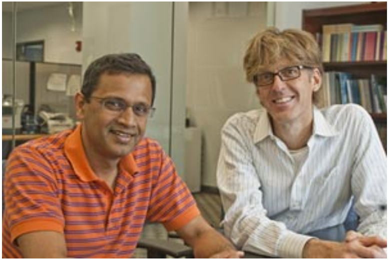
Between Linearizability and Quiescent Consistency



Radha Jagadeesan James Riely

DePaul University
Chicago, USA

ICALP 2014

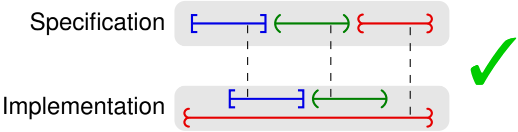
Linearizability (Herlihy/Wing 1990)

- “Each method call should appear to take effect instantaneously at some moment between its invocation and response.”
  (Herlihy/Shavit 2008)
- Ie, for every invocation, exists a *linearization point* such that
  - linearization point is between call and return
  - real-time order corresponds to some sequential execution

- Compositional (Herlihy/Wing 1990)
  Composition of the histories of two non-interfering linearizable objects is linearizable
- Intrinsically inefficient (Dwork/Herlihy/Waarts 1997)
  Trade-off between high contention and using many variables
  Data Structures in the Multicore Age (Shavit 2011, CACM)

Linearizability (Herlihy/Wing 1990)

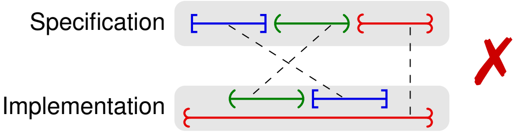
- "Each method call should appear to take effect instantaneously at some moment between its invocation and response." (Herlihy/Shavit 2008)
- Ie, for every invocation, exists a *linearization point* such that
  - linearization point is between call and return
  - real-time order corresponds to some sequential execution



- Compositional (Herlihy/Wing 1990)
  Composition of the histories of two non-interfering linearizable objects is linearizable
- Intrinsically inefficient (Dwork/Herlihy/Waarts 1997)
  Trade-off between high contention and using many variables
  Data Structures in the Multicore Age (Shavit 2011, CACM)

Linearizability (Herlihy/Wing 1990)

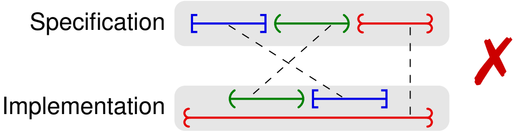
- "Each method call should appear to take effect instantaneously at some moment between its invocation and response." (Herlihy/Shavit 2008)
- Ie, for every invocation, exists a *linearization point* such that
  - linearization point is between call and return
  - real-time order corresponds to some sequential execution



- Compositional (Herlihy/Wing 1990)
  Composition of the histories of two non-interfering linearizable objects is linearizable
- Intrinsically inefficient (Dwork/Herlihy/Waarts 1997)
  Trade-off between high contention and using many variables
  Data Structures in the Multicore Age (Shavit 2011, CACM)

Linearizability (Herlihy/Wing 1990)

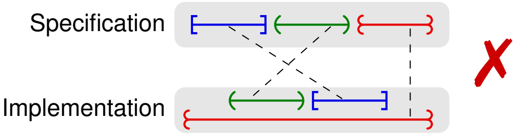
- "Each method call should appear to take effect instantaneously at some moment between its invocation and response." (Herlihy/Shavit 2008)
- Ie, for every invocation, exists a *linearization point* such that
  - linearization point is between call and return
  - real-time order corresponds to some sequential execution



- Compositional (Herlihy/Wing 1990)
  Composition of the histories of two non-interfering linearizable objects is linearizable
- Intrinsically inefficient (Dwork/Herlihy/Waarts 1997)
  Trade-off between high contention and using many variables
  Data Structures in the Multicore Age (Shavit 2011, CACM)

Linearizability (Herlihy/Wing 1990)

- "Each method call should appear to take effect instantaneously at some moment between its invocation and response." (Herlihy/Shavit 2008)
- Ie, for every invocation, exists a *linearization point* such that
  - linearization point is between call and return
  - real-time order corresponds to some sequential execution



- Compositional (Herlihy/Wing 1990)

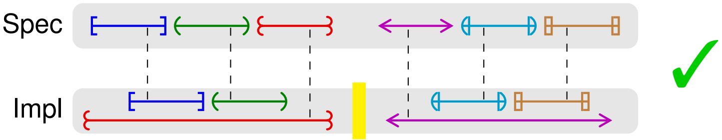
  Composition of the histories of two non-interfering linearizable objects is linearizable
- Intrinsically inefficient (Dwork/Herlihy/Waarts 1997)

  Trade-off between high contention and using many variables

  Data Structures in the Multicore Age (Shavit 2011, CACM)

Quiescent Consistency (Aspnes/Herlihy/Shavit 1991)

- Weaker than Linearizability (Lin $\Rightarrow$ QC)
- Compositional
- "Method calls separated *by a period of quiescence* should appear to take effect in their real-time order."
  (Herlihy/Shavit 2008)

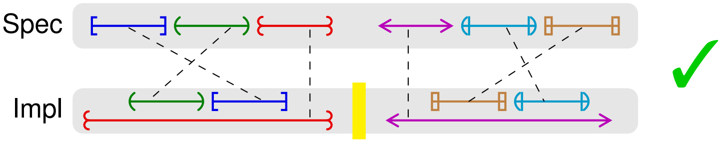


is a quiescent point

- Aspnes/Herlihy/Shavit (1991) actually prove other things
  - Step property (weaker than QC)
    Concretely: When quiescent, state is "very sensible"
    Abstractly: *If at any point accessed sequentially, behaves sequentially*
  - Gap property (morally "stronger" than QC)
    Concretely: Even when not quiescent, state is "pretty sensible"
    Abstractly: *??? This paper*

Quiescent Consistency (Aspnes/Herlihy/Shavit 1991)

- Weaker than Linearizability (Lin $\Rightarrow$ QC)
- Compositional
- "Method calls separated *by a period of quiescence* should appear to take effect in their real-time order."
  (Herlihy/Shavit 2008)

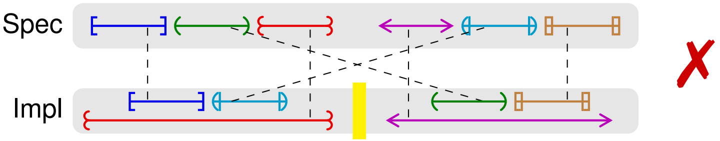


is a quiescent point

- Aspnes/Herlihy/Shavit (1991) actually prove other things
  - Step property (weaker than QC)
    Concretely: When quiescent, state is "very sensible"
    Abstractly: *If at any point accessed sequentially, behaves sequentially*
  - Gap property (morally "stronger" than QC)
    Concretely: Even when not quiescent, state is "pretty sensible"
    Abstractly: *??? This paper*

Quiescent Consistency (Aspnes/Herlihy/Shavit 1991)

- Weaker than Linearizability (Lin $\Rightarrow$ QC)
- Compositional
- "Method calls separated *by a period of quiescence* should appear to take effect in their real-time order."
  (Herlihy/Shavit 2008)

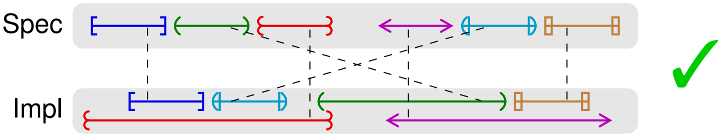


 is a quiescent point

- Aspnes/Herlihy/Shavit (1991) actually prove other things
  - Step property (weaker than QC)
    Concretely: When quiescent, state is "very sensible"
    Abstractly: *If at any point accessed sequentially, behaves sequentially*
  - Gap property (morally "stronger" than QC)
    Concretely: Even when not quiescent, state is "pretty sensible"
    Abstractly: *??? This paper*

Quiescent Consistency (Aspnes/Herlihy/Shavit 1991)

- Weaker than Linearizability (Lin $\Rightarrow$ QC)
- Compositional
- "Method calls separated *by a period of quiescence* should appear to take effect in their real-time order."
  (Herlihy/Shavit 2008)

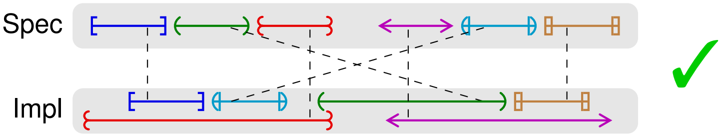


is a quiescent point

- Aspnes/Herlihy/Shavit (1991) actually prove other things
  - Step property (weaker than QC)
    Concretely: When quiescent, state is "very sensible"
    Abstractly: *If at any point accessed sequentially, behaves sequentially*
  - Gap property (morally "stronger" than QC)
    Concretely: Even when not quiescent, state is "pretty sensible"
    Abstractly: *??? This paper*

Quiescent Consistency (Aspnes/Herlihy/Shavit 1991)

- Weaker than Linearizability (Lin ⇒ QC)
- Compositional
- “Method calls separated *by a period of quiescence* should appear to take effect in their real-time order.”
  (Herlihy/Shavit 2008)

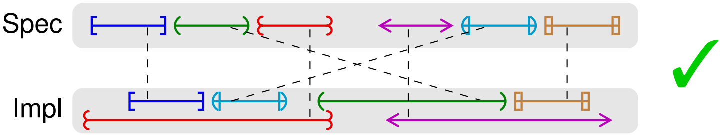


is a quiescent point

- Aspnes/Herlihy/Shavit (1991) actually prove other things
  - Step property (weaker than QC)
    Concretely: When quiescent, state is “very sensible”
    Abstractly: *If at any point accessed sequentially, behaves sequentially*
  - Gap property (morally “stronger” than QC)
    Concretely: Even when not quiescent, state is “pretty sensible”
    Abstractly: *??? This paper*

Quiescent Consistency (Aspnes/Herlihy/Shavit 1991)

- Weaker than Linearizability (Lin $\Rightarrow$ QC)
- Compositional
- "Method calls separated *by a period of quiescence* should appear to take effect in their real-time order."
  (Herlihy/Shavit 2008)

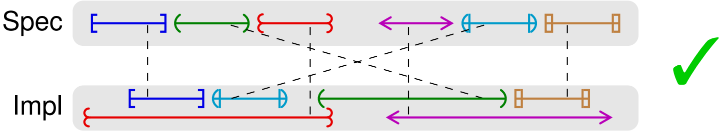


is a quiescent point

- Aspnes/Herlihy/Shavit (1991) actually prove other things
  - Step property (weaker than QC)
    Concretely: When quiescent, state is "very sensible"
    Abstractly: *If at any point accessed sequentially, behaves sequentially*
  - Gap property (morally "stronger" than QC)
    Concretely: Even when not quiescent, state is "pretty sensible"
    Abstractly: ??? *This paper*

Quiescent Consistency (Aspnes/Herlihy/Shavit 1991)

- Weaker than Linearizability (Lin ⇒ QC)
- Compositional
- “Method calls separated *by a period of quiescence* should appear to take effect in their real-time order.”
  (Herlihy/Shavit 2008)

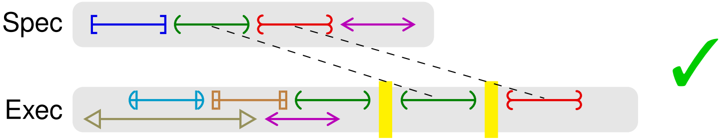


is a quiescent point

- Aspnes/Herlihy/Shavit (1991) actually prove other things
  - Step property (weaker than QC)
    Concretely: When quiescent, state is “very sensible”
    Abstractly: *If at any point accessed sequentially, behaves sequentially*
  - Gap property (morally “stronger” than QC)
    Concretely: Even when not quiescent, state is “pretty sensible”
    Abstractly: *???* *This paper*

But first... Weak Quiescent Consistency

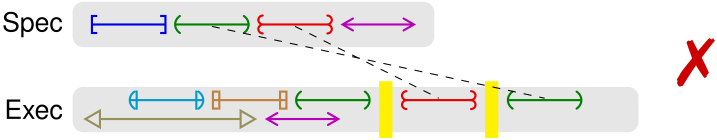
- Abstract view of "step property"
- If at any point accessed sequentially, behaves sequentially



- No comment about periods of concurrency
  QC requires *permutation*
  Weak QC does not (may be no spec trace with same set of events)

But first. . . Weak Quiescent Consistency

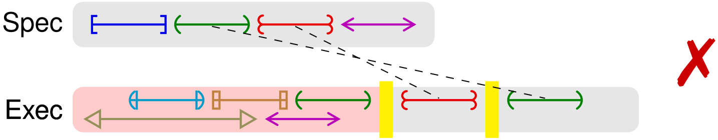
- Abstract view of “step property”
- If at any point accessed sequentially, behaves sequentially



- No comment about periods of concurrency
  QC requires *permutation*
  Weak QC does not (may be no spec trace with same set of events)

But first... Weak Quiescent Consistency

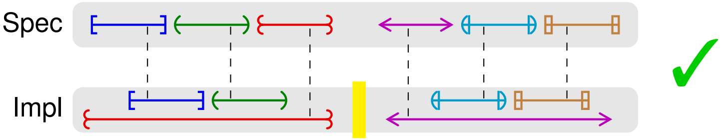
- Abstract view of "step property"
- If at any point accessed sequentially, behaves sequentially



- No comment about periods of concurrency
  QC requires *permutation*
  Weak QC does not (may be no spec trace with same set of events)

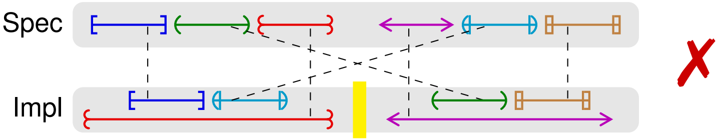
This paper... Quantitative Quiescent Consistency

- Between Linearizability and QC (Lin $\Rightarrow$ QQC $\Rightarrow$ QC)
- Compositional
- "Nonlinearizable behavior proportional to number of *early concurrent* calls"



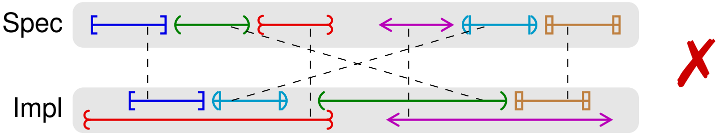
This paper... Quantitative Quiescent Consistency

- Between Linearizability and QC (Lin ⇒ QQC ⇒ QC)
- Compositional
- "Nonlinearizable behavior proportional to number of *early concurrent* calls"



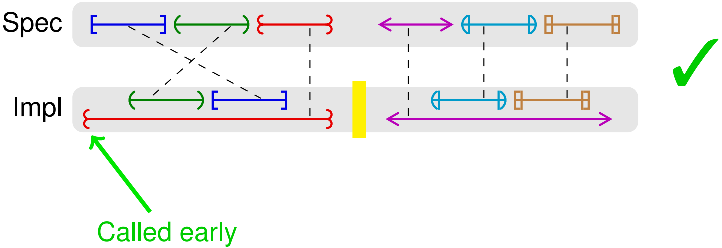
This paper... Quantitative Quiescent Consistency

