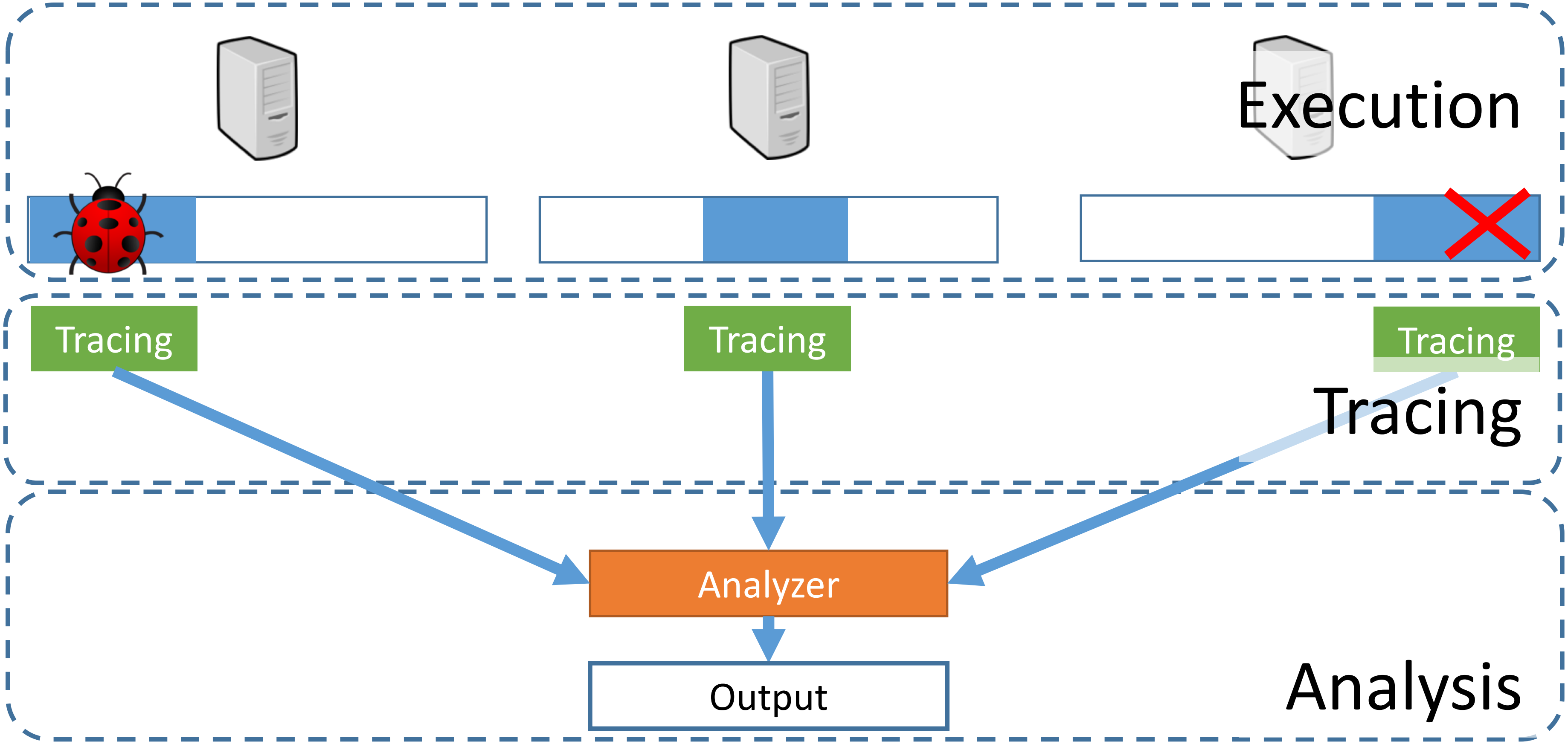
Compiler-based Isolation

- Instrument tracing code to log updates
- Revert updates using log



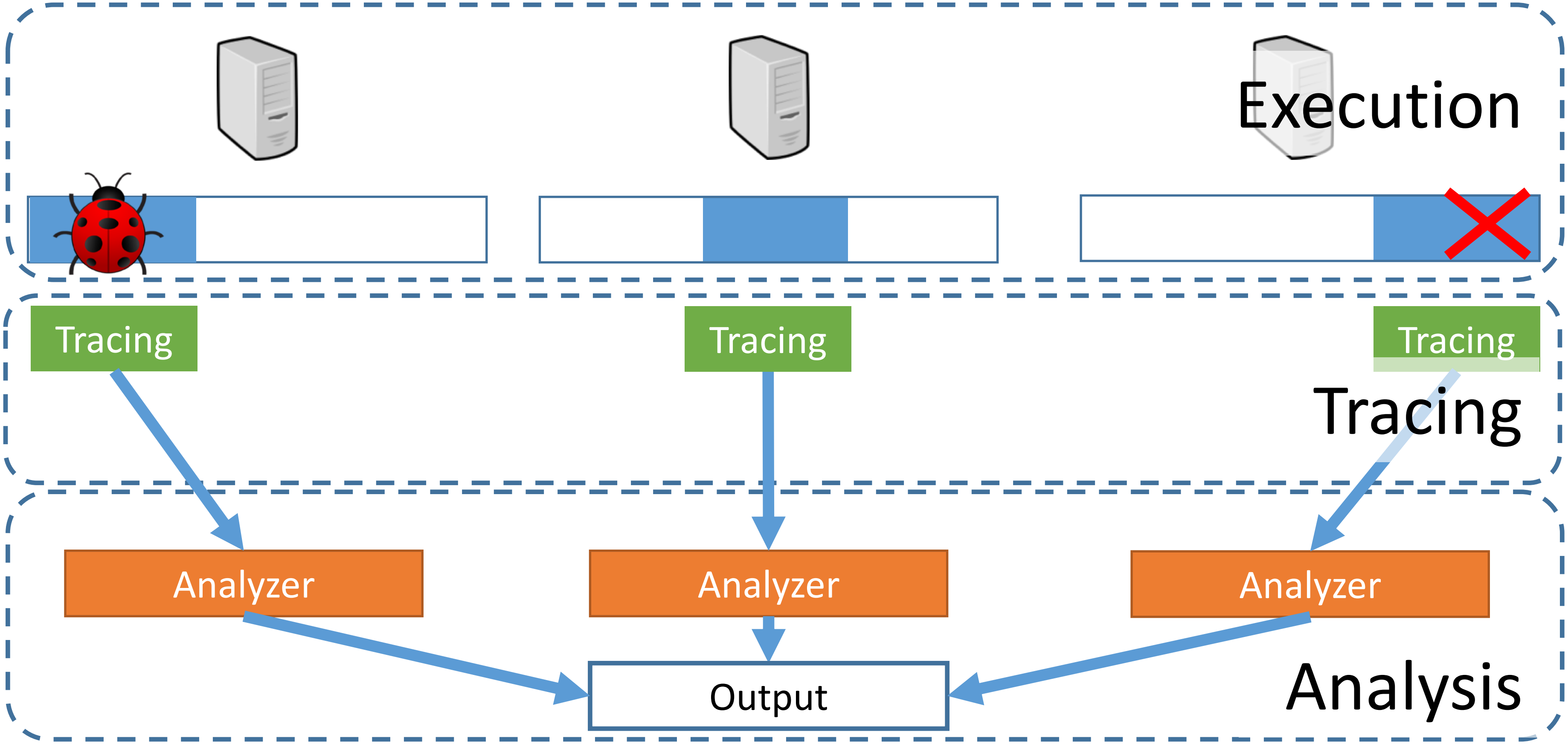
Parallelizing a query

