

# Analysis of Overhead in Dynamic Java Performance Monitoring

Vojtěch Horký, Jaroslav Kotrč, Peter Libič and Petr Tůma

Charles University in Prague



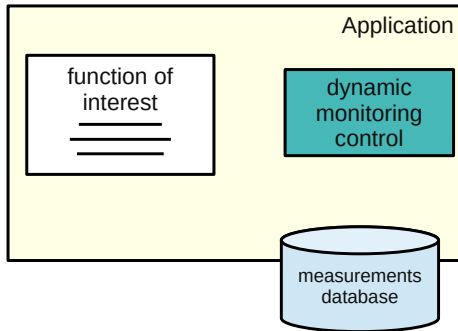
# Context: Dynamic Monitoring of Production Systems

# Dynamic Monitoring of Production Systems

Measurement probes are active only when needed,  
measuring everything all the time might not be practical.

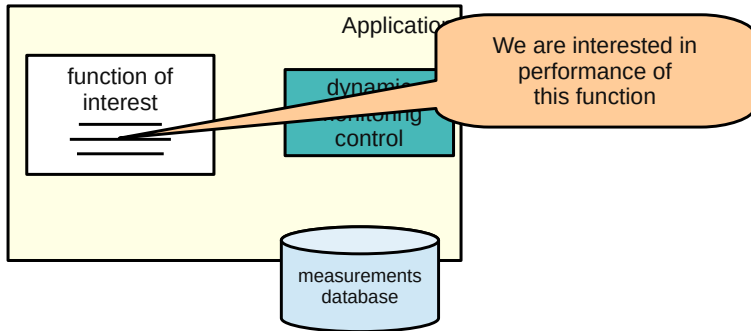
# Dynamic Monitoring of Production Systems

Measurement probes are active only when needed, measuring everything all the time might not be practical.



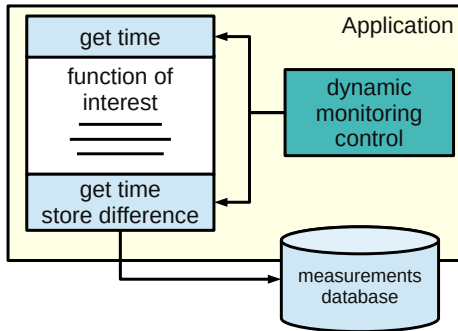
# Dynamic Monitoring of Production Systems

Measurement probes are active only when needed, measuring everything all the time might not be practical.



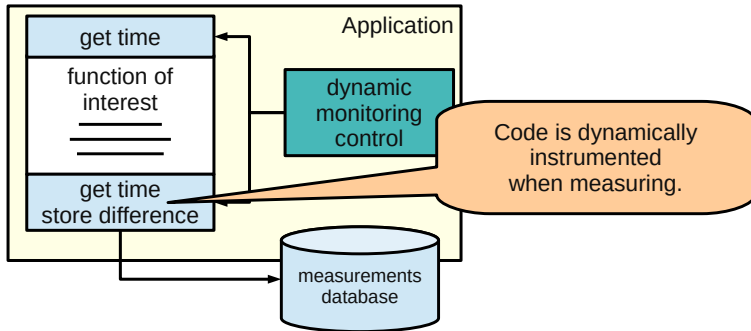
# Dynamic Monitoring of Production Systems

Measurement probes are active only when needed,  
measuring everything all the time might not be practical.



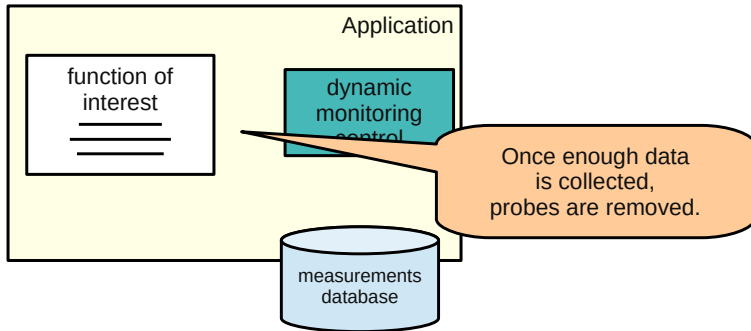
# Dynamic Monitoring of Production Systems

Measurement probes are active only when needed, measuring everything all the time might not be practical.



# Dynamic Monitoring of Production Systems

Measurement probes are active only when needed, measuring everything all the time might not be practical.





# Issues of Dynamic Monitoring

In managed environments, code is compiled at run-time; probe insertion (removal) causes recompilation.

Monitored application can thus behave differently.

# Issues of Dynamic Monitoring

In managed environments, code is compiled at run-time; probe insertion (removal) causes recompilation. Monitored application can thus behave differently.

## Interesting Questions

How do the code manipulations affect the application?

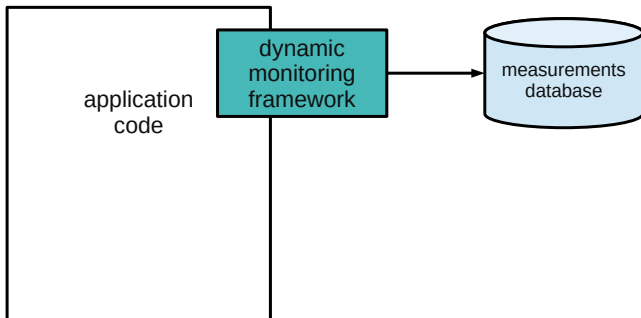
What is the overhead of such probe?

Is the observed performance representative?