- Between Linearizability and QC (Lin ⇒ QQC ⇒ QC)
- Compositional
- "Nonlinearizable behavior proportional to number of *early concurrent* calls"



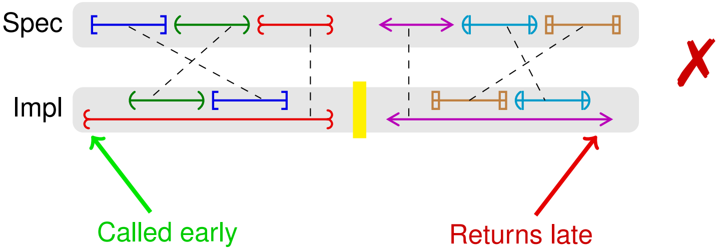
This paper... Quantitative Quiescent Consistency

- Between Linearizability and QC (Lin ⇒ QQC ⇒ QC)
- Compositional
- "Nonlinearizable behavior proportional to number of *early concurrent* calls"



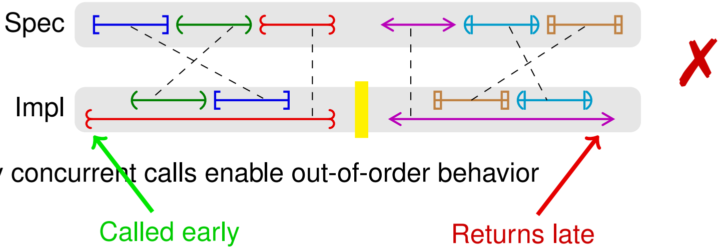
This paper... Quantitative Quiescent Consistency

- Between Linearizability and QC (Lin ⇒ QQC ⇒ QC)
- Compositional
- “Nonlinearizable behavior proportional to number of *early concurrent* calls”



This paper... Quantitative Quiescent Consistency

- Between Linearizability and QC (Lin $\Rightarrow$ QQC $\Rightarrow$ QC)
- Compositional
- "Nonlinearizable behavior proportional to number of *early concurrent* calls"

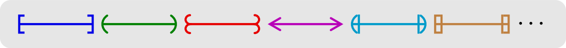


- Early concurrent calls enable out-of-order behavior

Definitions

- Number the call/return pairs of the specification

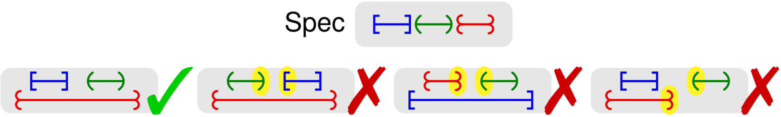
$$[_1\,]_1\, (_2\,)_2\, \{_3\, \}_3\, \langle_4\, \rangle_4\, (\!|_5\, |\!)_5\, [\![_6\,]\!]_6 \cdots$$



- Linearizability: If $)_i \xrightarrow{\textit{precedes}} [_j$ then $i < j$ (Herlihy/Wing 1990)
- QC: If $\}_i \xrightarrow{\textit{quiescent}} (_j$ then $i < j$ (Definition 3.1)
- Linearizability: If $i < j$ then $[_i \xrightarrow{\textit{precedes}})_j$ (Theorem 2.2)
- Linearizability: $\{i \mid [_i \xrightarrow{\textit{precedes}})_j\} \supseteq \{1, \ldots, j\}$ (Calculation)
- QQC: $\left|\{i \mid [_i \xrightarrow{\textit{precedes}} \}_j\}\right| \geq \left|\{1, \ldots, j\}\right| = j$ (Theorem 4.3)

Definitions

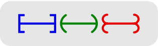
- Number the call/return pairs of the specification



- Linearizability: If $)_i \xrightarrow{precedes} [_j$ then $i < j$ (Herlihy/Wing 1990)
- QC: If $\}_i \xrightarrow{quiescent} (_j$ then $i < j$ (Definition 3.1)
- Linearizability: If $i < j$ then $[_i \xrightarrow{precedes})_j$ (Theorem 2.2)
- Linearizability: $\{i \mid [_i \xrightarrow{precedes})_j\} \supseteq \{1, \ldots, j\}$ (Calculation)
- QQC: $\left|\{i \mid [_i \xrightarrow{precedes} \}_j\}\right| \geq \left|\{1, \ldots, j\}\right| = j$ (Theorem 4.3)

Definitions

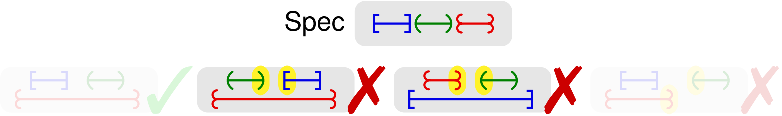
- Number the call/return pairs of the specification

Spec 

- Linearizability: If $)_i \xrightarrow{precedes} [_j$ then $i < j$ (Herlihy/Wing 1990)
- QC: If $\}_i \xrightarrow{quiescent} (_j$ then $i < j$ (Definition 3.1)
- Linearizability: If $i < j$ then $[_i \xrightarrow{precedes})_j$ (Theorem 2.2)
- Linearizability: $\{i \mid [_i \xrightarrow{precedes})_j\} \supseteq \{1, \ldots, j\}$ (Calculation)
- QQC: $|\{i \mid [_i \xrightarrow{precedes} \}_j\}| \geq |\{1, \ldots, j\}| = j$ (Theorem 4.3)

Definitions

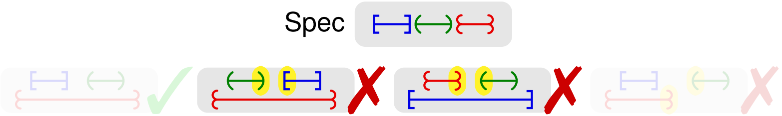
- Number the call/return pairs of the specification



- Linearizability: If $)_i \xrightarrow{precedes} [_j$ then $i < j$ (Return-to-call)
- QC: If $\}_i \xrightarrow{quiescent} (_j$ then $i < j$ (Definition 3.1)
- Linearizability: If $i < j$ then $[_i \xrightarrow{precedes})_j$ (Call-to-return)
- Linearizability: $\{i \mid [_i \xrightarrow{precedes})_j\} \supseteq \{1, \ldots, j\}$ (Calculation)
- QQC: $\left|\{i \mid [_i \xrightarrow{precedes} \}_j\}\right| \geq \left|\{1, \ldots, j\}\right| = j$ (Theorem 4.3)

Definitions

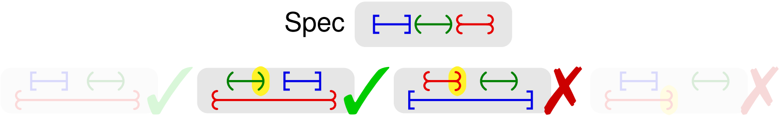
- Number the call/return pairs of the specification



- Linearizability: If $)_i \xrightarrow{precedes} [_j$ then $i < j$ (Herlihy/Wing 1990)
- QC: If $\}_i \xrightarrow{quiescent} (_j$ then $i < j$ (Definition 3.1)
- Linearizability: If $i < j$ then $[_i \xrightarrow{precedes})_j$ (Theorem 2.2)
- Linearizability: $\{i \mid [_i \xrightarrow{precedes})_j\} \supseteq \{1, \ldots, j\}$ (Calculation)
- QQC: $\left|\{i \mid [_i \xrightarrow{precedes} \}_j\}\right| \geq \left|\{1, \ldots, j\}\right| = j$ (Theorem 4.3)

Definitions

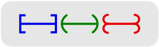
- Number the call/return pairs of the specification



- Linearizability: If $)_i \xrightarrow{precedes} [_j$ then $i < j$ (Herlihy/Wing 1990)
- QC: If $\}_i \xrightarrow{quiescent} (_j$ then $i < j$ (Definition 3.1)
- Linearizability: If $i < j$ then $[_i \xrightarrow{precedes})_j$ (Theorem 2.2)
- Linearizability: $\{i \mid [_i \xrightarrow{precedes})_j\} \supseteq \{1, \ldots, j\}$ (Calculation)
- QQC: $\left|\{i \mid [_i \xrightarrow{precedes} \}_j\}\right| \geq \left|\{1, \ldots, j\}\right| = j$ (Theorem 4.3)

Definitions

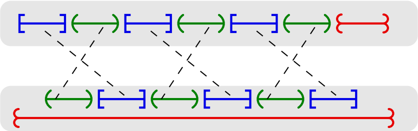
- Number the call/return pairs of the specification

Spec 

- Linearizability: If $)_i \xrightarrow{\textit{precedes}} [_j$ then $i < j$ (Return-to-call)
- QC: If $\}_i \xrightarrow{\textit{quiescent}} (_j$ then $i < j$ (Return-to-call)
- Linearizability: If $i < j$ then $[_i \xrightarrow{\textit{precedes}})_j$ (Call-to-return)
- Linearizability: $\{i \mid [_i \xrightarrow{\textit{precedes}})_j\} \supseteq \{1, \ldots, j\}$ (Call-to-return)
- QQC: $\left|\{i \mid [_i \xrightarrow{\textit{precedes}} \}_j\}\right| \geq \left|\{1, \ldots, j\}\right| = j$ (Call-to-return)

Interesting examples

- Enabling early call can be used repeatedly

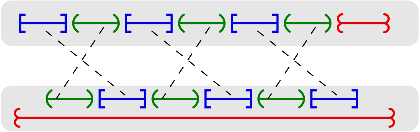


- Enablers can accumulate

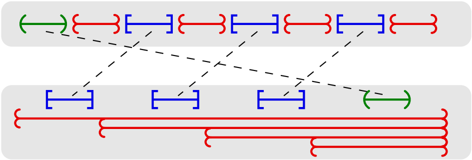
- Enablers can themselves be out-of-order

Interesting examples

- Enabling early call can be used repeatedly



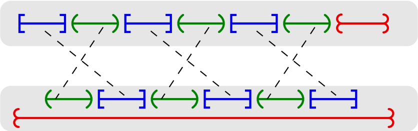
- Enablers can accumulate



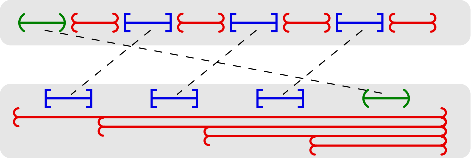
- Enablers can themselves be out-of-order

Interesting examples

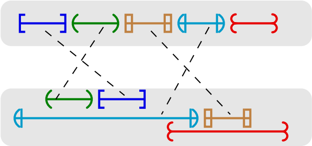
- Enabling early call can be used repeatedly



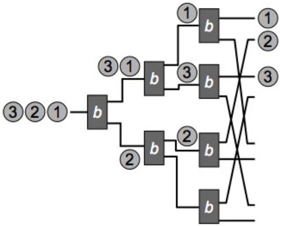
- Enablers can accumulate



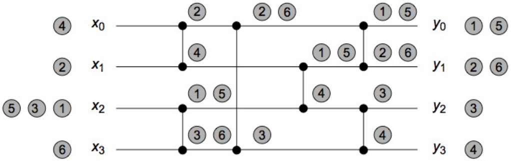
- Enablers can themselves be out-of-order



Quiescently Consistent Data Structures



- Counting networks
  - Bitonic Networks (Aspnes/Herlihy/Shavit 1991)
  - Diffracting Trees (Shavit/Zemach 1994)
  - Decrement/increment(Shavit/Touitou 1995) (Aiello/Busch/Herlihy/Mavronicolas/Shavit/Toutoui 1999)
- Stacks and Bags (aka, Pools)
  - Elimination Arrays/Trees (Shavit/Touitou 1995)
- “Almost” Linearizable
  - Experimental results
  - Theory involving max/min times (Lynch/Shavit/Shvartsman/Touitou 1996)
- *The Art of Multiprocessor Programming* (Herlihy/Shavit 2008)



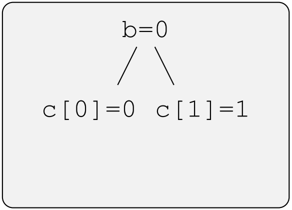
N-counter (simplified from Aspnes/Herlihy/Shavit 1991)

```
class Counter<N:Int> {
   field b:[0..N-1] = 0;                     // 1 balancer
   field c:Int[]    = [0, 1, ..., N-1];      // N counters
   method getAndIncrement():Int {
      val i:[0..N-1];
      atomic { i = b; b++; }
      atomic { val v = c[i]; c[i] += N; return v; } } }
```

$$\langle \mathrm{b}=0, \mathrm{c}=[0,1]\rangle \xrightarrow{[^{\mathtt{inc}}} \langle \mathrm{b}=1, \mathrm{c}=[0,1]\rangle \xrightarrow{]_0^{\mathtt{inc}}} \langle \mathrm{b}=1, \mathrm{c}=[2,1]\rangle$$

$$\xrightarrow{(^{\mathtt{inc}}} \langle \mathrm{b}=0, \mathrm{c}=[2,1]\rangle \xrightarrow{)_1^{\mathtt{inc}}} \langle \mathrm{b}=0, \mathrm{c}=[2,3]\rangle$$

$$\xrightarrow{\{^{\mathtt{inc}}} \langle \mathrm{b}=1, \mathrm{c}=[2,3]\rangle \xrightarrow{\}_2^{\mathtt{inc}}} \langle \mathrm{b}=1, \mathrm{c}=[4,3]\rangle$$



