Efficient Formalism-Independent Monitoring of Parametric Properties

Feng Chen **Patrick Meredith** Dongyun Jin Grigore Rosu
University of Illinois at Urbana-Champaign
2009-11-20

Overview

- Motivation
- Parametric Monitoring
- Enable Sets Based Optimization
- Results
- Conclusion

Monitoring Examples

- Require authentication before allowing access
- Events
  - authenticate - when the program authenticates
  - access - just before the program accesses the resource
- Property (using past time linear temporal logic)
  - access → *eventually in the past* authenticate
- Handler
  - Perform the authenticate

Monitoring Examples

- No write after file close
- Events
  - open - the open call for a file
  - write - just before a write to a file
  - close - the close call for a file
- Property (using a regular expression)
  - open write* close write
- Handler
  - reopen the file, or disallow the write with a warning

Monitoring Examples

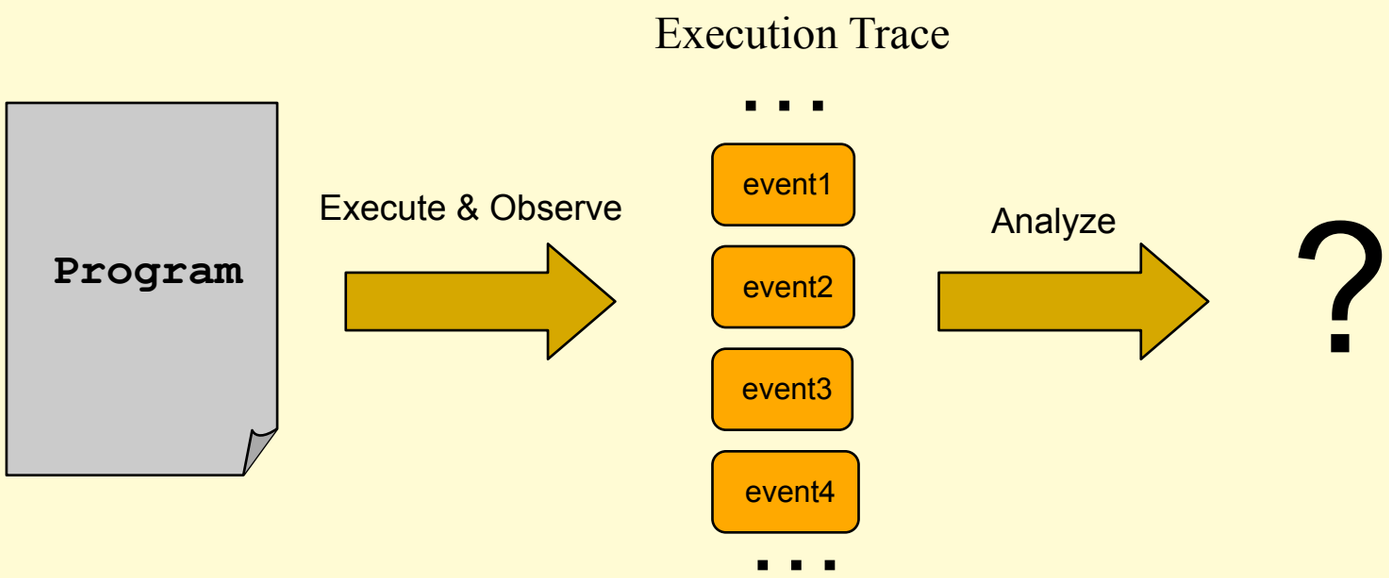
- Releases and acquires of a lock must be matched
- Events
  - begin - begin of a method
  - end - end of a method
  - acq - acquire of a lock
  - rel - release of a lock
- Property (using a context free grammar)
  - S → ε | S S | begin S end | acq S rel
- Handler
  - Issue an error message

Applications of Monitoring

- Debugging
  - Deployment - development
  - Handler - error messages
- Testing
  - Deployment - development
  - Handler - error messages
- Security/Reliablity/Runtime Verification
  - Deployment - production systems
  - Handler - recovery code

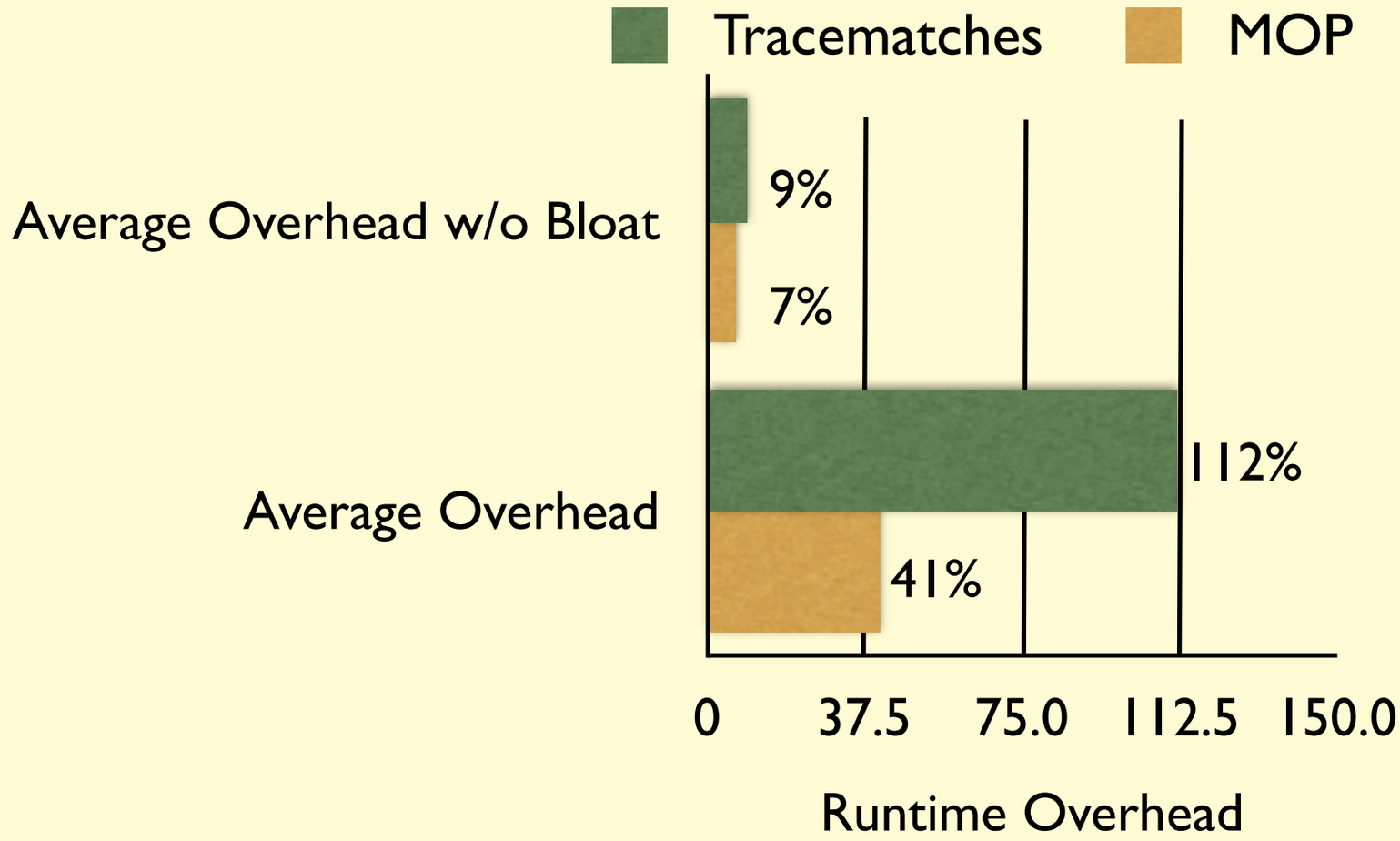
Monitoring in a Nutshell

- Observe run of a system



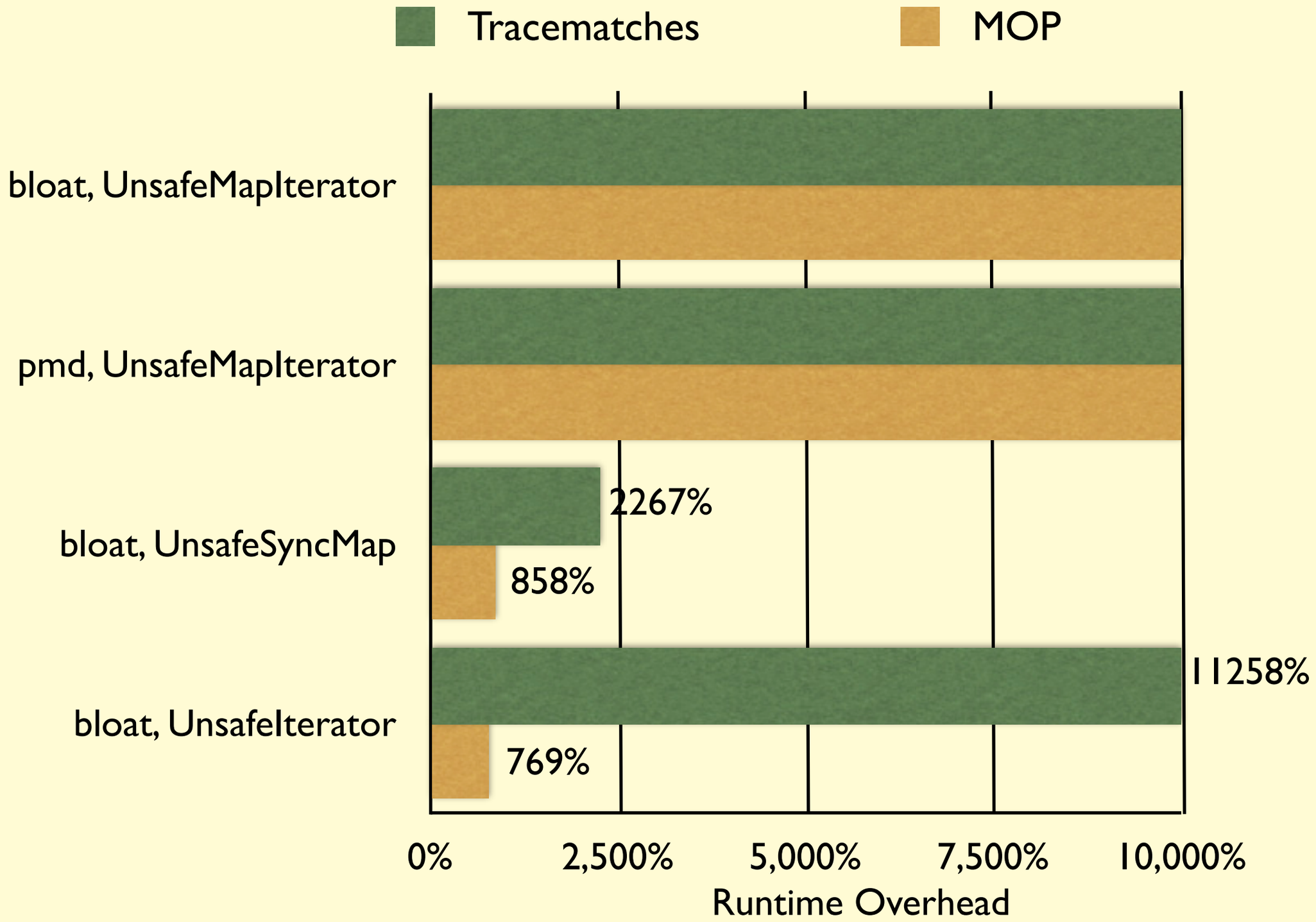
- Analyze it against desired properties
- React/Report using handlers (if needed)
- Scalable!

Expected Overheads

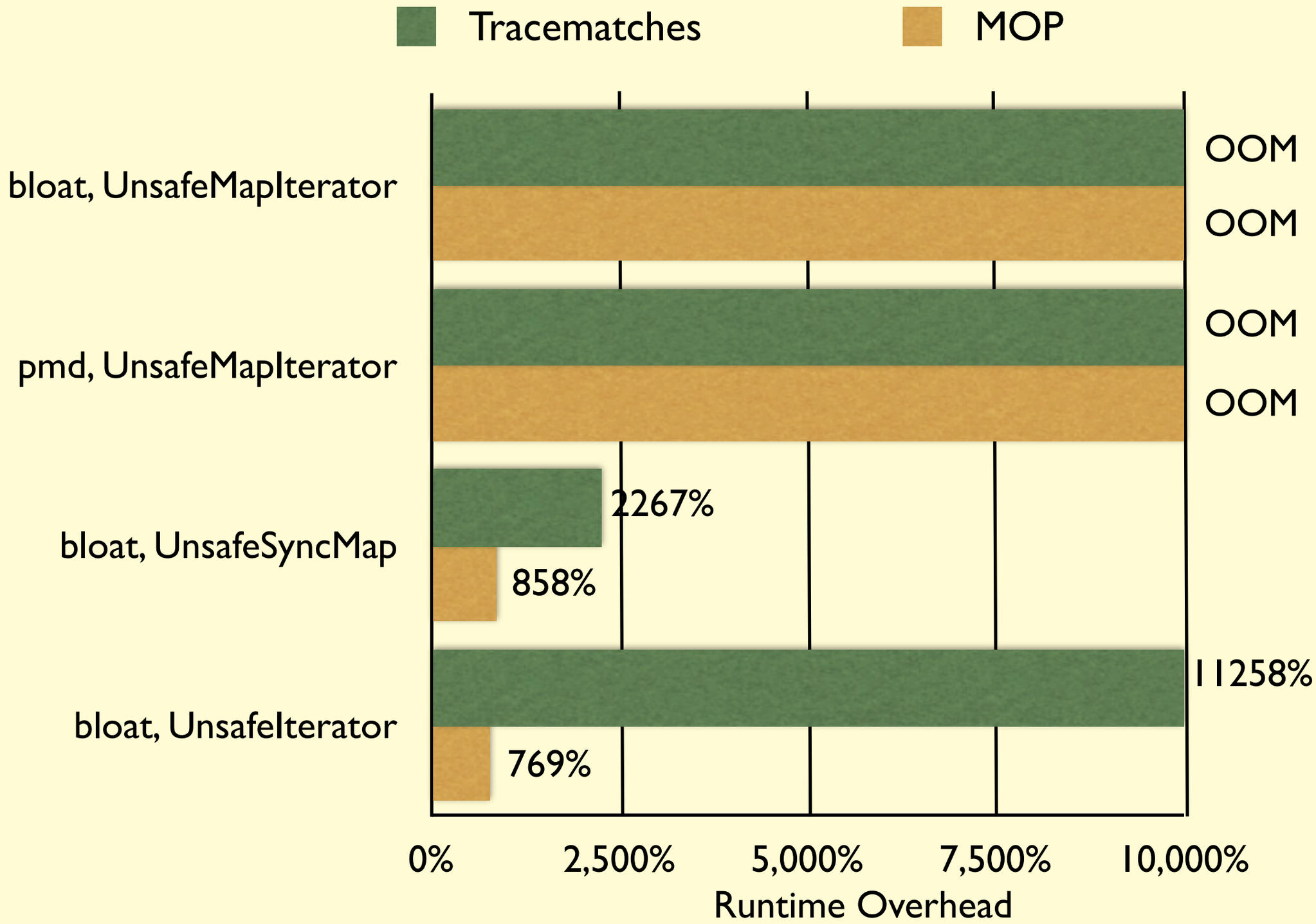


- Tracematches - competing system
- MOP - our system (without optimization)

Bad Overheads



Bad Overheads



Monitoring Systems

- MAC (UPenn)
- PAX (NASA)
- TimeRover (commercial)
- HAWK/Eagle (NASA)
- MOP (UIUC)
- POTA (UTA)
- PQL (Stanford)
- Tracematches (Oxford)
- PTQL (Berkeley/Stanford/Novell)
- Pal (UPenn)

Overview

- Motivation
- Parametric Monitoring
- Enable Sets Based Optimization
- Results
- Conclusions and Future Work

Parametric Monitoring Example

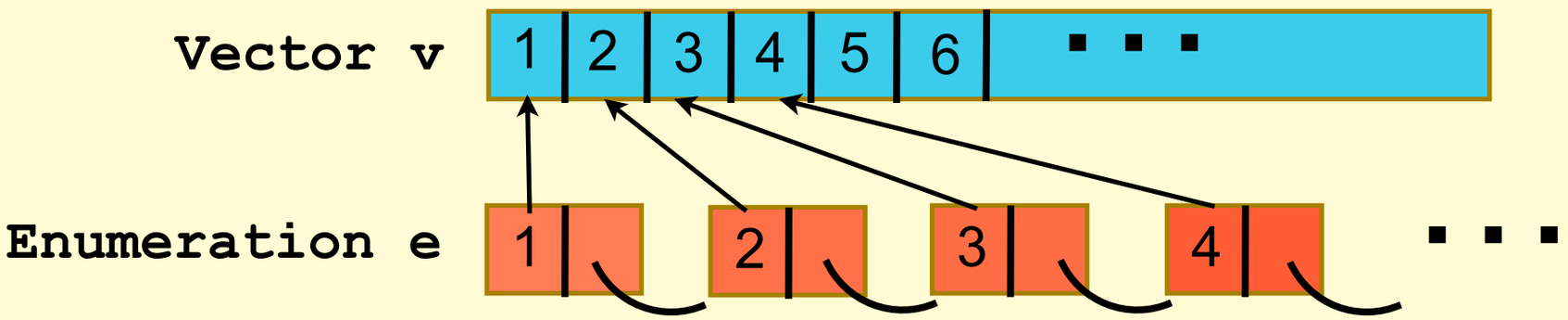
- Actually almost all properties are parametric
  - Events carry parameters
- We wish to apply authenticate before use property to *all* resources in a system
  - access<r> → *eventually in the past* authenticate<r>
- Allows for multiple resources
  - authenticate<r> may require a specific identification per resource

