Quantitatively Relaxed Concurrent Data Structures

Thomas A. Henzinger IST Austria
Christoph M. Kirsch University of Salzburg
Hannes Payer University of Salzburg
Ali Sezgin IST Austria
Ana Sokolova University of Salzburg

Dagstuhl RiSE 16.11.2012

Semantics of concurrent data structures

- Sequential specification - set of legal sequences
- Correctness condition - linearizability

Semantics of concurrent data structures

Stack - legal sequence

push(a)push(b)pop(b)

- Sequential specification - set of legal sequences
- Correctness condition - linearizability

Semantics of concurrent data structures

Stack - legal sequence

push(a)push(b)pop(b)

- Sequential specification - set of legal sequences
- Correctness condition - linearizability

Stack - concurrent history

begin-push(a)begin-push(b) end-push(a) end-push(b)begin-pop(b)end-pop(b)

Semantics of concurrent data structures

Stack - legal sequence

push(a)push(b)pop(b)

- Sequential specification - set of legal sequences
- Correctness condition - linearizability

linearizable wrt seq.spec.

Stack - concurrent history

begin-push(a)begin-push(b) end-push(a) end-push(b)begin-pop(b)end-pop(b)

Semantics of concurrent data structures

Stack - legal sequence

push(a)push(b)pop(b)

we relax this

- Sequential specification - set of legal sequences

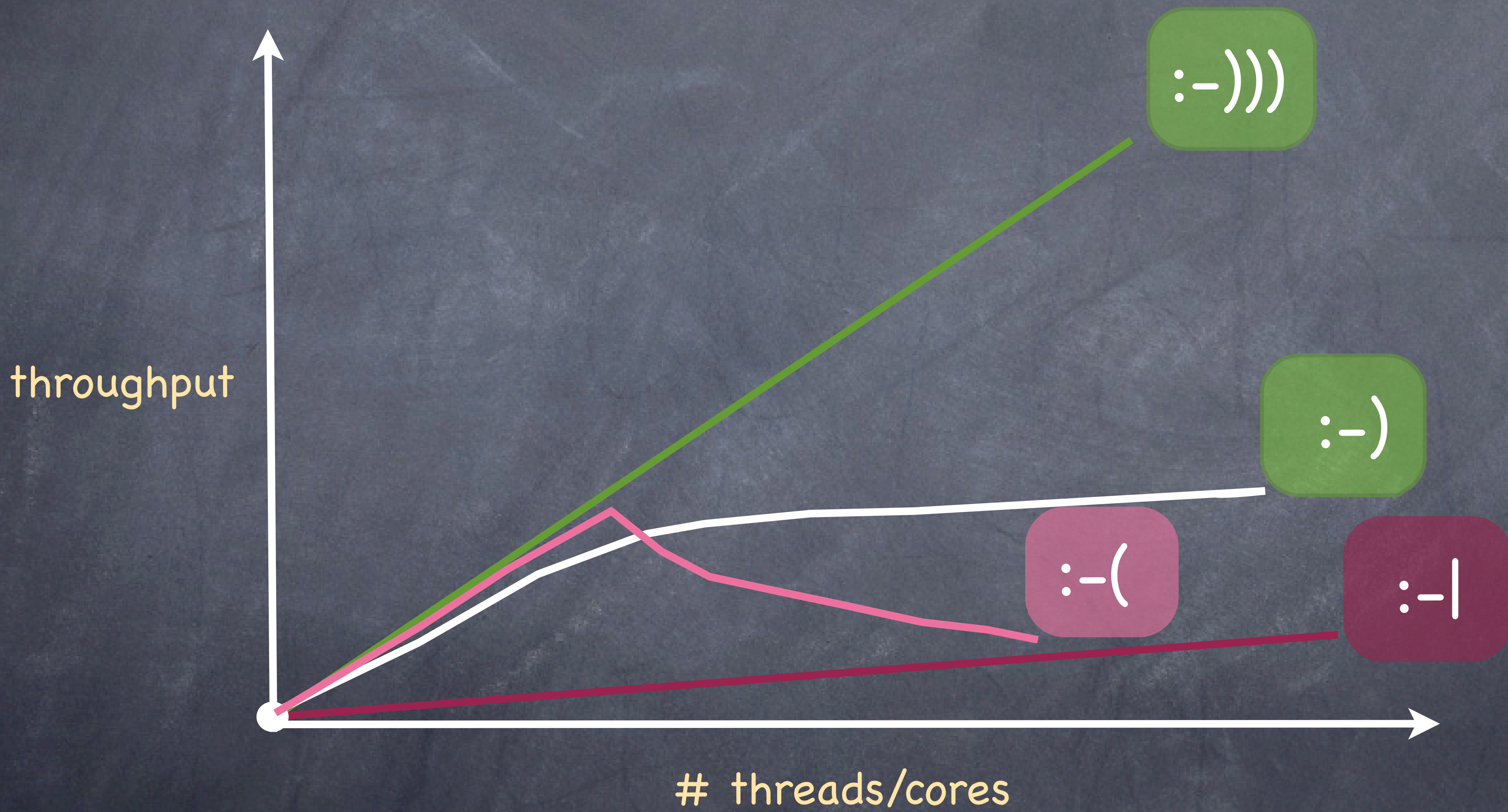
linearizable wrt seq.spec.