Behaves sequentially ☺

N-counter (simplified from Aspnes/Herlihy/Shavit 1991)

```
class Counter<N:Int> {
   field b:[0..N-1] = 0;                       // 1 balancer
   field c:Int[]    = [0, 1, ..., N-1];        // N counters
   method getAndIncrement():Int {
      val i:[0..N-1];
      atomic { i = b; b++; }
      atomic { val v = c[i]; c[i] += N; return v; } } }
```

$$\langle b = 0, c = [0, 1] \rangle \xrightarrow{[^{\text{inc}}} \langle b = 1, c = [0, 1] \rangle \xrightarrow{]_0^{\text{inc}}} \langle b = 1, c = [2, 1] \rangle$$

$$\xrightarrow{(^{\text{inc}}} \langle b = 0, c = [2, 1] \rangle \xrightarrow{)_1^{\text{inc}}} \langle b = 0, c = [2, 3] \rangle$$

$$\xrightarrow{\{^{\text{inc}}} \langle b = 1, c = [2, 3] \rangle \xrightarrow{\}_2^{\text{inc}}} \langle b = 1, c = [4, 3] \rangle$$

b=1

c[0]=0 c[1]=1

$[^{\text{inc}}$

Behaves sequentially ☺

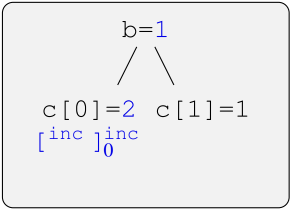
N-counter (simplified from Aspnes/Herlihy/Shavit 1991)

```
class Counter<N:Int> {
   field b:[0..N-1] = 0;                      // 1 balancer
   field c:Int[]    = [0, 1, ..., N-1];       // N counters
   method getAndIncrement():Int {
      val i:[0..N-1];
      atomic { i = b; b++; }
      atomic { val v = c[i]; c[i] += N; return v; } } }
```

$$\langle \mathrm{b}=0, \mathrm{c}=[0,1]\rangle \xrightarrow{[^{\mathtt{inc}}} \langle \mathrm{b}=1, \mathrm{c}=[0,1]\rangle \xrightarrow{]^{\mathtt{inc}}_0} \langle \mathrm{b}=1, \mathrm{c}=[2,1]\rangle$$

$$\xrightarrow{(^{\mathtt{inc}}} \langle \mathrm{b}=0, \mathrm{c}=[2,1]\rangle \xrightarrow{)^{\mathtt{inc}}_1} \langle \mathrm{b}=0, \mathrm{c}=[2,3]\rangle$$

$$\xrightarrow{\{^{\mathtt{inc}}} \langle \mathrm{b}=1, \mathrm{c}=[2,3]\rangle \xrightarrow{\}^{\mathtt{inc}}_2} \langle \mathrm{b}=1, \mathrm{c}=[4,3]\rangle$$



Behaves sequentially ☺

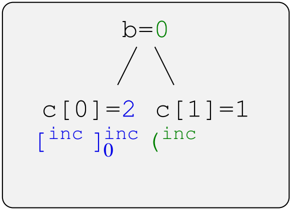
N-counter (simplified from Aspnes/Herlihy/Shavit 1991)

```
class Counter<N:Int> {
   field b:[0..N-1] = 0;                      // 1 balancer
   field c:Int[]    = [0, 1, ..., N-1];       // N counters
   method getAndIncrement():Int {
      val i:[0..N-1];
      atomic { i = b; b++; }
      atomic { val v = c[i]; c[i] += N; return v; } } }
```

$$\langle b = 0, c = [0, 1]\rangle \xrightarrow{[^{inc}} \langle b = 1, c = [0, 1]\rangle \xrightarrow{]^{inc}_0} \langle b = 1, c = [2, 1]\rangle$$

$$\xrightarrow{(^{inc}} \langle b = 0, c = [2, 1]\rangle \xrightarrow{)^{inc}_1} \langle b = 0, c = [2, 3]\rangle$$

$$\xrightarrow{\{^{inc}} \langle b = 1, c = [2, 3]\rangle \xrightarrow{\}^{inc}_2} \langle b = 1, c = [4, 3]\rangle$$



Behaves sequentially ☺

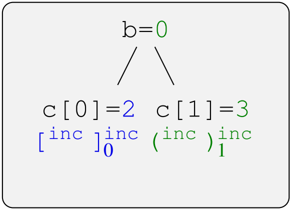
N-counter (simplified from Aspnes/Herlihy/Shavit 1991)

```
class Counter<N:Int> {
   field b:[0..N-1] = 0;                       // 1 balancer
   field c:Int[]    = [0, 1, ..., N-1];        // N counters
   method getAndIncrement():Int {
      val i:[0..N-1];
      atomic { i = b; b++; }
      atomic { val v = c[i]; c[i] += N; return v; } } }
```

$$\langle \mathrm{b} = 0, \mathrm{c} = [0, 1] \rangle \xrightarrow{[^{\mathtt{inc}}} \langle \mathrm{b} = 1, \mathrm{c} = [0, 1] \rangle \xrightarrow{]^{\mathtt{inc}}_0} \langle \mathrm{b} = 1, \mathrm{c} = [2, 1] \rangle$$

$$\xrightarrow{(^{\mathtt{inc}}} \langle \mathrm{b} = 0, \mathrm{c} = [2, 1] \rangle \xrightarrow{)^{\mathtt{inc}}_1} \langle \mathrm{b} = 0, \mathrm{c} = [2, 3] \rangle$$

$$\xrightarrow{\{^{\mathtt{inc}}} \langle \mathrm{b} = 1, \mathrm{c} = [2, 3] \rangle \xrightarrow{\}^{\mathtt{inc}}_2} \langle \mathrm{b} = 1, \mathrm{c} = [4, 3] \rangle$$



Behaves sequentially ☺

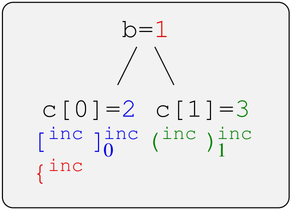
N-counter (simplified from Aspnes/Herlihy/Shavit 1991)

```
class Counter<N:Int> {
   field b:[0..N-1] = 0;                        // 1 balancer
   field c:Int[]    = [0, 1, ..., N-1];         // N counters
   method getAndIncrement():Int {
      val i:[0..N-1];
      atomic { i = b; b++; }
      atomic { val v = c[i]; c[i] += N; return v; } } }
```

$$\langle b = 0, c = [0, 1]\rangle \xrightarrow{[^{inc}} \langle b = 1, c = [0, 1]\rangle \xrightarrow{]^{inc}_{0}} \langle b = 1, c = [2, 1]\rangle$$

$$\xrightarrow{(^{inc}} \langle b = 0, c = [2, 1]\rangle \xrightarrow{)^{inc}_{1}} \langle b = 0, c = [2, 3]\rangle$$

$$\xrightarrow{\{^{inc}} \langle b = 1, c = [2, 3]\rangle \xrightarrow{\}^{inc}_{2}} \langle b = 1, c = [4, 3]\rangle$$



Behaves sequentially ☺

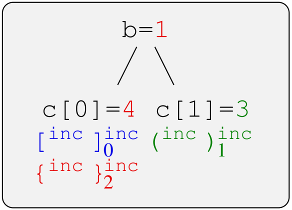
N-counter (simplified from Aspnes/Herlihy/Shavit 1991)

```
class Counter<N:Int> {
   field b:[0..N-1] = 0;                   // 1 balancer
   field c:Int[]    = [0, 1, ..., N-1];    // N counters
   method getAndIncrement():Int {
      val i:[0..N-1];
      atomic { i = b; b++; }
      atomic { val v = c[i]; c[i] += N; return v; } } }
```

$$\langle \mathrm{b}=0, \mathrm{c}=[0,1]\rangle \xrightarrow{[^{\mathtt{inc}}} \langle \mathrm{b}=1, \mathrm{c}=[0,1]\rangle \xrightarrow{]_0^{\mathtt{inc}}} \langle \mathrm{b}=1, \mathrm{c}=[2,1]\rangle$$

$$\xrightarrow{(^{\mathtt{inc}}} \langle \mathrm{b}=0, \mathrm{c}=[2,1]\rangle \xrightarrow{)_1^{\mathtt{inc}}} \langle \mathrm{b}=0, \mathrm{c}=[2,3]\rangle$$

$$\xrightarrow{\{^{\mathtt{inc}}} \langle \mathrm{b}=1, \mathrm{c}=[2,3]\rangle \xrightarrow{\}_2^{\mathtt{inc}}} \langle \mathrm{b}=1, \mathrm{c}=[4,3]\rangle$$



Behaves sequentially ☺

N-counter — Execution 2

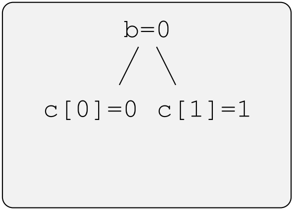
```
class Counter<N:Int> {
  field b:[0..N-1] = 0;              // 1 balancer
  field c:Int[]    = [0, 1, ..., N-1]; // N counters
  method getAndIncrement():Int {
    val i:[0..N-1];
    atomic { i = b; b++; }
    atomic { val v = c[i]; c[i] += N; return v; } } }
```

$$\langle \mathtt{b}=0, \mathtt{c}=[0,1]\rangle \xrightarrow{\{^{\mathtt{inc}}} \langle \mathtt{b}=1, \mathtt{c}=[0,1]\rangle$$

$$\xrightarrow{(^{\mathtt{inc}}} \langle \mathtt{b}=0, \mathtt{c}=[0,1]\rangle \xrightarrow{)_1^{\mathtt{inc}}} \langle \mathtt{b}=0, \mathtt{c}=[0,3]\rangle$$

$$\xrightarrow{[^{\mathtt{inc}}} \langle \mathtt{b}=1, \mathtt{c}=[0,3]\rangle \xrightarrow{]_0^{\mathtt{inc}}} \langle \mathtt{b}=1, \mathtt{c}=[2,3]\rangle$$

$$\xrightarrow{\}_2^{\mathtt{inc}}} \langle \mathtt{b}=1, \mathtt{c}=[4,3]\rangle$$



Not Linearizable ☹, but QQC ☺

N-counter — Execution 2

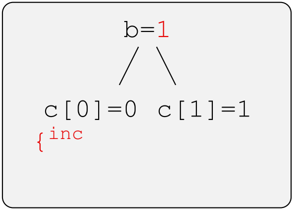
```
class Counter<N:Int> {
    field b:[0..N-1] = 0;                     // 1 balancer
    field c:Int[]    = [0, 1, ..., N-1];      // N counters
    method getAndIncrement():Int {
        val i:[0..N-1];
        atomic { i = b; b++; }
        atomic { val v = c[i]; c[i] += N; return v; } } }
```

$$\langle \mathrm{b} = 0, \mathrm{c} = [0, 1] \rangle \xrightarrow{\{^{\mathrm{inc}}} \langle \mathrm{b} = 1, \mathrm{c} = [0, 1] \rangle$$

$$\xrightarrow{(^{\mathrm{inc}}} \langle \mathrm{b} = 0, \mathrm{c} = [0, 1] \rangle \xrightarrow{)_1^{\mathrm{inc}}} \langle \mathrm{b} = 0, \mathrm{c} = [0, 3] \rangle$$

$$\xrightarrow{[^{\mathrm{inc}}} \langle \mathrm{b} = 1, \mathrm{c} = [0, 3] \rangle \xrightarrow{]_0^{\mathrm{inc}}} \langle \mathrm{b} = 1, \mathrm{c} = [2, 3] \rangle$$

$$\xrightarrow{\}_2^{\mathrm{inc}}} \langle \mathrm{b} = 1, \mathrm{c} = [4, 3] \rangle$$



Not Linearizable ☹, but QQC ☺

N-counter — Execution 2

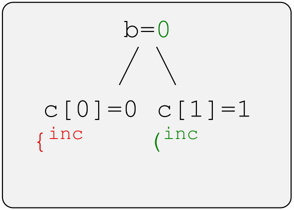
```
class Counter<N:Int> {
   field b:[0..N-1] = 0;                    // 1 balancer
   field c:Int[]    = [0, 1, ..., N-1];    // N counters
   method getAndIncrement():Int {
      val i:[0..N-1];
      atomic { i = b; b++; }
      atomic { val v = c[i]; c[i] += N; return v; } } }
```

$$\langle \mathrm{b}=0, \mathrm{c}=[0,1]\rangle \xrightarrow{\{\mathtt{inc}} \langle \mathrm{b}=1, \mathrm{c}=[0,1]\rangle$$

$$\xrightarrow{(\mathtt{inc}} \langle \mathrm{b}=0, \mathrm{c}=[0,1]\rangle \xrightarrow{)_1^{\mathtt{inc}}} \langle \mathrm{b}=0, \mathrm{c}=[0,3]\rangle$$

$$\xrightarrow{[\mathtt{inc}} \langle \mathrm{b}=1, \mathrm{c}=[0,3]\rangle \xrightarrow{]_0^{\mathtt{inc}}} \langle \mathrm{b}=1, \mathrm{c}=[2,3]\rangle$$

$$\xrightarrow{\}_2^{\mathtt{inc}}} \langle \mathrm{b}=1, \mathrm{c}=[4,3]\rangle$$



Not Linearizable ☹, but QQC ☺

N-counter — Execution 2

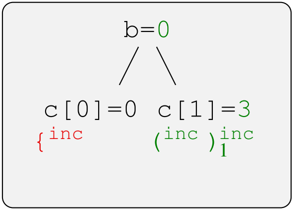
```
class Counter<N:Int> {
   field b:[0..N-1] = 0;                   // 1 balancer
   field c:Int[]    = [0, 1, ..., N-1];    // N counters
   method getAndIncrement():Int {
      val i:[0..N-1];
      atomic { i = b; b++; }
      atomic { val v = c[i]; c[i] += N; return v; } } }
```

$$\langle \mathrm{b} = 0, \mathrm{c} = [0, 1] \rangle \xrightarrow{\{^{\mathtt{inc}}} \langle \mathrm{b} = 1, \mathrm{c} = [0, 1] \rangle$$

$$\xrightarrow{(^{\mathtt{inc}}} \langle \mathrm{b} = 0, \mathrm{c} = [0, 1] \rangle \xrightarrow{)^{\mathtt{inc}}_{1}} \langle \mathrm{b} = 0, \mathrm{c} = [0, 3] \rangle$$

$$\xrightarrow{[^{\mathtt{inc}}} \langle \mathrm{b} = 1, \mathrm{c} = [0, 3] \rangle \xrightarrow{]^{\mathtt{inc}}_{0}} \langle \mathrm{b} = 1, \mathrm{c} = [2, 3] \rangle$$

$$\xrightarrow{\}^{\mathtt{inc}}_{2}} \langle \mathrm{b} = 1, \mathrm{c} = [4, 3] \rangle$$



Not Linearizable ☹, but QQC ☺

N-counter — Execution 2

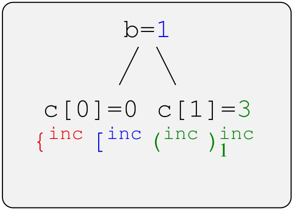
```
class Counter<N:Int> {
    field b:[0..N-1] = 0;                    // 1 balancer
    field c:Int[]    = [0, 1, ..., N-1];     // N counters
    method getAndIncrement():Int {
        val i:[0..N-1];
        atomic { i = b; b++; }
        atomic { val v = c[i]; c[i] += N; return v; } } }
```

$$\langle \mathrm{b}=0, \mathrm{c}=[0,1]\rangle \xrightarrow{\{^{\mathtt{inc}}} \langle \mathrm{b}=1, \mathrm{c}=[0,1]\rangle$$

$$\xrightarrow{(^{\mathtt{inc}}} \langle \mathrm{b}=0, \mathrm{c}=[0,1]\rangle \xrightarrow{)^{\mathtt{inc}}_1} \langle \mathrm{b}=0, \mathrm{c}=[0,3]\rangle$$

$$\xrightarrow{[^{\mathtt{inc}}} \langle \mathrm{b}=1, \mathrm{c}=[0,3]\rangle \xrightarrow{]^{\mathtt{inc}}_0} \langle \mathrm{b}=1, \mathrm{c}=[2,3]\rangle$$

$$\xrightarrow{\}^{\mathtt{inc}}_2} \langle \mathrm{b}=1, \mathrm{c}=[4,3]\rangle$$



