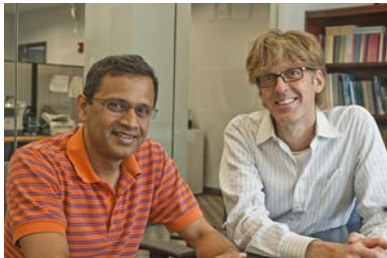


Between Linearizability and Quiescent Consistency



Radha Jagadeesan James Riely

DePaul University
Chicago, USA

ICALP 2014

Linearizability (Herlihy/Wing 1990)

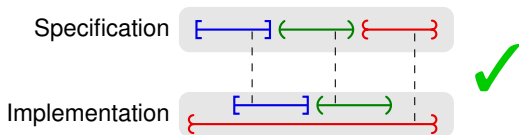
- “Each method call should appear to take effect instantaneously at some moment between its invocation and response.”
(Herlihy/Shavit 2008)
- I.e., for every invocation, exists a *linearization point* such that
 - linearization point is between call and return
 - real-time order corresponds to some sequential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

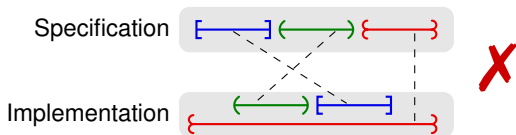
- “Each method call should appear to take effect instantaneously at some moment between its invocation and response.”
(Herlihy/Shavit 2008)
- I.e., for every invocation, exists a *linearization point* such that
 - linearization point is between call and return
 - real-time order corresponds to some sequential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

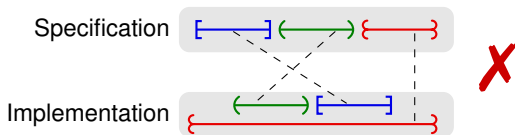
- “Each method call should appear to take effect instantaneously at some moment between its invocation and response.”
(Herlihy/Shavit 2008)
- I.e., for every invocation, exists a *linearization point* such that
 - linearization point is between call and return
 - real-time order corresponds to some sequential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

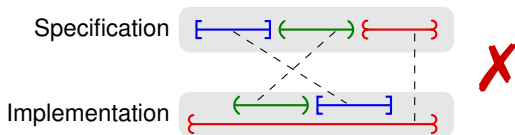
- “Each method call should appear to take effect instantaneously at some moment between its invocation and response.”
(Herlihy/Shavit 2008)
- I.e., for every invocation, exists a *linearization point* such that
 - linearization point is between call and return
 - real-time order corresponds to some sequential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

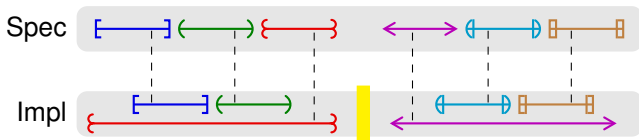
- “Each method call should appear to take effect instantaneously at some moment between its invocation and response.”
(Herlihy/Shavit 2008)
- I.e., for every invocation, exists a *linearization point* such that
 - linearization point is between call and return
 - real-time order corresponds to some sequential 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 ($\text{Lin} \Rightarrow \text{QC}$)
- Compositional
- “Method calls separated *by a period of quiescence* should appear to take effect in their real-time order.”
(Herlihy/Shavit 2008)

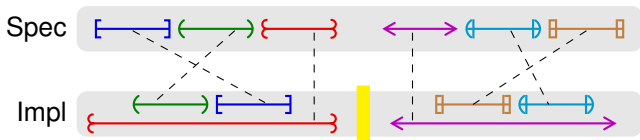


■ is a quiescent point

- 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 ($\text{Lin} \Rightarrow \text{QC}$)
- Compositional
- “Method calls separated *by a period of quiescence* should appear to take effect in their real-time order.”
(Herlihy/Shavit 2008)

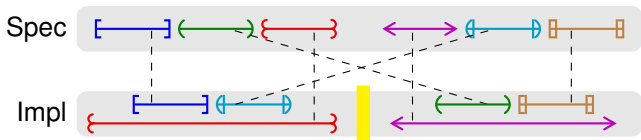


■ is a quiescent point

- 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 ($\text{Lin} \Rightarrow \text{QC}$)
- Compositional
- “Method calls separated *by a period of quiescence* should appear to take effect in their real-time order.”
(Herlihy/Shavit 2008)

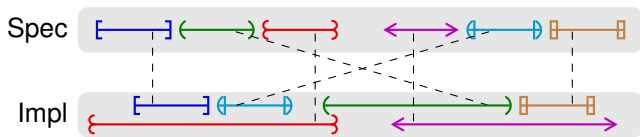


■ is a quiescent point

- 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 ($\text{Lin} \Rightarrow \text{QC}$)
- Compositional
- “Method calls separated *by a period of quiescence* should appear to take effect in their real-time order.”
(Herlihy/Shavit 2008)

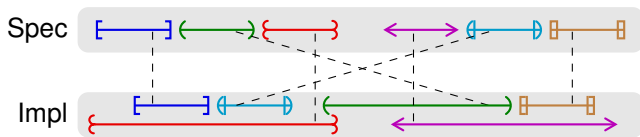


is a quiescent point

- 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 ($\text{Lin} \Rightarrow \text{QC}$)
- Compositional
- “Method calls separated *by a period of quiescence* should appear to take effect in their real-time order.”
(Herlihy/Shavit 2008)

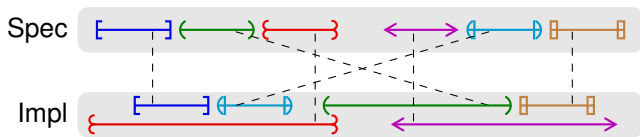


■ is a quiescent point

- 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 ($\text{Lin} \Rightarrow \text{QC}$)
- Compositional
- “Method calls separated *by a period of quiescence* should appear to take effect in their real-time order.”
(Herlihy/Shavit 2008)

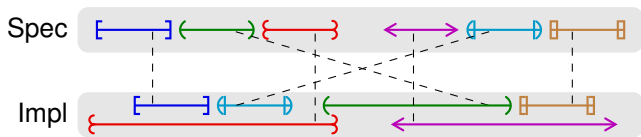


■ is a quiescent point

- 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 ($\text{Lin} \Rightarrow \text{QC}$)
- Compositional
- “Method calls separated *by a period of quiescence* should appear to take effect in their real-time order.”
(Herlihy/Shavit 2008)

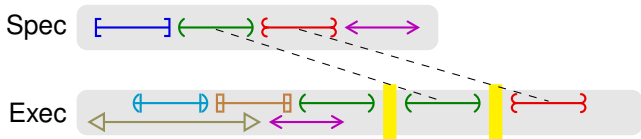


■ is a quiescent point

- 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

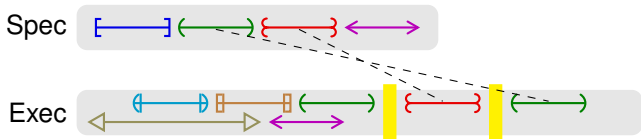
- Abstract view of “step property”
- If at any point accessed sequentially, behaves sequentially



- 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

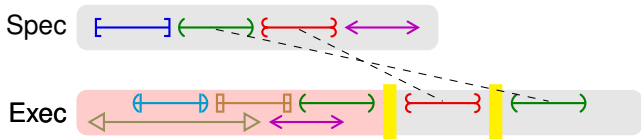
- Abstract view of “step property”
- If at any point accessed sequentially, behaves sequentially