- Correctness condition - linearizability

Stack - concurrent history

begin-push(a)begin-push(b) end-push(a) end-push(b)begin-pop(b)end-pop(b)

Performance and scalability



The goal

- Trading correctness for performance
- In a controlled way with quantitative bounds

measure the error from correct behavior

The goal

Stack - incorrect behavior

push(a)push(b)push(c)pop(a)pop(b)

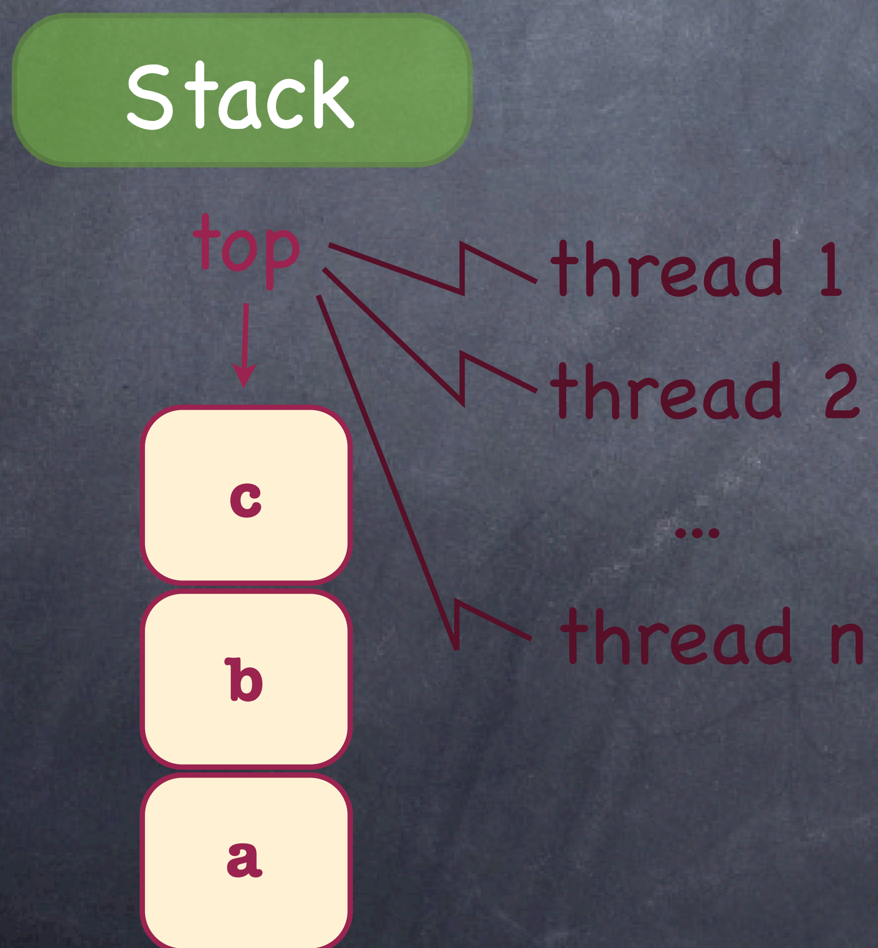
- Trading correctness for performance
- In a controlled way with quantitative bounds

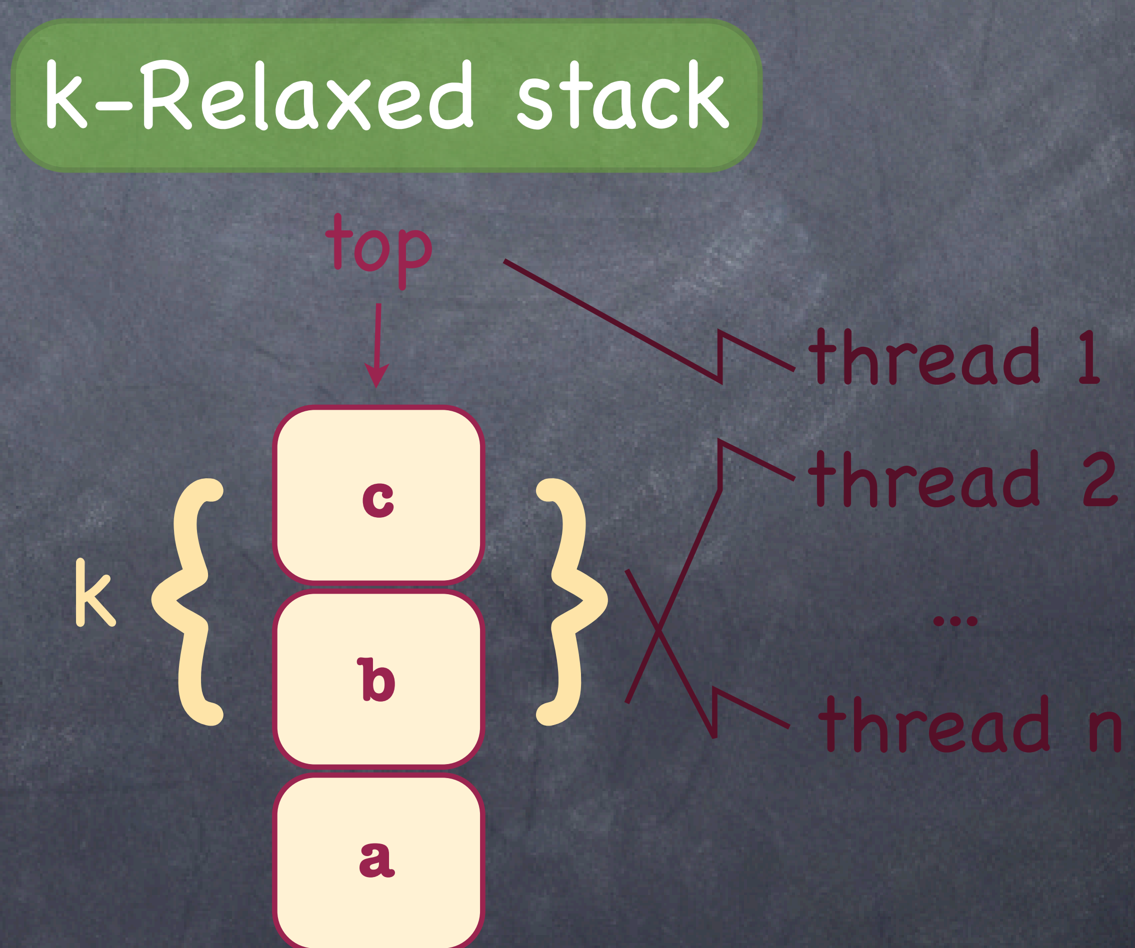
correct in a relaxed stack
... 2-relaxed? 3-relaxed?

measure the error from correct behavior

Why relax?

