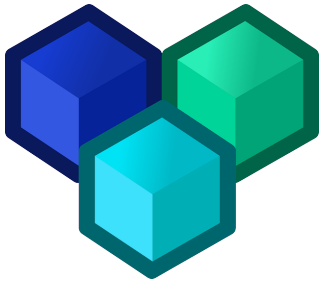
Adaptive Online First-Order Monitoring

Joshua Schneider David Basin Frederik Brix
Srđan Krstić Dmitriy Traytel

Department of Computer Science

ETH zürich

Verification



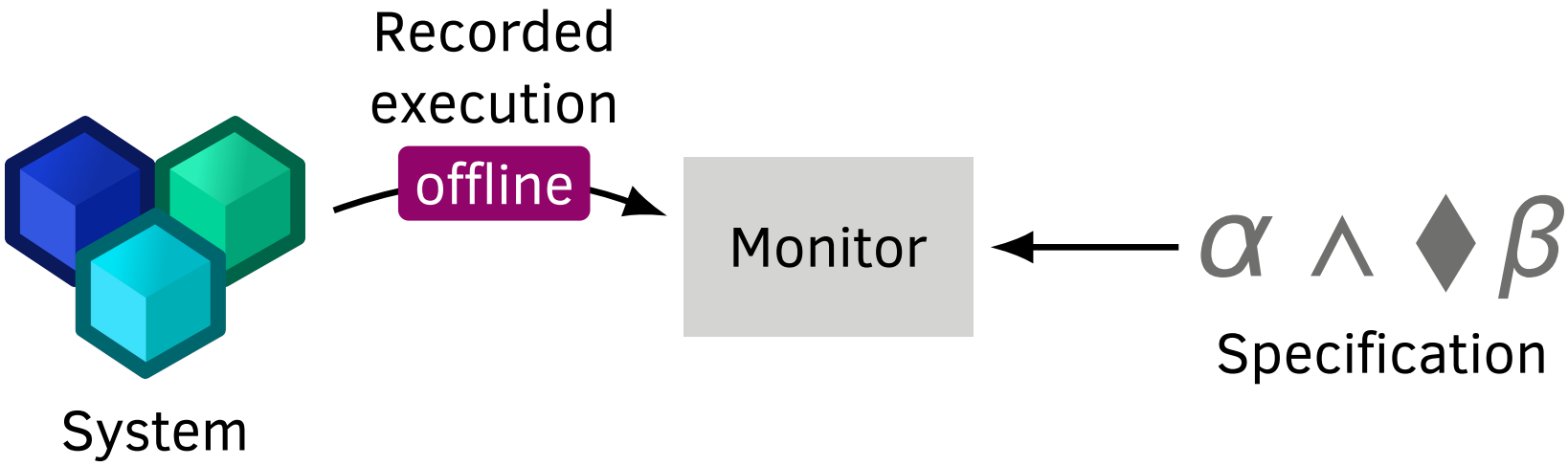
System

?

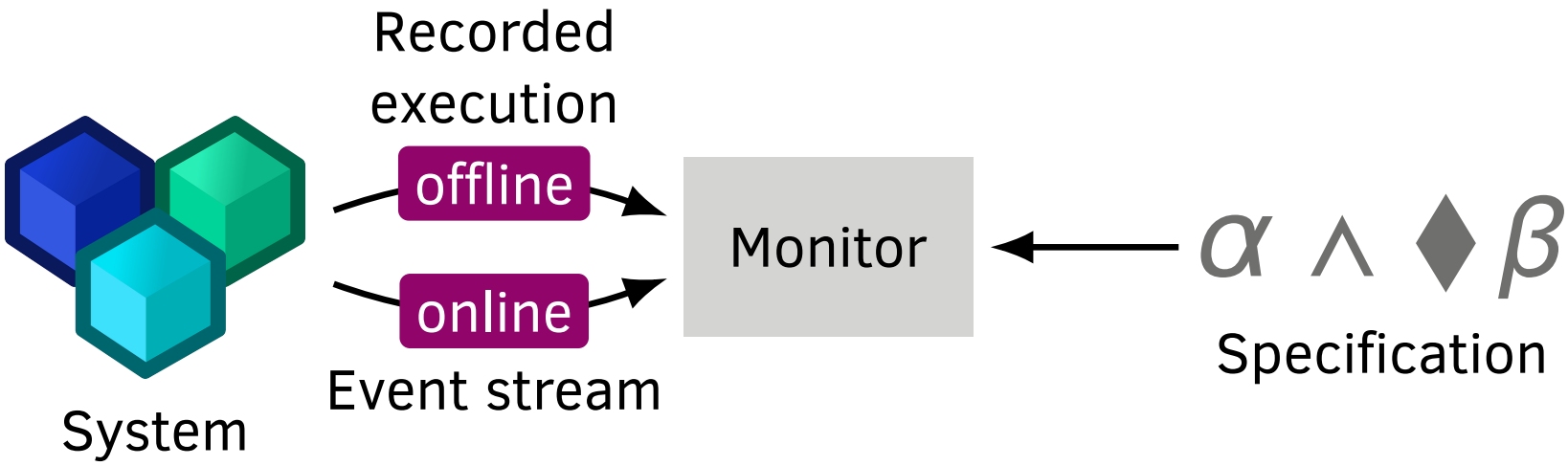
$\alpha \wedge \blacklozenge \beta$

Specification

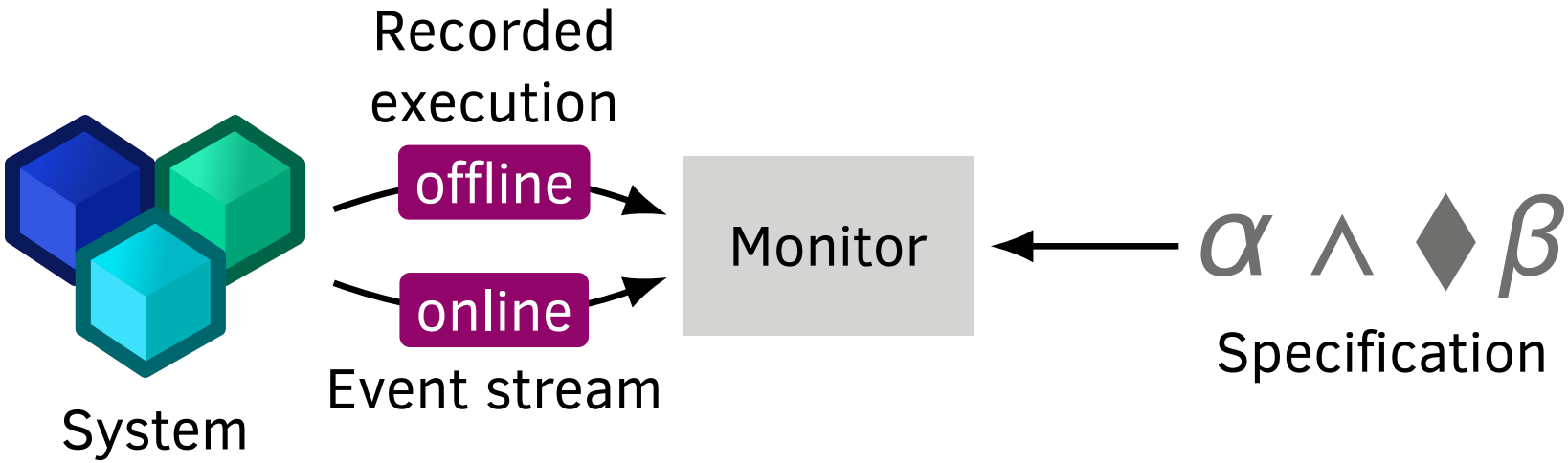
Runtime Verification



Runtime Verification



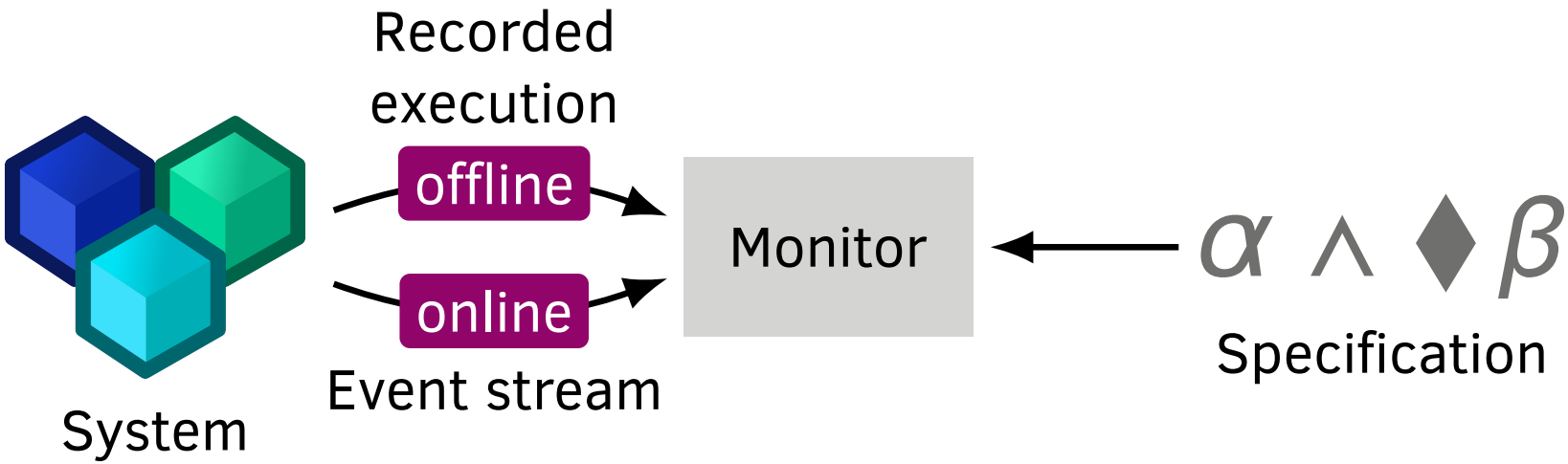
Runtime Verification



Example: a document management system

"A cached document must be updated to the latest version of its embedded resources (e.g., images) before being sent to a user."

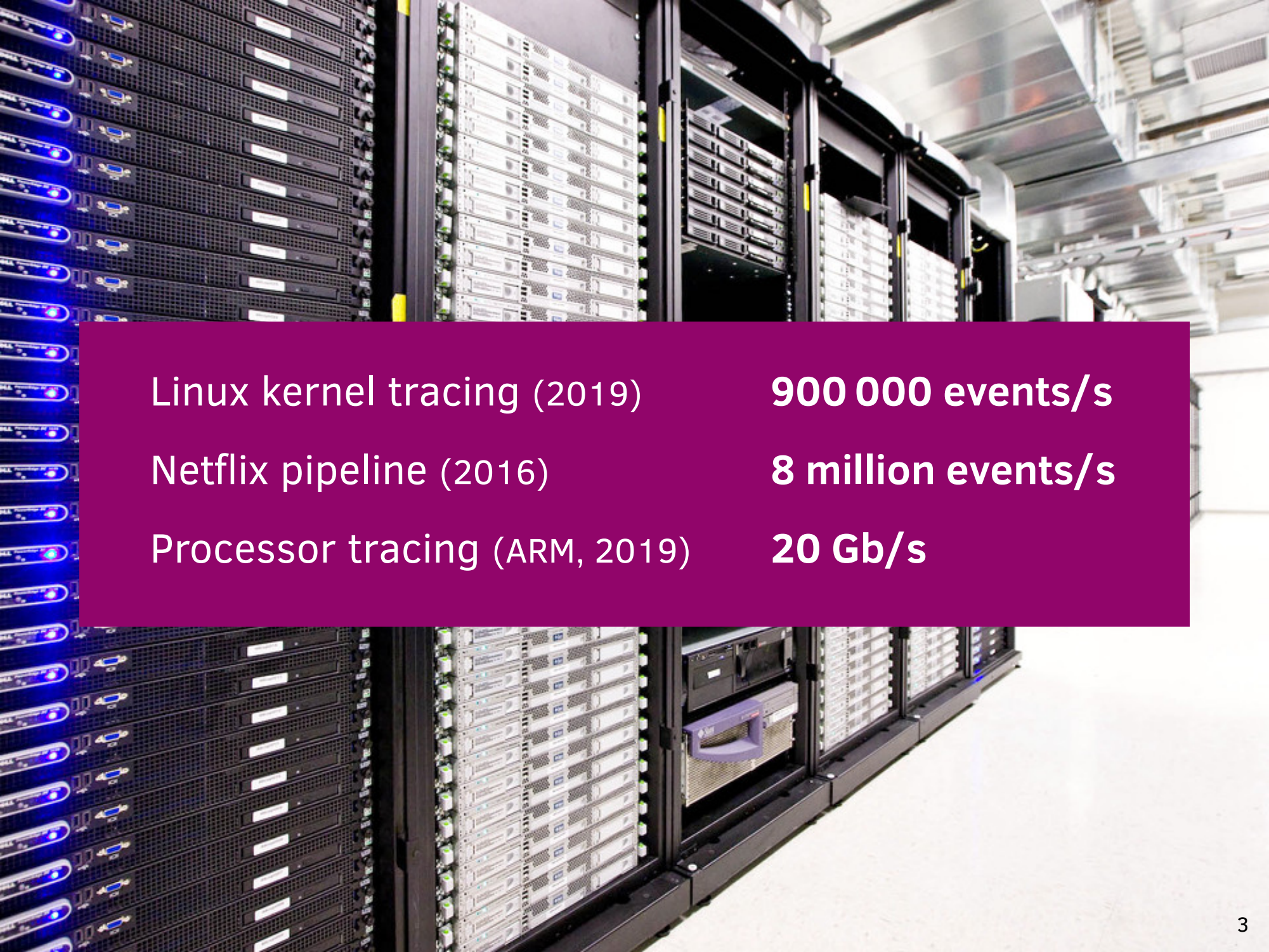
Runtime Verification



Example: a document management system

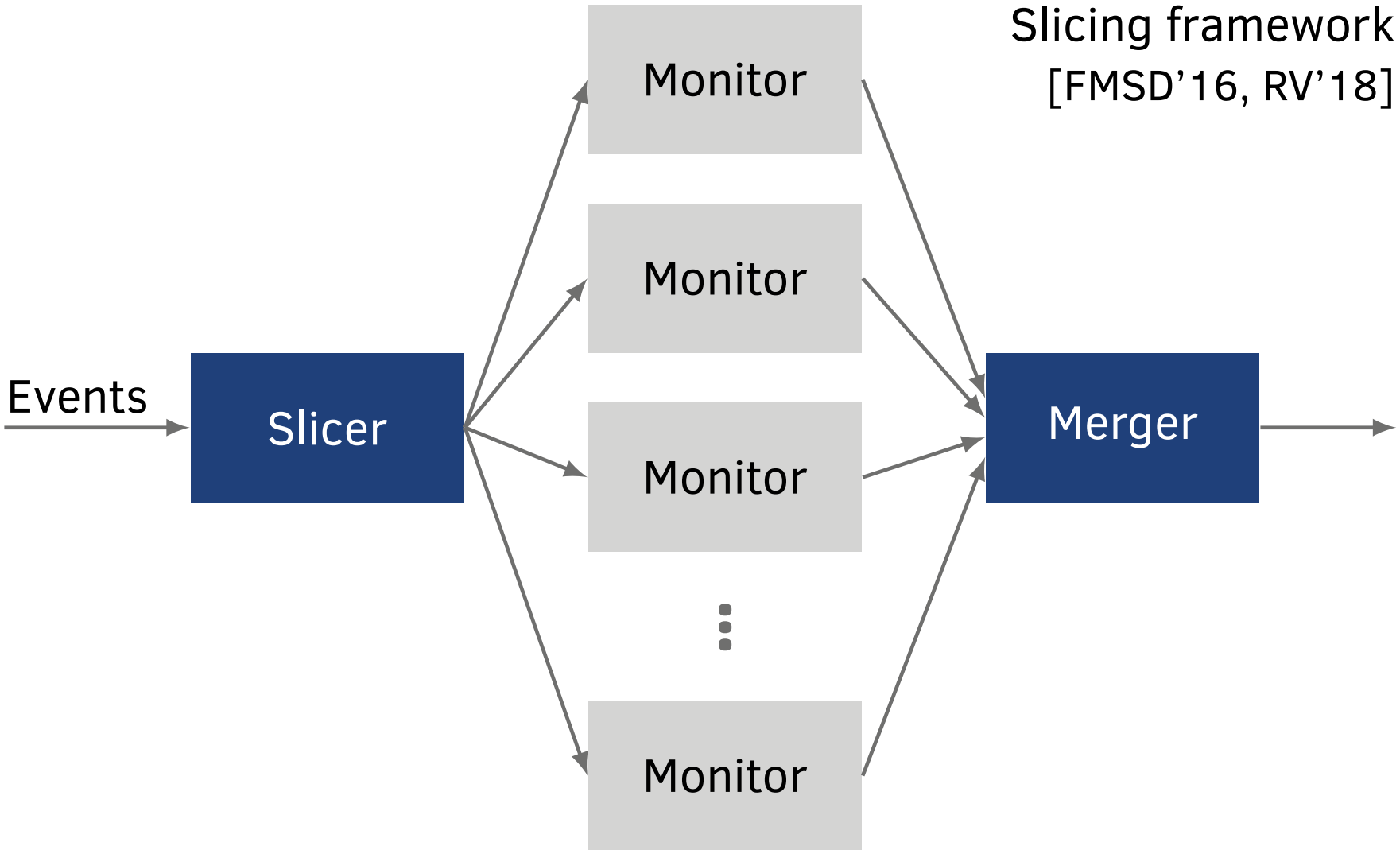
"A cached document must be updated to the latest version of its embedded resources (e.g., images) before being sent to a user."

Events: update(*d*), mod(*r*), addResource(*d, r*), send(*d*)

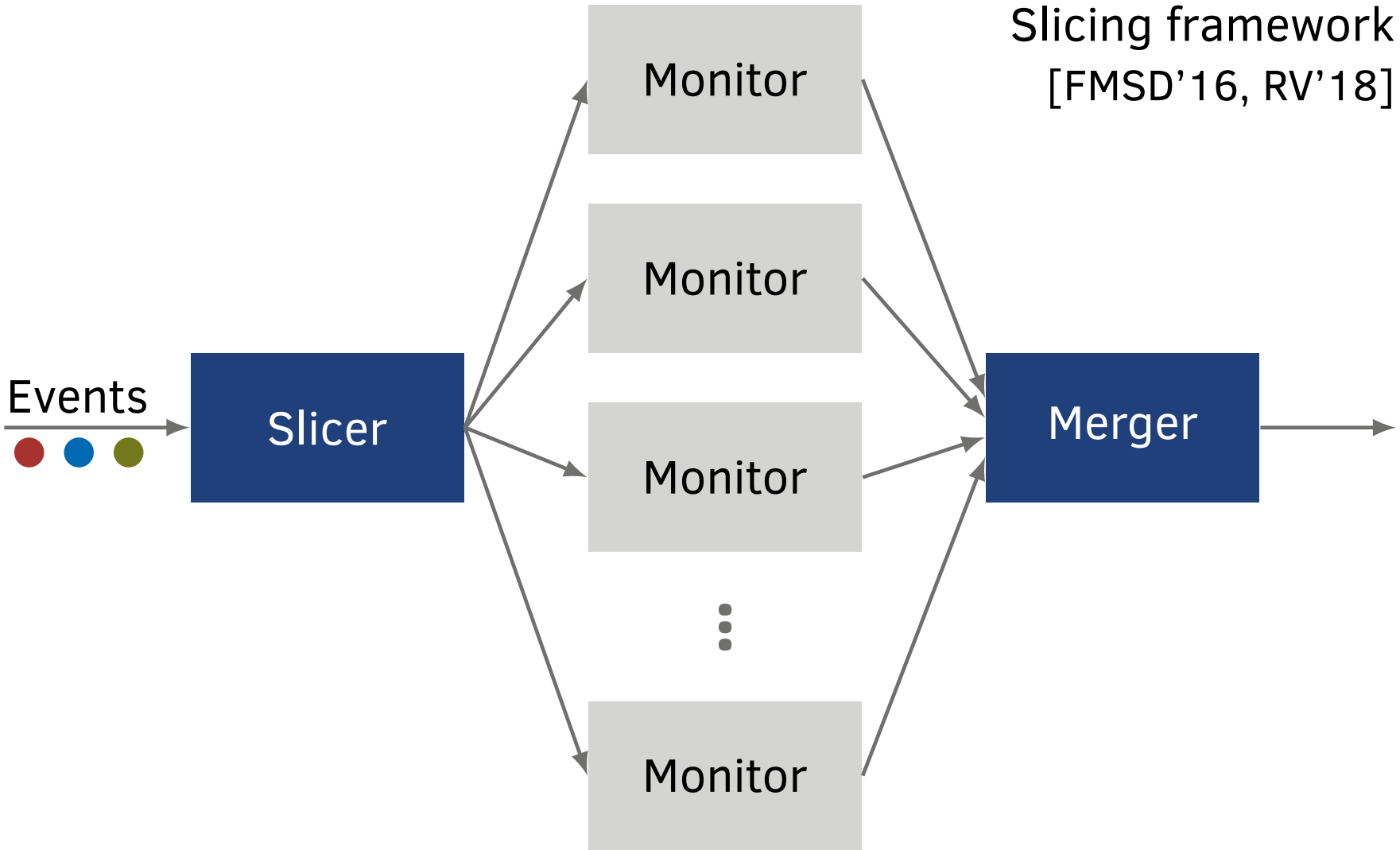


Linux kernel tracing (2019)	**900 000 events/s**
Netflix pipeline (2016)	**8 million events/s**
Processor tracing (ARM, 2019)	**20 Gb/s**