Not Linearizable ☹, but QQC ☺

N-counter — Execution 2

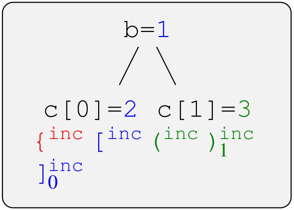
```
class Counter<N:Int> {
   field b:[0..N-1] = 0;                      // 1 balancer
   field c:Int[]    = [0, 1, ..., N-1];  // N counters
   method getAndIncrement():Int {
      val i:[0..N-1];
      atomic { i = b; b++; }
      atomic { val v = c[i]; c[i] += N; return v; } } }
```

$$\langle \mathrm{b} = 0, \mathrm{c} = [0, 1]\rangle \xrightarrow{\{^{\texttt{inc}}} \langle \mathrm{b} = 1, \mathrm{c} = [0, 1]\rangle$$

$$\xrightarrow{(^{\texttt{inc}}} \langle \mathrm{b} = 0, \mathrm{c} = [0, 1]\rangle \xrightarrow{)^{\texttt{inc}}_1} \langle \mathrm{b} = 0, \mathrm{c} = [0, 3]\rangle$$

$$\xrightarrow{[^{\texttt{inc}}} \langle \mathrm{b} = 1, \mathrm{c} = [0, 3]\rangle \xrightarrow{]^{\texttt{inc}}_0} \langle \mathrm{b} = 1, \mathrm{c} = [2, 3]\rangle$$

$$\xrightarrow{\}^{\texttt{inc}}_2} \langle \mathrm{b} = 1, \mathrm{c} = [4, 3]\rangle$$



Not Linearizable ☹, but QQC ☺

N-counter — Execution 2

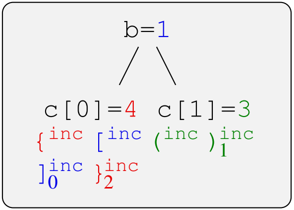
```
class Counter<N:Int> {
   field b:[0..N-1] = 0;                  // 1 balancer
   field c:Int[]    = [0, 1, ..., N-1];   // N counters
   method getAndIncrement():Int {
      val i:[0..N-1];
      atomic { i = b; b++; }
      atomic { val v = c[i]; c[i] += N; return v; } } }
```

$$\langle \mathrm{b}=0, \mathrm{c}=[0,1]\rangle \xrightarrow{\{^{\mathtt{inc}}} \langle \mathrm{b}=1, \mathrm{c}=[0,1]\rangle$$

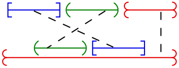
$$\xrightarrow{(^{\mathtt{inc}}} \langle \mathrm{b}=0, \mathrm{c}=[0,1]\rangle \xrightarrow{)^{\mathtt{inc}}_{1}} \langle \mathrm{b}=0, \mathrm{c}=[0,3]\rangle$$

$$\xrightarrow{[^{\mathtt{inc}}} \langle \mathrm{b}=1, \mathrm{c}=[0,3]\rangle \xrightarrow{]^{\mathtt{inc}}_{0}} \langle \mathrm{b}=1, \mathrm{c}=[2,3]\rangle$$

$$\xrightarrow{\}^{\mathtt{inc}}_{2}} \langle \mathrm{b}=1, \mathrm{c}=[4,3]\rangle$$



Not Linearizable ☹, but QQC ☺



Increment/Decrement counter

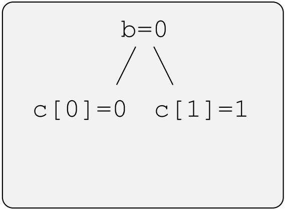
```
class Counter<N:Int> {
   field b:[0..N-1] = 0;                    // 1 balancer
   field c:Int[]    = [0, 1, ..., N-1];     // N counters
   method getAndIncrement():Int {
      val i:[0..N-1];
      atomic { i = b; b++; }
      atomic { val v = c[i]; c[i] += N; return v; } } }
   method decrementAndGet():Int {
      val i:[0..N-1];
      atomic { i = b-1; b--; }
      atomic { c[i] -= N; return c[i]; } }
```

$\langle b = 0, c = [0, 1] \rangle \xrightarrow{[^{inc}} \langle b = 1, c = [0, 1] \rangle \xrightarrow{(^{inc}} \langle b = 0, c = [0, 1] \rangle$

$\xrightarrow{\{^{dec}} \langle b = 1, c = [0, 1] \rangle \xrightarrow{\langle^{dec}} \langle b = 0, c = [0, 1] \rangle$

$\xrightarrow{\rangle^{dec}_{-2}} \langle b = 0, c = [-2, 1] \rangle \xrightarrow{]^{inc}_{-2}} \langle b = 0, c = [0, 1] \rangle$

$\xrightarrow{)^{inc}_{1}} \langle b = 0, c = [0, 3] \rangle \xrightarrow{\}^{dec}_{1}} \langle b = 0, c = [0, 1] \rangle$



Only weak QC ☹

$dec_{-2}\, inc_{-2}\, inc_{1}\, dec_{1}$ not a permutation of any spec trace!

Increment/Decrement counter

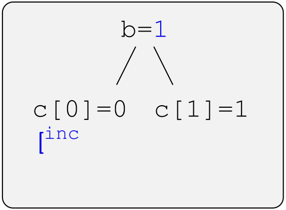
```
class Counter<N:Int> {
   field b:[0..N-1] = 0;                    // 1 balancer
   field c:Int[]    = [0, 1, ..., N-1];     // N counters
   method getAndIncrement():Int {
      val i:[0..N-1];
      atomic { i = b; b++; }
      atomic { val v = c[i]; c[i] += N; return v; } } }
   method decrementAndGet():Int {
      val i:[0..N-1];
      atomic { i = b-1; b--; }
      atomic { c[i] -= N; return c[i]; } }
```

$$\langle b=0, c=[0,1]\rangle \xrightarrow{[^{inc}} \langle b=1, c=[0,1]\rangle \xrightarrow{(^{inc}} \langle b=0, c=[0,1]\rangle$$

$$\xrightarrow{\{^{dec}} \langle b=1, c=[0,1]\rangle \xrightarrow{(^{dec}} \langle b=0, c=[0,1]\rangle$$

$$\xrightarrow{)^{dec}_{-2}} \langle b=0, c=[-2,1]\rangle \xrightarrow{]^{inc}_{-2}} \langle b=0, c=[0,1]\rangle$$

$$\xrightarrow{)^{inc}_{1}} \langle b=0, c=[0,3]\rangle \xrightarrow{\}^{dec}_{1}} \langle b=0, c=[0,1]\rangle$$



Only weak QC ☹

$dec_{-2}\, inc_{-2}\, inc_{1}\, dec_{1}$ not a permutation of any spec trace!

Increment/Decrement counter

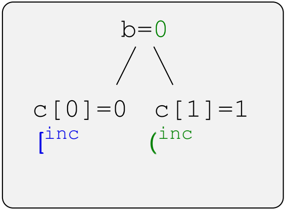
```
class Counter<N:Int> {
   field b:[0..N-1] = 0;                       // 1 balancer
   field c:Int[]    = [0, 1, ..., N-1];  // N counters
   method getAndIncrement():Int {
      val i:[0..N-1];
      atomic { i = b; b++; }
      atomic { val v = c[i]; c[i] += N; return v; } } }
   method decrementAndGet():Int {
      val i:[0..N-1];
      atomic { i = b-1; b--; }
      atomic { c[i] -= N; return c[i]; } }
```

$$\langle b=0, c=[0,1]\rangle \xrightarrow{[^{inc}} \langle b=1, c=[0,1]\ \rangle \xrightarrow{(^{inc}} \langle b=0, c=[0,1]\rangle$$

$$\xrightarrow{\{^{dec}} \langle b=1, c=[0,1]\ \rangle \xrightarrow{(^{dec}} \langle b=0, c=[0,1]\rangle$$

$$\xrightarrow{\}^{dec}_{-2}} \langle b=0, c=[-2,1]\rangle \xrightarrow{]^{inc}_{-2}} \langle b=0, c=[0,1]\rangle$$

$$\xrightarrow{)^{inc}_{1}} \langle b=0, c=[0,3]\ \rangle \xrightarrow{\}^{dec}_{1}} \langle b=0, c=[0,1]\rangle$$



Only weak QC ☹

$dec_{-2}\,inc_{-2}\,inc_{1}\,dec_{1}$ not a permutation of any spec trace!

Increment/Decrement counter

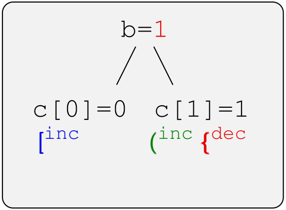
```
class Counter<N:Int> {
   field b:[0..N-1] = 0;                  // 1 balancer
   field c:Int[]    = [0, 1, ..., N-1];   // N counters
   method getAndIncrement():Int {
      val i:[0..N-1];
      atomic { i = b; b++; }
      atomic { val v = c[i]; c[i] += N; return v; } } }
   method decrementAndGet():Int {
      val i:[0..N-1];
      atomic { i = b-1; b--; }
      atomic { c[i] -= N; return c[i]; } }
```

$$\langle b = 0, c = [0, 1]\rangle \xrightarrow{[^{\mathtt{inc}}} \langle b = 1, c = [0, 1]\rangle \xrightarrow{(^{\mathtt{inc}}} \langle b = 0, c = [0, 1]\rangle$$

$$\xrightarrow{\{^{\mathtt{dec}}} \langle b = 1, c = [0, 1]\rangle \xrightarrow{(^{\mathtt{dec}}} \langle b = 0, c = [0, 1]\rangle$$

$$\xrightarrow{)^{\mathtt{dec}}_{-2}} \langle b = 0, c = [-2, 1]\rangle \xrightarrow{]^{\mathtt{inc}}_{-2}} \langle b = 0, c = [0, 1]\rangle$$

$$\xrightarrow{)^{\mathtt{inc}}_{1}} \langle b = 0, c = [0, 3]\rangle \xrightarrow{\}^{\mathtt{dec}}_{1}} \langle b = 0, c = [0, 1]\rangle$$



Only weak QC ☹

$\mathtt{dec}_{-2}\,\mathtt{inc}_{-2}\,\mathtt{inc}_{1}\,\mathtt{dec}_{1}$ not a permutation of any spec trace!

Increment/Decrement counter

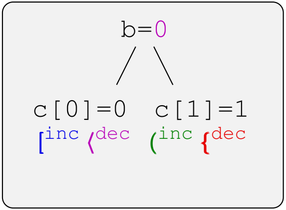
```
class Counter<N:Int> {
   field b:[0..N-1] = 0;                   // 1 balancer
   field c:Int[]    = [0, 1, ..., N-1];    // N counters
   method getAndIncrement():Int {
      val i:[0..N-1];
      atomic { i = b; b++; }
      atomic { val v = c[i]; c[i] += N; return v; } } }
   method decrementAndGet():Int {
      val i:[0..N-1];
      atomic { i = b-1; b--; }
      atomic { c[i] -= N; return c[i]; } }
```

$$\langle b=0, c=[0,1]\rangle \xrightarrow{[^{inc}} \langle b=1, c=[0,1]\rangle \xrightarrow{(^{inc}} \langle b=0, c=[0,1]\rangle$$

$$\xrightarrow{\{^{dec}} \langle b=1, c=[0,1]\rangle \xrightarrow{\langle^{dec}} \langle b=0, c=[0,1]\rangle$$

$$\xrightarrow{)^{dec}_{-2}} \langle b=0, c=[-2,1]\rangle \xrightarrow{]^{inc}_{-2}} \langle b=0, c=[0,1]\rangle$$

$$\xrightarrow{)^{inc}_{1}} \langle b=0, c=[0,3]\rangle \xrightarrow{\}^{dec}_{1}} \langle b=0, c=[0,1]\rangle$$



Only weak QC ☹

$dec_{-2}\, inc_{-2}\, inc_{1}\, dec_{1}$ not a permutation of any spec trace!

Increment/Decrement counter

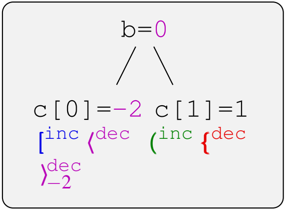
```
class Counter<N:Int> {
   field b:[0..N-1] = 0;                      // 1 balancer
   field c:Int[]    = [0, 1, ..., N-1];       // N counters
   method getAndIncrement():Int {
      val i:[0..N-1];
      atomic { i = b; b++; }
      atomic { val v = c[i]; c[i] += N; return v; } } }
   method decrementAndGet():Int {
      val i:[0..N-1];
      atomic { i = b-1; b--; }
      atomic { c[i] -= N; return c[i]; } }
```

$$\langle b = 0, c = [0, 1] \rangle \xrightarrow{[^{\mathtt{inc}}} \langle b = 1, c = [0, 1] \rangle \xrightarrow{(^{\mathtt{inc}}} \langle b = 0, c = [0, 1] \rangle$$

$$\xrightarrow{\{^{\mathtt{dec}}} \langle b = 1, c = [0, 1] \rangle \xrightarrow{\langle^{\mathtt{dec}}} \langle b = 0, c = [0, 1] \rangle$$

$$\xrightarrow{\rangle^{\mathtt{dec}}_{-2}} \langle b = 0, c = [-2, 1] \rangle \xrightarrow{]^{\mathtt{inc}}_{-2}} \langle b = 0, c = [0, 1] \rangle$$

$$\xrightarrow{)^{\mathtt{inc}}_{1}} \langle b = 0, c = [0, 3] \rangle \xrightarrow{\}^{\mathtt{dec}}_{1}} \langle b = 0, c = [0, 1] \rangle$$



Only weak QC ☹

$\mathtt{dec}_{-2}\,\mathtt{inc}_{-2}\,\mathtt{inc}_{1}\,\mathtt{dec}_{1}$ not a permutation of any spec trace!

Increment/Decrement counter

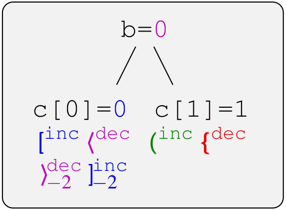
```
class Counter<N:Int> {
   field b:[0..N-1] = 0;                     // 1 balancer
   field c:Int[]    = [0, 1, ..., N-1];  // N counters
   method getAndIncrement():Int {
      val i:[0..N-1];
      atomic { i = b; b++; }
      atomic { val v = c[i]; c[i] += N; return v; } } }
   method decrementAndGet():Int {
      val i:[0..N-1];
      atomic { i = b-1; b--; }
      atomic { c[i] -= N; return c[i]; } }
```

$$\langle b = 0, c = [0, 1]\rangle \xrightarrow{[^{inc}} \langle b = 1, c = [0, 1] \rangle \xrightarrow{(^{inc}} \langle b = 0, c = [0, 1]\rangle$$

$$\xrightarrow{\{^{dec}} \langle b = 1, c = [0, 1] \rangle \xrightarrow{(^{dec}} \langle b = 0, c = [0, 1]\rangle$$

$$\xrightarrow{)^{dec}_{-2}} \langle b = 0, c = [-2, 1]\rangle \xrightarrow{]^{inc}_{-2}} \langle b = 0, c = [0, 1]\rangle$$

$$\xrightarrow{)^{inc}_{1}} \langle b = 0, c = [0, 3] \rangle \xrightarrow{\}^{dec}_{1}} \langle b = 0, c = [0, 1]\rangle$$



Only weak QC ☹

$dec_{-2}\,inc_{-2}\,inc_{1}\,dec_{1}$ not a permutation of any spec trace!