- 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

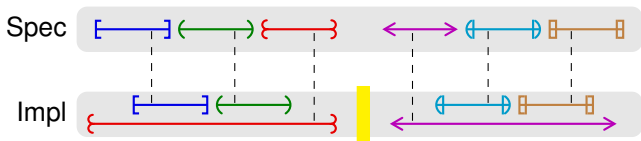
- Abstract view of “step property”
- If at any point accessed sequentially, behaves sequentially



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

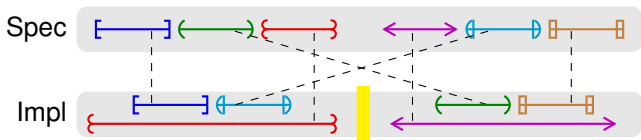
This paper... Quantitative Quiescent Consistency

- Between Linearizability and QC ($\text{Lin} \Rightarrow \text{QQC} \Rightarrow \text{QC}$)
- Compositional
- “Nonlinearizable behavior proportional to number of *early concurrent calls*”



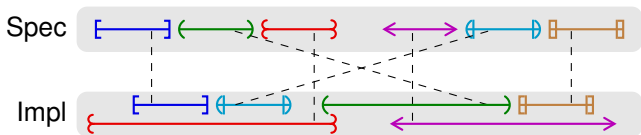
This paper... Quantitative Quiescent Consistency

- Between Linearizability and QC ($\text{Lin} \Rightarrow \text{QQC} \Rightarrow \text{QC}$)
- Compositional
- “Nonlinearizable behavior proportional to number of *early concurrent calls*”



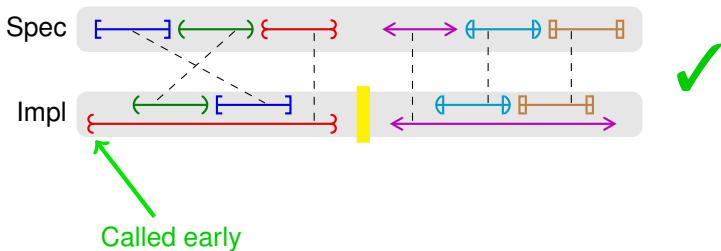
This paper... Quantitative Quiescent Consistency

- Between Linearizability and QC ($\text{Lin} \Rightarrow \text{QQC} \Rightarrow \text{QC}$)
- Compositional
- “Nonlinearizable behavior proportional to number of *early concurrent calls*”



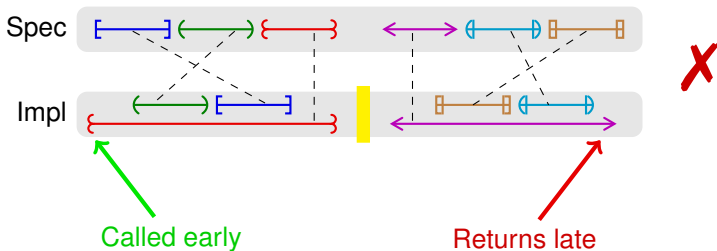
This paper... Quantitative Quiescent Consistency

- Between Linearizability and QC ($\text{Lin} \Rightarrow \text{QQC} \Rightarrow \text{QC}$)
- Compositional
- “Nonlinearizable behavior proportional to number of *early concurrent calls*”



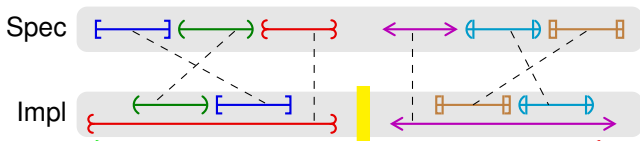
This paper... Quantitative Quiescent Consistency

- Between Linearizability and QC ($\text{Lin} \Rightarrow \text{QQC} \Rightarrow \text{QC}$)
- Compositional
- “Nonlinearizable behavior proportional to number of *early concurrent calls*”



This paper... Quantitative Quiescent Consistency

- Between Linearizability and QC ($\text{Lin} \Rightarrow \text{QQC} \Rightarrow \text{QC}$)
- Compositional
- “Nonlinearizable behavior proportional to number of *early concurrent calls*”



- Early concurrent calls enable out-of-order behavior

Called early

Returns late

Definitions

- Number the call/return pairs of the specification

[1]1 (2)2 {3}3 <4>4 (5)5 [6]6 ...



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

Definitions

- Number the call/return pairs of the specification

Spec



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

Definitions

- Number the call/return pairs of the specification

Spec $[] \leftrightarrow []$



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

Definitions

- Number the call/return pairs of the specification

Spec



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

Definitions

- Number the call/return pairs of the specification


Spec



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

Definitions

- Number the call/return pairs of the specification


Spec 



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

Definitions

- Number the call/return pairs of the specification

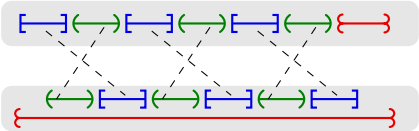
Spec 



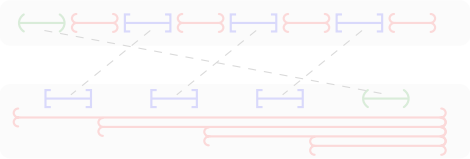
- Linearizability: If $\}_{i} \xrightarrow{\text{precedes}} [_{j}$ then $i < j$ (Return-to-call)
- QC: If $\}_{i} \xrightarrow{\text{quiescent}} (_{j}$ then $i < j$ (Return-to-call)
- Linearizability: If $i < j$ then $[_{i} \xrightarrow{\text{precedes}})_{j}$ (Call-to-return)
- Linearizability: $\{i \mid [_{i} \xrightarrow{\text{precedes}})_{j}\} \supseteq \{1, \dots, j\}$ (Call-to-return)
- QQC: $|\{i \mid [_{i} \xrightarrow{\text{precedes}} \}_{j}\}| \geq |\{1, \dots, j\}| = j$ (Call-to-return)

Interesting examples

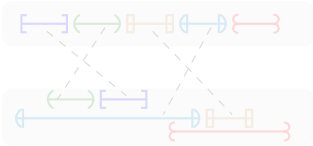
- Enabling early call can be used repeatedly



- Enablers can accumulate

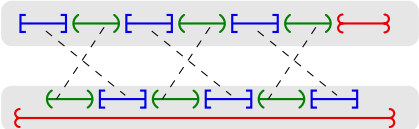


- Enablers can themselves be out-of-order

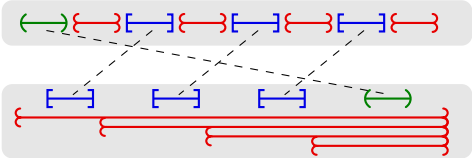


Interesting examples

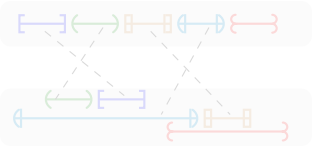
- Enabling early call can be used repeatedly



- Enablers can accumulate

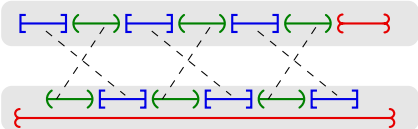


- Enablers can themselves be out-of-order

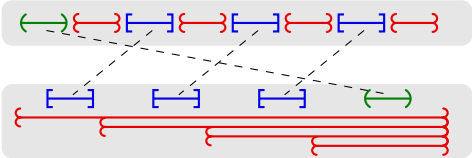


Interesting examples

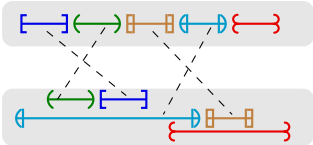
- Enabling early call can be used repeatedly



