Garbage Collection for Monitoring Parametric Properties

Dongyun Jin, Patrick Meredith, Dennis Griffith, Grigore Rosu

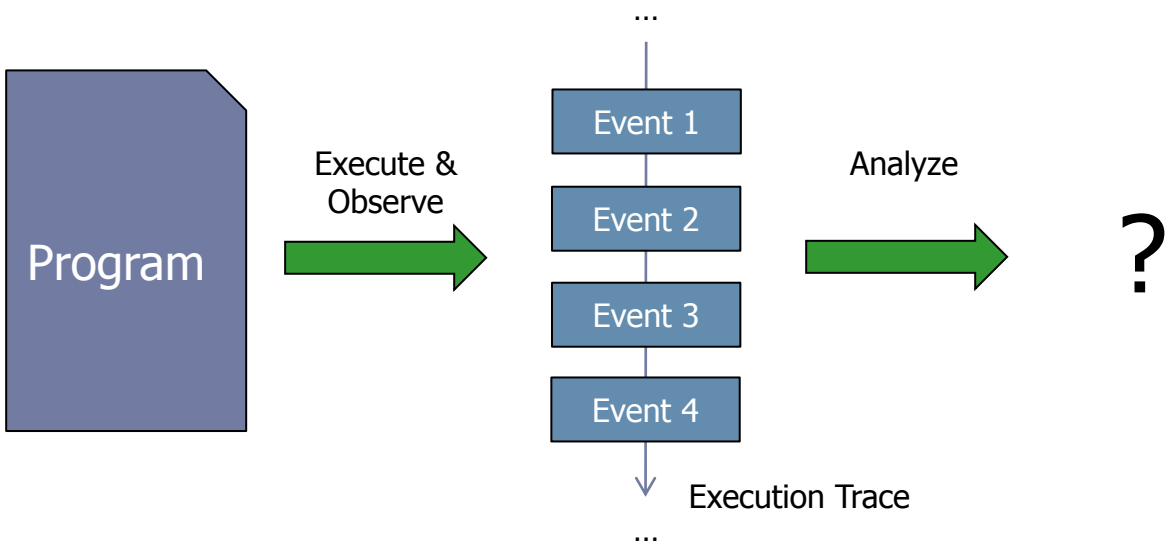
University of Illinois at Urbana-Champaign

Outline

- **Motivation**
- Property Static Analysis
- Double Lazy Garbage Collection
- Evaluation
- Conclusion

Monitoring

- Scalable technique for ensuring software reliability
- Observe run of a program



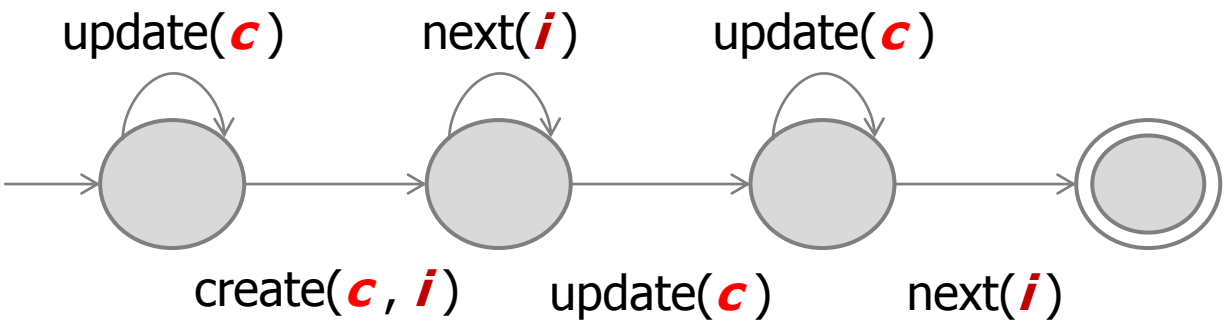
- Analyze execution trace against desired properties
- React/Report using handlers (if needed)

Applications of Monitoring

- Development
  - Debugging
  - Testing
- Deployment
  - Security
  - Reliability
  - Runtime Verification

Parametric Properties

- Properties referring to object instances
- The following property describes a bad behavior between Each Collection ***c*** and Iterator ***i*** :



- Generalize typestates
  - Typestates are parametric properties with one parameter

Parametric Property Monitoring Systems

- **Tracematches** [de Moor et al. 05]
  - Regular Patterns
- **PQL** [Lam et al. 05]
  - Context-Free Patterns
- **PTQL** [Aiken et al. 05]
  - SQL-like language
- **Eagle then RuleR** [Barringer et al. 04]
  - Rule-Based Rewrite Properties (e.g. Context-Free Patterns, ..)
- **MOP** [Rosu et al. 03]
  - Formalism Independent Approach
- many others

Parametric Monitoring in MOP

- Keep one monitor for each parameter instance
  - A parameter instance binds parameters to objects
  - E.g., $(c \mapsto c_2,\ i \mapsto i_3)$
- Each monitor knows nothing of parameters; operates exclusively on only one trace slice

Parametric Monitoring in MOP

- Keep one monitor for each parameter instance
  - A parameter instance binds parameters to objects
  - E.g., $(c \mapsto c_2,\ i \mapsto i_3)$
- Each monitor knows nothing of parameters; operates exclusively on only one trace slice

Trace: create(c_1, i_1) create(c_1, i_2) next(i_1) update(c_1) next(i_2)

Parametric Monitoring in MOP

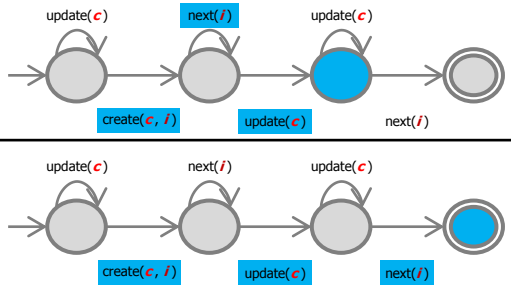
- Keep one monitor for each parameter instance
  - A parameter instance binds parameters to objects
  - E.g., $(c \mapsto c_2,\ i \mapsto i_3)$
- Each monitor knows nothing of parameters; operates exclusively on only one trace slice