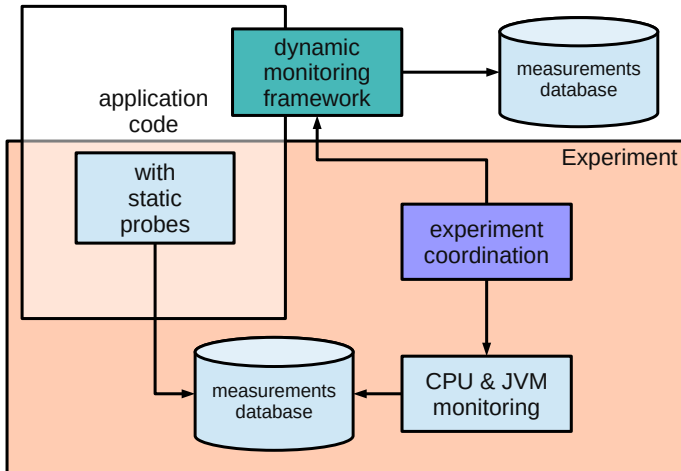
Is there zero overhead once the probe is removed?

# Experiment Setup

# Experiment Coordination



# Experiment Coordination



# Two Measurement Infrastructures

---

Self-measurement

Dynamic monitoring

---

# Two Measurement Infrastructures

---

	Self-measurement	Dynamic monitoring
Performance	Baseline	“Observed”

---

# Two Measurement Infrastructures

---

Self-measurement

Dynamic monitoring

---

Performance

Baseline

“Observed”

Location

Method entry and exit points (both)



# Two Measurement Infrastructures

---

	Self-measurement	Dynamic monitoring
Performance	Baseline	“Observed”
Location	Method entry and exit points (both)	
Instrumentation	Static	Dynamic (run-time)

---

# Two Measurement Infrastructures

---

	Self-measurement	Dynamic monitoring
Performance	Baseline	“Observed”
Location	Method entry and exit points (both)	
Instrumentation	Static	Dynamic (run-time)
Data collection	Continuous	On demand

---

# Two Measurement Infrastructures

---

	Self-measurement	Dynamic monitoring
Performance	Baseline	“Observed”
Location	Method entry and exit points (both)	
Instrumentation	Static	Dynamic (run-time)
Data collection	Continuous	On demand
Implementation	Native method (in C)	Pure Java

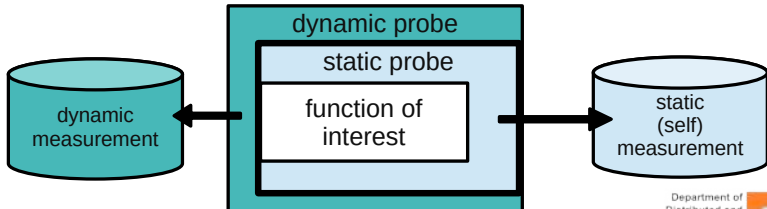
---

# Two Measurement Infrastructures

---

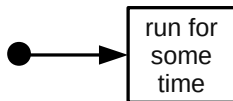
	Self-measurement	Dynamic monitoring
Performance	Baseline	“Observed”
Location	Method entry and exit points (both)	
Instrumentation	Static	Dynamic (run-time)
Data collection	Continuous	On demand
Implementation	Native method (in C)	Pure Java

