Verifying Concurrent Programs using Contracts

Ricardo J. Dias, Carla Ferreira, Jan Fiedor, João M. Lourenço, Aleš Smrčka, Diogo G. Sousa, Tomáš Vojnar

Brno University of Technology (BUT)

Universidade Nova de Lisboa (UNL)

ICST, March 15, 2017

Outline

Contracts

Contract

Consists of a pre- and post-condition of a method. When a call of the method satisfies its pre-condition, the post-condition is guaranteed to be satisfied upon return from the method.

Contracts

Contract

Consists of a pre- and post-condition of a method. When a call of the method satisfies its pre-condition, the post-condition is guaranteed to be satisfied upon return from the method.

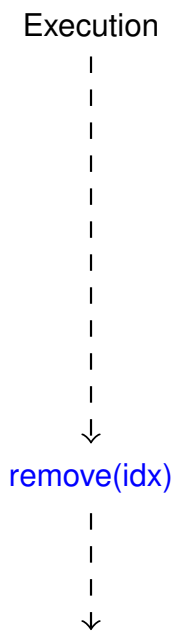
- Example: `java.util.ArrayList` class

Contracts

Contract

Consists of a pre- and post-condition of a method. When a call of the method satisfies its pre-condition, the post-condition is guaranteed to be satisfied upon return from the method.

- Example: `java.util.ArrayList` class
  - Method: `remove(idx)`

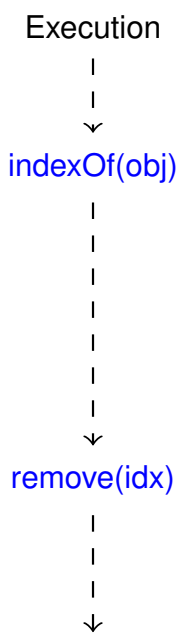


Contracts

Contract

Consists of a pre- and post-condition of a method. When a call of the method satisfies its pre-condition, the post-condition is guaranteed to be satisfied upon return from the method.

- Example: `java.util.ArrayList` class
  - Method: `remove(idx)`

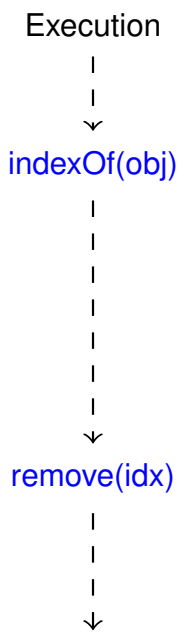


Contracts

Contract

Consists of a pre- and post-condition of a method. When a call of the method satisfies its pre-condition, the post-condition is guaranteed to be satisfied upon return from the method.

- Example: `java.util.ArrayList` class
  - Method: `remove(idx)`
    - Pre-condition: `indexOf(obj)` called
    - Post-condition: `obj` removed

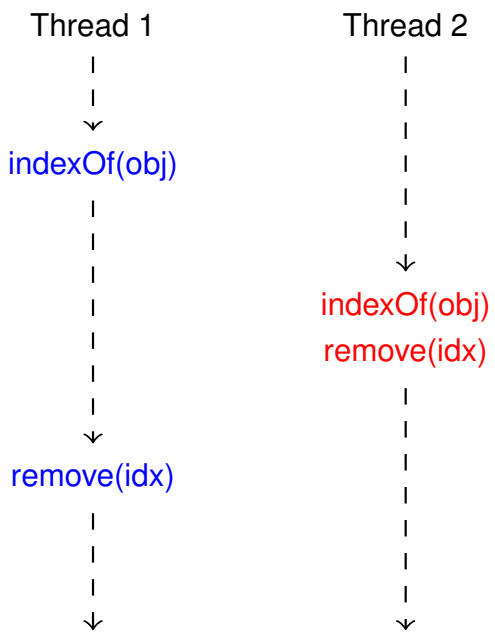


Contracts

Contract

Consists of a pre- and post-condition of a method. When a call of the method satisfies its pre-condition, the post-condition is guaranteed to be satisfied upon return from the method.

- Example: `java.util.ArrayList` class
  - Method: `remove(idx)`
    - Pre-condition: `indexOf(obj)` called
    - Post-condition: `obj` removed

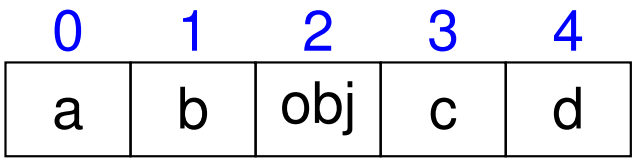


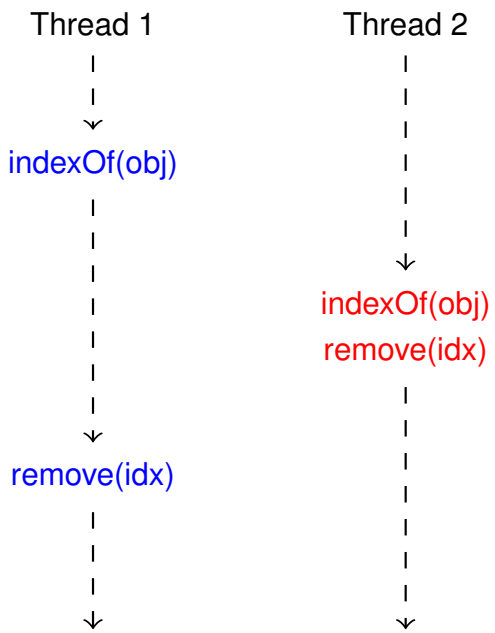
Contracts

Contract

Consists of a pre- and post-condition of a method. When a call of the method satisfies its pre-condition, the post-condition is guaranteed to be satisfied upon return from the method.

- Example: `java.util.ArrayList` class
  - Method: `remove(idx)`
    - Pre-condition: `indexOf(obj)` called
    - Post-condition: `obj` removed





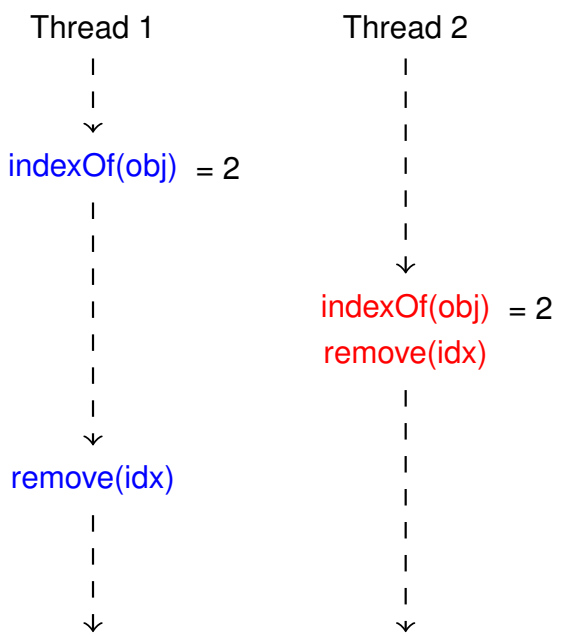
Contracts

Contract

Consists of a pre- and post-condition of a method. When a call of the method satisfies its pre-condition, the post-condition is guaranteed to be satisfied upon return from the method.

- Example: `java.util.ArrayList` class
  - Method: `remove(idx)`
    - Pre-condition: `indexOf(obj)` called
    - Post-condition: `obj` removed

0	1	2	3	4
a	b	obj	c	d

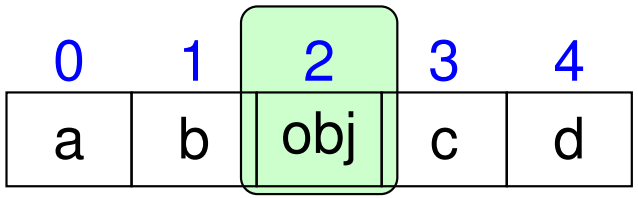


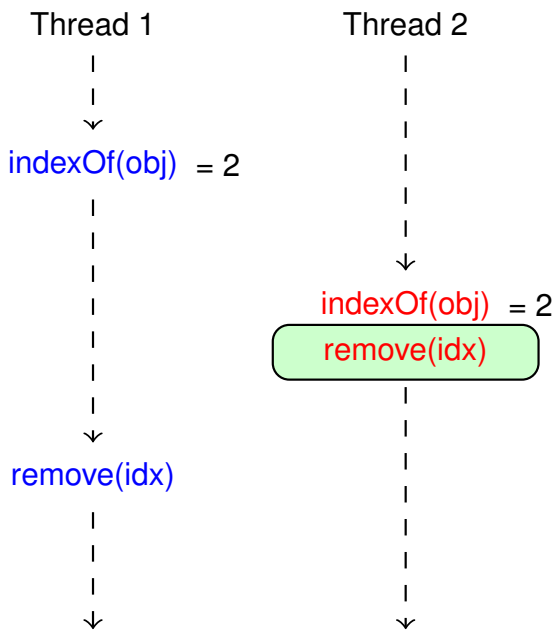
Contracts

Contract

Consists of a pre- and post-condition of a method. When a call of the method satisfies its pre-condition, the post-condition is guaranteed to be satisfied upon return from the method.

- Example: `java.util.ArrayList` class
  - Method: `remove(idx)`
    - Pre-condition: `indexOf(obj)` called
    - Post-condition: `obj` removed



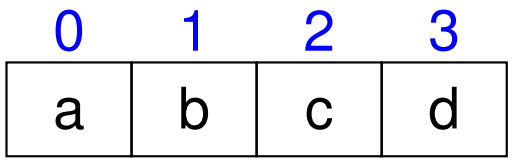


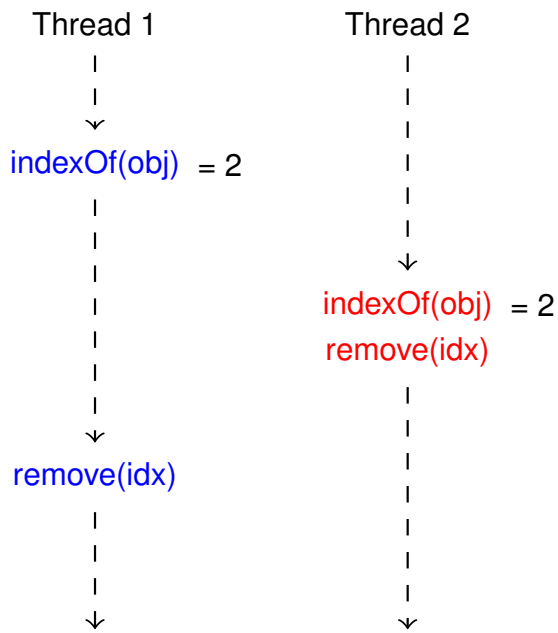
Contracts

Contract

Consists of a pre- and post-condition of a method. When a call of the method satisfies its pre-condition, the post-condition is guaranteed to be satisfied upon return from the method.

- Example: `java.util.ArrayList` class
  - Method: `remove(idx)`
    - Pre-condition: `indexOf(obj)` called
    - Post-condition: `obj` removed



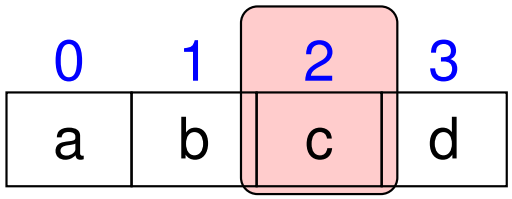


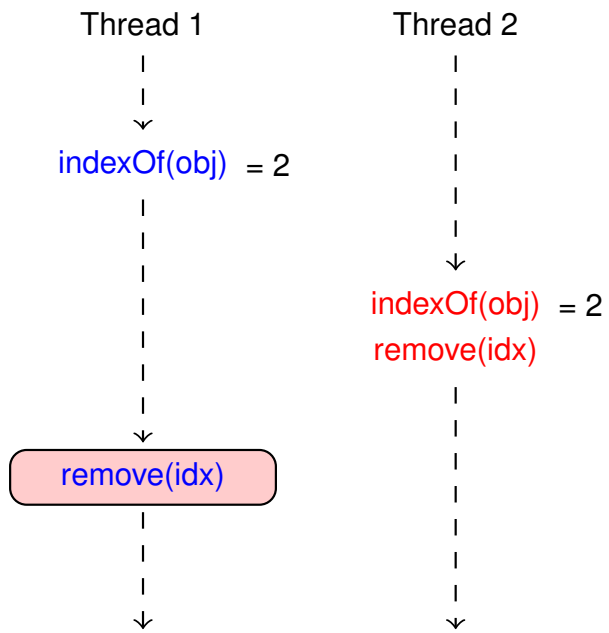
Contracts

Contract

Consists of a pre- and post-condition of a method. When a call of the method satisfies its pre-condition, the post-condition is guaranteed to be satisfied upon return from the method.

- Example: `java.util.ArrayList` class
  - Method: `remove(idx)`
    - Pre-condition: `indexOf(obj)` called
    - Post-condition: `obj` removed



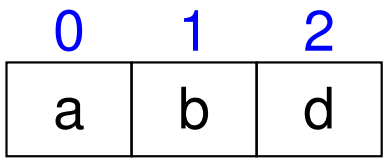


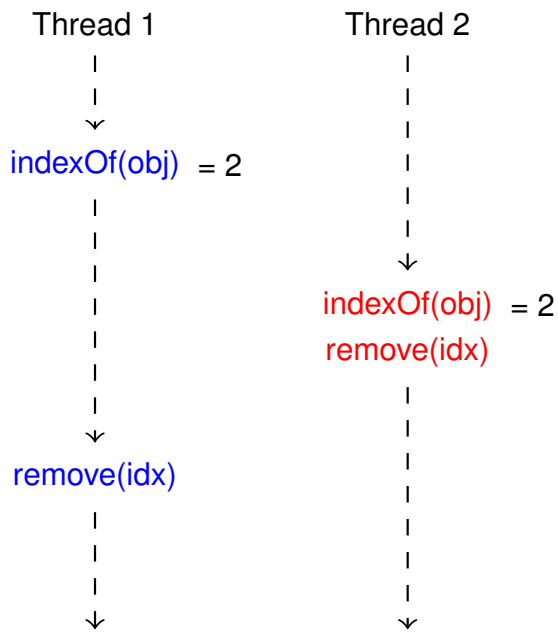
Contracts

Contract

Consists of a pre- and post-condition of a method. When a call of the method satisfies its pre-condition, the post-condition is guaranteed to be satisfied upon return from the method.

- Example: `java.util.ArrayList` class
  - Method: `remove(idx)`
    - Pre-condition: `indexOf(obj)` called
    - Post-condition: `obj` removed





Contracts for Concurrency

Contract (in concurrent setting)

A sequence of method calls which must be executed atomically.

Definition

Let $\Sigma_{\mathbb{M}}$ be a set of all public method names (the API) of a software module (or library). A *contract* is a set $\mathbb{R}$ of *clauses* where each clause $\varrho \in \mathbb{R}$ is a regular expression over $\Sigma_{\mathbb{M}}$. A contract violation occurs if any of the sequences represented by the contract clauses is interleaved with an execution of methods from $\Sigma_{\mathbb{M}}$.

Contract for the `java.util.ArrayList` class

(ϱ_1) `contains indexOf`
(ϱ_2) `indexOf ( set | remove | get )`
(ϱ_3) `size ( remove | set | get )`
(ϱ_4) `add ( get | indexOf )`

Contracts for Concurrency

Contract (in concurrent setting)

A sequence of method calls which must be executed atomically.

Definition

Let $\Sigma_{\mathbb{M}}$ be a set of all public method names (the API) of a software module (or library). A *contract* is a set $\mathbb{R}$ of *clauses* where each clause $\varrho \in \mathbb{R}$ is a regular expression over $\Sigma_{\mathbb{M}}$. A contract violation occurs if any of the sequences represented by the contract clauses is interleaved with an execution of methods from $\Sigma_{\mathbb{M}}$.

Contract for the `java.util.ArrayList` class

(ϱ_1) `contains indexOf`
(ϱ_2) `indexOf ( set | remove | get )`
(ϱ_3) `size ( remove | set | get )`
(ϱ_4) `add ( get | indexOf )`

Contracts for Concurrency

Contract (in concurrent setting)

A sequence of method calls which must be executed atomically.

Definition

Let $\Sigma_{\mathbb{M}}$ be a set of all public method names (the API) of a software module (or library). A *contract* is a set $\mathbb{R}$ of *clauses* where each clause $\varrho \in \mathbb{R}$ is a regular expression over $\Sigma_{\mathbb{M}}$. A contract violation occurs if any of the sequences represented by the contract clauses is interleaved with an execution of methods from $\Sigma_{\mathbb{M}}$.

Contract for the `java.util.ArrayList` class

(ϱ_1) `contains indexOf`
(ϱ_2) `indexOf ( set | remove | get )`
(ϱ_3) `size ( remove | set | get )`
(ϱ_4) `add ( get | indexOf )`

Extending Contracts with Parameters

Extending Contracts with Parameters

Motivation

Extending Contracts with Parameters

Motivation

```
void replace(Object a, Object b) {
  if (array.contains(a)) {
    int idx=array.indexOf(a);
    array.set(idx,b);
  }
}
```

Extending Contracts with Parameters

Motivation

```
void replace(Object a, Object b) {
  if (array.contains(a)) {
    int idx=array.indexOf(a);
    array.set(idx,b);
  }
}
```

Extending Contracts with Parameters

Motivation

```
void replace(Object a, Object b) {
  if (array.contains(a)) {
    int idx=array.indexOf(a);
    array.set(idx,b);
  }
}
```

Extending Contracts with Parameters

Motivation

```
void replace(Object a, Object b) {
  if (array.contains(a)) {
    int idx=array.indexOf(a);
    array.set(idx,b);
  }
}
```

Extending Contracts with Parameters

Motivation

```
void replace(Object a, Object b) {
  if (array.contains(a)) {
    int idx=array.indexOf(a);
    array.set(idx,b);
  }
}
```

Extending Contracts with Parameters

Motivation

```
void replace(Object a, Object b) {
  if (array.contains(a)) {
    int idx=array.indexOf(a);
    array.set(idx,b);
  }
}
```

```
void erase(Object c) {
  int idx;
  while ((idx=array.indexOf(c)) > -1) {
    array.remove(idx);
  }
}
```

Extending Contracts with Parameters

Motivation

```
void atomic replace(Object a, Object b) {
  if (array.contains(a)) {
    int idx=array.indexOf(a);
    array.set(idx,b);
  }
}
```

```
void atomic erase(Object c) {
  int idx;
  while ((idx=array.indexOf(c)) > -1) {
    array.remove(idx);
  }
}
```

Extending Contracts with Parameters

Motivation

```
void atomic replace(Object a, Object b) {
  if (array.contains(a)) {
    int idx=array.indexOf(a);
    array.set(idx,b);
  }
}
```

```
void f() {
  Object x,y,z;
  ...
  replace(x,y);
  ...
  erase(z);
}
```

```
void atomic erase(Object c) {
  int idx;
  while ((idx=array.indexOf(c)) > -1) {
    array.remove(idx);
  }
}
```

Extending Contracts with Parameters

Motivation

```
void atomic replace(Object a, Object b) {
  if (array.contains(a)) {
    int idx=array.indexOf(a);
    array.set(idx,b);
  }
}
```

```
void f() {
  Object x,y,z;
  ...
  replace(x,y);
  ...
  erase(z);
}
```

```
void atomic erase(Object c) {
  int idx;
  while ((idx=array.indexOf(c)) > -1) {
    array.remove(idx);
  }
}
```

Extending Contracts with Parameters

Motivation

```
void atomic replace(Object a, Object b) {
  if (array.contains(a)) {
    int idx=array.indexOf(a);
    array.set(idx,b);
  }
}
```

$x \notin array$

```
void f() {
  Object x,y,z;
  ...
  replace(x,y);
  ...
  erase(z);
}
```

```
void atomic erase(Object c) {
  int idx;
  while ((idx=array.indexOf(c)) > -1) {
    array.remove(idx);
  }
}
```

Extending Contracts with Parameters

Motivation

```
void atomic replace(Object a, Object b) {
  if (array.contains(a)) {
    int idx=array.indexOf(a);
    array.set(idx,b);
  }
}
```

$x \notin array$

```
void f() {
  Object x,y,z;
  ...
  replace(x,y);
  ...
  erase(z);
}
```

```
void atomic erase(Object c) {
  int idx;
  while ((idx=array.indexOf(c)) > -1) {
    array.remove(idx);
  }
}
```

Extending Contracts with Parameters

Motivation

```
void atomic replace(Object a, Object b) {
  if (array.contains(a)) {
    int idx=array.indexOf(a);
    array.set(idx,b);
  }
}
```

$x \notin array$

Contract contains indexOf violated!

```
void atomic erase(Object c) {
  int idx;
  while ((idx=array.indexOf(c)) > -1) {
    array.remove(idx);
  }
}
```

```
void f() {
  Object x,y,z;
  ...
  replace(x,y);
  ...
  erase(z);
}
```

Extending Contracts with Parameters

- Allows one to express how the flow of data influences the dependencies between methods
- Contract specification extended by considering
  - Method call parameters
  - Return values
- Expressed as meta-variables

Contract for the `java.util.ArrayList` class

(ϱ'_1) `contains(X)` `indexOf(X)`
(ϱ'_2) `X = indexOf(_)` (`remove(X)` | `set(X,_)` | `get(X)`)
(ϱ'_3) `X = size()` (`remove(X)` | `set(X,_)` | `get(X)`)
(ϱ'_4) `add(X)` (`get(X)` | `indexOf(X)`)

Extending Contracts with Parameters

- Allows one to express how the flow of data influences the dependencies between methods
- Contract specification extended by considering
  - Method call parameters
  - Return values
- Expressed as meta-variables

Contract for the `java.util.ArrayList` class

(ϱ'_1) `contains(X)` `indexOf(X)`
(ϱ'_2) `X` $=$ `indexOf(_)` (`remove(X)` | `set(X,_)` | `get(X)`)
(ϱ'_3) `X` $=$ `size()` (`remove(X)` | `set(X,_)` | `get(X)`)
(ϱ'_4) `add(X)` (`get(X)` | `indexOf(X)`)

Extending Contracts with Spoilers

Extending Contracts with Spoilers

Motivation

Extending Contracts with Spoilers

Motivation

Contract for the `java.util.ArrayList` class

(ϱ_1) `contains indexOf`
(ϱ_2) `indexOf ( set | remove | get )`
(ϱ_3) `size ( remove | set | get )`
(ϱ_4) `add ( get | indexOf )`

Extending Contracts with Spoilers

Motivation

Contract for the `java.util.ArrayList` class

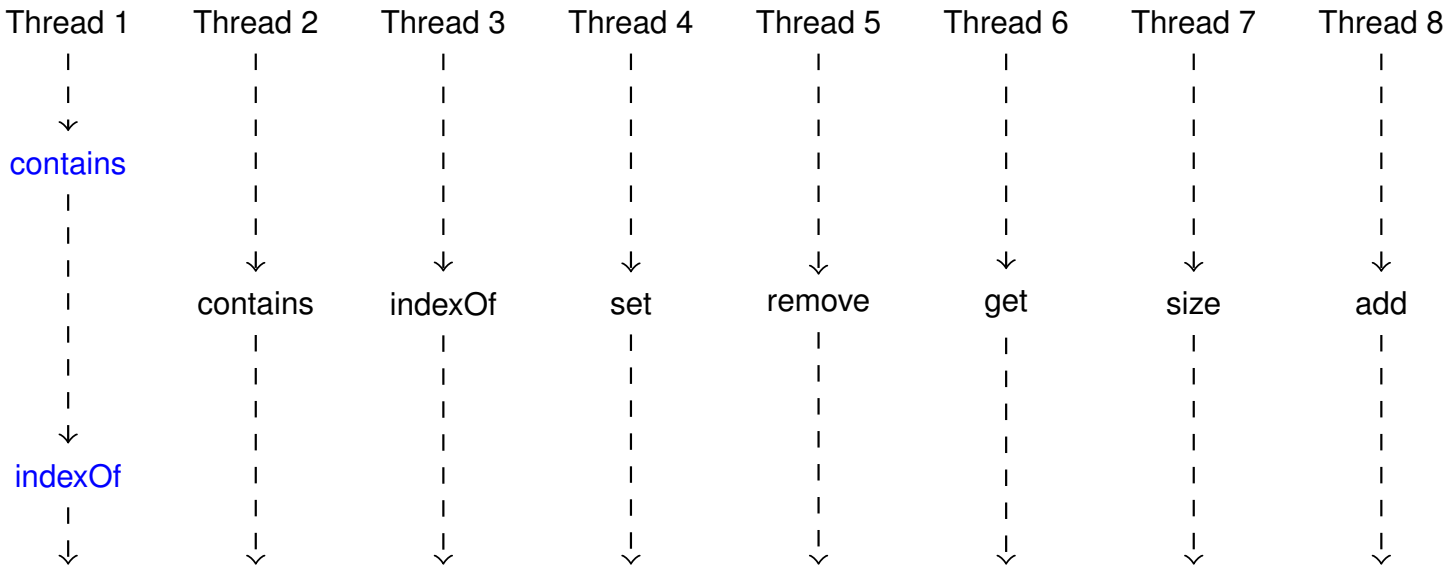
(ϱ_1) `contains indexOf`
(ϱ_2) `indexOf ( set | remove | get )`
(ϱ_3) `size ( remove | set | get )`
(ϱ_4) `add ( get | indexOf )`

Extending Contracts with Spoilers

Motivation

Contract for the `java.util.ArrayList` class

(ϱ_1) `contains indexOf`
(ϱ_2) `indexOf ( set | remove | get )`
(ϱ_3) `size ( remove | set | get )`
(ϱ_4) `add ( get | indexOf )`

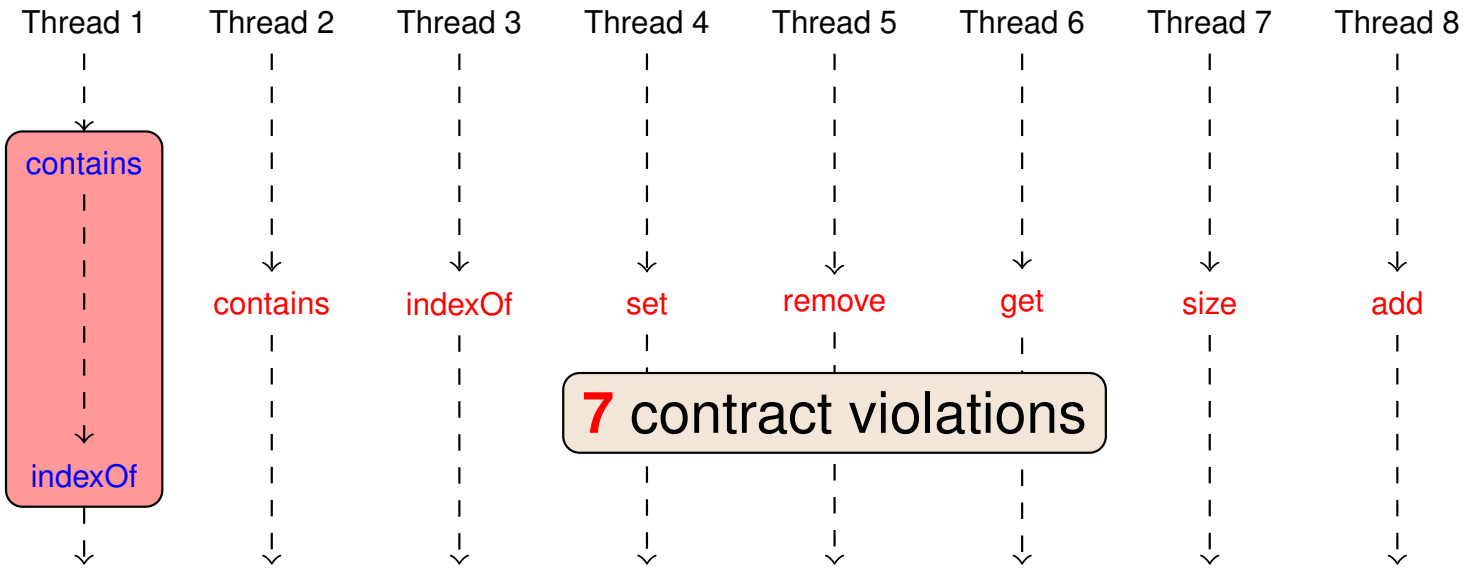


Extending Contracts with Spoilers

Motivation

Contract for the `java.util.ArrayList` class

(ϱ_1) `contains indexOf`
(ϱ_2) `indexOf ( set | remove | get )`
(ϱ_3) `size ( remove | set | get )`
(ϱ_4) `add ( get | indexOf )`



Extending Contracts with Spoilers

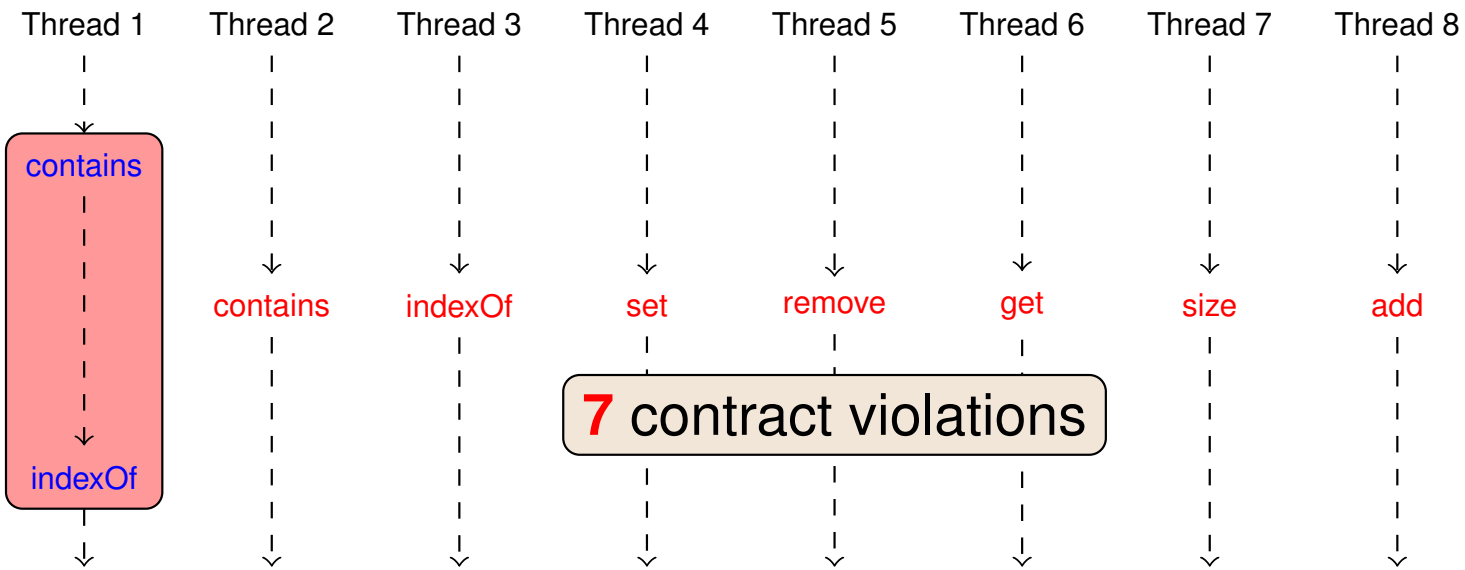
Motivation

Contract for the `java.util.ArrayList` class

(ϱ_1) `contains indexOf`
(ϱ_2) `indexOf ( set | remove | get )`
(ϱ_3) `size ( remove | set | get )`
(ϱ_4) `add ( get | indexOf )`

$\Sigma_{\mathbb{M}}$ (methods)