- Multi-tiered approach to parallelize analysis



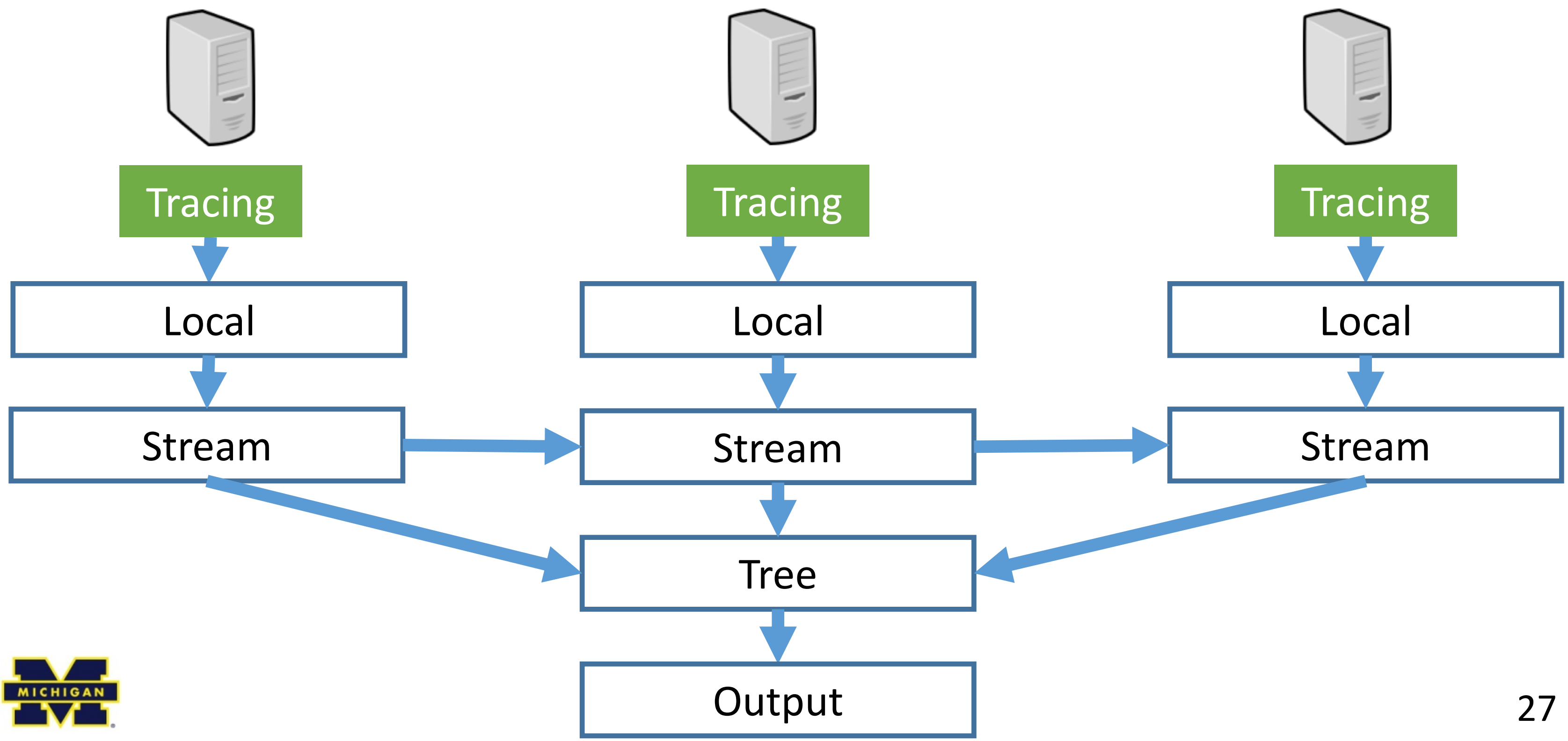
Parallelizing a query

- Multi-tiered approach to parallelize analysis



Analysis