- It is interesting
- Provides potential for better performing concurrent implementations





What we have

- Framework
- Generic examples
- Concrete relaxation examples
- Efficient concurrent implementations

for semantic relaxations

out-of-order / stuttering

stacks, queues, priority queues,.. / CAS, shared counter

of relaxation instances

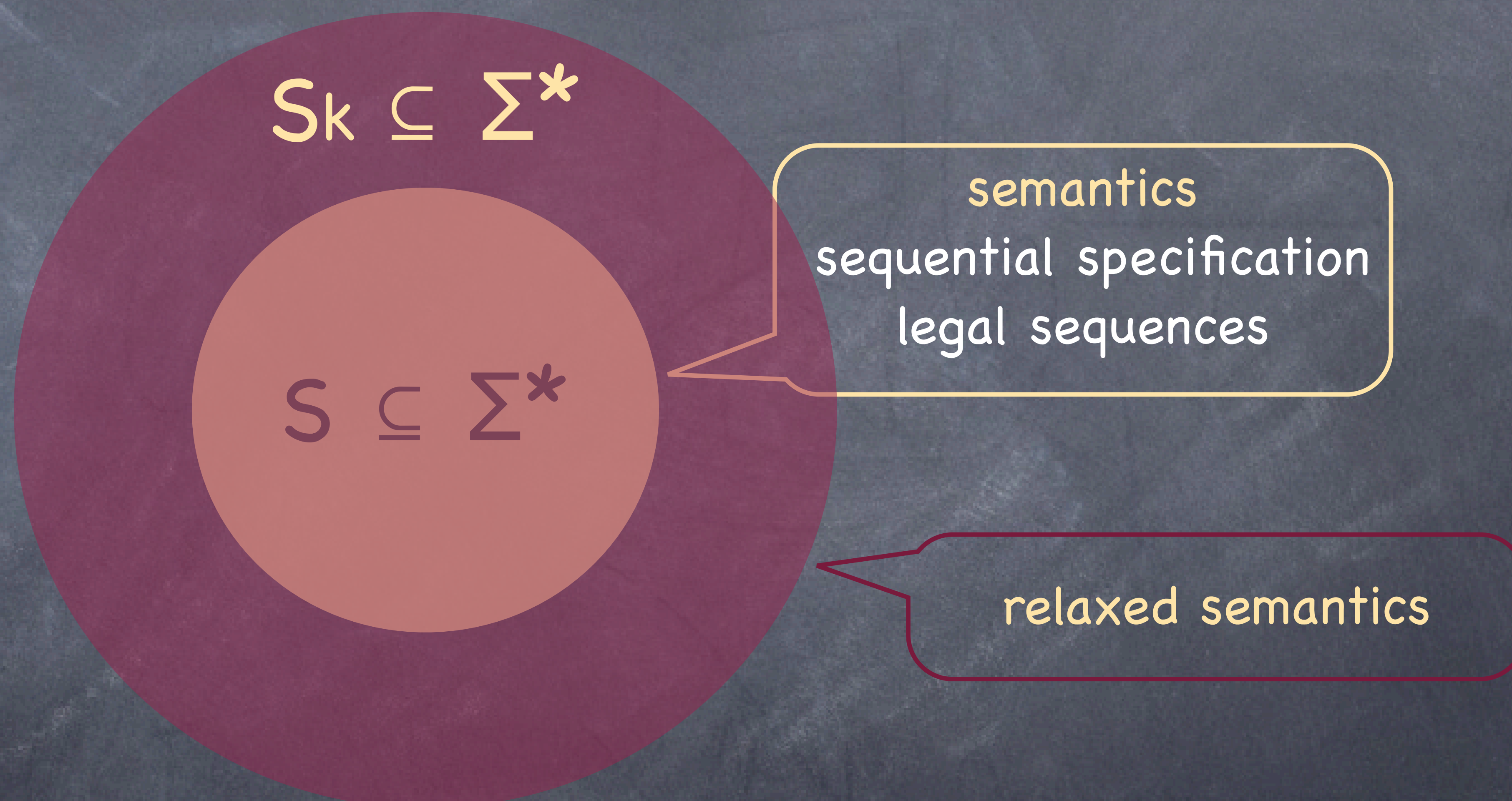
The big picture

$S \subseteq \Sigma^*$

semantics
sequential specification
legal sequences

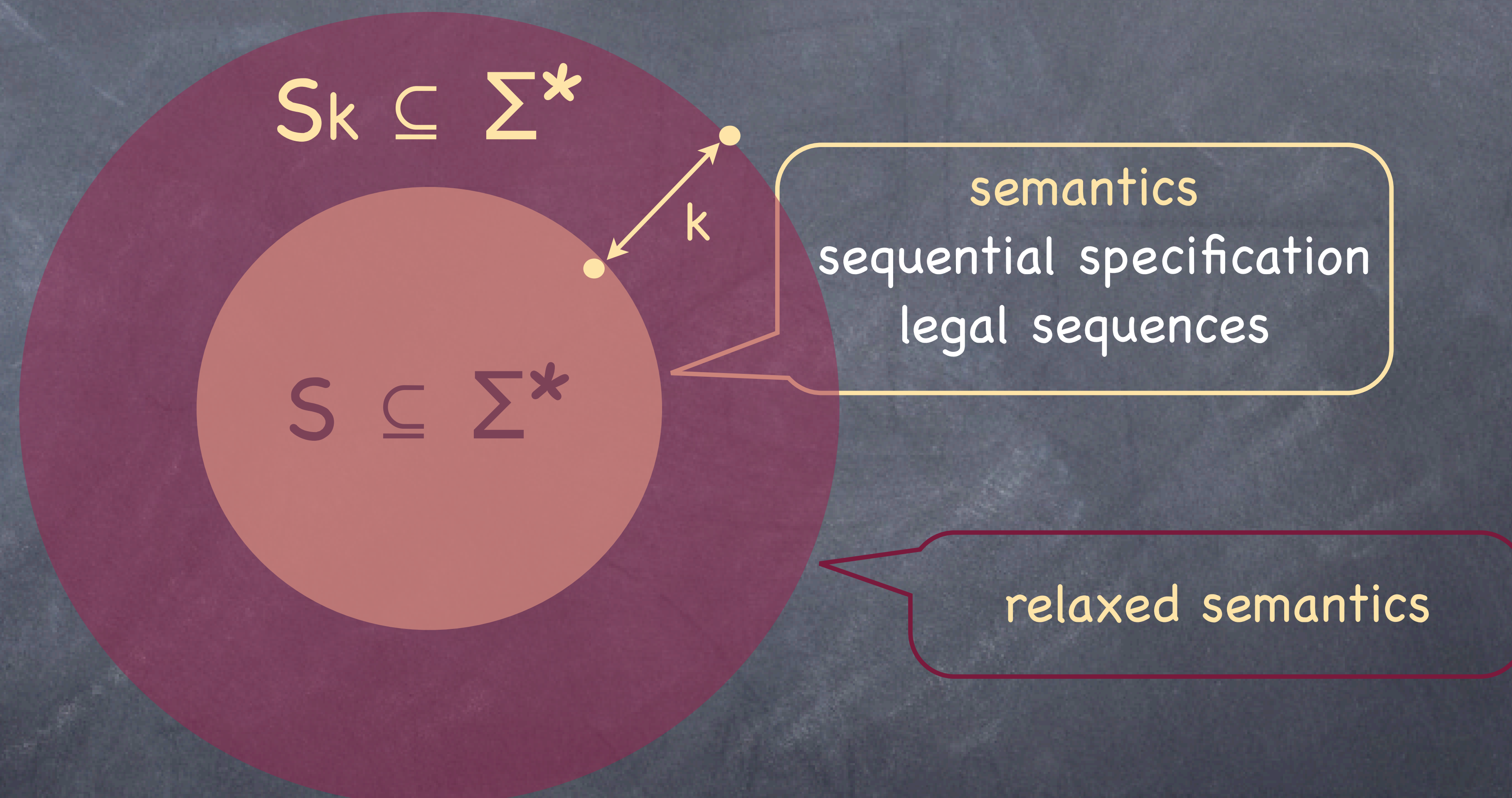
Σ - methods with arguments

The big picture



Σ - methods with arguments

The big picture