- 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/Toutou 1995)
(Aiello/Busch/Herlihy/Mavronicolas/Shavit/Toutou 1999)

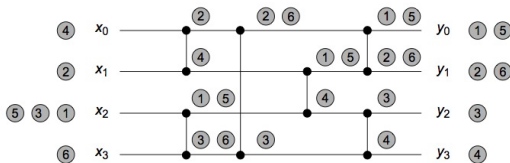
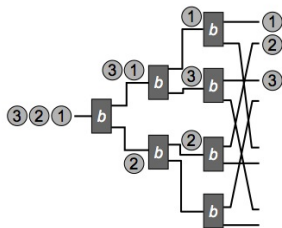
■ Stacks and Bags (aka, Pools)

- Elimination Arrays/Trees (Shavit/Toutou 1995)

■ “Almost” Linearizable

- Experimental results
- Theory involving max/min times (Lynch/Shavit/Shvartsman/Toutou 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; } } }
```

$\langle b = 0, c = [0, 1] \rangle \xrightarrow{\text{inc}} \langle b = 1, c = [0, 1] \rangle \xrightarrow{j_0^{\text{inc}}} \langle b = 1, c = [2, 1] \rangle$
 $\xrightarrow{\text{inc}} \langle b = 0, c = [2, 1] \rangle \xrightarrow{j_1^{\text{inc}}} \langle b = 0, c = [2, 3] \rangle$
 $\xrightarrow{\text{inc}} \langle b = 1, c = [2, 3] \rangle \xrightarrow{j_2^{\text{inc}}} \langle b = 1, c = [4, 3] \rangle$

b=0
/ \
c[0]=0 c[1]=1

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; } } }
```

$\langle b = 0, c = [0, 1] \rangle \xrightarrow{[inc]} \langle b = 1, c = [0, 1] \rangle \xrightarrow{]_0^{inc}} \langle b = 1, c = [2, 1] \rangle$
 $\xrightarrow{(\inc)} \langle b = 0, c = [2, 1] \rangle \xrightarrow{]_1^{inc}} \langle b = 0, c = [2, 3] \rangle$
 $\xrightarrow{(\inc)} \langle b = 1, c = [2, 3] \rangle \xrightarrow{]_2^{inc}} \langle b = 1, c = [4, 3] \rangle$

$b=1$
/ \
 $c[0]=0$ $c[1]=1$
[inc

Behaves sequentially ☺

$\{ \leftarrow \} \{ \leftarrow \} \{ \leftarrow \}$

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; } } }
```

$\langle b = 0, c = [0, 1] \rangle \xrightarrow{[inc]} \langle b = 1, c = [0, 1] \rangle \xrightarrow{]_0^{inc}} \langle b = 1, c = [2, 1] \rangle$
 $\xrightarrow{[inc]} \langle b = 0, c = [2, 1] \rangle \xrightarrow{]_1^{inc}} \langle b = 0, c = [2, 3] \rangle$
 $\xrightarrow{[inc]} \langle b = 1, c = [2, 3] \rangle \xrightarrow{]_2^{inc}} \langle b = 1, c = [4, 3] \rangle$

$b=1$
/ \

$c[0]=2$ $c[1]=1$
[inc]]_0^{inc}

Behaves sequentially ☺

$\left[\text{---} \right] \left(\text{---} \right) \left\{ \text{---} \right\}$

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; } } }
```

$\langle b = 0, c = [0, 1] \rangle \xrightarrow{[inc]} \langle b = 1, c = [0, 1] \rangle \xrightarrow{]_0^{inc}} \langle b = 1, c = [2, 1] \rangle$
 $\xrightarrow{[inc]} \langle b = 0, c = [2, 1] \rangle \xrightarrow{]_1^{inc}} \langle b = 0, c = [2, 3] \rangle$
 $\xrightarrow{[inc]} \langle b = 1, c = [2, 3] \rangle \xrightarrow{]_2^{inc}} \langle b = 1, c = [4, 3] \rangle$

b=0

c[0]=2 c[1]=1
[inc]]_0^{inc} [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; } } }
```

$\langle b = 0, c = [0, 1] \rangle \xrightarrow{[inc]} \langle b = 1, c = [0, 1] \rangle \xrightarrow{]_0^{inc}} \langle b = 1, c = [2, 1] \rangle$
 $\xrightarrow{(<inc)} \langle b = 0, c = [2, 1] \rangle \xrightarrow{]_1^{inc}} \langle b = 0, c = [2, 3] \rangle$
 $\xrightarrow{(<inc)} \langle b = 1, c = [2, 3] \rangle \xrightarrow{]_2^{inc}} \langle b = 1, c = [4, 3] \rangle$

b=0

c[0]=2 c[1]=3
[inc]]_0^{inc} (inc)]_1^{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; } } }
  
```

$$\begin{aligned}
 \langle b = 0, c = [0, 1] \rangle &\xrightarrow{\text{[inc]}} \langle b = 1, c = [0, 1] \rangle \xrightarrow{\text{]}_0^{\text{inc}}} \langle b = 1, c = [2, 1] \rangle \\
 &\xrightarrow{\text{(inc)}} \langle b = 0, c = [2, 1] \rangle \xrightarrow{\text{)}_1^{\text{inc}}} \langle b = 0, c = [2, 3] \rangle \\
 &\xrightarrow{\text{{inc}}} \langle b = 1, c = [2, 3] \rangle \xrightarrow{\text{)}_2^{\text{inc}}} \langle b = 1, c = [4, 3] \rangle
 \end{aligned}$$

b=1

/ \
 c[0]=2 c[1]=3
 [inc]]_0^inc (inc)]_1^inc
 {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; } } }
  
```

$$\begin{aligned}
 \langle b = 0, c = [0, 1] \rangle &\xrightarrow{[inc]} \langle b = 1, c = [0, 1] \rangle \xrightarrow{]_0^{inc}} \langle b = 1, c = [2, 1] \rangle \\
 &\xrightarrow{(inc)} \langle b = 0, c = [2, 1] \rangle \xrightarrow{]_1^{inc}} \langle b = 0, c = [2, 3] \rangle \\
 &\xrightarrow{\{inc\}} \langle b = 1, c = [2, 3] \rangle \xrightarrow{\}_2^{inc}} \langle b = 1, c = [4, 3] \rangle
 \end{aligned}$$

b=1



c[0]=4 c[1]=3
 [inc]]_0^{inc} (inc)]_1^{inc}
 {inc} }_2^{inc}

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; } } }
```

$\langle b = 0, c = [0, 1] \rangle \xrightarrow{\{inc\}} \langle b = 1, c = [0, 1] \rangle$
 $\xrightarrow{\{inc\}} \langle b = 0, c = [0, 1] \rangle \xrightarrow{\}_1^{inc} \langle b = 0, c = [0, 3] \rangle$
 $\xrightarrow{\{inc\}} \langle b = 1, c = [0, 3] \rangle \xrightarrow{\}_0^{inc} \langle b = 1, c = [2, 3] \rangle$
 $\xrightarrow{\}_2^{inc} \langle b = 1, c = [4, 3] \rangle$