Increment/Decrement counter

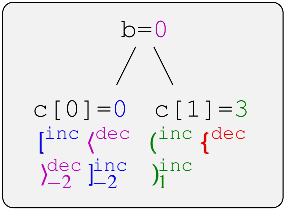
```
class Counter<N:Int> {
   field b:[0..N-1] = 0;                      // 1 balancer
   field c:Int[]    = [0, 1, ..., N-1];       // N counters
   method getAndIncrement():Int {
      val i:[0..N-1];
      atomic { i = b; b++; }
      atomic { val v = c[i]; c[i] += N; return v; } } }
   method decrementAndGet():Int {
      val i:[0..N-1];
      atomic { i = b-1; b--; }
      atomic { c[i] -= N; return c[i]; } }
```

$$\langle b=0, c=[0,1]\rangle \xrightarrow{[^{\mathtt{inc}}} \langle b=1, c=[0,1]\rangle \xrightarrow{(^{\mathtt{inc}}} \langle b=0, c=[0,1]\rangle$$

$$\xrightarrow{\{^{\mathtt{dec}}} \langle b=1, c=[0,1]\rangle \xrightarrow{\langle^{\mathtt{dec}}} \langle b=0, c=[0,1]\rangle$$

$$\xrightarrow{\rangle^{\mathtt{dec}}_{-2}} \langle b=0, c=[-2,1]\rangle \xrightarrow{]^{\mathtt{inc}}_{-2}} \langle b=0, c=[0,1]\rangle$$

$$\xrightarrow{)^{\mathtt{inc}}_{1}} \langle b=0, c=[0,3]\rangle \xrightarrow{\}^{\mathtt{dec}}_{1}} \langle b=0, c=[0,1]\rangle$$



Only weak QC ☹

$\mathtt{dec}_{-2}\,\mathtt{inc}_{-2}\,\mathtt{inc}_{1}\,\mathtt{dec}_{1}$ not a permutation of any spec trace!

Increment/Decrement counter

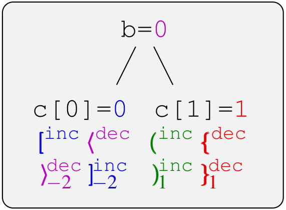
```
class Counter<N:Int> {
   field b:[0..N-1] = 0;                    // 1 balancer
   field c:Int[]     = [0, 1, ..., N-1];  // N counters
   method getAndIncrement():Int {
      val i:[0..N-1];
      atomic { i = b; b++; }
      atomic { val v = c[i]; c[i] += N; return v; } } }
   method decrementAndGet():Int {
      val i:[0..N-1];
      atomic { i = b-1; b--; }
      atomic { c[i] -= N; return c[i]; } }
```

$$\langle \mathtt{b}=0, \mathtt{c}=[0,1]\rangle \xrightarrow{[^{\mathtt{inc}}} \langle \mathtt{b}=1, \mathtt{c}=[0,1]\ \rangle \xrightarrow{(^{\mathtt{inc}}} \langle \mathtt{b}=0, \mathtt{c}=[0,1]\rangle$$

$$\xrightarrow{\{^{\mathtt{dec}}} \langle \mathtt{b}=1, \mathtt{c}=[0,1]\ \rangle \xrightarrow{(^{\mathtt{dec}}} \langle \mathtt{b}=0, \mathtt{c}=[0,1]\rangle$$

$$\xrightarrow{)^{\mathtt{dec}}_{-2}} \langle \mathtt{b}=0, \mathtt{c}=[-2,1]\rangle \xrightarrow{]^{\mathtt{inc}}_{-2}} \langle \mathtt{b}=0, \mathtt{c}=[0,1]\rangle$$

$$\xrightarrow{)^{\mathtt{inc}}_{1}} \langle \mathtt{b}=0, \mathtt{c}=[0,3]\ \rangle \xrightarrow{\}^{\mathtt{dec}}_{1}} \langle \mathtt{b}=0, \mathtt{c}=[0,1]\rangle$$



Only weak QC ☹

$\mathtt{dec}_{-2}\,\mathtt{inc}_{-2}\,\mathtt{inc}_{1}\,\mathtt{dec}_{1}$ not a permutation of any spec trace!

Stack

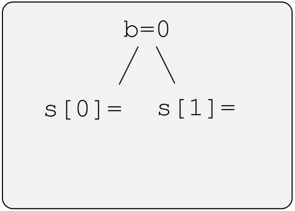
```
class Stack<N:Int> {
   field b:[0..N-1] = 0;                       // 1 balancer
   field s:Stack[]  = [[], [], ..., []]; // N stacks of values
   method push(x:Object):Unit {
      val i:[0..N-1];
      atomic { i = b; b++; }
      atomic { val v = s[i].push(x); return v; } }
   method pop():Object {
      val i:[0..N-1];
      atomic { i = b-1; b--; }
      atomic { val v = s[i].pop(); return v; } } }
```

$$\langle \mathrm{b}=0, \mathrm{s}=[[\,],[\,]]\,\rangle \xrightarrow{[^{\mathrm{psh}}_{a}} \langle \mathrm{b}=1, \mathrm{s}=[[\,],[\,]]\,\rangle \xrightarrow{(^{\mathrm{psh}}_{b}} \langle \mathrm{b}=0, \mathrm{s}=[[\,],[\,]]\,\rangle$$

$$\xrightarrow{\{^{\mathrm{pop}}} \langle \mathrm{b}=1, \mathrm{s}=[[\,],[\,]]\,\rangle \xrightarrow{<^{\mathrm{pop}}} \langle \mathrm{b}=0, \mathrm{s}=[[\,],[\,]]\,\rangle$$

$$\xrightarrow{>^{\mathrm{pop}}_{\mathrm{fail}}} \langle \mathrm{b}=0, \mathrm{s}=[[\,],[\,]]\,\rangle \xrightarrow{]^{\mathrm{psh}}} \langle \mathrm{b}=0, \mathrm{s}=[[a],[\,]]\,\rangle$$

$$\xrightarrow{)^{\mathrm{psh}}} \langle \mathrm{b}=0, \mathrm{s}=[[a],[b]]\rangle \xrightarrow{\}^{\mathrm{pop}}_{b}} \langle \mathrm{b}=0, \mathrm{s}=[[a],[\,]]\,\rangle$$



Linearizable ☺

Stack

```
class Stack<N:Int> {
   field b:[0..N-1] = 0;                    // 1 balancer
   field s:Stack[]  = [[], [], ..., []];  // N stacks of values
   method push(x:Object):Unit {
      val i:[0..N-1];
      atomic { i = b; b++; }
      atomic { val v = s[i].push(x); return v; } }
   method pop():Object {
      val i:[0..N-1];
      atomic { i = b-1; b--; }
      atomic { val v = s[i].pop(); return v; } } }
```

$$\langle \mathsf{b}=0, \mathsf{s}=[[\,],[\,]]\ \rangle \xrightarrow{[_a^{\mathsf{psh}}} \langle \mathsf{b}=1, \mathsf{s}=[[\,],[\,]]\ \rangle \xrightarrow{(_b^{\mathsf{psh}}} \langle \mathsf{b}=0, \mathsf{s}=[[\,],[\,]]\ \rangle$$

$$\xrightarrow{\{^{\mathsf{pop}}} \langle \mathsf{b}=1, \mathsf{s}=[[\,],[\,]]\ \rangle \xrightarrow{<^{\mathsf{pop}}} \langle \mathsf{b}=0, \mathsf{s}=[[\,],[\,]]\ \rangle$$

$$\xrightarrow{>^{\mathsf{pop}}_{\mathsf{fail}}} \langle \mathsf{b}=0, \mathsf{s}=[[\,],[\,]]\ \rangle \xrightarrow{]^{\mathsf{psh}}} \langle \mathsf{b}=0, \mathsf{s}=[[a],[\,]]\ \rangle$$

$$\xrightarrow{)^{\mathsf{psh}}} \langle \mathsf{b}=0, \mathsf{s}=[[a],[b]]\rangle \xrightarrow{\}_b^{\mathsf{pop}}} \langle \mathsf{b}=0, \mathsf{s}=[[a],[\,]]\ \rangle$$

b=1

s[0]= $[_a^{\mathsf{psh}}$ s[1]=

Linearizable ☺

Stack

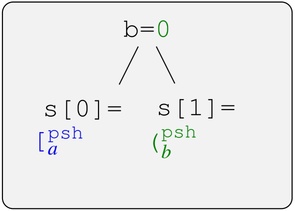
```
class Stack<N:Int> {
   field b:[0..N-1] = 0;                      // 1 balancer
   field s:Stack[]  = [[], [], ..., []]; // N stacks of values
   method push(x:Object):Unit {
      val i:[0..N-1];
      atomic { i = b; b++; }
      atomic { val v = s[i].push(x); return v; } }
   method pop():Object {
      val i:[0..N-1];
      atomic { i = b-1; b--; }
      atomic { val v = s[i].pop(); return v; } } }
```

$$\langle \mathrm{b}=0, \mathrm{s}=[[\,],[\,]]\ \rangle \xrightarrow{[_a^{\mathtt{psh}}} \langle \mathrm{b}=1, \mathrm{s}=[[\,],[\,]]\ \rangle \xrightarrow{(_b^{\mathtt{psh}}} \langle \mathrm{b}=0, \mathrm{s}=[[\,],[\,]]\ \rangle$$

$$\xrightarrow{\{^{\mathtt{pop}}} \langle \mathrm{b}=1, \mathrm{s}=[[\,],[\,]]\ \rangle \xrightarrow{<^{\mathtt{pop}}} \langle \mathrm{b}=0, \mathrm{s}=[[\,],[\,]]\ \rangle$$

$$\xrightarrow{>^{\mathtt{pop}}_{\mathrm{fail}}} \langle \mathrm{b}=0, \mathrm{s}=[[\,],[\,]]\ \rangle \xrightarrow{]^{\mathtt{psh}}} \langle \mathrm{b}=0, \mathrm{s}=[[a],[\,]]\ \rangle$$

$$\xrightarrow{)^{\mathtt{psh}}} \langle \mathrm{b}=0, \mathrm{s}=[[a],[b]]\rangle \xrightarrow{\}^{\mathtt{pop}}_b} \langle \mathrm{b}=0, \mathrm{s}=[[a],[\,]]\ \rangle$$



Linearizable ☺

Stack

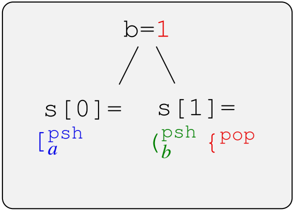
```
class Stack<N:Int> {
   field b:[0..N-1] = 0;                      // 1 balancer
   field s:Stack[]  = [[], [], ..., []];      // N stacks of values
   method push(x:Object):Unit {
      val i:[0..N-1];
      atomic { i = b; b++; }
      atomic { val v = s[i].push(x); return v; } }
   method pop():Object {
      val i:[0..N-1];
      atomic { i = b-1; b--; }
      atomic { val v = s[i].pop(); return v; } } }
```

$$\langle \mathrm{b}=0,\ \mathrm{s}=[[\,],[\,]]\ \rangle \xrightarrow{[^{\mathtt{psh}}_{a}} \langle \mathrm{b}=1,\ \mathrm{s}=[[\,],[\,]]\ \rangle \xrightarrow{(^{\mathtt{psh}}_{b}} \langle \mathrm{b}=0,\ \mathrm{s}=[[\,],[\,]]\ \rangle$$

$$\xrightarrow{\{^{\mathtt{pop}}} \langle \mathrm{b}=1,\ \mathrm{s}=[[\,],[\,]]\ \rangle \xrightarrow{<^{\mathtt{pop}}} \langle \mathrm{b}=0,\ \mathrm{s}=[[\,],[\,]]\ \rangle$$

$$\xrightarrow{>^{\mathtt{pop}}_{\text{fail}}} \langle \mathrm{b}=0,\ \mathrm{s}=[[\,],[\,]]\ \rangle \xrightarrow{]^{\mathtt{psh}}} \langle \mathrm{b}=0,\ \mathrm{s}=[[a],[\,]]\ \rangle$$

$$\xrightarrow{)^{\mathtt{psh}}} \langle \mathrm{b}=0,\ \mathrm{s}=[[a],[b]]\rangle \xrightarrow{\}^{\mathtt{pop}}_{b}} \langle \mathrm{b}=0,\ \mathrm{s}=[[a],[\,]]\ \rangle$$



Linearizable ☺

Stack

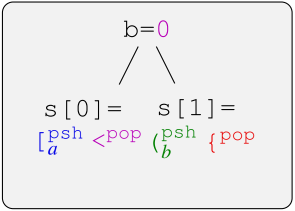
```
class Stack<N:Int> {
   field b:[0..N-1] = 0;                    // 1 balancer
   field s:Stack[]  = [[], [], ..., []]; // N stacks of values
   method push(x:Object):Unit {
      val i:[0..N-1];
      atomic { i = b; b++; }
      atomic { val v = s[i].push(x); return v; } }
   method pop():Object {
      val i:[0..N-1];
      atomic { i = b-1; b--; }
      atomic { val v = s[i].pop(); return v; } } }
```

$$\langle \mathrm{b}=0, \mathrm{s}=[[\,],[\,]]\ \rangle \xrightarrow{[_a^{\mathtt{psh}}} \langle \mathrm{b}=1, \mathrm{s}=[[\,],[\,]]\ \rangle \xrightarrow{(_b^{\mathtt{psh}}} \langle \mathrm{b}=0, \mathrm{s}=[[\,],[\,]]\ \rangle$$

$$\xrightarrow{\{^{\mathtt{pop}}} \langle \mathrm{b}=1, \mathrm{s}=[[\,],[\,]]\ \rangle \xrightarrow{<^{\mathtt{pop}}} \langle \mathrm{b}=0, \mathrm{s}=[[\,],[\,]]\ \rangle$$

$$\xrightarrow{>^{\mathtt{pop}}_{\mathsf{fail}}} \langle \mathrm{b}=0, \mathrm{s}=[[\,],[\,]]\ \rangle \xrightarrow{]^{\mathtt{psh}}} \langle \mathrm{b}=0, \mathrm{s}=[[a],[\,]]\ \rangle$$

$$\xrightarrow{)^{\mathtt{psh}}} \langle \mathrm{b}=0, \mathrm{s}=[[a],[b]]\rangle \xrightarrow{\}_b^{\mathtt{pop}}} \langle \mathrm{b}=0, \mathrm{s}=[[a],[\,]]\ \rangle$$