---

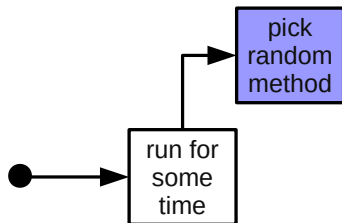


# Experiment Process

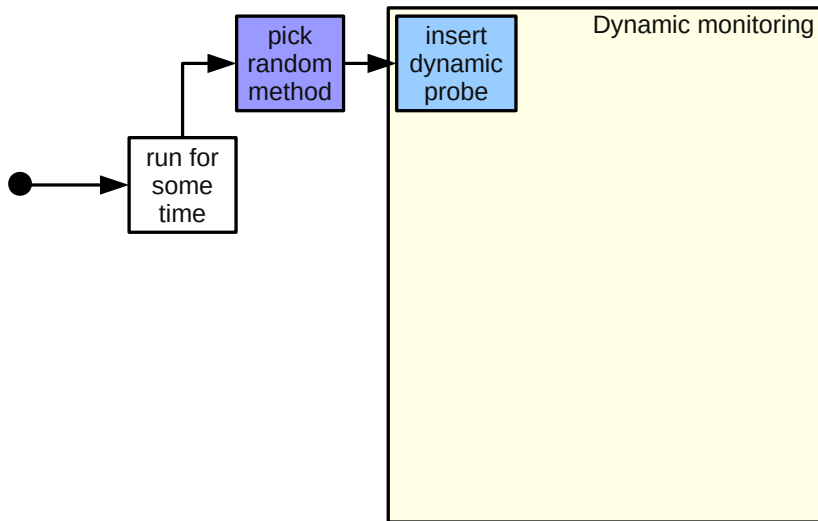
# Experiment Process



# Experiment Process

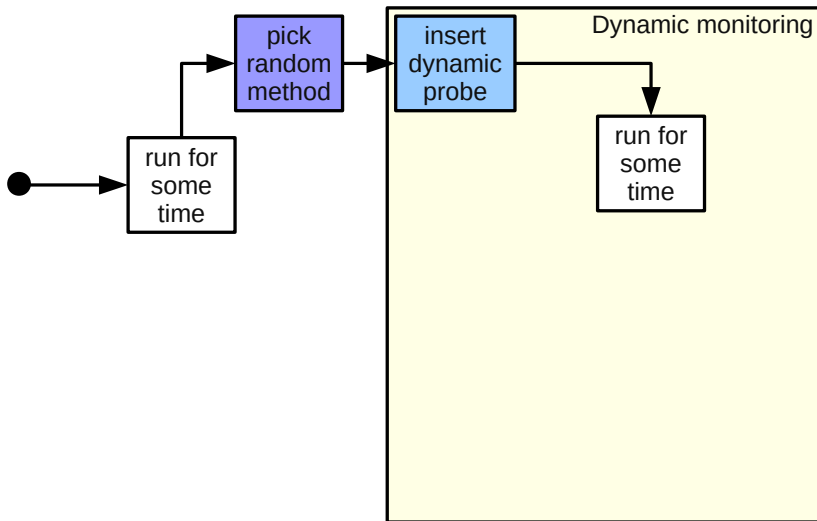


# Experiment Process

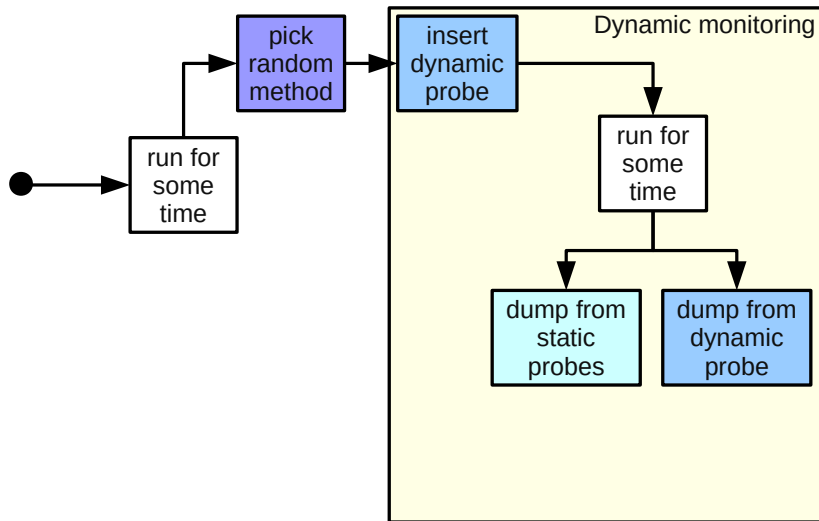




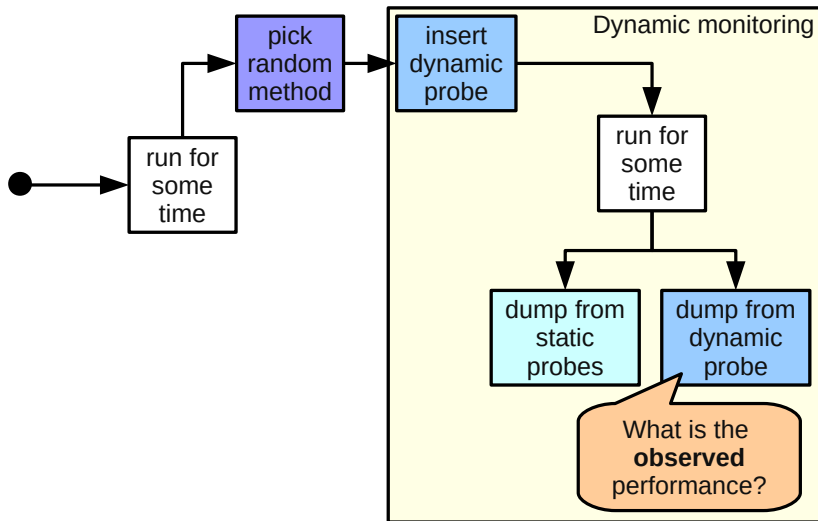
# Experiment Process



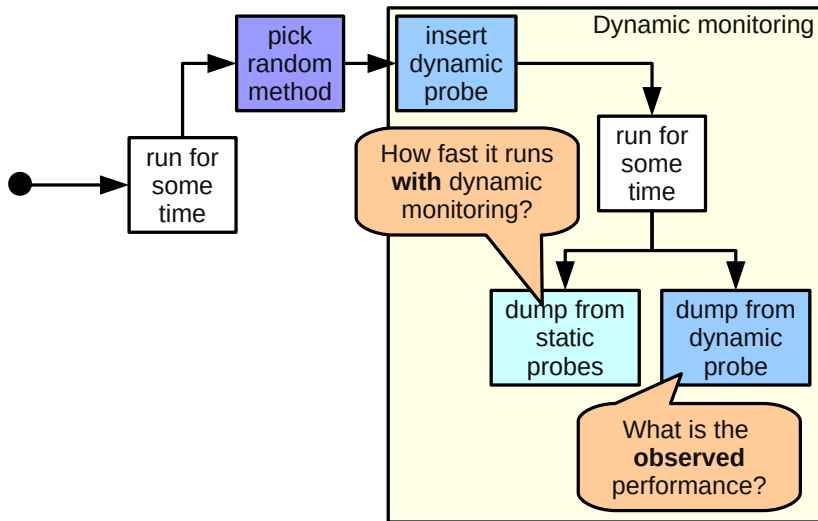
# Experiment Process



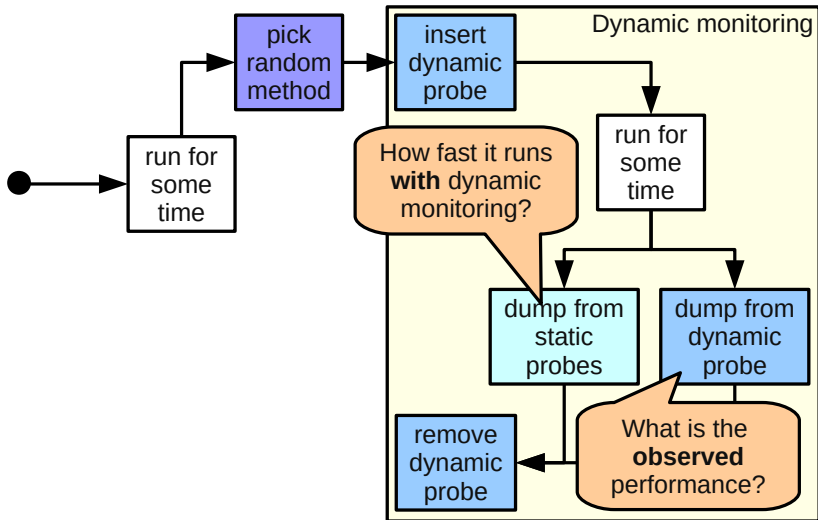
# Experiment Process



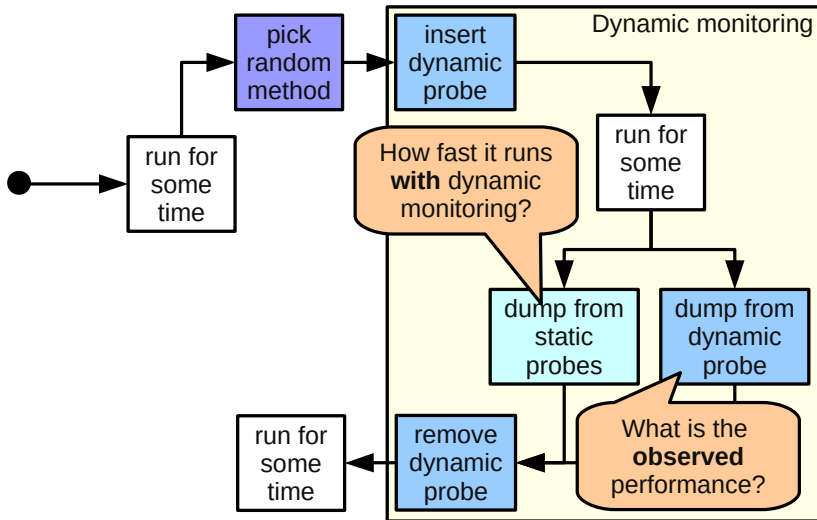
# Experiment Process



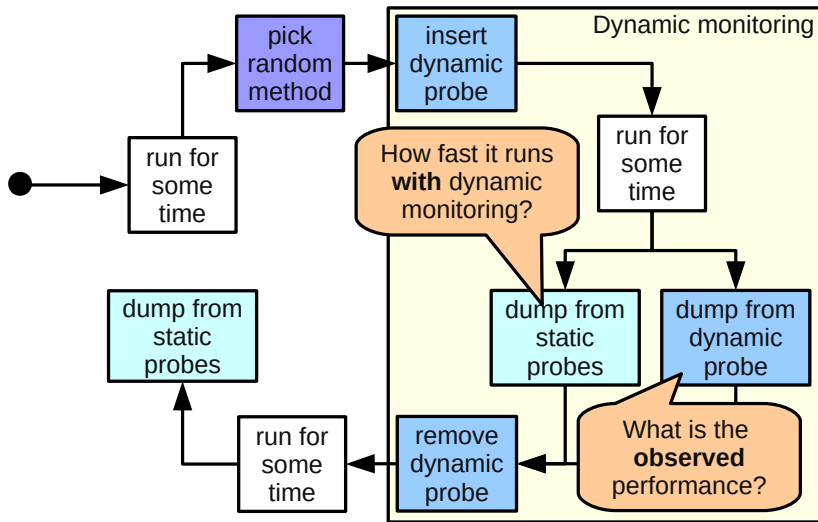
# Experiment Process



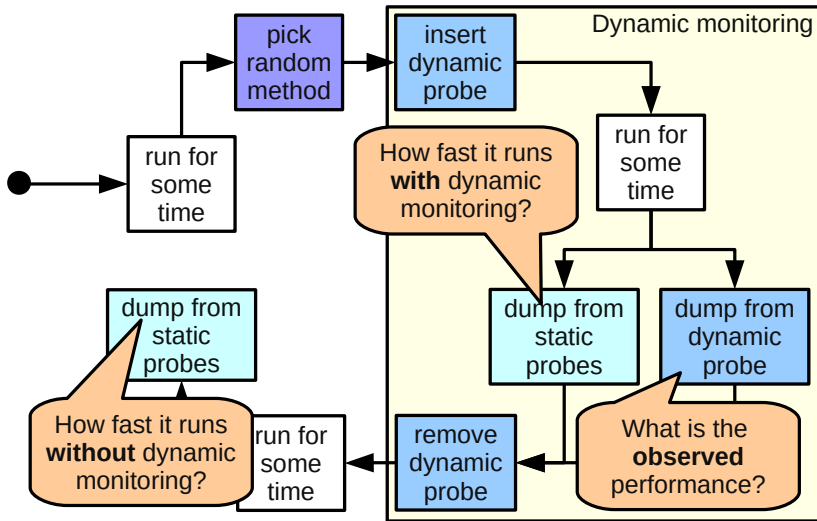