Σ - methods with arguments

distance?

Challenge

There are natural concrete relaxations...

Stack

Each **pop** pops one of the (k+1)-youngest elements

Each **push** pushes .....

k-out-of-order relaxation

Challenge

There are natural concrete relaxations...

Stack

Each **pop** pops one of the (k+1)-youngest elements

Each **push** pushes .....

k-out-of-order relaxation

makes sense also for queues, priority queues, ...

How is it reflected by a distance between sequences?

one distance for all?

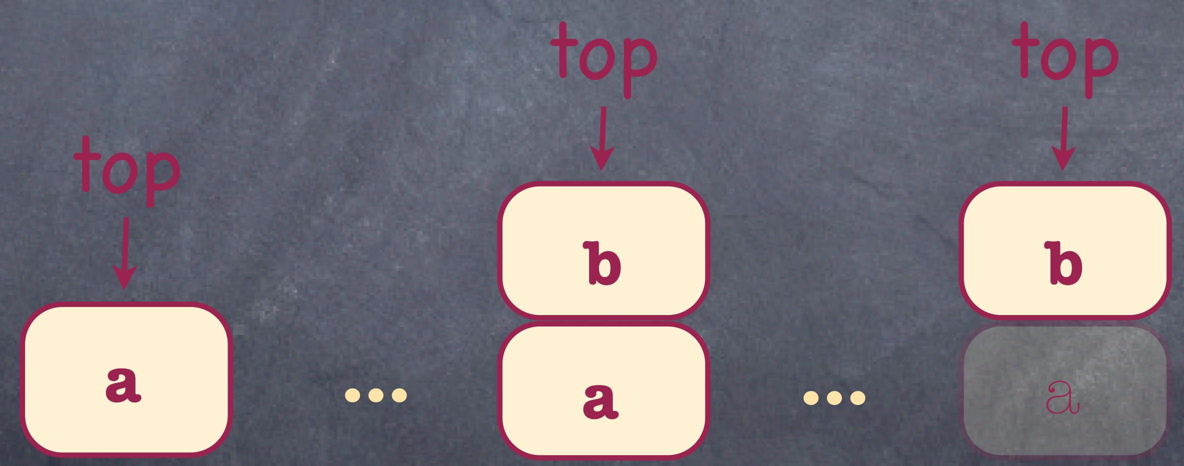
Syntactic distances do not help

push(a)[push(i)pop(i)]^{n}push(b)[push(j)pop(j)]^{m}pop(a)