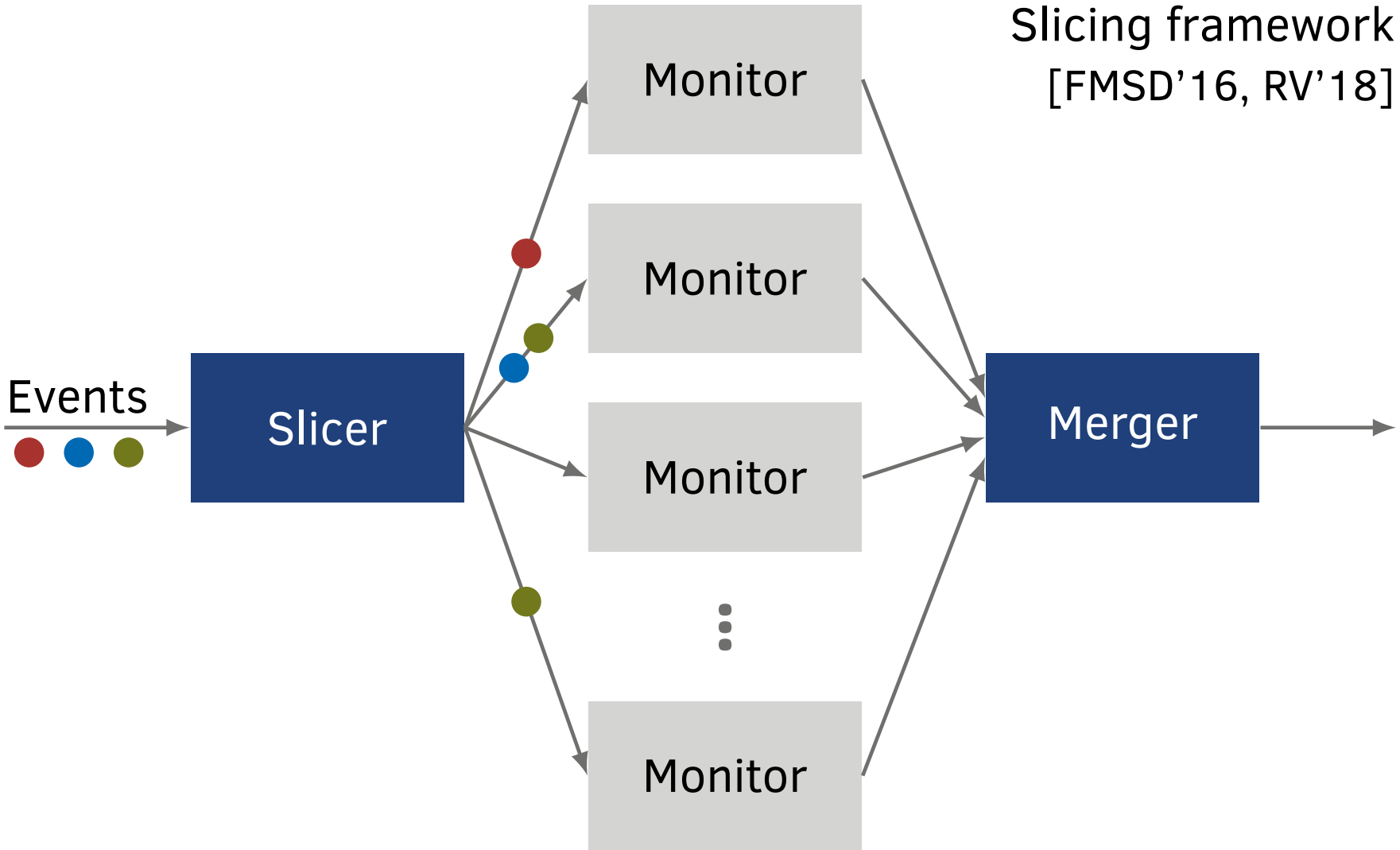
Parallel Online Monitoring



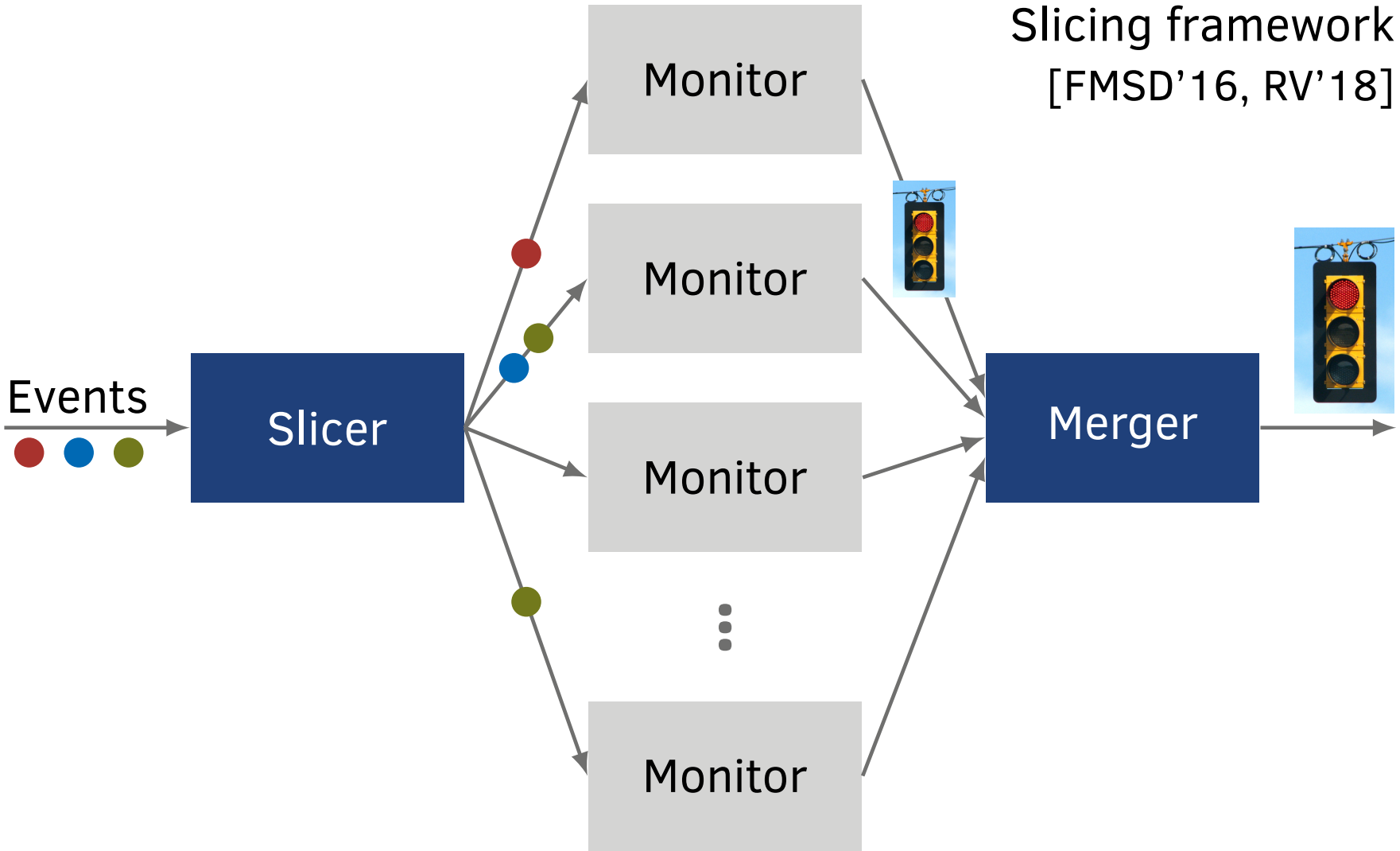
Parallel Online Monitoring



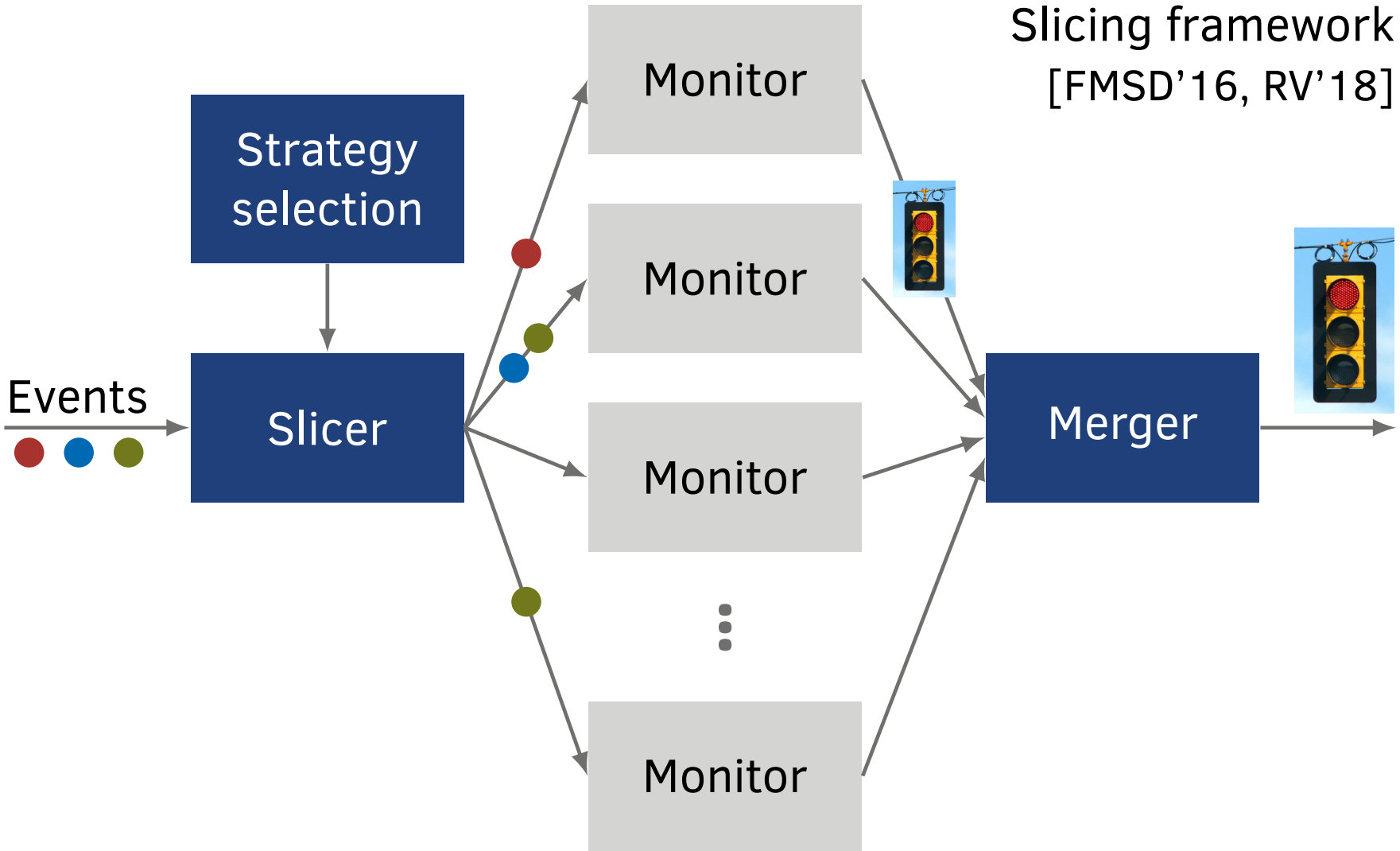
Parallel Online Monitoring



Parallel Online Monitoring



Parallel Online Monitoring



Slicing Strategies

The slicing strategy affects throughput and latency!

Slicing Strategies

The slicing strategy affects throughput and latency!

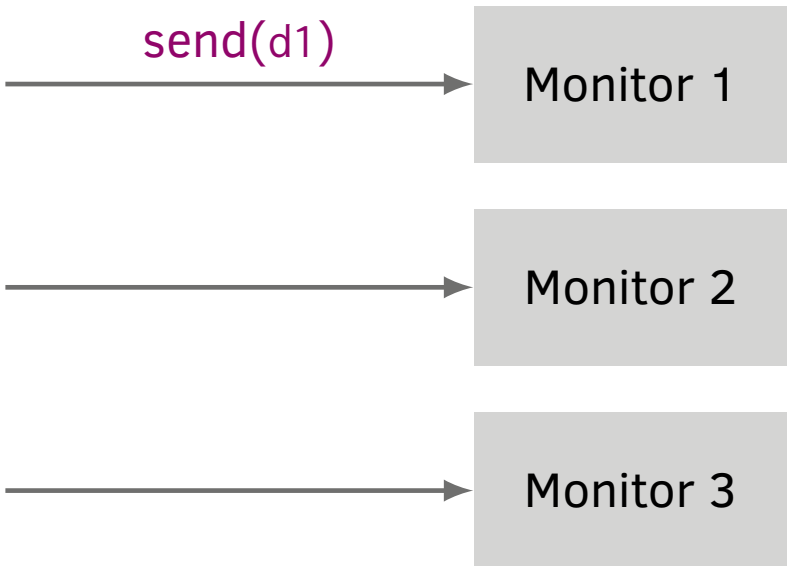
"A cached document must be updated to the latest version of its embedded resources (e.g., images) before being sent to a user."

Slicing Strategies

The slicing strategy affects throughput and latency!

"A cached document must be updated to the latest version of its embedded resources (e.g., images) before being sent to a user."

Partition documents:

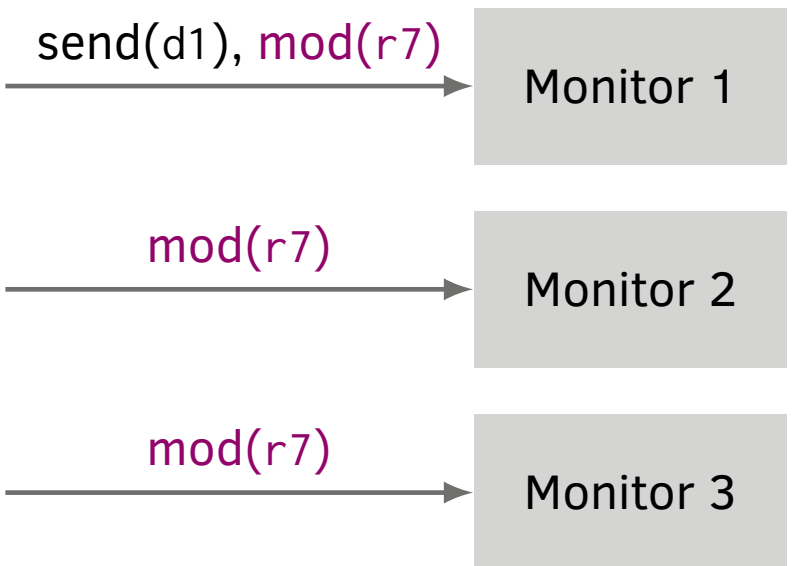


Slicing Strategies

The slicing strategy affects throughput and latency!

"A cached document must be updated to the latest version of its embedded resources (e.g., images) before being sent to a user."

Partition documents:

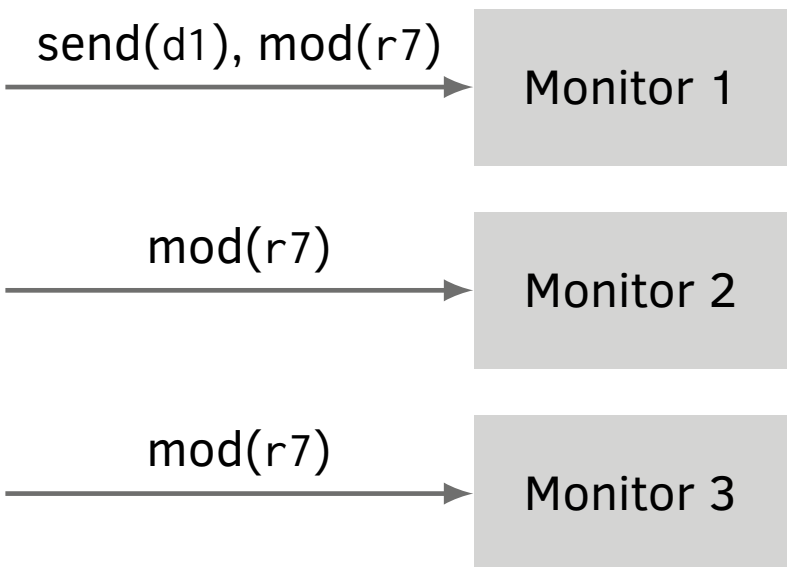


Slicing Strategies

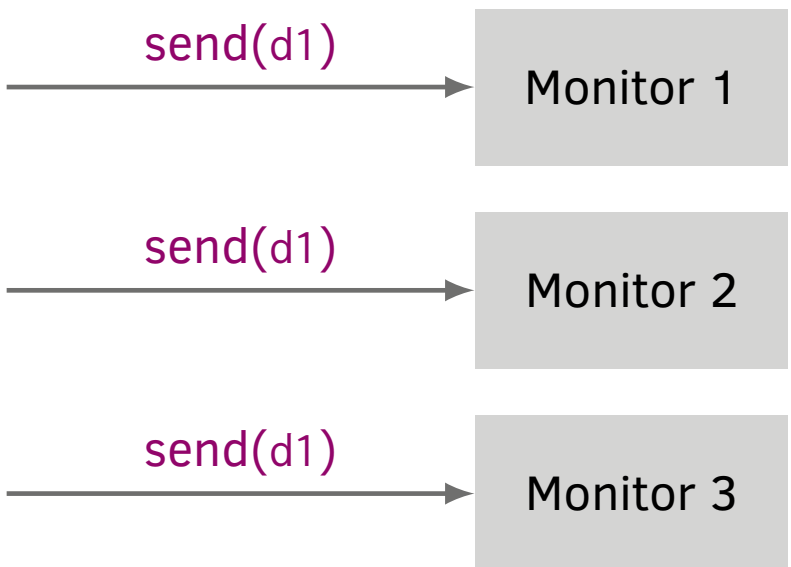
The slicing strategy affects throughput and latency!

"A cached document must be updated to the latest version of its embedded resources (e.g., images) before being sent to a user."

Partition documents:



Partition resources:

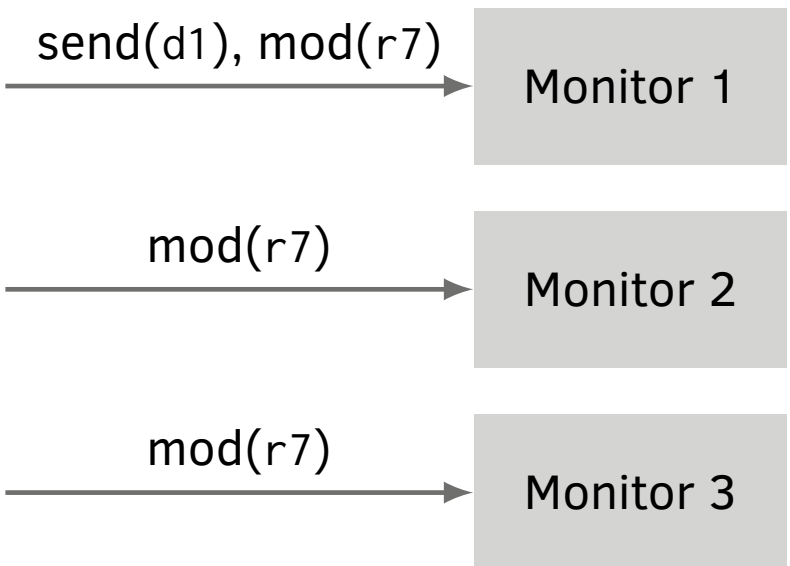


Slicing Strategies

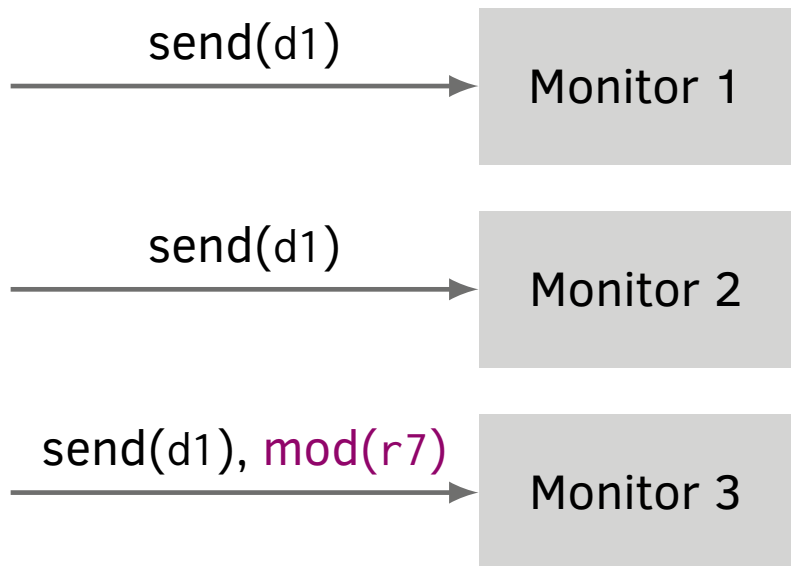
The slicing strategy affects throughput and latency!