Linearizable ☺

Stack

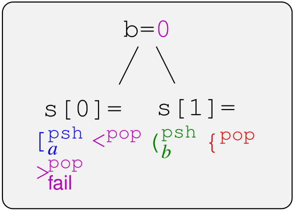
```
class Stack<N:Int> {
   field b:[0..N-1] = 0;                      // 1 balancer
   field s:Stack[]  = [[], [], ..., []]; // N stacks of values
   method push(x:Object):Unit {
      val i:[0..N-1];
      atomic { i = b; b++; }
      atomic { val v = s[i].push(x); return v; } }
   method pop():Object {
      val i:[0..N-1];
      atomic { i = b-1; b--; }
      atomic { val v = s[i].pop(); return v; } } }
```

$$\langle b=0, s=[[\,],[\,]]\rangle \xrightarrow{[^{psh}_{a}} \langle b=1, s=[[\,],[\,]]\rangle \xrightarrow{(^{psh}_{b}} \langle b=0, s=[[\,],[\,]]\rangle$$

$$\xrightarrow{\{^{pop}} \langle b=1, s=[[\,],[\,]]\rangle \xrightarrow{<^{pop}} \langle b=0, s=[[\,],[\,]]\rangle$$

$$\xrightarrow{>^{pop}_{fail}} \langle b=0, s=[[\,],[\,]]\rangle \xrightarrow{]^{psh}} \langle b=0, s=[[a],[\,]]\rangle$$

$$\xrightarrow{)^{psh}} \langle b=0, s=[[a],[b]]\rangle \xrightarrow{\}^{pop}_{b}} \langle b=0, s=[[a],[\,]]\rangle$$



Linearizable ☺

Stack

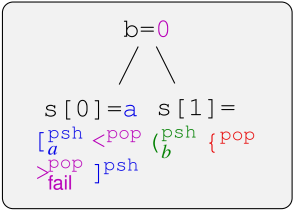
```
class Stack<N:Int> {
   field b:[0..N-1] = 0;                        // 1 balancer
   field s:Stack[]  = [[], [], ..., []];  // N stacks of values
   method push(x:Object):Unit {
      val i:[0..N-1];
      atomic { i = b; b++; }
      atomic { val v = s[i].push(x); return v; } }
   method pop():Object {
      val i:[0..N-1];
      atomic { i = b-1; b--; }
      atomic { val v = s[i].pop(); return v; } } }
```

$$\langle \mathrm{b}=0, \mathrm{s}=[[\,],[\,]]\ \rangle \xrightarrow{[^{\mathtt{psh}}_{a}} \langle \mathrm{b}=1, \mathrm{s}=[[\,],[\,]]\ \rangle \xrightarrow{(^{\mathtt{psh}}_{b}} \langle \mathrm{b}=0, \mathrm{s}=[[\,],[\,]]\ \rangle$$

$$\xrightarrow{\{^{\mathtt{pop}}} \langle \mathrm{b}=1, \mathrm{s}=[[\,],[\,]]\ \rangle \xrightarrow{<^{\mathtt{pop}}} \langle \mathrm{b}=0, \mathrm{s}=[[\,],[\,]]\ \rangle$$

$$\xrightarrow{>^{\mathtt{pop}}_{\mathsf{fail}}} \langle \mathrm{b}=0, \mathrm{s}=[[\,],[\,]]\ \rangle \xrightarrow{]^{\mathtt{psh}}} \langle \mathrm{b}=0, \mathrm{s}=[[a],[\,]]\ \rangle$$

$$\xrightarrow{)^{\mathtt{psh}}} \langle \mathrm{b}=0, \mathrm{s}=[[a],[b]]\rangle \xrightarrow{\}^{\mathtt{pop}}_{b}} \langle \mathrm{b}=0, \mathrm{s}=[[a],[\,]]\ \rangle$$



Linearizable ☺

Stack

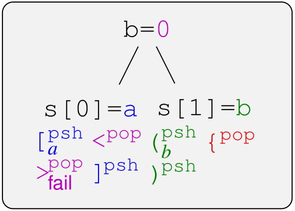
```
class Stack<N:Int> {
   field b:[0..N-1] = 0;                       // 1 balancer
   field s:Stack[]  = [[], [], ..., []]; // N stacks of values
   method push(x:Object):Unit {
      val i:[0..N-1];
      atomic { i = b; b++; }
      atomic { val v = s[i].push(x); return v; } }
   method pop():Object {
      val i:[0..N-1];
      atomic { i = b-1; b--; }
      atomic { val v = s[i].pop(); return v; } } }
```

$$\langle b=0, s=[[\,],[\,]]\rangle \xrightarrow{[^{psh}_{a}} \langle b=1, s=[[\,],[\,]]\rangle \xrightarrow{(^{psh}_{b}} \langle b=0, s=[[\,],[\,]]\rangle$$

$$\xrightarrow{\{^{pop}} \langle b=1, s=[[\,],[\,]]\rangle \xrightarrow{<^{pop}} \langle b=0, s=[[\,],[\,]]\rangle$$

$$\xrightarrow{>^{pop}_{fail}} \langle b=0, s=[[\,],[\,]]\rangle \xrightarrow{]^{psh}} \langle b=0, s=[[a],[\,]]\rangle$$

$$\xrightarrow{)^{psh}} \langle b=0, s=[[a],[b]]\rangle \xrightarrow{\}^{pop}_{b}} \langle b=0, s=[[a],[\,]]\rangle$$



Stack

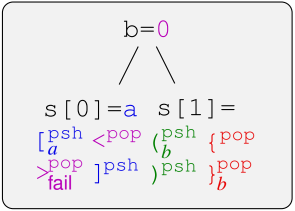
```
class Stack<N:Int> {
   field b:[0..N-1] = 0;                       // 1 balancer
   field s:Stack[]  = [[], [], ..., []]; // N stacks of values
   method push(x:Object):Unit {
      val i:[0..N-1];
      atomic { i = b; b++; }
      atomic { val v = s[i].push(x); return v; } }
   method pop():Object {
      val i:[0..N-1];
      atomic { i = b-1; b--; }
      atomic { val v = s[i].pop(); return v; } } }
```

$$\langle \mathrm{b}=0, \mathrm{s}=[[\,],[\,]]\ \rangle \xrightarrow{[^{\mathtt{psh}}_{a}} \langle \mathrm{b}=1, \mathrm{s}=[[\,],[\,]]\ \rangle \xrightarrow{(^{\mathtt{psh}}_{b}} \langle \mathrm{b}=0, \mathrm{s}=[[\,],[\,]]\ \rangle$$

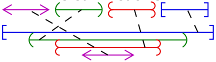
$$\xrightarrow{\{^{\mathtt{pop}}} \langle \mathrm{b}=1, \mathrm{s}=[[\,],[\,]]\ \rangle \xrightarrow{<^{\mathtt{pop}}} \langle \mathrm{b}=0, \mathrm{s}=[[\,],[\,]]\ \rangle$$

$$\xrightarrow{>^{\mathtt{pop}}_{\text{fail}}} \langle \mathrm{b}=0, \mathrm{s}=[[\,],[\,]]\ \rangle \xrightarrow{]^{\mathtt{psh}}} \langle \mathrm{b}=0, \mathrm{s}=[[a],[\,]]\ \rangle$$

$$\xrightarrow{)^{\mathtt{psh}}} \langle \mathrm{b}=0, \mathrm{s}=[[a],[b]]\rangle \xrightarrow{\}^{\mathtt{pop}}_{b}} \langle \mathrm{b}=0, \mathrm{s}=[[a],[\,]]\ \rangle$$



Linearizable ☺



Stack — Execution 2

```
class Stack<N:Int> {
   field b:[0..N-1] = 0;                    // 1 balancer
   field s:Stack[]  = [[], [], ..., []];    // N stacks of values
   method push(x:Object):Unit {
      val i:[0..N-1];
      atomic { i = b; b++; }
      atomic { val v = s[i].push(x); return v; } }
   method pop():Object {
      val i:[0..N-1];
      atomic { i = b-1; b--; }
      atomic { val v = s[i].pop(); return v; } } }
```

$$\langle \mathrm{b}=0, \mathrm{s}=[[\,],[\,]] \rangle \xrightarrow{[^{psh}_{a}} \langle \mathrm{b}=1, \mathrm{s}=[[\,],[\,]] \rangle \xrightarrow{]^{psh}} \langle \mathrm{b}=1, \mathrm{s}=[[a],[\,]] \rangle$$

$$\xrightarrow{\{^{psh}_{b}} \langle \mathrm{b}=0, \mathrm{s}=[[a],[\,]] \rangle \xrightarrow{\}^{psh}} \langle \mathrm{b}=0, \mathrm{s}=[[a],[b]] \rangle$$

$$\xrightarrow{(^{psh}_{c}} \langle \mathrm{b}=1, \mathrm{s}=[[a],[b]] \rangle \xrightarrow{<^{pop}} \langle \mathrm{b}=0, \mathrm{s}=[[a],[b]] \rangle$$

$$\xrightarrow{>^{pop}_{a}} \langle \mathrm{b}=0, \mathrm{s}=[[\_],[b]] \rangle \xrightarrow{)^{psh}} \langle \mathrm{b}=0, \mathrm{s}=[[c],[b]] \rangle$$

```
        b=0
        / \
s[0]=      s[1]=
```

Not even quiescent consistent ☹

$\longleftrightarrow$ should pop from $\longleftrightarrow$ or $\longleftrightarrow$, but not $\longleftrightarrow$

Stack — Execution 2

```
class Stack<N:Int> {
   field b:[0..N-1] = 0;                        // 1 balancer
   field s:Stack[]  = [[], [], ..., []];        // N stacks of values
   method push(x:Object):Unit {
      val i:[0..N-1];
      atomic { i = b; b++; }
      atomic { val v = s[i].push(x); return v; } }
   method pop():Object {
      val i:[0..N-1];
      atomic { i = b-1; b--; }
      atomic { val v = s[i].pop(); return v; } } }
```

$\langle b=0, s=[[\,],[\,]]\ \rangle \xrightarrow{[^{psh}_{a}} \langle b=1, s=[[\,],[\,]]\ \rangle \xrightarrow{]^{psh}} \langle b=1, s=[[a],[\,]]\ \rangle$

$\xrightarrow{\{^{psh}_{b}} \langle b=0, s=[[a],[\,]]\ \rangle \xrightarrow{\}^{psh}} \langle b=0, s=[[a],[b]] \rangle$

$\xrightarrow{(^{psh}_{c}} \langle b=1, s=[[a],[b]] \rangle \xrightarrow{<^{pop}} \langle b=0, s=[[a],[b]] \rangle$

$\xrightarrow{>^{pop}_{a}} \langle b=0, s=[[\_],[b]] \rangle \xrightarrow{)^{psh}} \langle b=0, s=[[c],[b]] \rangle$

Not even quiescent consistent ☹

↔ should pop from {—} or (—), but not [—]

```
      b=1
      / \
 s[0]=   s[1]=
[psh
 a
```

Stack — Execution 2

```
class Stack<N:Int> {
   field b:[0..N-1] = 0;                       // 1 balancer
   field s:Stack[]  = [[], [], ..., []]; // N stacks of values
   method push(x:Object):Unit {
      val i:[0..N-1];
      atomic { i = b; b++; }
      atomic { val v = s[i].push(x); return v; } }
   method pop():Object {
      val i:[0..N-1];
      atomic { i = b-1; b--; }
      atomic { val v = s[i].pop(); return v; } } }
```

$$\langle \mathrm{b}=0, \mathrm{s}=[[\,],[\,]]\ \rangle \xrightarrow{[^{\mathtt{psh}}_{a}} \langle \mathrm{b}=1, \mathrm{s}=[[\,],[\,]]\ \rangle \xrightarrow{]^{\mathtt{psh}}} \langle \mathrm{b}=1, \mathrm{s}=[[a],[\,]]\ \rangle$$

$$\xrightarrow{\{^{\mathtt{psh}}_{b}} \langle \mathrm{b}=0, \mathrm{s}=[[a],[\,]]\ \rangle \xrightarrow{\}^{\mathtt{psh}}} \langle \mathrm{b}=0, \mathrm{s}=[[a],[b]]\rangle$$

$$\xrightarrow{(^{\mathtt{psh}}_{c}} \langle \mathrm{b}=1, \mathrm{s}=[[a],[b]]\rangle \xrightarrow{<^{\mathtt{pop}}} \langle \mathrm{b}=0, \mathrm{s}=[[a],[b]]\rangle$$

$$\xrightarrow{>^{\mathtt{pop}}_{a}} \langle \mathrm{b}=0, \mathrm{s}=[[\_],[b]]\rangle \xrightarrow{)^{\mathtt{psh}}} \langle \mathrm{b}=0, \mathrm{s}=[[c],[b]]\rangle$$

Not even quiescent consistent ☹

↔ should pop from {—} or (—), but not [—]

```
        b=1
        / \
s[0]=a   s[1]=
```
$[^{\mathtt{psh}}_{a}\ ]^{\mathtt{psh}}$

Stack — Execution 2

```
class Stack<N:Int> {
   field b:[0..N-1] = 0;                  // 1 balancer
   field s:Stack[]  = [[], [], ..., []];  // N stacks of values
   method push(x:Object):Unit {
      val i:[0..N-1];
      atomic { i = b; b++; }
      atomic { val v = s[i].push(x); return v; } }
   method pop():Object {
      val i:[0..N-1];
      atomic { i = b-1; b--; }
      atomic { val v = s[i].pop(); return v; } } }
```

$$\langle b = 0, s = [[\,],[\,]]\rangle \xrightarrow{[^{psh}_{a}} \langle b = 1, s = [[\,],[\,]]\rangle \xrightarrow{]^{psh}} \langle b = 1, s = [[a],[\,]]\rangle$$

$$\xrightarrow{\{^{psh}_{b}} \langle b = 0, s = [[a],[\,]]\rangle \xrightarrow{\}^{psh}} \langle b = 0, s = [[a],[b]]\rangle$$

$$\xrightarrow{(^{psh}_{c}} \langle b = 1, s = [[a],[b]]\rangle \xrightarrow{<^{pop}} \langle b = 0, s = [[a],[b]]\rangle$$

$$\xrightarrow{>^{pop}_{a}} \langle b = 0, s = [[\_],[b]]\rangle \xrightarrow{)^{psh}} \langle b = 0, s = [[c],[b]]\rangle$$

b=0
/ \
s[0]=a s[1]=
$[^{psh}_{a}$ $]^{psh}$ $\{^{psh}_{b}$

Not even quiescent consistent ☹

⟷ should pop from ⟷ or ⟷, but not ⟷

Stack — Execution 2

```
class Stack<N:Int> {
   field b:[0..N-1] = 0;                       // 1 balancer
   field s:Stack[]  = [[], [], ..., []];  // N stacks of values
   method push(x:Object):Unit {
      val i:[0..N-1];
      atomic { i = b; b++; }
      atomic { val v = s[i].push(x); return v; } }
   method pop():Object {
      val i:[0..N-1];
      atomic { i = b-1; b--; }
      atomic { val v = s[i].pop(); return v; } } }
```

$$\langle b=0,\ s=[[\,],[\,]]\ \rangle \xrightarrow{[^{psh}_{a}} \langle b=1,\ s=[[\,],[\,]]\ \rangle \xrightarrow{]^{psh}} \langle b=1,\ s=[[a],[\,]]\ \rangle$$

$$\xrightarrow{\{^{psh}_{b}} \langle b=0,\ s=[[a],[\,]]\ \rangle \xrightarrow{\}^{psh}} \langle b=0,\ s=[[a],[b]]\rangle$$

$$\xrightarrow{(^{psh}_{c}} \langle b=1,\ s=[[a],[b]]\rangle \xrightarrow{<^{pop}} \langle b=0,\ s=[[a],[b]]\rangle$$

$$\xrightarrow{>^{pop}_{a}} \langle b=0,\ s=[[\_],[b]]\rangle \xrightarrow{)^{psh}} \langle b=0,\ s=[[c],[b]]\rangle$$

b=0

/ \

s[0]=a s[1]=b

$[^{psh}_{a}\ ]^{psh}$ $\{^{psh}_{b}\ \}^{psh}$

Not even quiescent consistent ☹

$\langle\!\!\rightarrow$ should pop from $\{\!\!\rightarrow\!\!\}$ or $(\!\!\rightarrow\!\!)$, but not $[\!\!\rightarrow\!\!]$