Syntactic distances do not help

$\mathtt{push(a)[push(i)pop(i)]^{n}push(b)[push(j)pop(j)]^{m}pop(a)}$

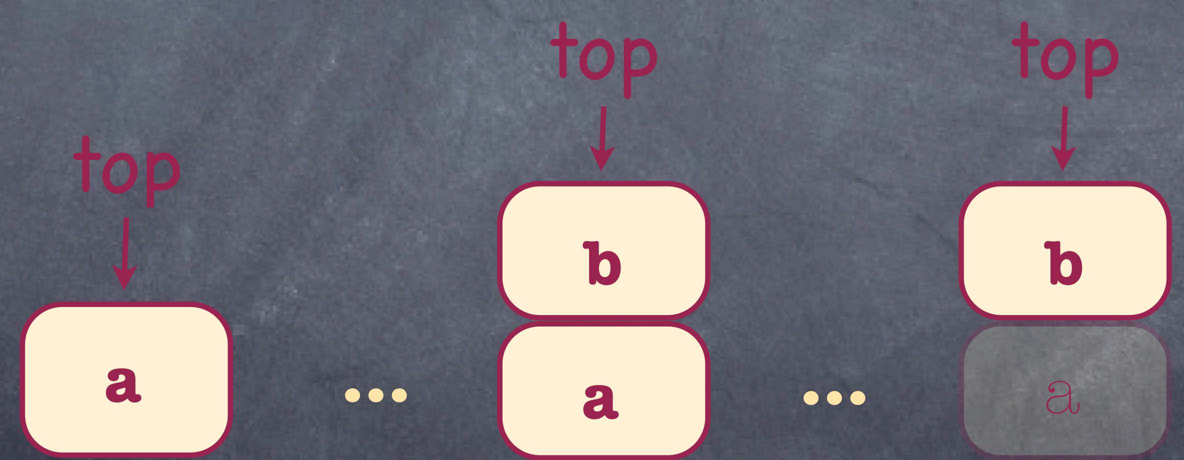
is a 1-out-of-order stack sequence



Syntactic distances do not help

$\texttt{push(a)[push(i)pop(i)]}^{n}\texttt{push(b)[push(j)pop(j)]}^{m}\texttt{pop(a)}$

is a 1-out-of-order stack sequence

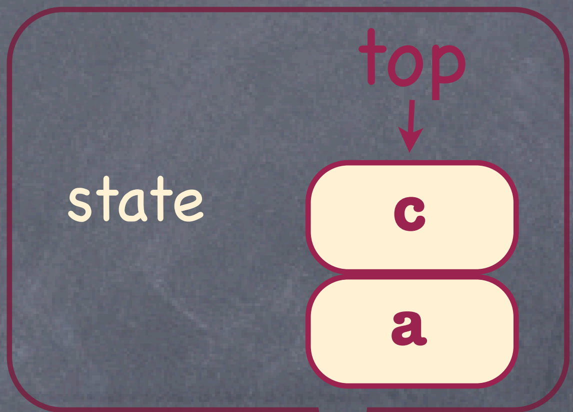


its permutation distance is min(n,m)

Semantic distances need a notion of state

- States are equivalence classes of sequences in S
- Two sequences in S are equivalent if they have an indistinguishable future

Semantic distances need a notion of state



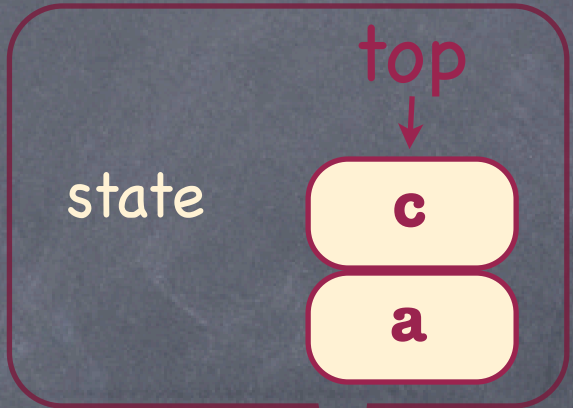
- States are equivalence classes of sequences in S

example: for stack

push(a)push(b)pop(b)push(c) ≡ push(a)push(c)

- Two sequences in S are equivalent if they have an indistinguishable future

Semantic distances need a notion of state



- States are equivalence classes of sequences in S

example: for stack

$$\mathbf{push(a)push(b)pop(b)push(c)} \equiv \mathbf{push(a)push(c)}$$

- Two sequences in S are equivalent if they have an indistinguishable future

$$\mathbf{x} \equiv \mathbf{y} \quad \Leftrightarrow \quad \forall \mathbf{u} \in \Sigma^*.\ (\mathbf{xu} \in \mathbf{S} \Leftrightarrow \mathbf{yu} \in \mathbf{S})$$