More Parametric Properties

- No write after the close of a specific file
  - open<f> write<f>* close<f> write<f>
  - Without parameters one needs to write this property for *each* file
  - Or close of one file may trigger a handler on write to a *different* file!
- Releases and acquires of a lock must be matched
  - S → ε | S S | begin<t> S end<t> | acq<l> S rel<l>
  - We only wish to match acquires and release to the *same* lock
  - The t (thread) parameter can further ensure that the locks are matched within a given thread

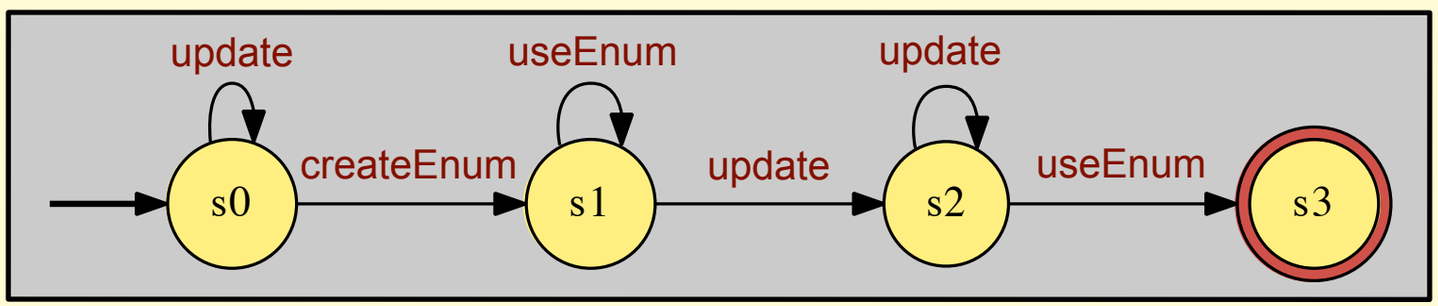
Running Example: Unsafe Enumeration

- One should not change a vector while being accessed via one or more enumeration objects



Unsafe Enumeration as a Parametric Property

- Violation pattern of three events
  - update<v> : change in vector v
  - createEnum<v,e> : create enumeration e from vector v
  - useEnum<e> : use enumeration e
- update* createEnum useEnum* update+ useEnum



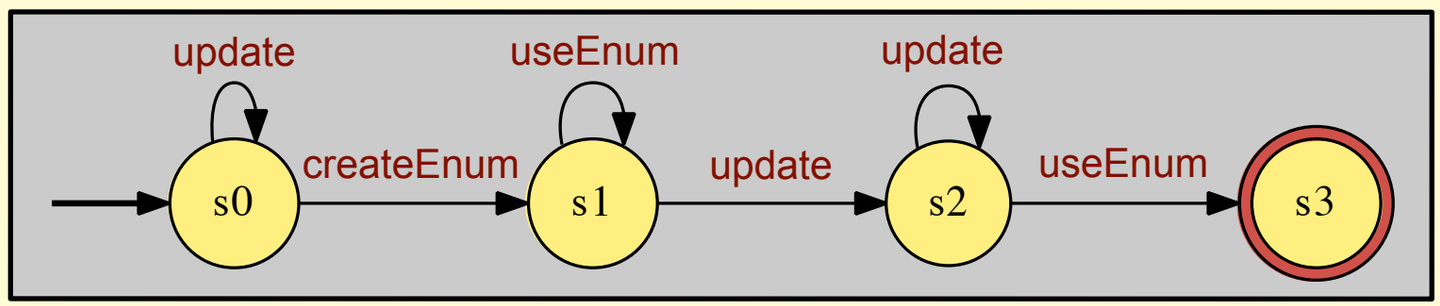
Generic Parametric Monitoring Idea

[TACAS 2009]

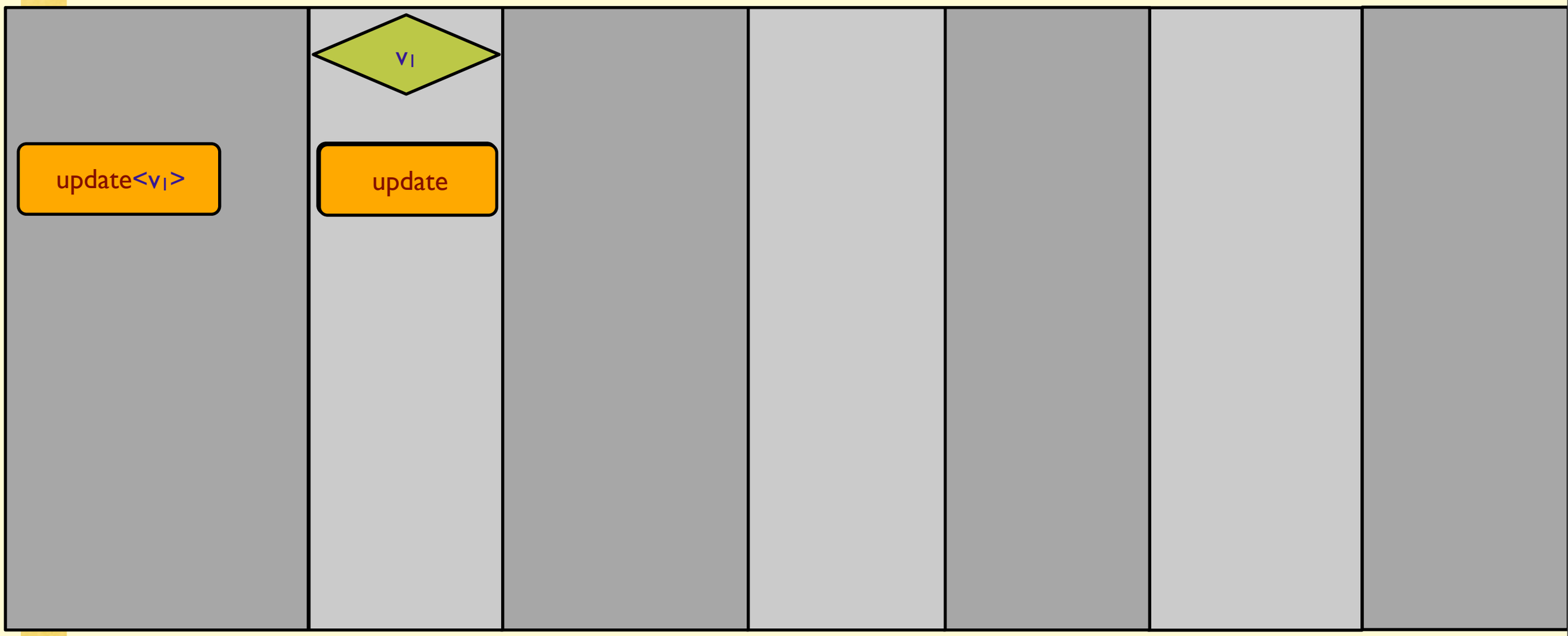
- Keep one monitor for each *parameter instance*
  - A parameter instance is the bindings of specific objects to its parameter
  - E.g., $\langle v \rightarrow v_2, e \rightarrow e_3 \rangle$
- Each monitor knows nothing of parameters, operates on one trace *slice*

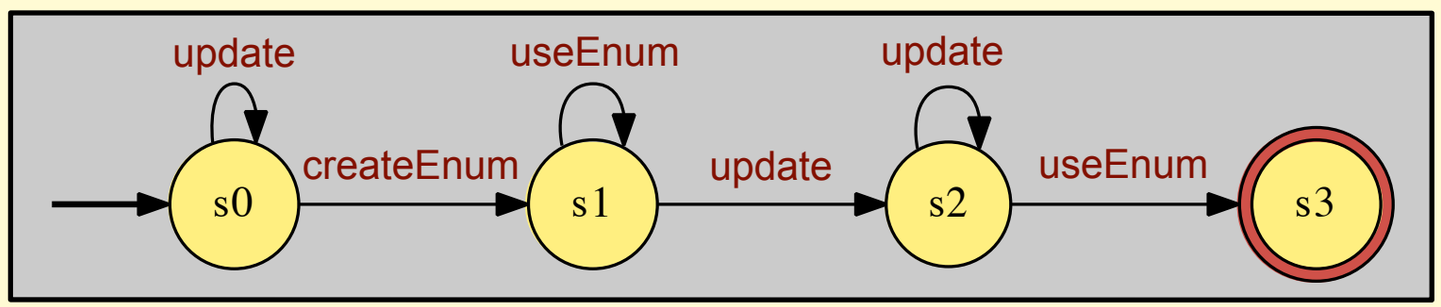
Possible trace - createEnum$\langle v_1,e_1 \rangle$ createEnum$\langle v_1,e_2 \rangle$ useEnum$\langle e_1 \rangle$ update$\langle v_1 \rangle$ useEnum$\langle e_2 \rangle$
$\langle v_1,e_1 \rangle$ slice - createEnum useEnum update
$\langle v_1, e_2 \rangle$ slice - createEnum update useEnum