Trace:	create(c_1, i_1)	create(c_1, i_2)	next(i_1)	update(c_1)	next(i_2)
(c_1, i_1) Trace Slice:	create		next	update	
(c_1, i_2) Trace Slice:		create		update	next

Parametric Monitoring in MOP

- Keep one monitor for each parameter instance
  - A parameter instance binds parameters to objects
  - E.g., $(c \mapsto c_2,\ i \mapsto i_3)$
- Each monitor knows nothing of parameters; operates exclusively on only one trace slice

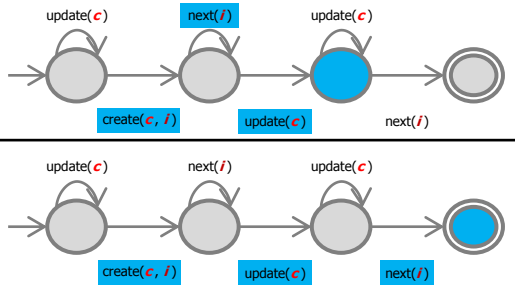
Trace:	create(c_1, i_1)	create(c_1, i_2)	next(i_1)	update(c_1)	next(i_2)
(c_1, i_1) Trace Slice:	create		next	update	
(c_1, i_2) Trace Slice:		create		update	next



Parametric Monitoring in MOP

- Keep one monitor for each parameter instance
  - A parameter instance binds parameters to objects
  - E.g., $(c \mapsto c_2,\ i \mapsto i_3)$
- Each monitor knows nothing of parameters; operates exclusively on only one trace slice

Trace:	create(c_1, i_1)	create(c_1, i_2)	next(i_1)	update(c_1)	next(i_2)
(c_1, i_1) Trace Slice:	create		next	update	
(c_1, i_2) Trace Slice:		create		update	next



Challenge: Large number of monitors

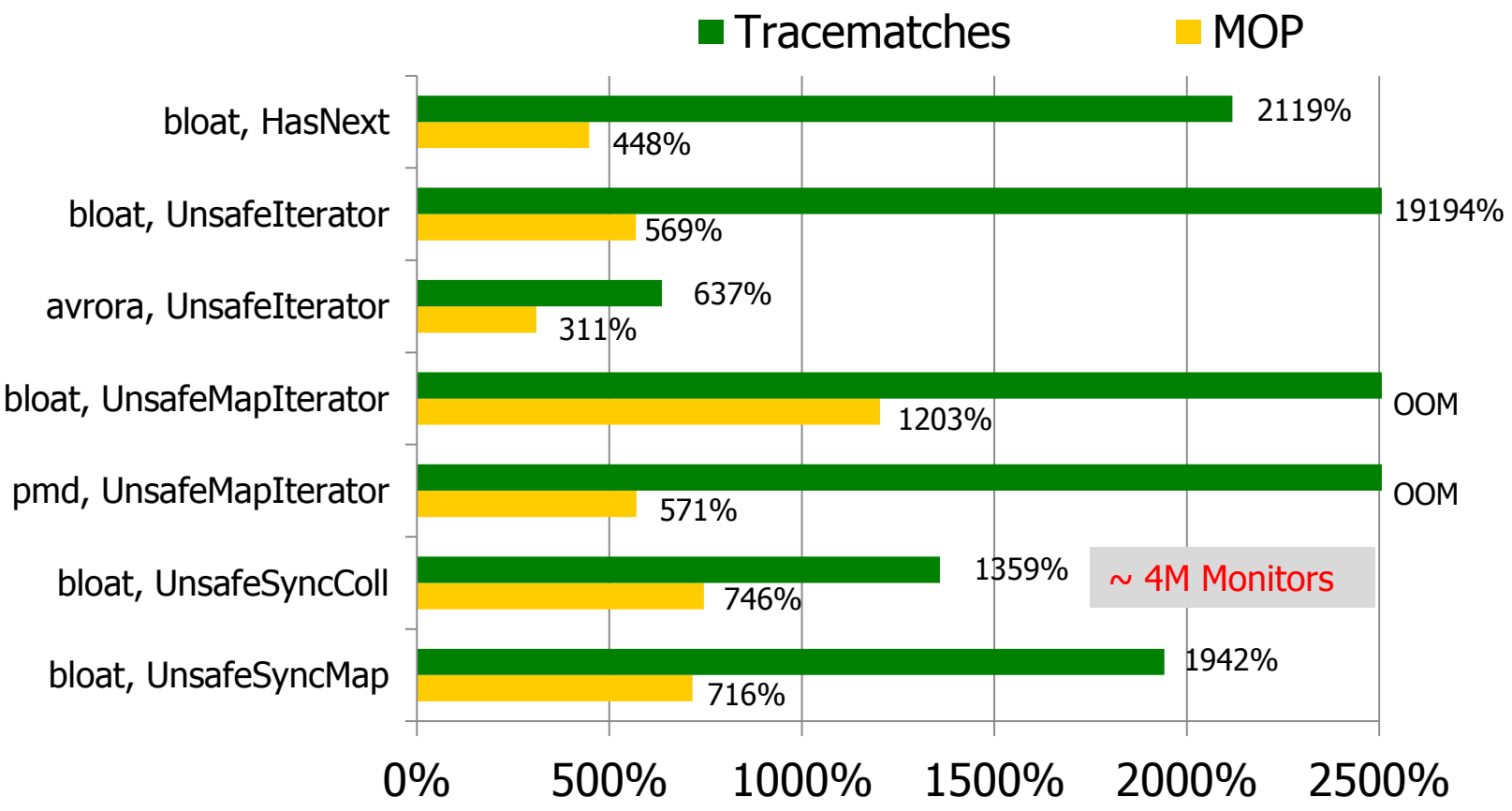
Previously…

- Benchmark based on DaCapo 9.12 [Blackburn et al. 06]
  - On average for 65 cases (13 examples × 5 properties)
- Tracematches [de Moor et al. 05]
  - One of the most memory efficient monitoring systems
  - 142% Runtime overhead [1]
  - 12% Memory overhead [1]
- MOP [Rosu et al. 03]
  - 33% Runtime overhead
  - 140% Memory overhead