$b=0$
/ \
 $c[0]=0$ $c[1]=1$

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; } } }
```

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

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

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

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

b=1

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

{inc

Not Linearizable ☹, but QQC ☺



N-counter — Execution 2

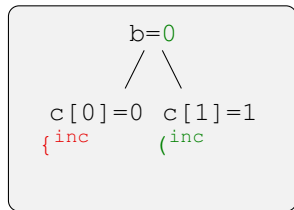
```
class Counter<N:Int> {  
  field b:[0..N-1] = 0; // 1 balancer  
  field c:Int[] = [0, 1, ..., N-1]; // N counters  
  method getAndIncrement():Int {  
    val i:[0..N-1];  
    atomic { i = b; b++; }  
    atomic { val v = c[i]; c[i] += N; return v; } } }
```

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

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

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

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



Not Linearizable ☹, but QQC ☺



N-counter — Execution 2

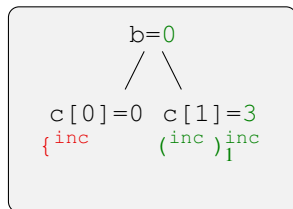
```

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; } } }
  
```

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

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

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

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


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; } } }
```

$\langle b = 0, c = [0, 1] \rangle \xrightarrow{\{inc\}} \langle b = 1, c = [0, 1] \rangle$
 $\xrightarrow{\{inc\}} \langle b = 0, c = [0, 1] \rangle \xrightarrow{\}^1_{inc} \langle b = 0, c = [0, 3] \rangle$
 $\xrightarrow{\{inc\}} \langle b = 1, c = [0, 3] \rangle \xrightarrow{\}^0_{inc} \langle b = 1, c = [2, 3] \rangle$
 $\xrightarrow{\}^2_{inc} \langle b = 1, c = [4, 3] \rangle$

b=1

c[0]=0 c[1]=3
{inc [inc (inc)]_1^inc

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; } } }
  
```

$$\begin{aligned}
 \langle b = 0, c = [0, 1] \rangle &\xrightarrow{\{inc\}} \langle b = 1, c = [0, 1] \rangle \\
 &\xrightarrow{\{inc\}} \langle b = 0, c = [0, 1] \rangle \xrightarrow{\}_1^{inc} \langle b = 0, c = [0, 3] \rangle \\
 &\xrightarrow{\{inc\}} \langle b = 1, c = [0, 3] \rangle \xrightarrow{\}_0^{inc} \langle b = 1, c = [2, 3] \rangle \\
 &\xrightarrow{\}_2^{inc} \langle b = 1, c = [4, 3] \rangle
 \end{aligned}$$

b=1

c[0]=2 c[1]=3

{inc [inc (inc)_1^{inc}
]_0^{inc}

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; } } }
  
```

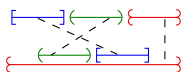
$$\begin{aligned}
 \langle b = 0, c = [0, 1] \rangle &\xrightarrow{\{inc\}} \langle b = 1, c = [0, 1] \rangle \\
 &\xrightarrow{\{inc\}} \langle b = 0, c = [0, 1] \rangle \xrightarrow{\}_1^{inc}} \langle b = 0, c = [0, 3] \rangle \\
 &\xrightarrow{\{inc\}} \langle b = 1, c = [0, 3] \rangle \xrightarrow{\}_0^{inc}} \langle b = 1, c = [2, 3] \rangle \\
 &\xrightarrow{\}_2^{inc}} \langle b = 1, c = [4, 3] \rangle
 \end{aligned}$$

b=1

c[0]=4 c[1]=3

{inc [inc (inc)inc
]inc }inc
0 2

Not Linearizable ☹, but QQC ☺



Increment/Decrement counter

```

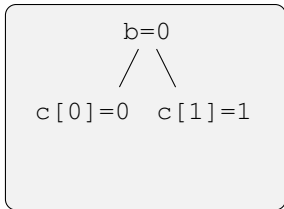
class Counter<N:Int> {
  field b:[0..N-1] = 0; // 1 balancer
  field c: Int[] = [0, 1, ..., N-1]; // N counters
  method getAndIncrement(): Int {
    val i:[0..N-1];
    atomic { i = b; b++; }
    atomic { val v = c[i]; c[i] += N; return v; } }
  method decrementAndGet(): Int {
    val i:[0..N-1];
 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

```

$\langle b = 0, c = [0, 1] \rangle \xrightarrow{\text{[inc}} \langle b = 1, c = [0, 1] \rangle \xrightarrow{\text{[inc}} \langle b = 0, c = [0, 1] \rangle$
 $\xrightarrow{\text{\{dec}} \langle b = 1, c = [0, 1] \rangle \xrightarrow{\text{\{dec}} \langle b = 0, c = [0, 1] \rangle$
 $\xrightarrow{\text{\}]dec}_{-2}} \langle b = 0, c = [-2, 1] \rangle \xrightarrow{\text{\]}inc}_{-2}} \langle b = 0, c = [0, 1] \rangle$
 $\xrightarrow{\text{\]}inc}_1} \langle b = 0, c = [0, 3] \rangle \xrightarrow{\text{\]}dec}_1} \langle b = 0, c = [0, 1] \rangle$

Only weak QC ☹

dec inc inc dec
-2 -2 1 1 not a permutation of any spec trace!



Increment/Decrement counter

```

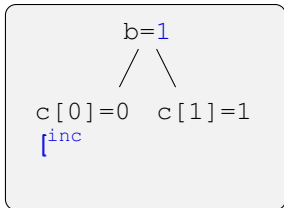
class Counter<N:Int> {
  field b:[0..N-1] = 0; // 1 balancer
  field c: Int[] = [0, 1, ..., N-1]; // N counters
  method getAndIncrement(): Int {
    val i:[0..N-1];
    atomic { i = b; b++; }
    atomic { val v = c[i]; c[i] += N; return v; } }
  method decrementAndGet(): Int {
    val i:[0..N-1];
 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

```

$\langle b = 0, c = [0, 1] \rangle \xrightarrow{\text{[inc}} \langle b = 1, c = [0, 1] \rangle \xrightarrow{\text{[inc}} \langle b = 0, c = [0, 1] \rangle$
 $\xrightarrow{\text{[dec}} \langle b = 1, c = [0, 1] \rangle \xrightarrow{\text{[dec}} \langle b = 0, c = [0, 1] \rangle$
 $\xrightarrow{\text{[dec}_{-2}} \langle b = 0, c = [-2, 1] \rangle \xrightarrow{\text{[inc}_{-2}} \langle b = 0, c = [0, 1] \rangle$
 $\xrightarrow{\text{[inc}_1} \langle b = 0, c = [0, 3] \rangle \xrightarrow{\text{[dec}_1} \langle b = 0, c = [0, 1] \rangle$

Only weak QC ☹

dec inc inc dec 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]; } }
}

```

$$\begin{aligned}
 \langle b = 0, c = [0, 1] \rangle &\xrightarrow{\text{[inc}} \langle b = 1, c = [0, 1] \rangle \xrightarrow{\text{[inc}} \langle b = 0, c = [0, 1] \rangle \\
 &\xrightarrow{\text{\{dec}} \langle b = 1, c = [0, 1] \rangle \xrightarrow{\text{\{dec}} \langle b = 0, c = [0, 1] \rangle \\
 &\xrightarrow{\text{\}dec_{-2}} \langle b = 0, c = [-2, 1] \rangle \xrightarrow{\text{\}inc_{-2}} \langle b = 0, c = [0, 1] \rangle \\
 &\xrightarrow{\text{\}inc_1} \langle b = 0, c = [0, 3] \rangle \xrightarrow{\text{\}dec_1} \langle b = 0, c = [0, 1] \rangle
 \end{aligned}$$

Only weak QC ☹

dec inc inc dec
-2 -2 1 1 not a permutation of any spec trace!

b=0
 / \
 c[0]=0 c[1]=1
 [inc [inc

Increment/Decrement counter