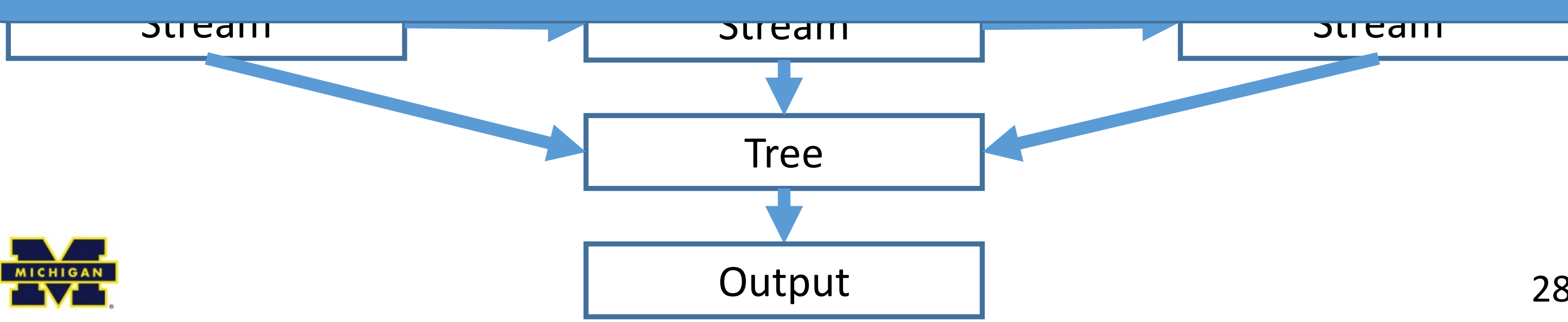
- Developers construct…
    - Local runs on each core in parallel
    - Stream passes information over chain
    - Tree combines input from multiple analyzers