Example Run

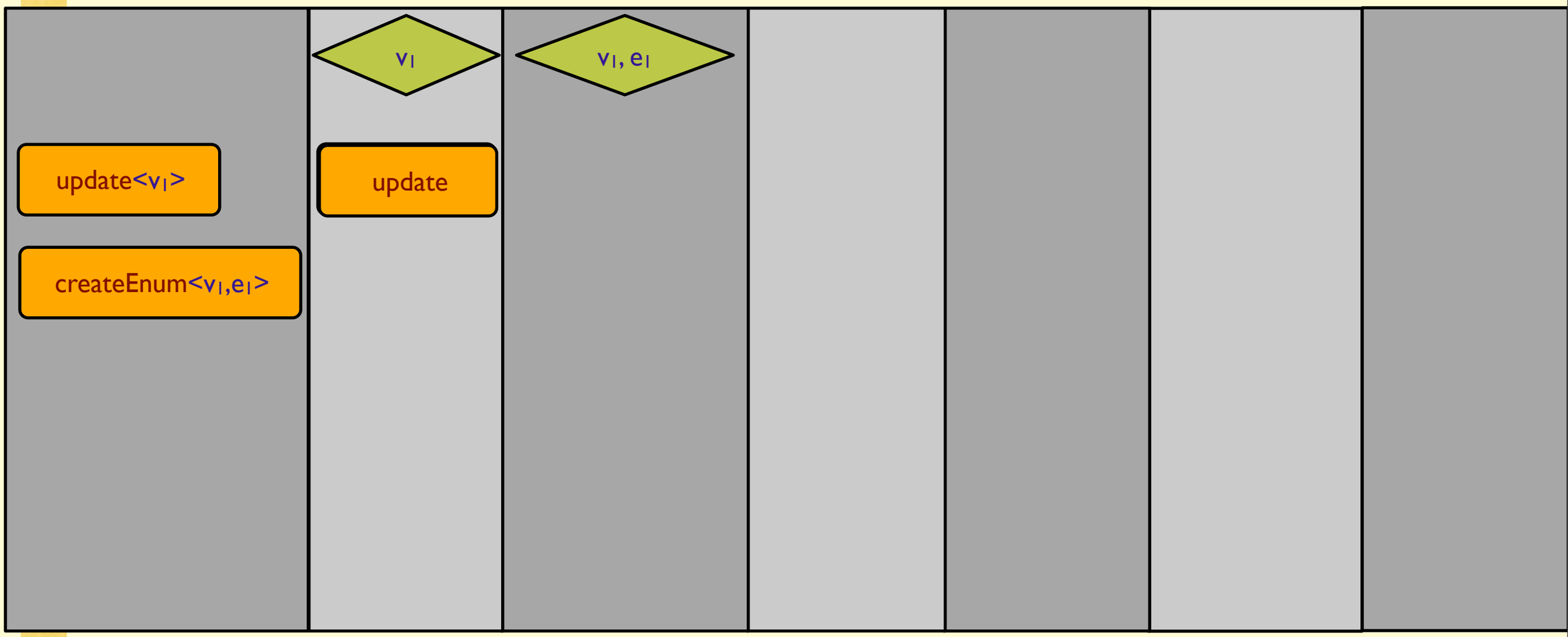

update
useEnum
update
createEnum
update
useEnum
s0
s1
s2
s3

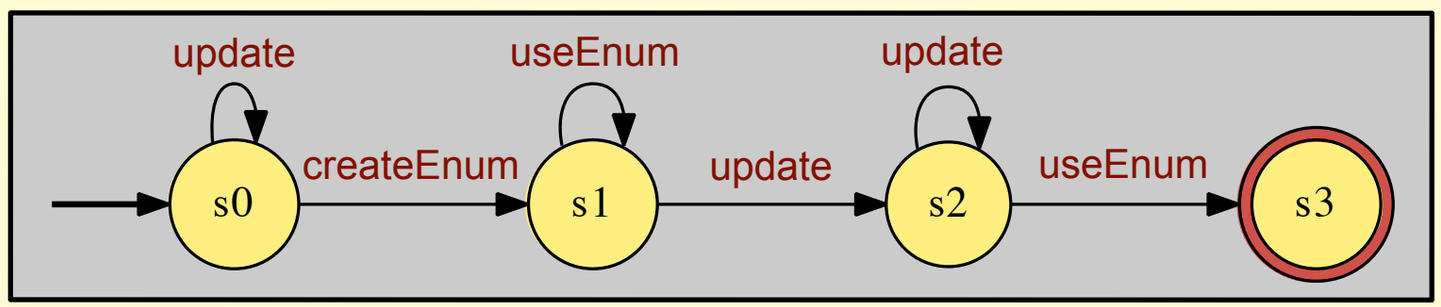
Example Run


v1
update<v1>
update

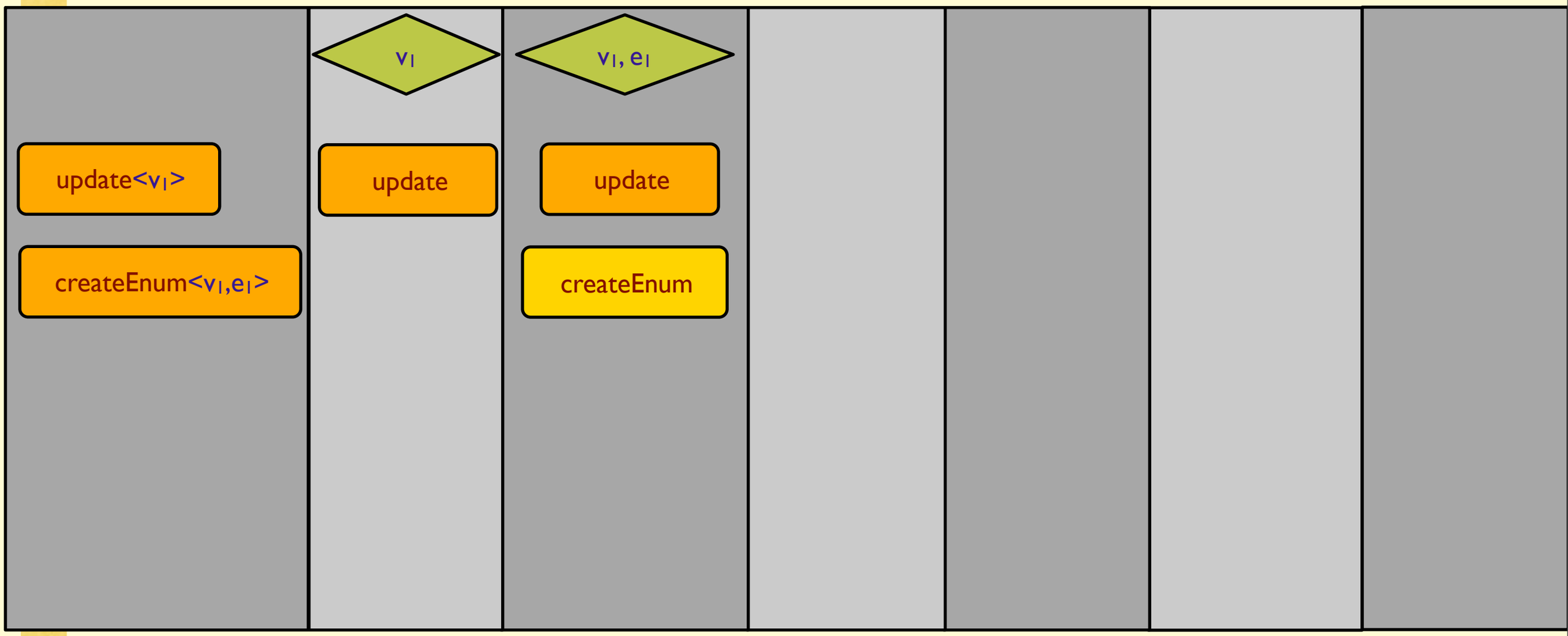

update
useEnum
update
createEnum
update
useEnum
s0
s1
s2
s3

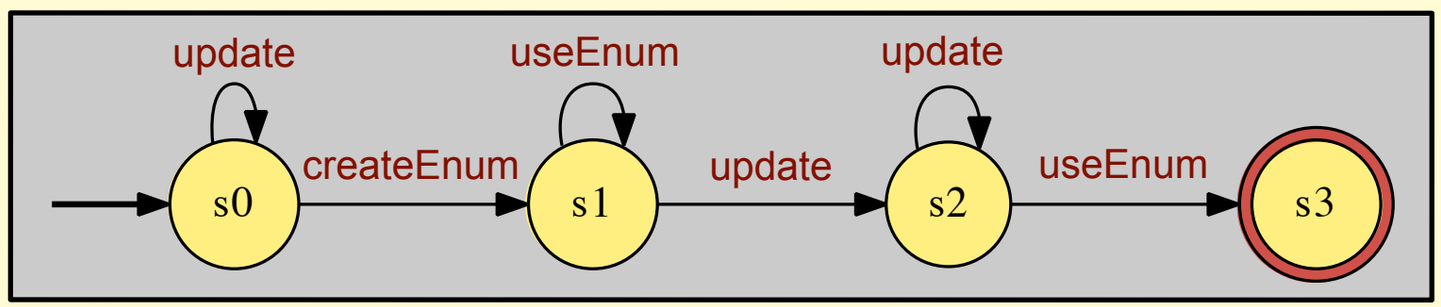
Example Run


v_I
v_I, e_I
update<v_I>
update
createEnum<v_I,e_I>

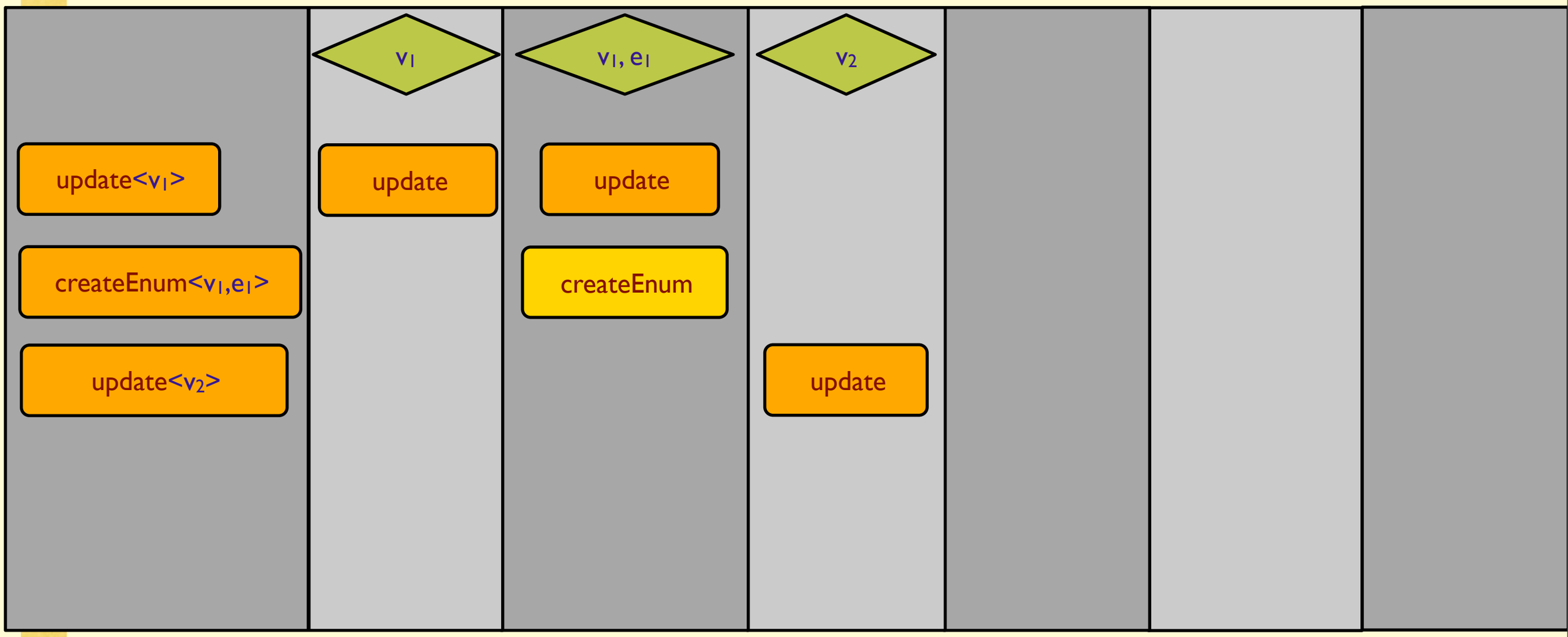

update
useEnum
update
createEnum
update
useEnum
s0
s1
s2
s3

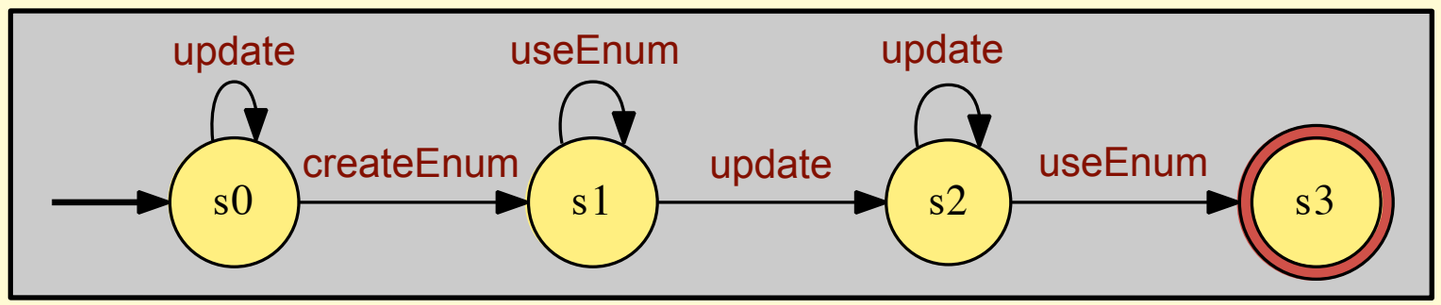
Example Run


v_1
v_1, e_1
update<v_1>
update
update
createEnum<v_1,e_1>
createEnum

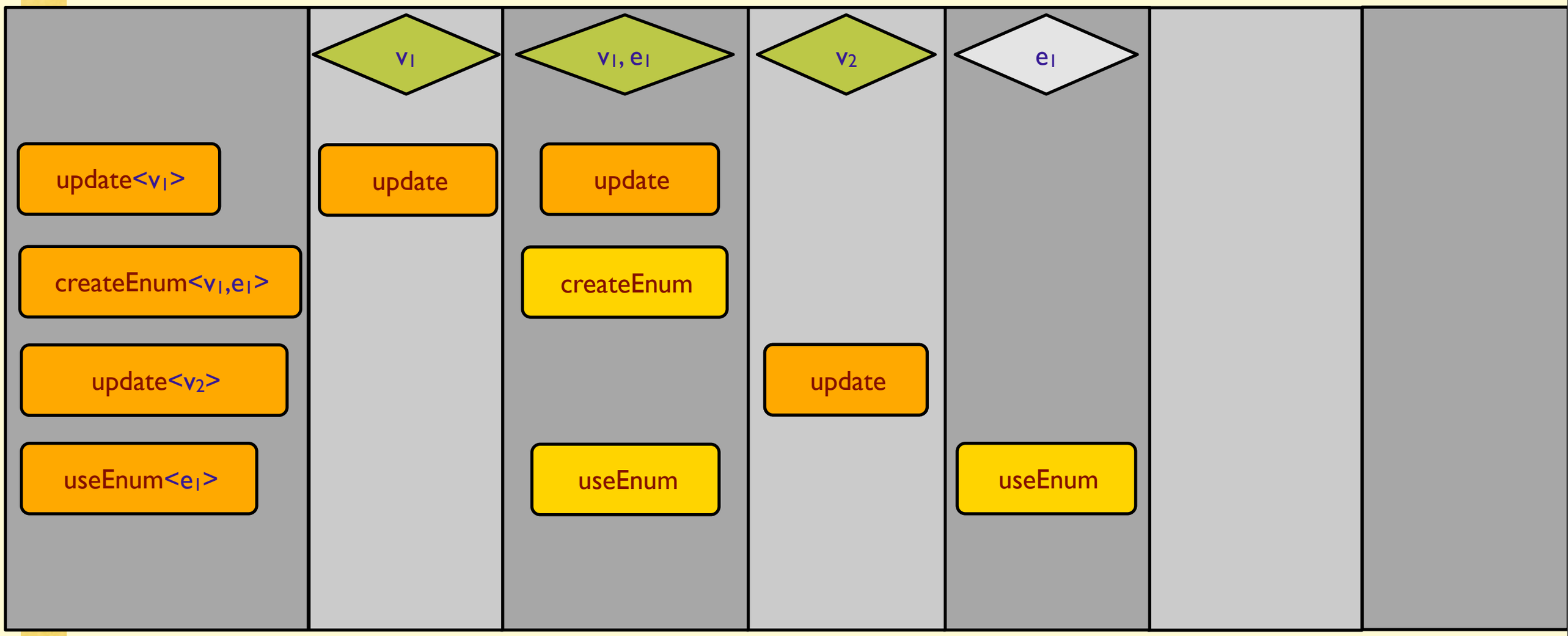

update
useEnum
update
createEnum
update
useEnum
s0
s1
s2
s3

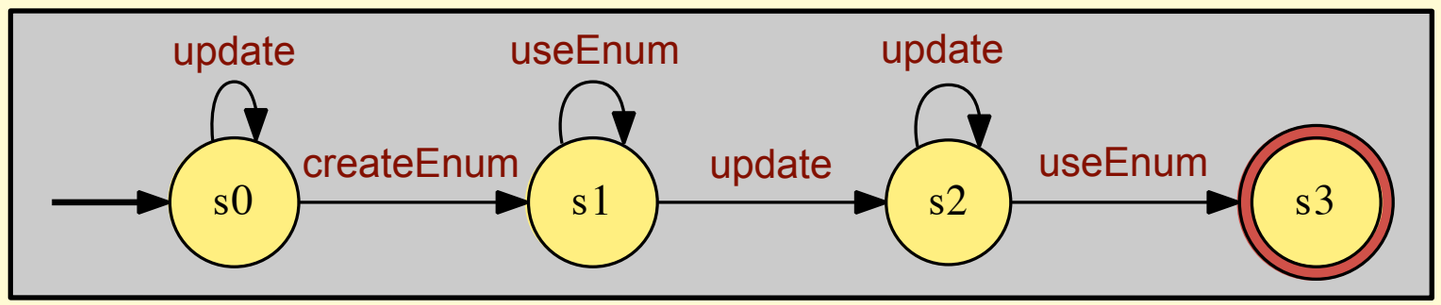
Example Run


v1
v1, e1
v2
update<v1>
update
update
createEnum<v1,e1>
createEnum
update<v2>
update

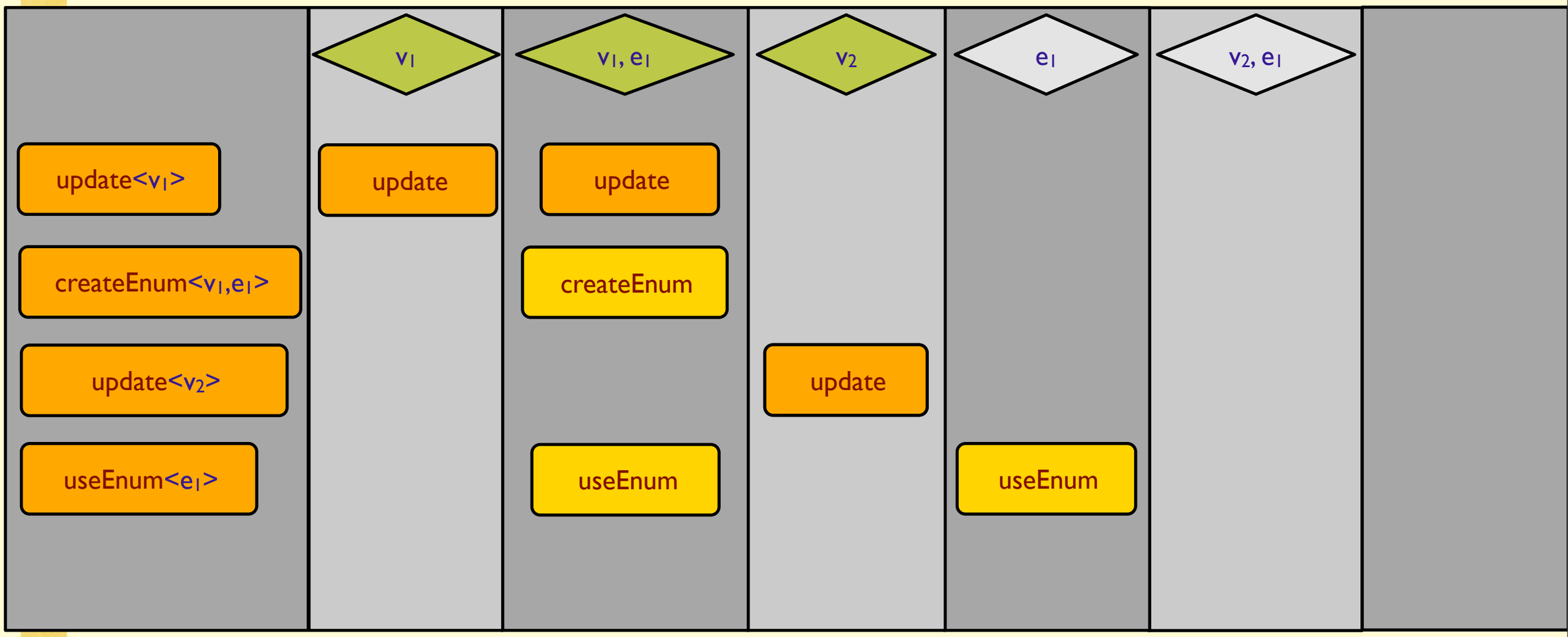

update
useEnum
update
createEnum
update
useEnum
s0
s1
s2
s3

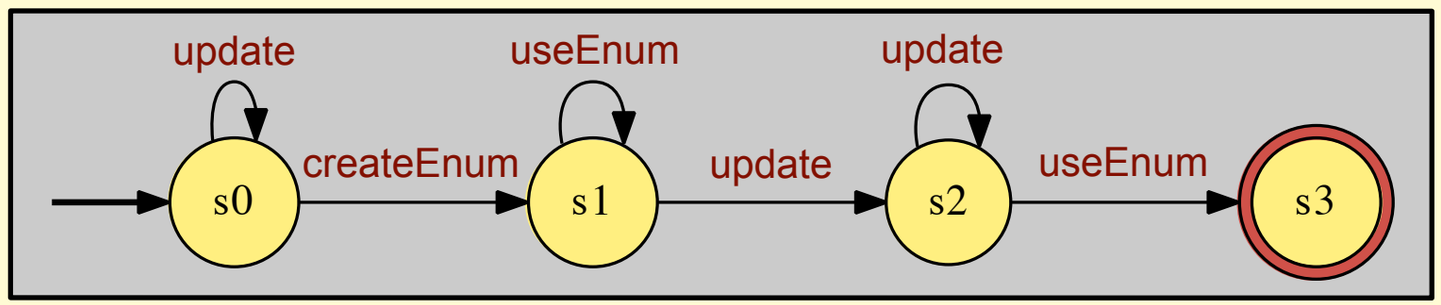
Example Run


v1
v1, e1
v2
e1
update<v1>
update
update
createEnum<v1,e1>
createEnum
update<v2>
update
useEnum<e1>
useEnum
useEnum