```

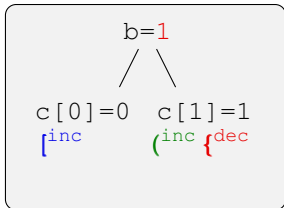
class Counter<N:Int> {
  field b:[0..N-1] = 0; // 1 balancer
  field c:Int[] = [0, 1, ..., N-1]; // N counters
  method getAndIncrement():Int {
    val i:[0..N-1];
    atomic { i = b; b++; }
    atomic { val v = c[i]; c[i] += N; return v; } }
  method decrementAndGet():Int {
    val i:[0..N-1];
    atomic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

```

$\langle b = 0, c = [0, 1] \rangle \xrightarrow{\text{[inc]}} \langle b = 1, c = [0, 1] \rangle \xrightarrow{\text{[inc]}} \langle b = 0, c = [0, 1] \rangle$
 $\xrightarrow{\text{[dec]}} \langle b = 1, c = [0, 1] \rangle \xrightarrow{\text{[dec]}} \langle b = 0, c = [0, 1] \rangle$
 $\xrightarrow{\text{[dec]}} \langle b = 0, c = [-2, 1] \rangle \xrightarrow{\text{[inc]}} \langle b = 0, c = [0, 1] \rangle$
 $\xrightarrow{\text{[inc]}} \langle b = 0, c = [0, 3] \rangle \xrightarrow{\text{[dec]}} \langle b = 0, c = [0, 1] \rangle$

Only weak QC ☹

dec inc inc dec not a permutation of any spec trace!



Increment/Decrement counter

```

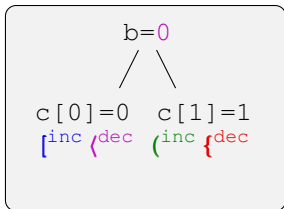
class Counter<N:Int> {
  field b:[0..N-1] = 0; // 1 balancer
  field c:Int[] = [0, 1, ..., N-1]; // N counters
  method getAndIncrement():Int {
    val i:[0..N-1];
    atomic { i = b; b++; }
    atomic { val v = c[i]; c[i] += N; return v; } }
  method decrementAndGet():Int {
    val i:[0..N-1];
    atomic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

```

$$\begin{aligned}
 \langle b = 0, c = [0, 1] \rangle &\xrightarrow{\text{[inc}} \langle b = 1, c = [0, 1] \rangle \xrightarrow{\text{(inc}} \langle b = 0, c = [0, 1] \rangle \\
 &\xrightarrow{\text{{dec}} \langle b = 1, c = [0, 1] \rangle \xrightarrow{\text{<dec}} \langle b = 0, c = [0, 1] \rangle \\
 &\xrightarrow{\text{]>dec}_{-2}} \langle b = 0, c = [-2, 1] \rangle \xrightarrow{\text{[inc}_{-2}} \langle b = 0, c = [0, 1] \rangle \\
 &\xrightarrow{\text{]>inc}_1} \langle b = 0, c = [0, 3] \rangle \xrightarrow{\text{>dec}_1} \langle b = 0, c = [0, 1] \rangle
 \end{aligned}$$

Only weak QC ☹

dec inc inc dec
-2 -2 1 1 not a permutation of any spec trace!



Increment/Decrement counter

```

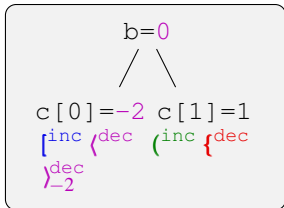
class Counter<N:Int> {
  field b:[0..N-1] = 0; // 1 balancer
  field c:Int[] = [0, 1, ..., N-1]; // N counters
  method getAndIncrement():Int {
    val i:[0..N-1];
    atomic { i = b; b++; }
    atomic { val v = c[i]; c[i] += N; return v; } }
  method decrementAndGet():Int {
    val i:[0..N-1];
    atomic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

```

$$\begin{aligned}
 \langle b = 0, c = [0, 1] \rangle &\xrightarrow{\text{[inc}}} \langle b = 1, c = [0, 1] \rangle \xrightarrow{\text{(inc}}} \langle b = 0, c = [0, 1] \rangle \\
 &\xrightarrow{\text{{dec}}} \langle b = 1, c = [0, 1] \rangle \xrightarrow{\text{<dec}}} \langle b = 0, c = [0, 1] \rangle \\
 &\xrightarrow{\text{>dec}_{-2}} \langle b = 0, c = [-2, 1] \rangle \xrightarrow{\text{[inc}_{-2}} \langle b = 0, c = [0, 1] \rangle \\
 &\xrightarrow{\text{>inc}_1} \langle b = 0, c = [0, 3] \rangle \xrightarrow{\text{{dec}_1}} \langle b = 0, c = [0, 1] \rangle
 \end{aligned}$$

Only weak QC ☹

dec inc inc dec not a permutation of any spec trace!



Increment/Decrement counter

```

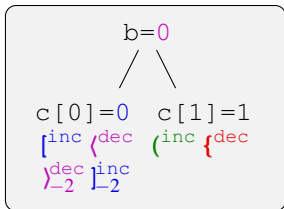
class Counter<N:Int> {
  field b:[0..N-1] = 0; // 1 balancer
  field c: Int[] = [0, 1, ..., N-1]; // N counters
  method getAndIncrement(): Int {
    val i:[0..N-1];
    atomic { i = b; b++; }
    atomic { val v = c[i]; c[i] += N; return v; } }
  method decrementAndGet(): Int {
    val i:[0..N-1];
 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

```

$$\begin{aligned}
 \langle b = 0, c = [0, 1] \rangle &\xrightarrow{\text{[inc}} \langle b = 1, c = [0, 1] \rangle \xrightarrow{\text{(<inc}} \langle b = 0, c = [0, 1] \rangle \\
 &\xrightarrow{\text{[dec}} \langle b = 1, c = [0, 1] \rangle \xrightarrow{\text{(<dec}} \langle b = 0, c = [0, 1] \rangle \\
 &\xrightarrow{\text{>dec}_{-2}} \langle b = 0, c = [-2, 1] \rangle \xrightarrow{\text{[inc}_{-2}} \langle b = 0, c = [0, 1] \rangle \\
 &\xrightarrow{\text{>inc}_1} \langle b = 0, c = [0, 3] \rangle \xrightarrow{\text{>dec}_1} \langle b = 0, c = [0, 1] \rangle
 \end{aligned}$$

Only weak QC ☹

decincincdec not a permutation of any spec trace!



Increment/Decrement counter

```

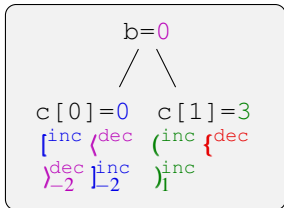
class Counter<N:Int> {
  field b:[0..N-1] = 0; // 1 balancer
  field c: Int[] = [0, 1, ..., N-1]; // N counters
  method getAndIncrement(): Int {
    val i:[0..N-1];
    atomic { i = b; b++; }
    atomic { val v = c[i]; c[i] += N; return v; } }
  method decrementAndGet(): Int {
    val i:[0..N-1];
 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

```

$$\begin{aligned}
 \langle b = 0, c = [0, 1] \rangle &\xrightarrow{\text{[inc}} \langle b = 1, c = [0, 1] \rangle \xrightarrow{\text{(inc}} \langle b = 0, c = [0, 1] \rangle \\
 &\xrightarrow{\text{{dec}} \langle b = 1, c = [0, 1] \rangle \xrightarrow{\text{<dec}} \langle b = 0, c = [0, 1] \rangle \\
 &\xrightarrow{\text{]dec}_{-2}} \langle b = 0, c = [-2, 1] \rangle \xrightarrow{\text{[inc}_{-2}} \langle b = 0, c = [0, 1] \rangle \\
 &\xrightarrow{\text{]inc}_1} \langle b = 0, c = [0, 3] \rangle \xrightarrow{\text{]dec}_1} \langle b = 0, c = [0, 1] \rangle
 \end{aligned}$$

Only weak QC ☹

decincincdec not a permutation of any spec trace!



Increment/Decrement counter