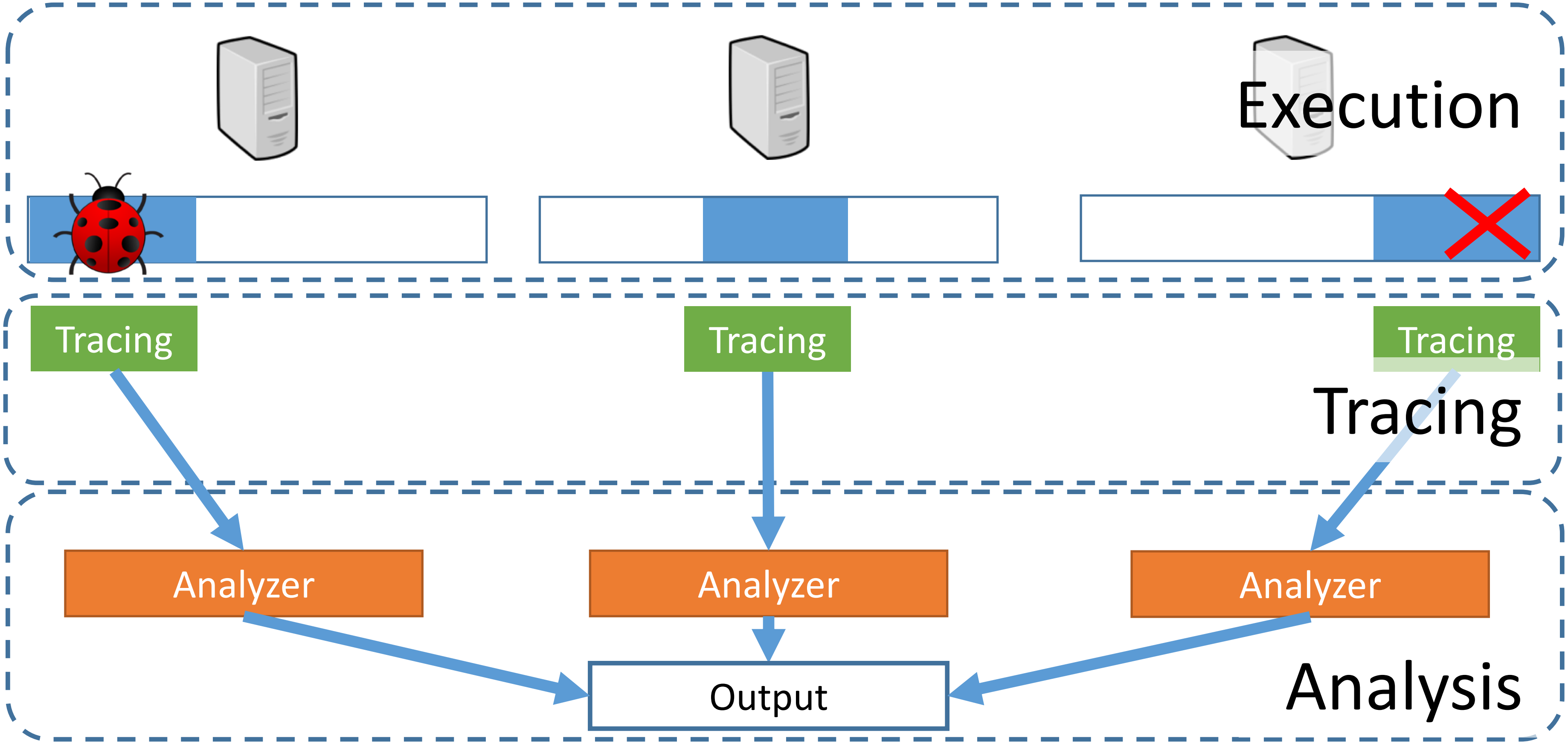
Analysis

- Developers construct…
  - Local runs on each core in parallel
  - Stream passes information over chain
  - Tree combines input from multiple analyzers

Parallel faster by up to 4x (mean 2x)



Parallelizing a query

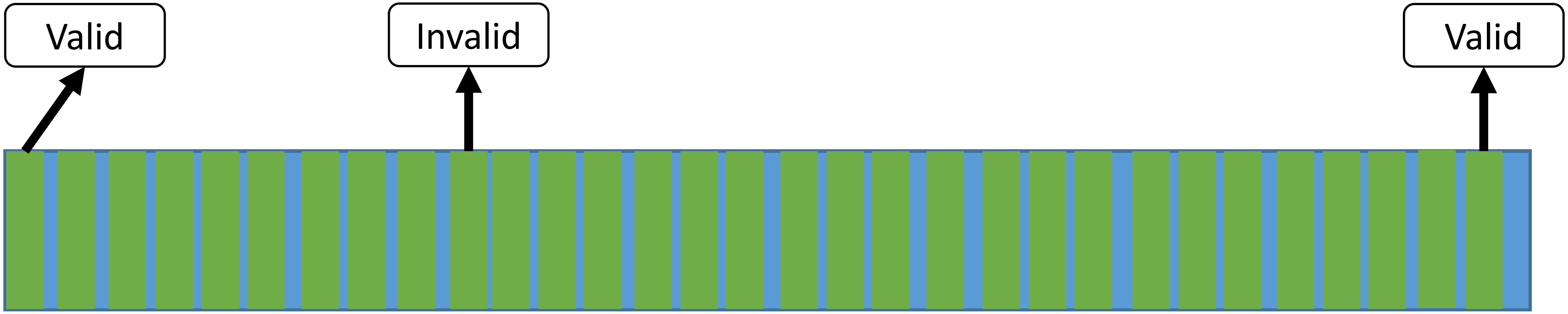


Sledgehammer Tools

- Continuous Function Evaluation: Run after every instruction
- Retro-Timing: Query timing information
- Retro-Logging: Inject new logging code