`contains`
`indexOf`
`set`
`remove`
`get`
`size`
`add`



Extending Contracts with Spoilers

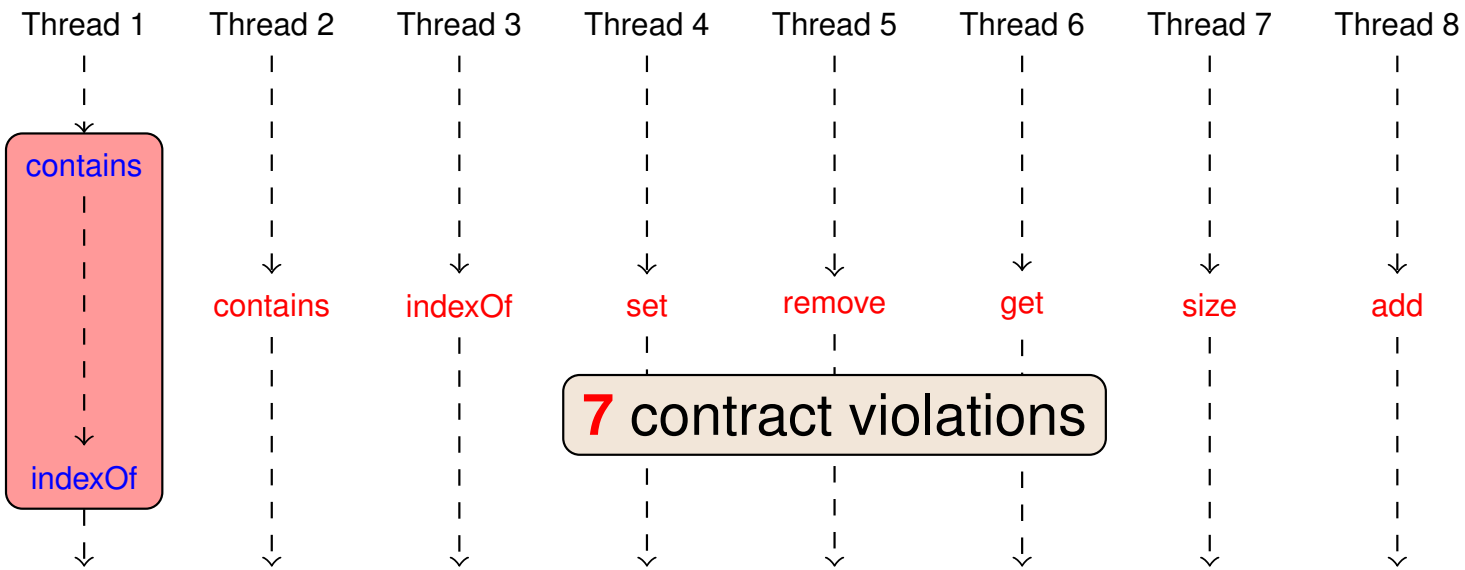
Motivation

Contract for the `java.util.ArrayList` class

(ϱ_1) `contains indexOf`
(ϱ_2) `indexOf ( set | remove | get )`
(ϱ_3) `size ( remove | set | get )`
(ϱ_4) `add ( get | indexOf )`

$\Sigma_{\mathbb{M}}$ (methods)

`contains`
`indexOf`
`set`
`remove`
`get`
`size`
`add`



Extending Contracts with Spoilers

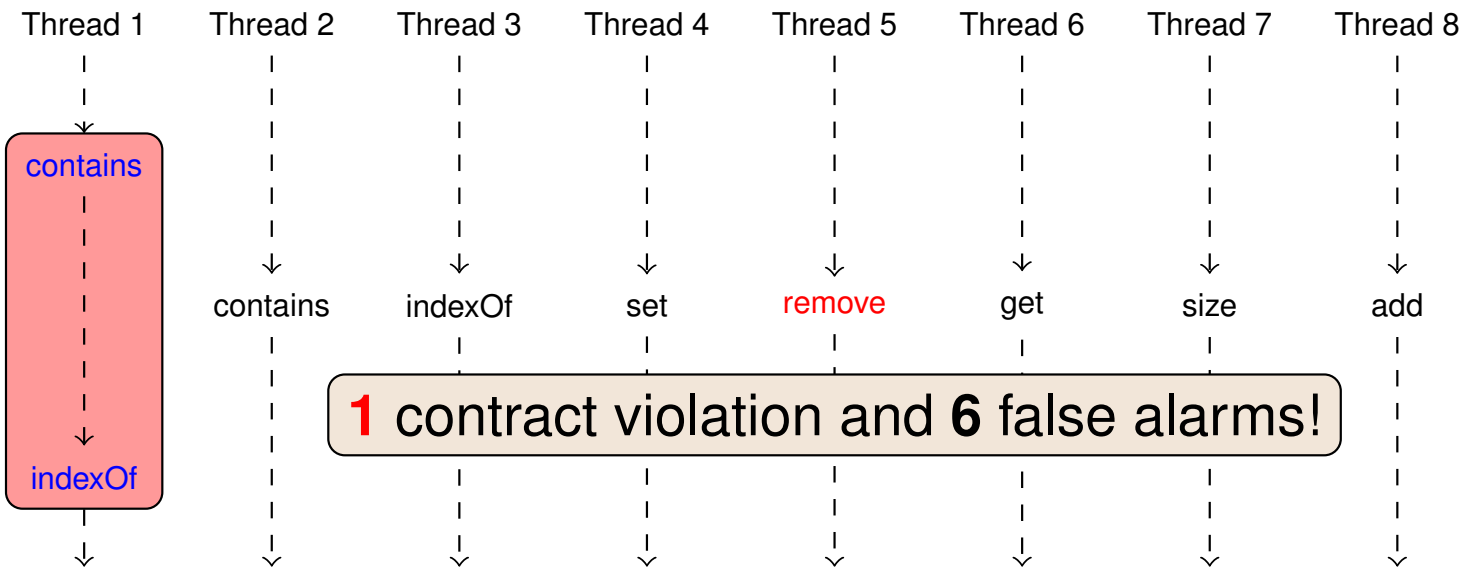
Motivation

Contract for the `java.util.ArrayList` class

(ϱ_1) `contains indexOf`
(ϱ_2) `indexOf ( set | remove | get )`
(ϱ_3) `size ( remove | set | get )`
(ϱ_4) `add ( get | indexOf )`

$\Sigma_{\mathbb{M}}$ (methods)

`contains`
`indexOf`
`set`
`remove`
`get`
`size`
`add`



Extending Contracts with Spoilers

- Allows one to express in which context the contract clauses shall be enforced

Definition

Let $\mathbb{R}$ be the set of *target* clauses where each target $\varrho \in \mathbb{R}$ is a regular expression over $\Sigma_{\mathbb{M}}$. Let $\mathbb{S}$ be the set of *spoilers* where each spoiler $\sigma \in \mathbb{S}$ is a regular expression over $\Sigma_{\mathbb{M}}$. A *contract* is then a relation $\mathbb{C} \subseteq \mathbb{R} \times \mathbb{S}$ which defines for each target the spoilers that may cause atomicity violations.

Contract for the `java.util.ArrayList` class

(ϱ''_1) `contains indexOf` $\nleftrightsquigarrow$ `remove`
(ϱ''_2) `indexOf (remove | set | get)` $\nleftrightsquigarrow$ `remove | add | set`
(ϱ''_3) `size (remove | set | get)` $\nleftrightsquigarrow$ `remove`
(ϱ''_4) `add indexOf` $\nleftrightsquigarrow$ `remove | set`

Extending Contracts with Spoilers

- Allows one to express in which context the contract clauses shall be enforced

Definition

Let $\mathbb{R}$ be the set of *target* clauses where each target $\varrho \in \mathbb{R}$ is a regular expression over $\Sigma_{\mathbb{M}}$. Let $\mathbb{S}$ be the set of *spoilers* where each spoiler $\sigma \in \mathbb{S}$ is a regular expression over $\Sigma_{\mathbb{M}}$. A *contract* is then a relation $\mathbb{C} \subseteq \mathbb{R} \times \mathbb{S}$ which defines for each target the spoilers that may cause atomicity violations.

Contract for the `java.util.ArrayList` class

(ϱ_1'') `contains indexOf` $\nleftrightsquigarrow$ `remove`

(ϱ_2'') `indexOf (remove | set | get)` $\nleftrightsquigarrow$ `remove | add | set`

(ϱ_3'') `size (remove | set | get)` $\nleftrightsquigarrow$ `remove`

(ϱ_4'') `add indexOf` $\nleftrightsquigarrow$ `remove | set`

Extending Contracts with Spoilers

- Allows one to express in which context the contract clauses shall be enforced

Definition

Let $\mathbb{R}$ be the set of *target* clauses where each target $\varrho \in \mathbb{R}$ is a regular expression over $\Sigma_{\mathbb{M}}$. Let $\mathbb{S}$ be the set of *spoilers* where each spoiler $\sigma \in \mathbb{S}$ is a regular expression over $\Sigma_{\mathbb{M}}$. A *contract* is then a relation $\mathbb{C} \subseteq \mathbb{R} \times \mathbb{S}$ which defines for each target the spoilers that may cause atomicity violations.

Contract for the `java.util.ArrayList` class

(ϱ_1'') `contains indexOf` $\nleftrightsquigarrow$ `remove`
(ϱ_2'') `indexOf (remove|set|get)` $\nleftrightsquigarrow$ `remove|add|set`
(ϱ_3'') `size (remove|set|get)` $\nleftrightsquigarrow$ `remove`
(ϱ_4'') `add indexOf` $\nleftrightsquigarrow$ `remove|set`

Static Validation

- Based on grammars and parsing trees
- Supports contracts with parameters only
- Analyses all executions of a program
  - May report false positives
- Uses points-to information to handle multiple instances of a module
- Class Scope Mode
  - Allows the analysis to handle large programs
  - Checks each class individually
  - Calls to other classes are ignored

Static Validation Algorithm

```
1  Require:  P: client's program, ℝ: module contract;
2  for t ∈ threads(P) do
3      G_t ← build_grammar(t);
4      G'_t ← subword_grammar(G_t);
5      for ϱ ∈ ℝ do
6          T ← parse(G'_t, ϱ);
7          for τ ∈ T do
8              N ← lowest_common_ancestor(τ, ϱ);
9              if ¬run_atomically(N) then return ERROR;
10 return OK;
```

Illustration of the Static Validation Approach

```
void run() {
  if (cond)
    f();
  else {
    m.indexOf();
    g();
  }
}
void atomic f() {
  m.indexOf();
  g();
}
void atomic g() {
  m.remove();
}
```

Illustration of the Static Validation Approach

```
void run() {
  if (cond)
    f();
  else {
    m.indexOf();
    g();
  }
}
void atomic f() {
  m.indexOf();
  g();
}
void atomic g() {
  m.remove();
}
```

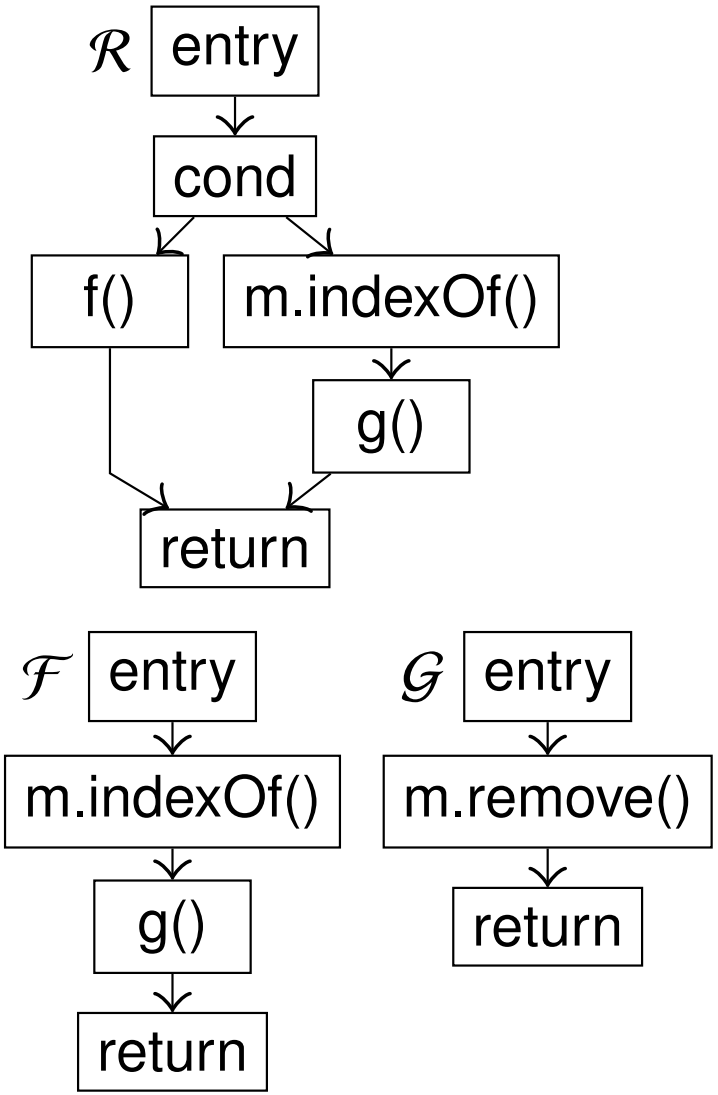
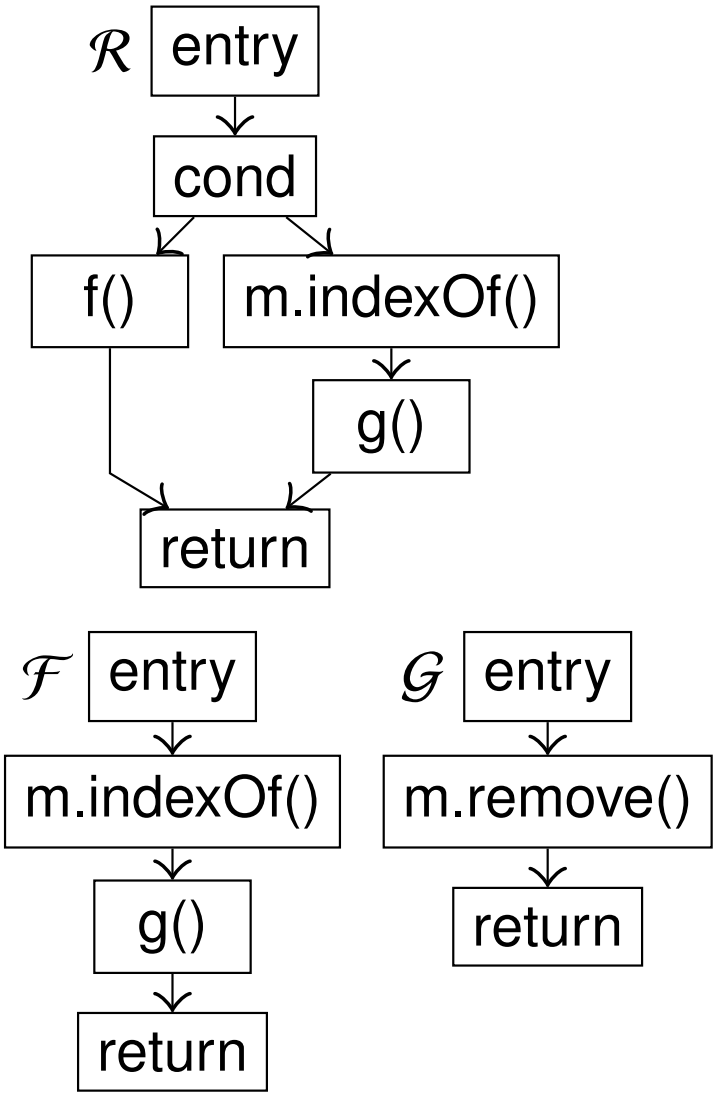


Illustration of the Static Validation Approach

```
void run() {
  if (cond)
    f();
  else {
    m.indexOf();
    g();
  }
}
void atomic f() {
  m.indexOf();
  g();
}
void atomic g() {
  m.remove();
}
```

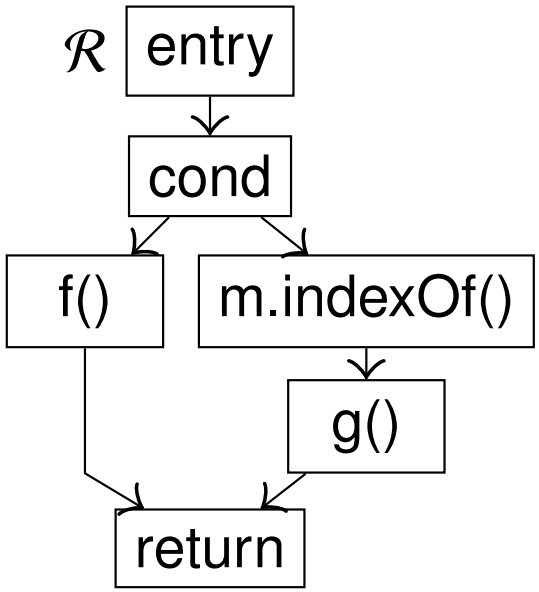


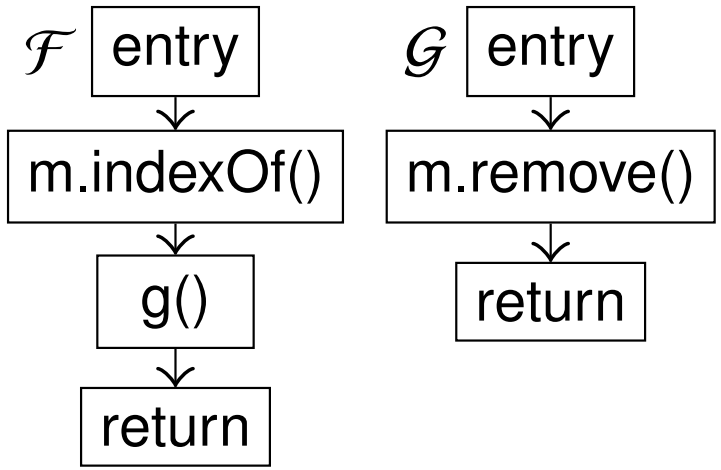
$\mathcal{R} \to \mathbf{a}\,\mathcal{G}$ $\quad$ $\mathcal{F} \to \mathbf{a}\,\mathcal{G}$

$\mathcal{R} \to \mathcal{F}$ $\quad$ $\mathcal{G} \to \mathbf{b}$

Illustration of the Static Validation Approach

```
void run() {
  if (cond)
    f();
  else {
    m.indexOf();
    g();
  }
}
void atomic f() {
  m.indexOf();
  g();
}
void atomic g() {
  m.remove();
}
```





$\mathcal{R} \to \mathbf{a}\,\mathcal{G}$ $\quad$ $\mathcal{F} \to \mathbf{a}\,\mathcal{G}$

$\mathcal{R} \to \mathcal{F}$ $\quad$ $\mathcal{G} \to \mathbf{b}$

$\mathcal{R} \to a\,\mathcal{G}$ $\quad$ $\mathcal{F} \to a\,\mathcal{G}$

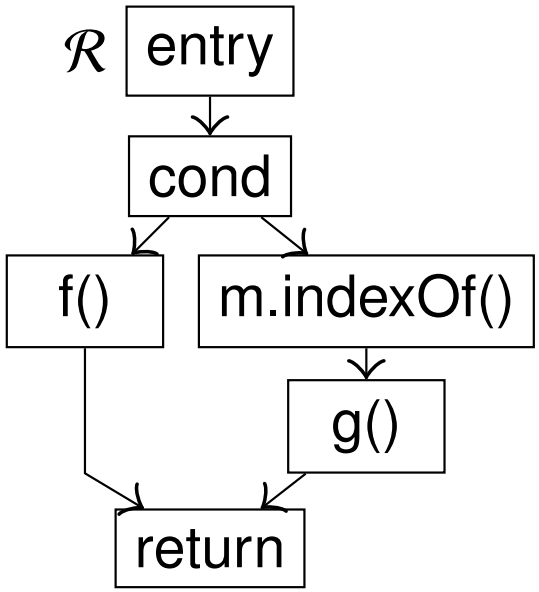
$\mathcal{R} \to \mathcal{F}$ $\quad$ $\mathcal{G} \to b$

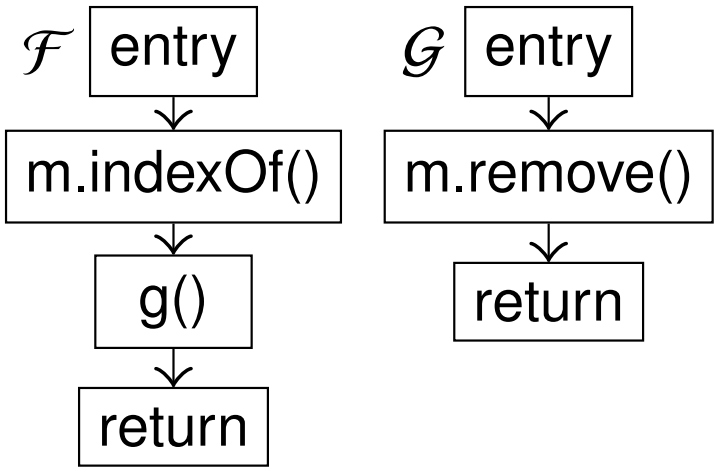
$a \to \mathbf{a}$ $\quad$ $b \to \mathbf{b}$

$a \to \epsilon$ $\quad$ $b \to \epsilon$

Illustration of the Static Validation Approach

```
void run() {
  if (cond)
    f();
  else {
    m.indexOf();
    g();
  }
}
void atomic f() {
  m.indexOf();
  g();
}
void atomic g() {
  m.remove();
}
```





$\mathcal{R} \to \mathbf{a}\,\mathcal{G}$ $\quad \mathcal{F} \to \mathbf{a}\,\mathcal{G}$

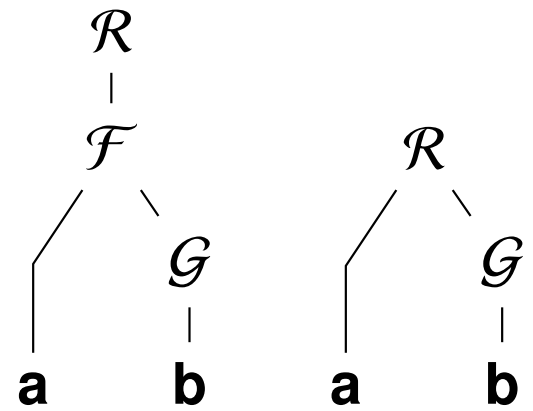
$\mathcal{R} \to \mathcal{F}$ $\quad \mathcal{G} \to \mathbf{b}$

$\mathcal{R} \to a\,\mathcal{G}$ $\quad \mathcal{F} \to a\,\mathcal{G}$

$\mathcal{R} \to \mathcal{F}$ $\quad \mathcal{G} \to b$

$a \to \mathbf{a}$ $\quad b \to \mathbf{b}$

$a \to \epsilon$ $\quad b \to \epsilon$



Experimental Results

Benchmark	Clauses	Contract Violations	False Positives	Potential AV	Real AV	SLOC	Time (s)
Allocate Vector	1	1	0	0	1	183	0.120
Coord03	4	1	0	0	1	151	0.093
Coord04	2	1	0	0	1	35	0.039
Jigsaw	1	1	0	0	1	100	0.044
Local	2	1	0	0	1	24	0.033
Knight	1	1	0	0	1	135	0.219
NASA	1	1	0	0	1	89	0.035
Store	1	1	0	0	1	621	0.090
StringBuffer	1	1	0	0	1	27	0.032
UnderReporting	1	1	0	0	1	20	0.029
VectorFail	2	1	0	0	1	70	0.048
Account	4	2	0	0	2	42	0.041
Arithmetic DB	2	2	0	0	2	243	0.272
Connection	2	2	0	0	2	74	0.058
Elevator	2	2	0	0	2	268	0.333
OpenJMS 0.7	6	54	10	28	4	163K	148
Tomcat 6.0	9	157	16	47	3	239K	3070
Cassandra 2.0	1	60	24	15	2	192K	246
Derby 10.10	1	19	5	7	1	793K	522
Lucene 4.6	3	136	21	76	0	478K	151

Dynamic Validation

- Based on happens-before relation and vector clocks
- Supports both contracts with spoilers and parameters
- Analyses a concrete execution of a program
  - If a contract is violated in the execution, it will be detected
  - Extrapolation based on the happens-before relation
  - Noise injection to force rare interleavings (executions)
- On-the-fly validation
  - Uses a partial trace (trace window)
  - Does not require a trace to be available
  - Each thread needs to remember
    1. Last instance of each spoiler
    2. Last instance of each target
    3. Up to $|\mathbb{T}|$ additional instances of each target

Dynamic Validation Algorithm

Data: trace window υ, event $e \in \mathbb{E}$ generated by thread $t \in \mathbb{T}$

1 **if** $\exists \varrho \in \mathbb{R}, r \in [\varrho]^{\upsilon}_{t} : e = end(r)$ **then**
2 **for** $\sigma \in \mathbb{C}(\varrho), u \in \mathbb{T} : u \neq t$ **do**
3 **if** $\exists s \in [\sigma]^{\upsilon}_{u} : start(s) \not\prec_{hb} start(r) \wedge end(r) \not\prec_{hb} end(s)$ **then** r is violated by s ;
4 **if** $\exists s \in [\sigma]^{\tau}_{u} : start(s) \in \upsilon \wedge end(s) \notin \upsilon$ **then**
5 **if** $start(s) \prec_{hb} start(r)$ **then**
6 **if** $\exists r' \in [\varrho]^{\upsilon}_{t} : r' \neq r \wedge start(s) \not\prec_{hb} start(r')$ **then** $PV^{\varrho,\sigma}_{t}(u) = VC_{end(r')}(t)$;
7 **if** $\exists r' \in [\varrho]^{\upsilon}_{t} : r' \neq r$ **then** $\upsilon \rightarrow r'$;
8 **if** $\sigma \in \mathbb{S}, s \in [\sigma]^{\upsilon}_{t} : end(s) = e$ **then**
9 **if** $\exists s' \in [\sigma]^{\upsilon}_{t} : s' \neq s$ **then** $\upsilon \rightarrow s'$;
10 **for** $\varrho \in \mathbb{C}(\sigma), u \in \mathbb{T} : u \neq t$ **do**
11 **if** $\exists r \in [\varrho]^{\upsilon}_{u} : start(s) \not\prec_{hb} start(r) \wedge end(r) \not\prec_{hb} end(s)$ **then** r is violated by s ;
12 **if** $PV^{\varrho,\sigma}_{u}(t) \neq 0 \wedge PV^{\varrho,\sigma}_{u}(t) \leq VC_{end(s)}(u)$ **then**
13 an instance of ϱ is violated by s;

Illustration of the Dynamic Validation Approach

```
void tmain() {
  thread_create(worker,input);
  queue_init();
  result = queue_receive();
}
void worker(void* data) {
  do_work();
  queue_send(result);
}
```

Illustration of the Dynamic Validation Approach

```
void tmain() {
  thread_create(worker,input);
  queue_init();
  result = queue_receive();
}
void worker(void* data) {
  do_work();
  queue_send(result);
}
```

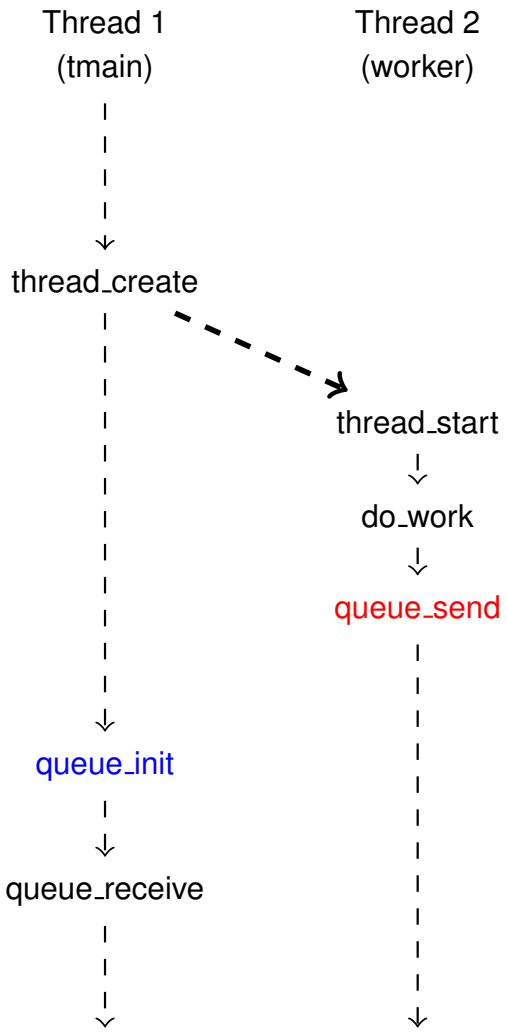


Illustration of the Dynamic Validation Approach

```
void tmain() {
  thread_create(worker,input);
  queue_init();
  result = queue_receive();
}
void worker(void* data) {
  do_work();
  queue_send(result);
}
```

Atomicity Violation?

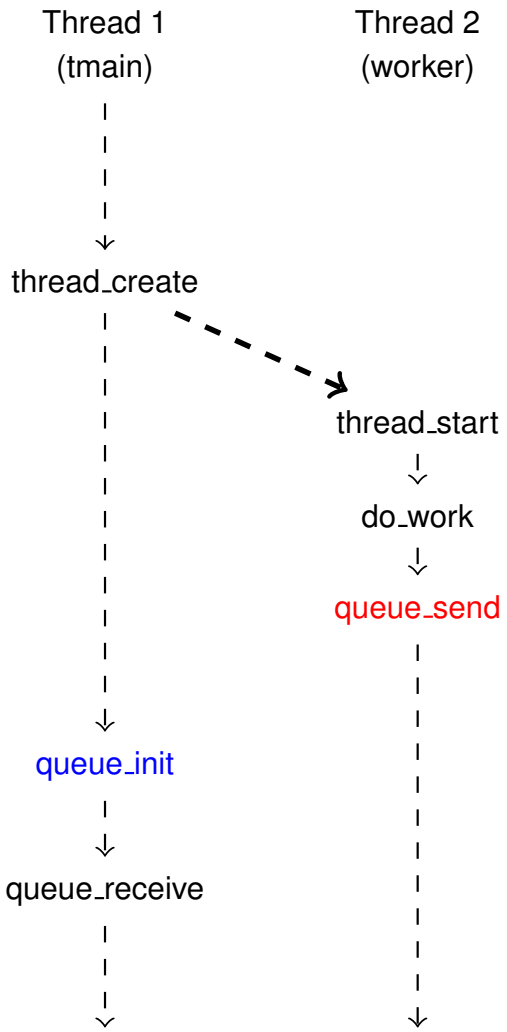


Illustration of the Dynamic Validation Approach

```
void tmain() {
  thread_create(worker,input);
  queue_init();
  result = queue_receive();
}
void worker(void* data) {
  do_work();
  queue_send(result);
}
```

Atomicity Violation? No

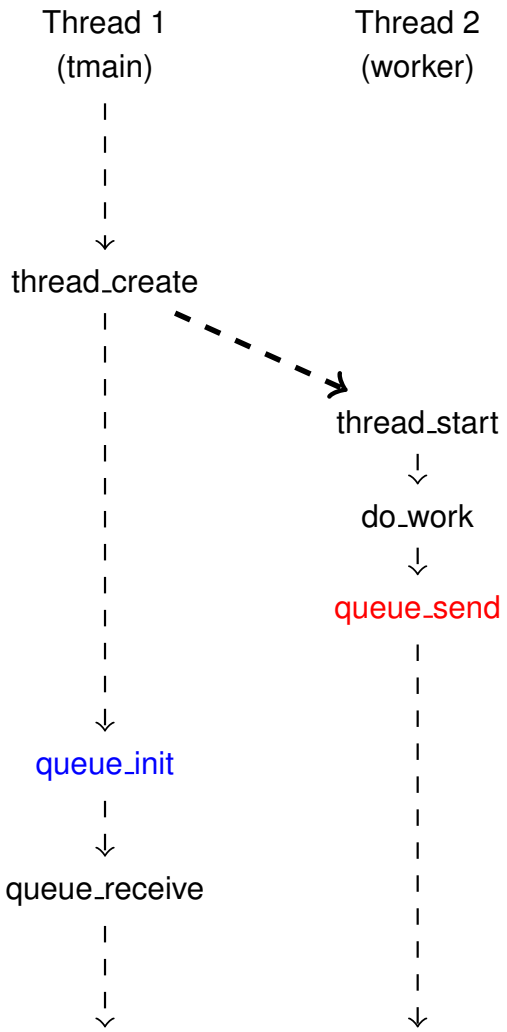


Illustration of the Dynamic Validation Approach

```
void tmain() {
  thread_create(worker,input);
  queue_init();
  result = queue_receive();
}
void worker(void* data) {
  do_work();
  queue_send(result);
}
```

Atomicity Violation? No

Data Race?