# Experiment Process



# Experiment Process

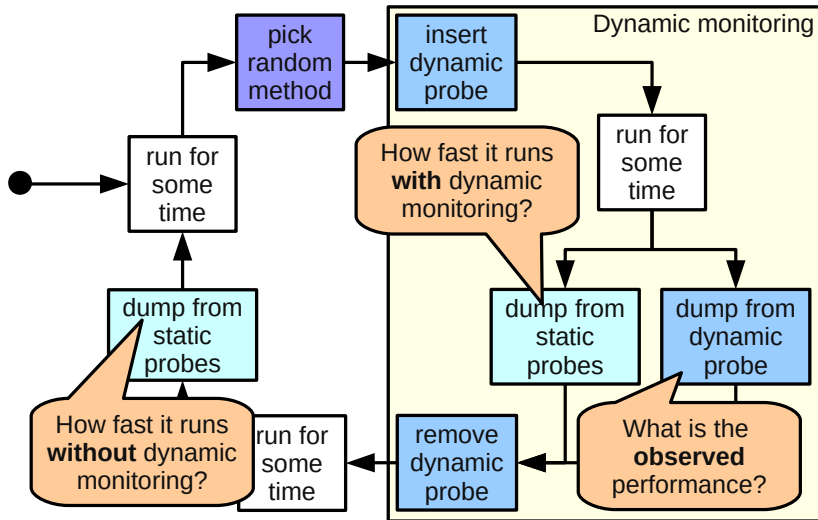


# Experiment Process





# Experiment Process



# Platform and Application Details

# Platform and Application Details

- Hardware: 32 CPUs, 2 NUMA nodes, 48G RAM.

# Platform and Application Details

- Hardware: 32 CPUs, 2 NUMA nodes, 48G RAM.
- SPECjbb2015 augmented with static probes.
  - Fixed request rate 4 000 reqs/s.  
(Close to maximum with static probes on our hardware.)
- Over 1 200 monitored methods.
  - Business code of the benchmark.
  - Practically all methods called frequently enough.