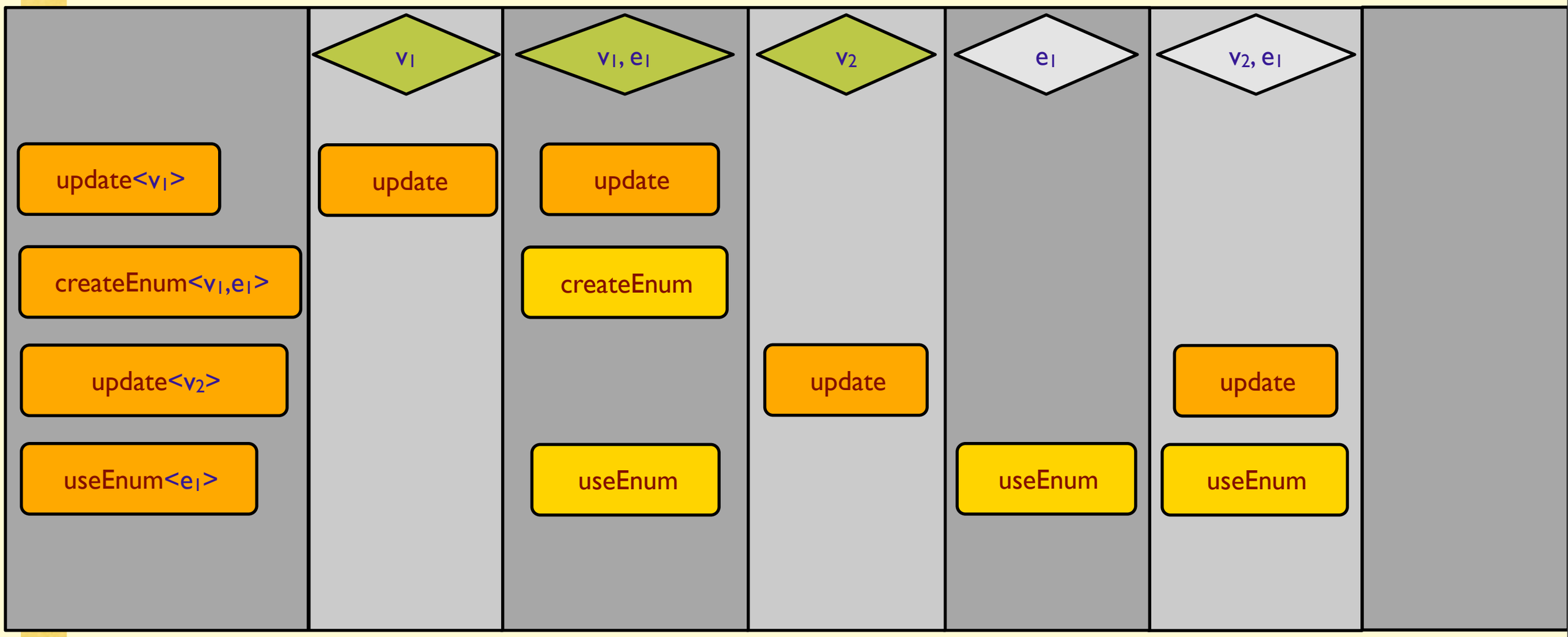

update
useEnum
update
createEnum
update
useEnum
s0
s1
s2
s3

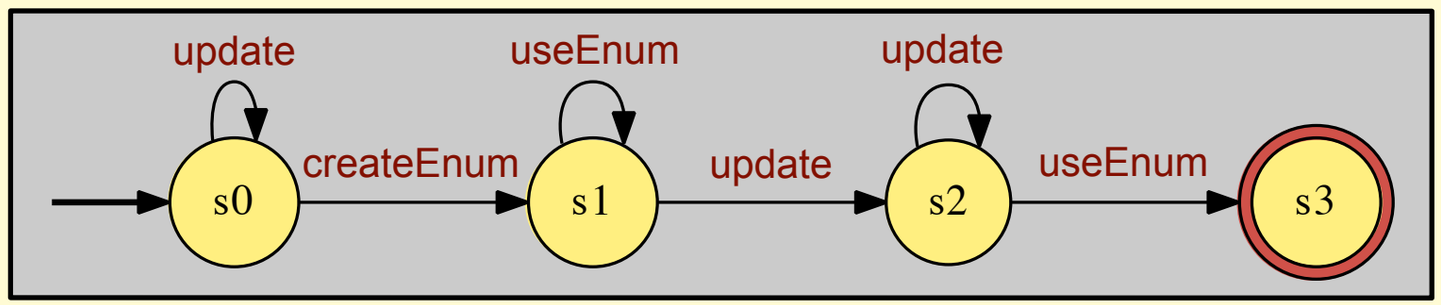
Example Run


v1
v1, e1
v2
e1
v2, e1
update<v1>
update
update
createEnum<v1,e1>
createEnum
update<v2>
update
useEnum<e1>
useEnum
useEnum

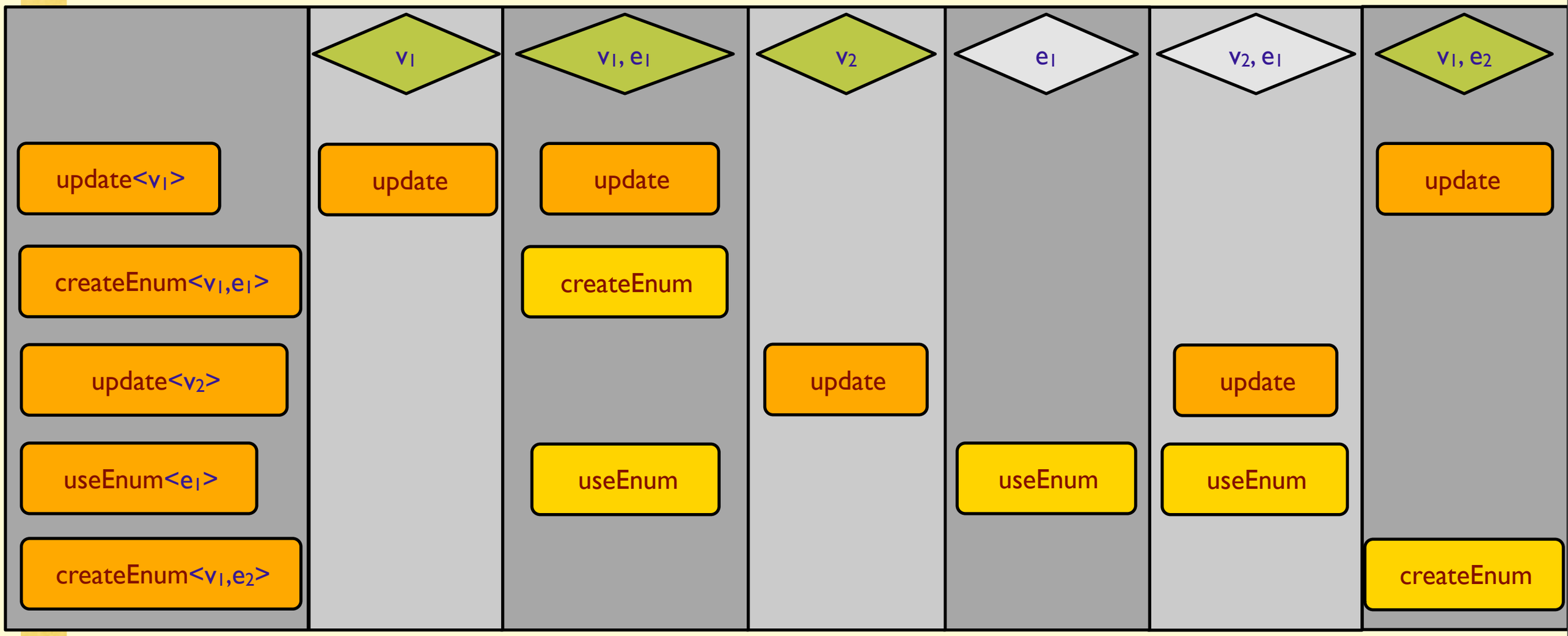

update
useEnum
update
createEnum
update
useEnum
s0
s1
s2
s3

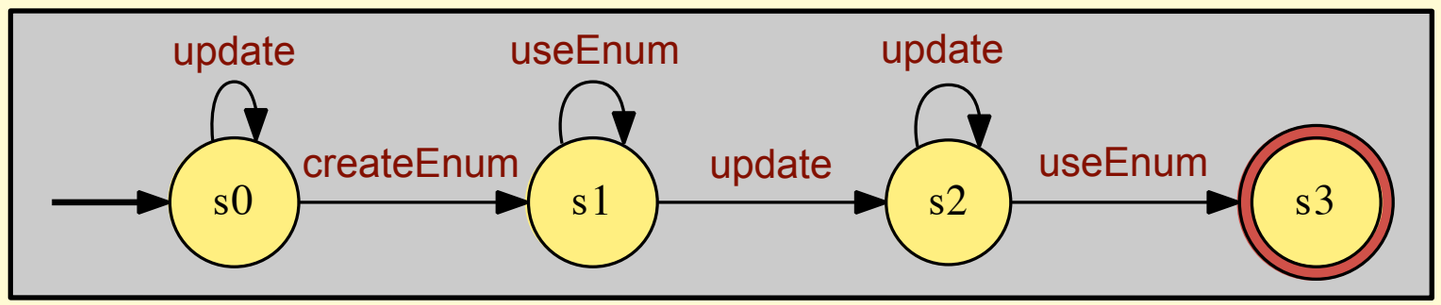
Example Run


v1
v1, e1
v2
e1
v2, e1
update<v1>
update
update
createEnum<v1,e1>
createEnum
update<v2>
update
update
useEnum<e1>
useEnum
useEnum
useEnum

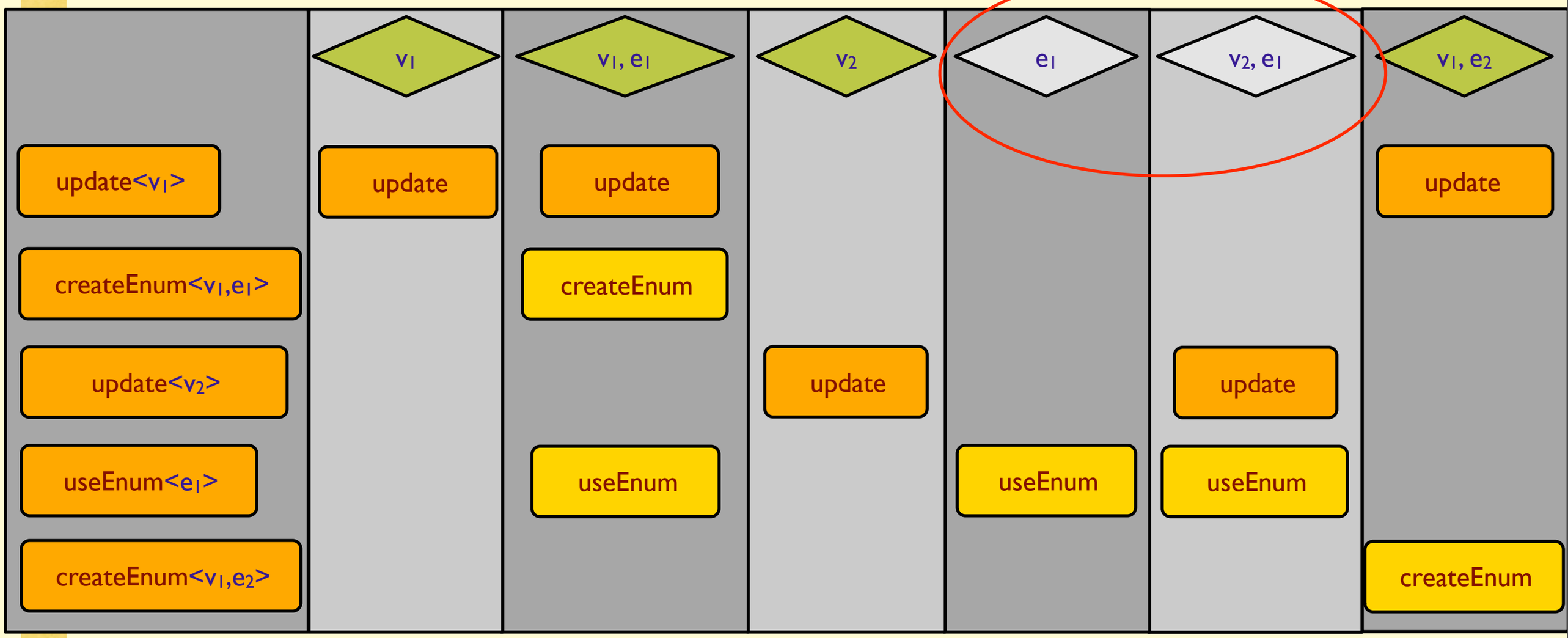

update
useEnum
update
createEnum
update
useEnum
s0
s1
s2
s3

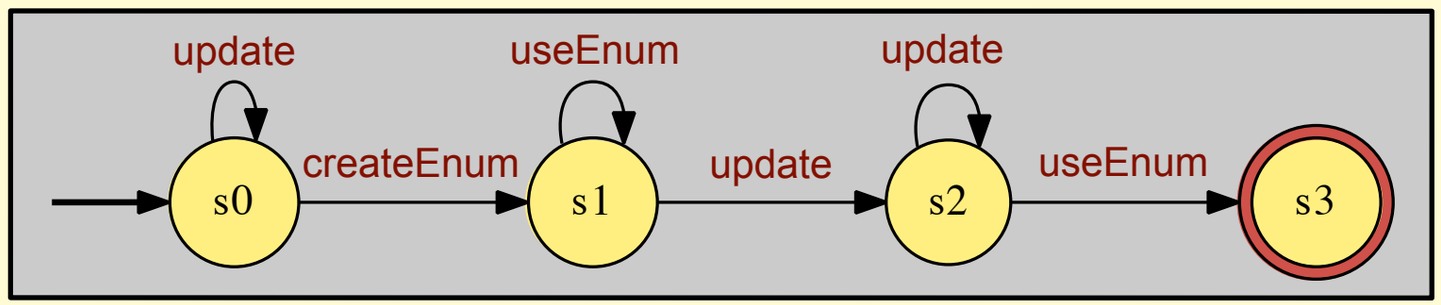
Example Run


v1
v1, e1
v2
e1
v2, e1
v1, e2
update<v1>
update
update
update
createEnum<v1,e1>
createEnum
update<v2>
update
update
useEnum<e1>
useEnum
useEnum
useEnum
createEnum<v1,e2>
createEnum


update
useEnum
update
createEnum
update
useEnum
s0
s1
s2
s3

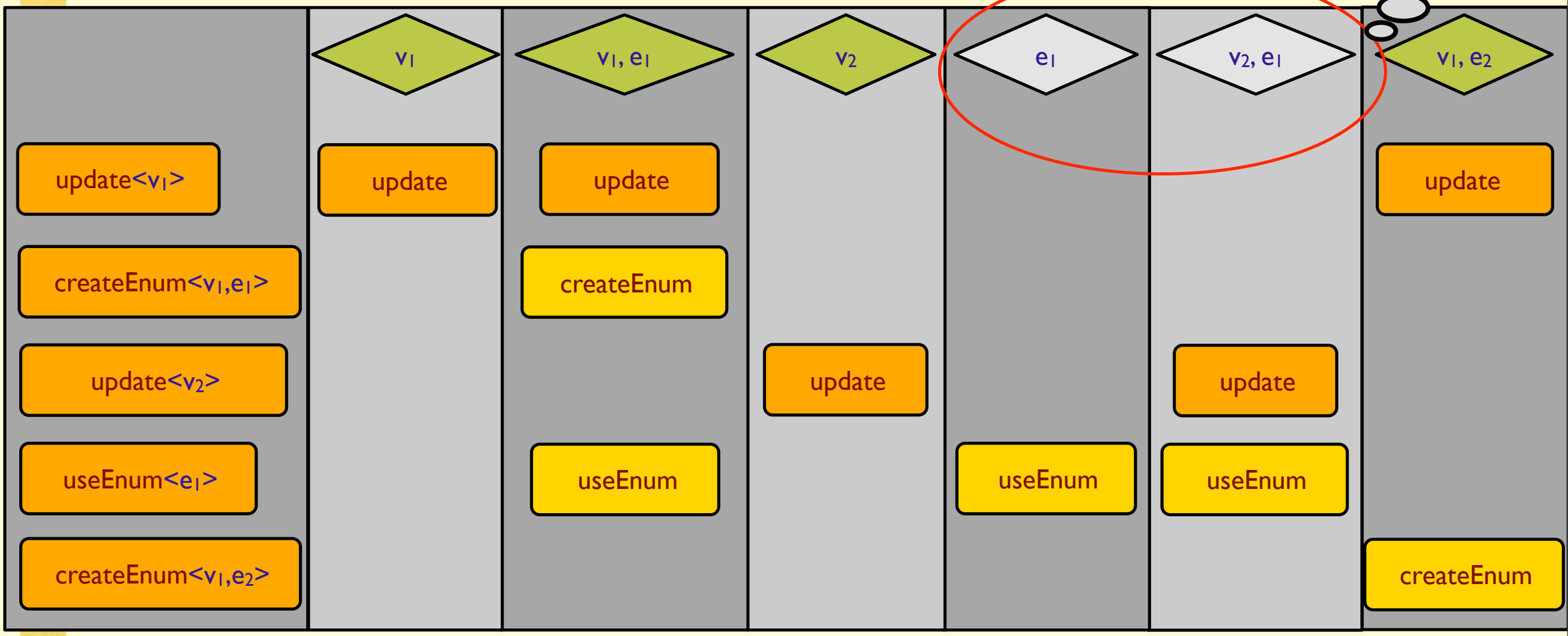
Example Run

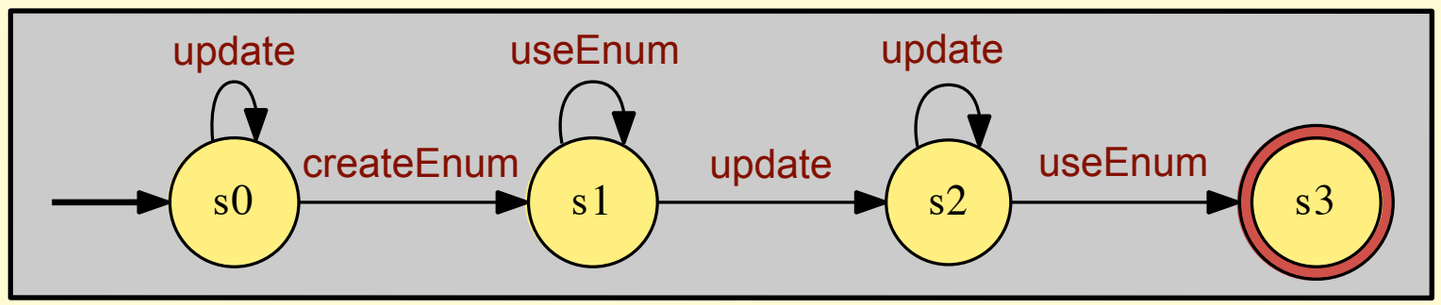

v1
v1, e1
v2
e1
v2, e1
v1, e2
update<v1>
update
update
update
createEnum<v1,e1>
createEnum
update<v2>
update
update
useEnum<e1>
useEnum
useEnum
useEnum
createEnum<v1,e2>
createEnum