Semantics goes operational

- $S \subseteq \Sigma^*$ is the sequential specification

states

labels

initial state

- $LTS(S) = (S/_\equiv, \Sigma, \rightarrow, [\varepsilon]_\equiv)$ with

transition relation

$$[s]_\equiv \xrightarrow{m} [sm]_\equiv \quad \Leftrightarrow \quad sm \in S$$

Semantics goes operational

- $S \subseteq \Sigma^*$ is the sequential specification

states

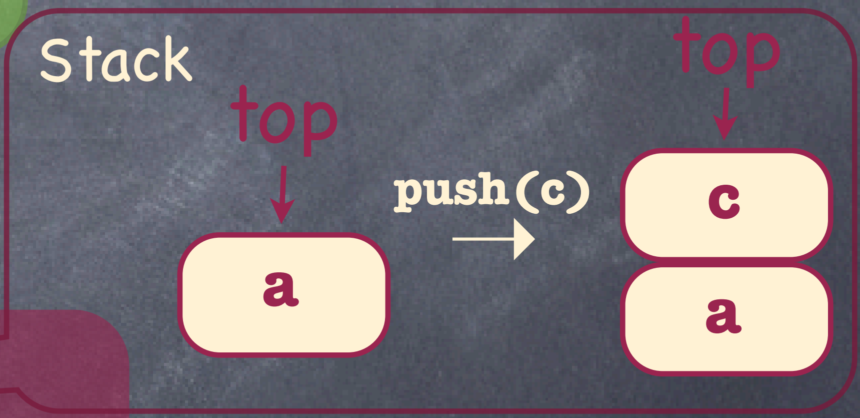
labels

initial state

- $LTS(S) = (S/_\equiv, \Sigma, \rightarrow, [\varepsilon]_\equiv)$ with

transition relation

$$[s]_\equiv \xrightarrow{m} [sm]_\equiv \quad \Leftrightarrow \quad sm \in S$$



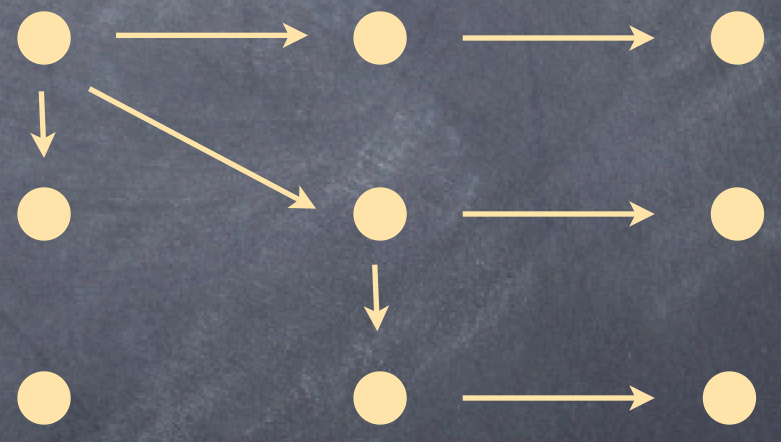
The framework

- Start from LTS(S)
- Add transitions with transition costs
- Fix a path cost function

The framework

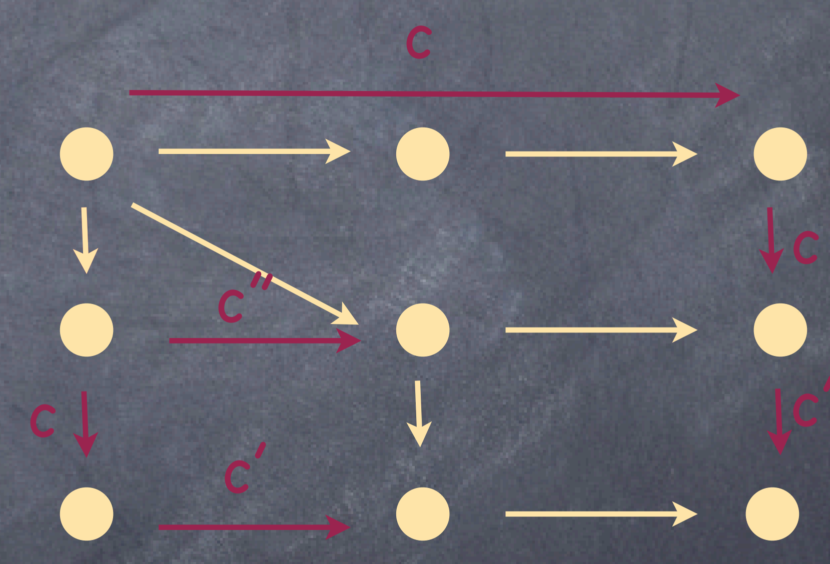
- Start from LTS(S)
- Add transitions with transition costs
- Fix a path cost function

Σ - singleton



The framework

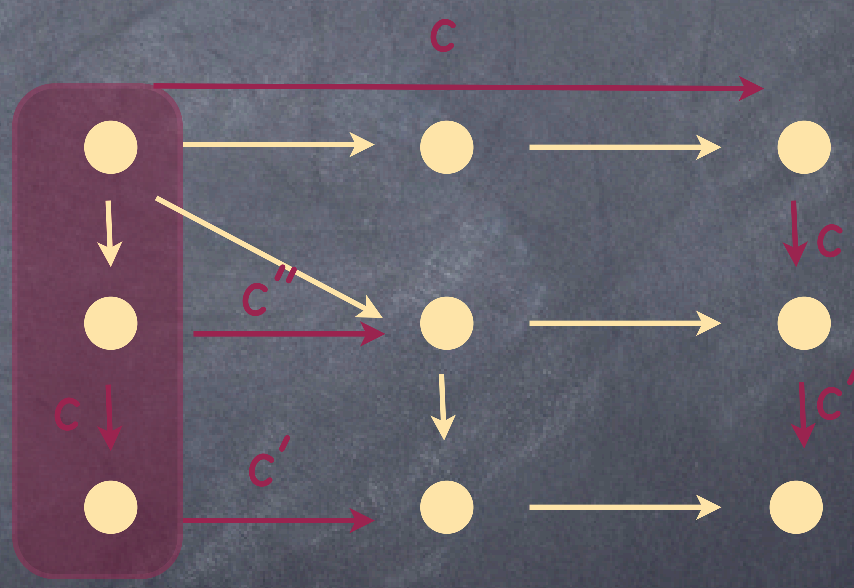
- Start from LTS(S)
- Add transitions with transition costs



- Fix a path cost function

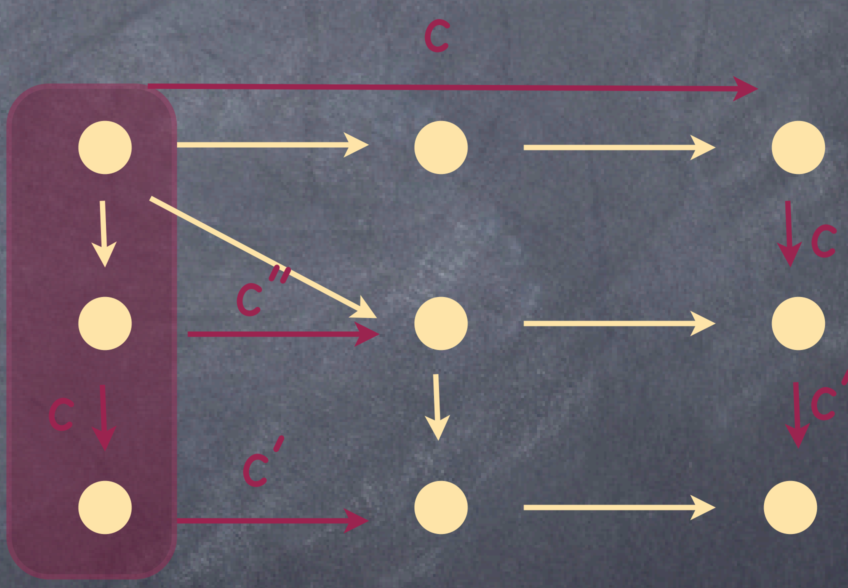
The framework

- Start from LTS(S)
- Add transitions with transition costs
- Fix a path cost function



The framework

- Start from LTS(S)
- Add transitions with transition costs



- Fix a path cost function

distance - minimal cost on all paths labelled by the sequence

Generic out-of-order

segment_cost($q \xrightarrow{m} q'$) $= |\mathbf{v}|$

transition cost

where $\mathbf{v}$ is a sequence of minimal length s.t.