  - About one minute of dynamic monitoring per method.

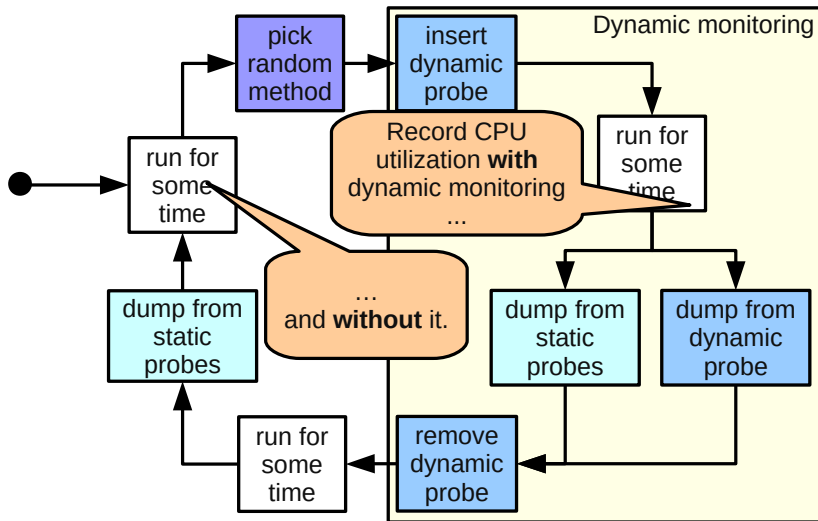
# Platform and Application Details

- Hardware: 32 CPUs, 2 NUMA nodes, 48G RAM.
- SPECjbb2015 augmented with static probes.
  - Fixed request rate 4 000 reqs/s.  
(Close to maximum with static probes on our hardware.)
- Over 1 200 monitored methods.
  - Business code of the benchmark.
  - Practically all methods called frequently enough.
  - About one minute of dynamic monitoring per method.
- Several TBs of raw data per week of run-time.

# Results

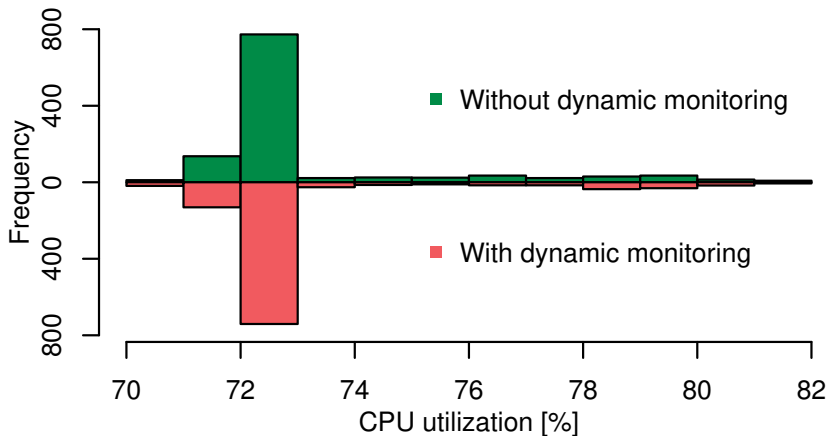
# Overall Overhead of Dynamic Monitoring