update
useEnum
update
createEnum
update
useEnum
s0
s1
s2
s3

Example Run

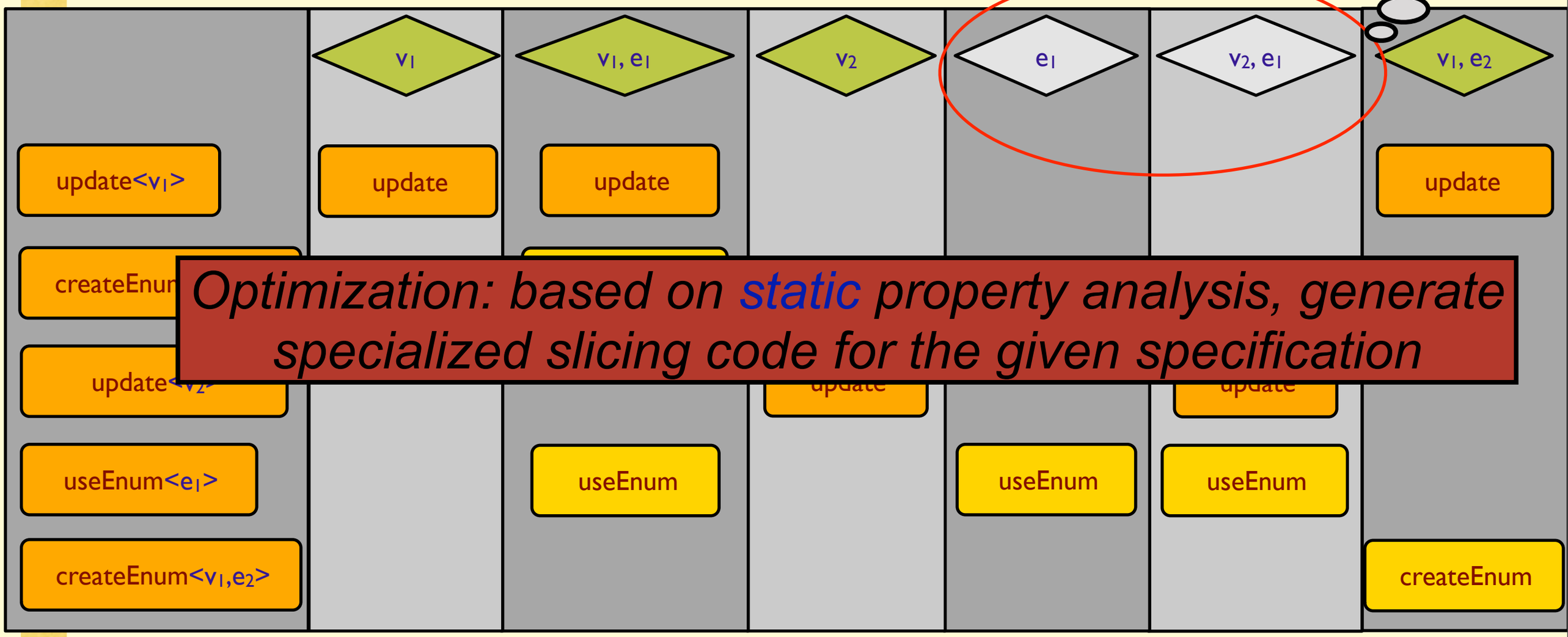
Real programs generate a LOT of worthless monitors!



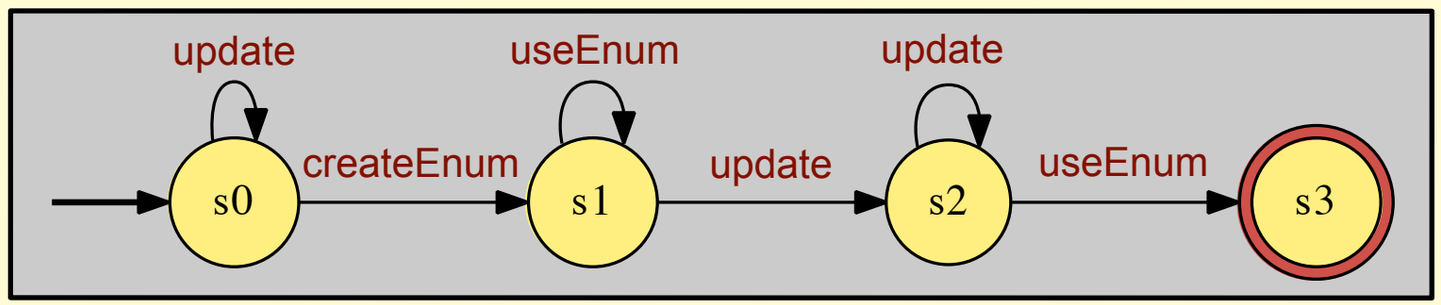


Example Run

Real programs generate a LOT of worthless monitors!

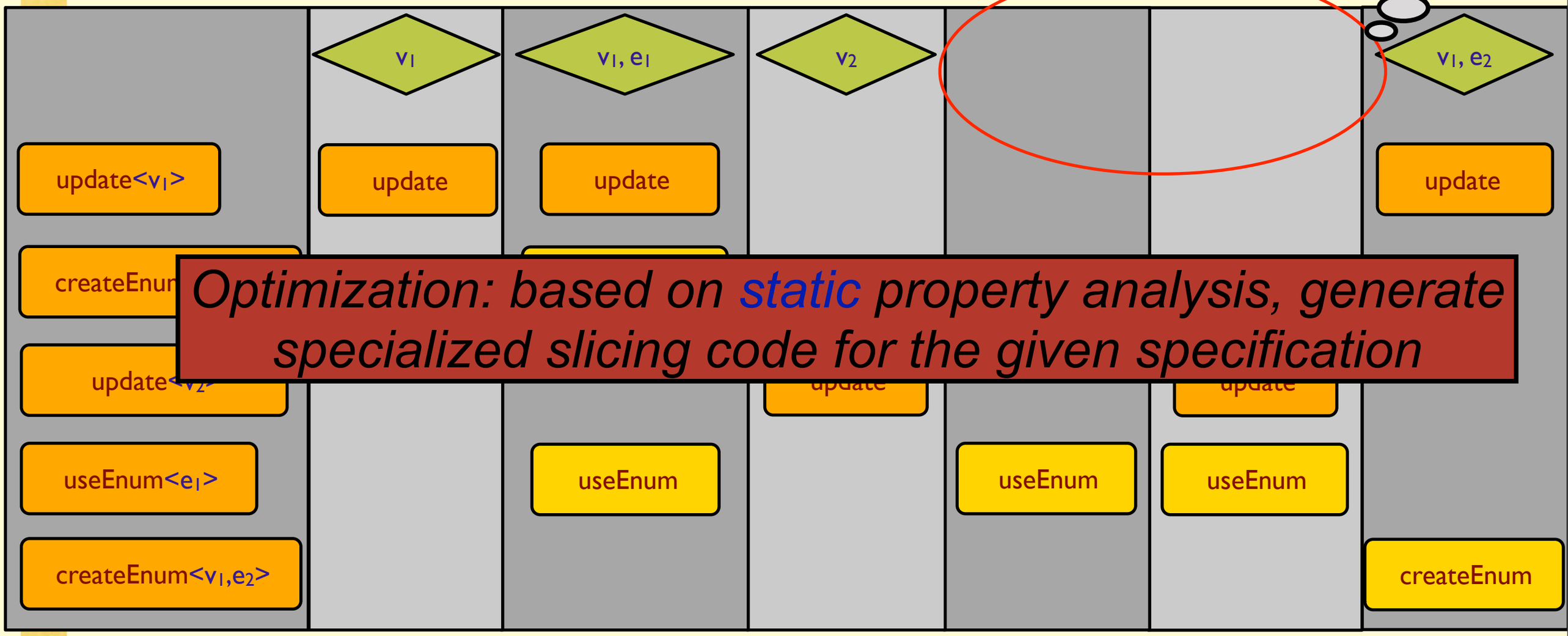


Optimization: based on static property analysis, generate specialized slicing code for the given specification

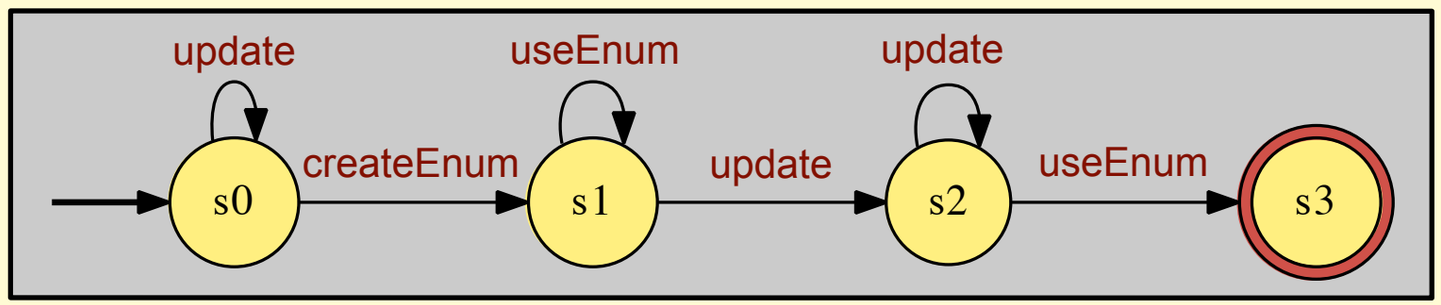


Example Run

Real programs generate a LOT of worthless monitors!



Optimization: based on static property analysis, generate specialized slicing code for the given specification



Overview

- Introduction
- Parametric Monitoring
- Enable Sets Based Optimization
- Results
- Conclusions and Future Work

Enable Set Optimization

- We need to remove worthless monitors!
- Two directions for optimization:
  - Program analysis [AOSD '09]
  - Property analysis [Here]
- Enable Sets are a way to abstract knowledge of the property
  - Solves the problem of useless monitor generation
  - Improves performance radically in some cases without slowdowns in any cases

Enable Sets

- For each event which parameters must be instantiated before it may occur
- Computed by the monitor generation code for each formalism
  - Monitor generation code does not know about parameters
  - Compute in terms of events, MOP infers parameters

For trace t, set$(t) = \{e \mid e \in t\}$, enable$(e, L) = \{\text{set}(t_1) \mid t_1 e t_2 \in L\}$

Enable Sets

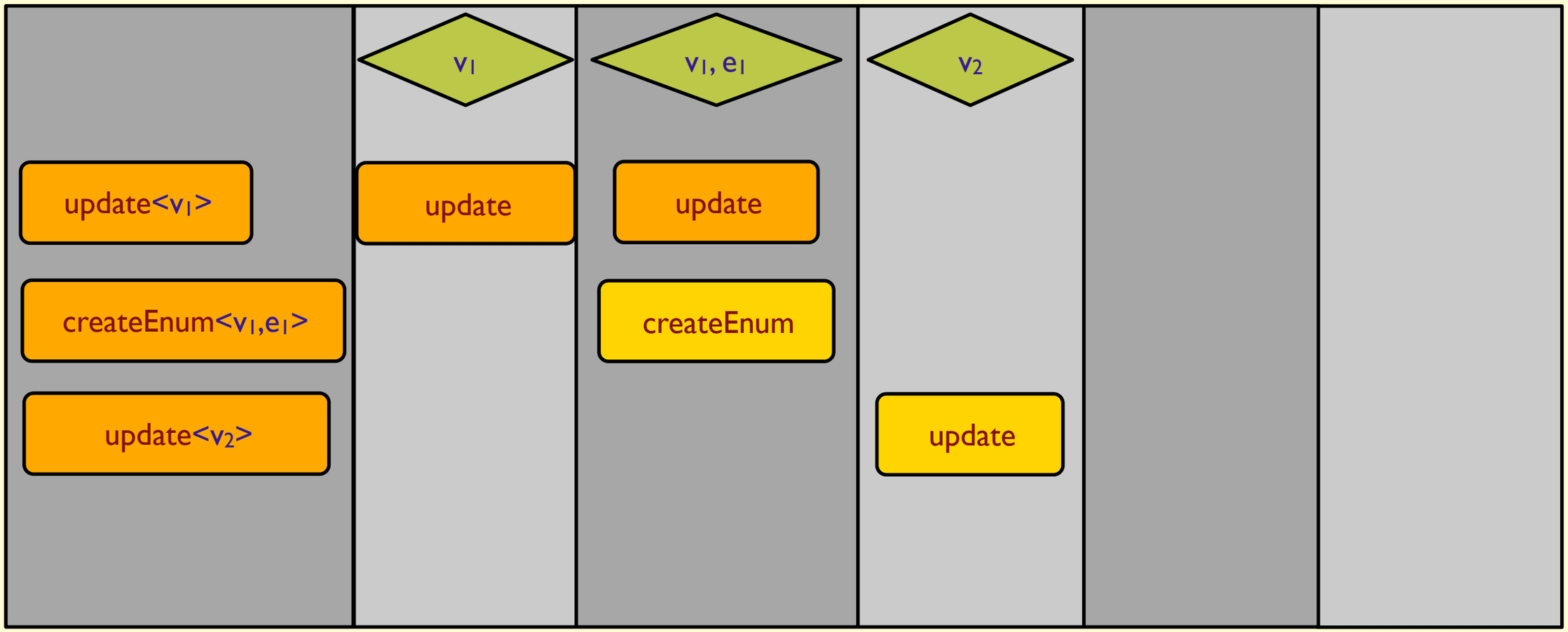
- For each event which parameters must be instantiated before it may occur
- Computed by the monitor generation code for each formalism
  - Monitor generation code does not know about parameters
  - Compute in terms of events, MOP infers parameters

For trace t, set(t) $= \{e \mid e \in t\}$, enable(e, L) $= \{\text{set}(t_1) \mid t_1 e t_2 \in L\}$