(1) removing $\mathbf{v}$ enables a transition

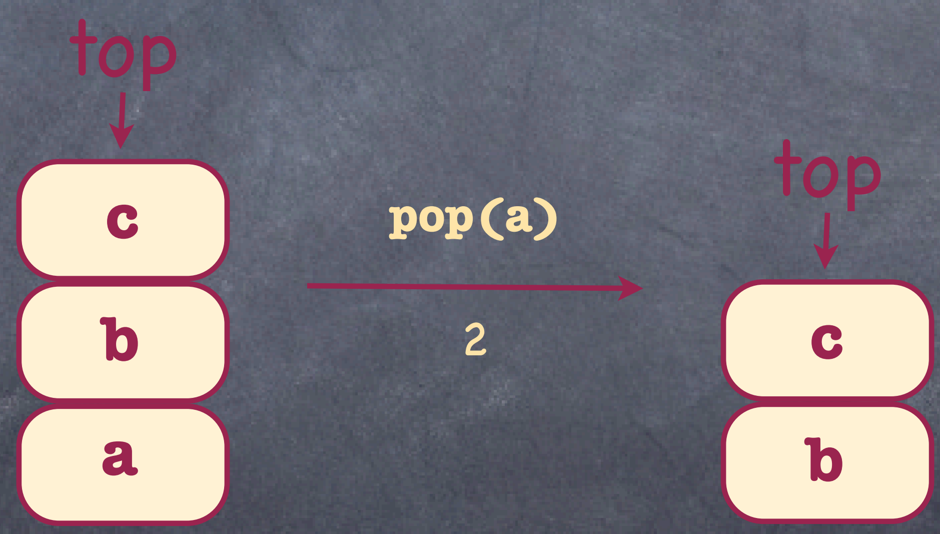
(2) inserting $\mathbf{v}$ enables a transition

goes with different path costs

Out-of-order stack

Sequence of **push**'s with no matching **pop**

- Canonical representative of a state
- Add incorrect transitions with segment-costs



- Possible path cost functions max, sum,...

also more advanced

Out-of-order queue

Sequence of **enq**'s with no matching **deq**

- Canonical representative of a state
- Add incorrect transitions with segment-costs

- Possible path cost functions max, sum,...

also more advanced

How about implementations? Performance?

Lessons learned

The way from sequential specification to concurrent implementation is hard

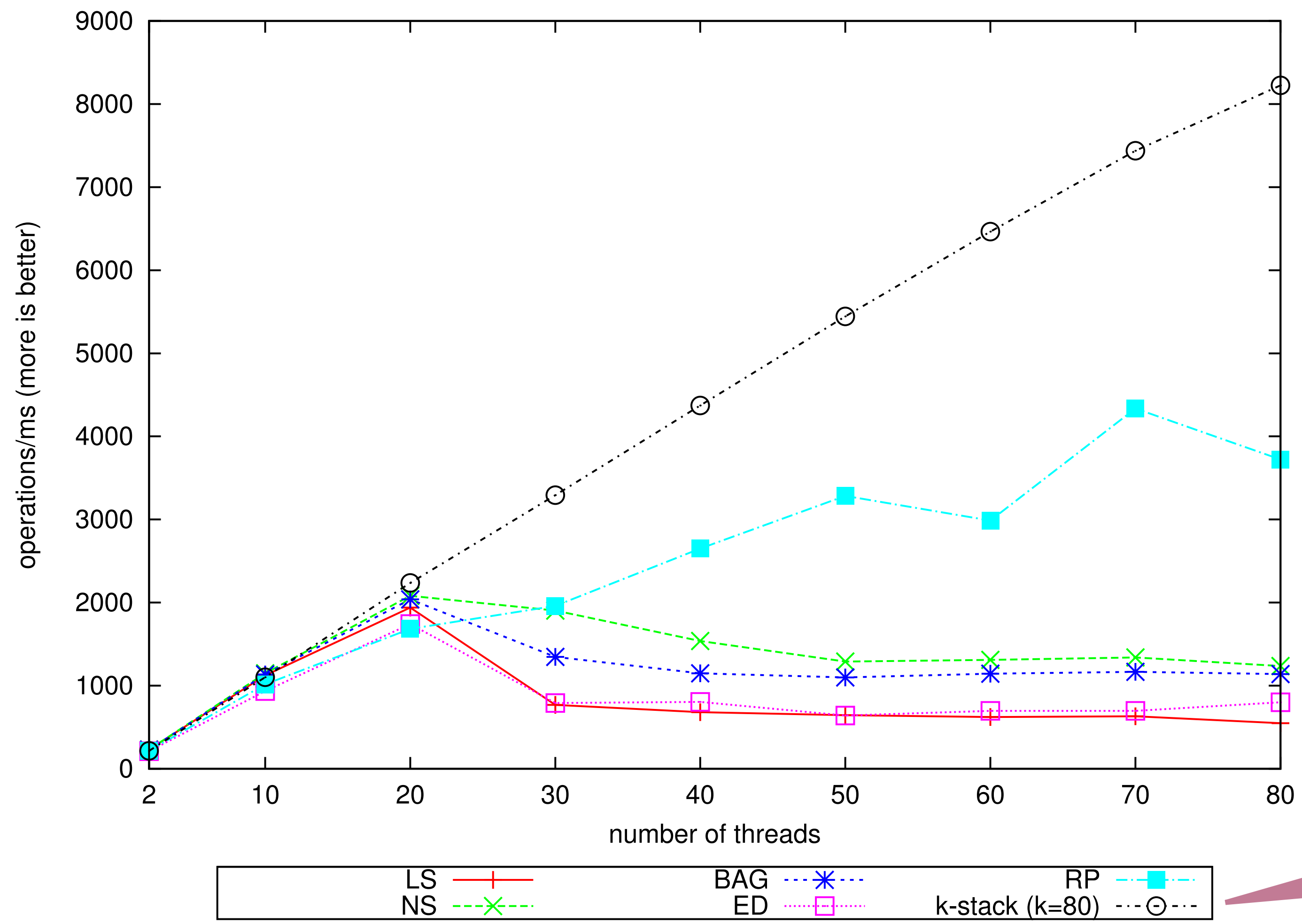
Being relaxed not necessarily means better performance