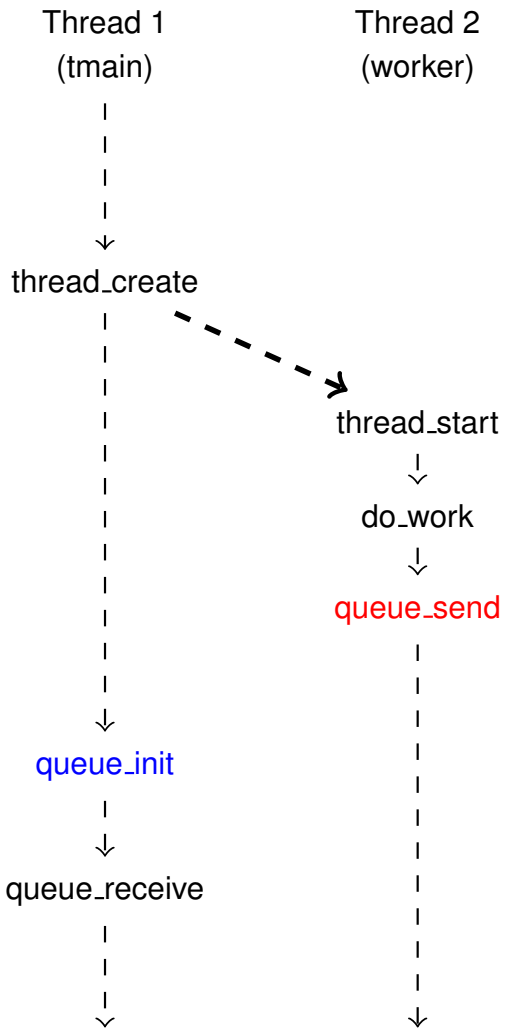


Illustration of the Dynamic Validation Approach

```
void tmain() {
  thread_create(worker,input);
  queue_init();
  result = queue_receive();
}
void worker(void* data) {
  do_work();
  queue_send(result);
}
```

Atomicity Violation? No

Data Race?

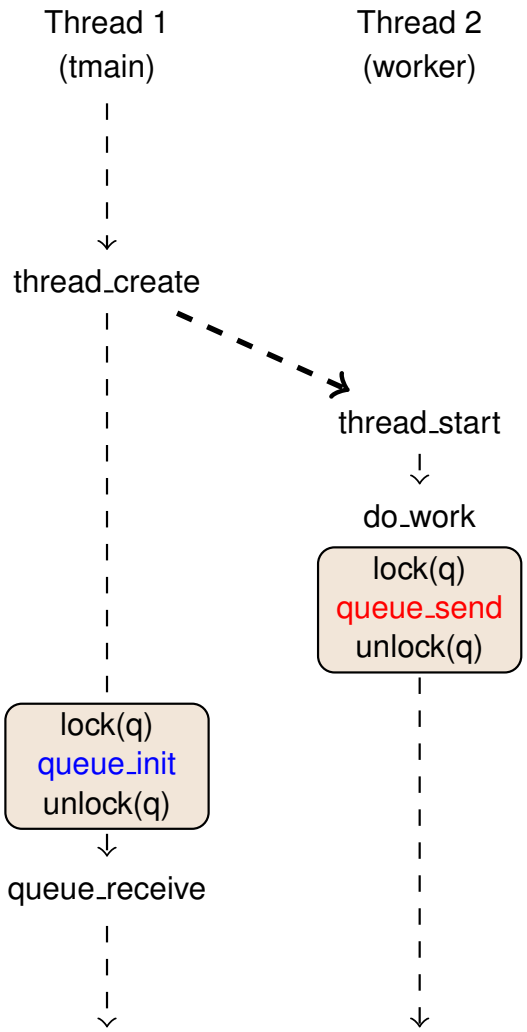


Illustration of the Dynamic Validation Approach

```
void tmain() {
  thread_create(worker,input);
  queue_init();
  result = queue_receive();
}
void worker(void* data) {
  do_work();
  queue_send(result);
}
```

Atomicity Violation? No

Data Race? No

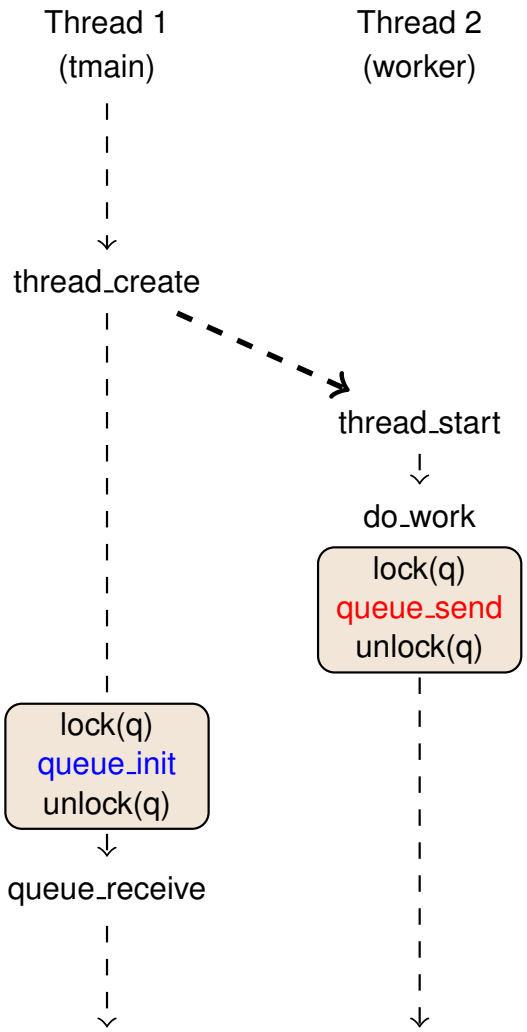


Illustration of the Dynamic Validation Approach

```
void tmain() {
  thread_create(worker,input);
  queue_init();
  result = queue_receive();
}
void worker(void* data) {
  do_work();
  queue_send(result);
}
```

Atomicity Violation? No

Data Race? No

Order Violation!

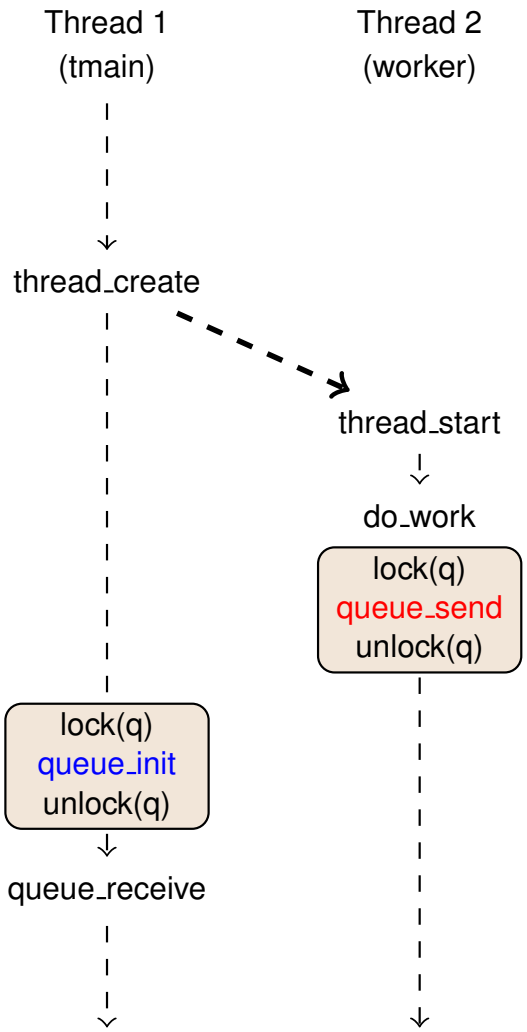
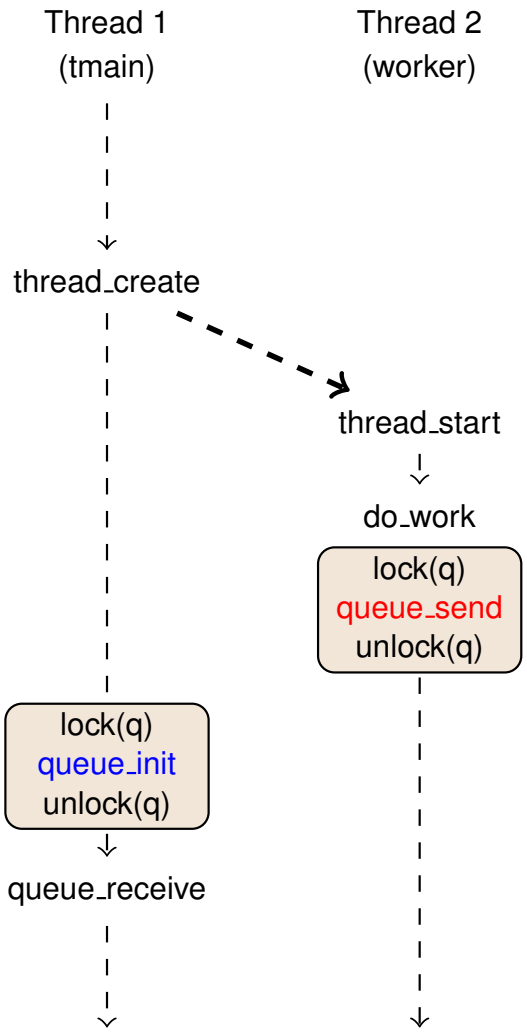


Illustration of the Dynamic Validation Approach

```
void tmain() {
  thread_create(worker,input);
  queue_init();
  result = queue_receive();
}
void worker(void* data) {
  do_work();
  queue_send(result);
}
```



Atomicity Violation? No

Data Race? No

Order Violation!

Can we detect it using contracts?

Illustration of the Dynamic Validation Approach

```
void tmain() {
  thread_create(worker,input);
  queue_init();
  result = queue_receive();
}
void worker(void* data) {
  do_work();
  queue_send(result);
}
```

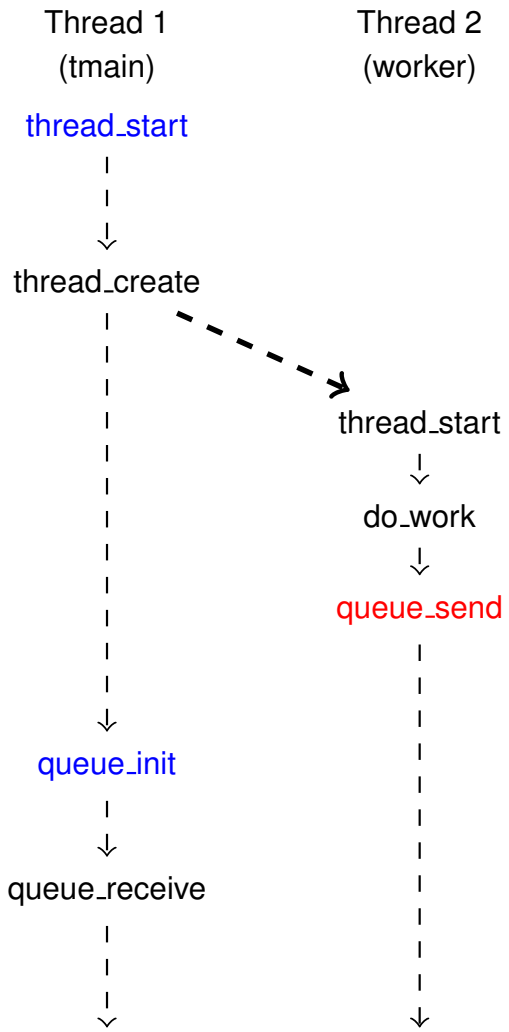


Illustration of the Dynamic Validation Approach

```
void tmain() {
  thread_create(worker,input);
  queue_init();
  result = queue_receive();
}
void worker(void* data) {
  do_work();
  queue_send(result);
}
```

Contract: thread_start queue_init
← {queue_send, queue_receive}

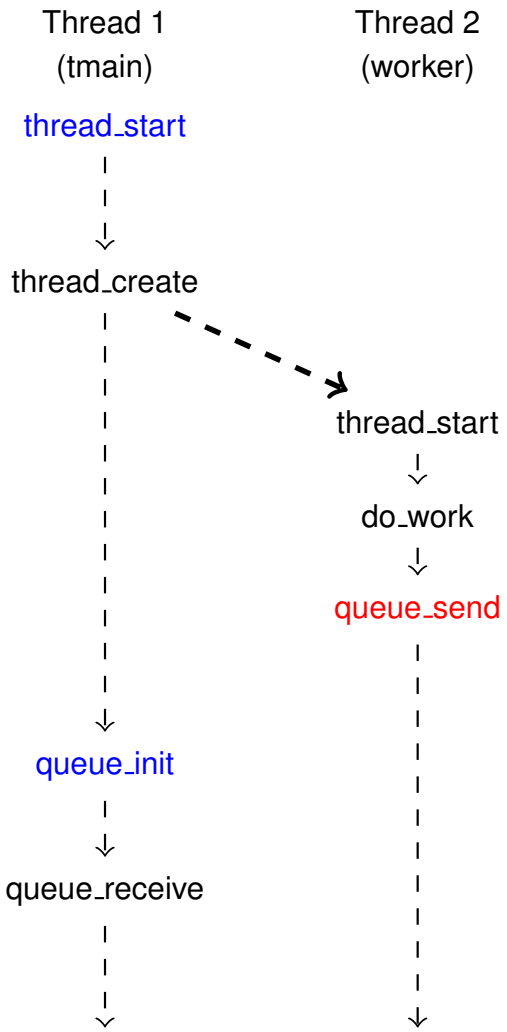


Illustration of the Dynamic Validation Approach

```
void tmain() {
  thread_create(worker,input);
  queue_init();
  result = queue_receive();
}
void worker(void* data) {
  do_work();
  queue_send(result);
}
```

Contract: thread_start queue_init
← {queue_send, queue_receive}

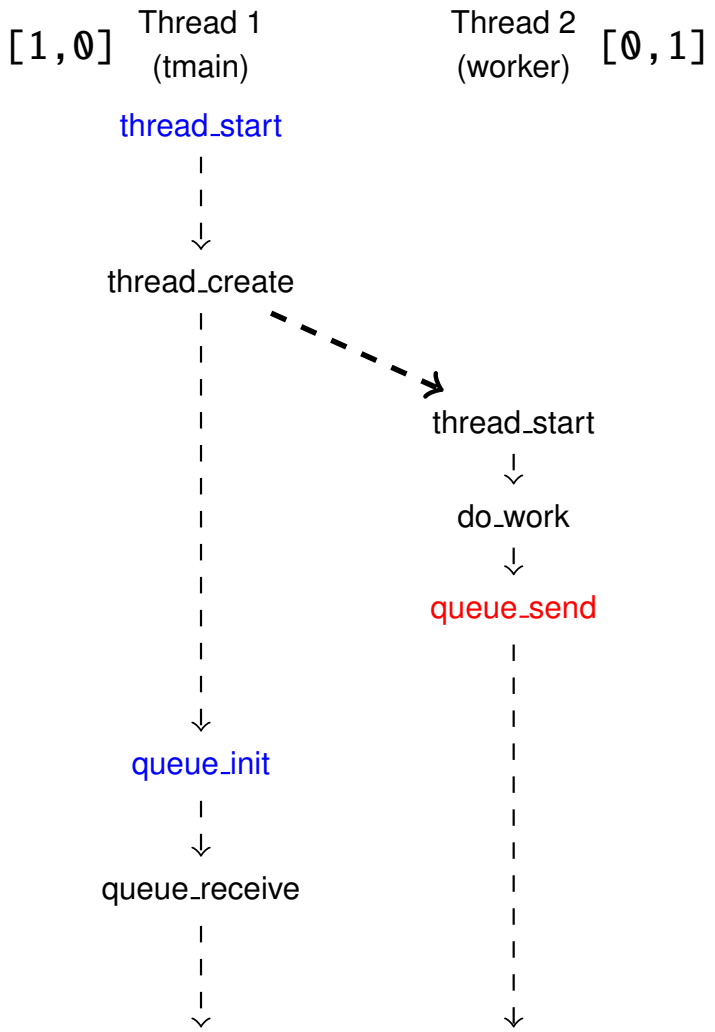


Illustration of the Dynamic Validation Approach

```
void tmain() {
  thread_create(worker,input);
  queue_init();
  result = queue_receive();
}
void worker(void* data) {
  do_work();
  queue_send(result);
}
```

Contract: thread_start queue_init
← {queue_send, queue_receive}

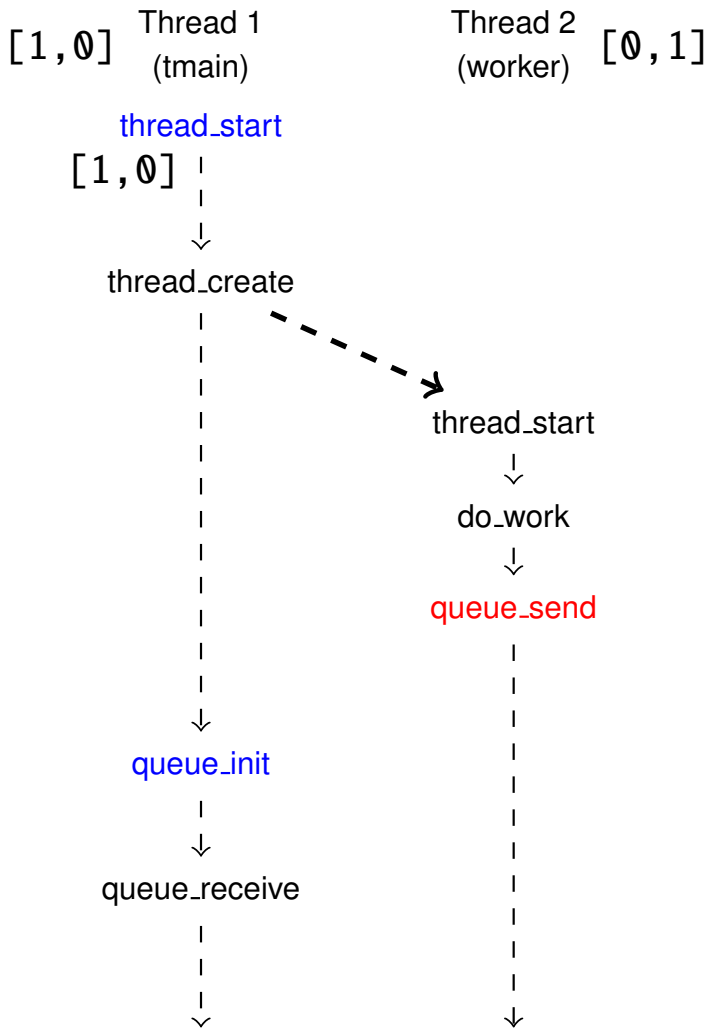


Illustration of the Dynamic Validation Approach

```
void tmain() {
  thread_create(worker,input);
  queue_init();
  result = queue_receive();
}
void worker(void* data) {
  do_work();
  queue_send(result);
}
```

Contract: thread_start queue_init
← {queue_send, queue_receive}

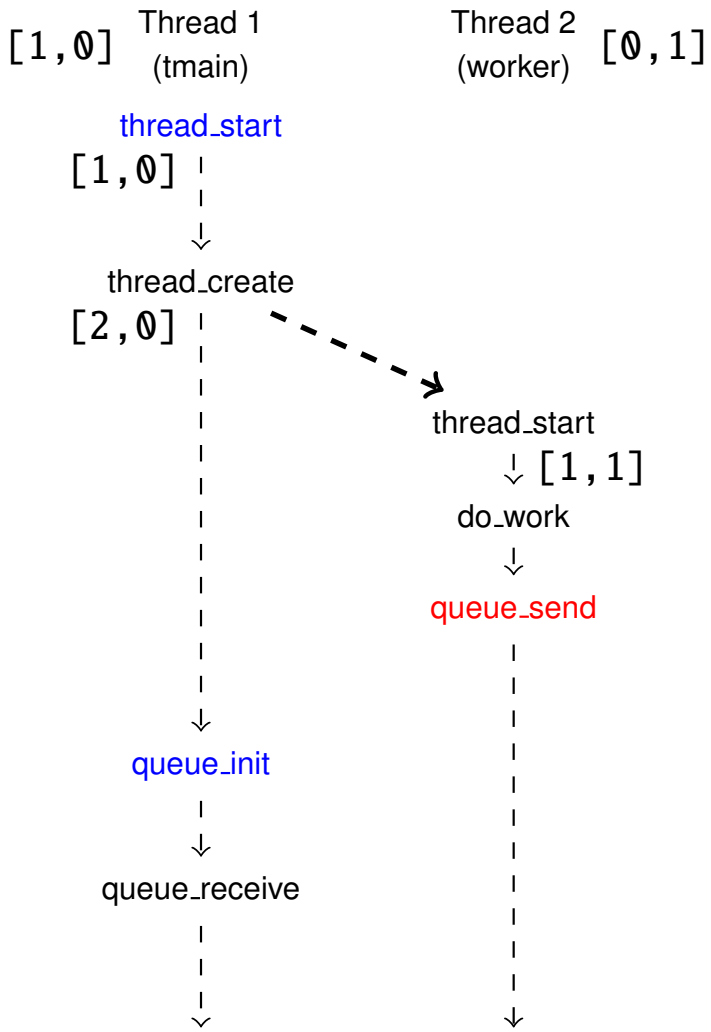


Illustration of the Dynamic Validation Approach

```
void tmain() {
  thread_create(worker,input);
  queue_init();
  result = queue_receive();
}
void worker(void* data) {
  do_work();
  queue_send(result);
}
```

Contract: thread_start queue_init
← {queue_send, queue_receive}

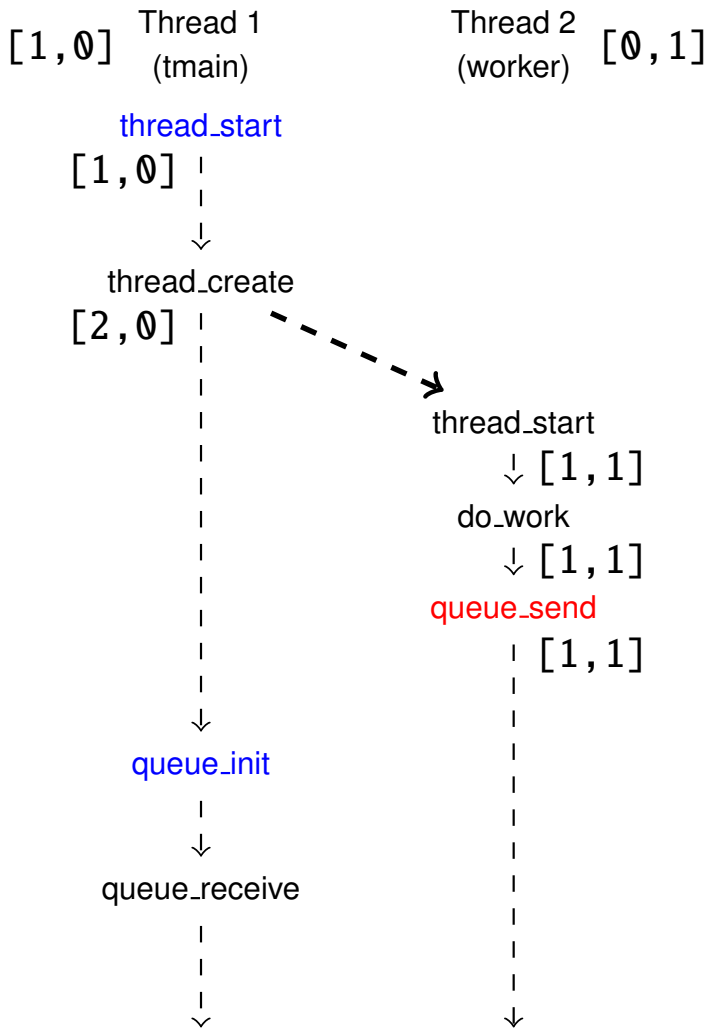


Illustration of the Dynamic Validation Approach

```
void tmain() {
  thread_create(worker,input);
  queue_init();
  result = queue_receive();
}
void worker(void* data) {
  do_work();
  queue_send(result);
}
```

Contract: thread_start queue_init
← {queue_send, queue_receive}

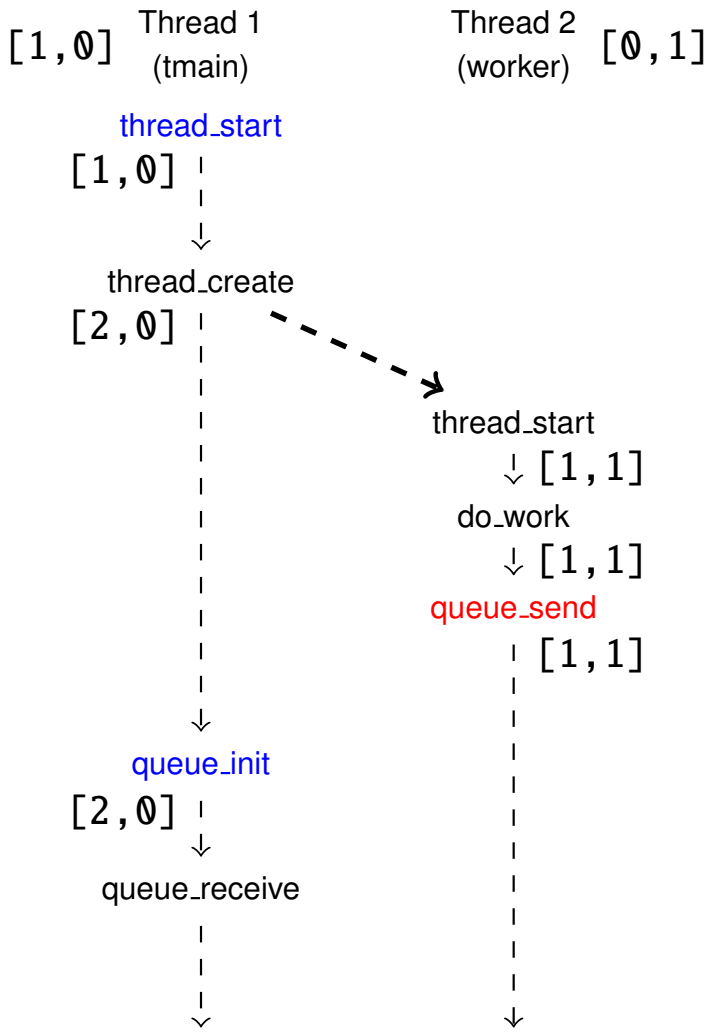


Illustration of the Dynamic Validation Approach

```
void tmain() {
  thread_create(worker,input);
  queue_init();
  result = queue_receive();
}
void worker(void* data) {
  do_work();
  queue_send(result);
}
```

Contract: thread_start queue_init
← {queue_send, queue_receive}

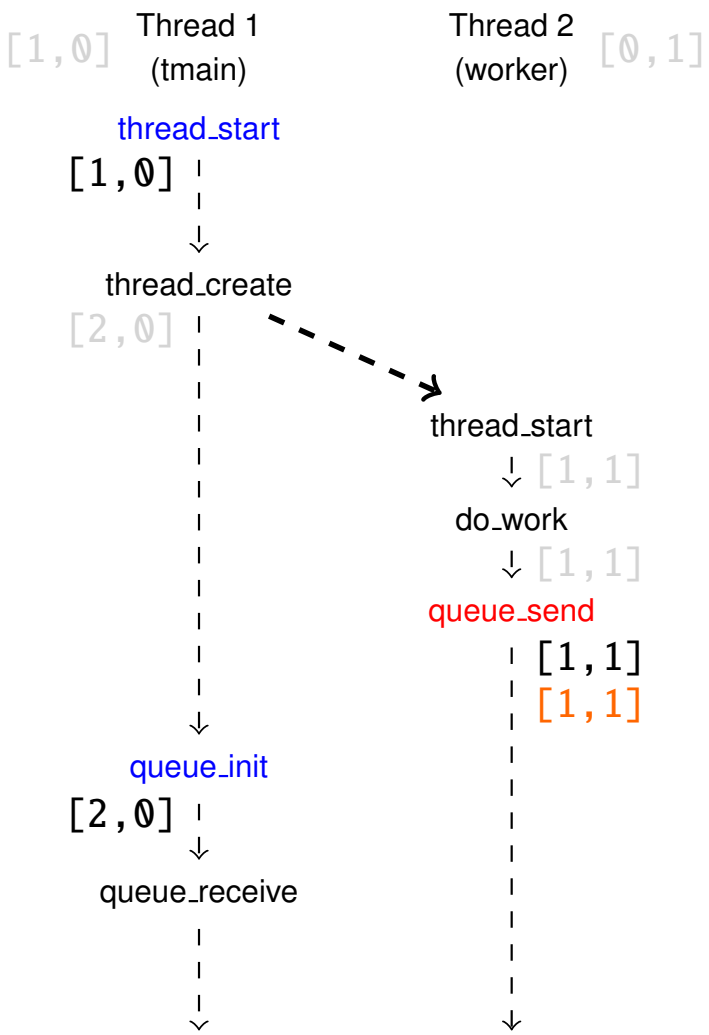


Illustration of the Dynamic Validation Approach

```
void tmain() {
  thread_create(worker,input);
  queue_init();
  result = queue_receive();
}
void worker(void* data) {
  do_work();
  queue_send(result);
}
```