"A cached document must be updated to the latest version of its embedded resources (e.g., images) before being sent to a user."

Partition documents:

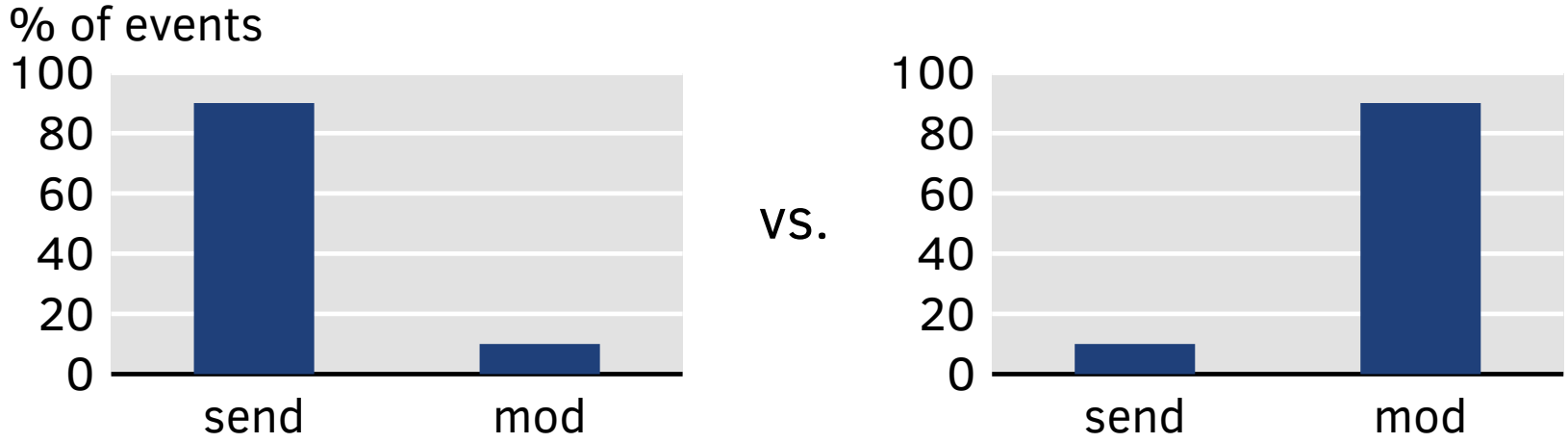


Partition resources:



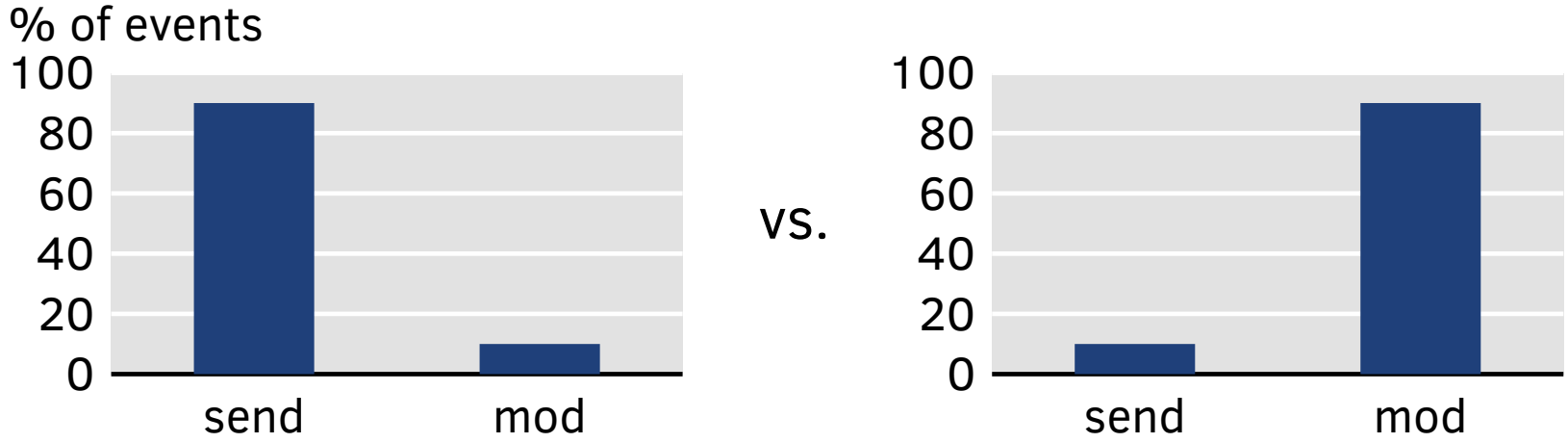
Stream Statistics

1. **Relative frequency** of the different types of events:

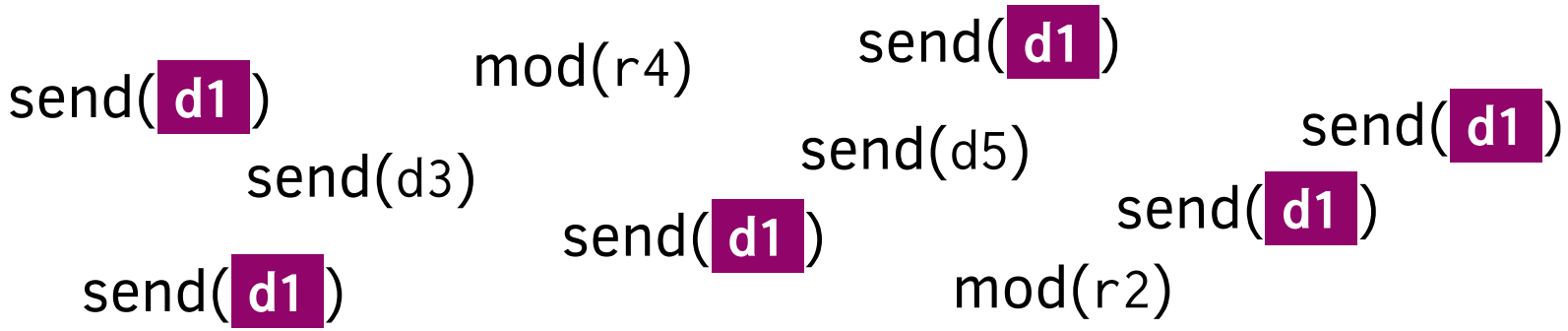


Stream Statistics

1. **Relative frequency** of the different types of events:



2. **Heavy hitters** (very frequent parameter values):



Problem Statement

What if stream statistics **change** over time?

Problem Statement

What if stream statistics **change** over time?

We want to **adapt the slicing strategy** at runtime to maintain balanced slices.

Problem Statement

What if stream statistics **change** over time?

We want to **adapt the slicing strategy** at runtime to maintain balanced slices.

Arising challenges:

- **When** should we adapt?
- **What** should the new strategy be?
- **How** can we implement the change correctly?

Problem Statement

What if stream statistics **change** over time?

We want to **adapt the slicing strategy** at runtime to maintain balanced slices.

Arising challenges:

- **When** should we adapt?
- **What** should the new strategy be?
- **How** can we implement the change correctly?
  (focus of this paper)

Contribution

The first **parallel online monitor** for first-order specifications capable of **adapting to changing stream statistics**.

Contribution

The first **parallel online monitor** for first-order specifications capable of **adapting to changing stream statistics**.

1. Adaptive monitoring algorithm
2. State migration operations for a metric first-order temporal logic (MFOTL) monitor
3. Correctness proved using Isabelle/HOL
4. State migration implemented in MonPoly
5. Evaluation: up to **tenfold run-time improvement**

Monitors

Monitor function $\mathcal{M}_\varphi$ for specification φ:

Monitors

Monitor function $\mathcal{M}_\varphi$ for specification φ:

Requirements:

- Monotonicity
- Soundness
- Completeness

} wrt. φ's semantics

Monitors

Monitor function $\mathcal{M}_\varphi$ for specification φ:

Requirements:

- Monotonicity
- Soundness
- Completeness

} wrt. φ's semantics

A (online) monitor computes $\mathcal{M}_\varphi$ (incrementally).

MFOTL

We focus on Metric First-Order Temporal Logic.

Example: $\mathrm{send}(d) \rightarrow \blacklozenge_{[0,1\mathrm{h}]}\, \mathrm{update}(d)$

MFOTL

We focus on Metric First-Order Temporal Logic.

Example: $\text{send}(d) \rightarrow \blacklozenge_{[0,1\text{h}]}\, \text{update}(d)$

	Time	*Events*	*Violations*
0	1.0h	{update(d1)}	{ }

MFOTL

We focus on Metric First-Order Temporal Logic.

Example: $\text{send}(d) \rightarrow \blacklozenge_{[0,1\text{h}]}\, \text{update}(d)$

	Time	*Events*	*Violations*
0	1.0h	{update(d1)}	{ }
1	1.5h	{send(d1), update(d2)}	{ }

MFOTL

We focus on Metric First-Order Temporal Logic.

Example: $\text{send}(d) \rightarrow \blacklozenge_{[0,1\text{h}]}\, \text{update}(d)$

	Time	*Events*	*Violations*
0	1.0h	{update(d1)}	{}
1	1.5h	{send(d1), update(d2)}	{}
2	2.5h	{send(d1), send(d3)}	{2 : d1, 2 : d3}

MFOTL

We focus on Metric First-Order Temporal Logic.

Example: $\text{send}(d) \rightarrow \blacklozenge_{[0,1\text{h}]}\, \text{update}(d)$

	Time	*Events*	*Violations*
0	1.0h	{update(d1)}	{}
1	1.5h	{send(d1), update(d2)}	{}
2	2.5h	{send(d1), send(d3)}	{2 : d1, 2 : d3}
3	3.0h	{update(d3)}	{}

MFOTL

We focus on Metric First-Order Temporal Logic.

Example: $\text{send}(d) \rightarrow \blacklozenge_{[0,1\text{h}]}\, \text{update}(d)$

	Time	*Events*	*Violations*
0	1.0h	{update(d1)}	{}
1	1.5h	{send(d1), update(d2)}	{}
2	2.5h	{send(d1), send(d3)}	{2 : d1, 2 : d3}
3	3.0h	{update(d3)}	{}
4	3.5h	{send(d3)}	{}

MFOTL

We focus on Metric First-Order Temporal Logic.

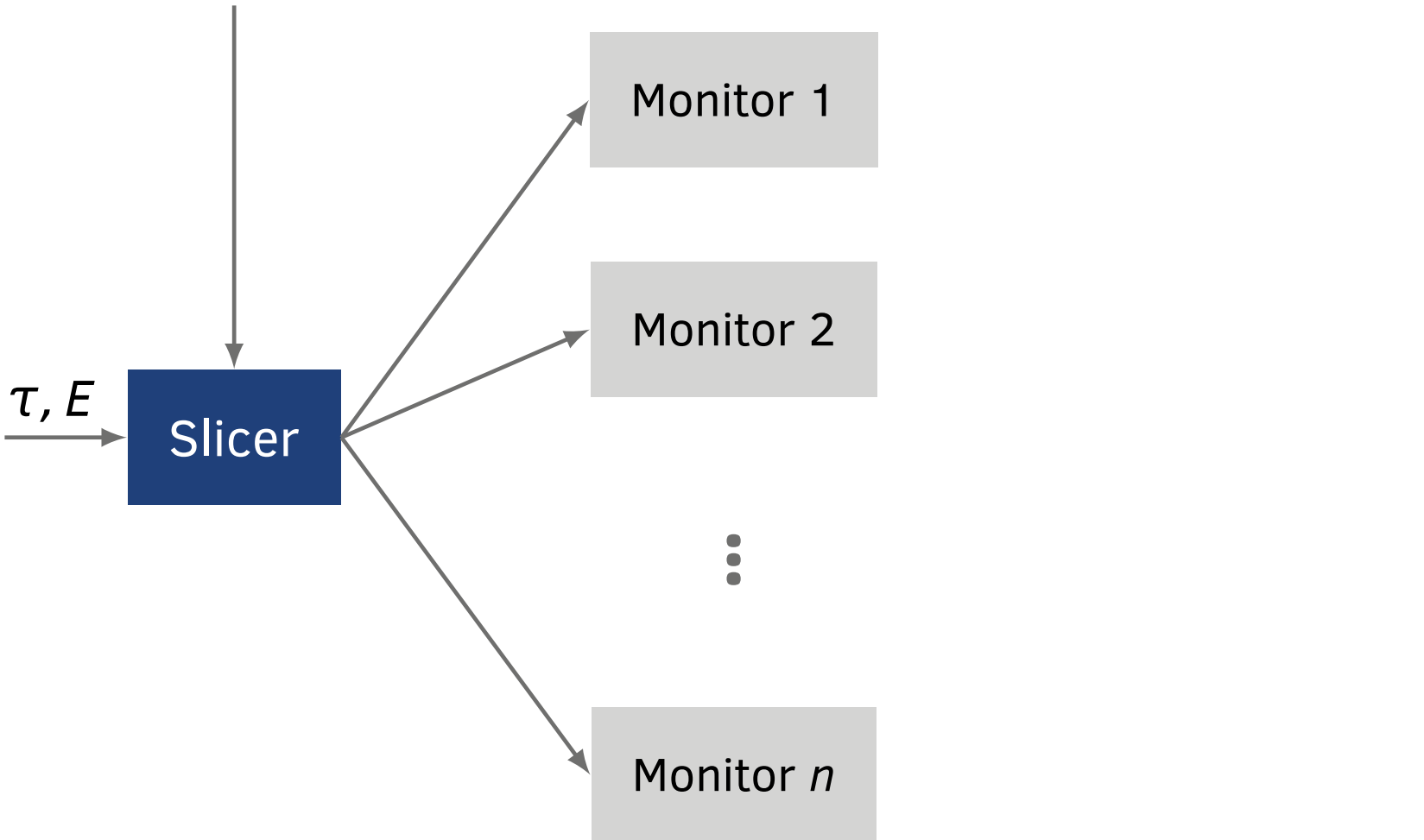
Example: $\text{send}(d) \rightarrow \blacklozenge_{[0,1\text{h}]}\, \text{update}(d)$

	Time	*Events*	*Violations*
0	1.0h	{update(d1)}	{}
1	1.5h	{send(d1), update(d2)}	{}
2	2.5h	{send(d1), send(d3)}	{2 : d1, ...
3	3.0h	{update(d3)}	
4	3.5h	{send(d3)}	

Monitor implemented in the MonPoly tool [JACM'15, RV-CuBES'17]

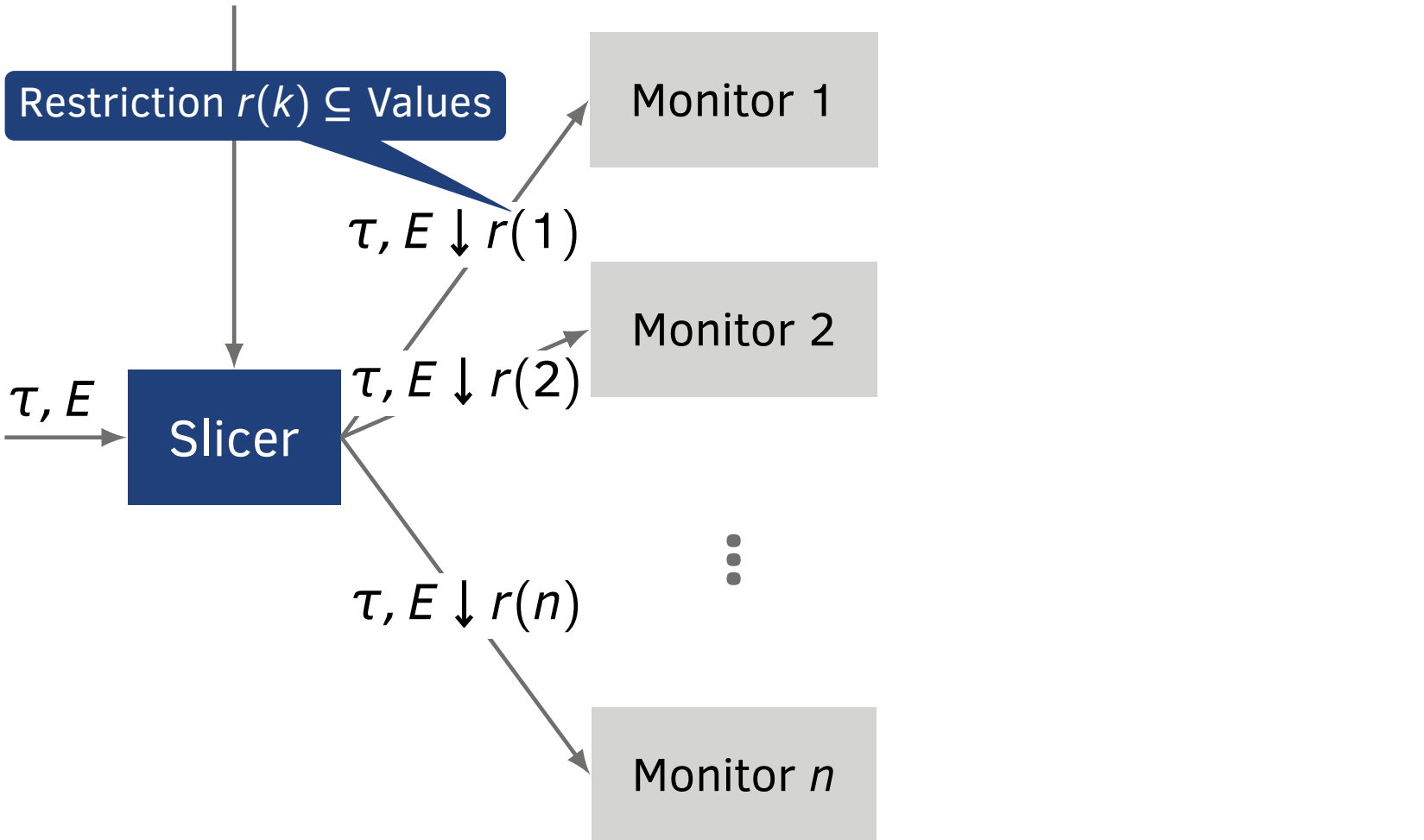
(Simplified) Data Slicer [RV'18]

Strategy $r : \{1, \ldots, n\} \to \mathcal{P}(\text{Values})$ with $\bigcup_k r(k) = \text{Values}$

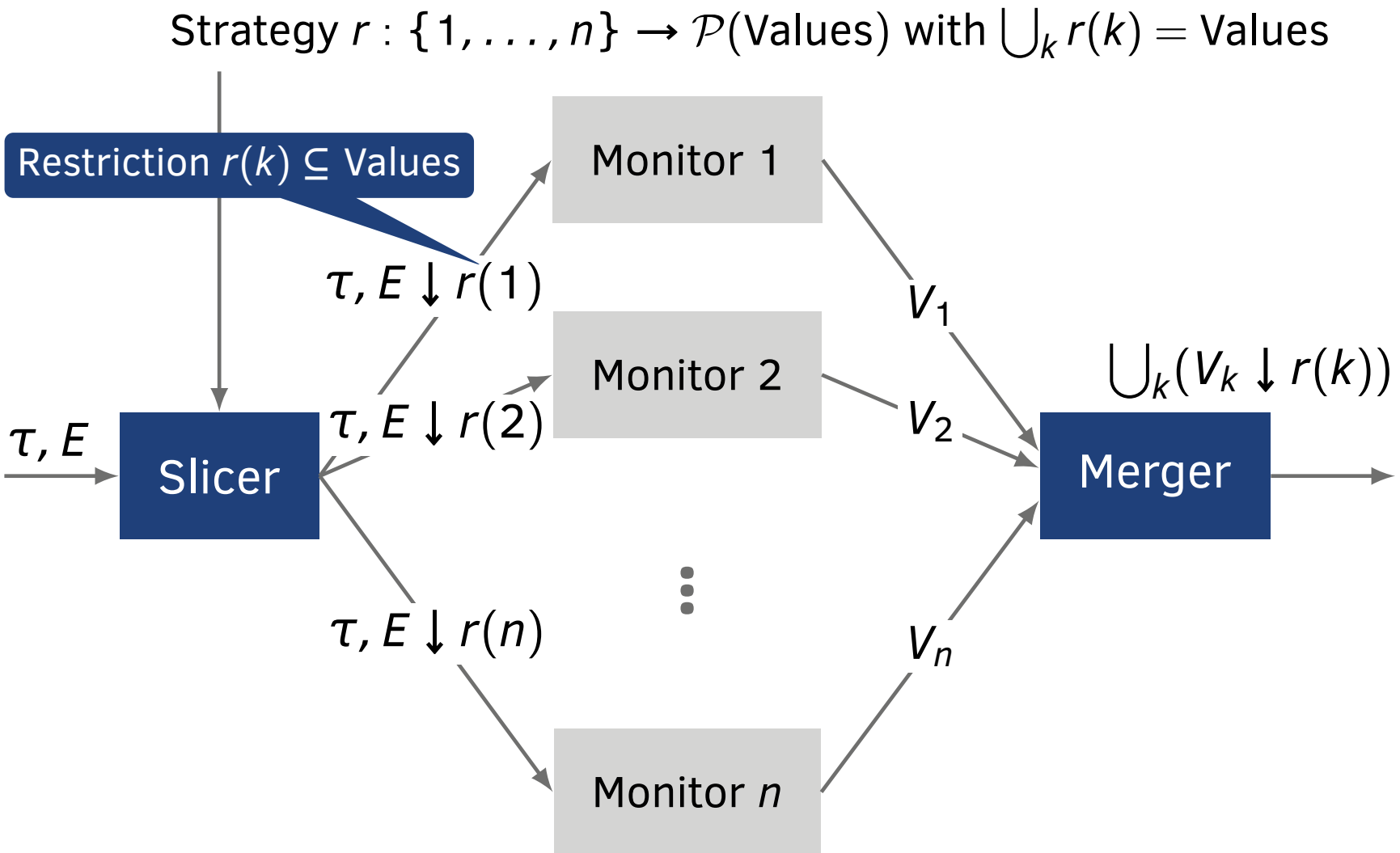


(Simplified) Data Slicer [RV'18]

Strategy $r : \{1, \ldots, n\} \to \mathcal{P}(\text{Values})$ with $\bigcup_k r(k) = \text{Values}$



(Simplified) Data Slicer [RV'18]



Problem: State Consistency

A submonitor's state stores a partial history of its input.