1: Except 6(of 65) crashes

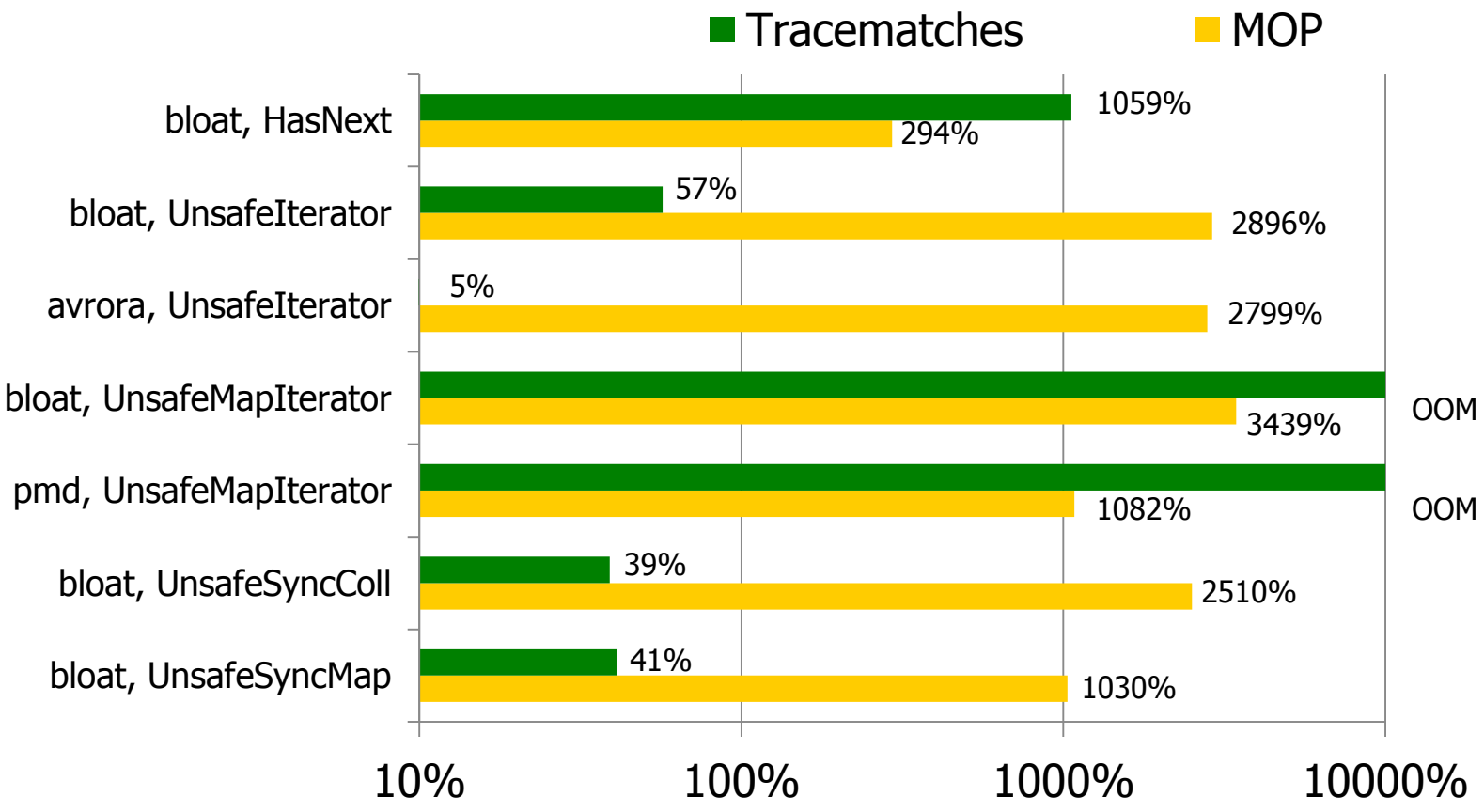
Worst Cases – Runtime Overhead

From DaCapo 9.12 and DaCapo 2006-10 MR2



Worst Cases – Memory Overhead

From DaCapo 9.12 and DaCapo 2006-10 MR2



Goal: Reduce the Runtime/Memory Overhead

- Static
  - Program Analysis
    [Bodden et al. 09, 10], [Dwyer et al. 10], …
  - Property Analysis
- Dynamic

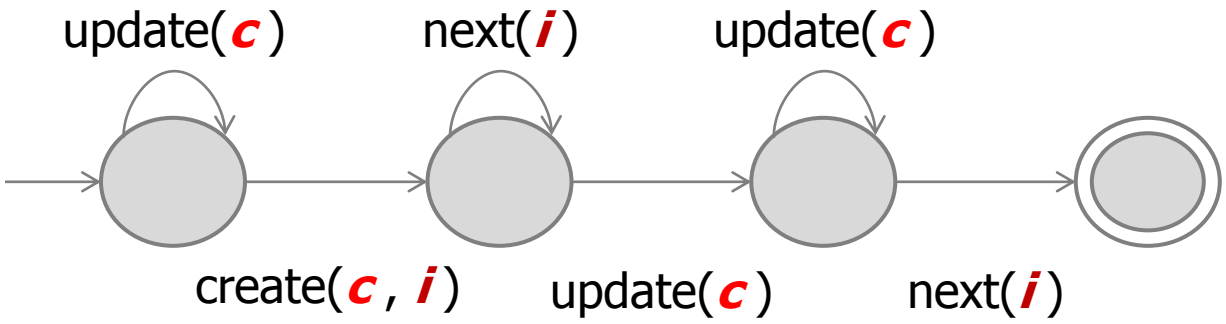
- By using information from property analysis,
  garbage collect monitors
  when they become useless during monitoring

Outline

- Motivation
- **Property Static Analysis**
- Double Lazy Garbage Collection
- Evaluation
- Conclusion