```

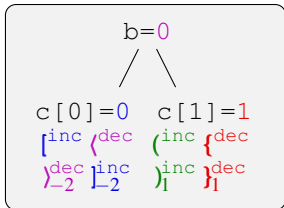
class Counter<N:Int> {
  field b:[0..N-1] = 0; // 1 balancer
  field c: Int[] = [0, 1, ..., N-1]; // N counters
  method getAndIncrement():Int {
    val i:[0..N-1];
    atomic { i = b; b++; }
    atomic { val v = c[i]; c[i] += N; return v; } }
  method decrementAndGet():Int {
    val i:[0..N-1];
   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

```

$$\begin{aligned}
 \langle b = 0, c = [0, 1] \rangle &\xrightarrow{\text{[inc}} \langle b = 1, c = [0, 1] \rangle \xrightarrow{\text{(inc}} \langle b = 0, c = [0, 1] \rangle \\
 &\xrightarrow{\text{{dec}} \langle b = 1, c = [0, 1] \rangle \xrightarrow{\text{<dec}} \langle b = 0, c = [0, 1] \rangle \\
 &\xrightarrow{\text{>dec}_{-2}} \langle b = 0, c = [-2, 1] \rangle \xrightarrow{\text{[inc}_{-2}} \langle b = 0, c = [0, 1] \rangle \\
 &\xrightarrow{\text{>inc}_1} \langle b = 0, c = [0, 3] \rangle \xrightarrow{\text{>dec}_1} \langle b = 0, c = [0, 1] \rangle
 \end{aligned}$$

Only weak QC ☹

dec inc inc dec not a permutation of any spec trace!

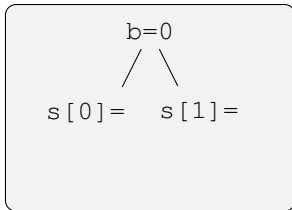
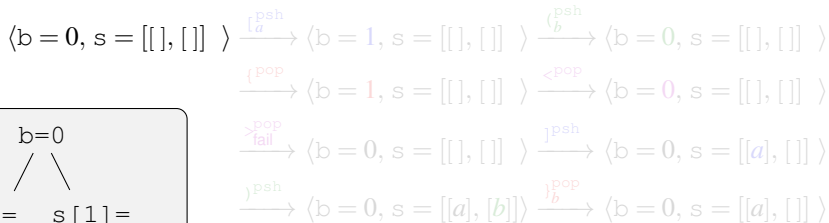


Stack

```

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; } } }

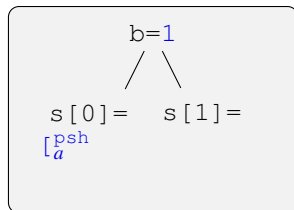
```



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; } } }
```

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



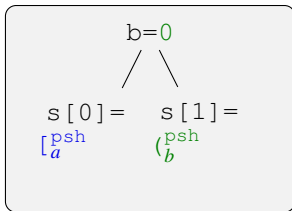
Stack

```

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; } } }

```

$\langle b = 0, s = [[], []] \rangle \xrightarrow{\text{push}_a} \langle b = 1, s = [[], []] \rangle \xrightarrow{\text{push}_b} \langle b = 0, s = [[], []] \rangle$
 $\xrightarrow{\text{pop}} \langle b = 1, s = [[], []] \rangle \xrightarrow{\text{pop}} \langle b = 0, s = [[], []] \rangle$
 $\xrightarrow{\text{pop}_{fail}} \langle b = 0, s = [[], []] \rangle \xrightarrow{\text{push}} \langle b = 0, s = [[a], []] \rangle$
 $\xrightarrow{\text{push}} \langle b = 0, s = [[a], [b]] \rangle \xrightarrow{\text{pop}_b} \langle b = 0, s = [[a], []] \rangle$



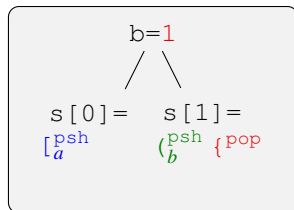
Stack

```

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; } } }

```

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

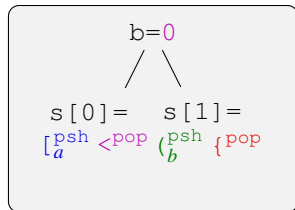


Stack

```

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; } } }
  
```

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



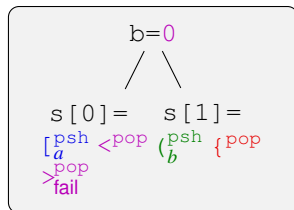
Stack

```

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; } } }

```

$\langle b = 0, s = [[], []] \rangle \xrightarrow{\text{push}_a} \langle b = 1, s = [[], []] \rangle \xrightarrow{\text{push}_b} \langle b = 0, s = [[], []] \rangle$
 $\xrightarrow{\text{pop}} \langle b = 1, s = [[], []] \rangle \xrightarrow{\text{pop}} \langle b = 0, s = [[], []] \rangle$
 $\xrightarrow{\text{pop}_{fail}} \langle b = 0, s = [[], []] \rangle \xrightarrow{\text{push}} \langle b = 0, s = [[a], []] \rangle$
 $\xrightarrow{\text{push}} \langle b = 0, s = [[a], [b]] \rangle \xrightarrow{\text{pop}_b} \langle b = 0, s = [[a], []] \rangle$



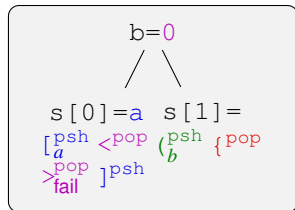
Stack

```

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; } } }

```

$\langle b = 0, s = [[], []] \rangle \xrightarrow{\text{push}_a} \langle b = 1, s = [[], []] \rangle \xrightarrow{\text{push}_b} \langle b = 0, s = [[], []] \rangle$
 $\xrightarrow{\text{pop}} \langle b = 1, s = [[], []] \rangle \xrightarrow{\text{pop}} \langle b = 0, s = [[], []] \rangle$
 $\xrightarrow{\text{pop}_{fail}} \langle b = 0, s = [[], []] \rangle \xrightarrow{\text{push}} \langle b = 0, s = [[a], []] \rangle$
 $\xrightarrow{\text{push}} \langle b = 0, s = [[a], [b]] \rangle \xrightarrow{\text{pop}_b} \langle b = 0, s = [[a], []] \rangle$



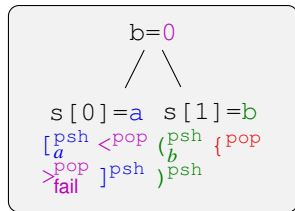
Stack

```

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; } } }

```

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



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; } } }

```

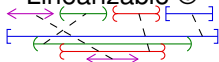
$\langle b = 0, s = [[], []] \rangle \xrightarrow{\text{[a]}^{\text{psh}}} \langle b = 1, s = [[], []] \rangle \xrightarrow{\text{[b]}^{\text{psh}}} \langle b = 0, s = [[], []] \rangle$
 $\xrightarrow{\text{[pop]}} \langle b = 1, s = [[], []] \rangle \xrightarrow{\text{[pop]}} \langle b = 0, s = [[],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

b=0

s[0]=a s[1]=

[a]^{psh} [pop] [b]^{psh} [pop]
 [pop fail] [psh] [psh] [pop]

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; } } }
  