# Overall Overhead of Dynamic Monitoring

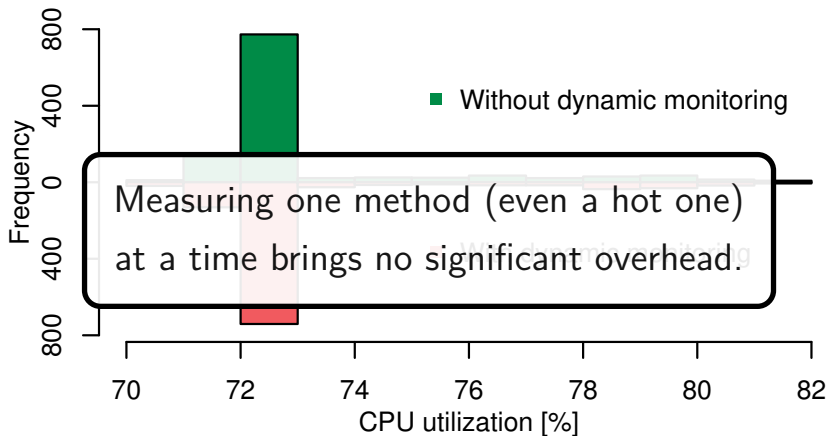




# Overall Overhead of Dynamic Monitoring

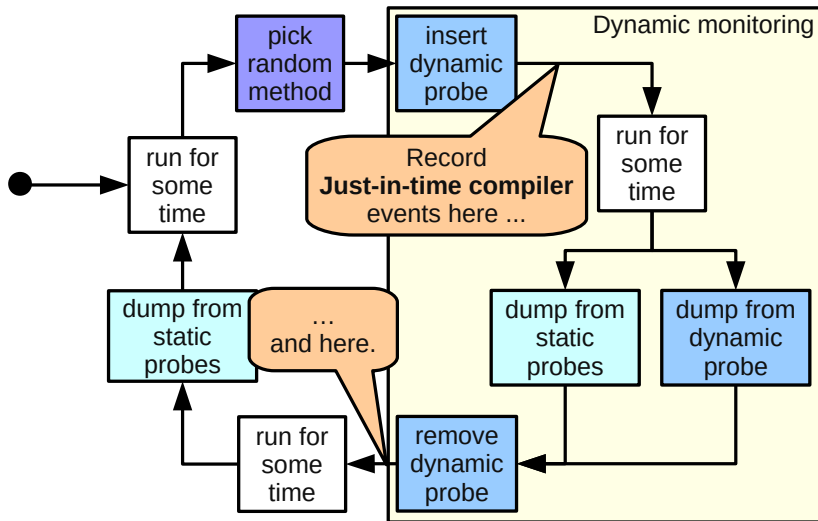


# Overall Overhead of Dynamic Monitoring

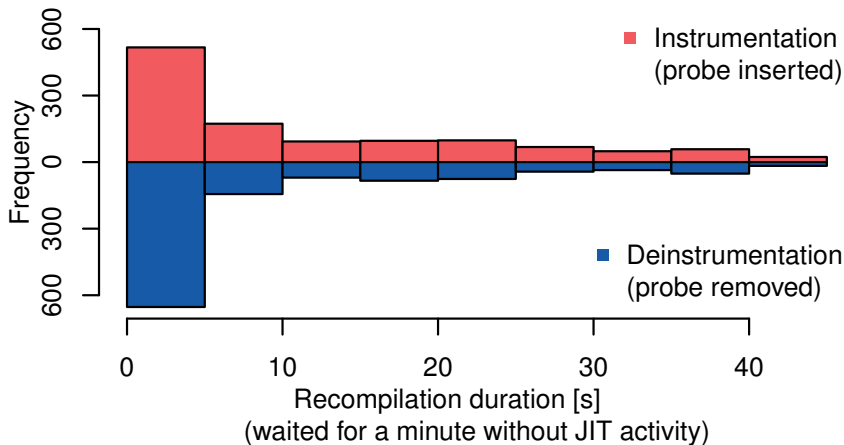


# Time Needed for Just-in-time Recompilation

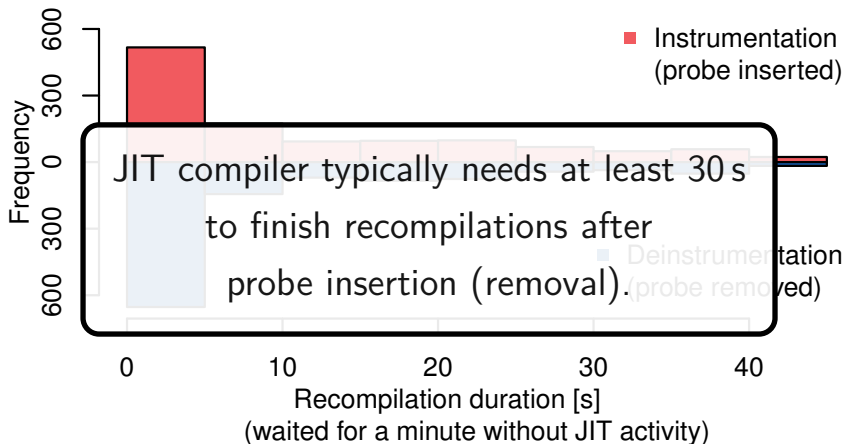
# Time Needed for Just-in-time Recompilation