Contract: thread_start queue_init
← {queue_send, queue_receive}

$start(s) \not\leq_{hb} start(r) \wedge end(r) \not\leq_{hb} end(s)$

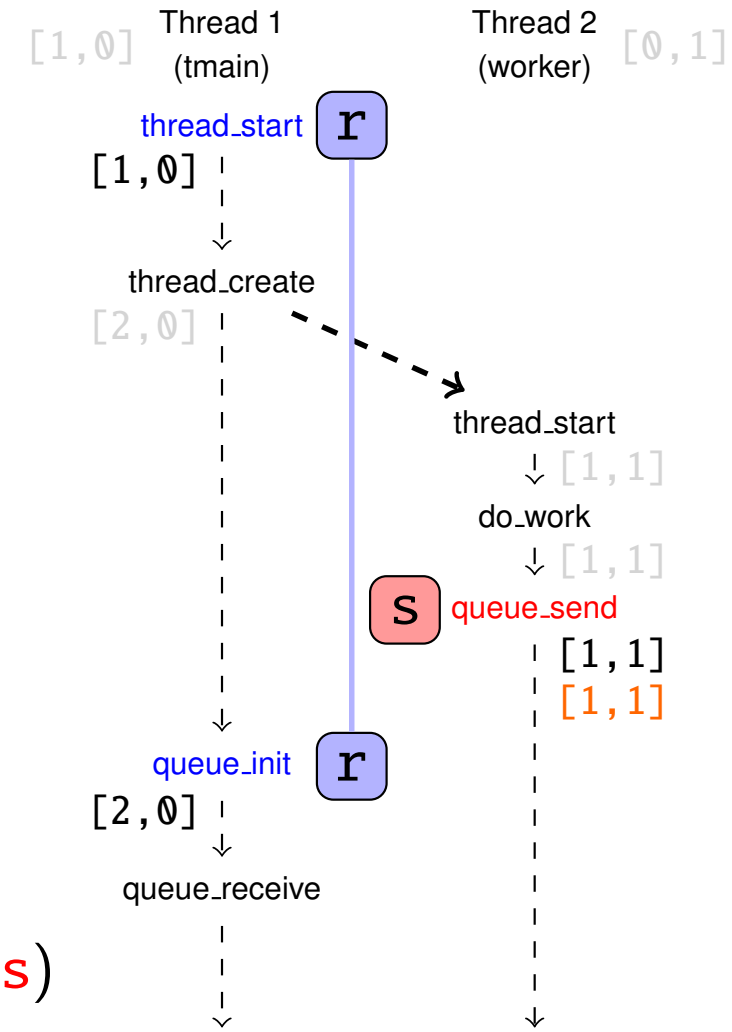


Illustration of the Dynamic Validation Approach

```
void tmain() {
  thread_create(worker,input);
  queue_init();
  result = queue_receive();
}
void worker(void* data) {
  do_work();
  queue_send(result);
}
```

$$e_t \prec_{hb} e_u \sim VC_{e_t}(t) \leq VC_{e_u}(t)$$

Contract: thread_start queue_init

$\leftarrow$ {queue_send, queue_receive}

$start(s) \not\prec_{hb} start(r) \wedge end(r) \not\prec_{hb} end(s)$

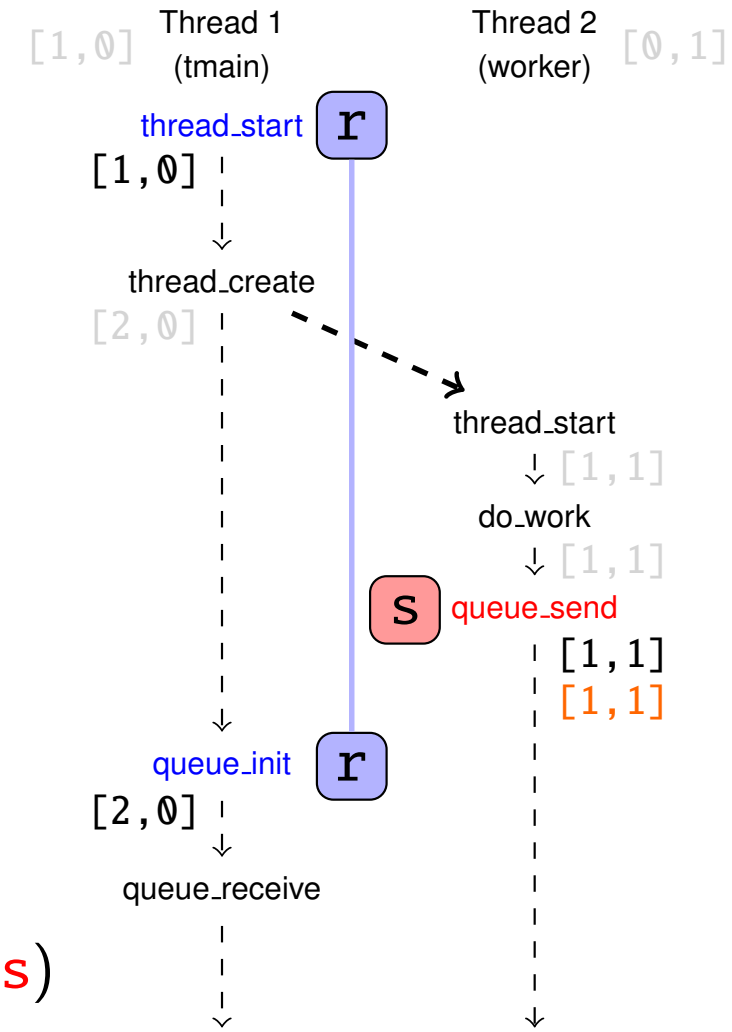


Illustration of the Dynamic Validation Approach

```
void tmain() {
  thread_create(worker,input);
  queue_init();
  result = queue_receive();
}
void worker(void* data) {
  do_work();
  queue_send(result);
}
```

$$e_t \prec_{hb} e_u \sim VC_{e_t}(t) \leq VC_{e_u}(t)$$

Contract: thread_start queue_init

$\leftarrow$ {queue_send, queue_receive}

$start(s) \not\prec_{hb} start(r) \wedge end(r) \not\prec_{hb} end(s)$

$\neg(1 \leq 0)$

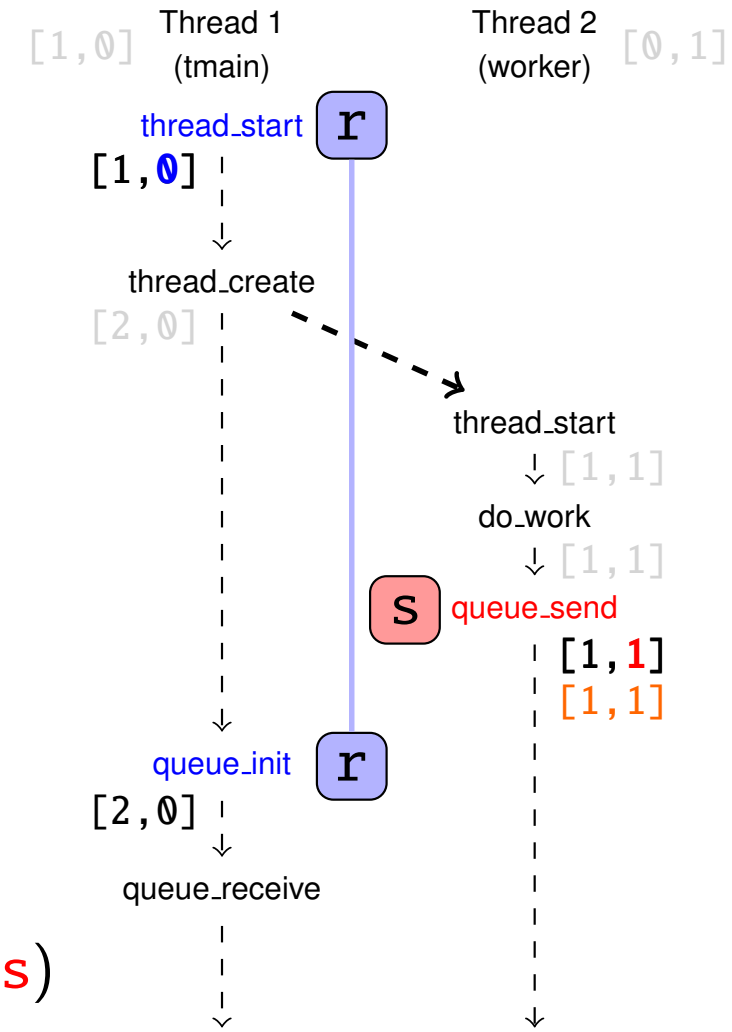


Illustration of the Dynamic Validation Approach

```
void tmain() {
  thread_create(worker,input);
  queue_init();
  result = queue_receive();
}
void worker(void* data) {
  do_work();
  queue_send(result);
}
```

$$e_t \prec_{hb} e_u \sim VC_{e_t}(t) \leq VC_{e_u}(t)$$

Contract: thread_start queue_init

$\leftarrow$ {queue_send, queue_receive}

$start(s) \not\prec_{hb} start(r) \wedge end(r) \not\prec_{hb} end(s)$

$\neg(1 \leq 0) \quad \wedge \quad \neg(2 \leq 1)$

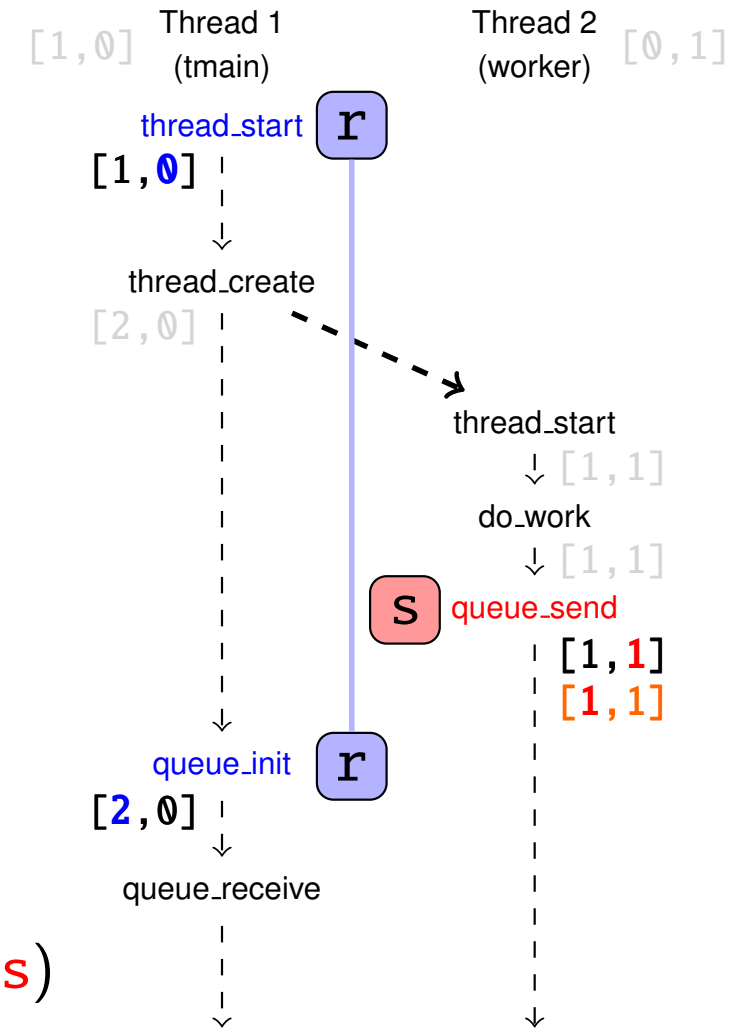


Illustration of the Dynamic Validation Approach

```
void tmain() {
  thread_create(worker,input);
  queue_init();
  result = queue_receive();
}
void worker(void* data) {
  do_work();
  queue_send(result);
}
```

$$e_t \prec_{hb} e_u \sim VC_{e_t}(t) \leq VC_{e_u}(t)$$

Contract: thread_start queue_init

$\leftarrow$ {queue_send, queue_receive}

$start(s) \not\prec_{hb} start(r) \wedge end(r) \not\prec_{hb} end(s)$

$\neg(1 \leq 0) \quad \wedge \quad \neg(2 \leq 1)$

Contract violated!

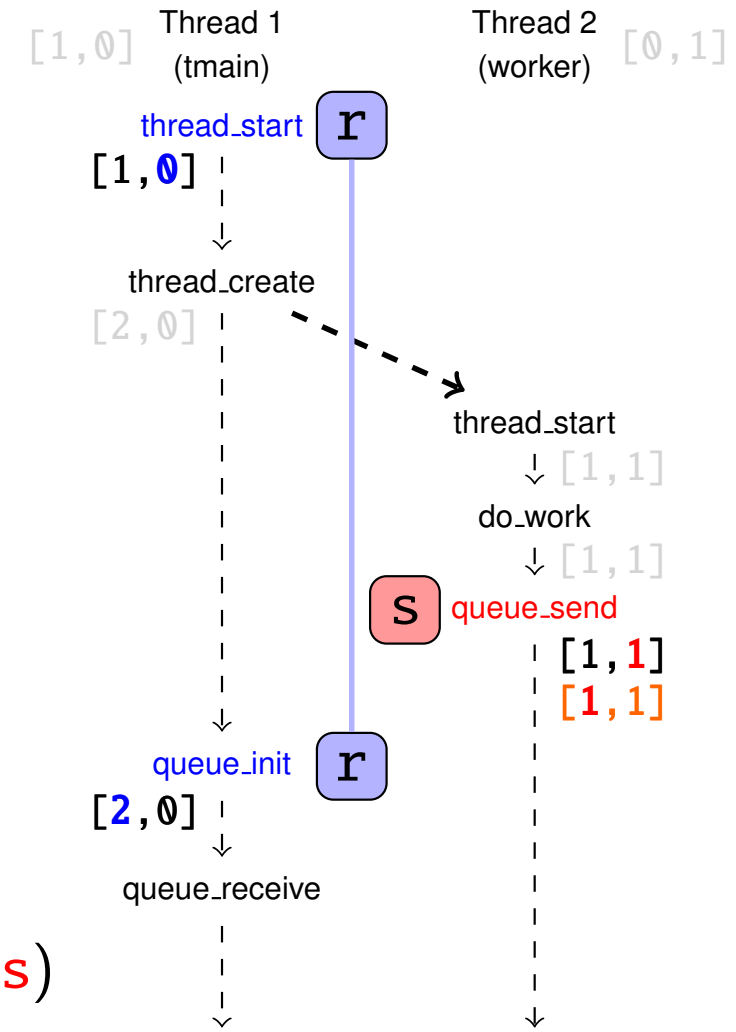


Illustration of the Dynamic Validation Approach

```
void tmain() {
  thread_create(worker,input);
  queue_init();
  result = queue_receive();
}
void worker(void* data) {
  do_work();
  queue_send(result);
}
```

$$e_t \prec_{hb} e_u \sim VC_{e_t}(t) \leq VC_{e_u}(t)$$

Contract: thread_start queue_init

$\leftarrow$ {queue_send, queue_receive}

$start(s) \not\prec_{hb} start(r) \wedge end(r) \not\prec_{hb} end(s)$

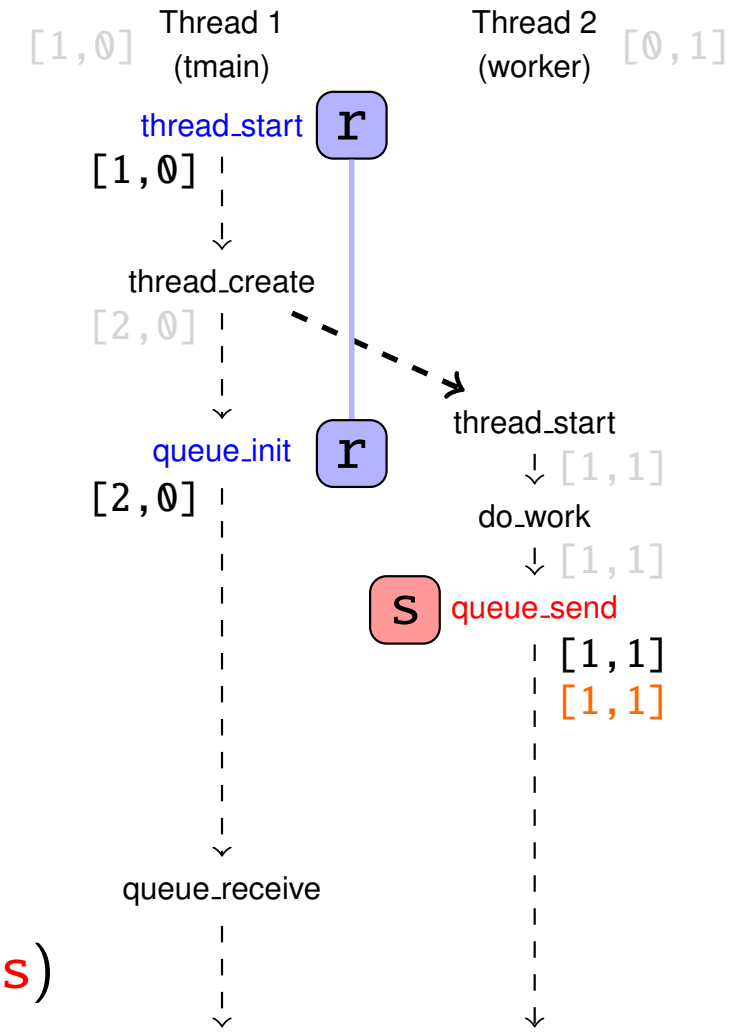


Illustration of the Dynamic Validation Approach

```
void tmain() {
  thread_create(worker,input);
  queue_init();
  result = queue_receive();
}
void worker(void* data) {
  do_work();
  queue_send(result);
}
```

$$e_t \prec_{hb} e_u \sim VC_{e_t}(t) \leq VC_{e_u}(t)$$

Contract: thread_start queue_init

$\leftarrow$ {queue_send, queue_receive}

$start(s) \not\prec_{hb} start(r) \wedge end(r) \not\prec_{hb} end(s)$

$\neg(1 \leq 0) \quad \wedge \quad \neg(2 \leq 1)$

Contract violated!

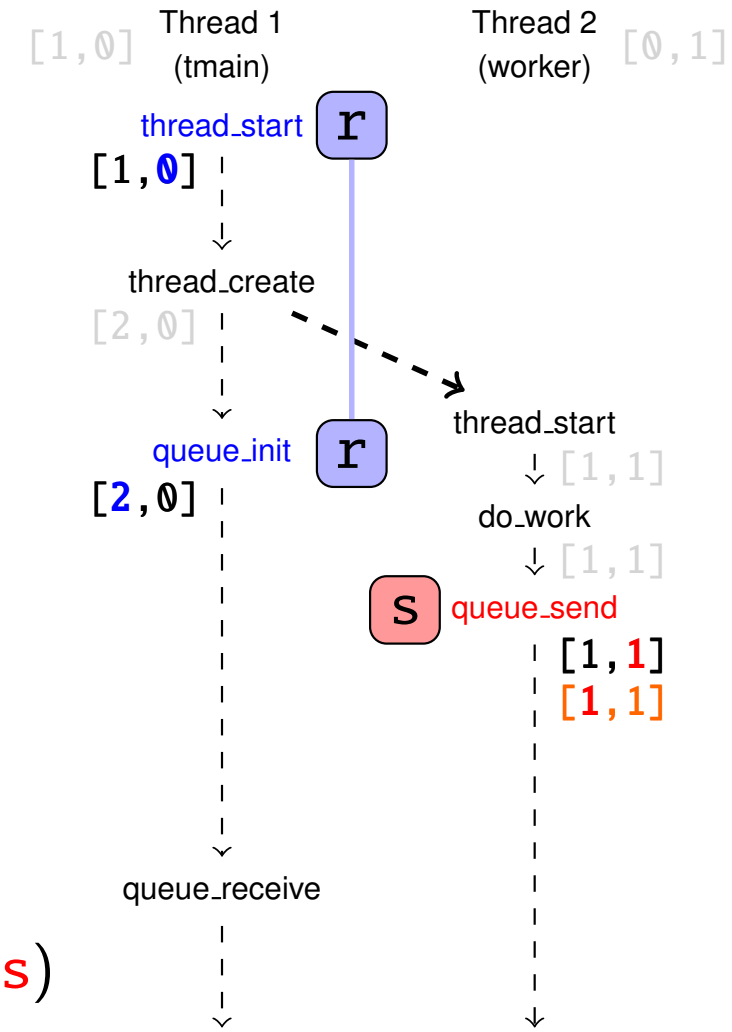


Illustration of the Dynamic Validation Approach

```
void tmain() {
  thread_create(worker,input);
  queue_init();
  result = queue_receive();
}
void worker(void* data) {
  do_work();
  queue_send(result);
}
```

$$e_t \prec_{hb} e_u \sim VC_{e_t}(t) \leq VC_{e_u}(t)$$

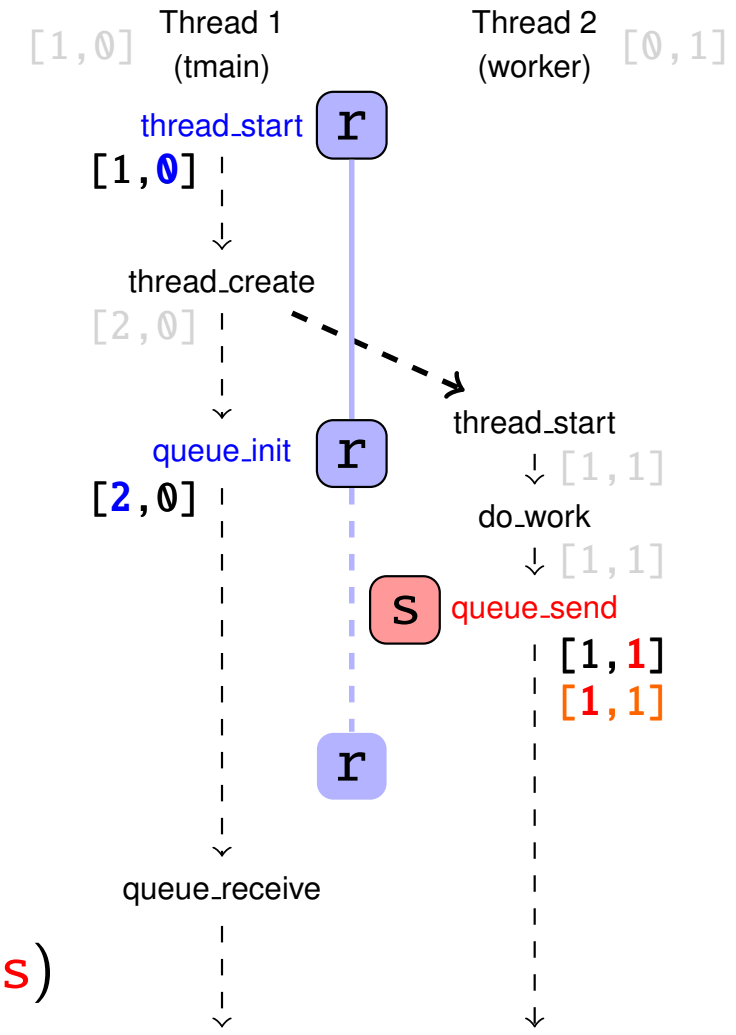
Contract: thread_start queue_init

$\leftarrow$ {queue_send, queue_receive}

$start(s) \not\prec_{hb} start(r) \wedge end(r) \not\prec_{hb} end(s)$

$\neg(1 \leq 0) \quad \wedge \quad \neg(2 \leq 1)$

Contract violated!



Extrapolation!

Illustration of the Dynamic Validation Approach

```
void tmain() {
  thread_create(worker,input);
  queue_init();
  result = queue_receive();
}
void worker(void* data) {
  do_work();
  queue_send(result);
}
```

$$e_t \prec_{hb} e_u \sim VC_{e_t}(t) \leq VC_{e_u}(t)$$

Contract: thread_start queue_init $\leftarrow$ {queue_send, queue_receive}

$start(s) \not\prec_{hb} start(r) \wedge end(r) \not\prec_{hb} end(s)$

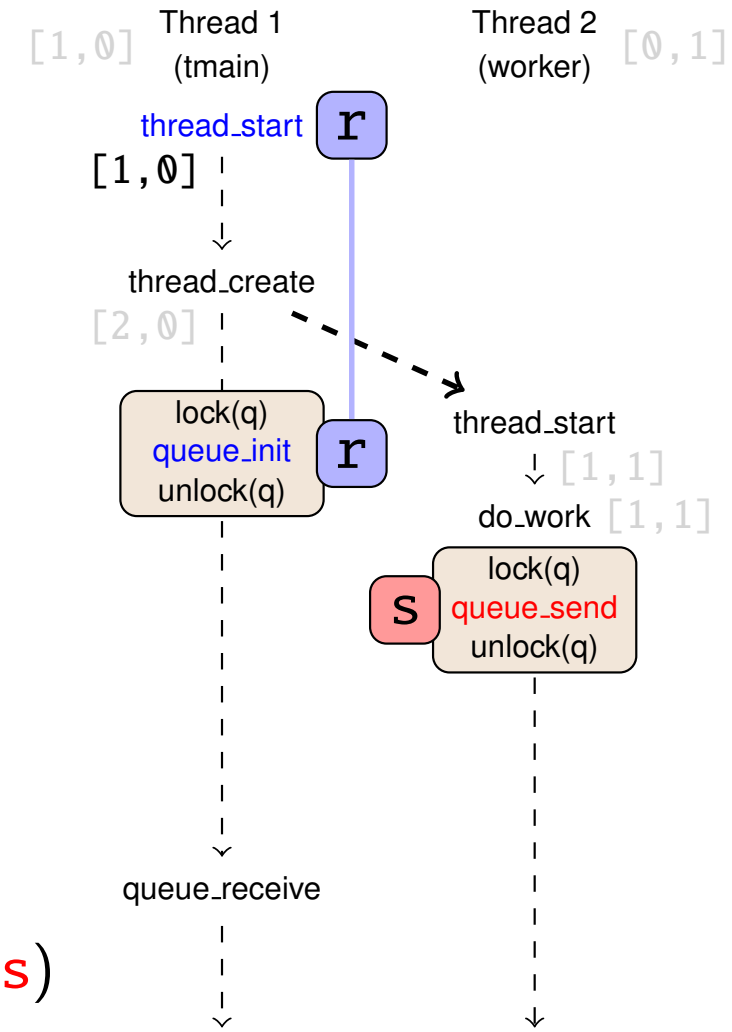


Illustration of the Dynamic Validation Approach

```
void tmain() {
  thread_create(worker,input);
  queue_init();
  result = queue_receive();
}
void worker(void* data) {
  do_work();
  queue_send(result);
}
```

$$e_t \prec_{hb} e_u \sim VC_{e_t}(t) \leq VC_{e_u}(t)$$

Contract: thread_start queue_init

$\leftarrow$ {queue_send, queue_receive}

$start(s) \not\prec_{hb} start(r) \wedge end(r) \not\prec_{hb} end(s)$

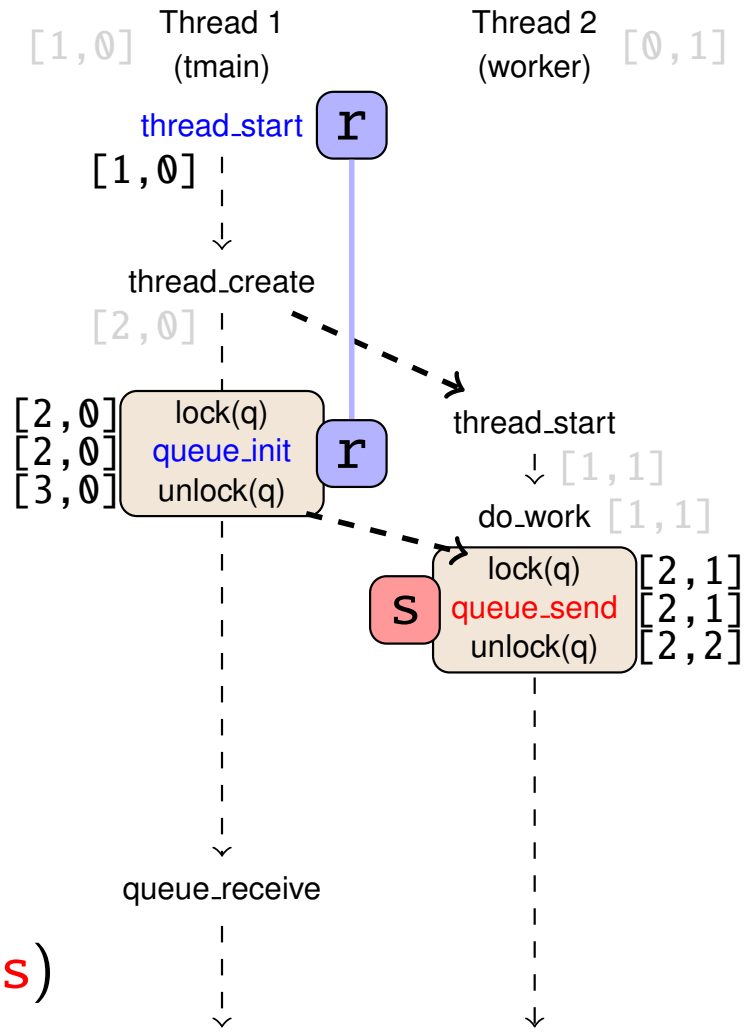


Illustration of the Dynamic Validation Approach

```
void tmain() {
  thread_create(worker,input);
  queue_init();
  result = queue_receive();
}
void worker(void* data) {
  do_work();
  queue_send(result);
}
```

$$e_t \prec_{hb} e_u \sim VC_{e_t}(t) \leq VC_{e_u}(t)$$

Contract: thread_start queue_init

$\leftarrow$ {queue_send, queue_receive}

$start(s) \not\prec_{hb} start(r) \wedge end(r) \not\prec_{hb} end(s)$

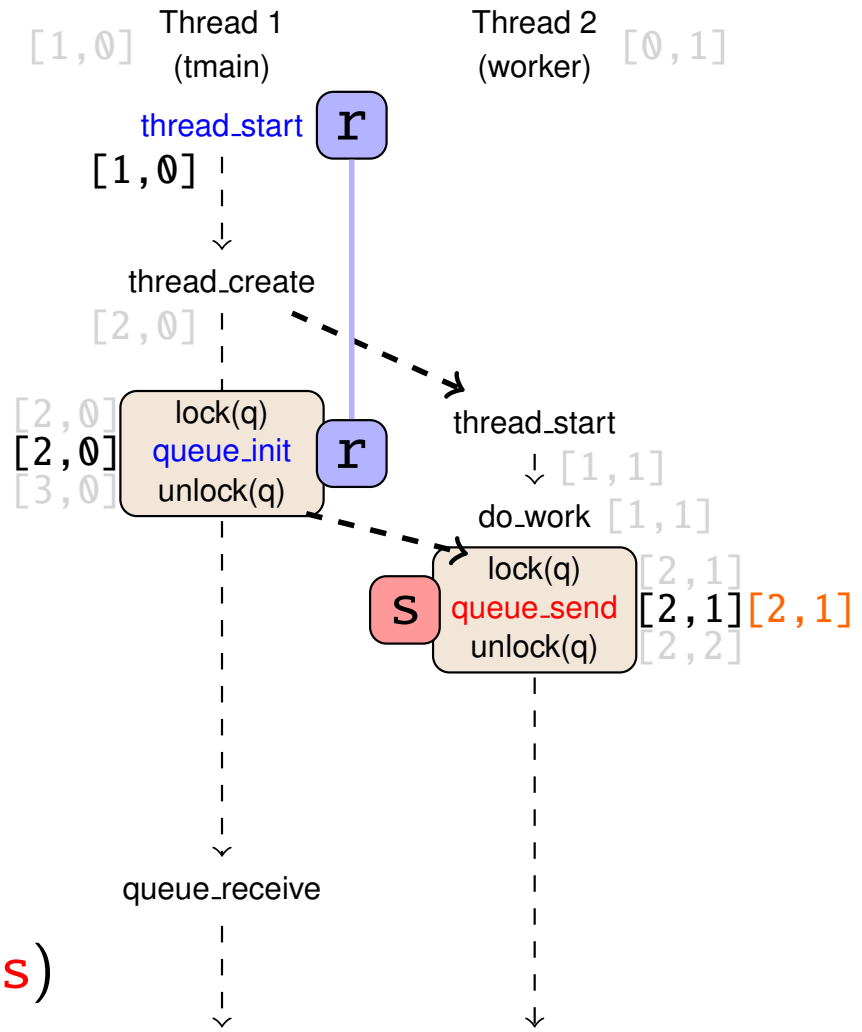


Illustration of the Dynamic Validation Approach

```
void tmain() {
  thread_create(worker,input);
  queue_init();
  result = queue_receive();
}
void worker(void* data) {
  do_work();
  queue_send(result);
}
```

$$e_t \prec_{hb} e_u \sim VC_{e_t}(t) \leq VC_{e_u}(t)$$

Contract: thread_start queue_init

$\leftarrow$ {queue_send, queue_receive}

$start(s) \not\prec_{hb} start(r) \wedge end(r) \not\prec_{hb} end(s)$

$\neg(1 \leq 0) \quad \wedge \quad \neg(2 \leq 2)$

No violation

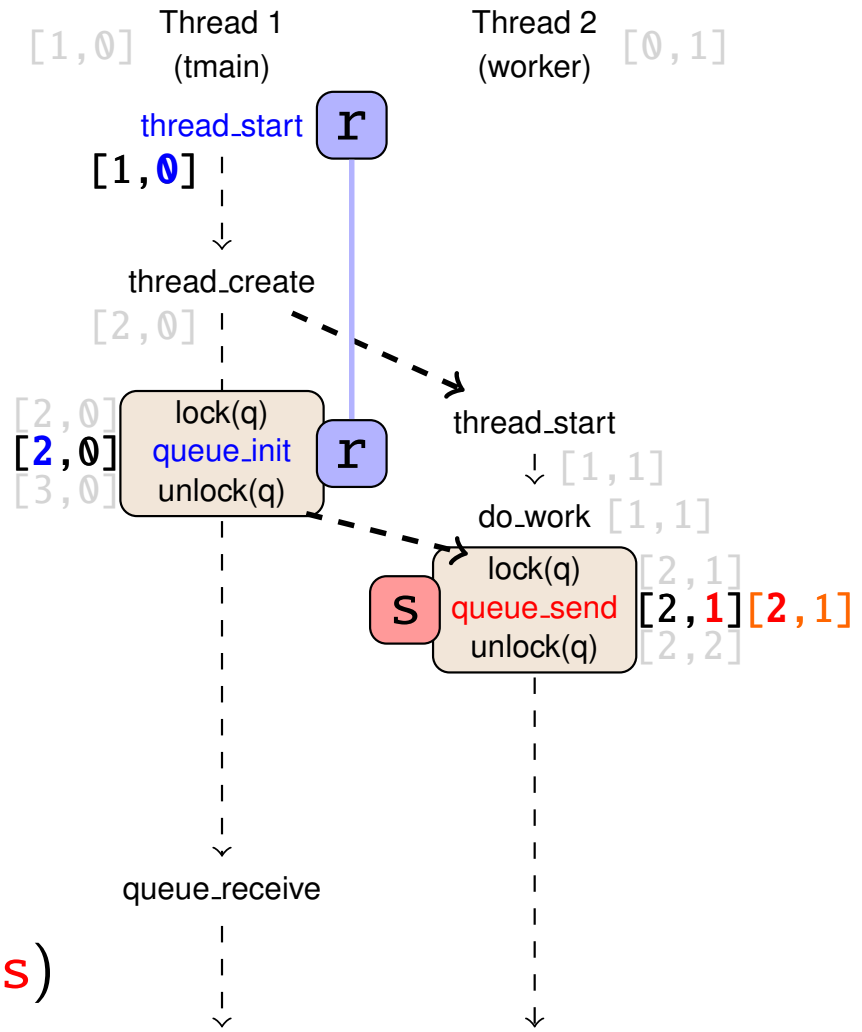


Illustration of the Dynamic Validation Approach

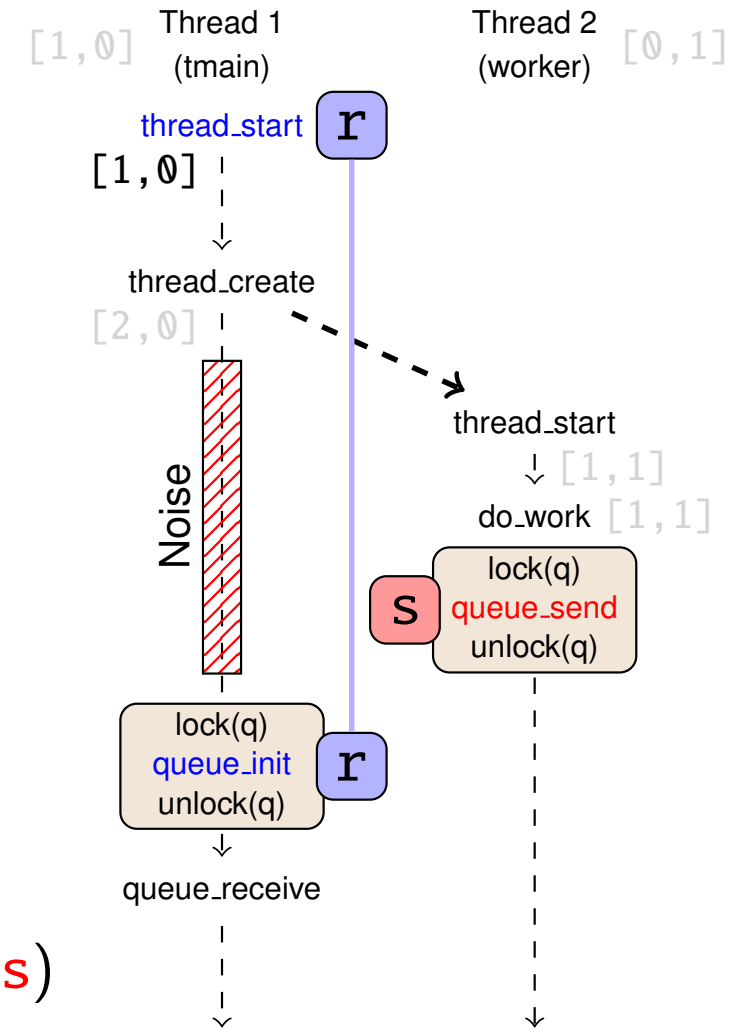
```
void tmain() {
  thread_create(worker,input);
  queue_init();
  result = queue_receive();
}
void worker(void* data) {
  do_work();
  queue_send(result);
}
```

$$e_t \prec_{hb} e_u \sim VC_{e_t}(t) \leq VC_{e_u}(t)$$

Contract: thread_start queue_init

$\leftarrow$ {queue_send, queue_receive}

$start(s) \not\prec_{hb} start(r) \wedge end(r) \not\prec_{hb} end(s)$



Noise injection!