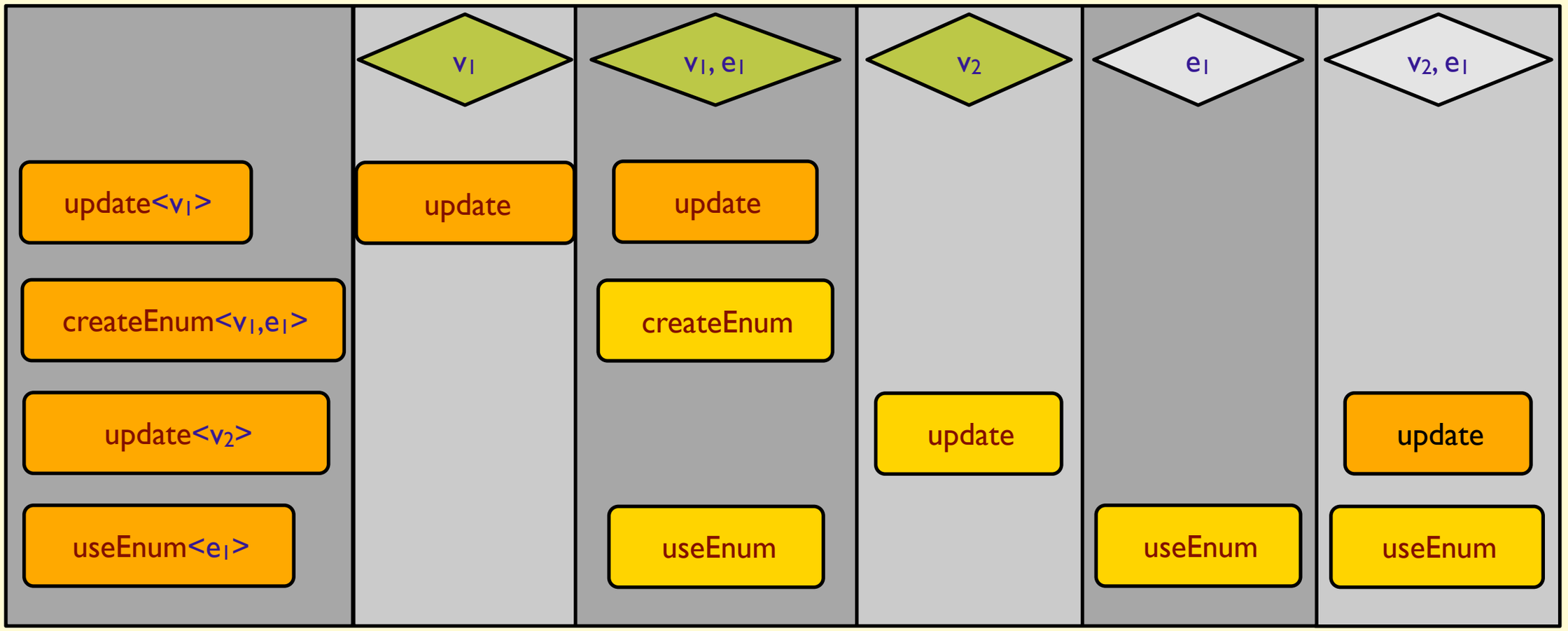
Possible traces - createEnum useEnum update useEnum
createEnum update useEnum

useEnum - {{createEnum},{update, createEnum},
{update, createEnum, useEnum}}

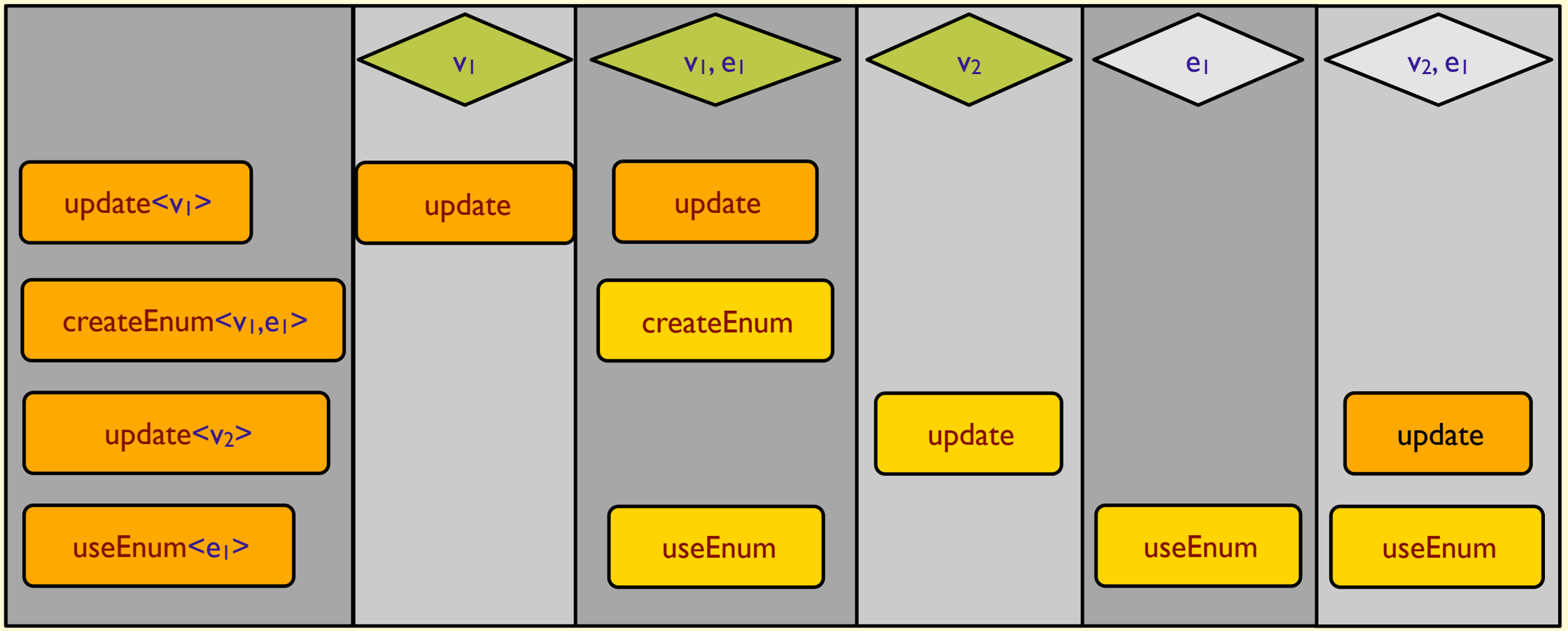
Optimizing with Enable Sets



Optimizing with Enable Sets

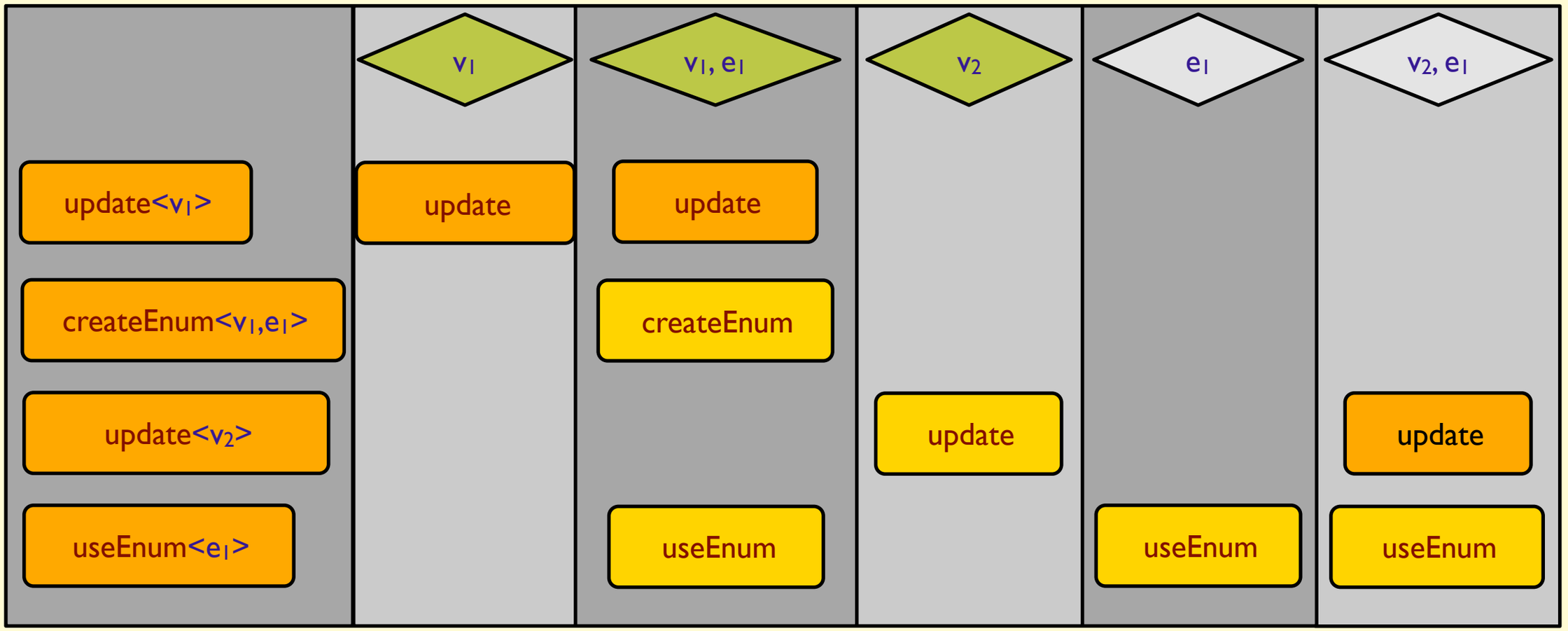


Optimizing with Enable Sets



useEnum - {{createEnum<v,e>},{update<v>, createEnum<v,e>},
{update<v>, createEnum<v,e>, useEnum<e>}}

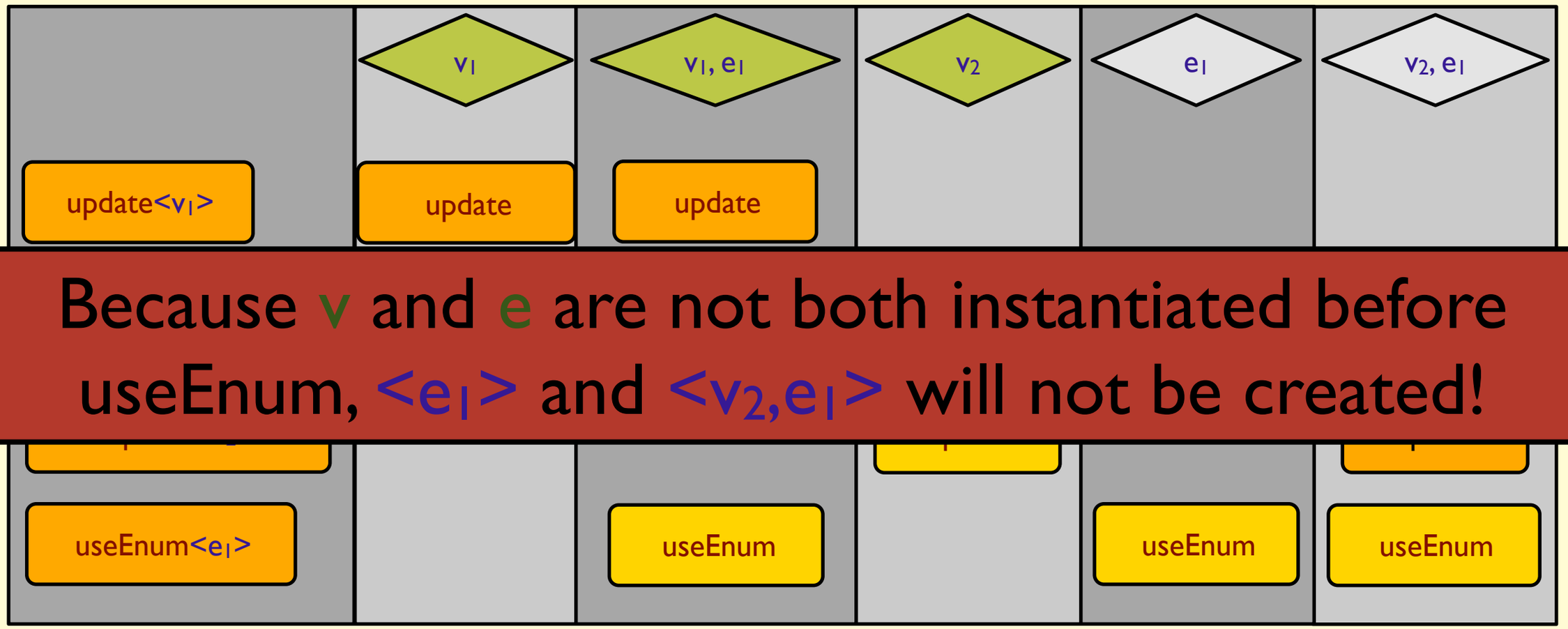
Optimizing with Enable Sets



useEnum - {{createEnum<v,e>},{update<v>, createEnum<v,e>},
{update<v>, createEnum<v,e>, useEnum<e>}}

useEnum - {{v,e}}

Optimizing with Enable Sets

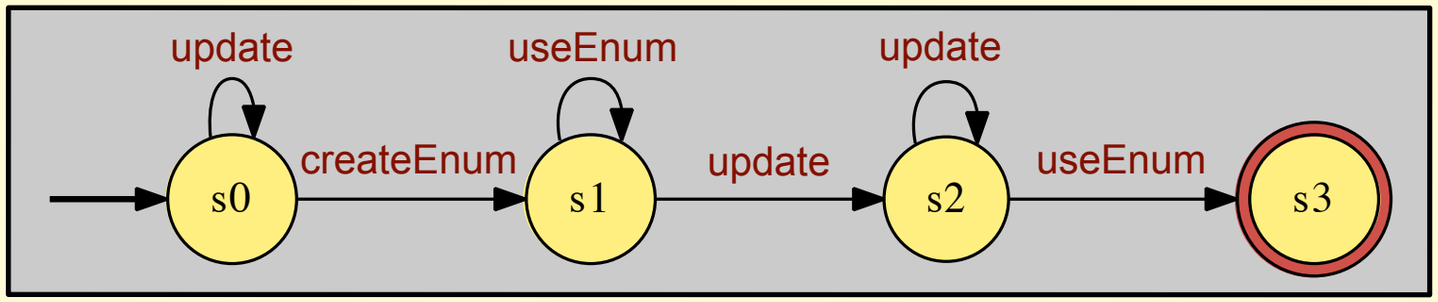


Because v and e are not both instantiated before useEnum, <e_1> and <v_2,e_1> will not be created!

useEnum - {{createEnum<v,e>},{update<v>, createEnum<v,e>},
{update<v>, createEnum<v,e>, useEnum<e>}}