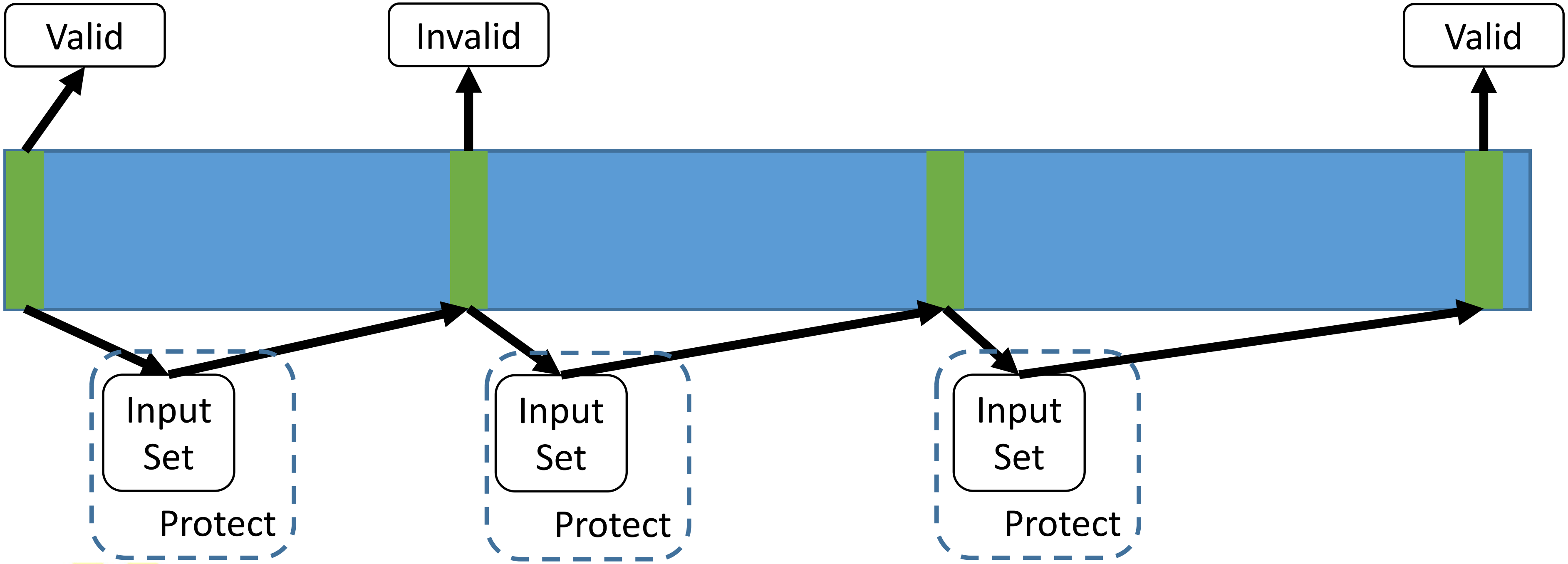
Continuous Function Evaluation

- Logically, execute function after each instruction
- Instead, force determinism and trigger on input changes



Continuous Function Evaluation

- Static instrumention produces input set
- Memory page protections track updates to input set