```

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

$b=0$
 $\swarrow \searrow$
 $s[0] = \quad s[1] =$

Not even quiescent consistent ☹



\leftrightarrow should pop from $\{ \}$ or $\langle \rangle$, 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; } } }

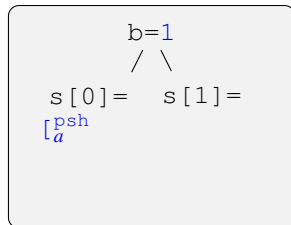
```

$$\begin{aligned}
 \langle b = 0, s = [[], []] \rangle &\xrightarrow{[a]^{psh}} \langle b = 1, s = [[], []] \rangle \xrightarrow{]^{psh}} \langle b = 1, s = [[a], []] \rangle \\
 &\xrightarrow{\{b\}^{psh}} \langle b = 0, s = [[a], []] \rangle \xrightarrow{]^{psh}} \langle b = 0, s = [[a], [b]] \rangle \\
 &\xrightarrow{[c]^{psh}} \langle b = 1, s = [[a], [b]] \rangle \xrightarrow{\leftarrow^{pop}} \langle b = 0, s = [[a], [b]] \rangle \\
 &\xrightarrow{>a^{pop}} \langle b = 0, s = [[], [b]] \rangle \xrightarrow{]^{psh}} \langle b = 0, s = [[c], [b]] \rangle
 \end{aligned}$$

Not even quiescent consistent ☹



\longleftrightarrow should pop from $\{\leftrightarrow\}$ or \leftarrow , but not \leftarrow



Stack — Execution 2

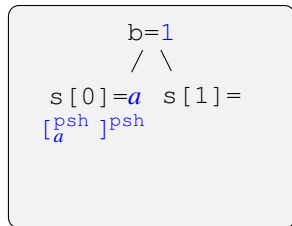
```

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; } } }
  
```

$$\begin{aligned}
 \langle b = 0, s = [[], []] \rangle &\xrightarrow{[a]^{psh}} \langle b = 1, s = [[], []] \rangle \xrightarrow{[b]^{psh}} \langle b = 1, s = [[a], []] \rangle \\
 &\xrightarrow{[c]^{psh}} \langle b = 0, s = [[a], []] \rangle \xrightarrow{[d]^{psh}} \langle b = 0, s = [[a], [b]] \rangle \\
 &\xrightarrow{[e]^{psh}} \langle b = 1, s = [[a], [b]] \rangle \xrightarrow{[f]^{pop}} \langle b = 0, s = [[a], [b]] \rangle \\
 &\xrightarrow{[g]^{pop}} \langle b = 0, s = [[], [b]] \rangle \xrightarrow{[h]^{psh}} \langle b = 0, s = [[c], [b]] \rangle
 \end{aligned}$$

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; } } }

```

$\langle b = 0, s = [[], []] \rangle \xrightarrow{[a]^{psh}} \langle b = 1, s = [[], []] \rangle \xrightarrow{[b]^{psh}} \langle b = 1, s = [[a], []] \rangle$
 $\xrightarrow{\{b\}^{psh}} \langle b = 0, s = [[a], []] \rangle \xrightarrow{[c]^{psh}} \langle b = 0, s = [[a], [b]] \rangle$
 $\xrightarrow{[c]^{psh}} \langle b = 1, s = [[a], [b]] \rangle \xrightarrow{\langle pop \rangle} \langle b = 0, s = [[a], [b]] \rangle$
 $\xrightarrow{[a]^{pop}} \langle b = 0, s = [[], [b]] \rangle \xrightarrow{[c]^{psh}} \langle b = 0, s = [[c], [b]] \rangle$
 Not even quiescent consistent ☹

b=0
/ \

s[0]=a s[1]=

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

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; } } }

```

$$\begin{aligned}
 \langle b = 0, s = [[], []] \rangle &\xrightarrow{[a]^{psh}} \langle b = 1, s = [[], []] \rangle \xrightarrow{[a]^{psh}} \langle b = 1, s = [[a], []] \rangle \\
 &\xrightarrow{\{b\}^{psh}} \langle b = 0, s = [[a], []] \rangle \xrightarrow{[b]^{psh}} \langle b = 0, s = [[a], [b]] \rangle \\
 &\xrightarrow{[c]^{psh}} \langle b = 1, s = [[a], [b]] \rangle \xrightarrow{\langle pop \rangle} \langle b = 0, s = [[a], [b]] \rangle \\
 &\xrightarrow{>[a]^{pop}} \langle b = 0, s = [[], [b]] \rangle \xrightarrow{[c]^{psh}} \langle b = 0, s = [[c], [b]] \rangle
 \end{aligned}$$

Not even quiescent consistent ☹

b=0
/ \

s[0]=a s[1]=b
[^{psh}a] ^{psh} {^{psh}b} ^{psh}

← →

↔ 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; } } }

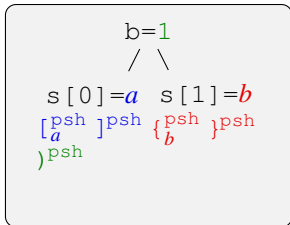
```

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

Not even quiescent consistent ☹



\leftrightarrow should pop from $\{ \}$ or $\langle \rangle$, 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; } } }

```

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

Not even quiescent consistent ☹



\leftrightarrow should pop from $\{ \leftrightarrow \}$ or $\langle \leftrightarrow \rangle$, but not $\{ \leftrightarrow \}$

b=0

/ \

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

[a]psh]psh {b}psh }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; } } }
  
```

$$\begin{aligned}
 \langle b = 0, s = [[], []] \rangle &\xrightarrow{[a]^{push}} \langle b = 1, s = [[], []] \rangle \xrightarrow{[a]^{push}} \langle b = 1, s = [[a], []] \rangle \\
 &\xrightarrow{\{b\}^{push}} \langle b = 0, s = [[a], []] \rangle \xrightarrow{[b]^{push}} \langle b = 0, s = [[a], [b]] \rangle \\
 &\xrightarrow{[c]^{push}} \langle b = 1, s = [[a], [b]] \rangle \xrightarrow{\langle pop \rangle} \langle b = 0, s = [[a], [b]] \rangle \\
 &\xrightarrow{\langle pop \rangle_a} \langle b = 0, s = [[], [b]] \rangle \xrightarrow{[c]^{push}} \langle b = 0, s = [[c], [b]] \rangle
 \end{aligned}$$

Not even quiescent consistent ☹



\leftrightarrow should pop from $\{ \leftrightarrow \}$ or $\langle \leftrightarrow \rangle$, but not $\{ \leftrightarrow \}$