Property Static Analysis

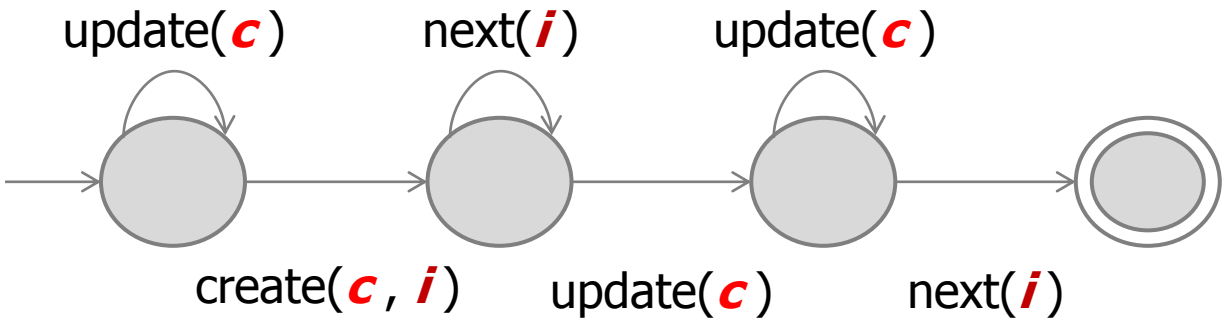
- Property analysis tells us when monitors are unnecessary
- Our analysis is formalism independent; we only show it for FSM monitors (see paper for CFG monitors)



The Last Event	Parameter instances that need to be alive

Property Static Analysis

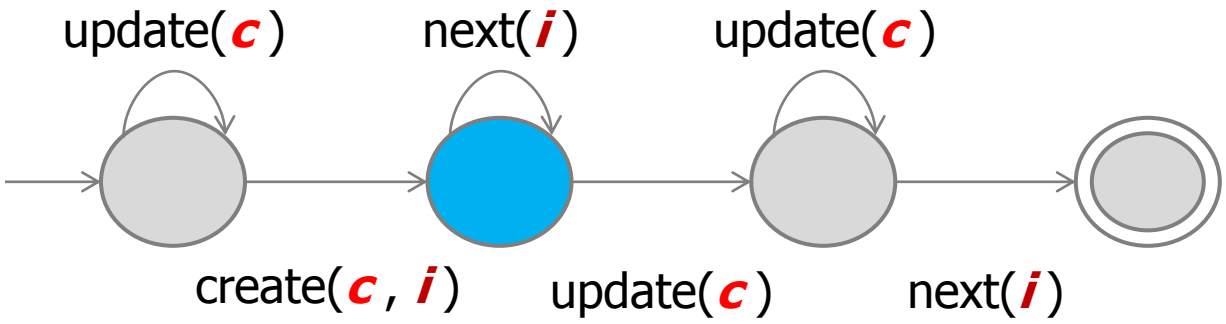
- Property analysis tells us when monitors are unnecessary
- Our analysis is formalism independent; we only show it for FSM monitors (see paper for CFG monitors)



The Last Event	Parameter instances that need to be alive
create	

Property Static Analysis

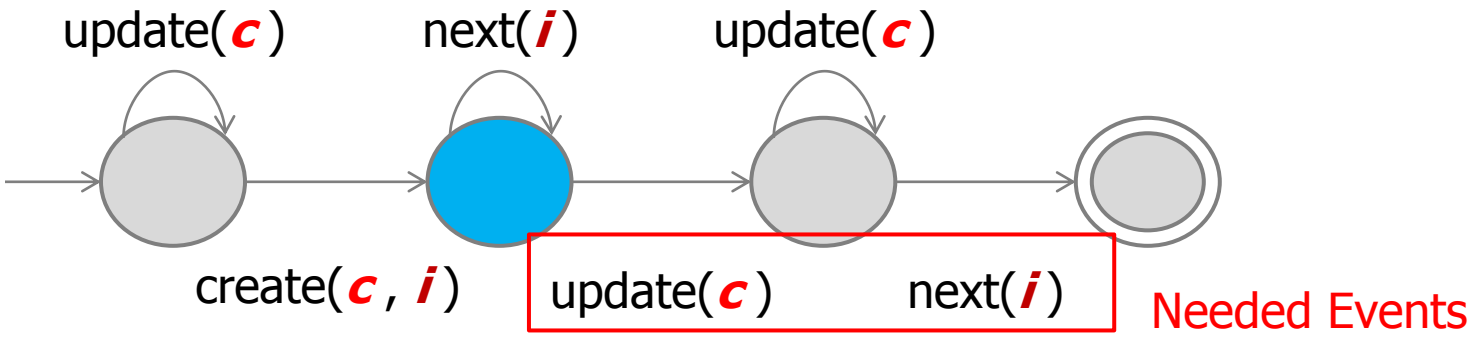
- Property analysis tells us when monitors are unnecessary
- Our analysis is formalism independent; we only show it for FSM monitors (see paper for CFG monitors)



The Last Event	Parameter instances that need to be alive
create	

Property Static Analysis

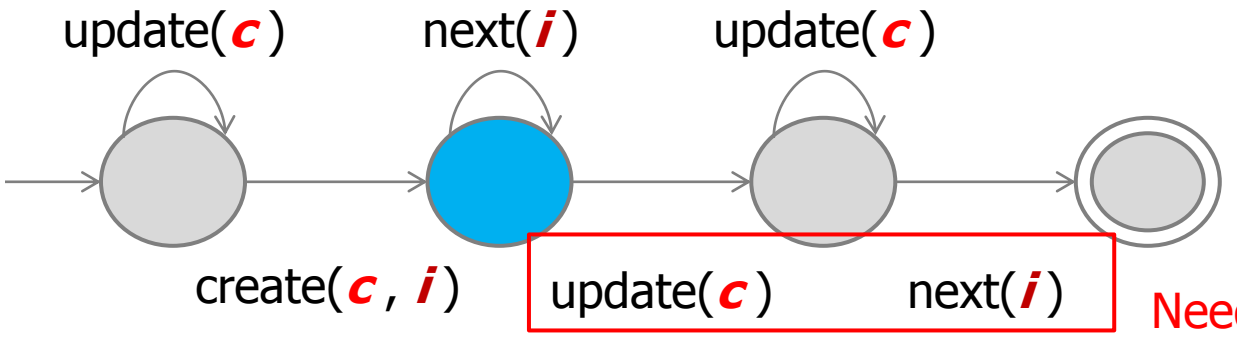
- Property analysis tells us when monitors are unnecessary
- Our analysis is formalism independent; we only show it for FSM monitors (see paper for CFG monitors)