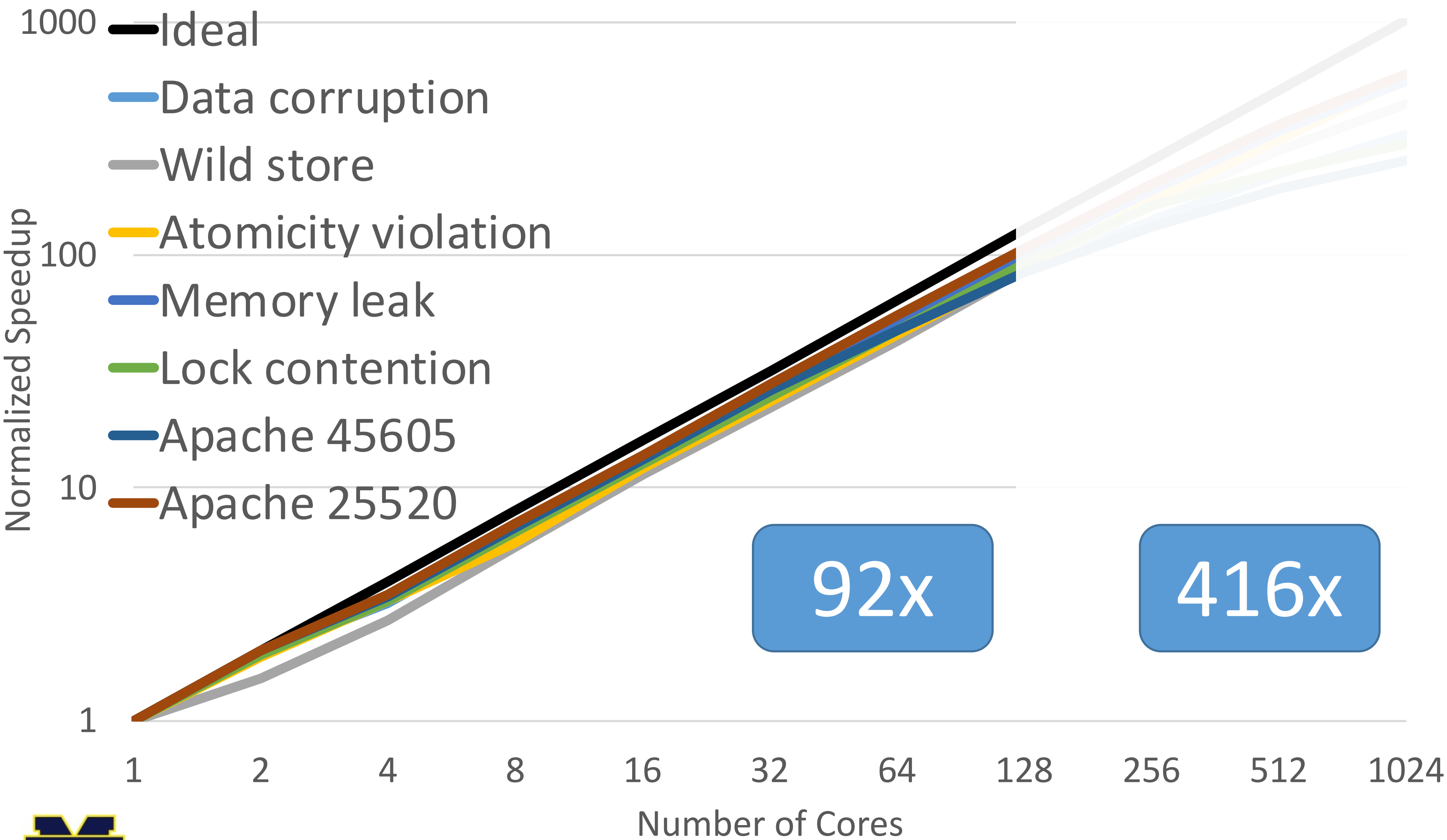
Outline

- Vision of Cluster-Fueled Debugging
- Debugging Scenario
- Parallelizing a Query
- Evaluation

Scenarios

Name	Benchmark	Replay Time	Tracing (millions)	1-Core Latency	1024-Core Latency
Data Corruption	Nginx	2s	8	5m25s	1s
Wild Store	MongoDB	30s	3	2h8m8s	17s
Atomicity Violation	Memcached	1m38s	43	2h10m52s	14s
Memory Leak	Nginx	1m16s	4	26m15s	3s
Lock Contention	Memcached	1m33s	76	54m42s	11s
Apache 45605	Apache	51s	2	4m10s	1s
Apache 25520	Apache	60s	4	11m57s	1s