Illustration of the Dynamic Validation Approach

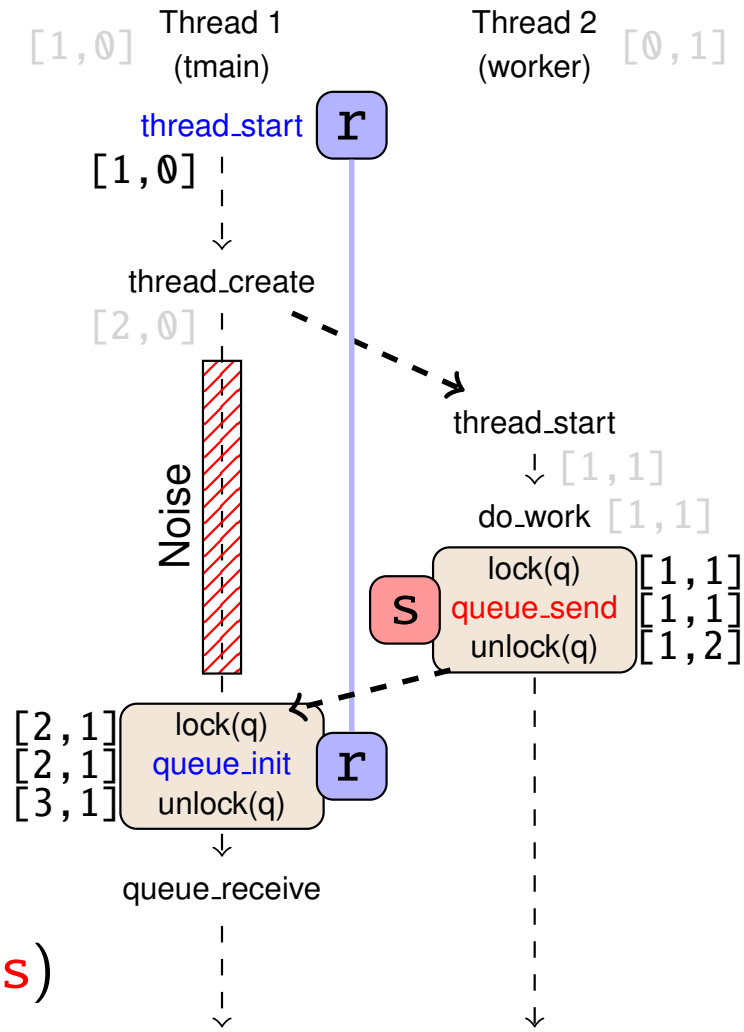
```
void tmain() {
  thread_create(worker,input);
  queue_init();
  result = queue_receive();
}
void worker(void* data) {
  do_work();
  queue_send(result);
}
```

$$e_t \prec_{hb} e_u \sim VC_{e_t}(t) \leq VC_{e_u}(t)$$

Contract: thread_start queue_init

$\leftarrow$ {queue_send, queue_receive}

$start(s) \not\prec_{hb} start(r) \wedge end(r) \not\prec_{hb} end(s)$



Noise injection!

Illustration of the Dynamic Validation Approach

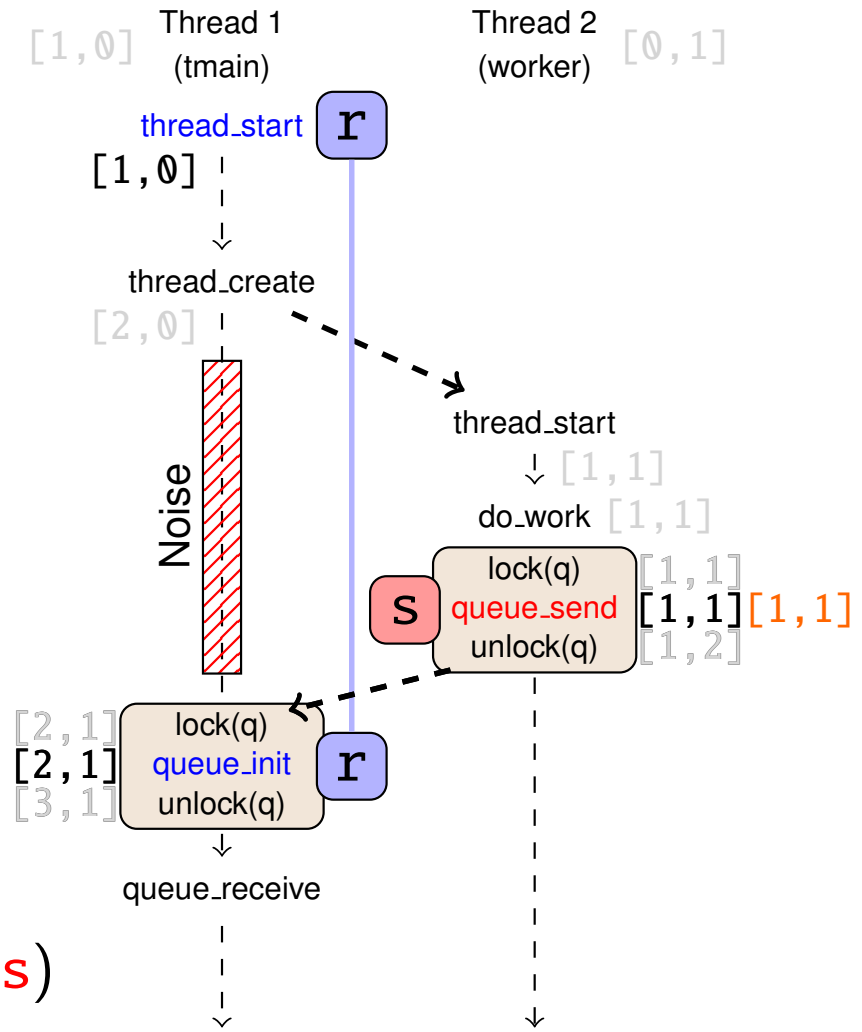
```
void tmain() {
  thread_create(worker,input);
  queue_init();
  result = queue_receive();
}
void worker(void* data) {
  do_work();
  queue_send(result);
}
```

$$e_t \prec_{hb} e_u \sim VC_{e_t}(t) \leq VC_{e_u}(t)$$

Contract: thread_start queue_init

$\leftarrow$ {queue_send, queue_receive}

$start(s) \not\prec_{hb} start(r) \wedge end(r) \not\prec_{hb} end(s)$



Noise injection!

Illustration of the Dynamic Validation Approach

```
void tmain() {
  thread_create(worker,input);
  queue_init();
  result = queue_receive();
}
void worker(void* data) {
  do_work();
  queue_send(result);
}
```

$$e_t \prec_{hb} e_u \sim VC_{e_t}(t) \leq VC_{e_u}(t)$$

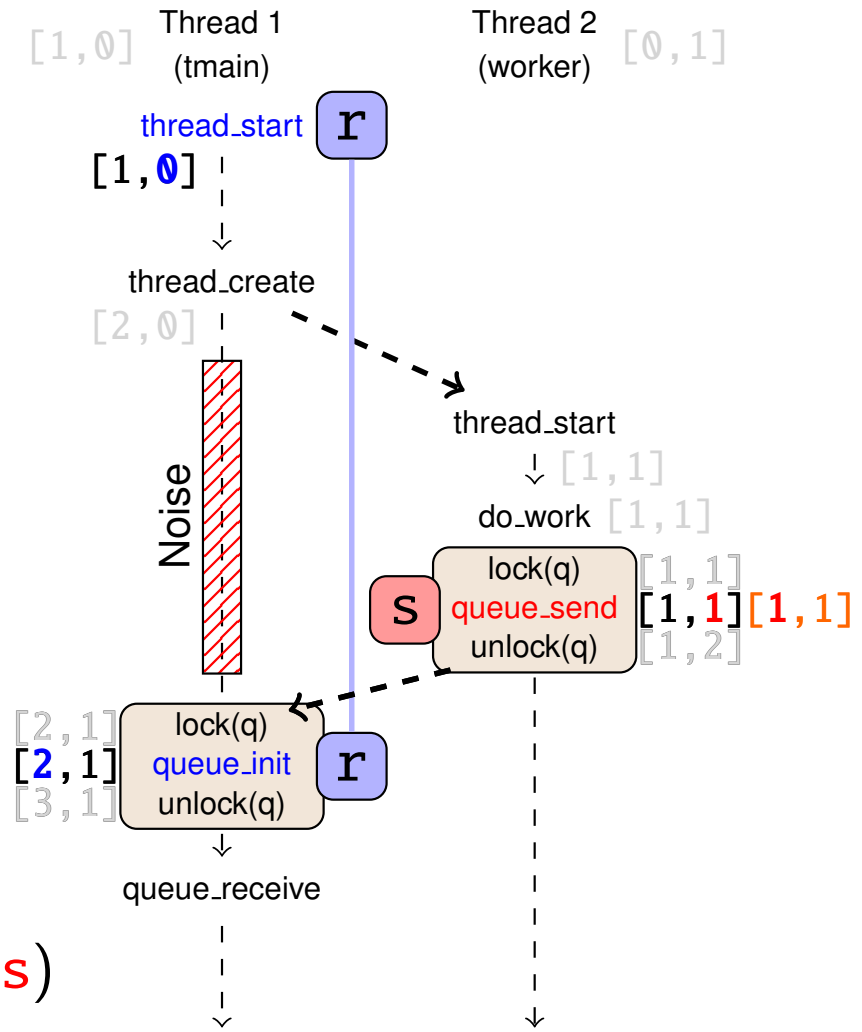
Contract: thread_start queue_init

$\leftarrow$ {queue_send, queue_receive}

$start(s) \not\prec_{hb} start(r) \wedge end(r) \not\prec_{hb} end(s)$

$\neg(1 \leq 0) \quad \wedge \quad \neg(2 \leq 1)$

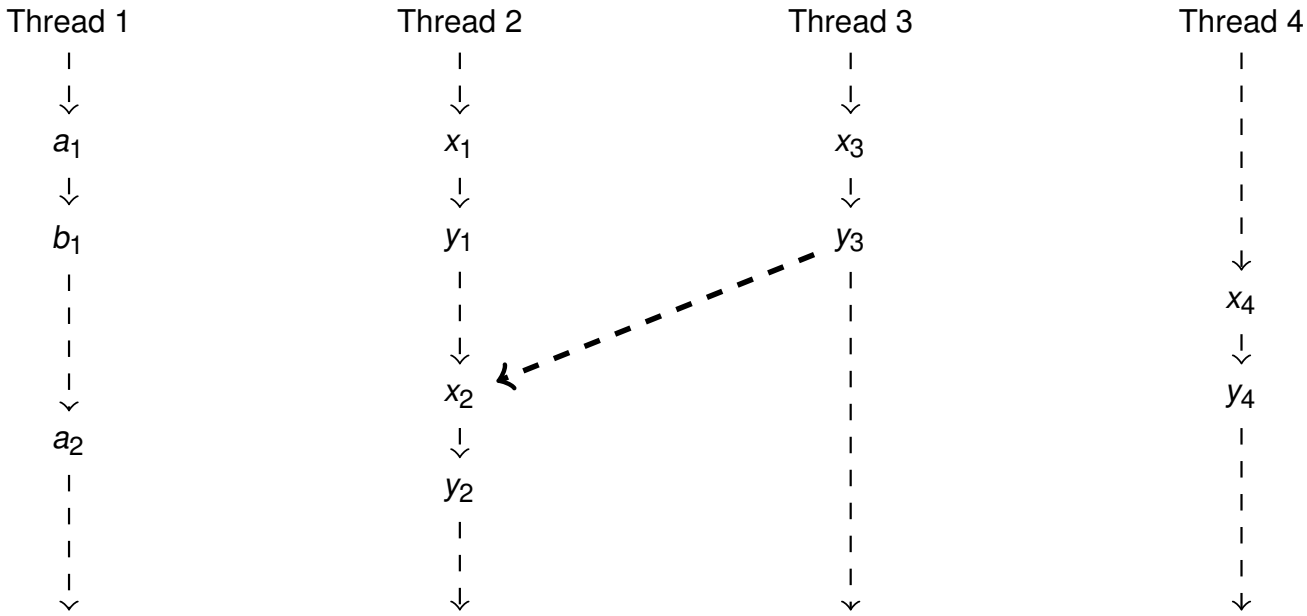
Contract violated!



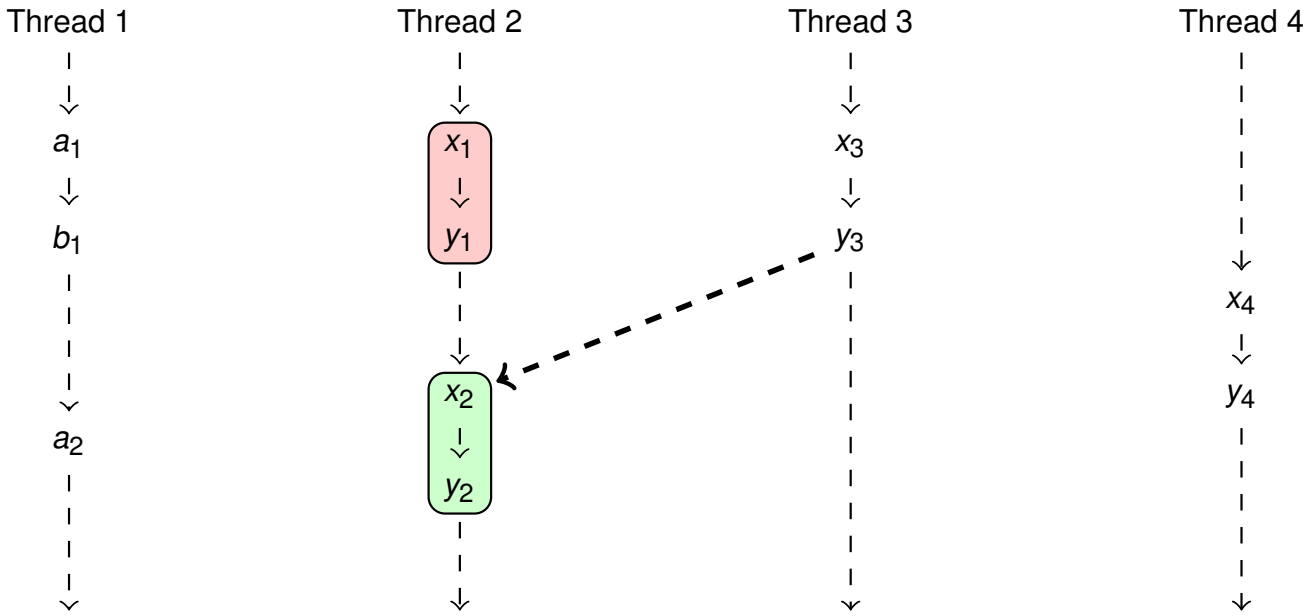
Noise injection!