The Last Event	Parameter instances that need to be alive
create	

Property Static Analysis

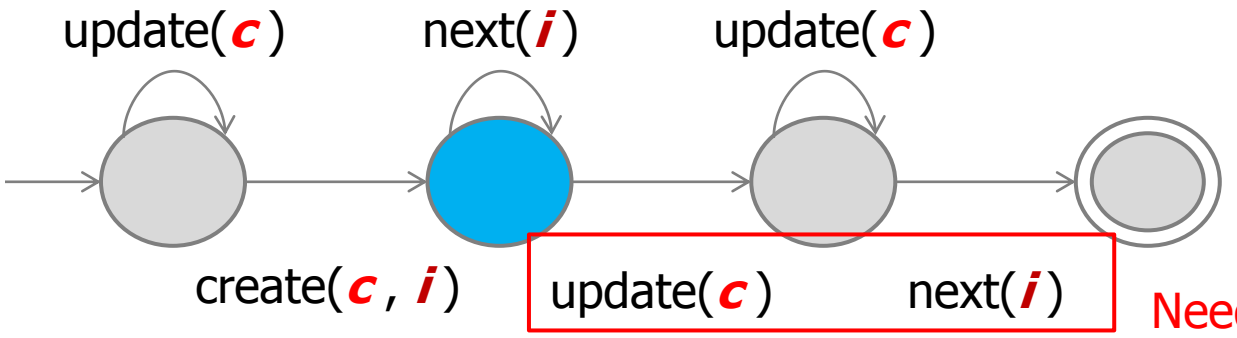
- Property analysis tells us when monitors are unnecessary
- Our analysis is formalism independent; we only show it for FSM monitors (see paper for CFG monitors)



The Last Event	Parameter instances that need to be alive
create	{c, i}

Property Static Analysis

- Property analysis tells us when monitors are unnecessary
- Our analysis is formalism independent; we only show it for FSM monitors (see paper for CFG monitors)

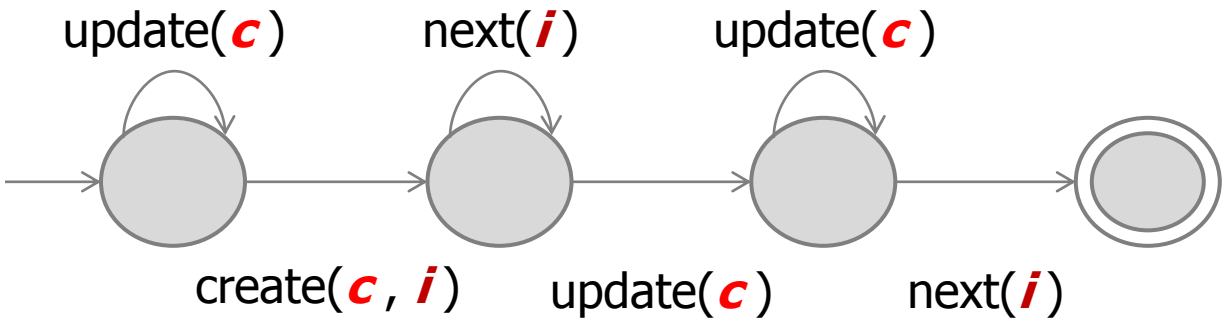


The Last Event	Parameter instances that need to be alive
create	{c, i}

If any of these parameters are garbage collected, this monitor cannot trigger; can be collected

Property Static Analysis

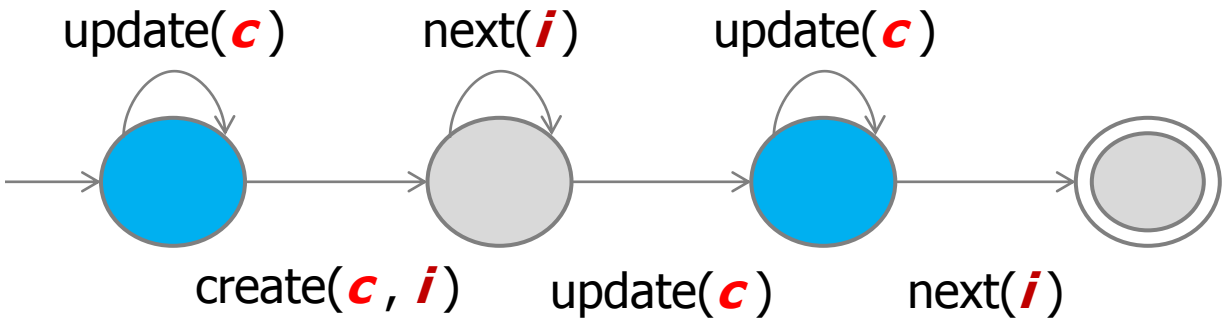
- Property analysis tells us when monitors are unnecessary
- Our analysis is formalism independent; we only show it for FSM monitors (see paper for CFG monitors)



The Last Event	Parameter instances that need to be alive
create	{c, i}
update	

Property Static Analysis

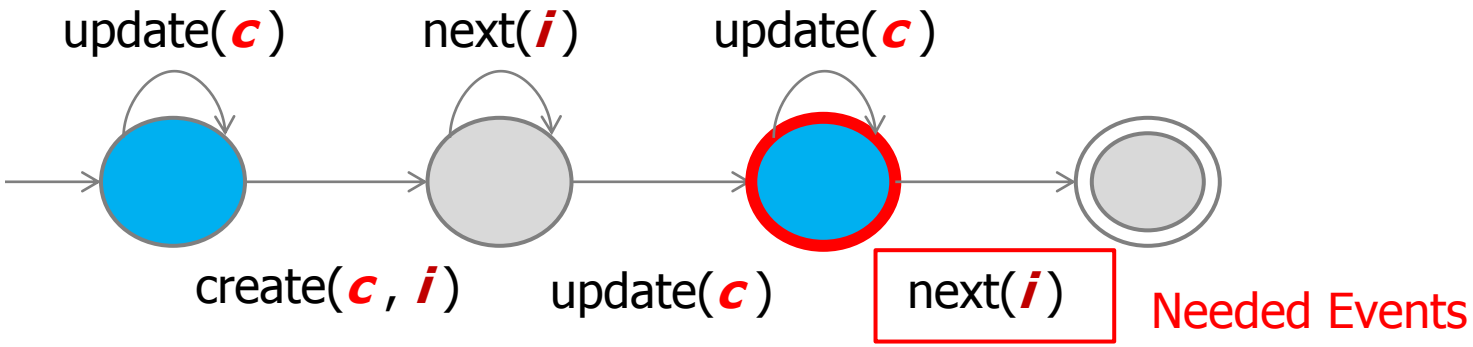
- Property analysis tells us when monitors are unnecessary
- Our analysis is formalism independent; we only show it for FSM monitors (see paper for CFG monitors)