Scalability

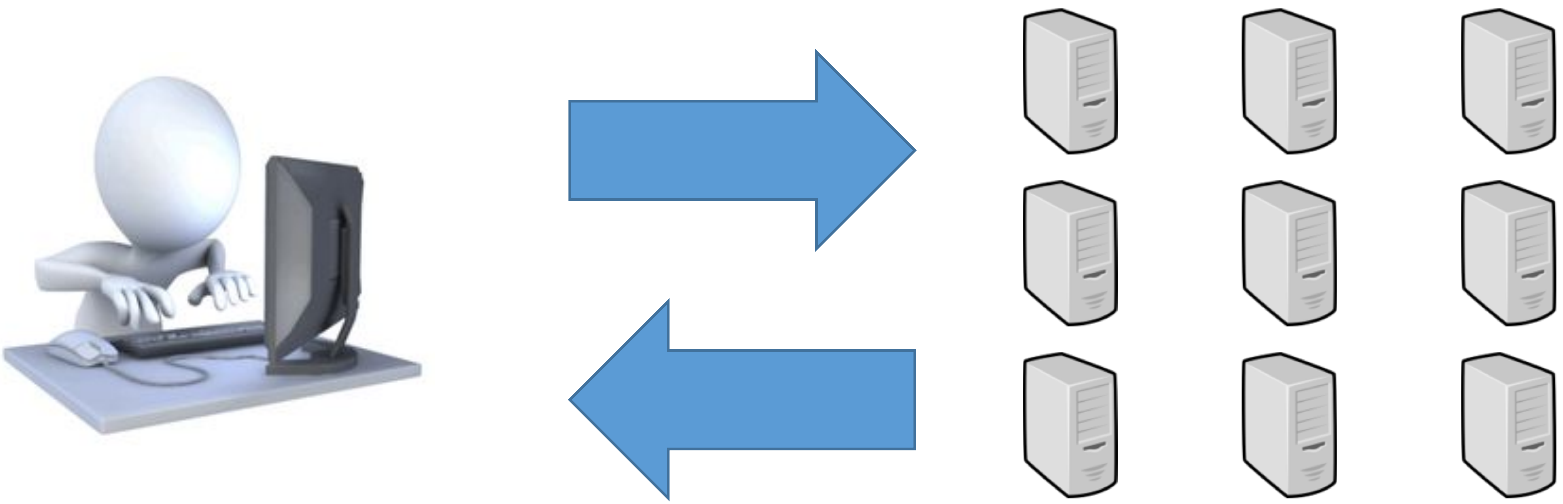


Scalability

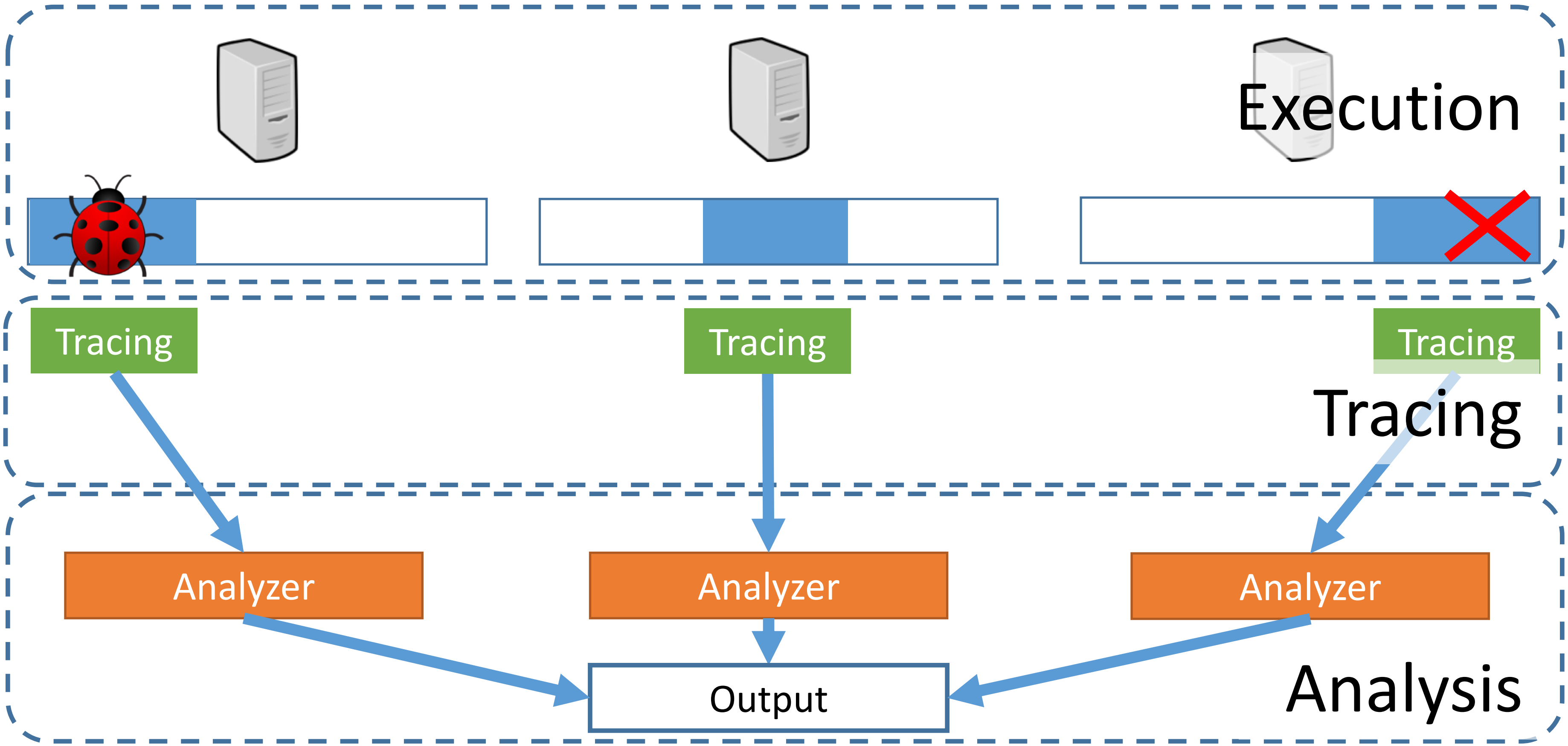
- Why does Sledgehammer not scale perfectly?
    - Initialization (tracer injection, restoring epoch start)
    - Outliers (latency of slowest vs. average epoch)
- More analysis in the paper