Discarding Spoilers

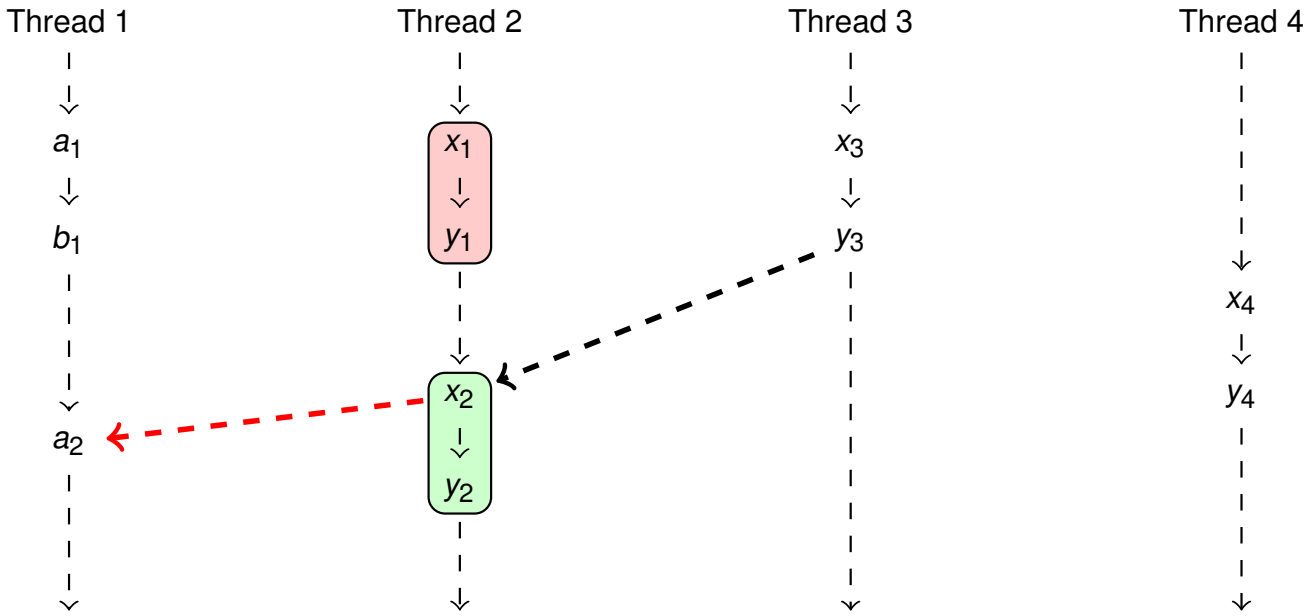
Discarding Spoilers



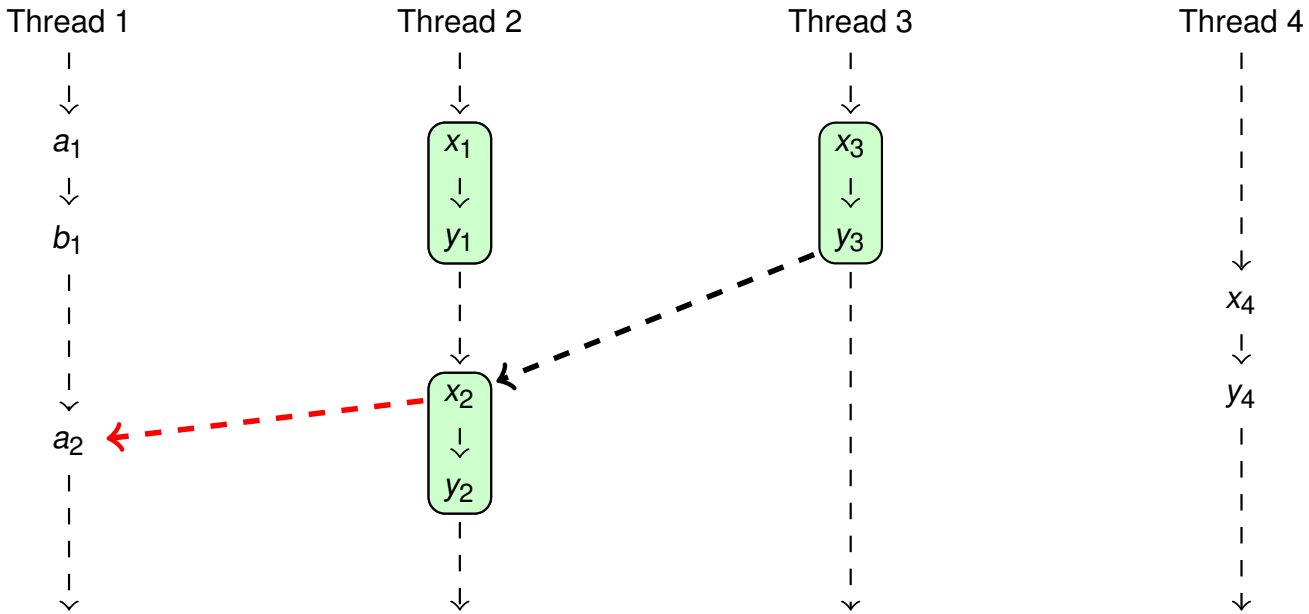
Discarding Spoilers



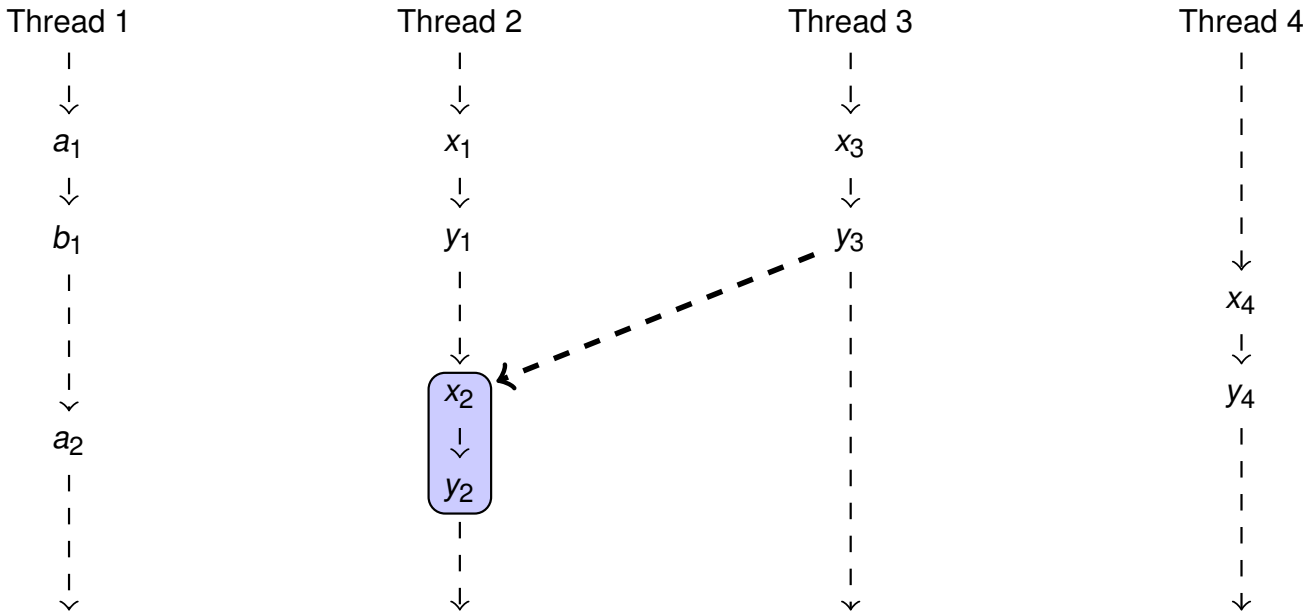
Discarding Spoilers



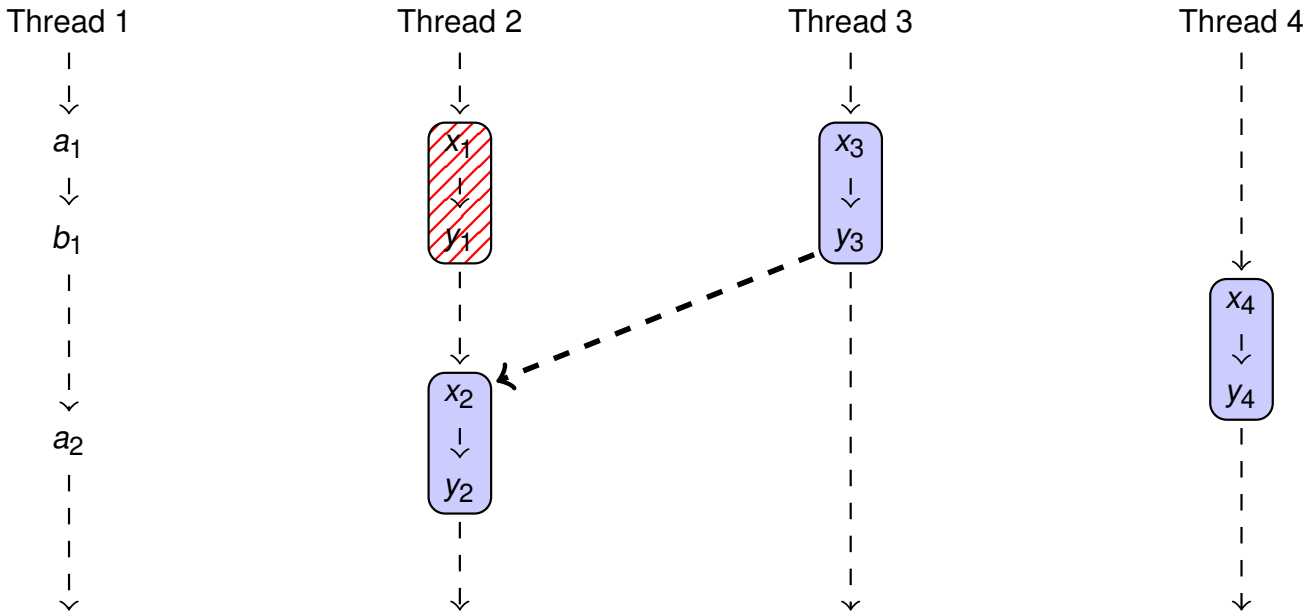
Discarding Spoilers



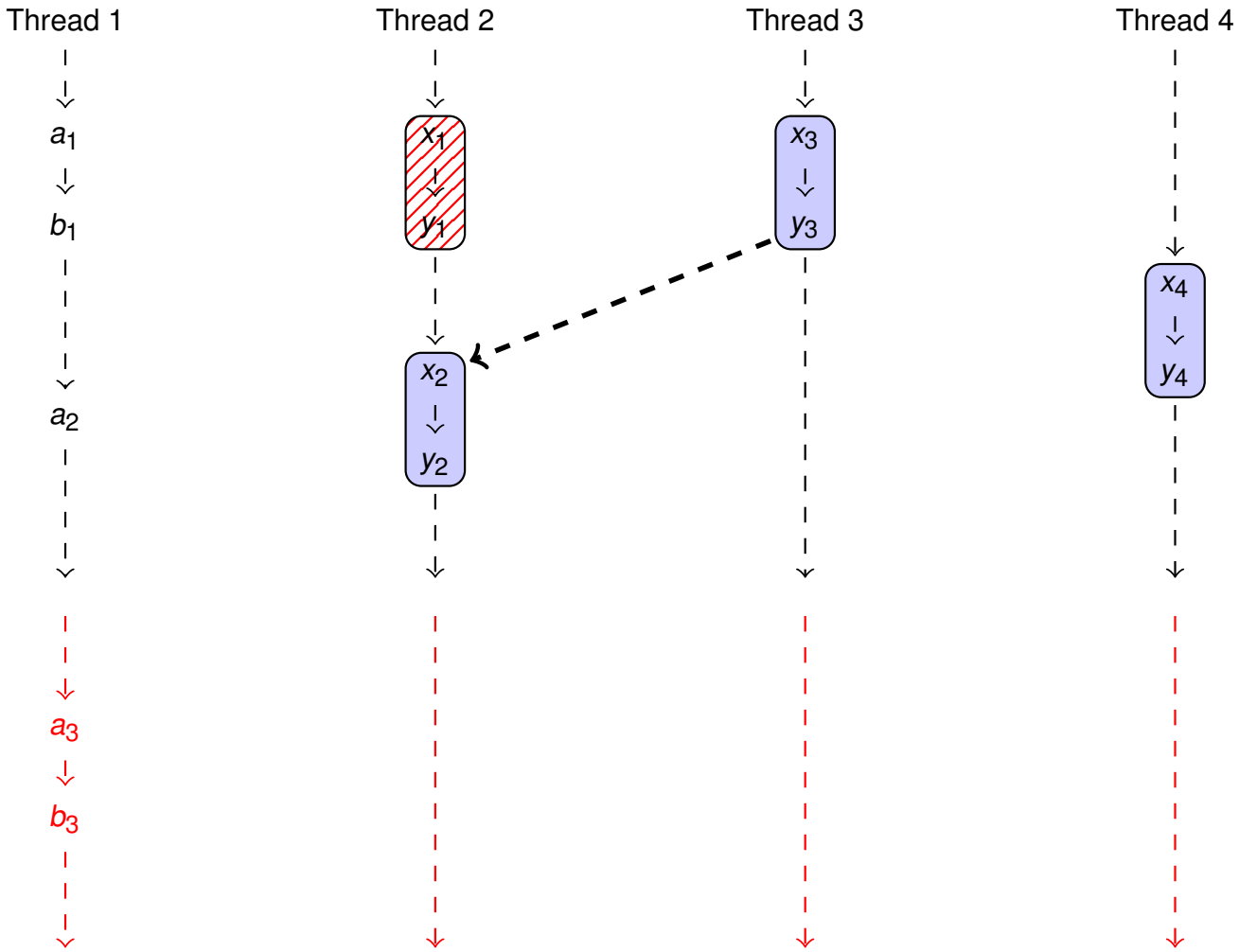
Discarding Spoilers



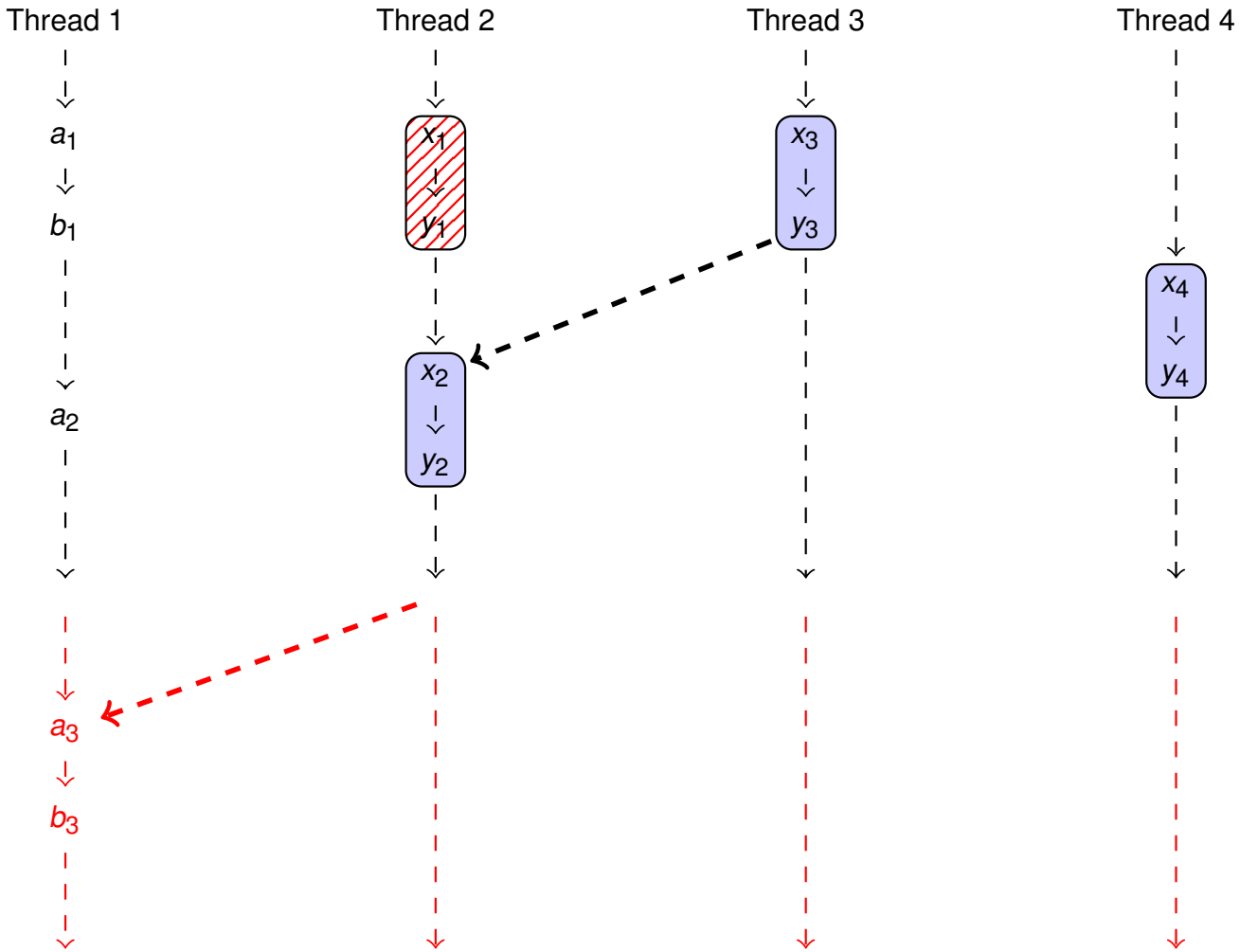
Discarding Spoilers



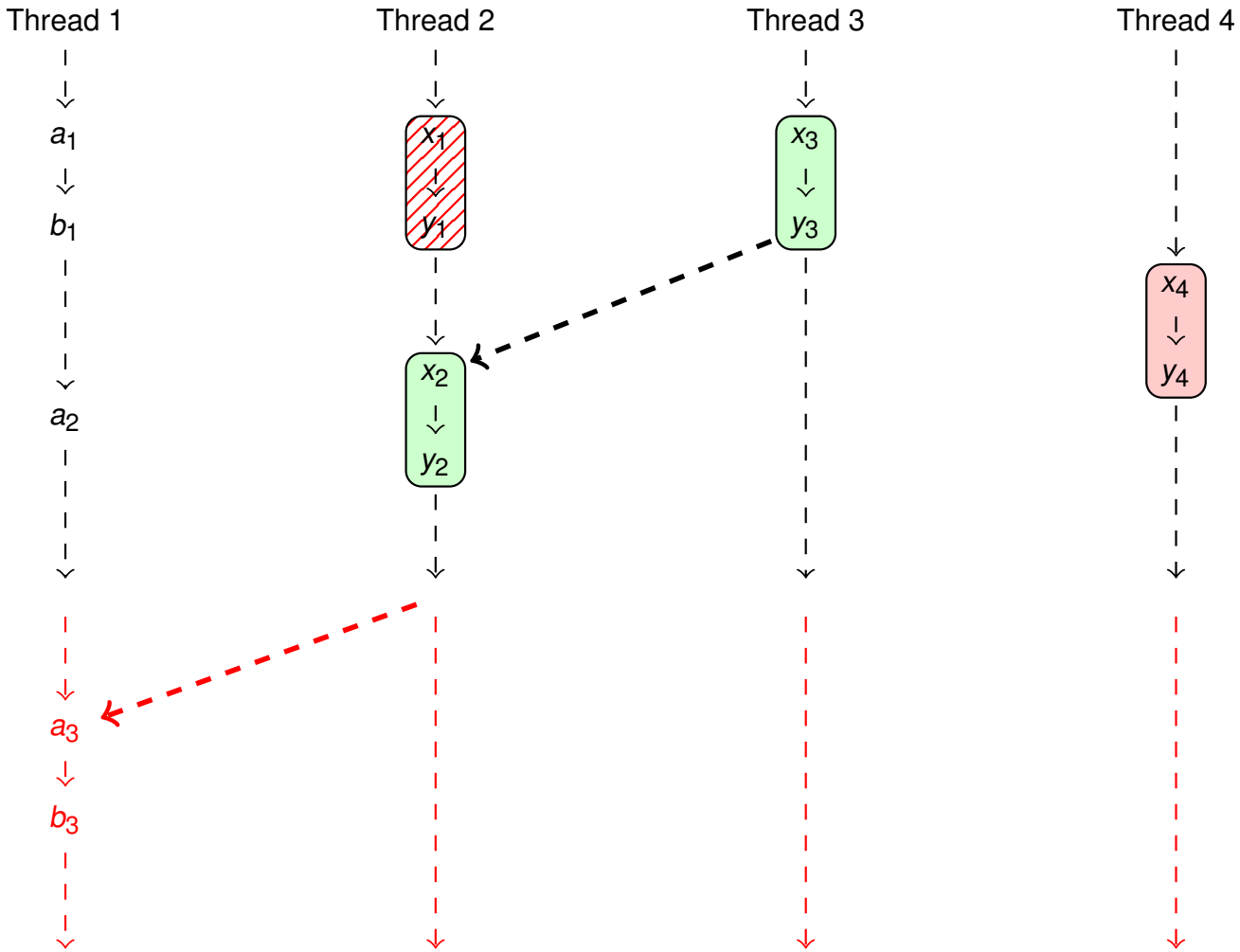
Discarding Spoilers



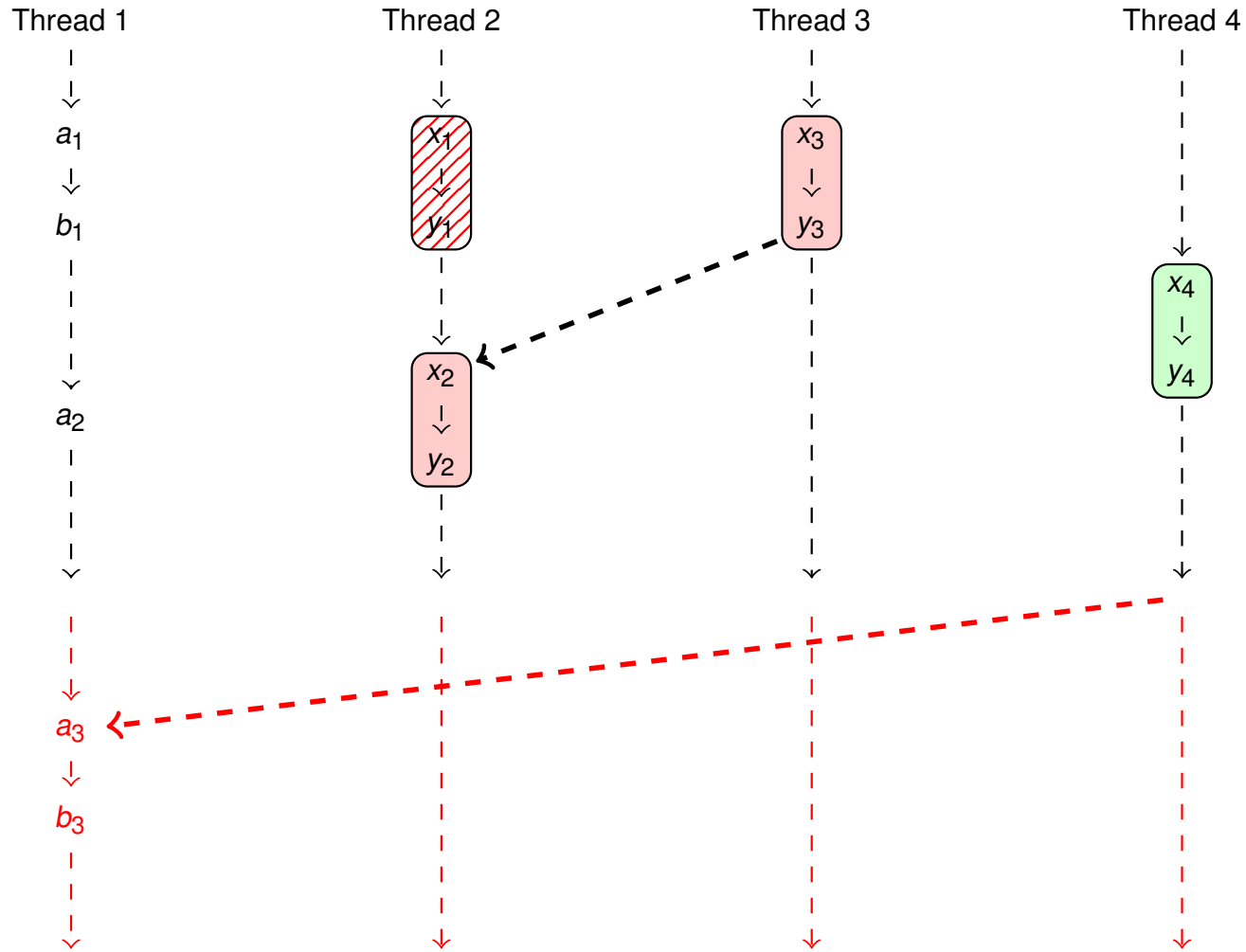
Discarding Spoilers



Discarding Spoilers

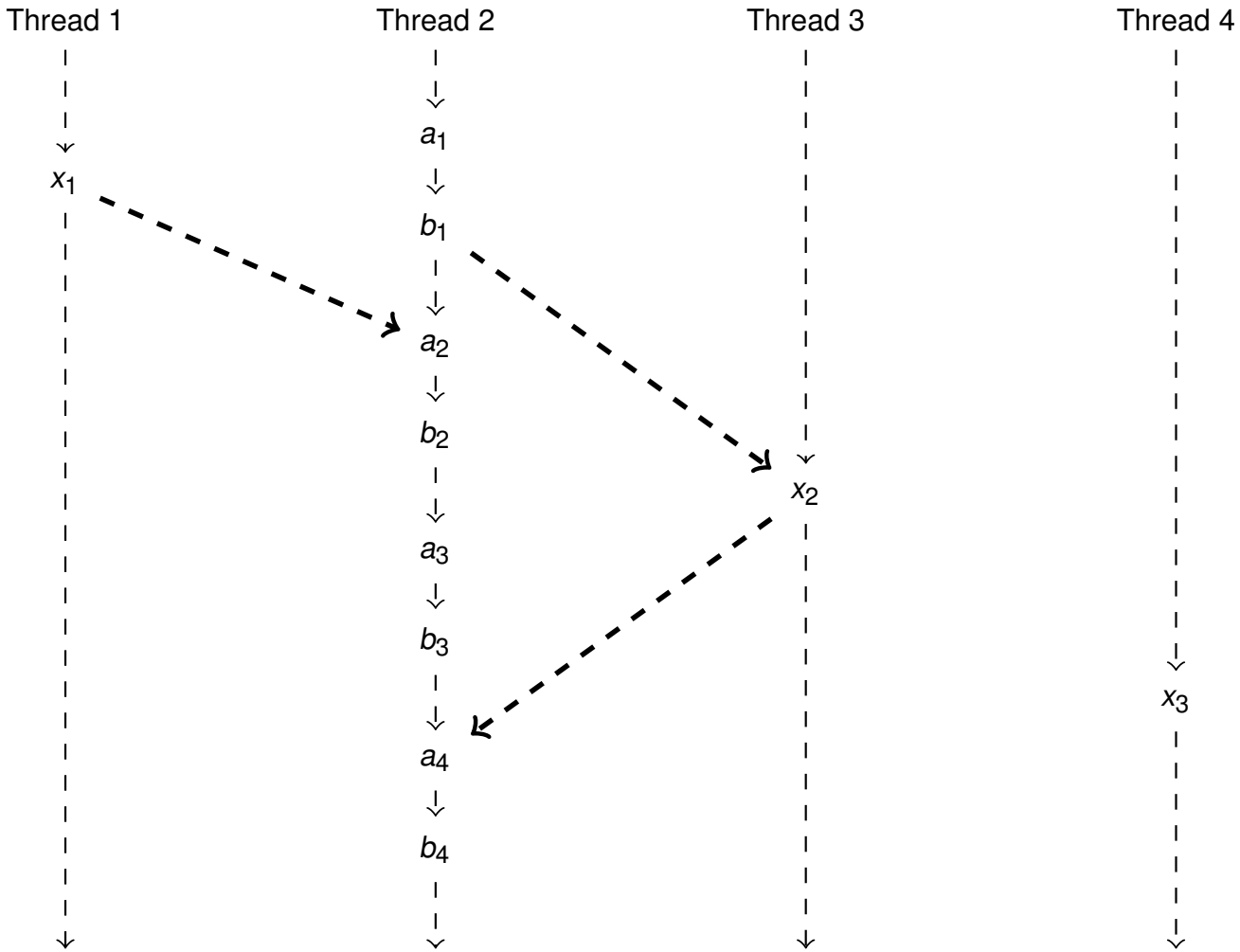


Discarding Spoilers

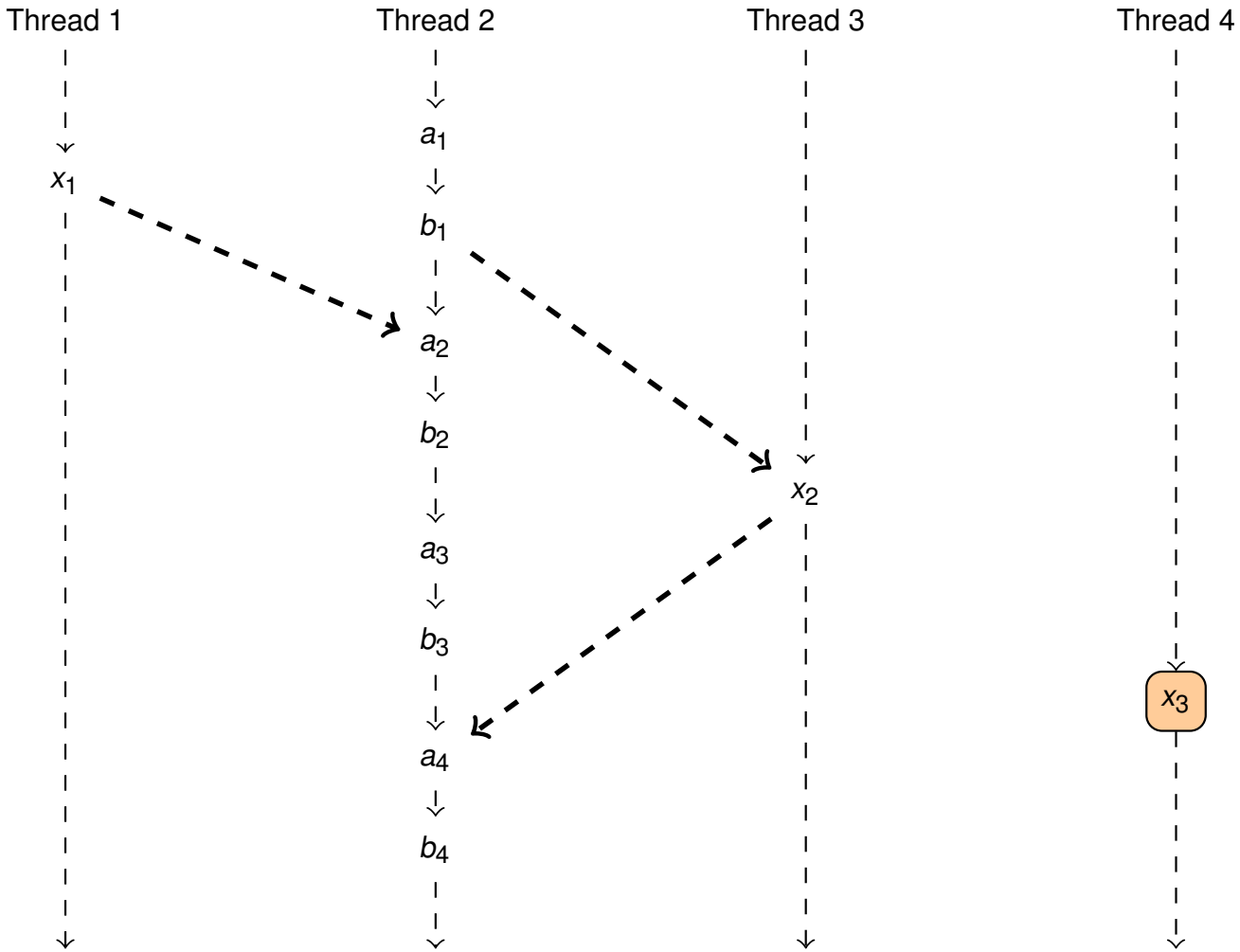


Discarding Targets

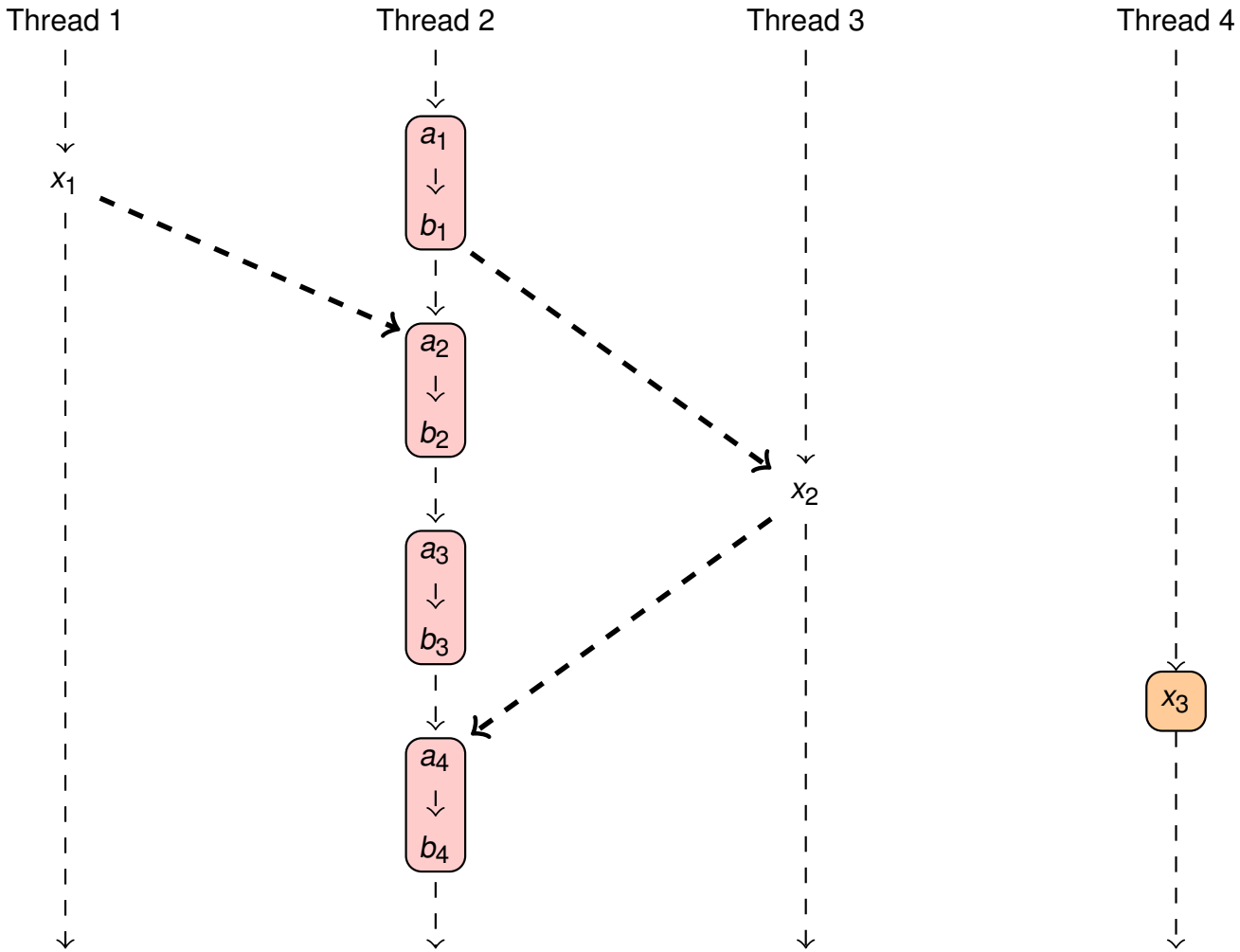
Discarding Targets



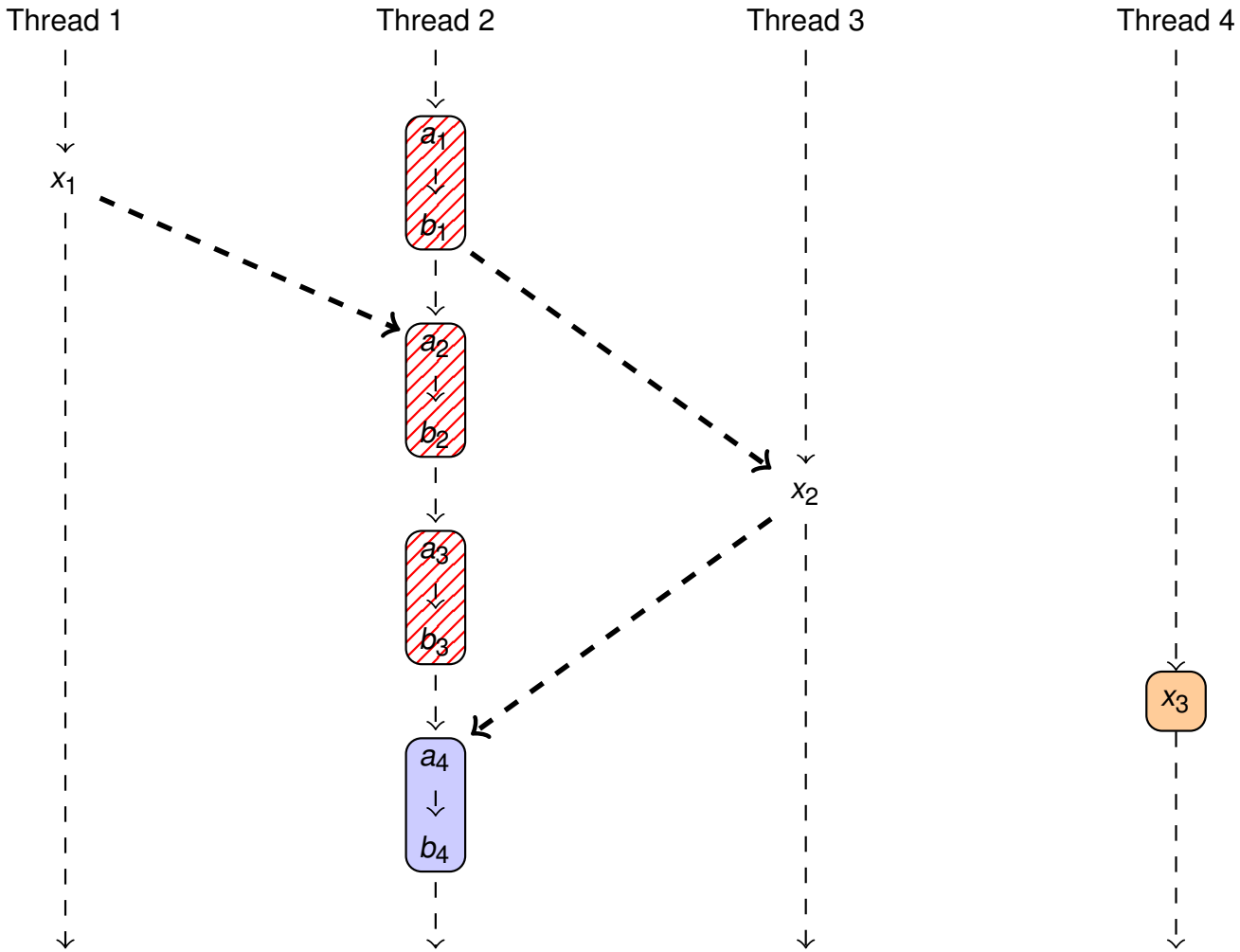
Discarding Targets



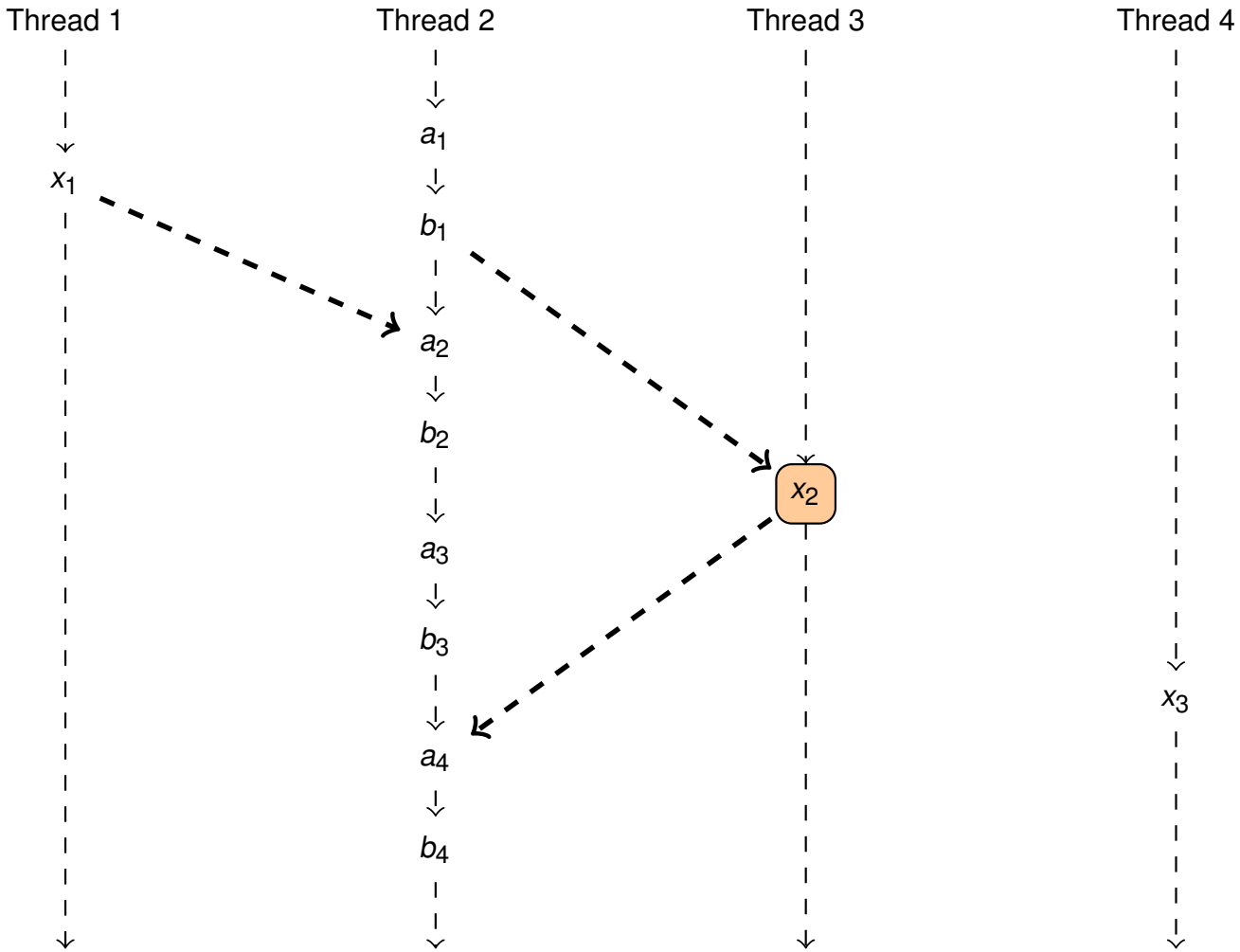
Discarding Targets



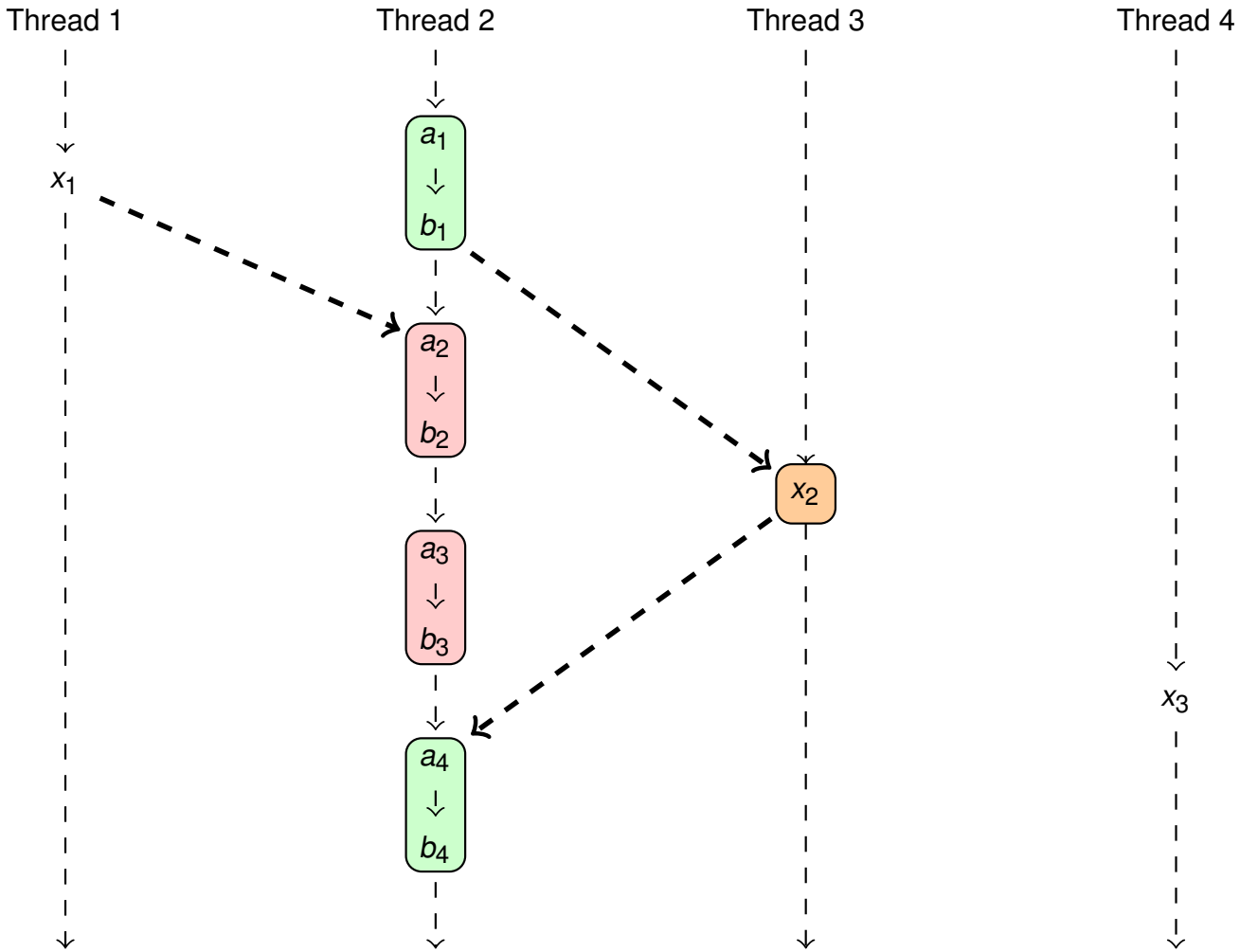
Discarding Targets



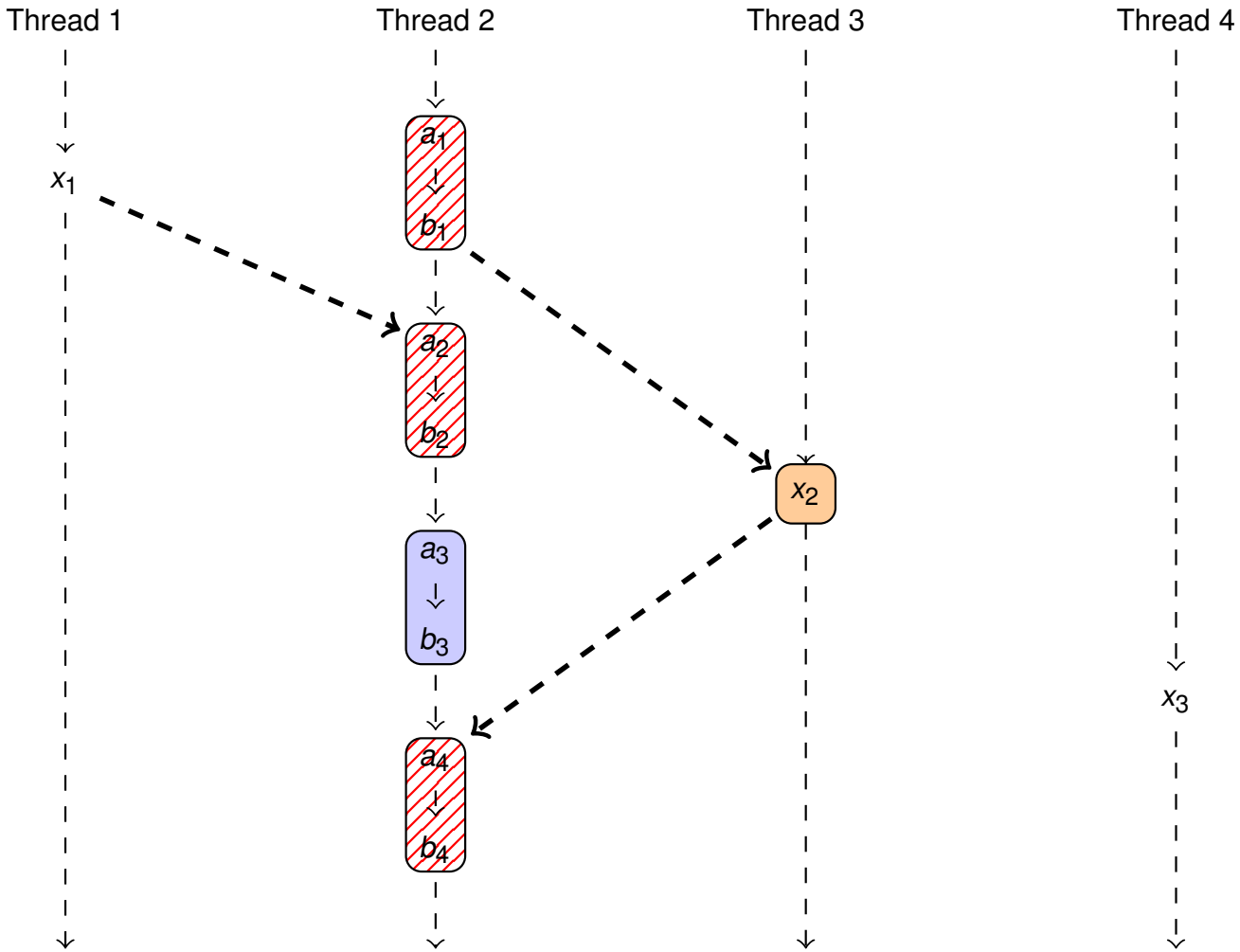
Discarding Targets