The Last Event	Parameter instances that need to be alive
create	{c, i}
update	

Property Static Analysis

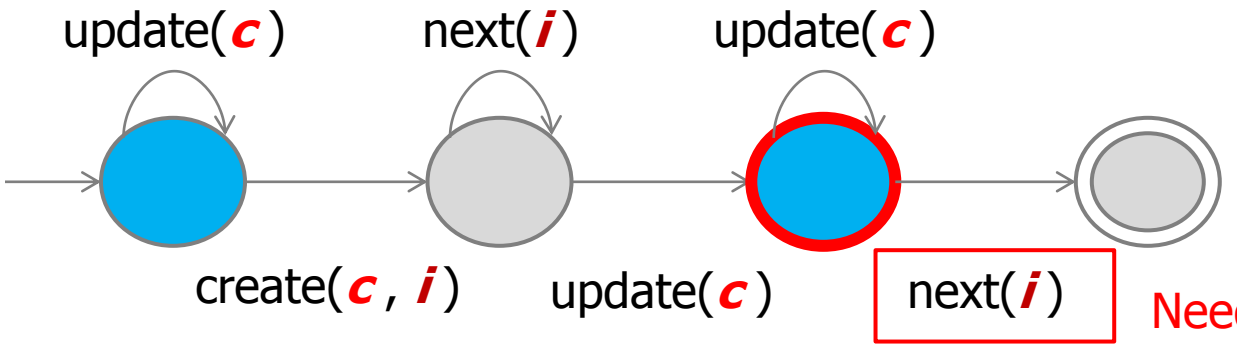
- Property analysis tells us when monitors are unnecessary
- Our analysis is formalism independent; we only show it for FSM monitors (see paper for CFG monitors)



The Last Event	Parameter instances that need to be alive
create	{c, i}
update	

Property Static Analysis

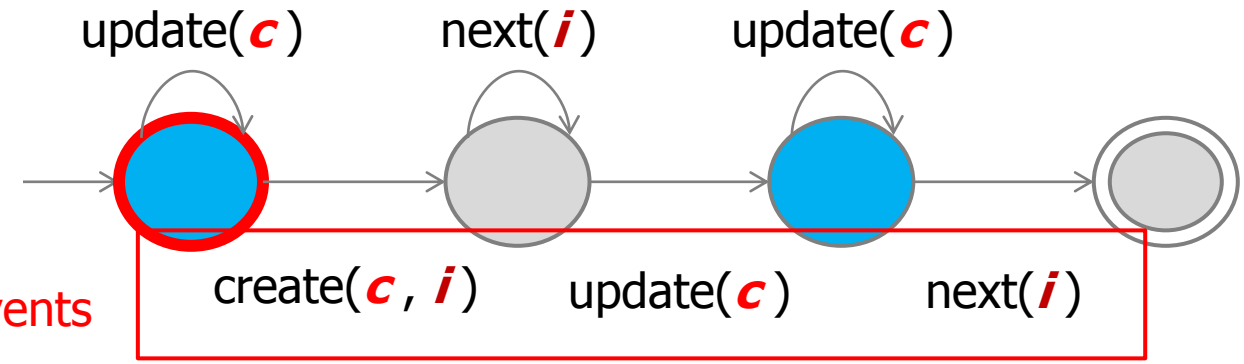
- Property analysis tells us when monitors are unnecessary
- Our analysis is formalism independent; we only show it for FSM monitors (see paper for CFG monitors)



The Last Event	Parameter instances that need to be alive
create	{c, i}
update	{i}

Property Static Analysis

- Property analysis tells us when monitors are unnecessary
- Our analysis is formalism independent; we only show it for FSM monitors (see paper for CFG monitors)

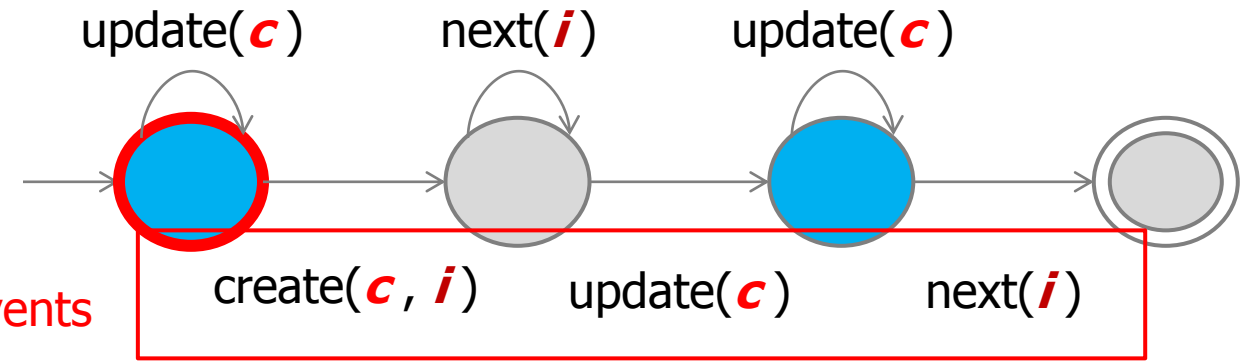


Needed Events

The Last Event	Parameter instances that need to be alive
create	{c, i}
update	{i}

Property Static Analysis

- Property analysis tells us when monitors are unnecessary
- Our analysis is formalism independent; we only show it for FSM monitors (see paper for CFG monitors)