Cluster-Fueled Debugging

- Accelerate complex queries to interactive latencies

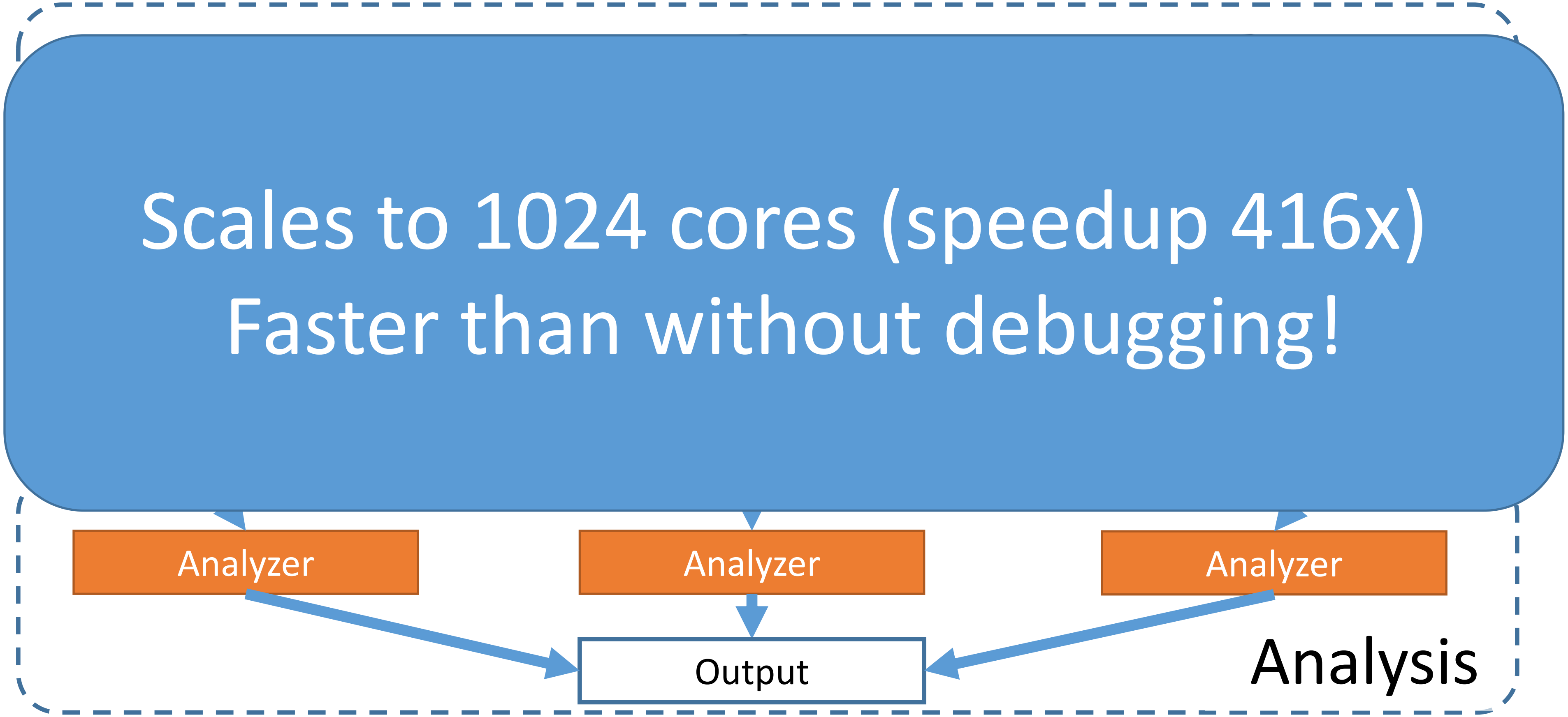


Sledgehammer



Sledgehammer

Scales to 1024 cores (speedup 416x)
Faster than without debugging!



Questions

Poster #13