b=0
/ \

s[0] = s[1] = b

[a]^{push}]^{push} {b}^{push}]^{push}
)^{push} <pop

>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; } } }

```

$$\begin{aligned}
 \langle b = 0, s = [[], []] \rangle &\xrightarrow{[a]^{push}} \langle b = 1, s = [[], []] \rangle \xrightarrow{[a]^{push}} \langle b = 1, s = [[a], []] \rangle \\
 &\xrightarrow{\{b\}^{push}} \langle b = 0, s = [[a], []] \rangle \xrightarrow{[b]^{push}} \langle b = 0, s = [[a], [b]] \rangle \\
 &\xrightarrow{[c]^{push}} \langle b = 1, s = [[a], [b]] \rangle \xrightarrow{<pop} \langle b = 0, s = [[a], [b]] \rangle \\
 &\xrightarrow{>a^{pop}} \langle b = 0, s = [[_], [b]] \rangle \xrightarrow{[c]^{push}} \langle b = 0, s = [[c], [b]] \rangle
 \end{aligned}$$

b=0
/ \

s[0]=c s[1]=b
 [a]push [b]push {b}push
)push <pop
 >a^pop)push

Not even quiescent consistent ☹

[←] [→] [↔]

↔ 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)
 - Proof that wait for concurrent push on same underlying stack
 - Proof sufficient to pop by wait on empty stack
 - Proof uses “reduced” N -stack that meets a QCC specification
- Tree of N -stacks (same as single N -stack)

Results

■ Three characterizations of QCC

- 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 QCC)
- Increment-only N -counter (QCC)
- General N -stack (QCC)
- “Properly popped” N -stack (QCC)
 - Can model wait for concurrent push on same underlying stack
 - Can model popper to wait for empty stack
 - Proof uses unbounded N -stack that meets QCC specification
- Tree of N -stacks (same as single N -stack)

Return-to-call characterization

Linearizability:

$\forall \text{prefix/suffix} = \text{exec}$

$\forall \text{ret} \in \text{prefix}$

$\forall \text{call} \in \text{suffix}$

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

Return-to-call characterization

Quiescent consistency:

$\forall \text{prefix/suffix} = \text{exec}$

if *prefix* has 0 open calls, then

$\forall \text{ret} \in \text{prefix}$

$\forall \text{call} \in \text{suffix}$

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

Return-to-call characterization

QQC:

$\forall \text{prefix/suffix} = \text{exec}$

if *prefix* has k open/early calls, then there exists $|\text{ignoredCalls}| \leq k$

$\forall \text{ret} \in \text{prefix}$

$\forall \text{call} \in \text{suffix} - \text{ignoredCalls}$

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

Results

■ Three characterizations of QCC

- 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 QCC)
- Increment-only N -counter (QCC)
- General N -stack (QCC)
- “Properly popped” N -stack (QCC)
 - This model is not for concurrent push/pop same underlying stack
 - This model is open to work on singly stack
 - This model does not generalize to stack that needs a QCC specification base
- 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 QCC)
 - Increment-only N -counter (QCC)
 - General N -stack (QCC)
 - “Properly popped” N -stack (QCC)
 - This model is not for concurrent push/pop on same underlying stack
 - This model is not for push/pop on same stack
 - This model is not for concurrent push/pop on same stack
 - 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 QOC)
- Increment-only N -counter (QOC)
- General N -stack (QOC)
- “Properly popped” N -stack (QOC)
- 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:

$(\text{push } [c] \text{ push } [a] \text{ push } [b] \text{ push } [a] \text{ push } [c] \text{ pop } [a])$

However,

$(\text{push } [c] \text{ push } [a] \text{ push } [b] \text{ push } [a] \text{ pop } [a] \text{ pop } [c]) \text{ push } [c]$

is QQC w.r.t. the stack spec

$(\text{push } [b] \text{ push } [a] \text{ push } [a] \text{ pop } [a] \text{ pop } [c] \text{ push } [c])$

- For stacks, it may be that QQC is finer than n -out-of-order (arbitrary n)

- Queue that is QQC but not $(n-1)$ -out-of-order:

$(\text{push } [a] \text{ push } [b_1] \text{ push } [b_1] \text{ push } \dots \text{ push } [b_n] \text{ push } [c] \text{ push } [c] \text{ pop } [c] \text{ pop } [c])$

This is QQC w.r.t.

$(\text{push } [c] \text{ push } [b_1] \text{ push } [b_1] \text{ push } \dots \text{ push } [b_n] \text{ push } [c] \text{ pop } [c] \text{ pop } [a])$

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:

$(\text{psh}_c [\text{psh}_a] \text{psh} \{\text{psh}_b\} \text{psh}) \text{psh} \langle \text{pop} \rangle_a$

However,

$(\text{psh}_c [\text{psh}_a] \text{psh} \{\text{psh}_b\} \text{psh} \langle \text{pop} \rangle_a \text{pop}) \text{psh}$

is QQC w.r.t. the stack spec

$\{\text{psh}_b\} \text{psh} [\text{psh}_a] \text{psh} \langle \text{pop} \rangle_a \text{pop} (\text{psh}_c) \text{psh}$

- For stacks, it may be that QQC is finer than n -out-of-order (arbitrary n)

- Queue that is QQC but not $(n-1)$ -out-of-order:

$(\text{psh}_a [\text{psh}_{b_1}] \text{psh} [\text{psh}_{b_1}] \text{psh} \dots [\text{psh}_{b_n}] \text{psh} \{\text{psh}_c\} \text{psh} \langle \text{pop} \rangle_c \text{pop}) \text{psh}$

This is QQC w.r.t.

$\{\text{psh}_c\} \text{psh} [\text{psh}_{b_1}] \text{psh} [\text{psh}_{b_1}] \text{psh} \dots [\text{psh}_{b_n}] \text{psh} \langle \text{pop} \rangle_c \text{pop} (\text{psh}_a) \text{psh}$

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:

$$(\overset{\text{psh}}{c} [\overset{\text{psh}}{a}] \text{psh} \{ \overset{\text{psh}}{b} \} \text{psh}) \overset{\text{psh}}{c} < \overset{\text{pop}}{c} > \overset{\text{pop}}{a}$$

However,

$$(\overset{\text{psh}}{c} [\overset{\text{psh}}{a}] \text{psh} \{ \overset{\text{psh}}{b} \} \text{psh} < \overset{\text{pop}}{c} > \overset{\text{pop}}{a}) \overset{\text{psh}}{c}$$

is QQC w.r.t. the stack spec

$$\{ \overset{\text{psh}}{b} \} \text{psh} [\overset{\text{psh}}{a}] \text{psh} < \overset{\text{pop}}{c} > \overset{\text{pop}}{a} (\overset{\text{psh}}{c}) \text{psh}$$

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

$$(\overset{\text{psh}}{a} [\overset{\text{psh}}{b_1}] \text{psh} [\overset{\text{psh}}{b_1}] \text{psh} \dots [\overset{\text{psh}}{b_n}] \text{psh} \{ \overset{\text{psh}}{c} \} \text{psh} < \overset{\text{pop}}{c} > \overset{\text{pop}}{c}) \overset{\text{psh}}{c}$$

This is QQC w.r.t.

$$\{ \overset{\text{psh}}{c} \} \text{psh} [\overset{\text{psh}}{b_1}] \text{psh} [\overset{\text{psh}}{b_1}] \text{psh} \dots [\overset{\text{psh}}{b_n}] \text{psh} < \overset{\text{pop}}{c} > \overset{\text{pop}}{c} (\overset{\text{psh}}{a}) \text{psh}$$

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:

$$(c^{psh} [a^{psh}]^{psh} \{b^{psh}\}^{psh})^{psh} <pop >_a^{pop}$$

However,

$$(c^{psh} [a^{psh}]^{psh} \{b^{psh}\}^{psh} <pop >_a^{pop})^{psh}$$

is QQC w.r.t. the stack spec

$$\{b^{psh}\}^{psh} [a^{psh}]^{psh} <pop >_a^{pop} (c^{psh})^{psh}$$

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

$$(a^{psh} [b_1^{psh}]^{psh} [b_1^{psh}]^{psh} \dots [b_n^{psh}]^{psh} \{c^{psh}\}^{psh} <pop >_c^{pop})^{psh}$$

This is QQC w.r.t.

$$\{c^{psh}\}^{psh} [b_1^{psh}]^{psh} [b_1^{psh}]^{psh} \dots [b_n^{psh}]^{psh} <pop >_c^{pop} (a^{psh})^{psh}$$

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(); } } } }
```