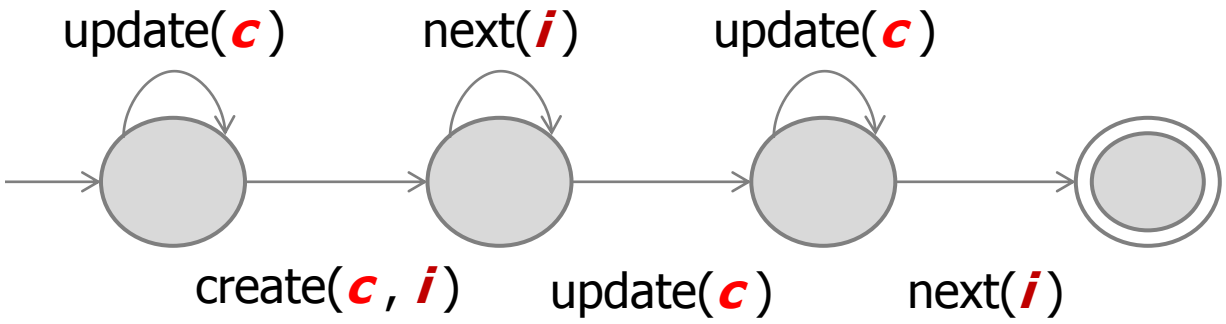


Needed Events

The Last Event	Parameter instances that need to be alive
create	{c, i}
update	{i} {c, i}

Property Static Analysis

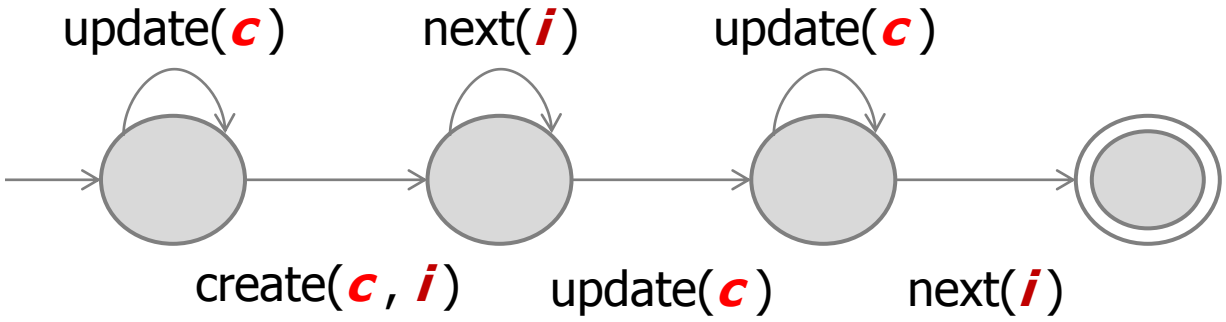
- Property analysis tells us when monitors are unnecessary
- Our analysis is formalism independent; we only show it for FSM monitors (see paper for CFG monitors)



The Last Event	Parameter instances that need to be alive
create	{c, i}
update	{i} {c, i}
next	{c, i}

Property Static Analysis

- Property analysis tells us when monitors are unnecessary
- Our analysis is formalism independent; we only show it for FSM monitors (see paper for CFG monitors)



The Last Event	Parameter instances that need to be alive
create	{c, i}
update	{i} {c, i}
next	{c, i}

If any of these are dead, then the current monitor can be collected

Outline

- Motivation
- Property Static Analysis
- **Double Lazy Garbage Collection**
- Evaluation
- Conclusion

Double Lazy Garbage Collection

- *Indexing Trees* map parameter instances to monitors

- Lazy propagation of info about dead parameter instances
  - Eager propagation yields high runtime overhead (like TM)
  - Idea: propagate when monitors are updated for other reasons

- Lazy removal of useless monitors
  - Removal is expensive (monitor can belong to many trees)
  - Idea: just mark unnecessary monitors, then clean up later

(technical; see paper for details and proof of correctness)

Outline

- Motivation
- Property Static Analysis
- Double Lazy Garbage Collection
- **Evaluation**
- Conclusion

Evaluation – Overall

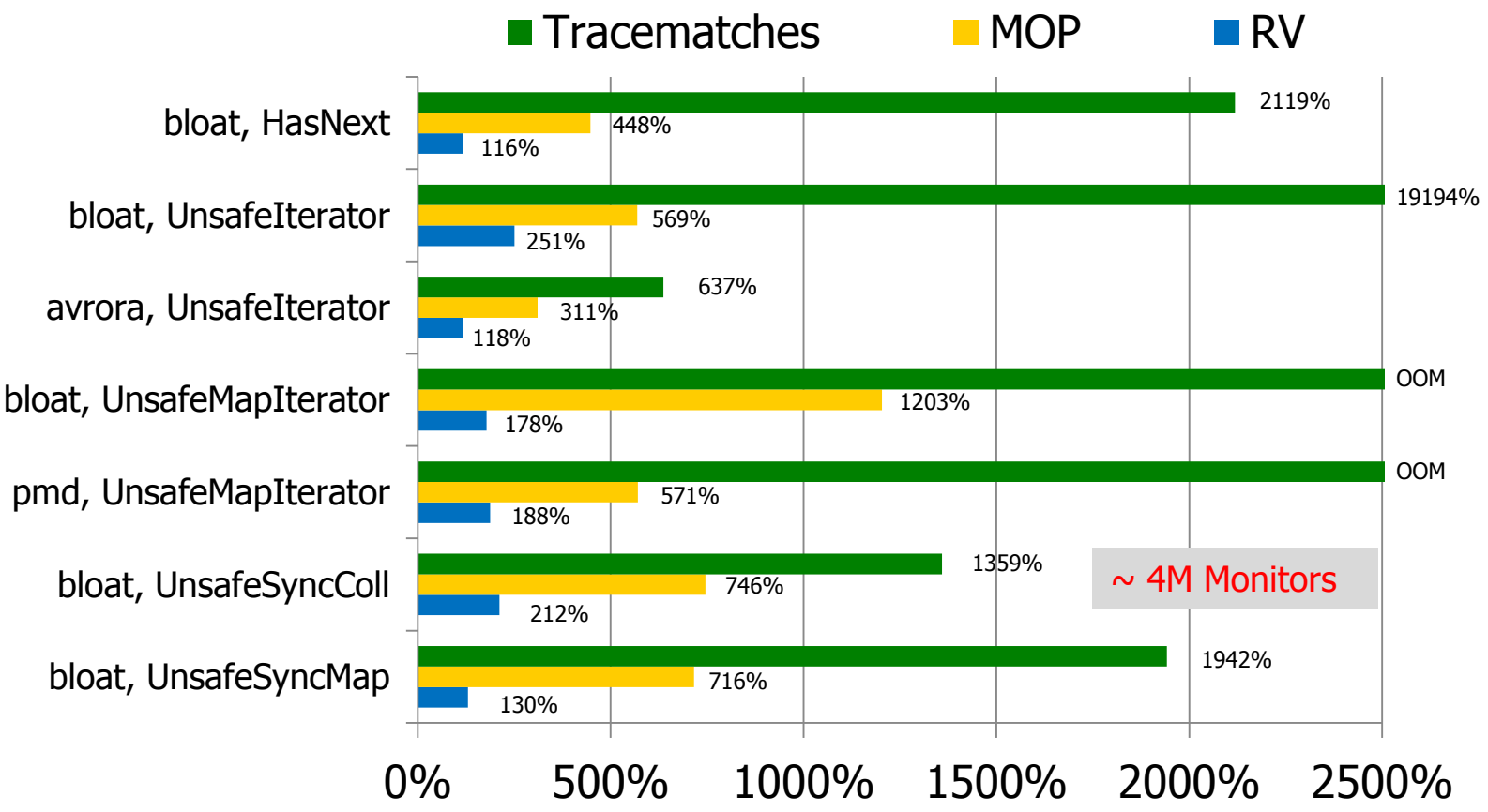
- On average, based on the DaCapo 9.12 Benchmark
  - 13 Programs and 5 Properties