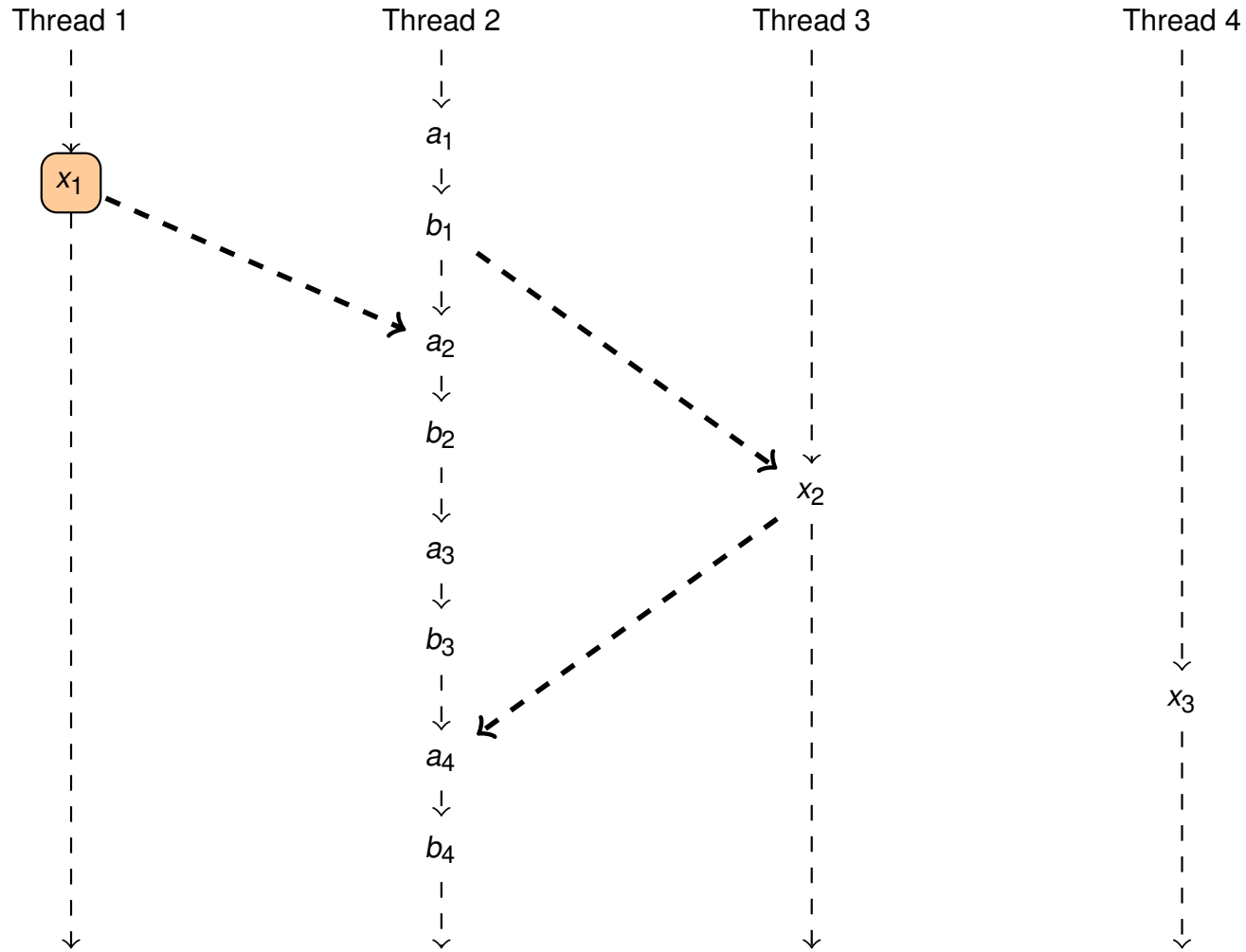
Discarding Targets



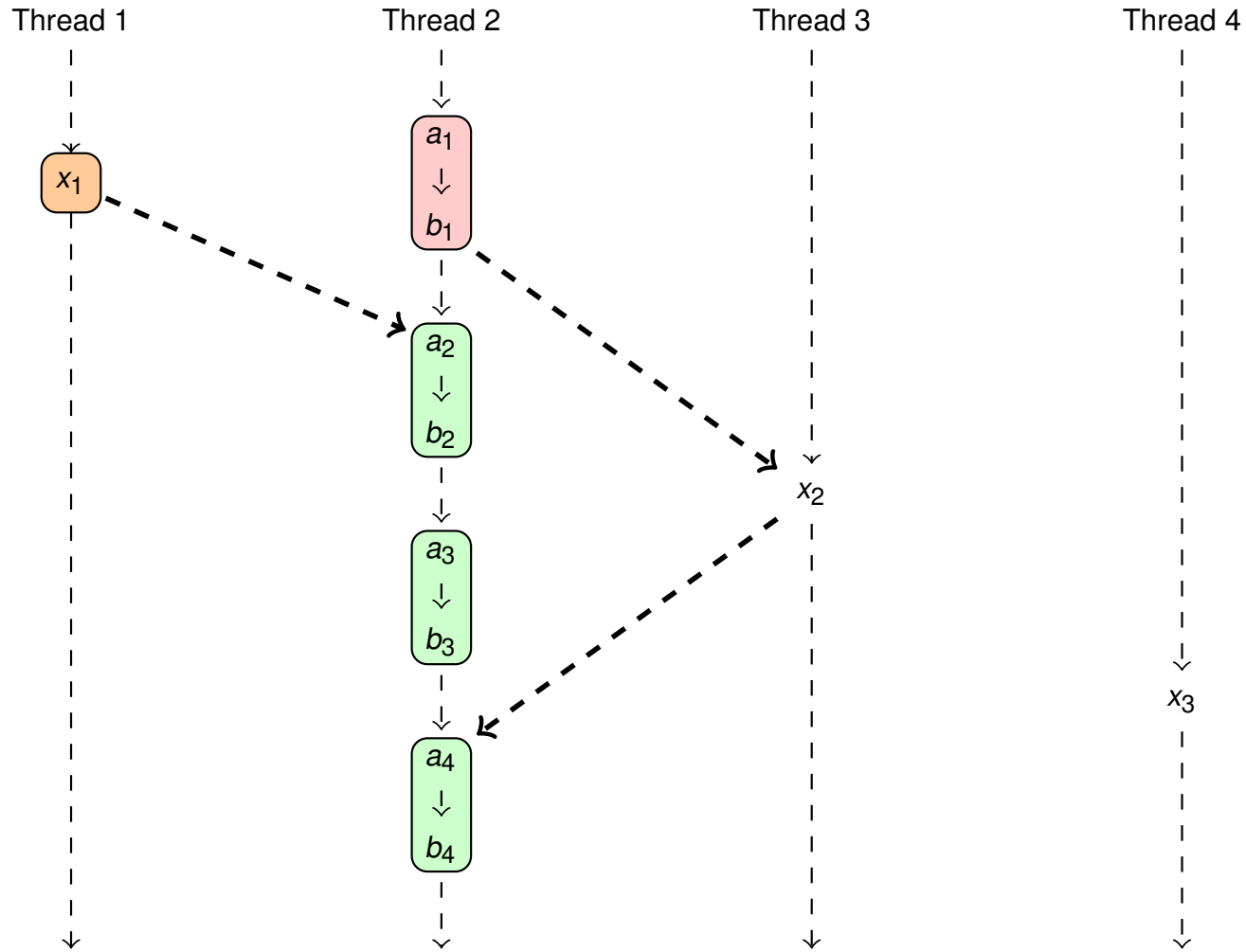
Discarding Targets



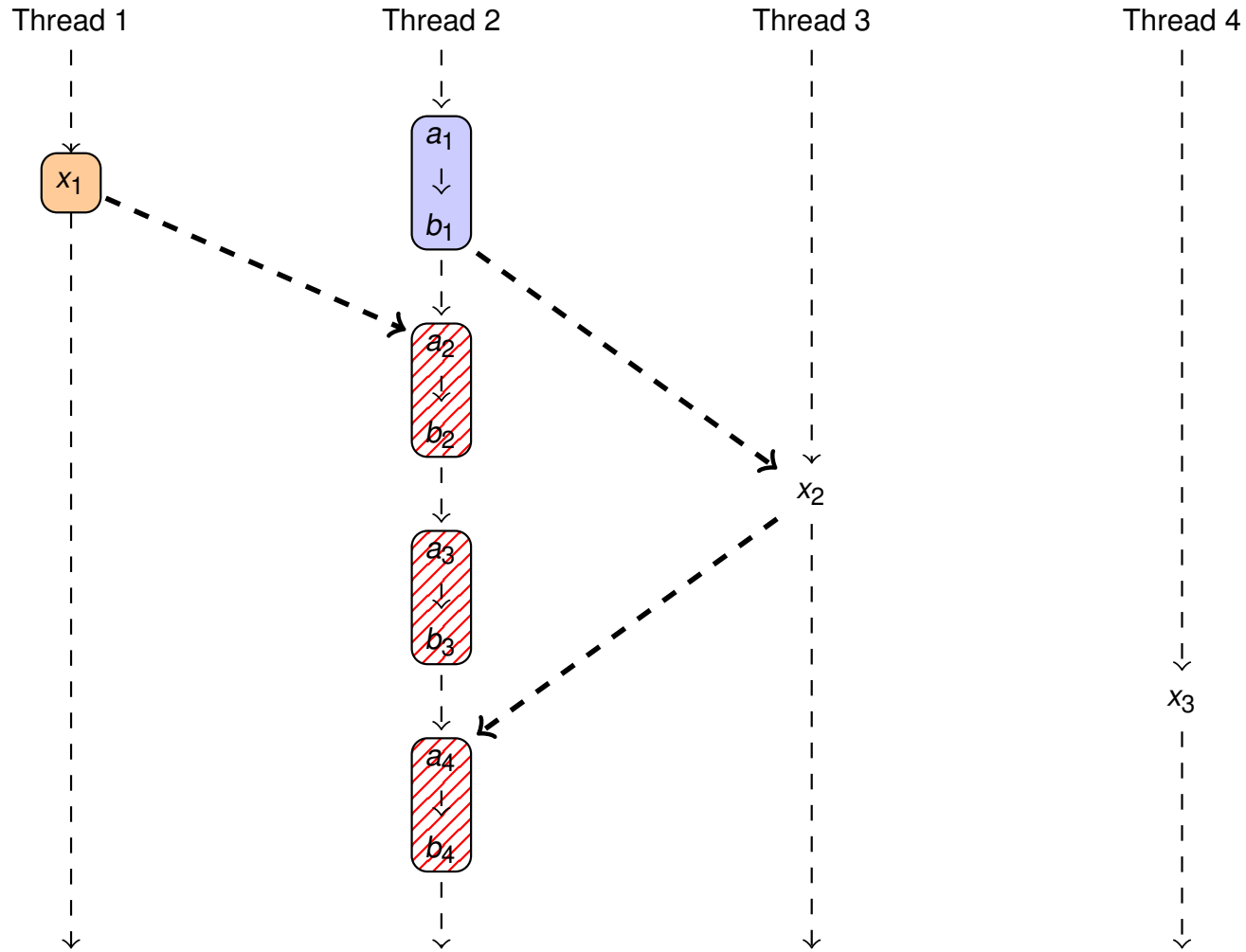
Discarding Targets



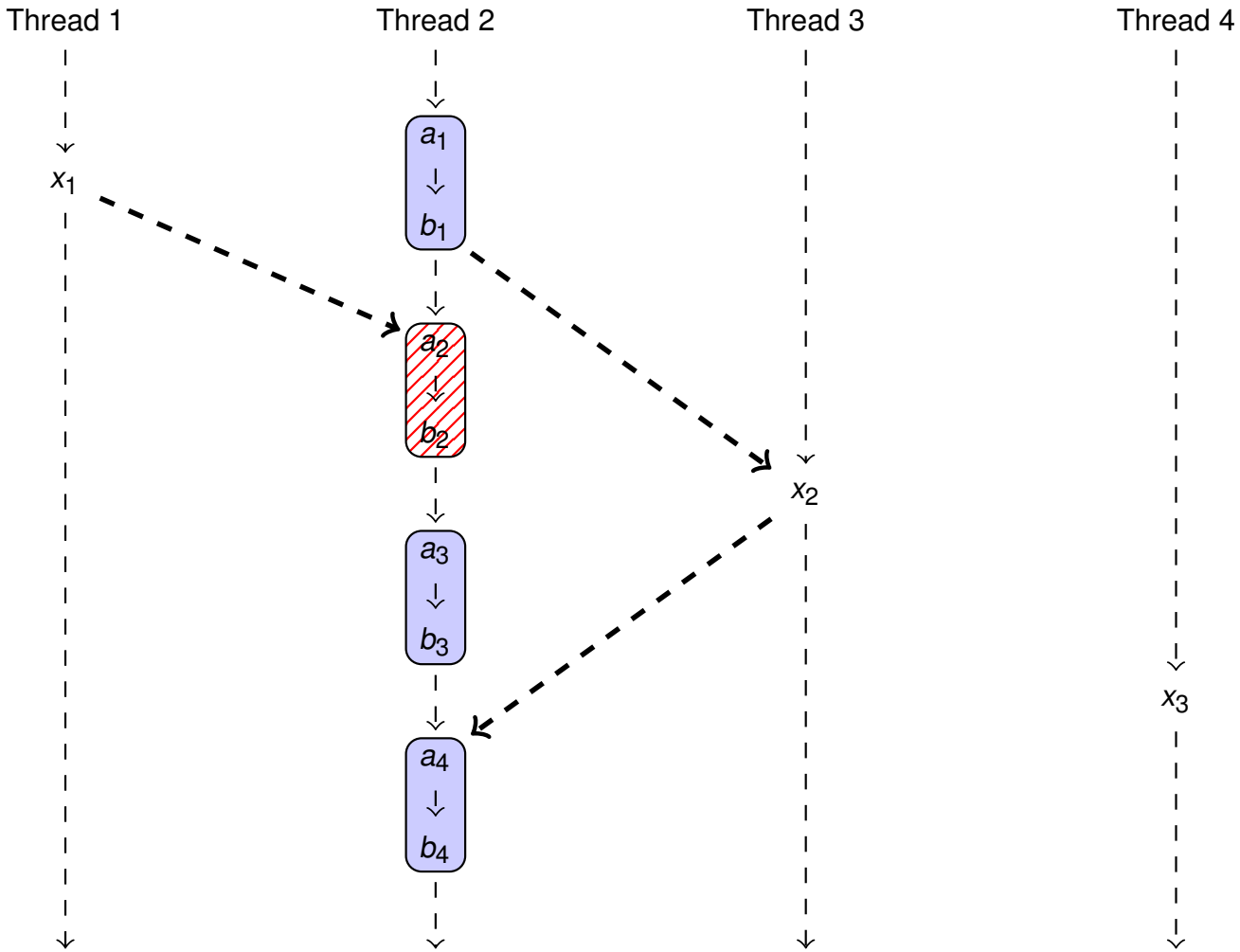
Discarding Targets



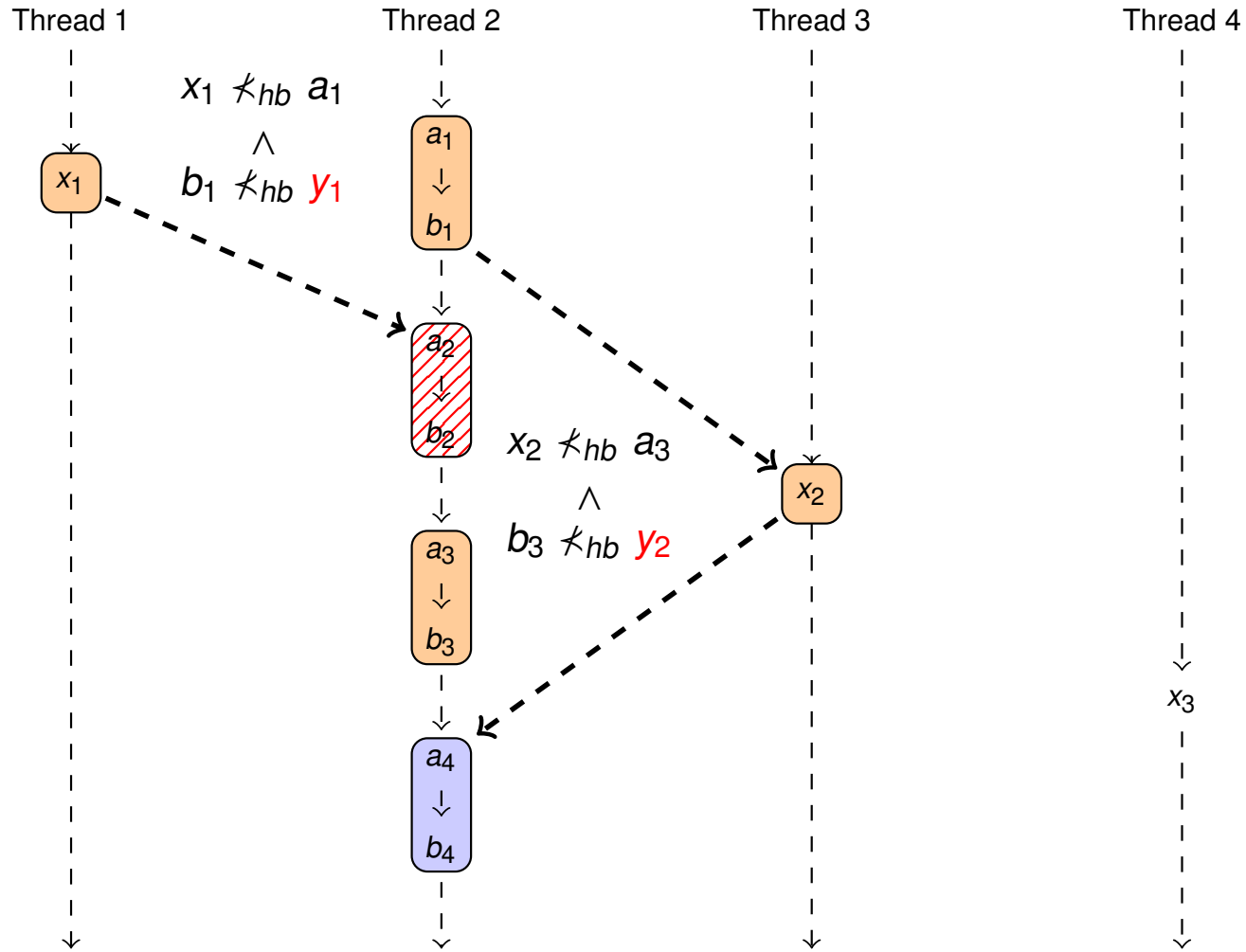
Discarding Targets



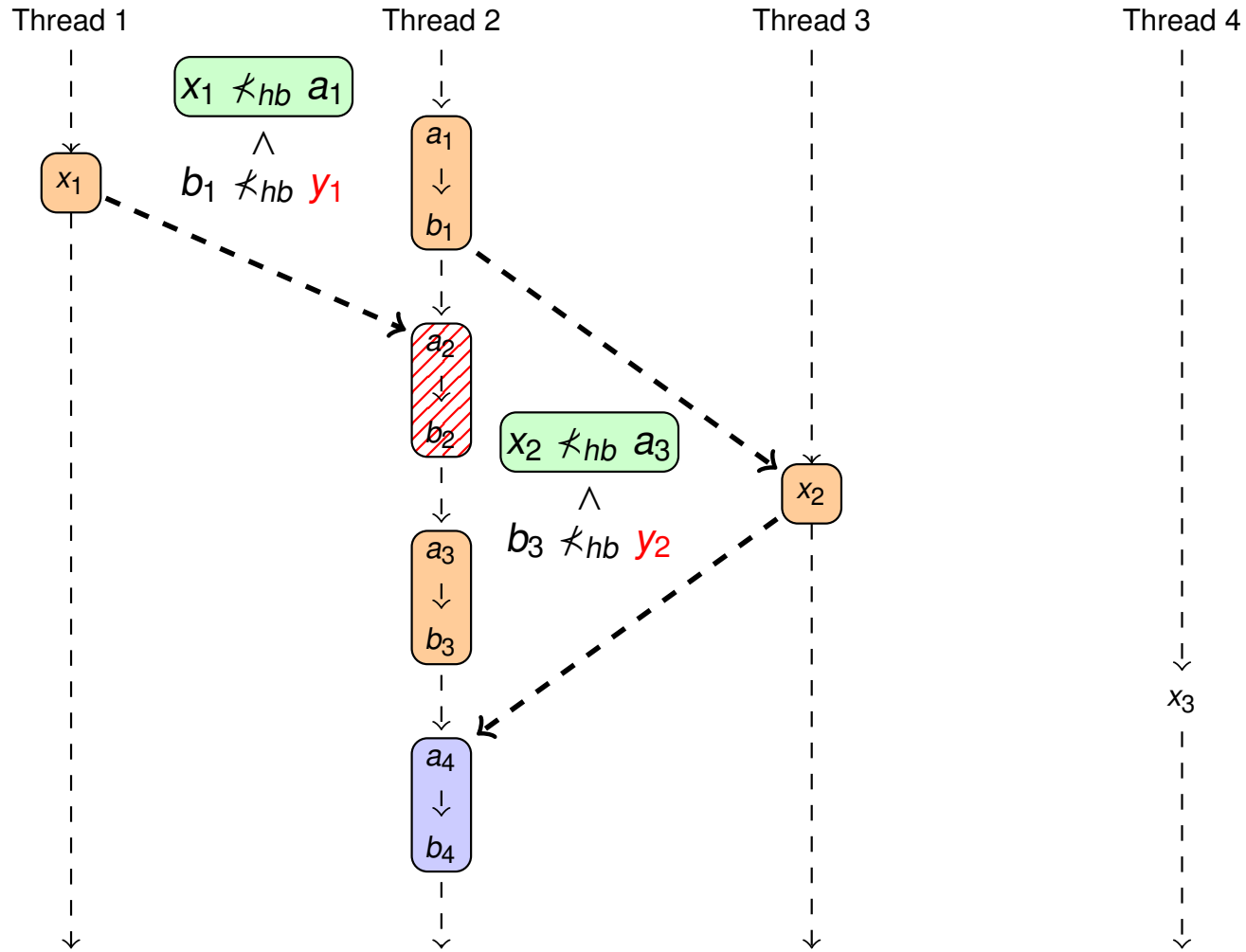
Discarding Targets



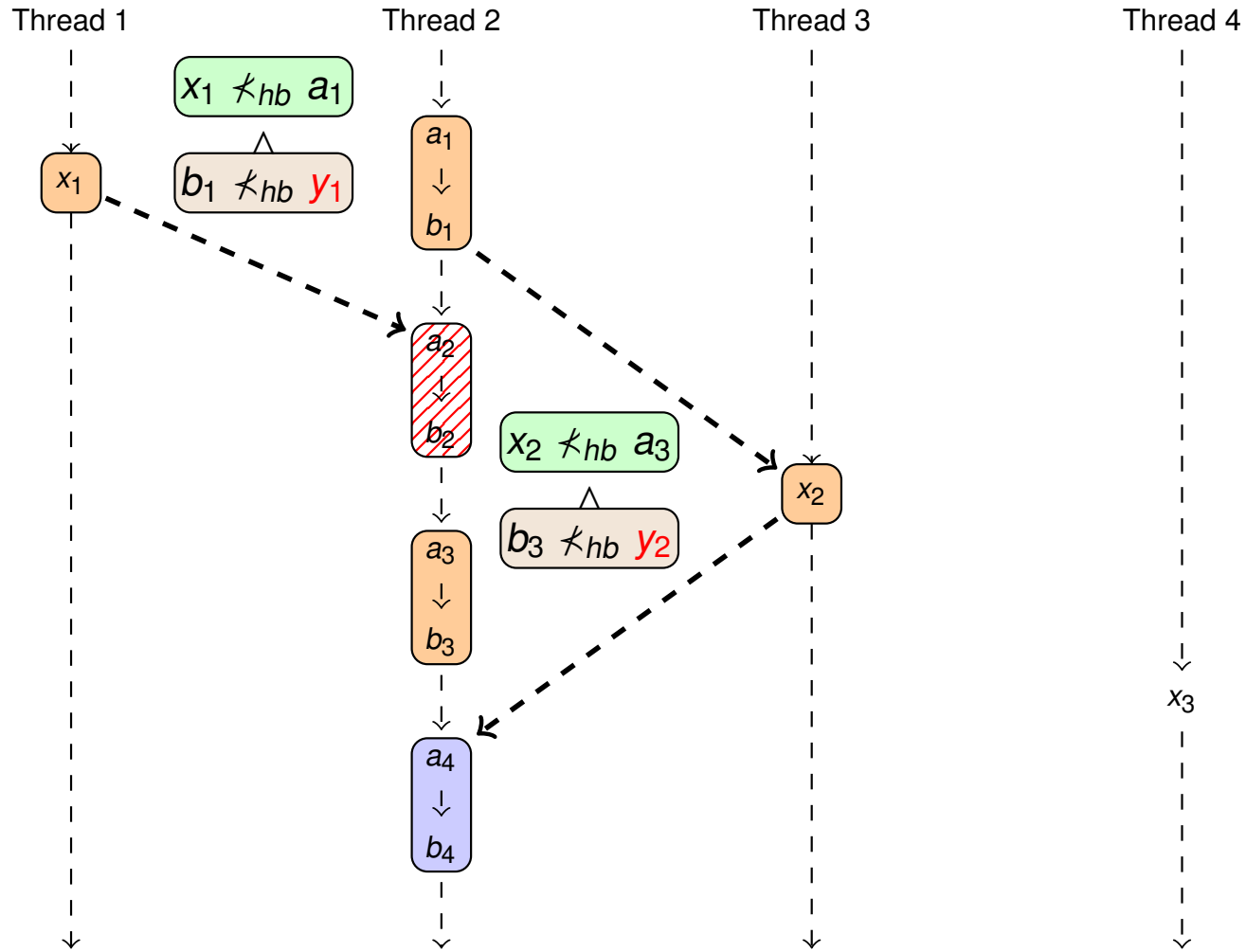
Discarding Targets



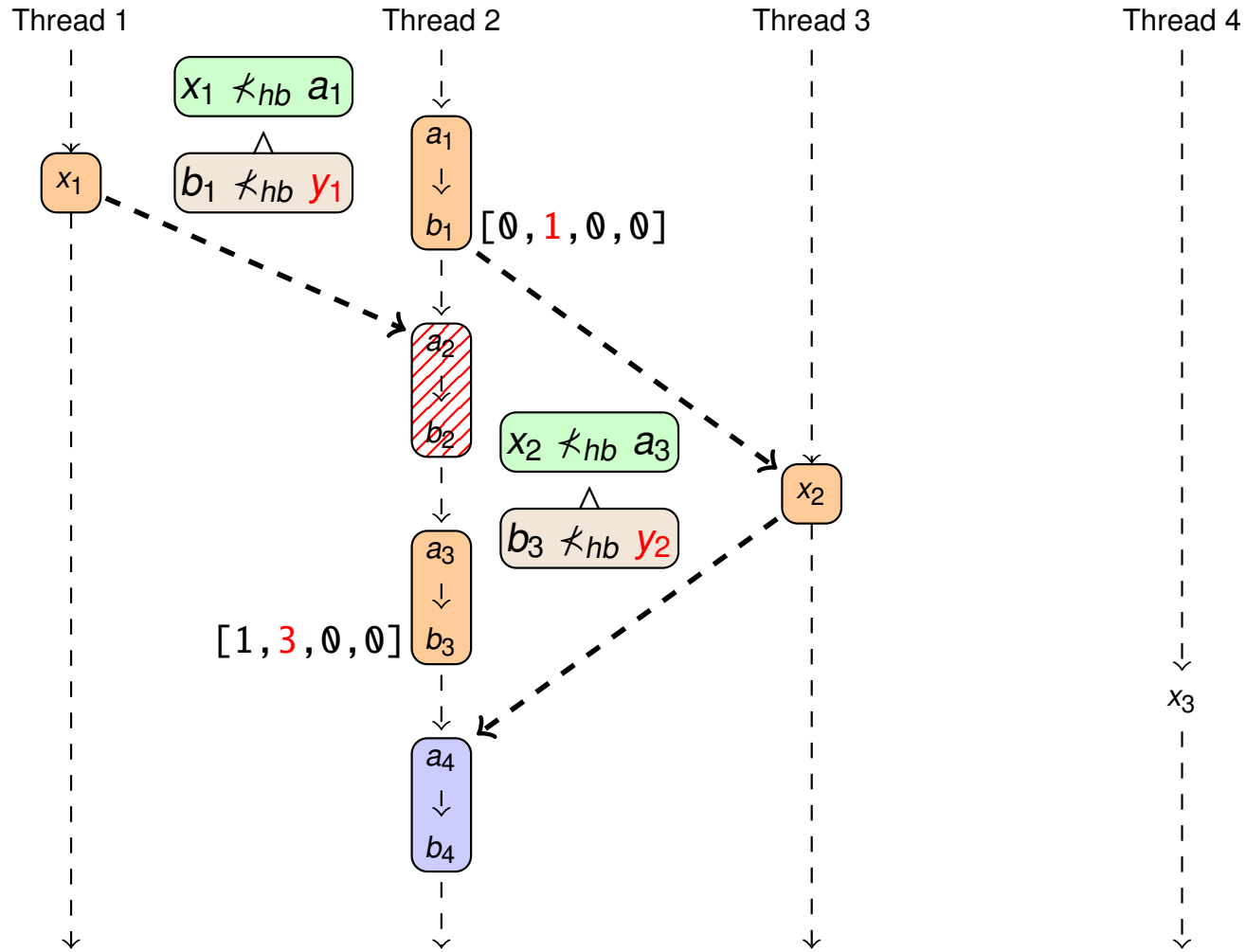
Discarding Targets



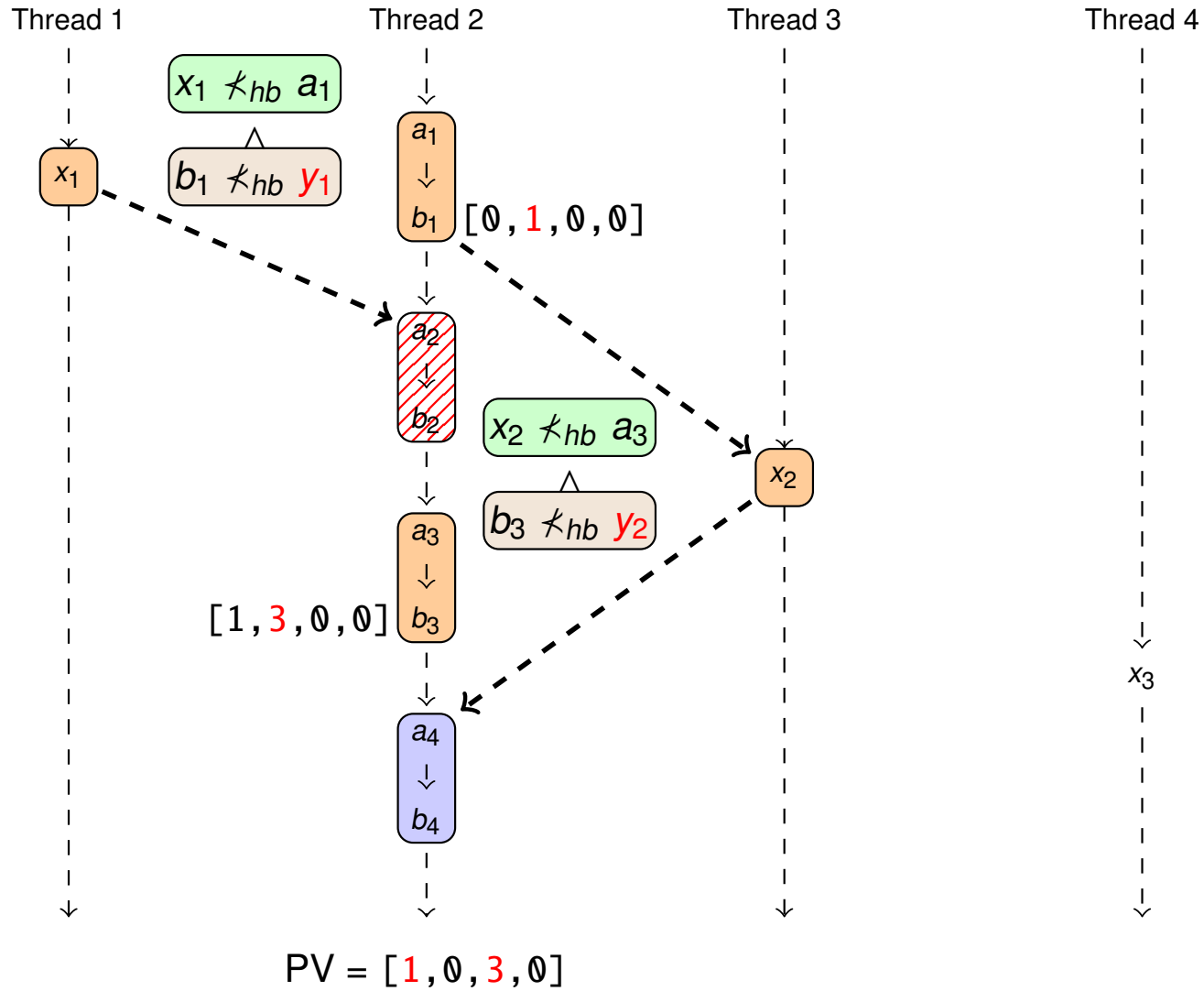
Discarding Targets



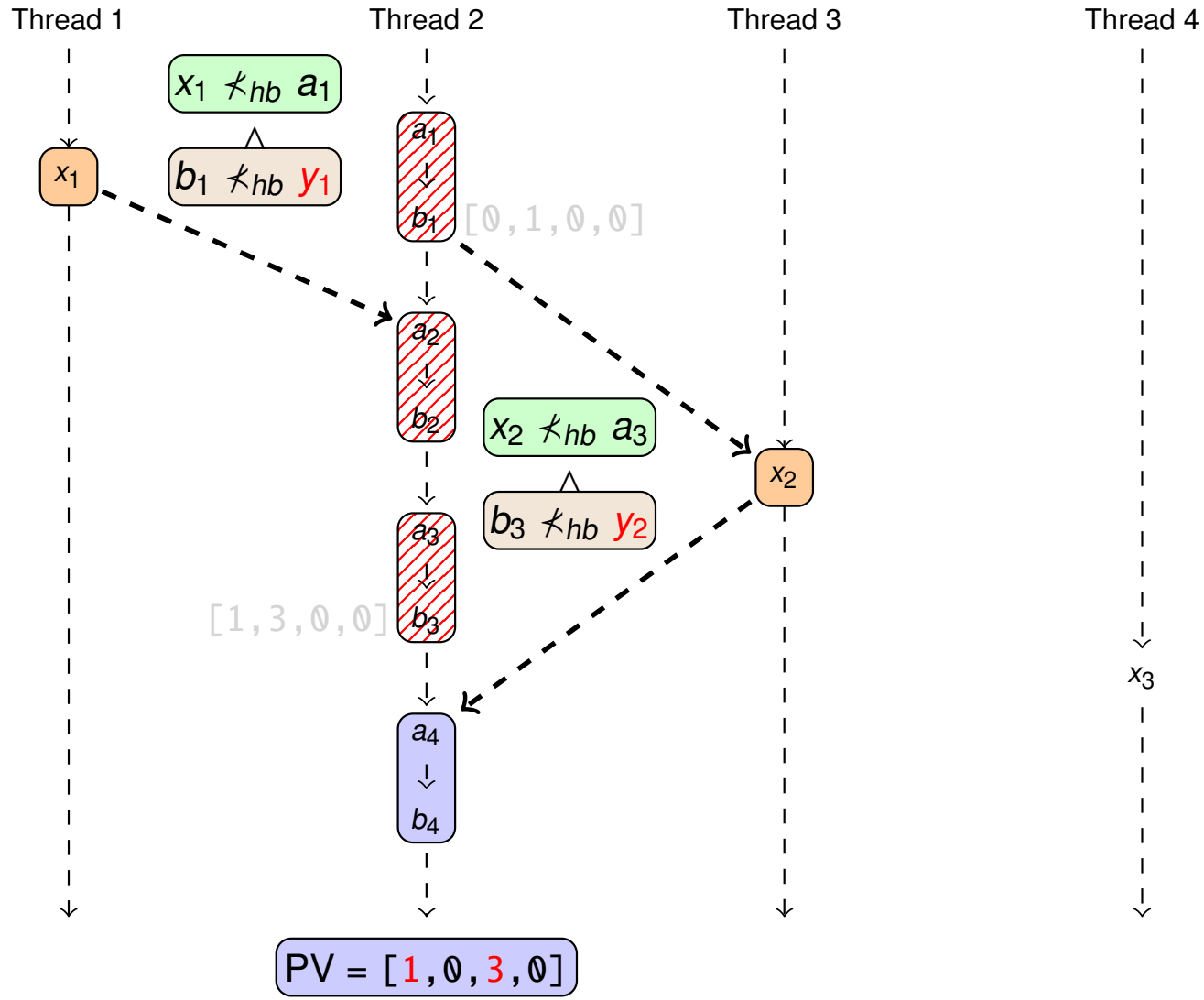
Discarding Targets



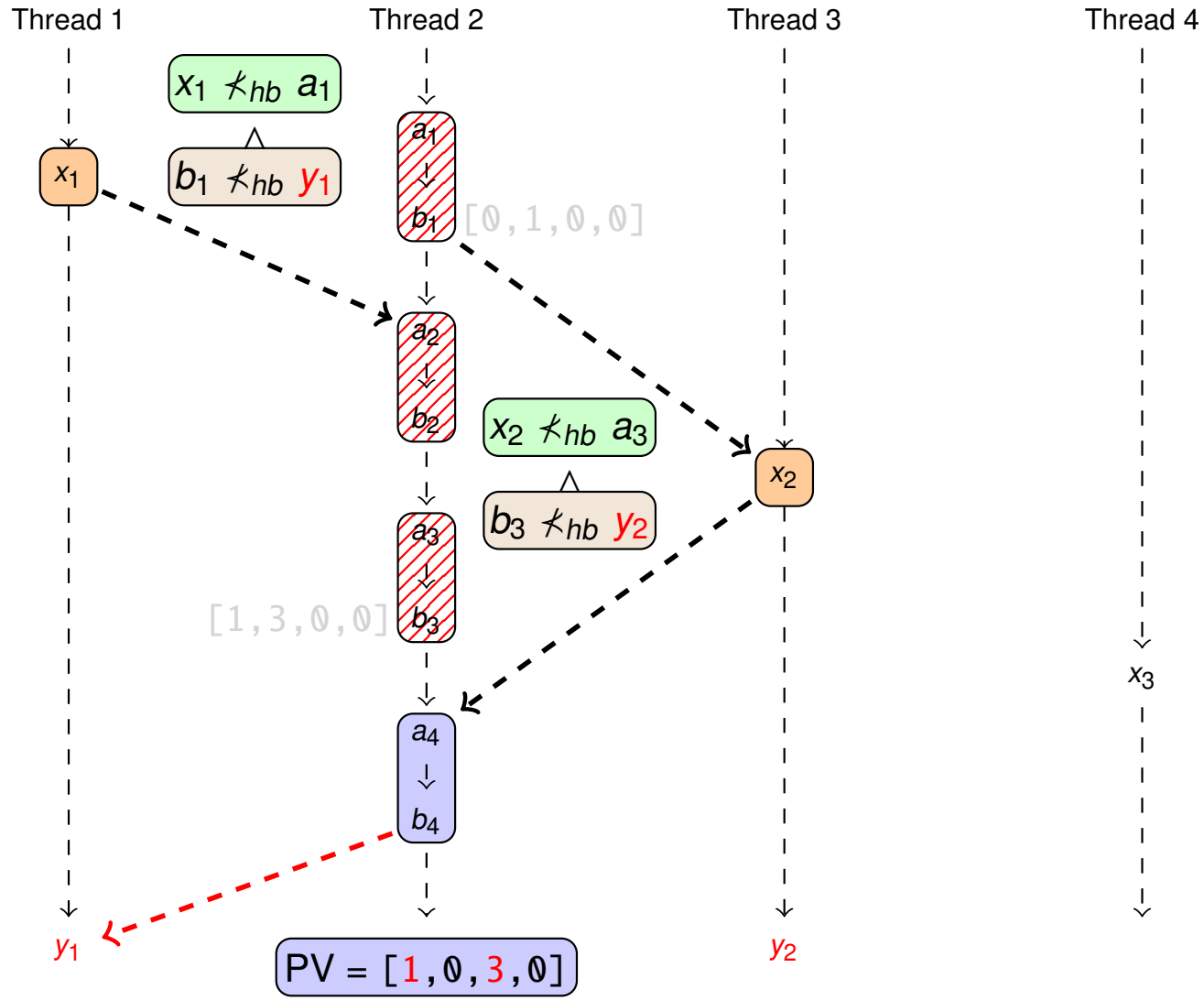
Discarding Targets



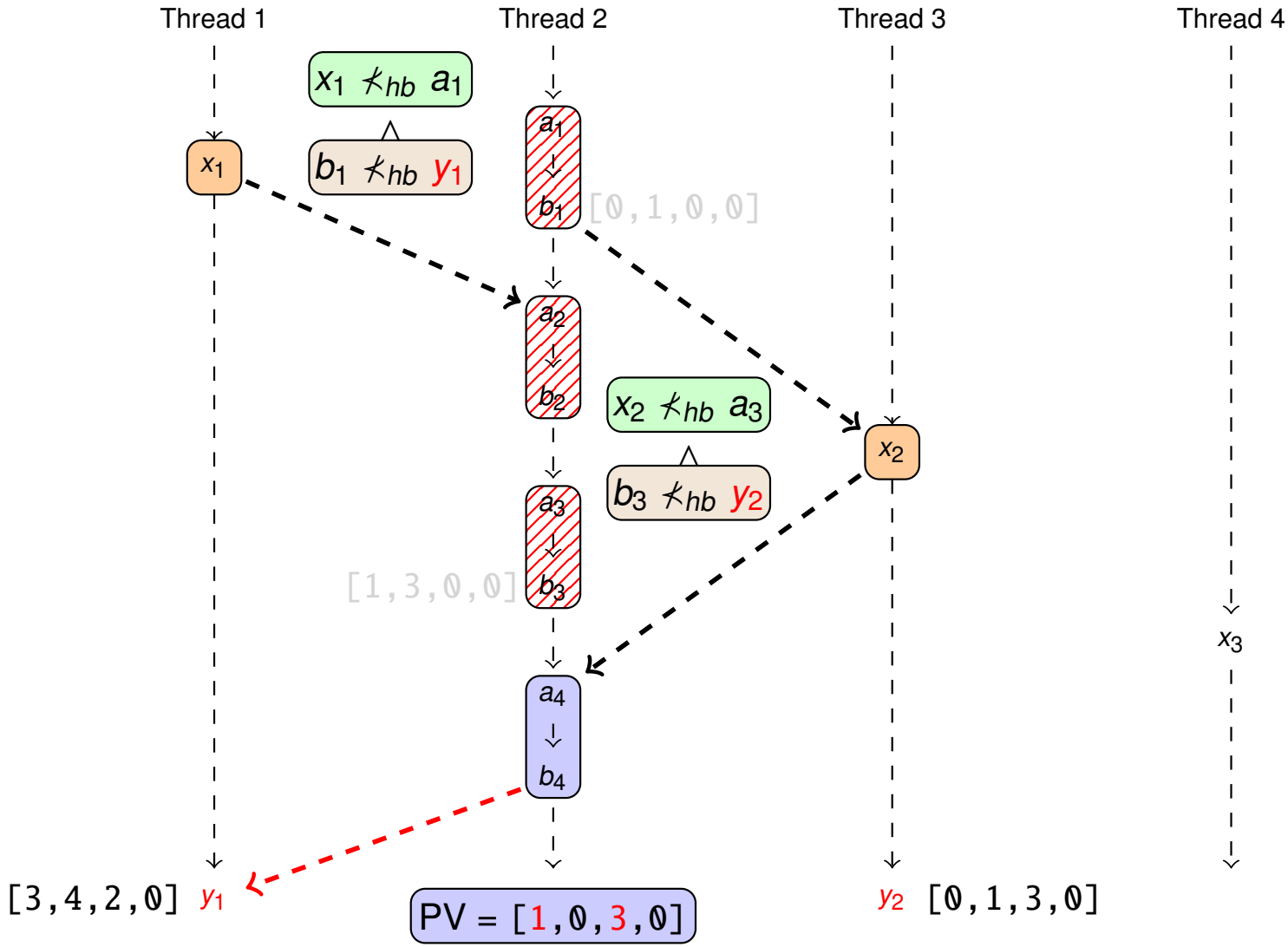
Discarding Targets



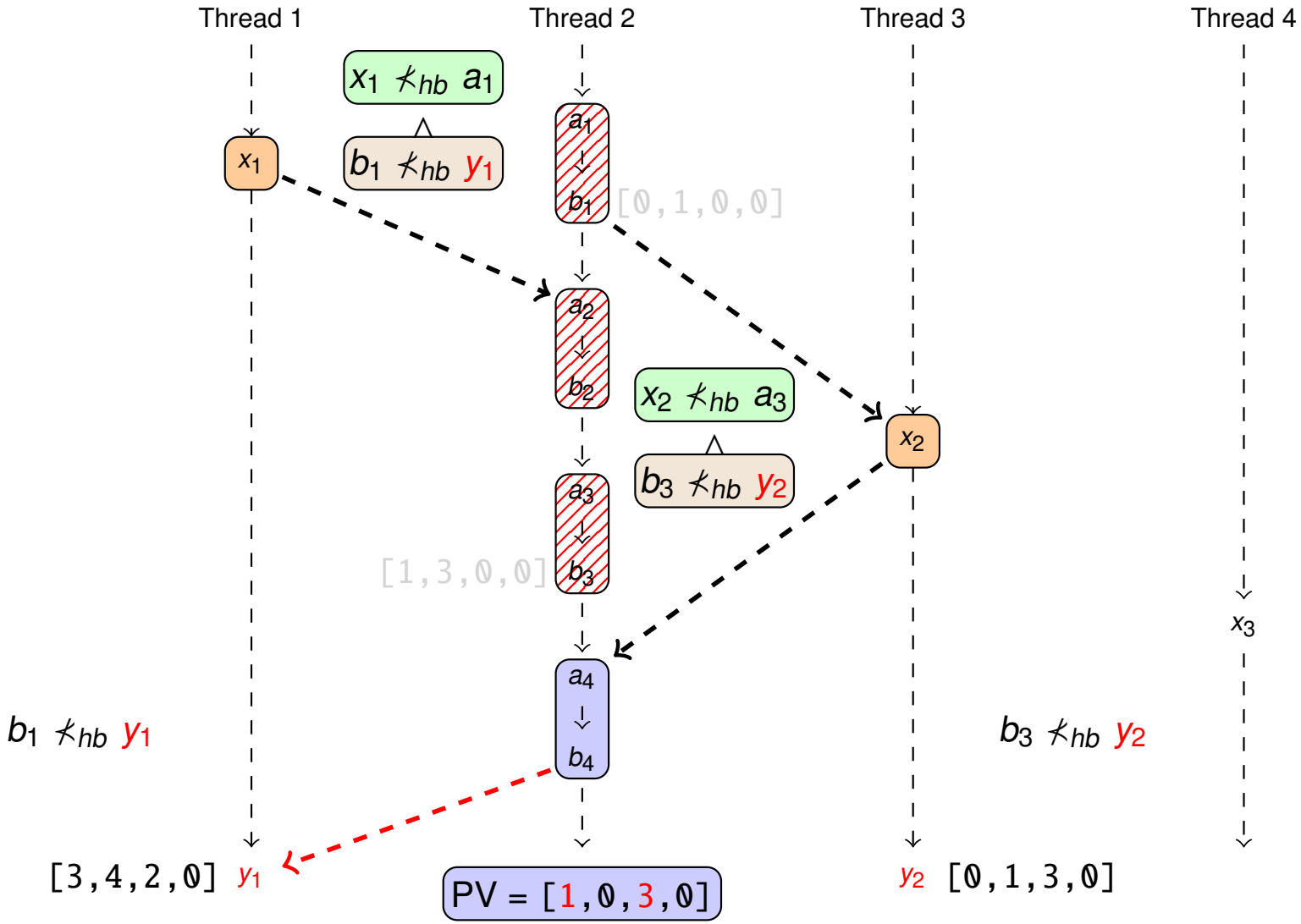
Discarding Targets



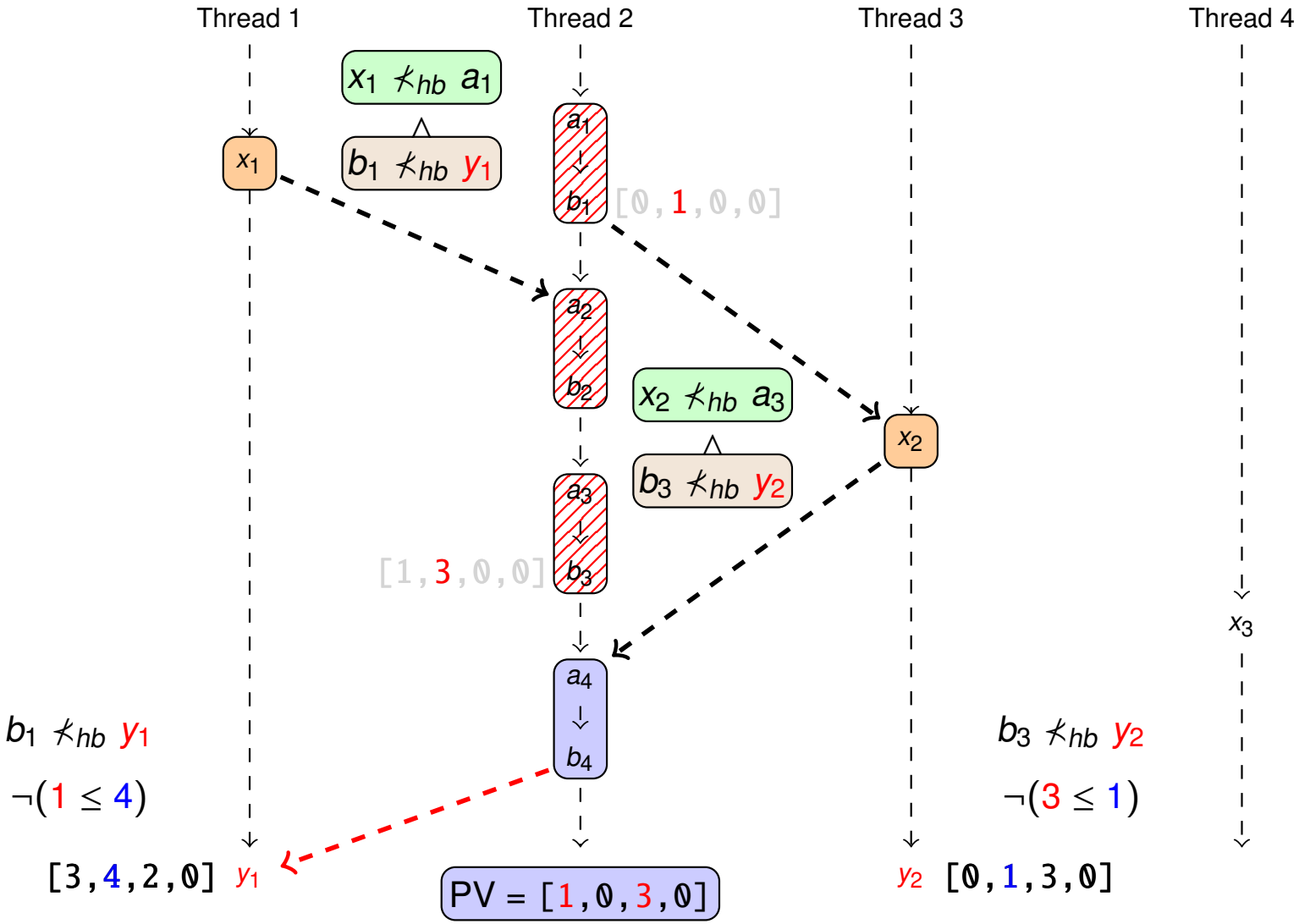
Discarding Targets



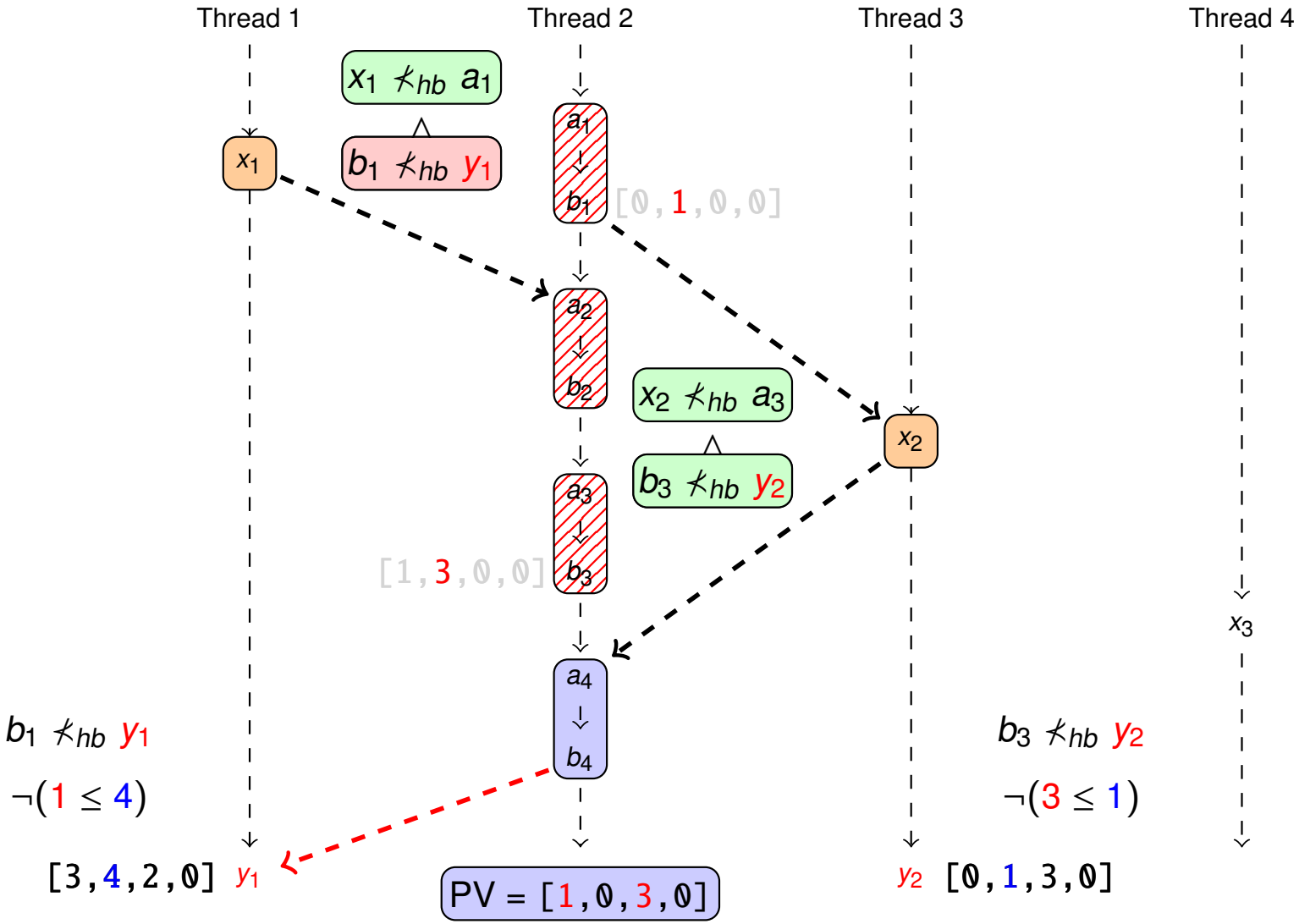
Discarding Targets



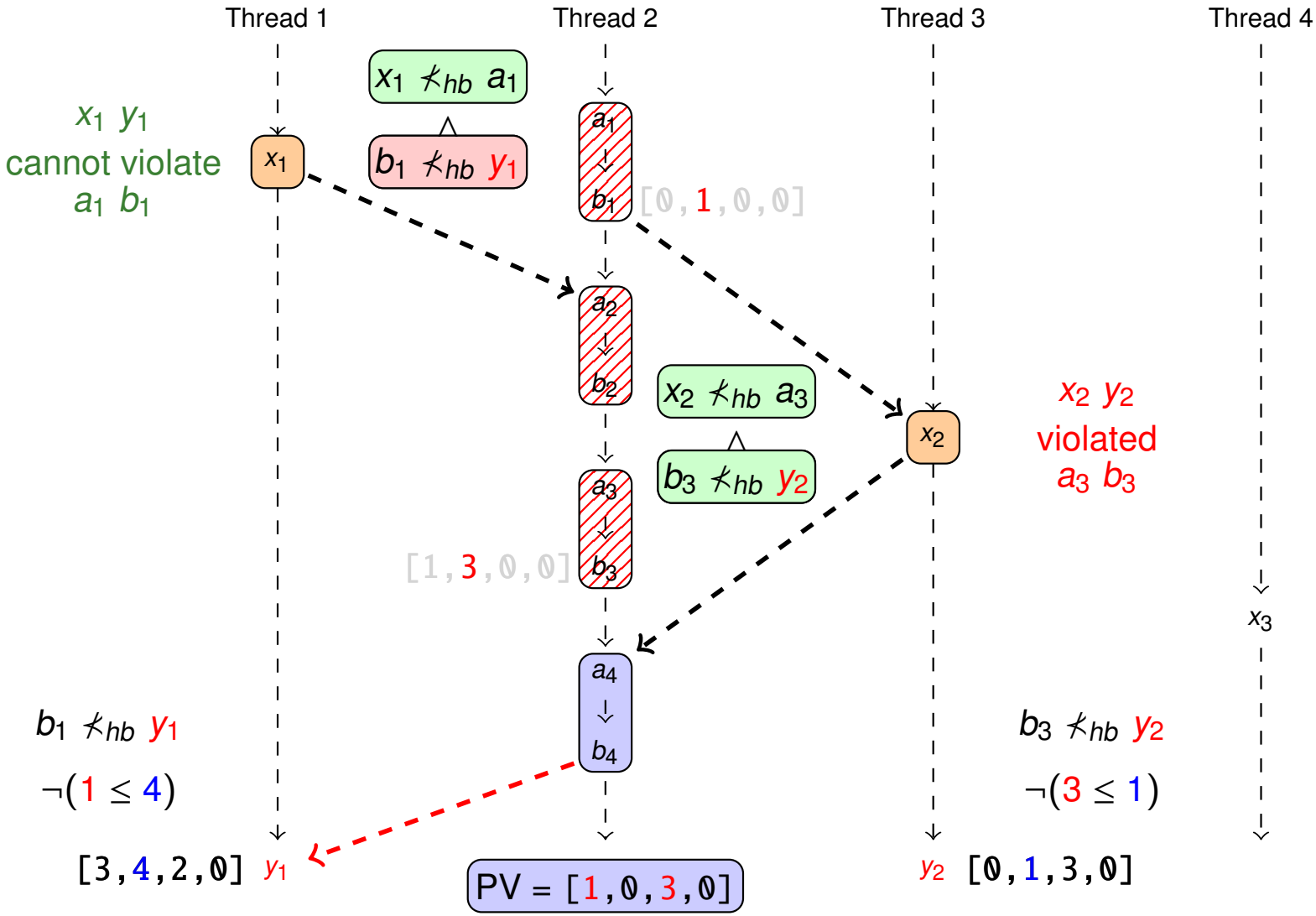
Discarding Targets



Discarding Targets



Discarding Targets



Experimental Results

Benchmark	T/S pairs	Contract Violations	False Positives	Potential AV	Real AV	SLOC	Time (s)
Coord03	8	380	0	0	380	116	1.01
Coord04	4	24	0	0	24	53	0.52
Local	4	2	0	0	2	27	0.52
NASA	1	100	0	0	100	96	0.60
Account	1	176	0	0	176	54	0.53
Link Manager	2	1	0	0	1	1.5K	1.14
Chromium-1	2	2	0	0	2	7.5M	49.12

Conclusion and Future Work

- We have extended contracts for concurrency with
  - Parameters (flow of data)
  - Spoilers (contextual information)
- We have proposed two methods to validate such contracts
  - Static method based on grammars and parsing trees
  - On-the-fly dynamic method based on happens-before relation and vector clocks
- We have evaluated both of these methods on both simple as well as real-world programs
- Future work
  - Support for more parameters in the dynamic approach
  - Support for spoilers in the static approach
  - Combine the static and dynamic approaches
  - Automatically derive contracts