useEnum - {{v,e}}

Computing Match Enable Sets for Finite State Machines



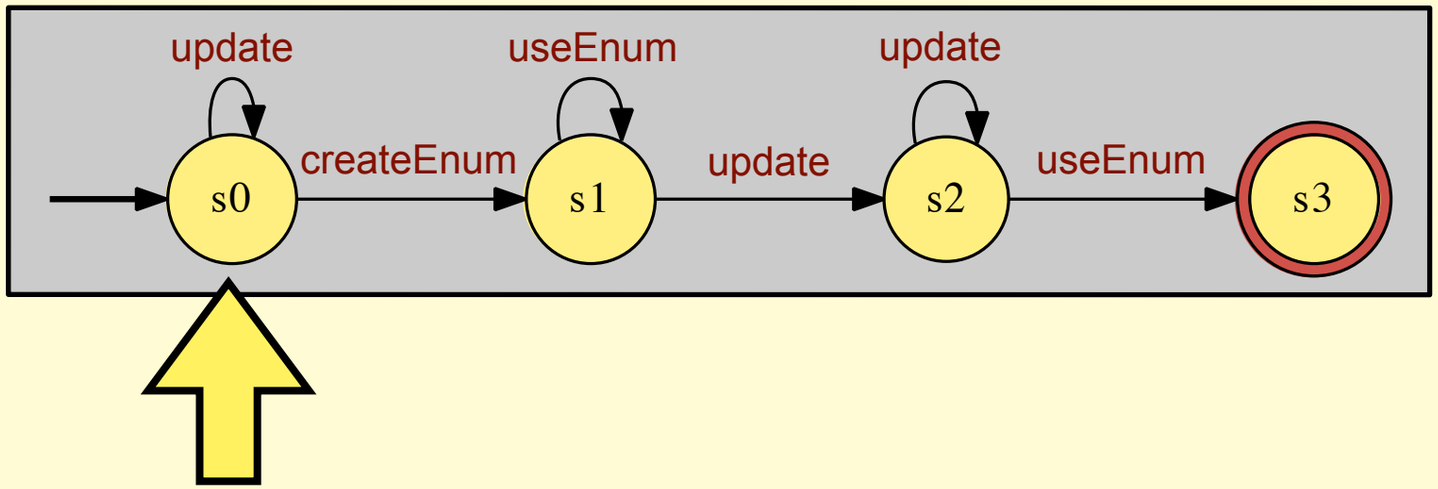
current set - {}

update - {}

createEnum - {}

useEnum - {}

Computing Match Enable Sets for Finite State Machines



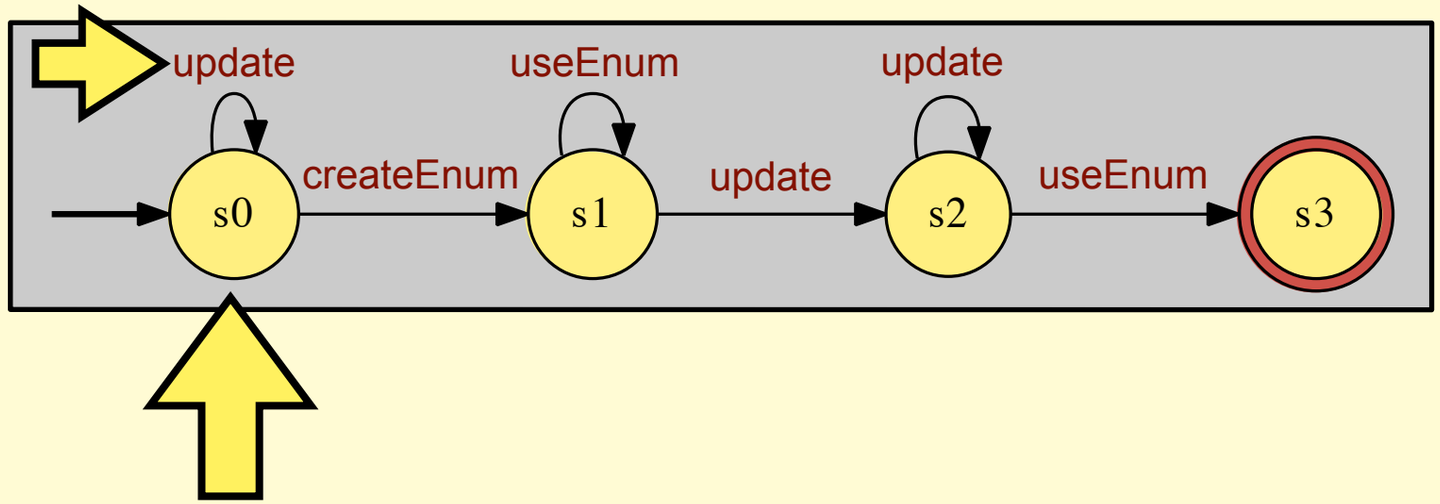
current set - {}

update - {}

createEnum - {}

useEnum - {}

Computing Match Enable Sets for Finite State Machines



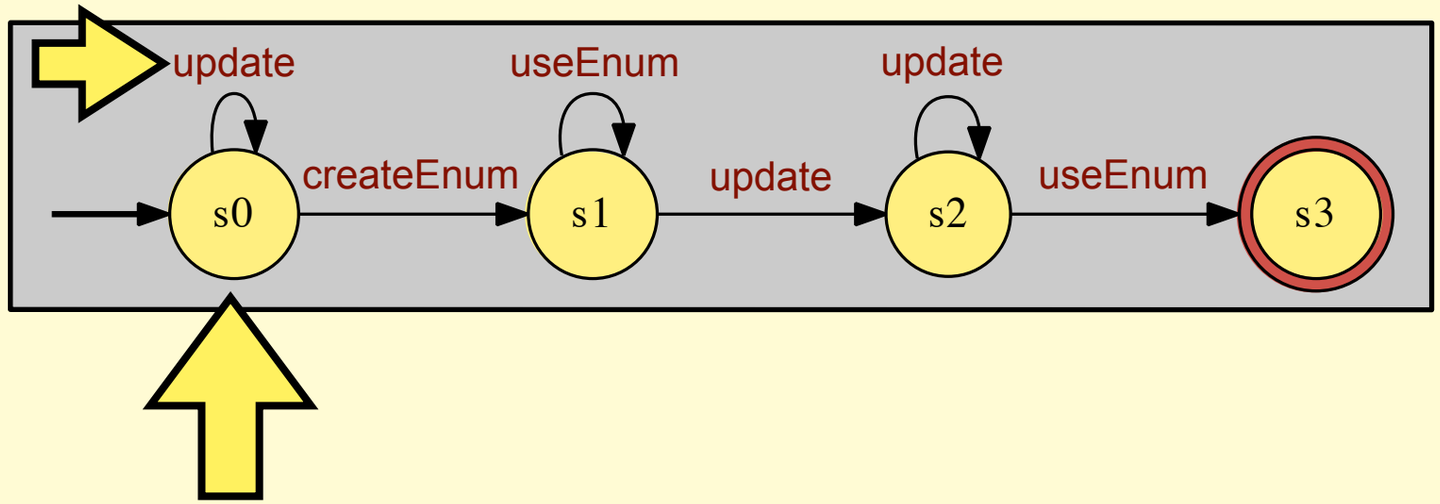
current set - {}

update - {}

createEnum - {}

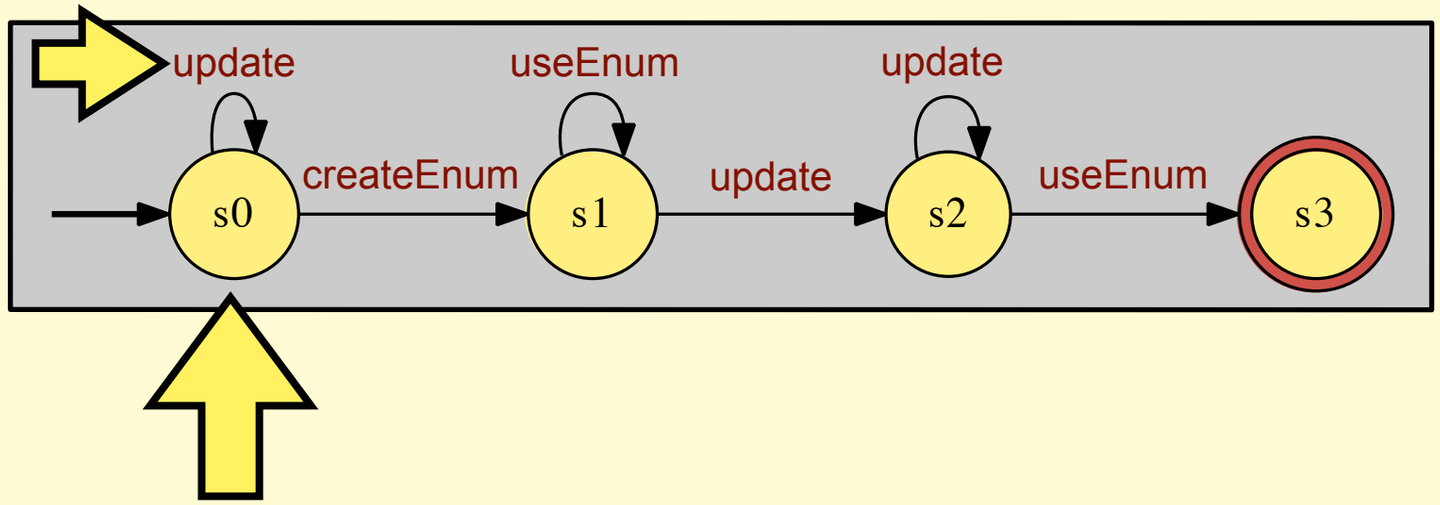
useEnum - {}

Computing Match Enable Sets for Finite State Machines



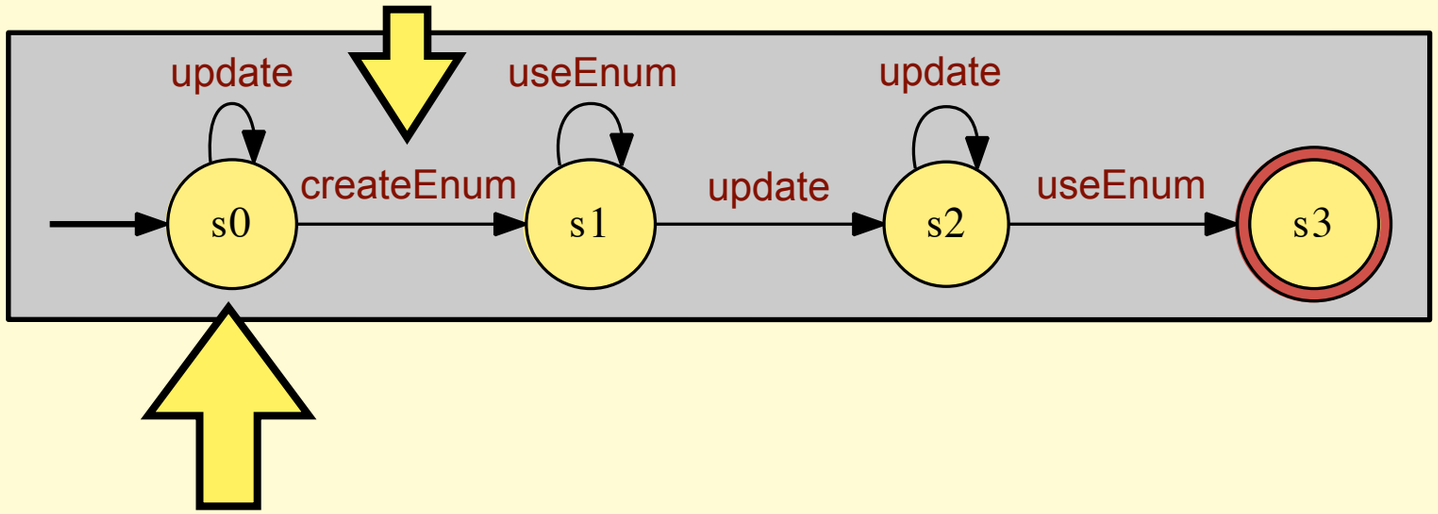
current set -	{}
update -	{{}}
createEnum -	{}
useEnum -	{}

Computing Match Enable Sets for Finite State Machines



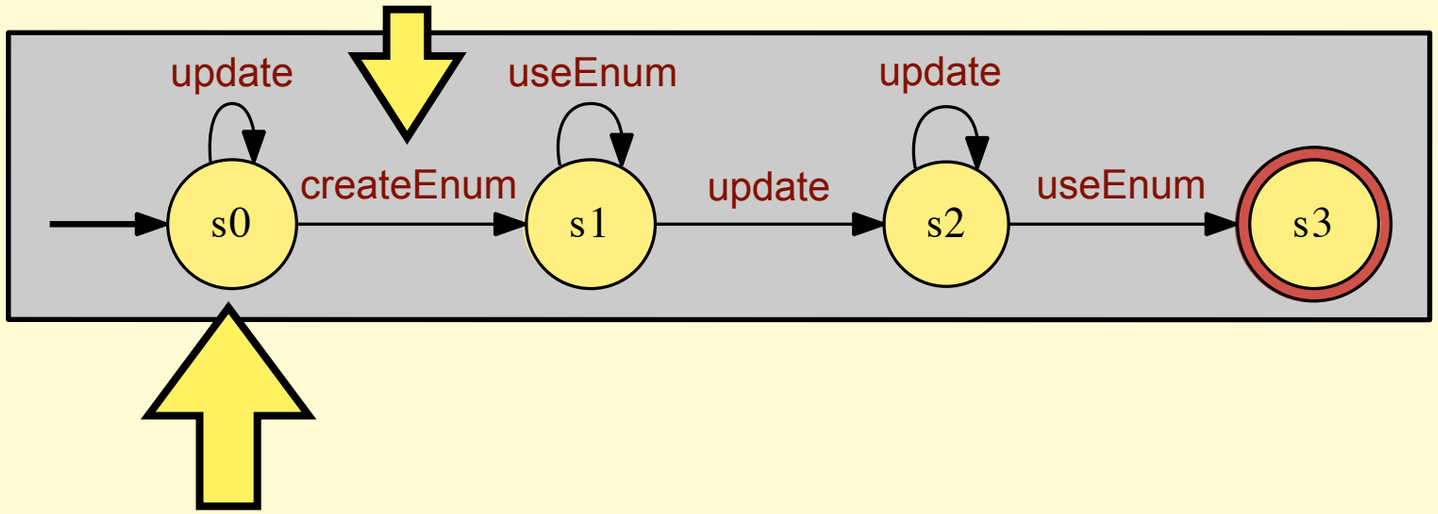
current set -	{update}
update -	{{}}
createEnum -	{}
useEnum -	{}

Computing Match Enable Sets for Finite State Machines



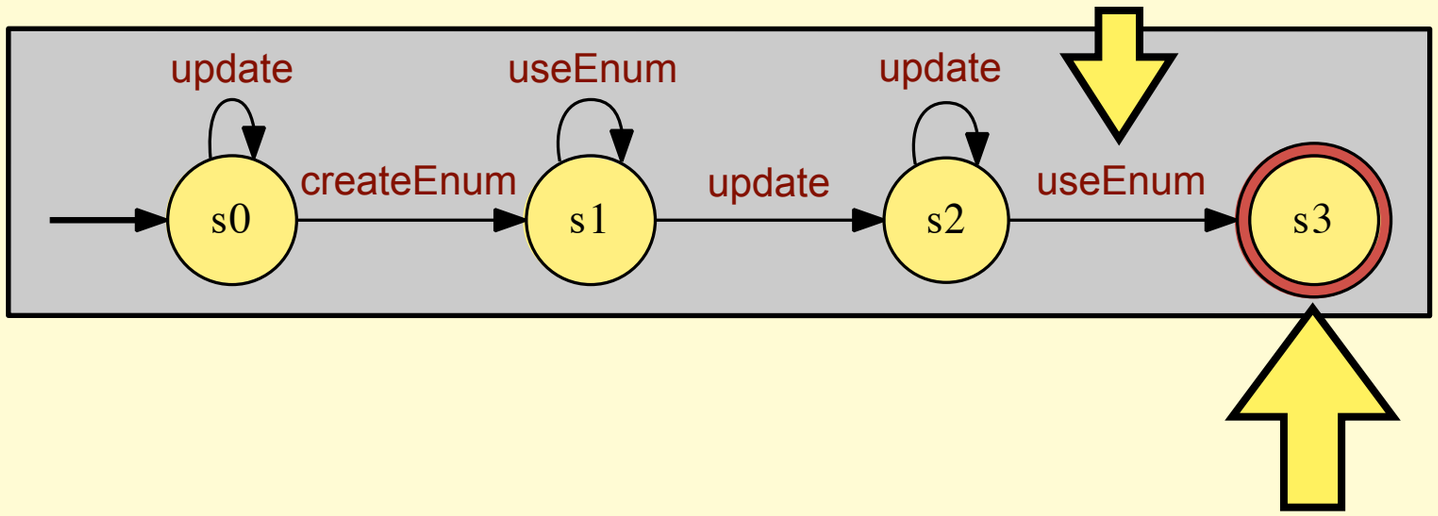
current set -	{update}
update -	{{}}
createEnum -	{}
useEnum -	{}

Computing Match Enable Sets for Finite State Machines



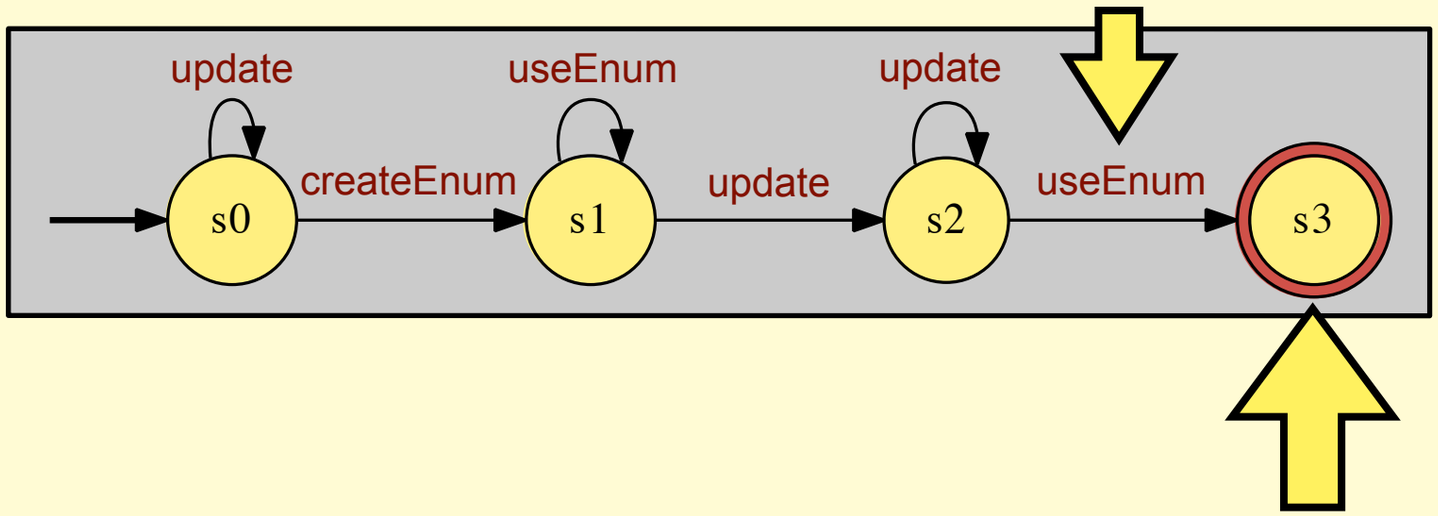
current set -	{update}
update -	{{}}
createEnum -	{{update}}
useEnum -	{}

Computing Match Enable Sets for Finite State Machines



current set -	{update, createEnum, useEnum}
update -	{{},{update, createEnum, useEnum}}
createEnum -	{{update}}
useEnum -	{{update, createEnum}, {update, createEnum, useEnum}}

Computing Match Enable Sets for Finite State Machines



current set -	{update, createEnum, useEnum}
update -	{{},{update, createEnum, useEnum}}
createEnum -	{{update}}
useEnum -	{{update, createEnum}, {update, createEnum, useEnum}}

Repeated for each path that results in a different set

Computing Enable Sets for Context Free Grammars

- We also can compute enable sets for the matching context free grammars
- They can be computed as the least fixed point of these equations:

$$
\begin{aligned}
\mathsf{GenSets}(\epsilon) &= \{\emptyset\} \\
\mathsf{GenSets}(e) &= \{\{e\}\} \\
\mathsf{GenSets}(A) &= \bigcup_{A \to \gamma} \mathsf{GenSets}(\gamma) \\
\mathsf{GenSets}(\gamma_1\gamma_2) &= \{T \cup U \mid T \in \mathsf{GenSets}(\gamma_1), U \in \mathsf{GenSets}(\gamma_2)\} \\
\mathsf{PreSymSets}(S) &= \{T \cup U \mid A \to \gamma_1 S \gamma_2, T \in \mathsf{PreSymSets}(A), U \in \mathsf{GenSets}(\gamma_1)\} \\
\mathsf{enable}(e,\ match) &= \mathsf{PreSymSets}(e)
\end{aligned}
$$

Overview

- Motivation
- Parametric Monitoring
- Enable Sets Based Optimization
- Results
- Conclusions and Future Work

DaCapo - Benchmark Suite Used for Evaluation

- A popular benchmark suite for Java
  - www.dacapobench.org

- Benchmark forms of non-trivial programs
  - Bloat: A bytecode optimizer
  - Antlr: A parser generator
  - Eclipse: A popular Java IDE
  - Jython: Python for the Java Virtual Machine
  - ...