	Tracematches	MOP	RV (MOP + Garbage Collection)
Runtime Overhead	142%[1]	33%	**15%**
Memory Overhead	12%[1]	140%	**39%**

1: Except 6(of 65) crashes

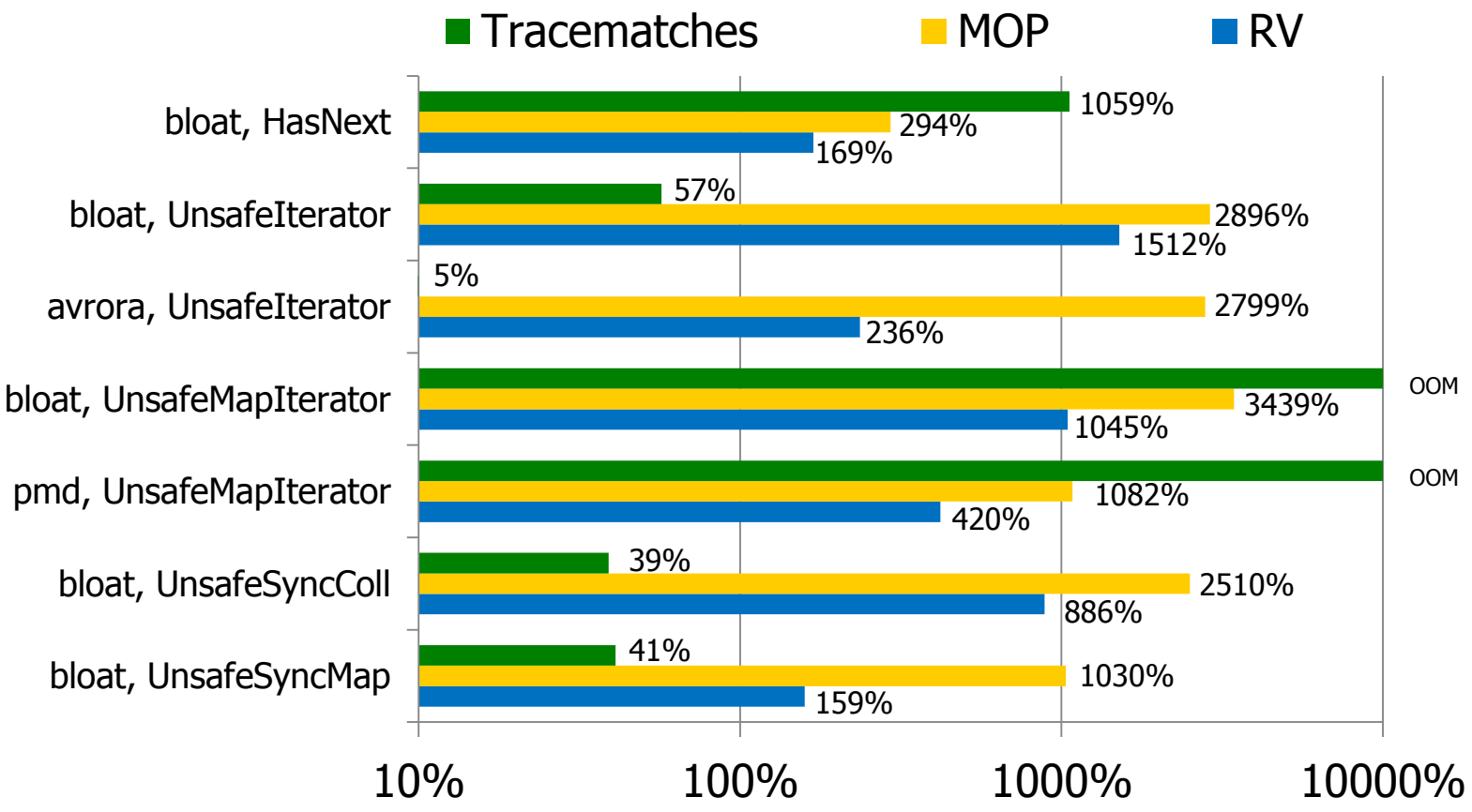
Evaluation – Worst Cases Runtime Overhead

From DaCapo 9.12 and DaCapo 2006-10 MR2



Evaluation – Worst Cases Memory Overhead

From DaCapo 9.12 and DaCapo 2006-10 MR2



Conclusions

- Garbage collection for parametric monitoring
  - Formalism-independent
  - Directed by property static analysis
- Double Lazy Garbage Collection
  - Lazy propagation of info
  - Lazy removal of monitors
- Evaluation
  - Lowest runtime overhead parametric monitoring system
  - Reasonable memory overhead