Stack — Execution 2

```
class Stack<N:Int> {
   field b:[0..N-1] = 0;                    // 1 balancer
   field s:Stack[]  = [[], [], ..., []];  // N stacks of values
   method push(x:Object):Unit {
      val i:[0..N-1];
      atomic { i = b; b++; }
      atomic { val v = s[i].push(x); return v; } }
   method pop():Object {
      val i:[0..N-1];
      atomic { i = b-1; b--; }
      atomic { val v = s[i].pop(); return v; } } }
```

$$\langle b = 0, s = [[\,], [\,]]\ \rangle \xrightarrow{[_a^{psh}} \langle b = 1, s = [[\,], [\,]]\ \rangle \xrightarrow{]^{psh}} \langle b = 1, s = [[a], [\,]]\ \rangle$$

$$\xrightarrow{\{_b^{psh}} \langle b = 0, s = [[a], [\,]]\ \rangle \xrightarrow{\}^{psh}} \langle b = 0, s = [[a], [b]] \rangle$$

$$\xrightarrow{(_c^{psh}} \langle b = 1, s = [[a], [b]] \rangle \xrightarrow{<^{pop}} \langle b = 0, s = [[a], [b]] \rangle$$

$$\xrightarrow{>_a^{pop}} \langle b = 0, s = [[\_], [b]] \rangle \xrightarrow{)^{psh}} \langle b = 0, s = [[c], [b]] \rangle$$

$b=1$

/ \

$s[0]=a$ $s[1]=b$

$[_a^{psh}$ $]^{psh}$ $\{_b^{psh}$ $\}^{psh}$

$)^{psh}$

Not even quiescent consistent ☹

$\longleftrightarrow$ should pop from $\{\!\!\longrightarrow\!\!\}$ or $(\!\longleftrightarrow\!)$, but not $[\!\longrightarrow\!]$

Stack — Execution 2

```
class Stack<N:Int> {
   field b:[0..N-1] = 0;                        // 1 balancer
   field s:Stack[]  = [[], [], ..., []];  // N stacks of values
   method push(x:Object):Unit {
      val i:[0..N-1];
      atomic { i = b; b++; }
      atomic { val v = s[i].push(x); return v; } }
   method pop():Object {
      val i:[0..N-1];
      atomic { i = b-1; b--; }
      atomic { val v = s[i].pop(); return v; } } }
```

$$\langle b=0, s=[[\,],[\,]]\rangle \xrightarrow{[^{psh}_{a}} \langle b=1, s=[[\,],[\,]]\rangle \xrightarrow{]^{psh}} \langle b=1, s=[[a],[\,]]\rangle$$

$$\xrightarrow{\{^{psh}_{b}} \langle b=0, s=[[a],[\,]]\rangle \xrightarrow{\}^{psh}} \langle b=0, s=[[a],[b]]\rangle$$

$$\xrightarrow{(^{psh}_{c}} \langle b=1, s=[[a],[b]]\rangle \xrightarrow{<^{pop}} \langle b=0, s=[[a],[b]]\rangle$$

$$\xrightarrow{>^{pop}_{a}} \langle b=0, s=[[\_],[b]]\rangle \xrightarrow{)^{psh}} \langle b=0, s=[[c],[b]]\rangle$$

Not even quiescent consistent ☹

↔ should pop from {—} or (—), but not [—]

b=0

/ \

s[0]= a s[1]= b

$[^{psh}_{a}$ $]^{psh}$ $\{^{psh}_{b}$ $\}^{psh}$

$)^{psh}$ $<^{pop}$

Stack — Execution 2

```
class Stack<N:Int> {
   field b:[0..N-1] = 0;                    // 1 balancer
   field s:Stack[]  = [[], [], ..., []];    // N stacks of values
   method push(x:Object):Unit {
      val i:[0..N-1];
      atomic { i = b; b++; }
      atomic { val v = s[i].push(x); return v; } }
   method pop():Object {
      val i:[0..N-1];
      atomic { i = b-1; b--; }
      atomic { val v = s[i].pop(); return v; } } }
```

$$\langle \mathrm{b}=0,\ \mathrm{s}=[[\,],[\,]]\ \rangle \xrightarrow{[^{\mathrm{psh}}_{a}} \langle \mathrm{b}=1,\ \mathrm{s}=[[\,],[\,]]\ \rangle \xrightarrow{]^{\mathrm{psh}}} \langle \mathrm{b}=1,\ \mathrm{s}=[[a],[\,]]\ \rangle$$

$$\xrightarrow{\{^{\mathrm{psh}}_{b}} \langle \mathrm{b}=0,\ \mathrm{s}=[[a],[\,]]\ \rangle \xrightarrow{\}^{\mathrm{psh}}} \langle \mathrm{b}=0,\ \mathrm{s}=[[a],[b]]\rangle$$

$$\xrightarrow{(^{\mathrm{psh}}_{c}} \langle \mathrm{b}=1,\ \mathrm{s}=[[a],[b]]\rangle \xrightarrow{<^{\mathrm{pop}}} \langle \mathrm{b}=0,\ \mathrm{s}=[[a],[b]]\rangle$$

$$\xrightarrow{>^{\mathrm{pop}}_{a}} \langle \mathrm{b}=0,\ \mathrm{s}=[[\_],[b]]\rangle \xrightarrow{)^{\mathrm{psh}}} \langle \mathrm{b}=0,\ \mathrm{s}=[[c],[b]]\rangle$$

Not even quiescent consistent ☹

$\longleftrightarrow$ should pop from $\{\!\!\longleftrightarrow\!\!\}$ or $(\longleftrightarrow)$, but not $[\longleftrightarrow]$

b=0

/ \

s[0]= s[1]= b

$[^{\mathrm{psh}}_{a}\ ]^{\mathrm{psh}}$ $\{^{\mathrm{psh}}_{b}\ \}^{\mathrm{psh}}$

$)^{\mathrm{psh}}\ <^{\mathrm{pop}}$

$>^{\mathrm{pop}}_{a}$

Stack — Execution 2

```
class Stack<N:Int> {
   field b:[0..N-1] = 0;                      // 1 balancer
   field s:Stack[]  = [[], [], ..., []];  // N stacks of values
   method push(x:Object):Unit {
      val i:[0..N-1];
      atomic { i = b; b++; }
      atomic { val v = s[i].push(x); return v; } }
   method pop():Object {
      val i:[0..N-1];
      atomic { i = b-1; b--; }
      atomic { val v = s[i].pop(); return v; } } }
```

$$\langle \mathrm{b}=0, \mathrm{s}=[[\,],[\,]]\ \rangle \xrightarrow{[_a^{\mathtt{psh}}} \langle \mathrm{b}=1, \mathrm{s}=[[\,],[\,]]\ \rangle \xrightarrow{]^{\mathtt{psh}}} \langle \mathrm{b}=1, \mathrm{s}=[[a],[\,]]\ \rangle$$

$$\xrightarrow{\{_b^{\mathtt{psh}}} \langle \mathrm{b}=0, \mathrm{s}=[[a],[\,]]\ \rangle \xrightarrow{\}^{\mathtt{psh}}} \langle \mathrm{b}=0, \mathrm{s}=[[a],[b]]\rangle$$

$$\xrightarrow{(_c^{\mathtt{psh}}} \langle \mathrm{b}=1, \mathrm{s}=[[a],[b]]\rangle \xrightarrow{<^{\mathtt{pop}}} \langle \mathrm{b}=0, \mathrm{s}=[[a],[b]]\rangle$$

$$\xrightarrow{>_a^{\mathtt{pop}}} \langle \mathrm{b}=0, \mathrm{s}=[[\_],[b]]\rangle \xrightarrow{)^{\mathtt{psh}}} \langle \mathrm{b}=0, \mathrm{s}=[[c],[b]]\rangle$$

b=0

/ \

s[0]= c s[1]= b

$[_a^{\mathtt{psh}}$ $]^{\mathtt{psh}}$ $\{_b^{\mathtt{psh}}$ $\}^{\mathtt{psh}}$

$)^{\mathtt{psh}}$ $<^{\mathtt{pop}}$

$>_a^{\mathtt{pop}}$ $)^{\mathtt{psh}}$

Not even quiescent consistent ☹

$\leftrightarrow$ should pop from $\{\!-\!\}$ or $(\!-\!)$, but not $[\!-\!]$

Results

- Three characterizations of QQC
  - Call-to-return (given earlier)
  - Return-to-call (à la Herlihy/Wing)
  - Proxy for sequential implementation (flat combiner + speculation)
    - Single thread accesses sequential structure
    - Upon receiving actual call, speculatively execute any method with any args
    - Only return when speculative call matches actual call
- Proof of compositionality
  - Global constraints that are solvable because of “flow” properties
- Proofs and counterexamples for tree-based structures
  - Increment/decrement N-counter (weak QC)
  - Increment-only N-counter (QQC)
  - General N-stack (QC)
  - “Properly popped” N-stack (QQC)
    - Pop must wait for concurrent push on same underlying stack
    - Not sufficient for pop to wait on empty stack
    - Proof uses instrumented N-stack that emits a QQC specification trace
  - Tree of N-stacks (same as single N-stack)

Results

- Three characterizations of QQC
  - Call-to-return (given earlier)
  - Return-to-call (à la Herlihy/Wing)
  - Proxy for sequential implementation (flat combiner + speculation)
    - Single thread accesses sequential structure
    - Upon receiving actual call, speculatively execute any method with any args
    - Only return when speculative call matches actual call
- Proof of compositionality
  - Global constraints that are solvable because of “flow” properties
- Proofs and counterexamples for tree-based structures
  - Increment/decrement N-counter (weak QC)
  - Increment-only N-counter (QQC)
  - General N-stack (QC)
  - “Properly popped” N-stack (QQC)
    - Pop must wait for concurrent push on same underlying stack
    - Not sufficient for pop to wait on empty stack
    - Proof uses instrumented N-stack that emits a QQC specification trace
  - Tree of N-stacks (same as single N-stack)

Return-to-call characterization

Linearizability:
$\forall prefix/suffix = exec$

$\forall ret \in prefix$
$\forall call \in suffix$

$$ret \xrightarrow{exec} call \quad \text{implies} \quad ret \xrightarrow{spec} call$$

Return-to-call characterization

Quiescent consistency:

$\forall prefix/suffix = exec$

if $prefix$ has 0 open calls, then

$\forall ret \in prefix$

$\forall call \in suffix$

$$ret \xrightarrow{exec} call \quad \text{implies} \quad ret \xrightarrow{spec} call$$

Return-to-call characterization

QQC:
$\forall prefix/suffix = exec$
if $prefix$ has k open/early calls, then there exists $|ignoredCalls| \leq k$
$\forall ret \in prefix$
$\forall call \in suffix - ignoredCalls$

$$ret \xrightarrow{exec} call \quad \text{implies} \quad ret \xrightarrow{spec} call$$

Results

- Three characterizations of QQC
  - Call-to-return (given earlier)
  - Return-to-call (à la Herlihy/Wing)
  - Proxy for sequential implementation (flat combiner + speculation)
    - Single thread accesses sequential structure
    - Upon receiving actual call, speculatively execute any method with any args
    - Only return when speculative call matches actual call
- Proof of compositionality
  - Global constraints that are solvable because of “flow” properties
- Proofs and counterexamples for tree-based structures
  - Increment/decrement N-counter (weak QC)
  - Increment-only N-counter (QQC)
  - General N-stack (QC)
  - “Properly popped” N-stack (QQC)
    - Pop must wait for concurrent push on same underlying stack
    - Not sufficient for pop to wait on empty stack
    - Proof uses instrumented N-stack that emits a QQC specification trace
  - Tree of N-stacks (same as single N-stack)

Results

- Three characterizations of QQC
  - Call-to-return (given earlier)
  - Return-to-call (à la Herlihy/Wing)
  - Proxy for sequential implementation (flat combiner + speculation)
    - Single thread accesses sequential structure
    - Upon receiving actual call, speculatively execute any method with any args
    - Only return when speculative call matches actual call
- Proof of compositionality
  - Global constraints that are solvable because of “flow” properties
- Proofs and counterexamples for tree-based structures
  - Increment/decrement N-counter (weak QC)
  - Increment-only N-counter (QQC)
  - General N-stack (QC)
  - “Properly popped” N-stack (QQC)
    - Pop must wait for concurrent push on same underlying stack
    - Not sufficient for pop to wait on empty stack
    - Proof uses instrumented N-stack that emits a QQC specification trace
  - Tree of N-stacks (same as single N-stack)

Results

- Three characterizations of QQC
  - Call-to-return (given earlier)
  - Return-to-call (à la Herlihy/Wing)
  - Proxy for sequential implementation (flat combiner + speculation)
    - Single thread accesses sequential structure
    - Upon receiving actual call, speculatively execute any method with any args
    - Only return when speculative call matches actual call
- Proof of compositionality
  - Global constraints that are solvable because of “flow” properties
- Proofs and counterexamples for tree-based structures
  - Increment/decrement N-counter (weak QC)
  - Increment-only N-counter (QQC)
  - General N-stack (QC)
  - “Properly popped” N-stack (QQC)
  - Tree of N-stacks (same as single N-stack)