Well-performing implementations of relaxed specifications do exist!

Stack

Scalability comparison

"80"-core machine

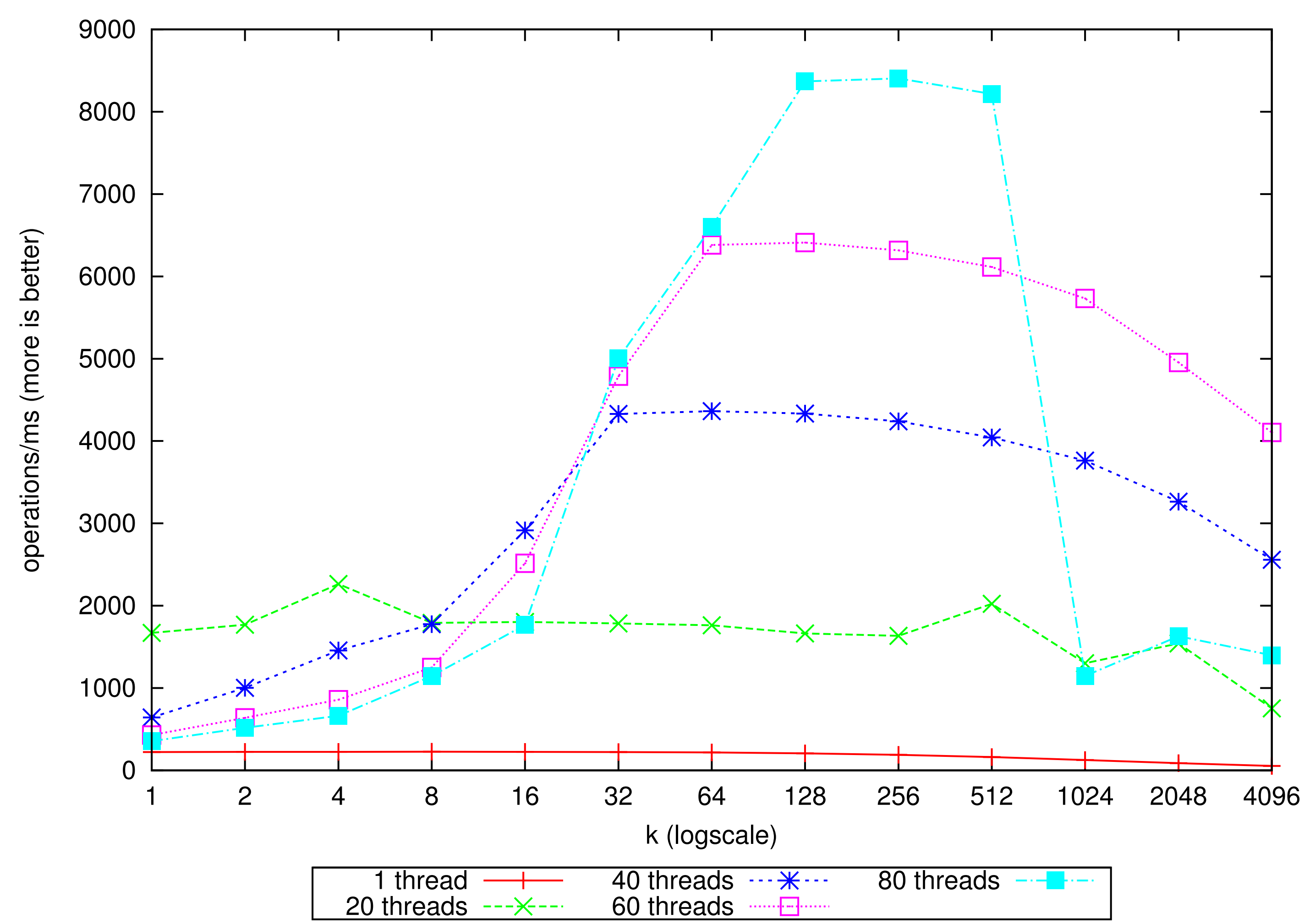


lock-free segment stack

k-Stack

The more relaxed, the better

lock-free segment stack



Conclusions

all kinds of

Contributions

Framework for quantitative relaxations
generic relaxations, concrete examples,
efficient implementations exist

THANK YOU

Difficult open problem

How to get from theory to practice?

For the future

- Study applicability
- Learn from efficient implementations

For the future

- Study applicability

which applications tolerate relaxation ?

maybe there is nothing to tolerate!

- Learn from efficient implementations

For the future

- Study applicability

which applications tolerate relaxation ?

maybe there is nothing to tolerate!

- Learn from efficient implementations

towards synthesis

lock-free universal construction ?

For the future

- Study applicability

which applications tolerate relaxation ?

maybe there is nothing to tolerate!

- Learn from efficient implementations

towards synthesis

lock-free universal construction ?

THANK YOU