Example: $\text{send}(d) \rightarrow \blacklozenge_{[0,1\text{h}]}\, \text{update}(d)$

Problem: State Consistency

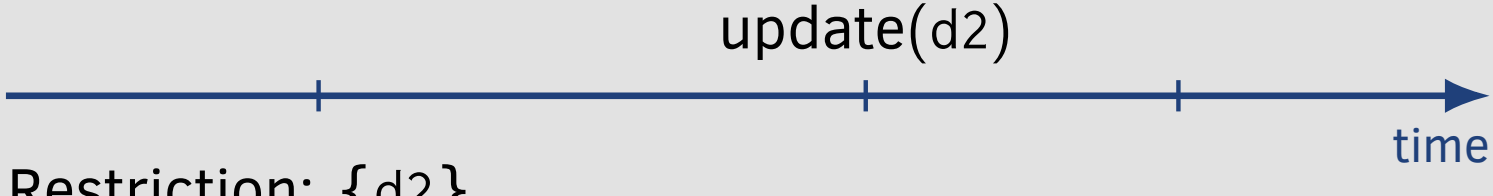
A submonitor's state stores a partial history of its input.

Example: $\text{send}(d) \rightarrow \blacklozenge_{[0,1\text{h}]}\, \text{update}(d)$

1

Restriction: {d1, d3}

2

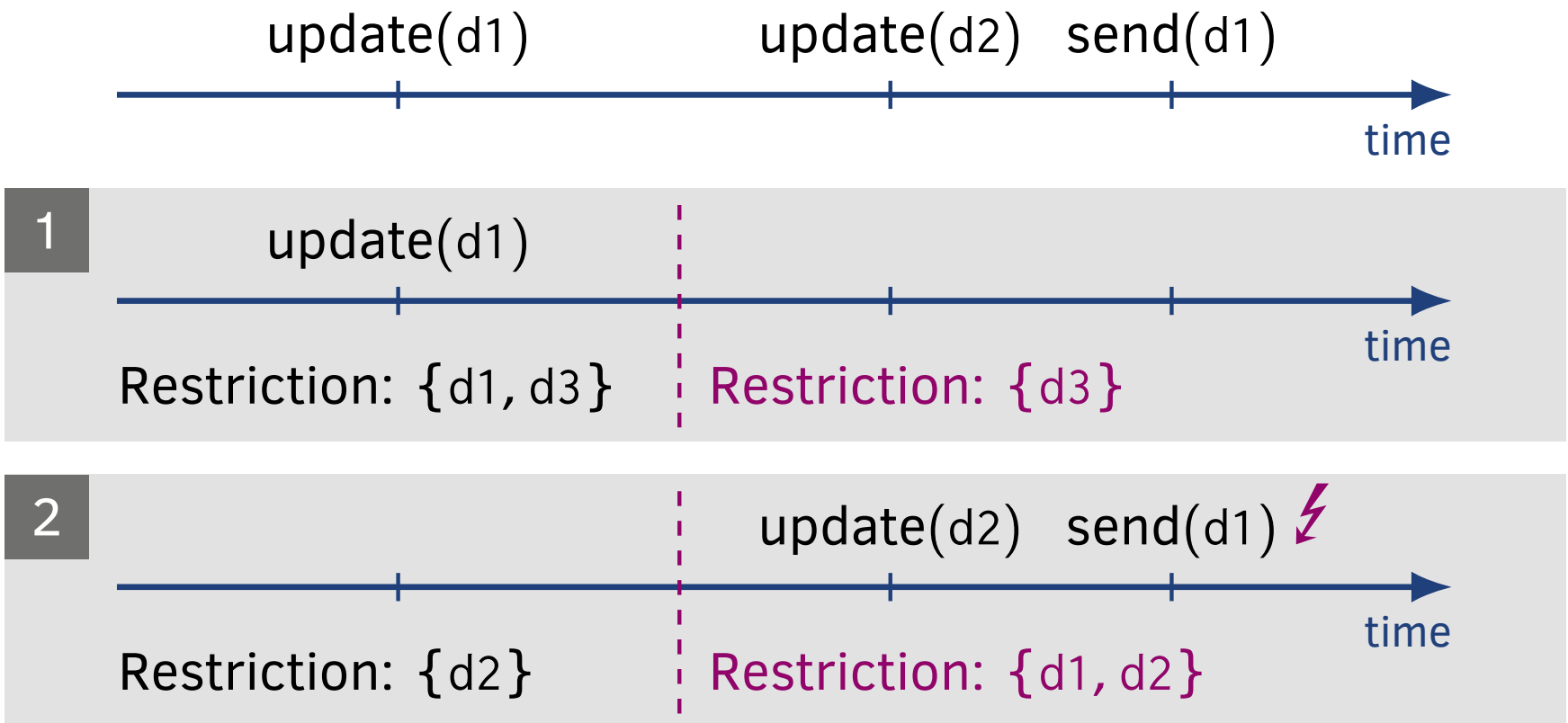


Restriction: {d2}

Problem: State Consistency

A submonitor's state stores a partial history of its input.

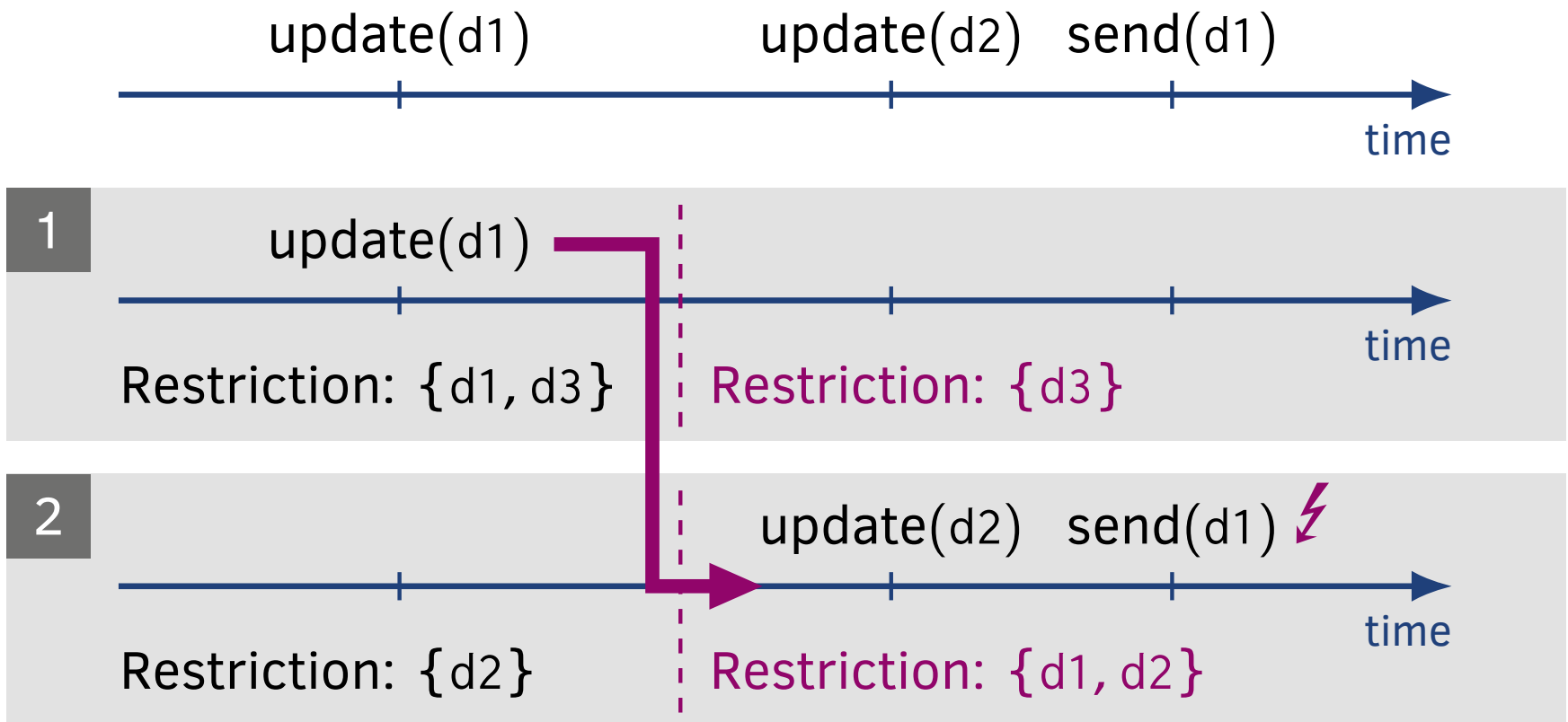
Example: $\text{send}(d) \rightarrow \blacklozenge_{[0,1\text{h}]}\, \text{update}(d)$



Problem: State Consistency

A submonitor's state stores a partial history of its input.

Example: $\text{send}(d) \rightarrow \blacklozenge_{[0,1\text{h}]}\, \text{update}(d)$



Solution: Adaptive Algorithm

Parameter: sequence $(r_i)_{i \geq 0}$ of slicing strategies

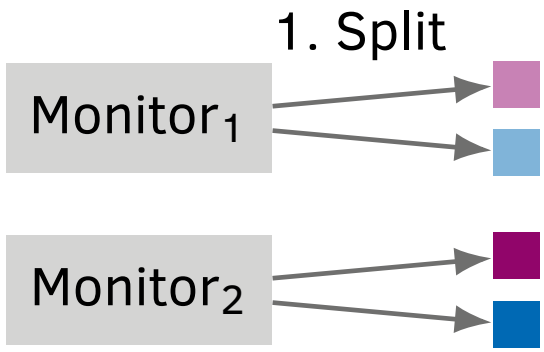
for the i-th input $\langle \tau_i, E_i \rangle$, $i \geq 1$, **do**

Solution: Adaptive Algorithm

Parameter: sequence $(r_i)_{i \geq 0}$ of slicing strategies

for the i-th input $\langle \tau_i, E_i \rangle$, $i \geq 1$, **do**

if $r_i \neq r_{i-1}$ **then**



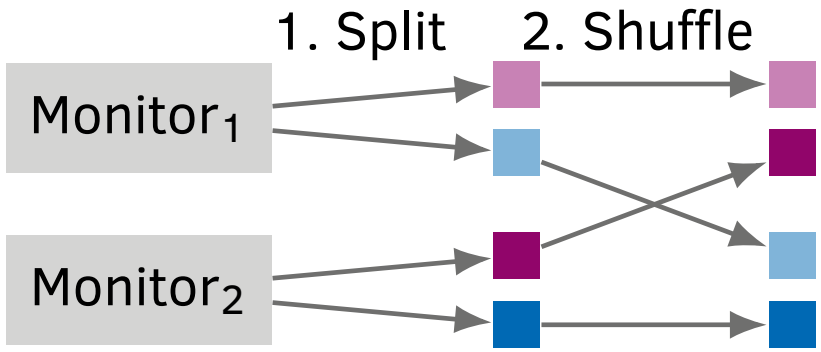
end

Solution: Adaptive Algorithm

Parameter: sequence $(r_i)_{i \geq 0}$ of slicing strategies

for the i-th input $\langle \tau_i, E_i \rangle$, $i \geq 1$, **do**

if $r_i \neq r_{i-1}$ **then**



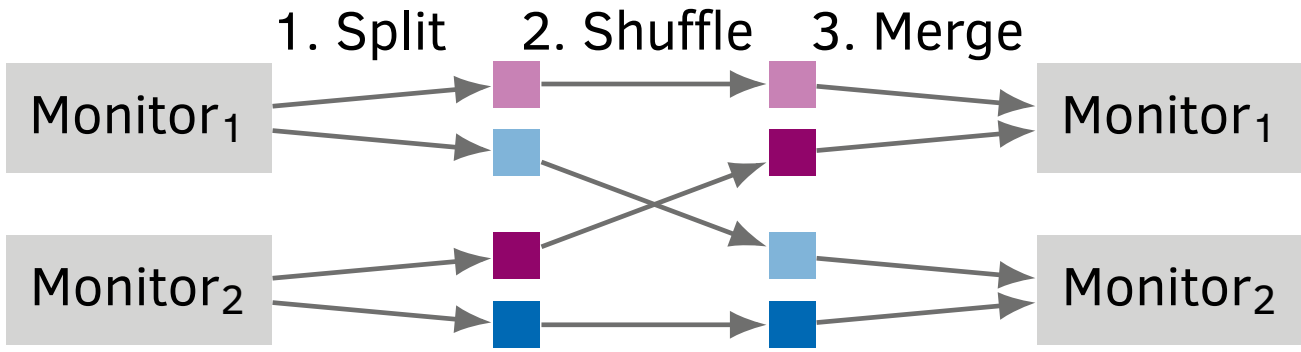
end

Solution: Adaptive Algorithm

Parameter: sequence $(r_i)_{i \geq 0}$ of slicing strategies

for the i-th input $\langle \tau_i, E_i \rangle$, $i \geq 1$, **do**

if $r_i \neq r_{i-1}$ **then**



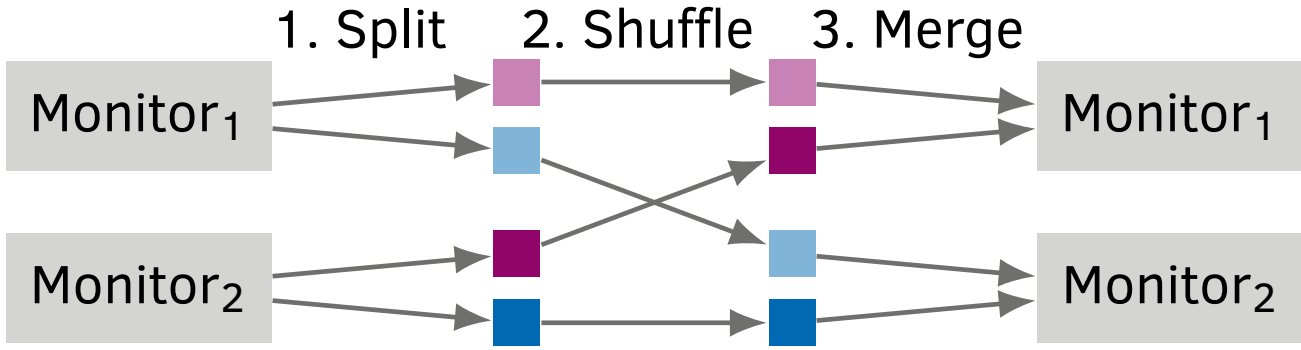
end

Solution: Adaptive Algorithm

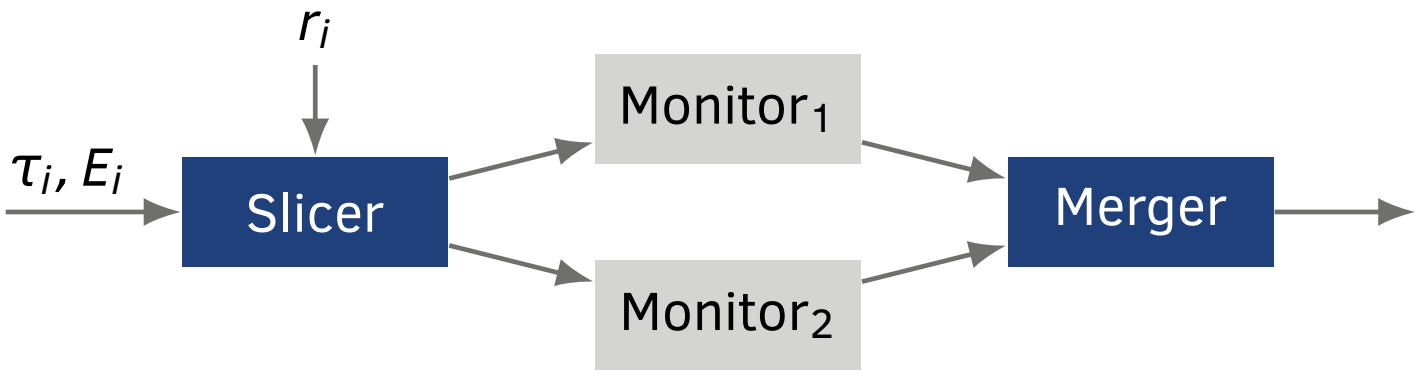
Parameter: sequence $(r_i)_{i \geq 0}$ of slicing strategies