# Time Needed for Just-in-time Recompilation

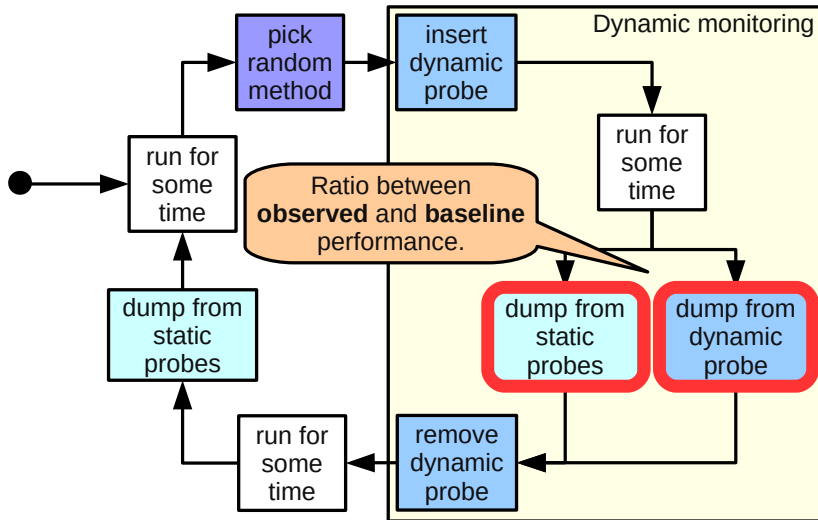


# Time Needed for Just-in-time Recompilation



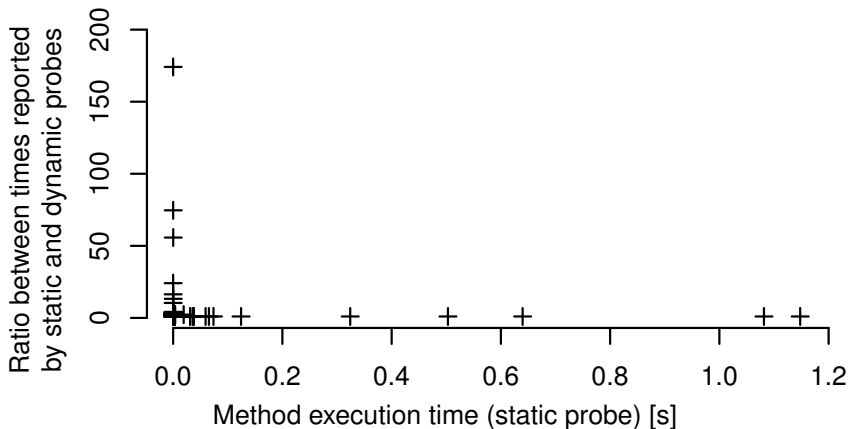
# Accuracy of Collected Data

# Accuracy of Collected Data

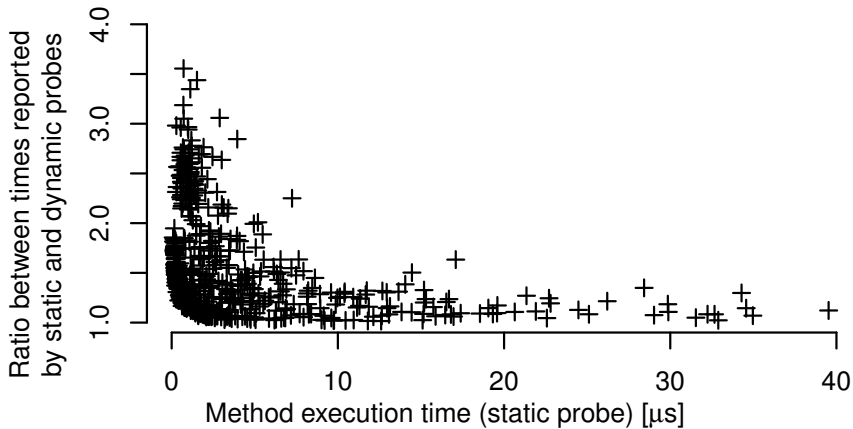




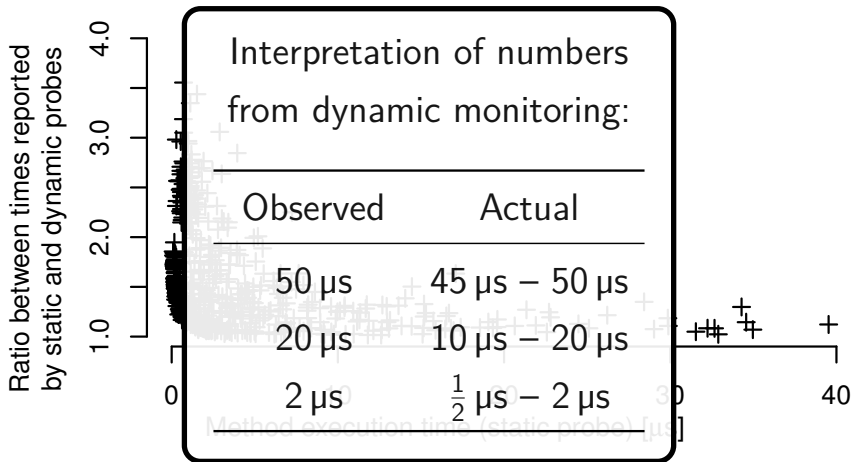
# Accuracy of Collected Data



# Accuracy of Collected Data

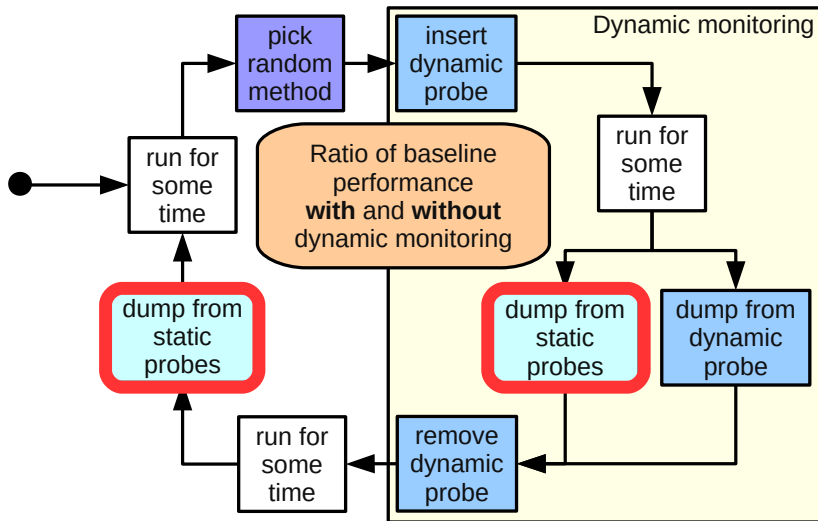


# Accuracy of Collected Data

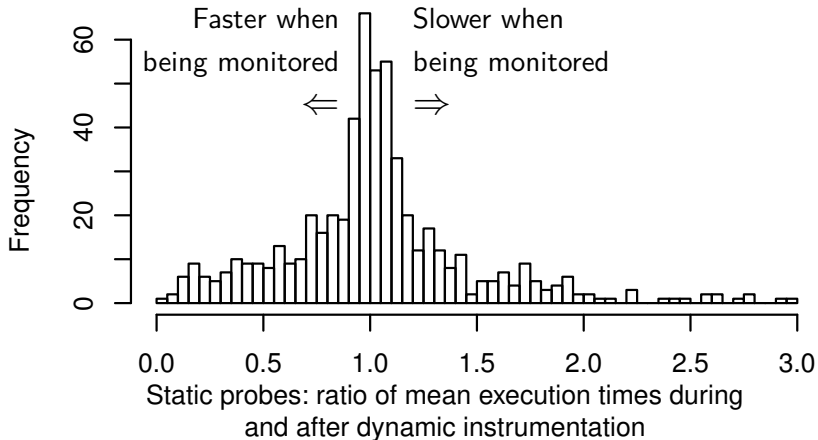


# Impact of Dynamic Monitoring

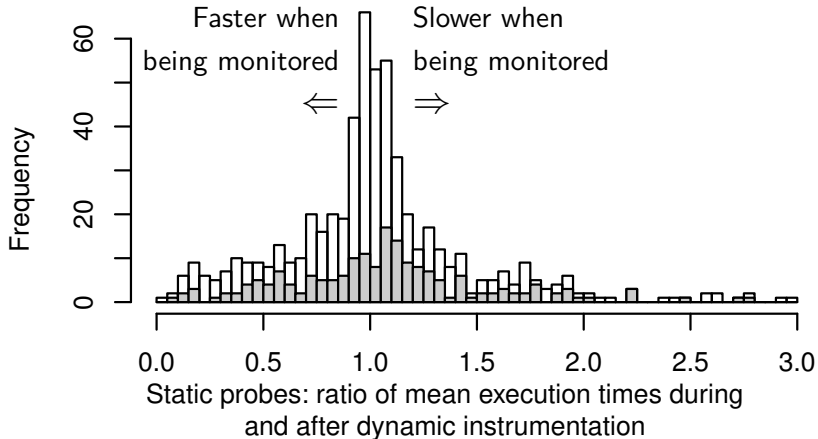
# Impact of Dynamic Monitoring



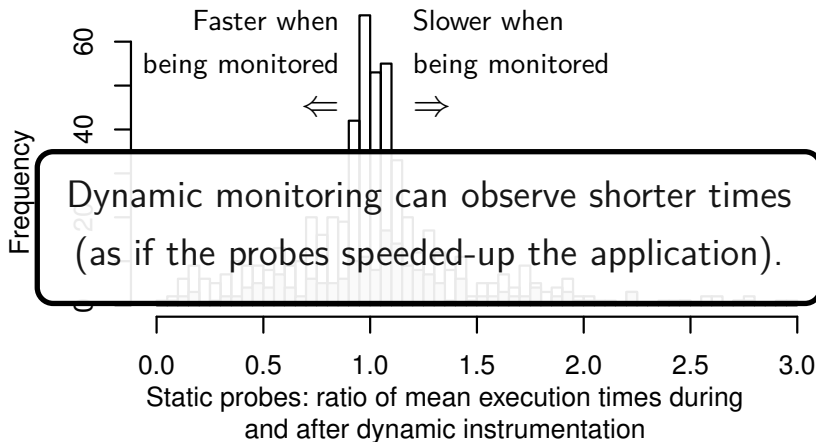
# Impact of Dynamic Monitoring



# Impact of Dynamic Monitoring



# Impact of Dynamic Monitoring





# Conclusion

# Analysis of Overhead in Dynamic Java Performance Monitoring

We evaluated how dynamic monitoring affects a running application and what is the accuracy of the obtained data.

# Analysis of Overhead in Dynamic Java Performance Monitoring

We evaluated how dynamic monitoring affects a running application and what is the accuracy of the obtained data.

## Rules of thumb coming from our experiment . . .

- Measuring one method at a time does not change CPU utilization.
- At least 30 s are needed for (JIT) recompilation.
- If the reported time is 30  $\mu$ s . . .