Results

- Three characterizations of QQC
  - Call-to-return (given earlier)
  - Return-to-call (à la Herlihy/Wing)
  - Proxy for sequential implementation (flat combiner + speculation)
    - Single thread accesses sequential structure
    - Upon receiving actual call, speculatively execute any method with any args
    - Only return when speculative call matches actual call
- Proof of compositionality
  - Global constraints that are solvable because of "flow" properties
- Proofs and counterexamples for tree-based structures
  - Increment/decrement N-counter (weak QC)
  - Increment-only N-counter (QQC)
  - General N-stack (QC)
  - "Properly popped" N-stack (QQC)
    - Pop must wait for concurrent push on same underlying stack
    - Not sufficient for pop to wait on empty stack
    - Proof uses instrumented N-stack that emits a QQC specification trace
  - Tree of N-stacks (same as single N-stack)

Results

- Three characterizations of QQC
  - Call-to-return (given earlier)
  - Return-to-call (à la Herlihy/Wing)
  - Proxy for sequential implementation (flat combiner + speculation)
    - Single thread accesses sequential structure
    - Upon receiving actual call, speculatively execute any method with any args
    - Only return when speculative call matches actual call
- Proof of compositionality
  - Global constraints that are solvable because of "flow" properties
- Proofs and counterexamples for tree-based structures
  - Increment/decrement N-counter (weak QC)
  - Increment-only N-counter (QQC)
  - General N-stack (QC)
  - "Properly popped" N-stack (QQC)
    - Pop must wait for concurrent push on same underlying stack
    - Not sufficient for pop to wait on empty stack
    - Proof uses instrumented N-stack that emits a QQC specification trace
  - Tree of N-stacks (same as single N-stack)

Results

- Three characterizations of QQC
  - Call-to-return (given earlier)
  - Return-to-call (à la Herlihy/Wing)
  - Proxy for sequential implementation (flat combiner + speculation)
    - Single thread accesses sequential structure
    - Upon receiving actual call, speculatively execute any method with any args
    - Only return when speculative call matches actual call
- Proof of compositionality
  - Global constraints that are solvable because of “flow” properties
- Proofs and counterexamples for tree-based structures
  - Increment/decrement N-counter (weak QC)
  - Increment-only N-counter (QQC)
  - General N-stack (QC)
  - “Properly popped” N-stack (QQC)
    - Pop must wait for concurrent push on same underlying stack
    - Not sufficient for pop to wait on empty stack
    - Proof uses instrumented N-stack that emits a QQC specification trace
  - Tree of N-stacks (same as single N-stack)

Results

- Three characterizations of QQC
  - Call-to-return (given earlier)
  - Return-to-call (à la Herlihy/Wing)
  - Proxy for sequential implementation (flat combiner + speculation)
    - Single thread accesses sequential structure
    - Upon receiving actual call, speculatively execute any method with any args
    - Only return when speculative call matches actual call
- Proof of compositionality
  - Global constraints that are solvable because of “flow” properties
- Proofs and counterexamples for tree-based structures
  - Increment/decrement N-counter (weak QC)
  - Increment-only N-counter (QQC)
  - General N-stack (QC)
  - “Properly popped” N-stack (QQC)
    - Pop must wait for concurrent push on same underlying stack
    - Not sufficient for pop to wait on empty stack
    - Proof uses instrumented N-stack that emits a QQC specification trace
  - Tree of N-stacks (same as single N-stack)

Results

- Three characterizations of QQC
  - Call-to-return (given earlier)
  - Return-to-call (à la Herlihy/Wing)
  - Proxy for sequential implementation (flat combiner + speculation)
    - Single thread accesses sequential structure
    - Upon receiving actual call, speculatively execute any method with any args
    - Only return when speculative call matches actual call
- Proof of compositionality
  - Global constraints that are solvable because of "flow" properties
- Proofs and counterexamples for tree-based structures
  - Increment/decrement N-counter (weak QC)
  - Increment-only N-counter (QQC)
  - General N-stack (QC)
  - "Properly popped" N-stack (QQC)
    - Pop must wait for concurrent push on same underlying stack
    - Not sufficient for pop to wait on empty stack
    - Proof uses instrumented N-stack that emits a QQC specification trace
  - Tree of N-stacks (same as single N-stack)

Related work

- *Quantitative Relaxation of Concurrent Data Structures*
  (Henzinger/Kirsch/Payer/Sezgin/Sokolova 2013)
- Incomparable (Examples from Sezgin)
  - Stack that is 1-out-of-order but not QQC:

    $$(_c^{\mathrm{psh}}\ [_a^{\mathrm{psh}}\ ]^{\mathrm{psh}}\ \{_b^{\mathrm{psh}}\ \}^{\mathrm{psh}}\ )^{\mathrm{psh}}\ <^{\mathrm{pop}}\ >_a^{\mathrm{pop}}$$

    However,

    $$(_c^{\mathrm{psh}}\ [_a^{\mathrm{psh}}\ ]^{\mathrm{psh}}\ \{_b^{\mathrm{psh}}\ \}^{\mathrm{psh}}\ <^{\mathrm{pop}}\ >_a^{\mathrm{pop}}\ )^{\mathrm{psh}}$$

    is QQC w.r.t. the stack spec

    $$\{_b^{\mathrm{psh}}\ \}^{\mathrm{psh}}\ [_a^{\mathrm{psh}}\ ]^{\mathrm{psh}}\ <^{\mathrm{pop}}\ >_a^{\mathrm{pop}}\ (_c^{\mathrm{psh}}\ )^{\mathrm{psh}}$$

  - For stacks, it may be that QQC is finer that n-out-of-order (arbitrary n)
  - Queue that is QQC but not $(n-1)$-out-of-order:

    $$(_a^{\mathrm{psh}}\ [_{b_1}^{\mathrm{psh}}\ ]^{\mathrm{psh}}\ [_{b_1}^{\mathrm{psh}}\ ]^{\mathrm{psh}}\ \cdots [_{b_n}^{\mathrm{psh}}\ ]^{\mathrm{psh}}\ \{_c^{\mathrm{psh}}\ \}^{\mathrm{psh}}\ <^{\mathrm{pop}}\ >_c^{\mathrm{pop}}\ )^{\mathrm{psh}}$$

    This is QQC w.r.t.

    $$\{_c^{\mathrm{psh}}\ \}^{\mathrm{psh}}\ [_{b_1}^{\mathrm{psh}}\ ]^{\mathrm{psh}}\ [_{b_1}^{\mathrm{psh}}\ ]^{\mathrm{psh}}\ \cdots [_{b_n}^{\mathrm{psh}}\ ]^{\mathrm{psh}}\ <^{\mathrm{pop}}\ >_c^{\mathrm{pop}}\ (_a^{\mathrm{psh}}\ )^{\mathrm{psh}}$$

Related work

- *Quantitative Relaxation of Concurrent Data Structures*
  (Henzinger/Kirsch/Payer/Sezgin/Sokolova 2013)
- Incomparable (Examples from Sezgin)
  - Stack that is 1-out-of-order but not QQC:

    $$(_c^{\mathrm{psh}}\ [_a^{\mathrm{psh}}\ ]^{\mathrm{psh}}\ \{_b^{\mathrm{psh}}\ \}^{\mathrm{psh}}\ )^{\mathrm{psh}}\ <^{\mathrm{pop}}\ >_a^{\mathrm{pop}}$$

    However,

    $$(_c^{\mathrm{psh}}\ [_a^{\mathrm{psh}}\ ]^{\mathrm{psh}}\ \{_b^{\mathrm{psh}}\ \}^{\mathrm{psh}}\ <^{\mathrm{pop}}\ >_a^{\mathrm{pop}}\ )^{\mathrm{psh}}$$

    is QQC w.r.t. the stack spec

    $$\{_b^{\mathrm{psh}}\ \}^{\mathrm{psh}}\ [_a^{\mathrm{psh}}\ ]^{\mathrm{psh}}\ <^{\mathrm{pop}}\ >_a^{\mathrm{pop}}\ (_c^{\mathrm{psh}}\ )^{\mathrm{psh}}$$

  - For stacks, it may be that QQC is finer that n-out-of-order (arbitrary n)
  - Queue that is QQC but not $(n-1)$-out-of-order:

    $$(_a^{\mathrm{psh}}\ [_{b_1}^{\mathrm{psh}}\ ]^{\mathrm{psh}}\ [_{b_1}^{\mathrm{psh}}\ ]^{\mathrm{psh}} \dots [_{b_n}^{\mathrm{psh}}\ ]^{\mathrm{psh}}\ \{_c^{\mathrm{psh}}\ \}^{\mathrm{psh}}\ <^{\mathrm{pop}}\ >_c^{\mathrm{pop}}\ )^{\mathrm{psh}}$$

    This is QQC w.r.t.

    $$\{_c^{\mathrm{psh}}\ \}^{\mathrm{psh}}\ [_{b_1}^{\mathrm{psh}}\ ]^{\mathrm{psh}}\ [_{b_1}^{\mathrm{psh}}\ ]^{\mathrm{psh}} \dots [_{b_n}^{\mathrm{psh}}\ ]^{\mathrm{psh}}\ <^{\mathrm{pop}}\ >_c^{\mathrm{pop}}\ (_a^{\mathrm{psh}}\ )^{\mathrm{psh}}$$

Related work

- *Quantitative Relaxation of Concurrent Data Structures*
  (Henzinger/Kirsch/Payer/Sezgin/Sokolova 2013)
- Incomparable (Examples from Sezgin)
  - Stack that is 1-out-of-order but not QQC:

$$(_c^{\mathtt{psh}}\ [_a^{\mathtt{psh}}\ ]^{\mathtt{psh}}\ \{_b^{\mathtt{psh}}\ \}^{\mathtt{psh}}\ )^{\mathtt{psh}}\ <^{\mathtt{pop}}\ >_a^{\mathtt{pop}}$$

However,

$$(_c^{\mathtt{psh}}\ [_a^{\mathtt{psh}}\ ]^{\mathtt{psh}}\ \{_b^{\mathtt{psh}}\ \}^{\mathtt{psh}}\ <^{\mathtt{pop}}\ >_a^{\mathtt{pop}}\ )^{\mathtt{psh}}$$

is QQC w.r.t. the stack spec

$$\{_b^{\mathtt{psh}}\ \}^{\mathtt{psh}}\ [_a^{\mathtt{psh}}\ ]^{\mathtt{psh}}\ <^{\mathtt{pop}}\ >_a^{\mathtt{pop}}\ (_c^{\mathtt{psh}}\ )^{\mathtt{psh}}$$

  - For stacks, it may be that QQC is finer that n-out-of-order (arbitrary n)
  - Queue that is QQC but not $(n-1)$-out-of-order:

$$(_a^{\mathtt{psh}}\ [_{b_1}^{\mathtt{psh}}\ ]^{\mathtt{psh}}\ [_{b_1}^{\mathtt{psh}}\ ]^{\mathtt{psh}} \cdots [_{b_n}^{\mathtt{psh}}\ ]^{\mathtt{psh}}\ \{_c^{\mathtt{psh}}\ \}^{\mathtt{psh}}\ <^{\mathtt{pop}}\ >_c^{\mathtt{pop}}\ )^{\mathtt{psh}}$$

This is QQC w.r.t.

$$\{_c^{\mathtt{psh}}\ \}^{\mathtt{psh}}\ [_{b_1}^{\mathtt{psh}}\ ]^{\mathtt{psh}}\ [_{b_1}^{\mathtt{psh}}\ ]^{\mathtt{psh}} \cdots [_{b_n}^{\mathtt{psh}}\ ]^{\mathtt{psh}}\ <^{\mathtt{pop}}\ >_c^{\mathtt{pop}}\ (_a^{\mathtt{psh}}\ )^{\mathtt{psh}}$$

Related work

- *Quantitative Relaxation of Concurrent Data Structures*
  (Henzinger/Kirsch/Payer/Sezgin/Sokolova 2013)
- Incomparable (Examples from Sezgin)
  - Stack that is 1-out-of-order but not QQC:

    $$(_c^{\mathtt{psh}}\ [_a^{\mathtt{psh}}\ ]^{\mathtt{psh}}\ \{_b^{\mathtt{psh}}\ \}^{\mathtt{psh}}\ )^{\mathtt{psh}}\ <^{\mathtt{pop}}\ >_a^{\mathtt{pop}}$$

    However,

    $$(_c^{\mathtt{psh}}\ [_a^{\mathtt{psh}}\ ]^{\mathtt{psh}}\ \{_b^{\mathtt{psh}}\ \}^{\mathtt{psh}}\ <^{\mathtt{pop}}\ >_a^{\mathtt{pop}}\ )^{\mathtt{psh}}$$

    is QQC w.r.t. the stack spec

    $$\{_b^{\mathtt{psh}}\ \}^{\mathtt{psh}}\ [_a^{\mathtt{psh}}\ ]^{\mathtt{psh}}\ <^{\mathtt{pop}}\ >_a^{\mathtt{pop}}\ (_c^{\mathtt{psh}}\ )^{\mathtt{psh}}$$

  - For stacks, it may be that QQC is finer that n-out-of-order (arbitrary n)
  - Queue that is QQC but not $(n-1)$-out-of-order:

    $$(_a^{\mathtt{psh}}\ [_{b_1}^{\mathtt{psh}}\ ]^{\mathtt{psh}}\ [_{b_1}^{\mathtt{psh}}\ ]^{\mathtt{psh}} \cdots [_{b_n}^{\mathtt{psh}}\ ]^{\mathtt{psh}}\ \{_c^{\mathtt{psh}}\ \}^{\mathtt{psh}}\ <^{\mathtt{pop}}\ >_c^{\mathtt{pop}}\ )^{\mathtt{psh}}$$

    This is QQC w.r.t.

    $$\{_c^{\mathtt{psh}}\ \}^{\mathtt{psh}}\ [_{b_1}^{\mathtt{psh}}\ ]^{\mathtt{psh}}\ [_{b_1}^{\mathtt{psh}}\ ]^{\mathtt{psh}} \cdots [_{b_n}^{\mathtt{psh}}\ ]^{\mathtt{psh}}\ <^{\mathtt{pop}}\ >_c^{\mathtt{pop}}\ (_a^{\mathtt{psh}}\ )^{\mathtt{psh}}$$

Proxy characterization code

```
interface Object {
   method run(i:Invocation):Response;
   method predict():Invocation;   }
class QQCProxy<o:Object> {
   field called:ThreadSafeMultiMap<Invocation,Semaphore> = [];
   field returned:ThreadSafeMap   <Semaphore, Response>  = [];
   method run(i:Invocation):Response { // proxy for external access to o
     val m:Semaphore = [];
     called.add(i, m);
     m.wait();
     return returned.remove(m); }
   thread { // single thread to interact with o
     val received:MultiMap<Invocation,Semaphore> = [];
     val executed:MultiMap<Invocation,Response>  = [];
     repeatedly choose {
        choice if called.notEmpty() {
           received.add(called.removeAny());
           val i:Invocation = o.predict();
           val r:Response   = o.run(i);
           executed.add(i, r); }
        choice if exists i in received.keys() intersect executed.keys() {
           val m:Semaphore = received.remove(i);
           val r:Response  = executed.remove(i);
           returned.add(m, r);
           m.signal(); } } } }
```