for the i-th input $\langle \tau_i, E_i \rangle$, $i \geq 1$, **do**

if $r_i \neq r_{i-1}$ **then**

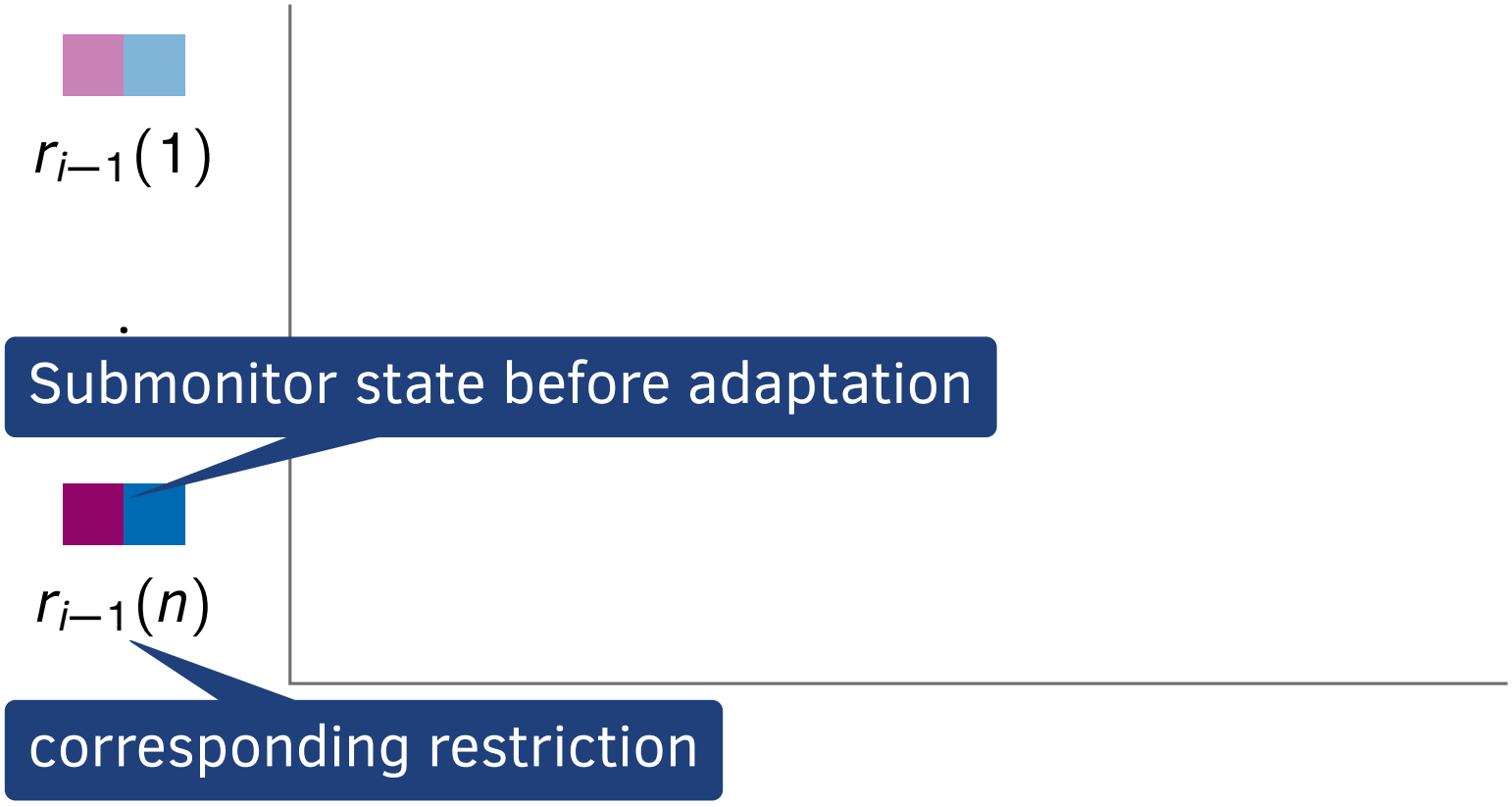


end

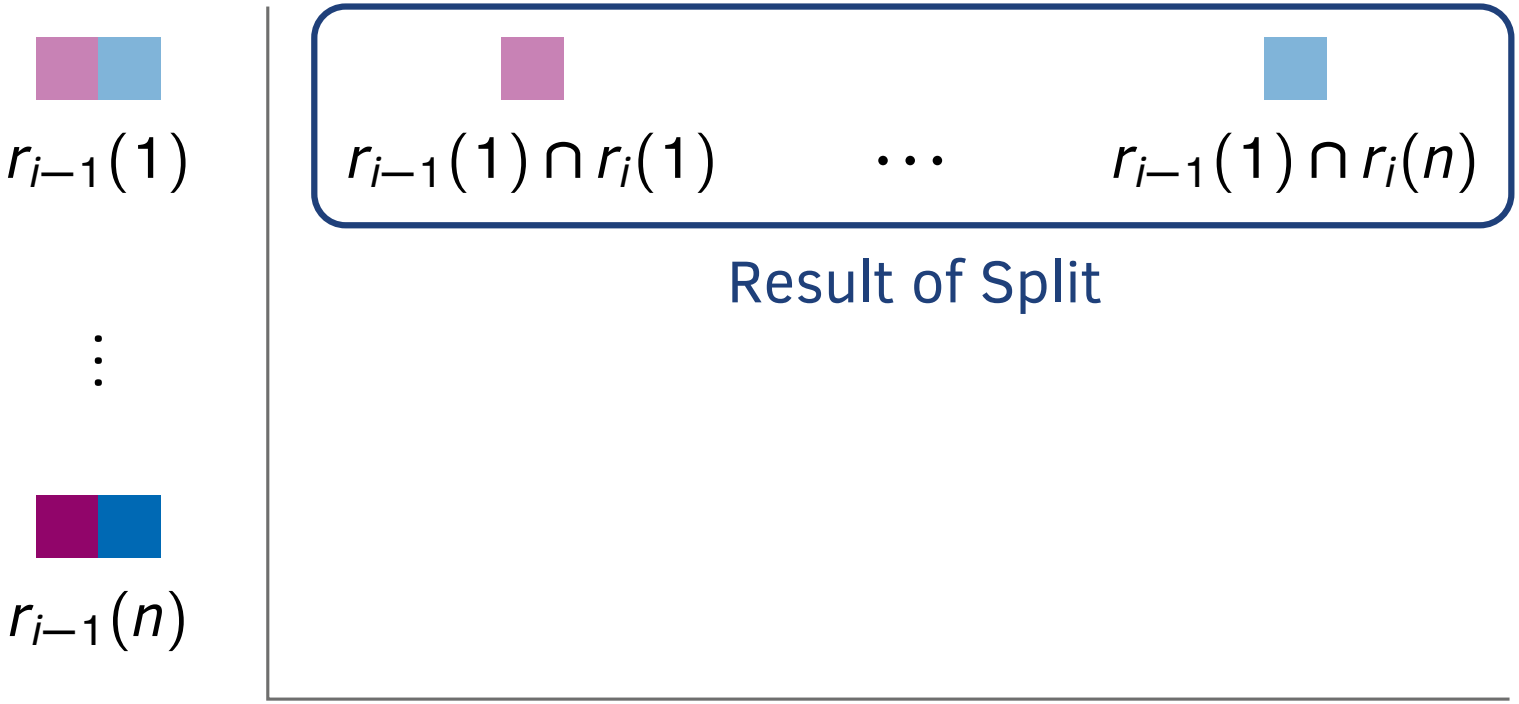


end

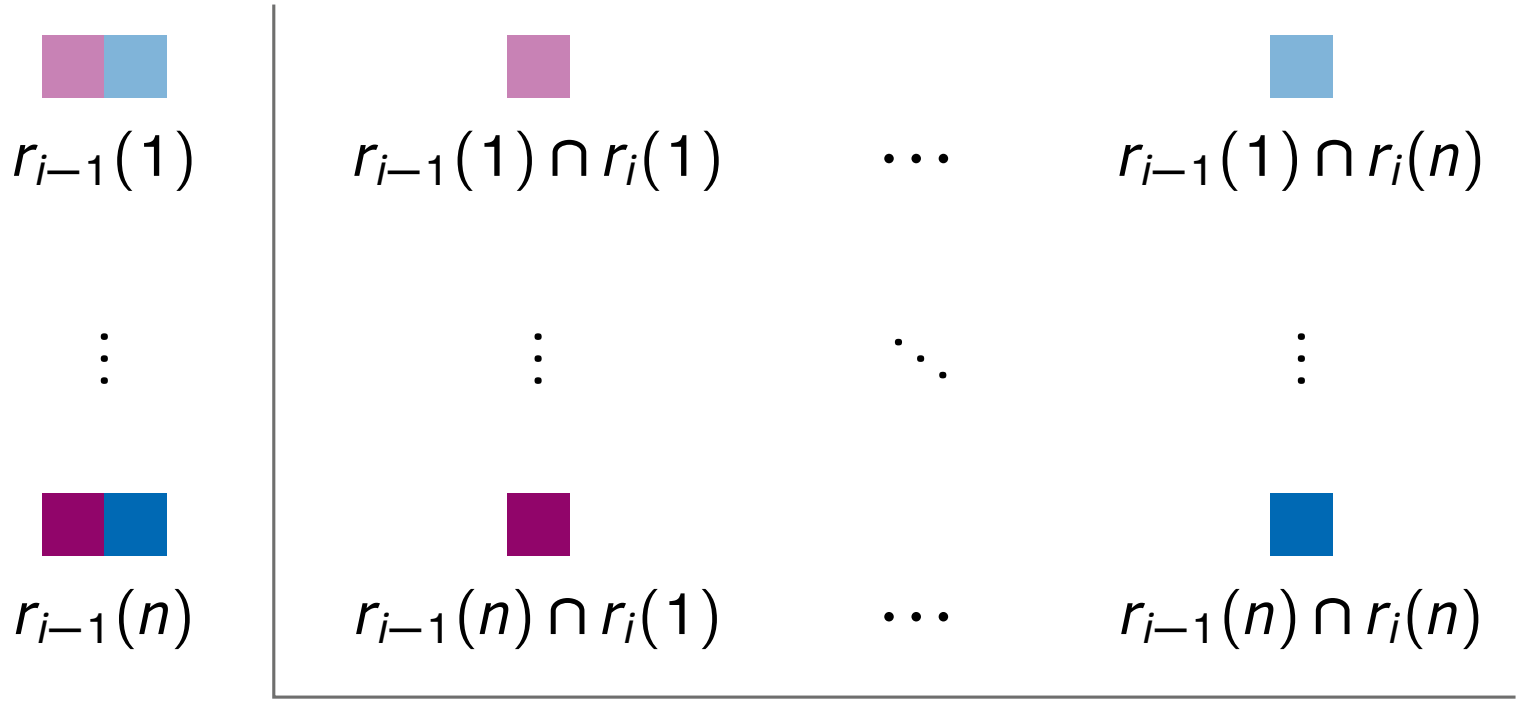
Split & Merge Interface



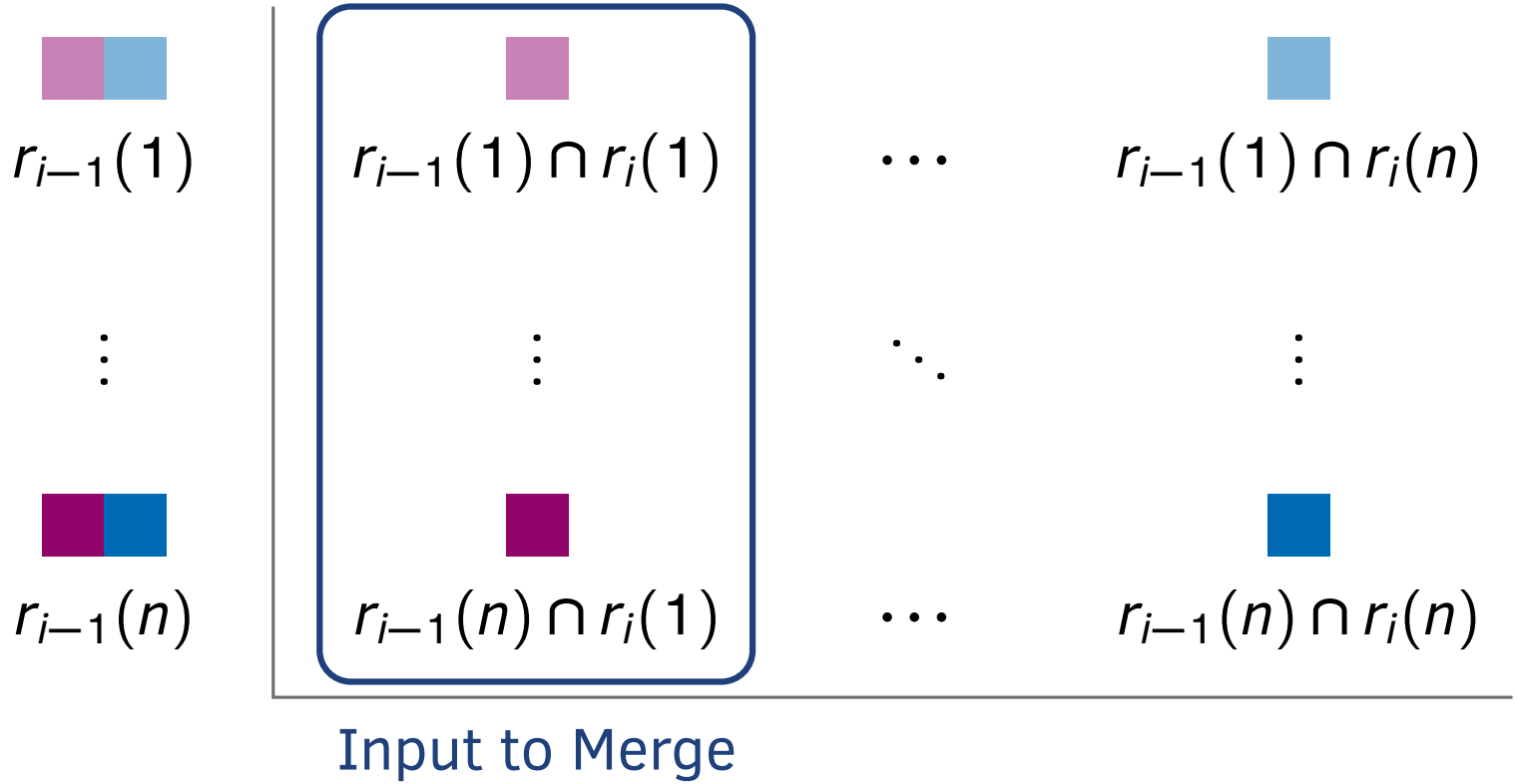
Split & Merge Interface



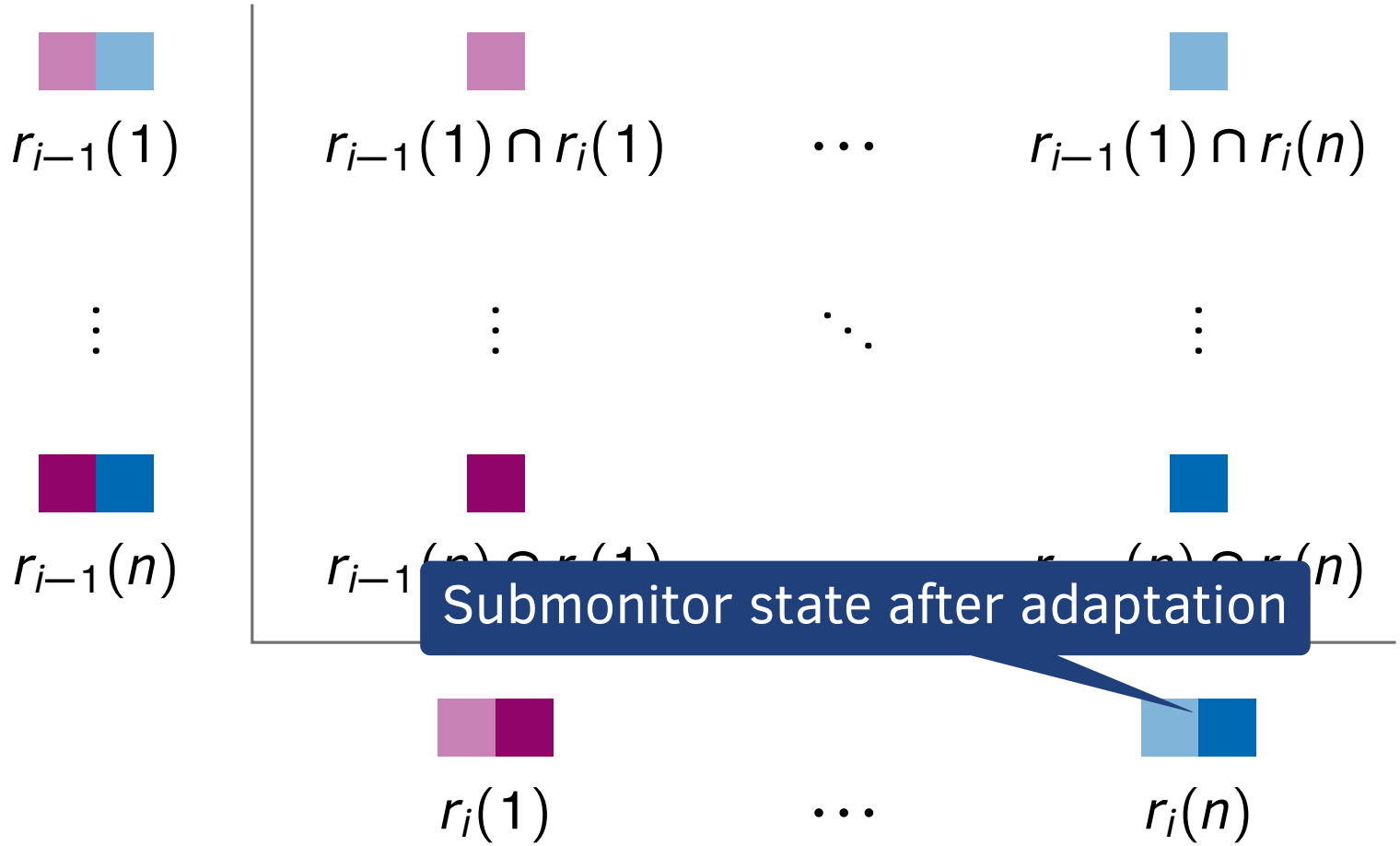
Split & Merge Interface



Split & Merge Interface

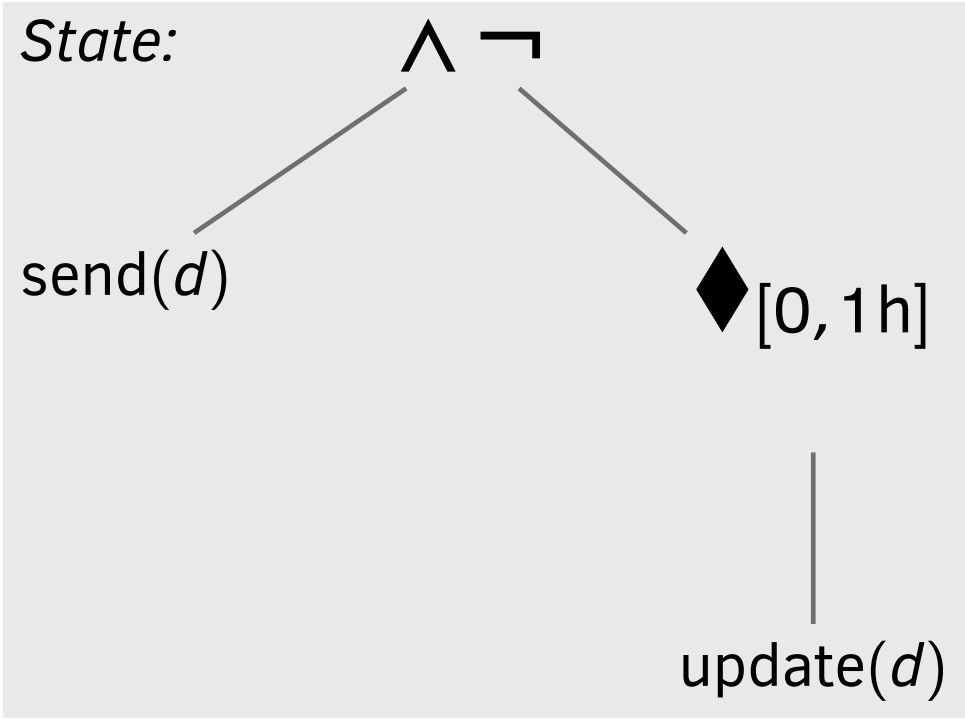


Split & Merge Interface



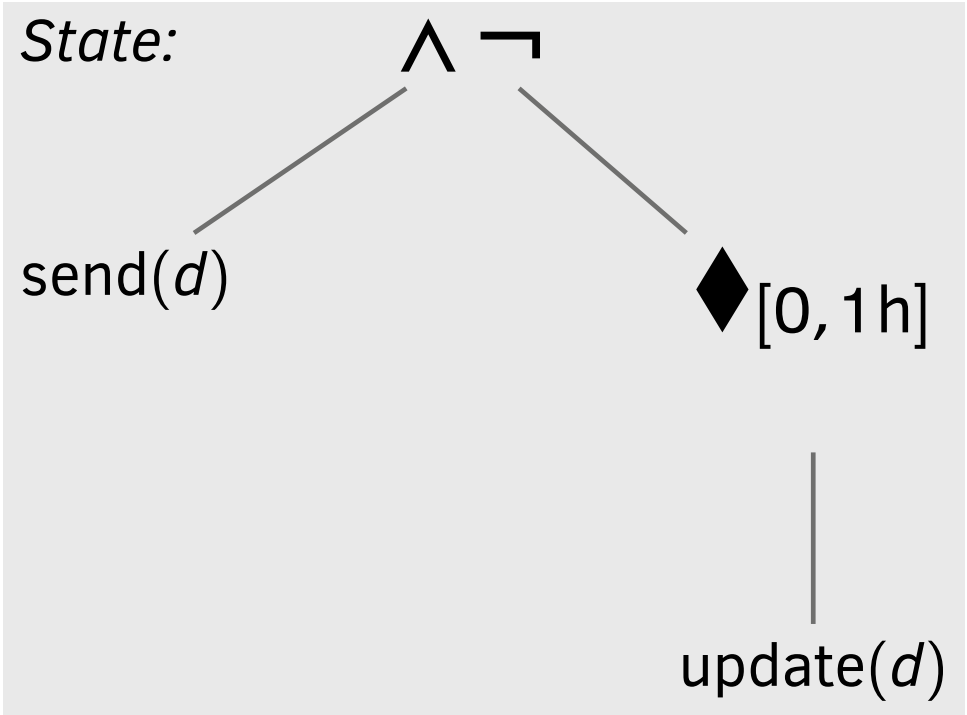
MonPoly's Monitor State

Evaluating $\text{send}(d) \wedge \neg \blacklozenge_{[0,1h]} \text{update}(d)$ to find violations:

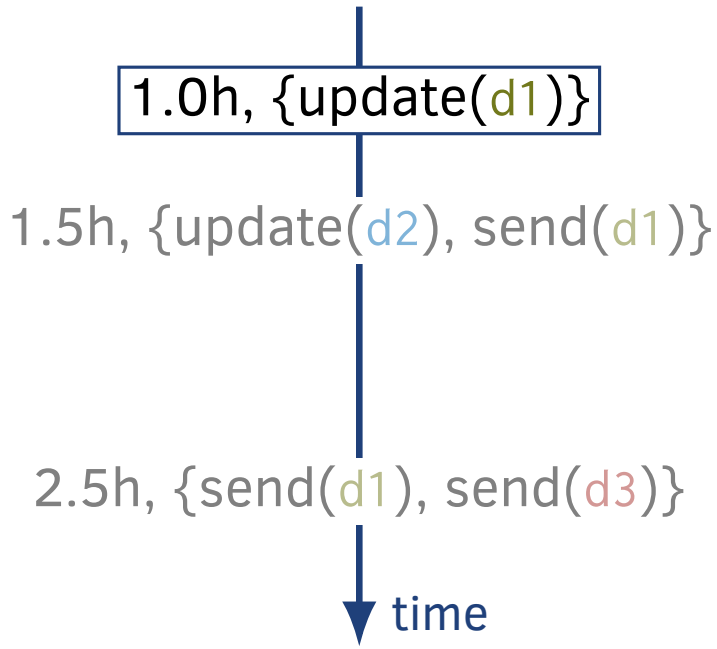


MonPoly's Monitor State

Evaluating $\text{send}(d) \wedge \neg \blacklozenge_{[0,1h]} \text{update}(d)$ to find violations:



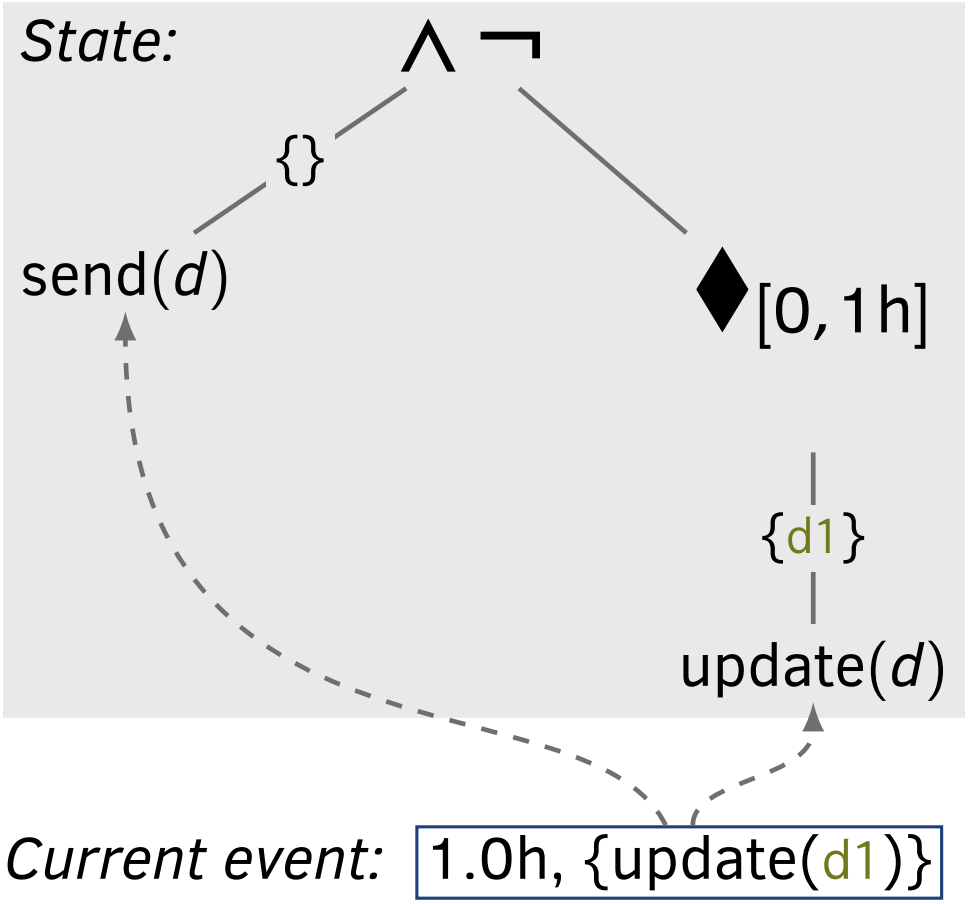
Trace:

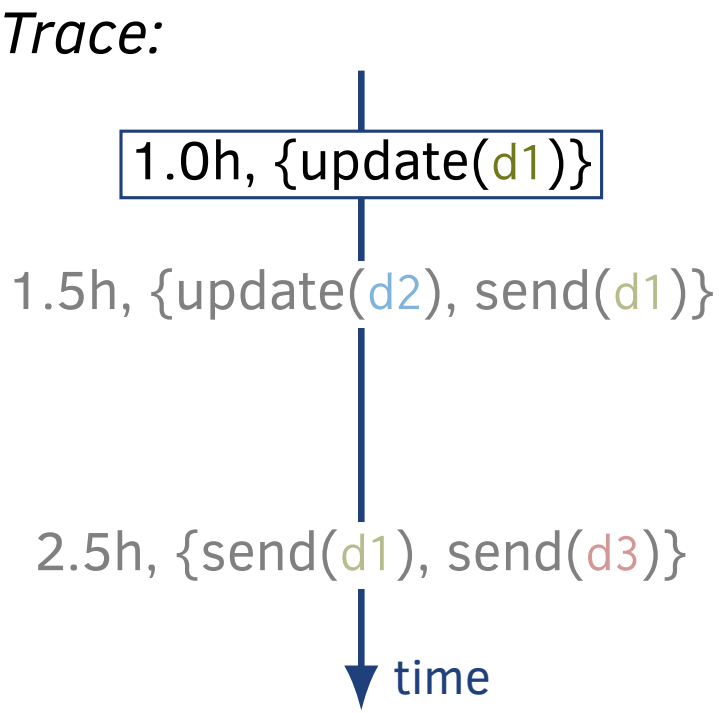


Current event: 1.0h, {update(d1)}

MonPoly's Monitor State

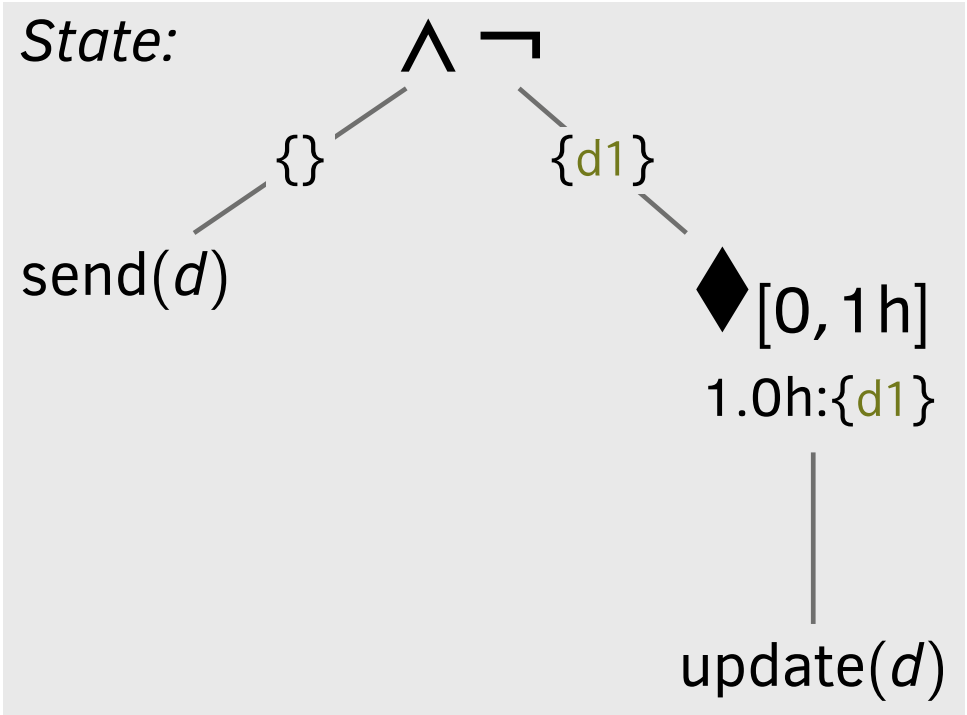
Evaluating $\text{send}(d) \wedge \neg \blacklozenge_{[0,1h]} \text{update}(d)$ to find violations:

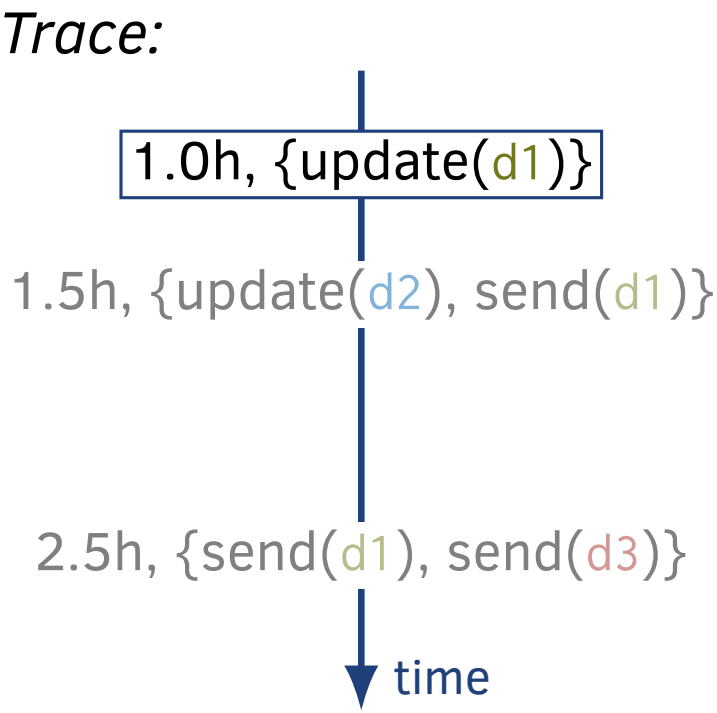




MonPoly's Monitor State

Evaluating $\text{send}(d) \wedge \neg \blacklozenge_{[0,1h]} \text{update}(d)$ to find violations:

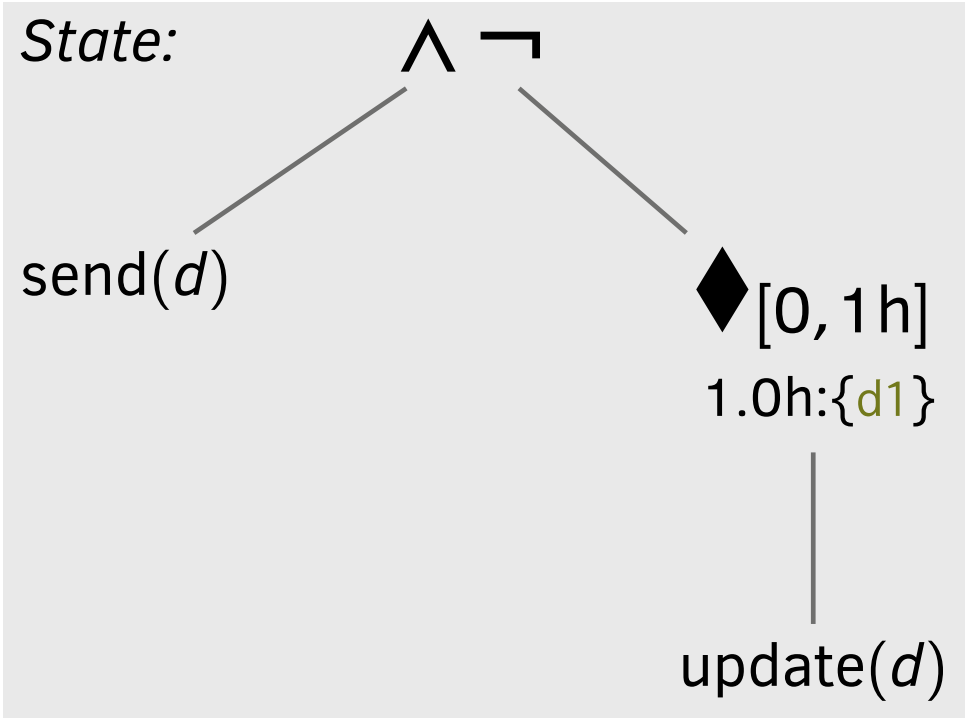


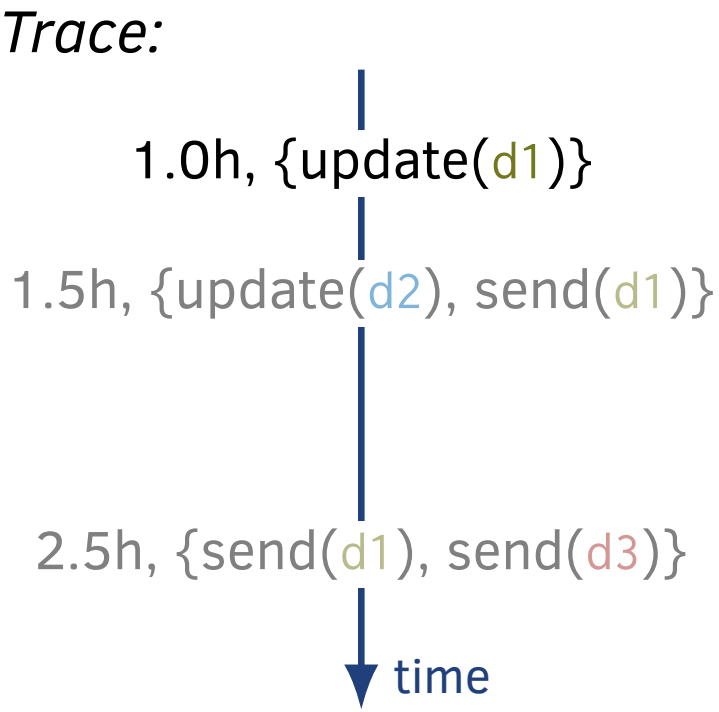


Current event: 1.0h, {update(d1)}

MonPoly's Monitor State

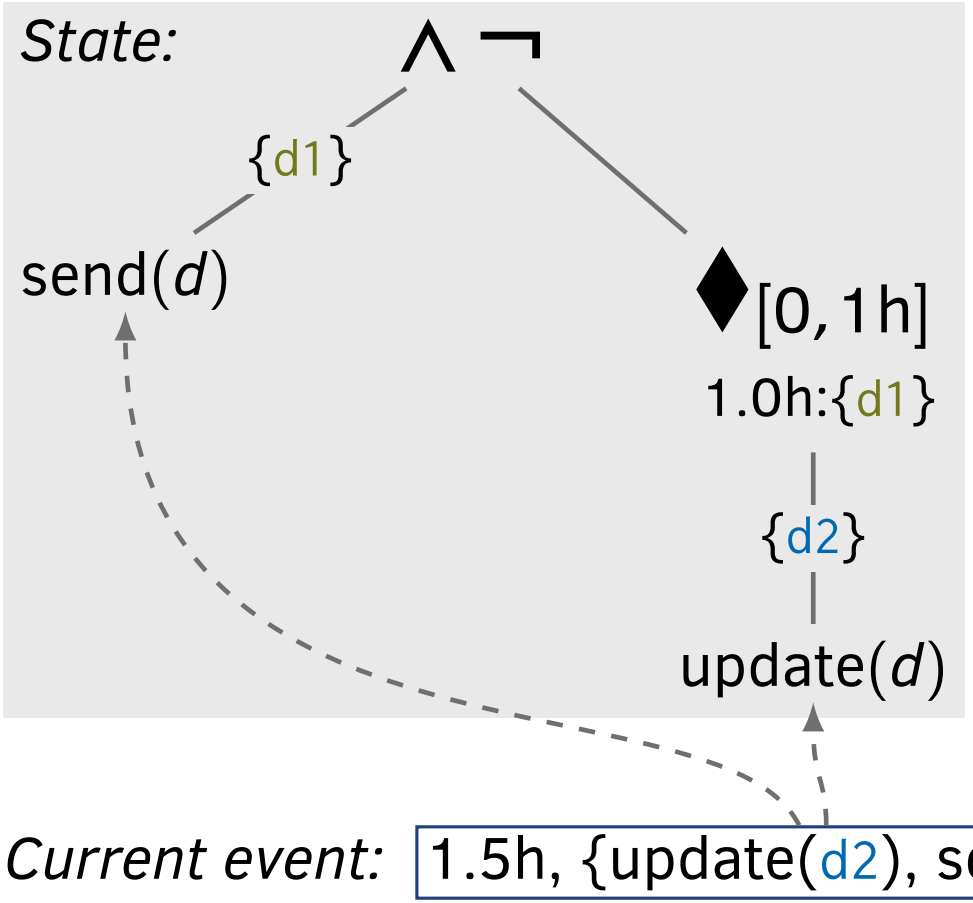
Evaluating $\text{send}(d) \wedge \neg \blacklozenge_{[0,1h]} \text{update}(d)$ to find violations:

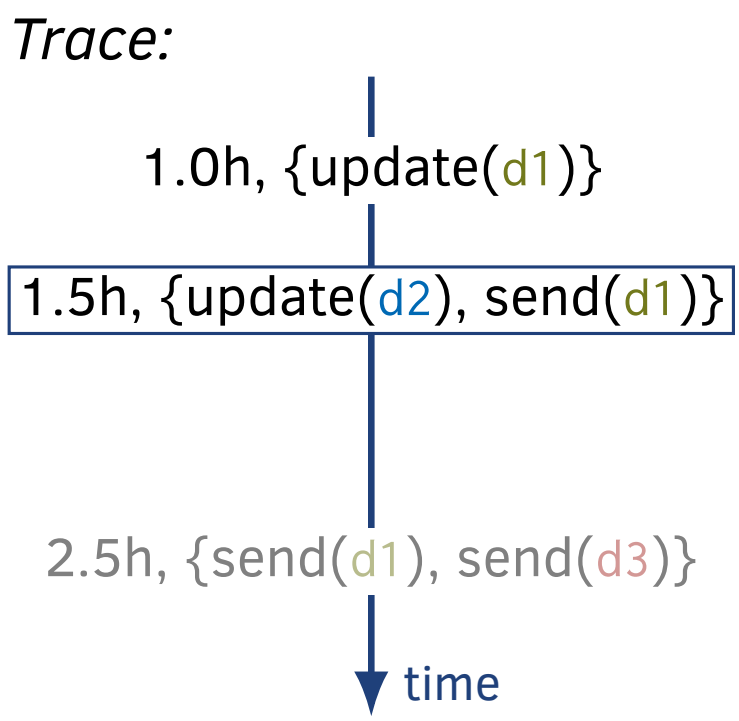




MonPoly's Monitor State

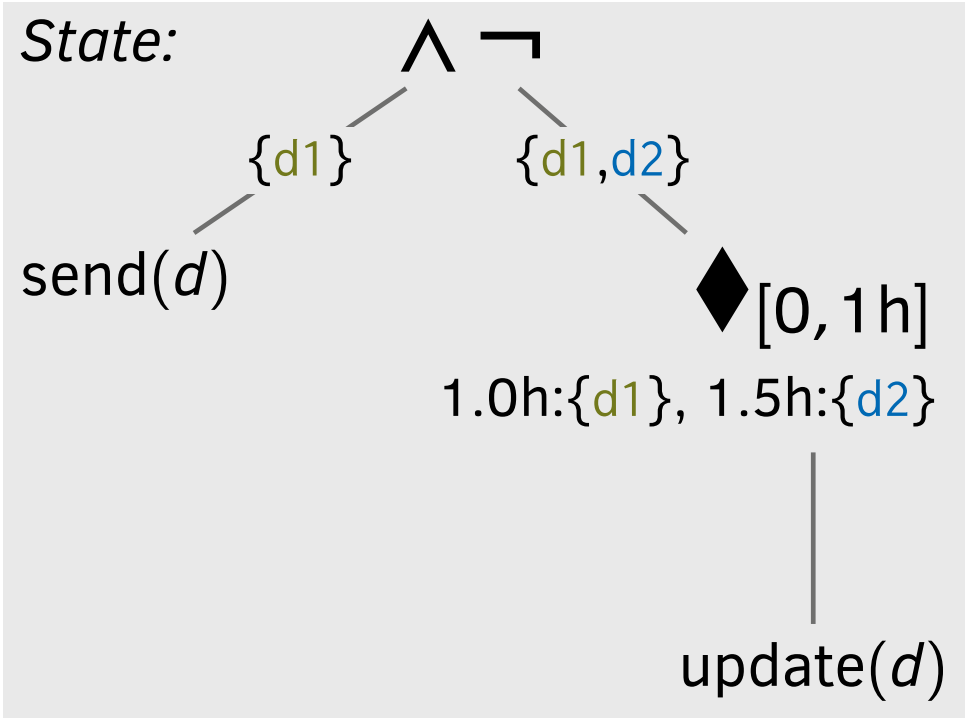
Evaluating $\text{send}(d) \wedge \neg \blacklozenge_{[0,1h]} \text{update}(d)$ to find violations:

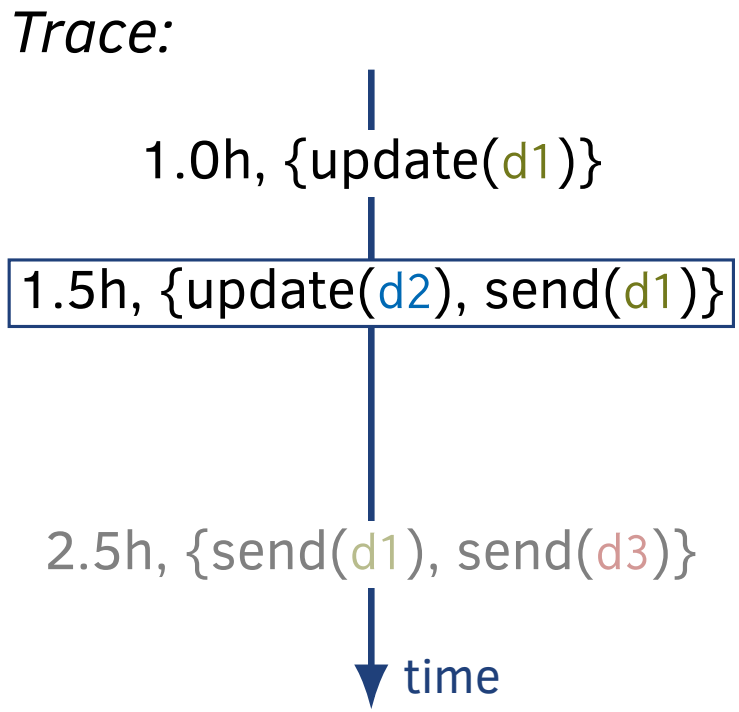




MonPoly's Monitor State

Evaluating $\text{send}(d) \wedge \neg \blacklozenge_{[0,1h]} \text{update}(d)$ to find violations:

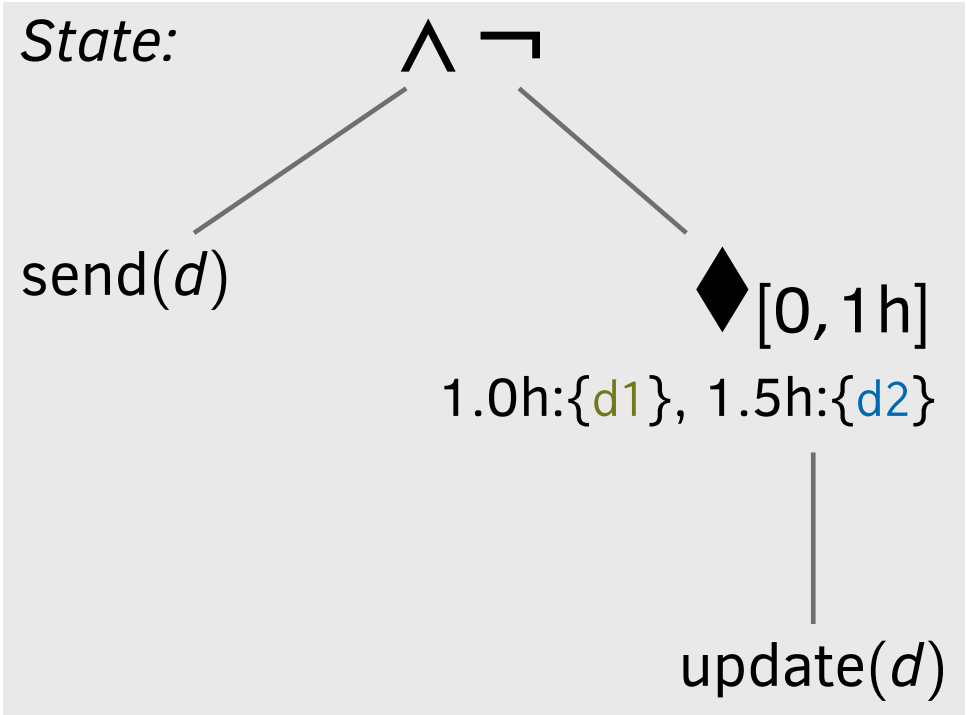


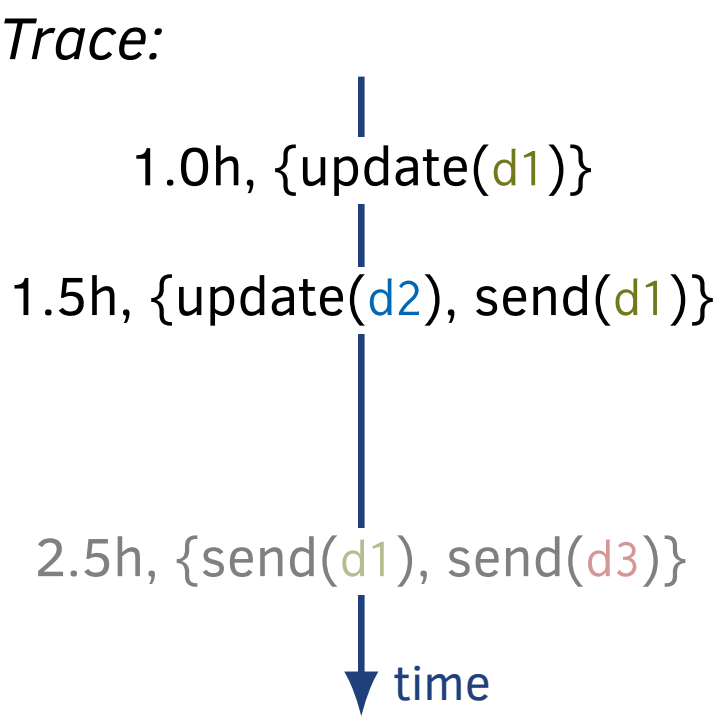


Current event: 1.5h, {update(d2), send(d1)}

MonPoly's Monitor State

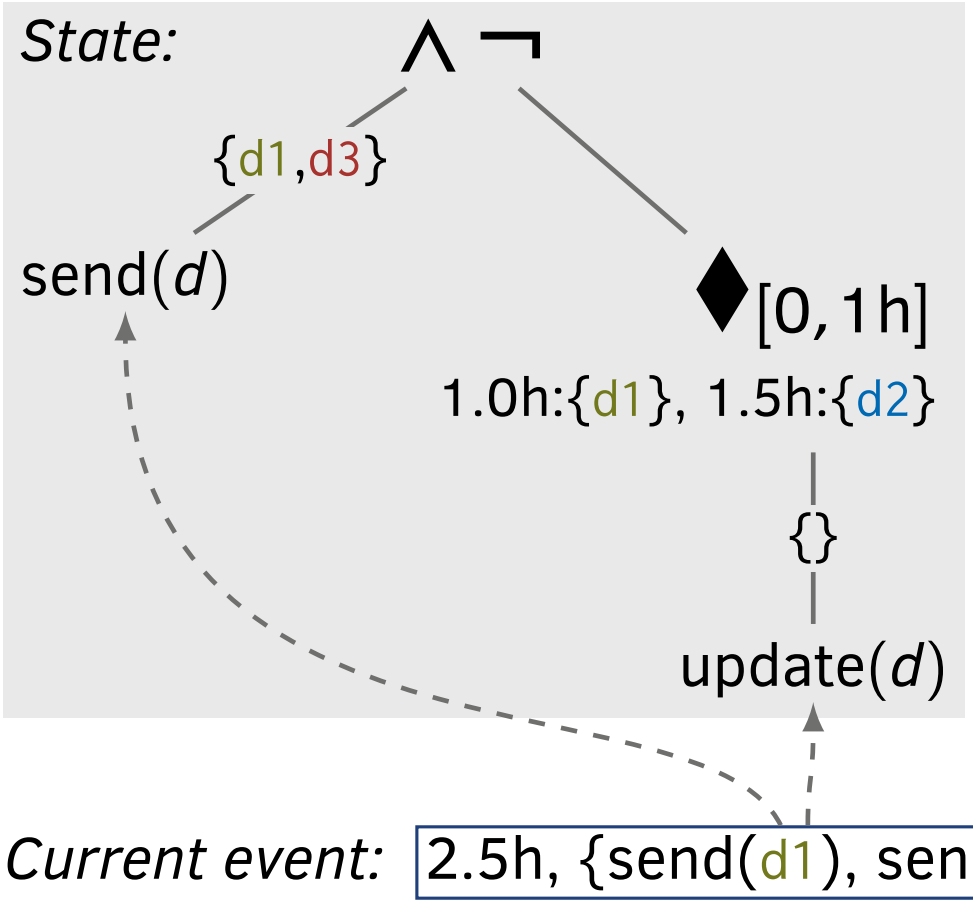
Evaluating $\text{send}(d) \wedge \neg \blacklozenge_{[0,1h]} \text{update}(d)$ to find violations:

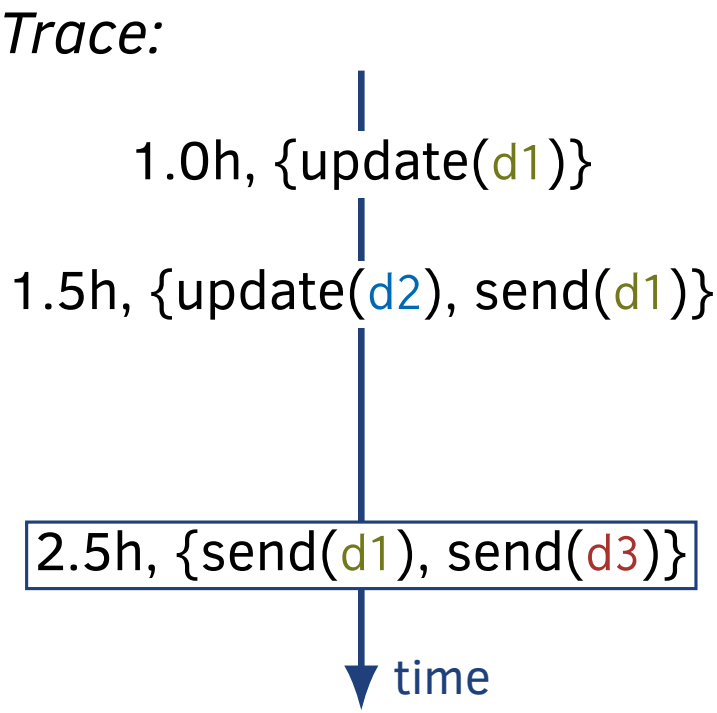




MonPoly's Monitor State

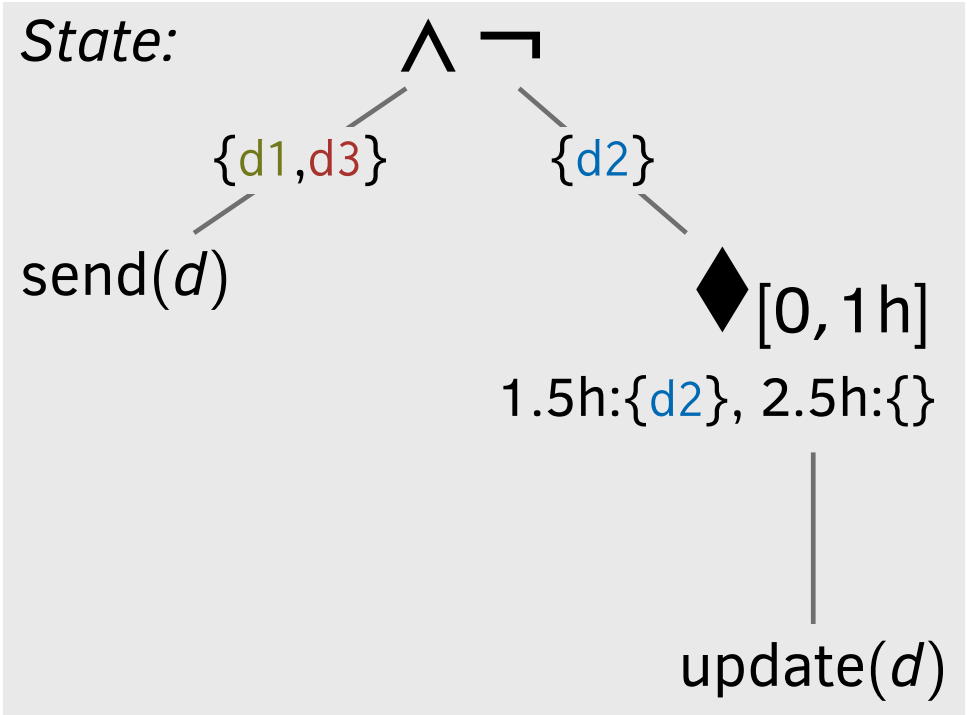
Evaluating $\text{send}(d) \wedge \neg \blacklozenge_{[0,1h]} \text{update}(d)$ to find violations:

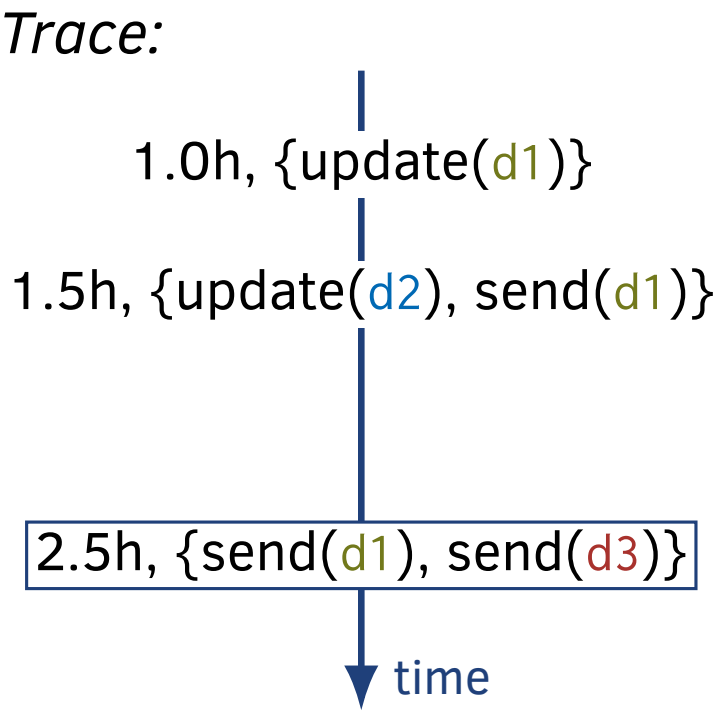




MonPoly's Monitor State

Evaluating $\text{send}(d) \wedge \neg \blacklozenge_{[0,1h]} \text{update}(d)$ to find violations:

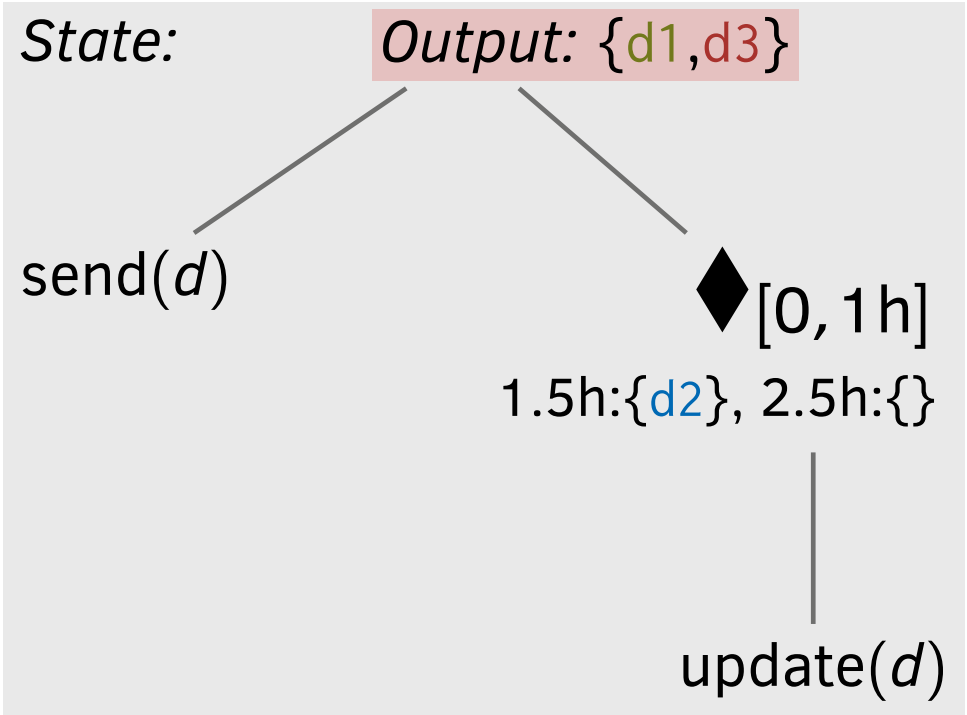


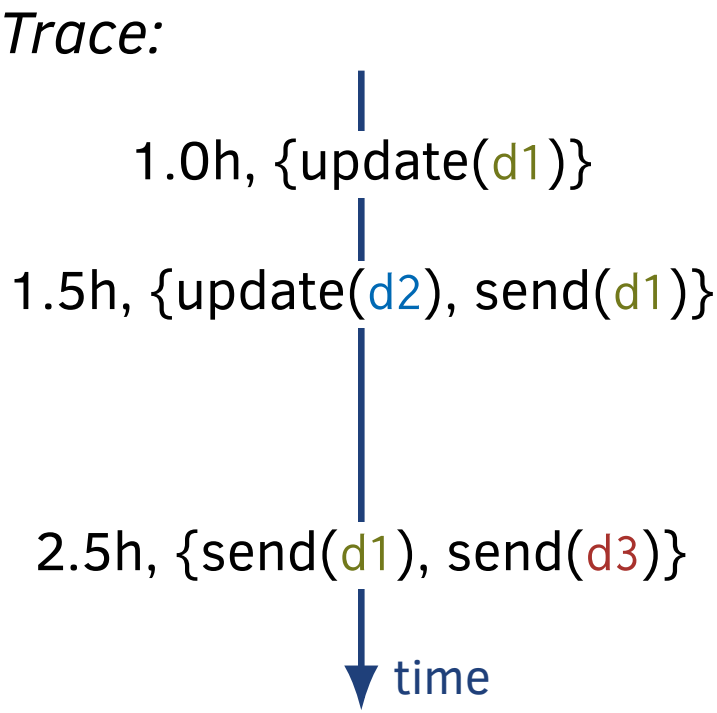


Current event: 2.5h, {send(d1), send(d3))}

MonPoly's Monitor State

Evaluating $\text{send}(d) \wedge \neg \blacklozenge_{[0,1h]} \text{update}(d)$ to find violations:





Split & Merge for MonPoly

Observations:

- The shape of the state tree remains constant.

Split & Merge for MonPoly

Observations:

- The shape of the state tree remains constant.
- The lengths of the lists in the state depend only on time (unaffected by slicing).

Split & Merge for MonPoly

Observations:

- The shape of the state tree remains constant.
- The lengths of the lists in the state depend only on time (unaffected by slicing).
- These lists contains tables, assigning values to variables
  ⇒ **must be correct with respect to restriction!**

Split & Merge for MonPoly

Observations:

- The shape of the state tree remains constant.
- The lengths of the lists in the state depend only on time (unaffected by slicing).
- These lists contains tables, assigning values to variables
  ⇒ **must be correct with respect to restriction!**

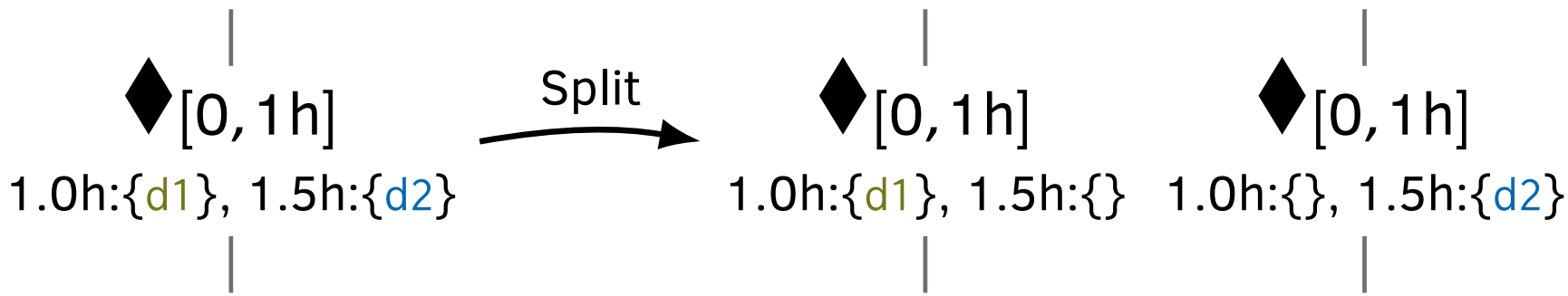
Split: apply the slicing operation $\_ \downarrow (r_{i-1}[k] \cap r_i[k'])$ pointwise
Merge: take the pointwise union of the tables

Split & Merge for MonPoly

Observations:

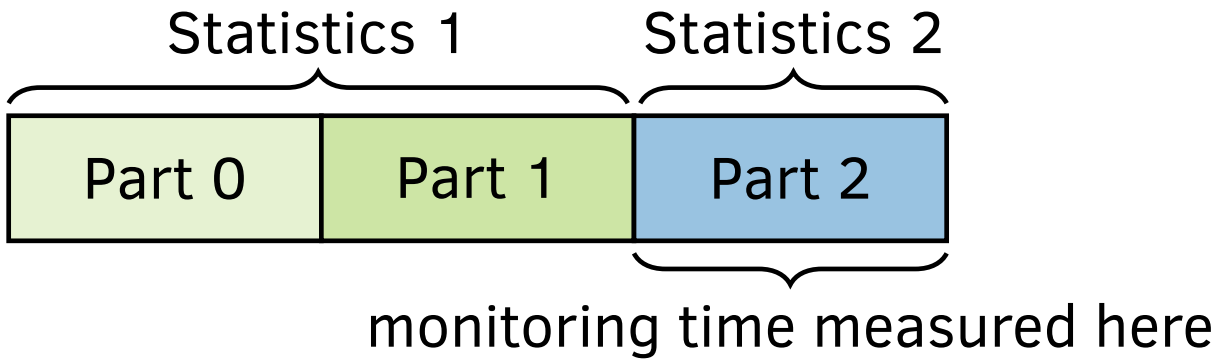
- The shape of the state tree remains constant.
- The lengths of the lists in the state depend only on time (unaffected by slicing).
- These lists contains tables, assigning values to variables
  ⇒ **must be correct with respect to restriction!**

Split: apply the slicing operation $\_ \downarrow (r_{i-1}[k] \cap r_i[k'])$ pointwise
Merge: take the pointwise union of the tables



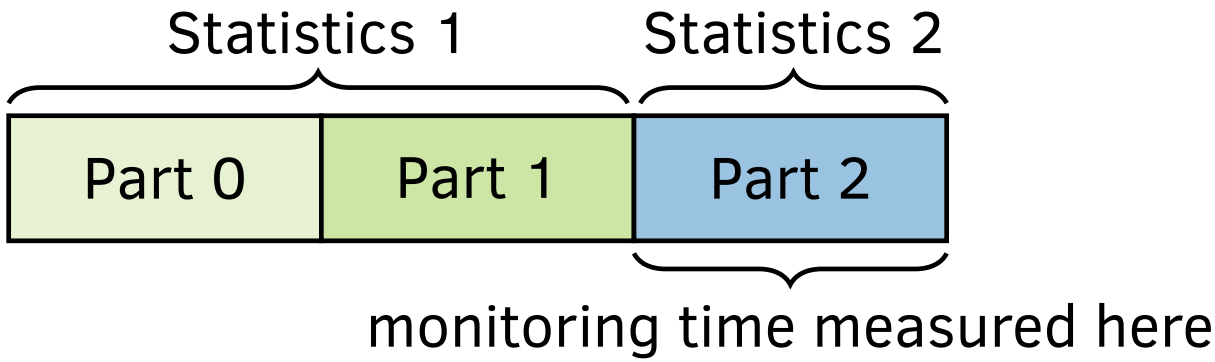
Evaluation

Synthesized stream fragments according to given statistics:



Evaluation

Synthesized stream fragments according to given statistics:

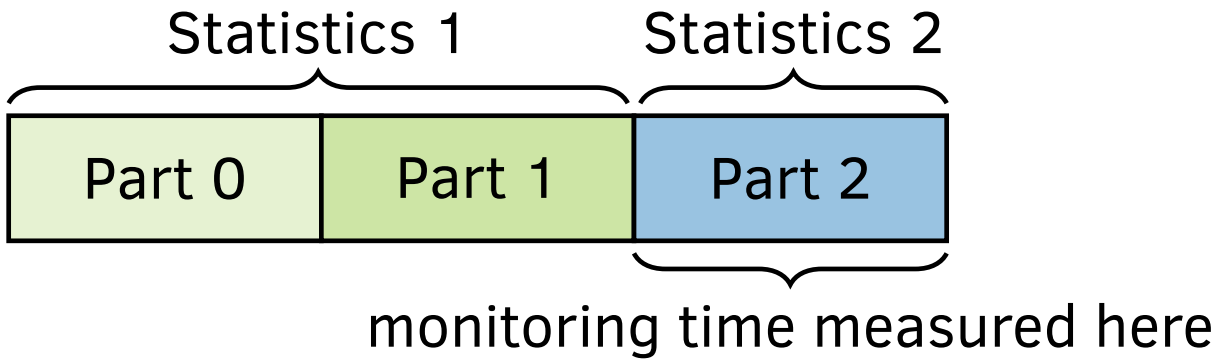


Compared two runs:

- **non-adaptive:** use single strategy optimized for Statistics 1
- **adaptive:** use strategies optimized for each part

Evaluation

Synthesized stream fragments according to given statistics:



Compared two runs:

- **non-adaptive:** use single strategy optimized for Statistics 1
- **adaptive:** use strategies optimized for each part

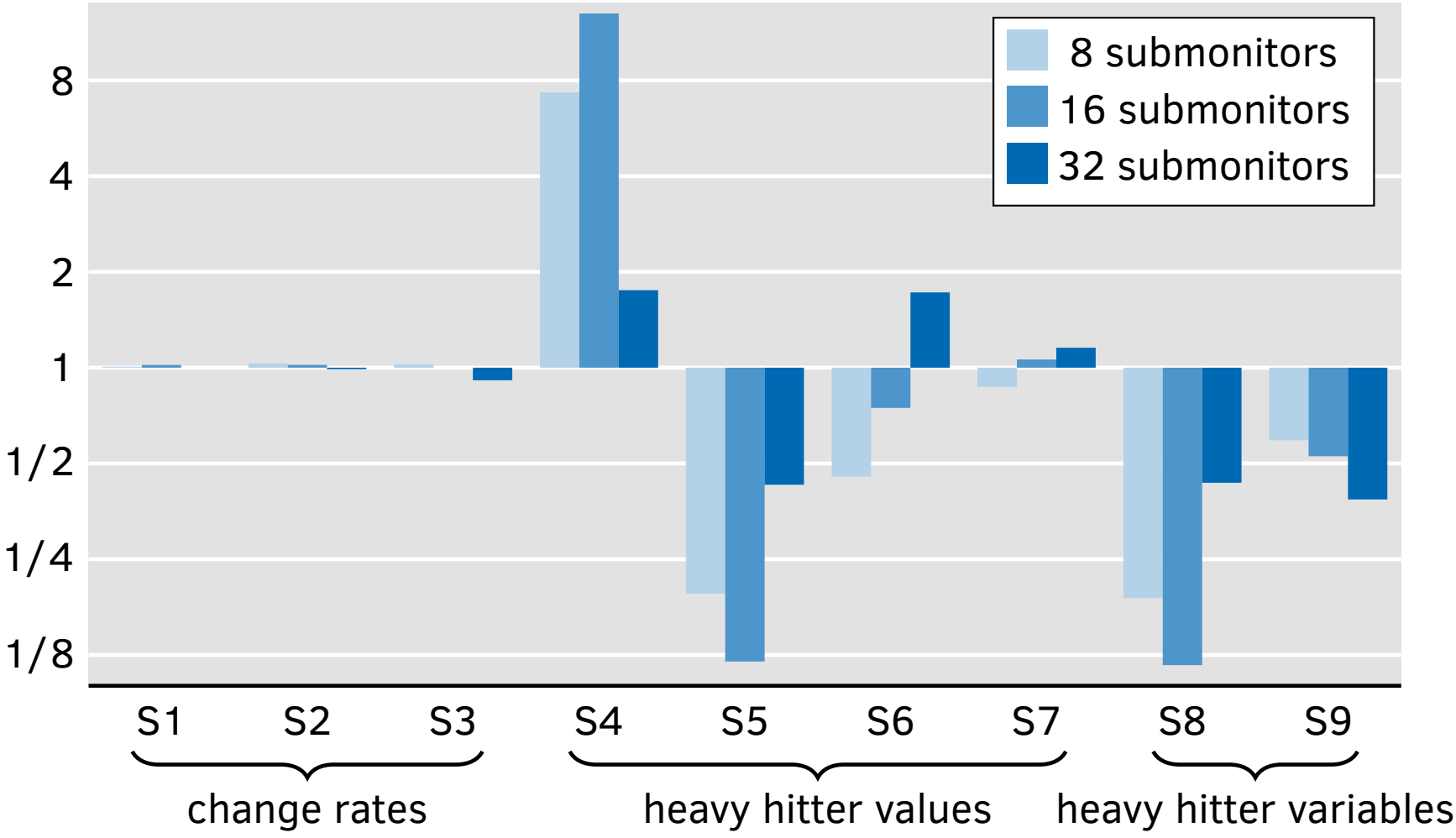
Parameters:

- 3 formulas with different variable patterns
- 9 pairs of statistics (change rates, add/remove heavy hitters)
- 8, 16, and 32 submonitors

Results (1)

Star formula: $\left(\left(\blacklozenge_{[0,10\,\mathrm{s})}\, P(\boldsymbol{a}, b)\right) \wedge Q(\boldsymbol{a}, c)\right) \wedge \Diamond_{[0,10\,\mathrm{s})}\, R(\boldsymbol{a}, d)$

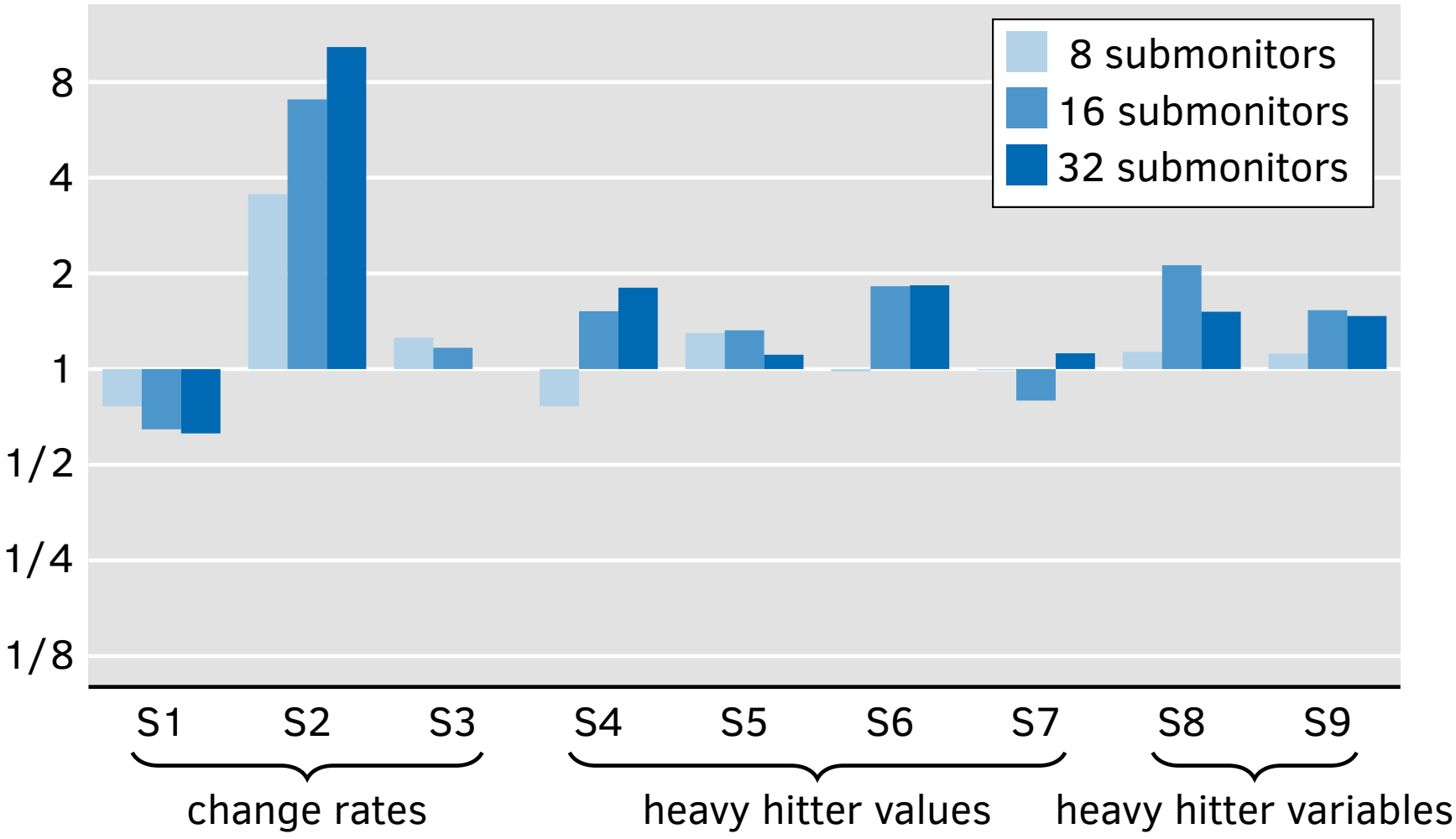
Speedup (log.) – higher is better



Results (2)

Triangle formula: $\big((\blacklozenge_{[0,10\,\mathrm{s})}\, P(a, b)) \wedge Q(b, c)\big) \wedge \lozenge_{[0,10\,\mathrm{s})}\, R(c, a)$

Speedup (log.) – higher is better



Outlook

Steps towards **when** to adapt & **what** strategy to use
[thanks to Christian Fania]:

- Monitor stream statistics
- Frequently optimize new candidate strategies
- Adapt whenever the expected runtime improvement exceeds the empirical cost of adaptation

Outlook

Steps towards **when** to adapt & **what** strategy to use
[thanks to Christian Fania]:

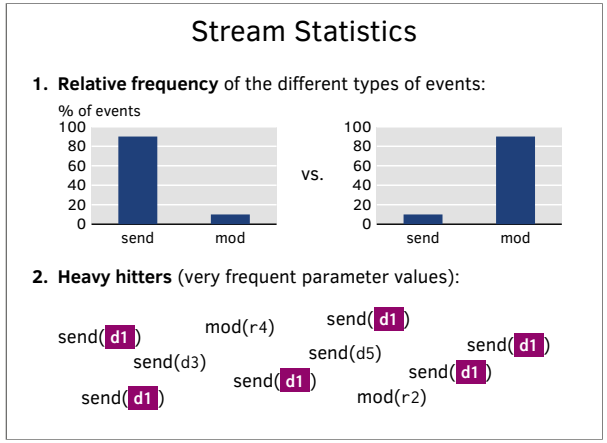
- Monitor stream statistics
- Frequently optimize new candidate strategies
- Adapt whenever the expected runtime improvement exceeds the empirical cost of adaptation

Future work:

- Taking state imbalance into account
- Prediction of future statistics
- Concurrent adaptation

Adaptive Online First-Order Monitoring

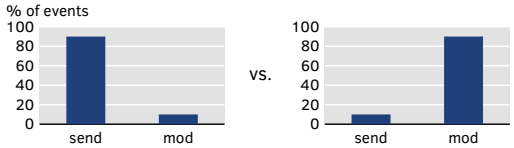
Adaptive Online First-Order Monitoring



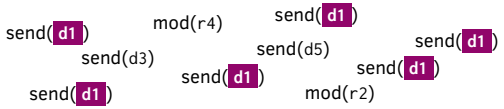
Adaptive Online First-Order Monitoring

Stream Statistics

1. **Relative frequency** of the different types of events:



2. **Heavy hitters** (very frequent parameter values):

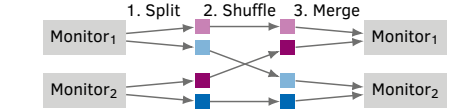


Solution: Adaptive Algorithm

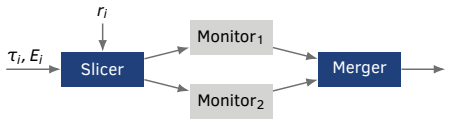
Parameter: sequence $(r_i)_{i \geq 0}$ of slicing strategies

for the i-th input $\langle \tau_i, E_i \rangle$, $i \geq 1$, **do**

if $r_i \neq r_{i-1}$ **then**



end

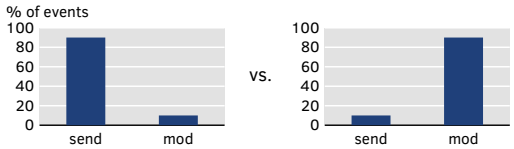


end

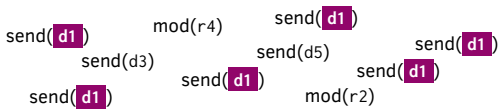
Adaptive Online First-Order Monitoring

Stream Statistics

1. **Relative frequency** of the different types of events:



2. **Heavy hitters** (very frequent parameter values):

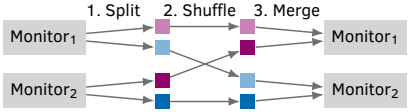


Solution: Adaptive Algorithm

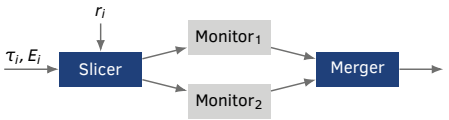
Parameter: sequence $(r_i)_{i\geq 0}$ of slicing strategies

for the i-th input $\langle \tau_i, E_i \rangle$, $i \geq 1$, **do**

if $r_i \neq r_{i-1}$ **then**



end



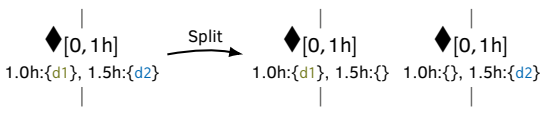
end

Split & Merge for MonPoly

Observations:

- The shape of the state tree remains constant.
- The lengths of the lists in the state depend only on time (unaffected by slicing).
- These lists contains tables, assigning values to variables
  ⇒ **must be correct with respect to restriction!**

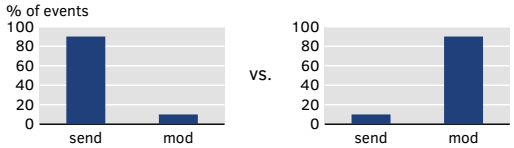
Split: apply the slicing operation $\_ \downarrow (r_{i-1}[k] \cap r_i[k'])$ pointwise
Merge: take the pointwise union of the tables



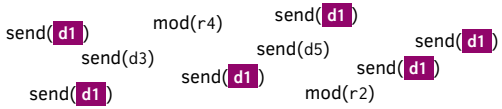
Adaptive Online First-Order Monitoring

Stream Statistics

1. **Relative frequency** of the different types of events:



2. **Heavy hitters** (very frequent parameter values):

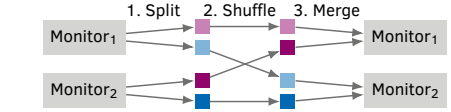


Solution: Adaptive Algorithm

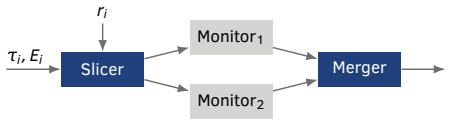
Parameter: sequence $(r_i)_{i\geq 0}$ of slicing strategies

for the i-th input $\langle \tau_i, E_i \rangle$, $i \geq 1$, **do**

if $r_i \neq r_{i-1}$ **then**



end



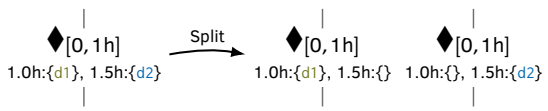
end

Split & Merge for MonPoly

Observations:

- The shape of the state tree remains constant.
- The lengths of the lists in the state depend only on time (unaffected by slicing).
- These lists contains tables, assigning values to variables
  ⇒ **must be correct with respect to restriction!**

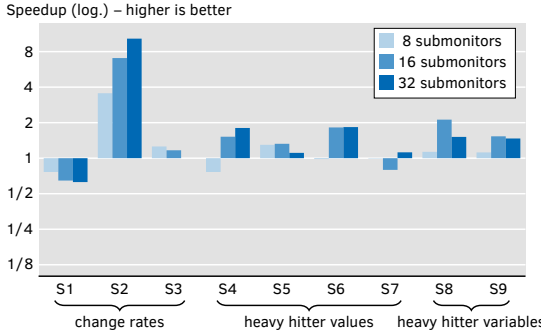
Split: apply the slicing operation $\_\downarrow (r_{i-1}[k] \cap r_i[k'])$ pointwise
Merge: take the pointwise union of the tables



Results (2)

Triangle formula: $((\blacklozenge_{[0,10\,s)}\, P(a, b)) \wedge Q(b, c)) \wedge \Diamond_{[0,10\,s)}\, R(c, a)$

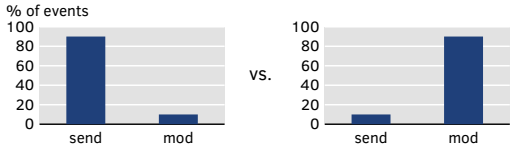
Speedup (log.) – higher is better



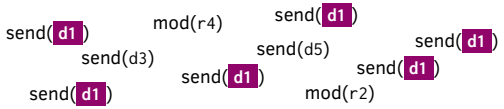
Adaptive Online First-Order Monitoring

Stream Statistics

1. **Relative frequency** of the different types of events:



2. **Heavy hitters** (very frequent parameter values):

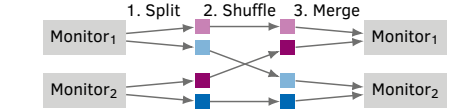


Solution: Adaptive Algorithm

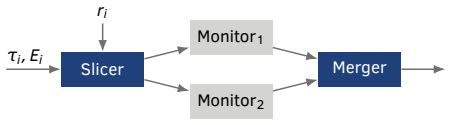
Parameter: sequence $(r_i)_{i \geq 0}$ of slicing strategies

for the i-th input $\langle \tau_i, E_i \rangle$, $i \geq 1$, **do**

if $r_i \neq r_{i-1}$ **then**



end



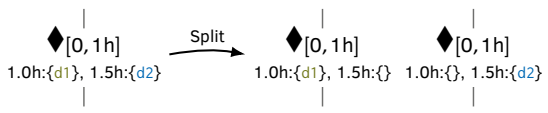
end

Split & Merge for MonPoly

Observations:

- The shape of the state tree remains constant.
- The lengths of the lists in the state depend only on time (unaffected by slicing).
- These lists contains tables, assigning values to variables
  ⇒ **must be correct with respect to restriction!**

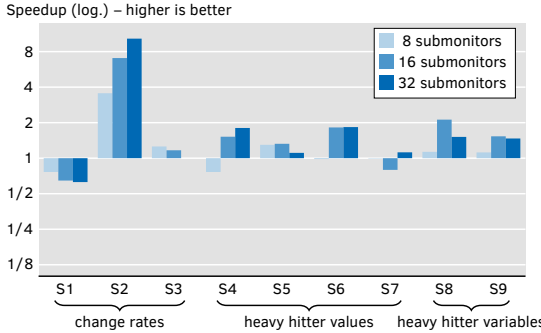
Split: apply the slicing operation $\_ \downarrow (r_{i-1}[k] \cap r_i[k'])$ pointwise
Merge: take the pointwise union of the tables



Results (2)

Triangle formula: $((\blacklozenge_{[0,10\,s)}\, P(a, b)) \wedge Q(b, c)) \wedge \lozenge_{[0,10\,s)}\, R(c, a)$

Speedup (log.) – higher is better



Joshua Schneider David Basin Frederik Brix
Srđan Krstić Dmitriy Traytel

ETH zürich

References

D. Basin et al.: *Monitoring metric first-order temporal properties*.
Journal of the ACM 62(2), 15:1–15:45 (2015).

D. Basin et al.: *Scalable offline monitoring of temporal specifications*.
Formal Methods in System Design 49(1-2), 75–108 (2016).

D. Basin, F. Klaedtke, E. Zalinescu: *The MonPoly monitoring tool*.
In: RV-CuBES 2017, pp. 19-28 (2017).

J. Schneider et al.: *Scalable online first-order monitoring*.
In: RV 2018, pp. 353–371 (2018).

Event rate examples:

D. Bristot de Oliveira, T. Cucinotta, R. Silva de Oliveira: *Efficient formal verification for the Linux Kernel*. In: SEFM 2019, pp. 315–332 (2019).

`https://medium.com/netflix-techblog/evolution-of-the-netflix-data-pipeline-da246ca36905`

`https://developer.arm.com/tools-and-software/embedded/debug-probes/dstream-family/dstream-pt`