  - . . . the actual duration is between 20  $\mu$ s and 40  $\mu$ s  
(durations of at least 100  $\mu$ s are more “trustworthy”, though).

# Analysis of Overhead in Dynamic Java Performance Monitoring

We evaluated how dynamic monitoring affects a running application and what is the accuracy of the obtained data.

## Rules of thumb coming from our experiment . . .

- Measuring one method at a time does not change CPU utilization.
- At least 30 s are needed for (JIT) recompilation.
- If the reported time is 30  $\mu$ s . . .
  - . . . the actual duration is between 20  $\mu$ s and 40  $\mu$ s
  - (durations of at least 100  $\mu$ s are more “trustworthy”, though).

<http://d3s.mff.cuni.cz/resources/icpe2016>

# Analysis of Overhead in Dynamic Java Performance Monitoring

We evaluated how dynamic monitoring affects a running application and what is the accuracy of the obtained data.

## Rules of thumb coming from our experiment . . .

- Measuring one method at a time does not change CPU utilization.
- At least 30 s are needed for (JIT) recompilation.
- If the reported time is 30  $\mu$ s . . .
  - . . . the actual duration is between 20  $\mu$ s and 40  $\mu$ s
  - (durations of at least 100  $\mu$ s are more “trustworthy”, though).

<http://d3s.mff.cuni.cz/resources/icpe2016>

Thank You!