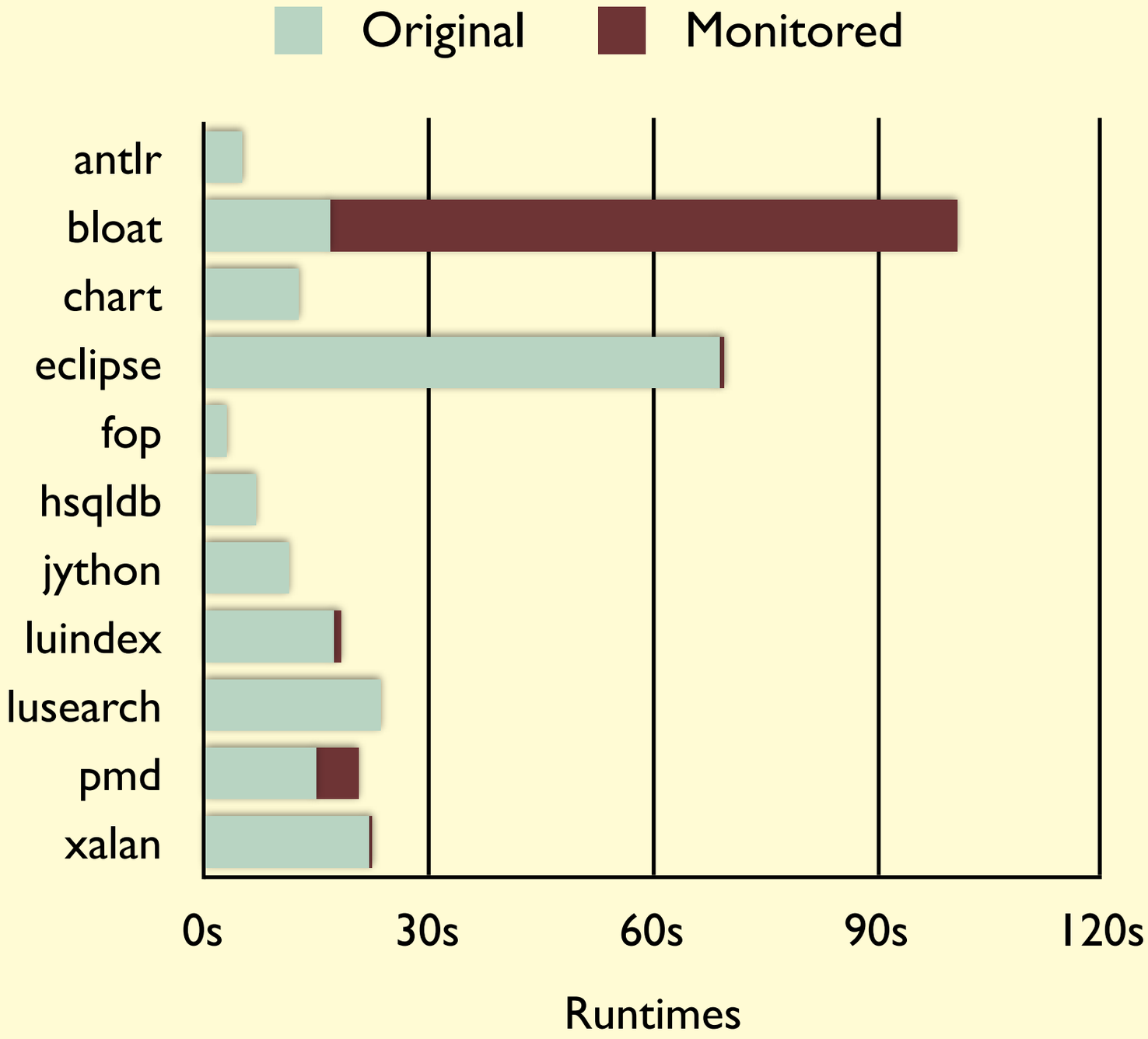
Properties Used for Evaluation

- UnsafeMapIterator
  - A Map *m* must not be updated while iterating over Collection *c* backed by *m*
  - createColl<m,c> updateMap<m>* createIter<c,i> useIter<i>* updateMap<m>+ useIter<i>
- UnsafeSyncCollection
  - A synchronized Collection *c* must not be accessed asynchronously
  - (sync<c> asyncCreateIter<c,i>)
    | (sync<c> syncCreateIter<c,i> accessIter<i>)

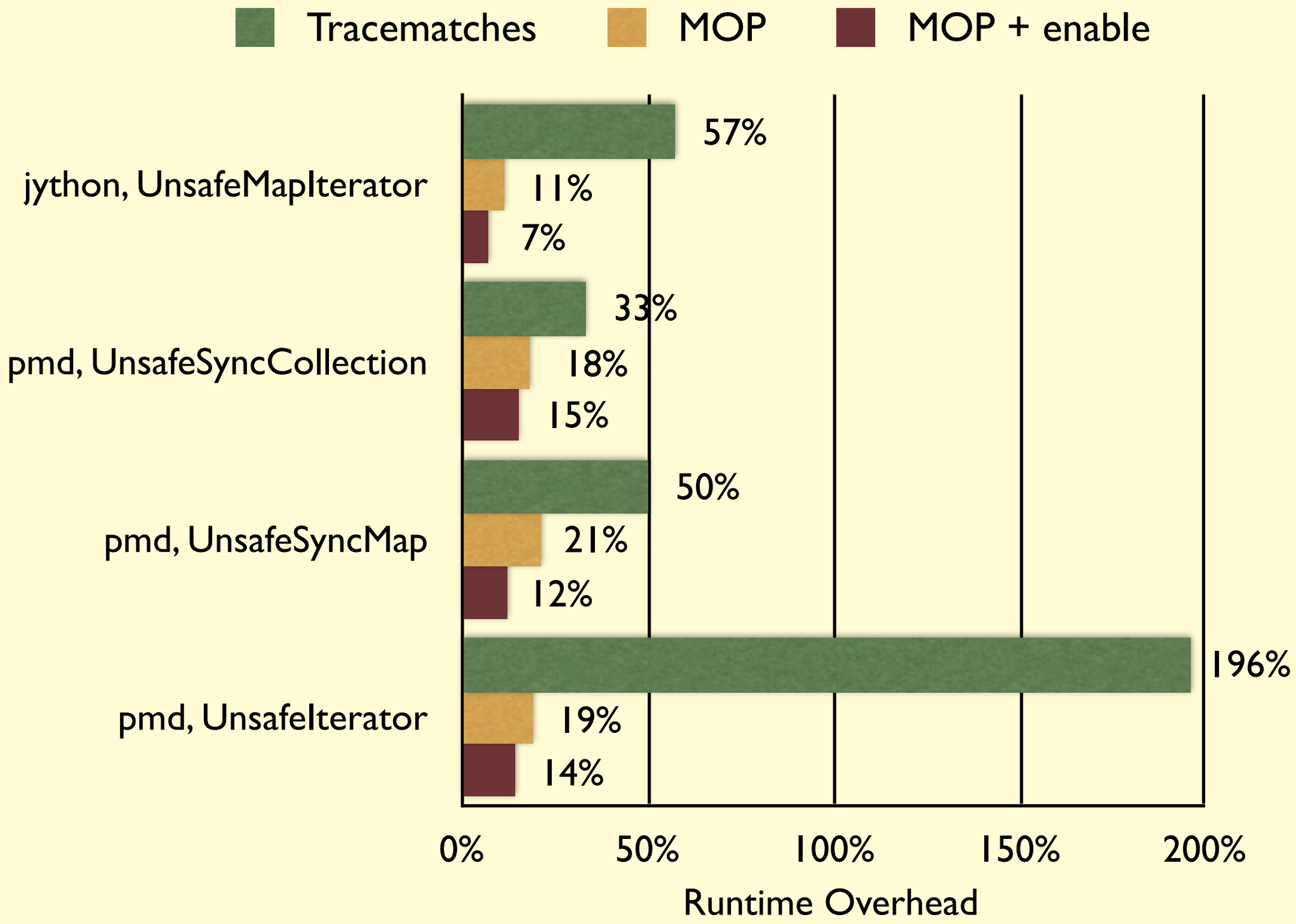
More Properties...

- UnsafeSyncMap
  - A Collection *c* created from a synchronized Map *m* must not be accessed in asynchronously
  - sync<m> createSet<m,c>
    (asyncCreateIter<c,i>
    |(syncCreateIter<c,i> accessIter<i>))
- SafeFile
  - A file opened in a given method must be closed in the same method
  - S -> S begin<t> S end<t> | S open<f> A close<f> | ε
    A -> A begin<t> A end<t> | ε

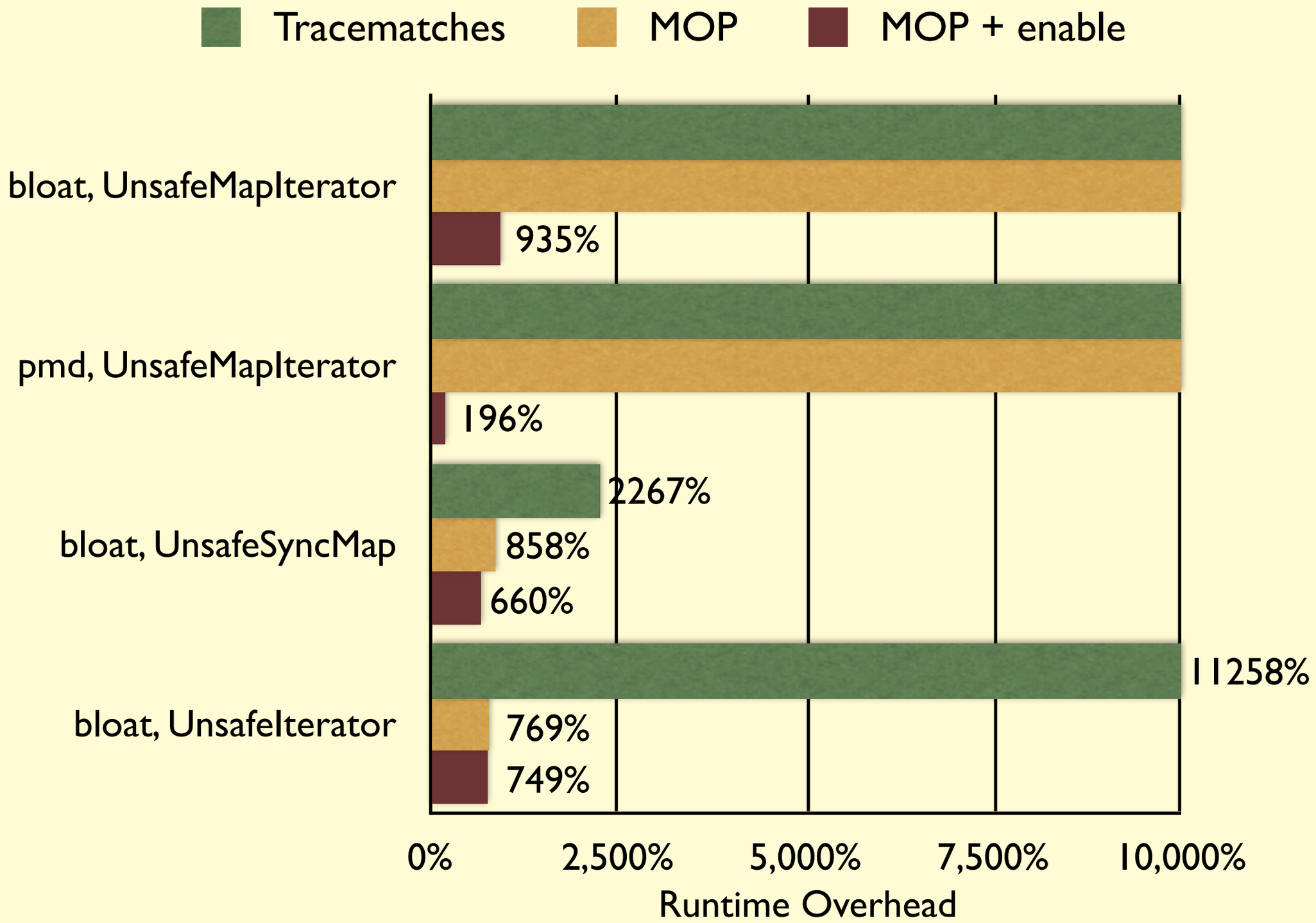
Original and Monitored Program Runtimes



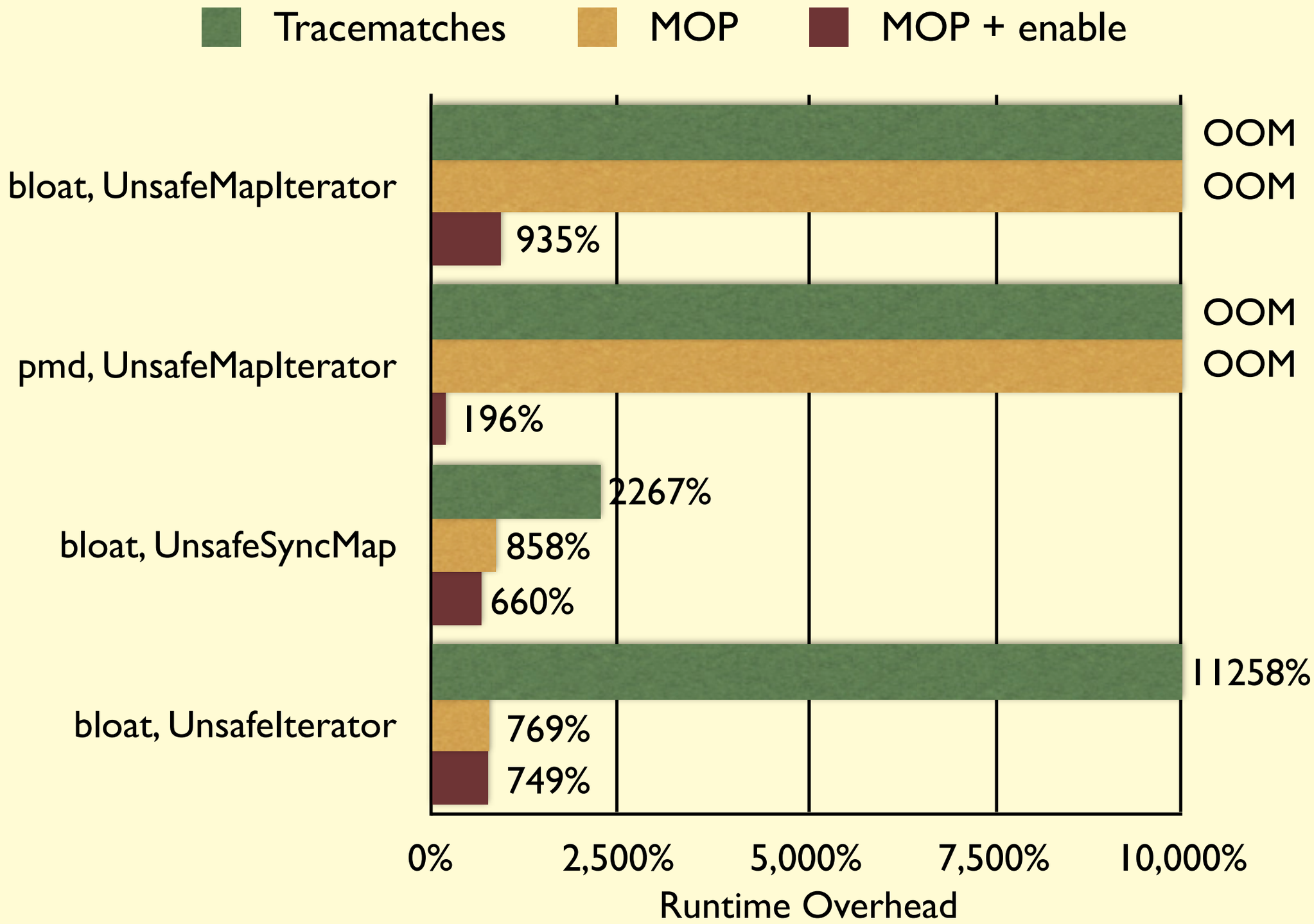
Non-Trivial Overheads



The Worst Overheads



The Worst Overheads



Conclusions and Future Work

- Parametric properties are useful and can be feasibly monitored
- Parametric trace slicing
  - Generic with respect to property formalism
- Enable set optimization makes trace slicing efficient
- Evaluation
  - Low runtime overhead
- Extend idea of Enable Sets to monitor termination