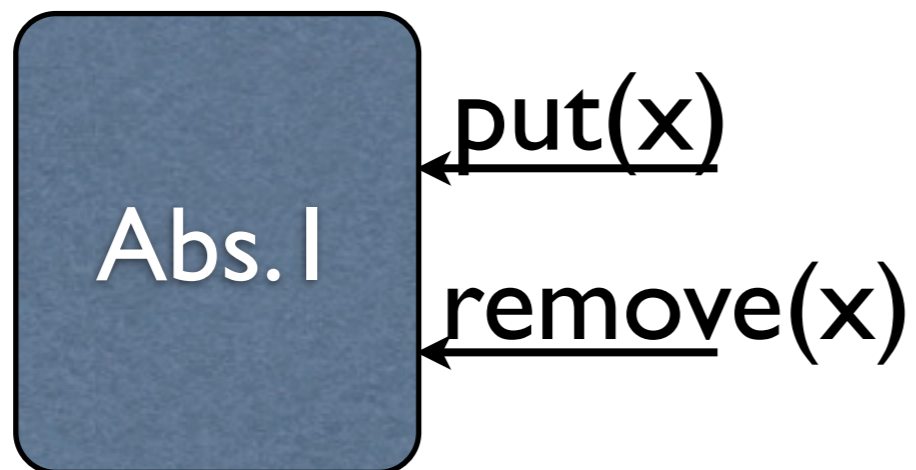


A TM Extensible Package

Vincent Gramoli
(joint work with Rachid Guerraoui and Mihai Letia)

Motivation

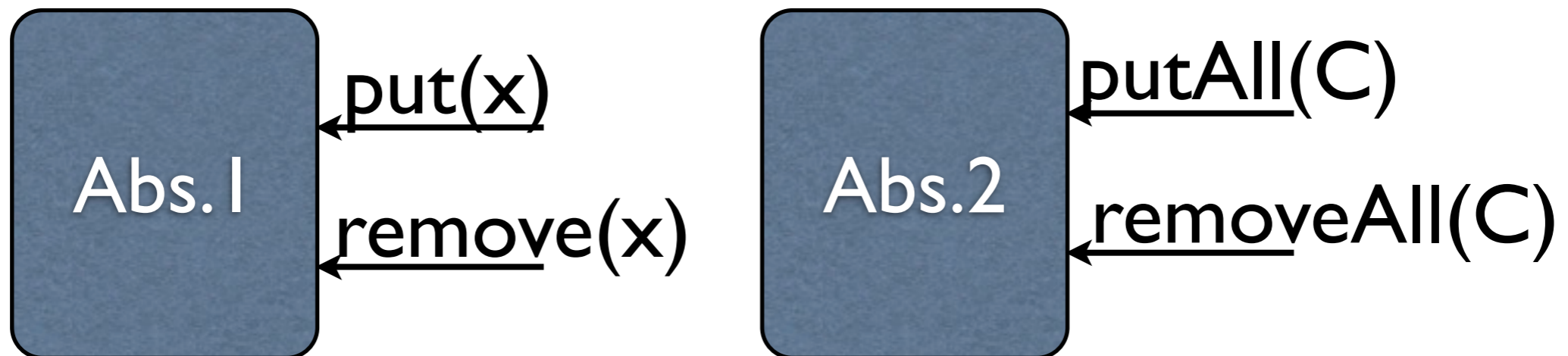
Extensibility



Given Abstraction I
coded by Alice

Motivation

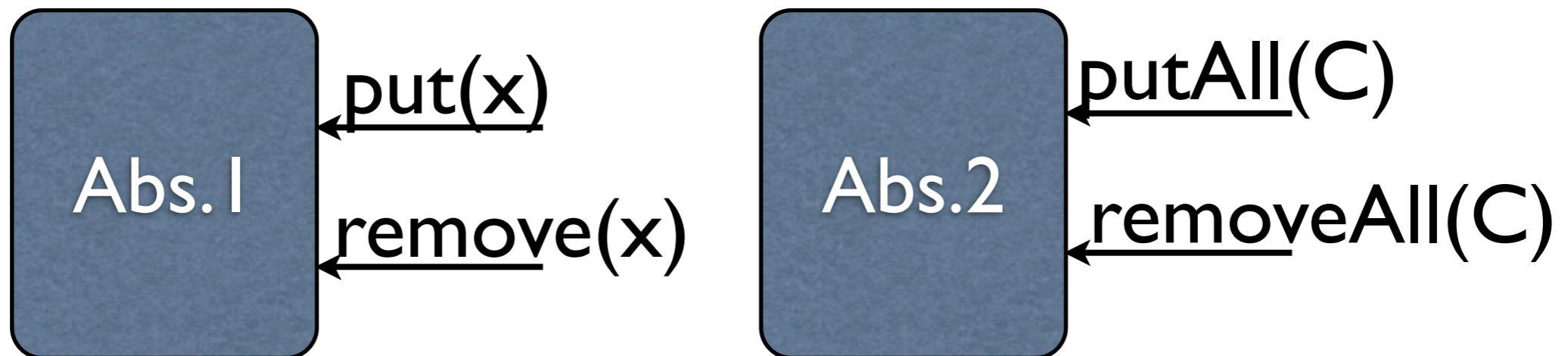
Extensibility



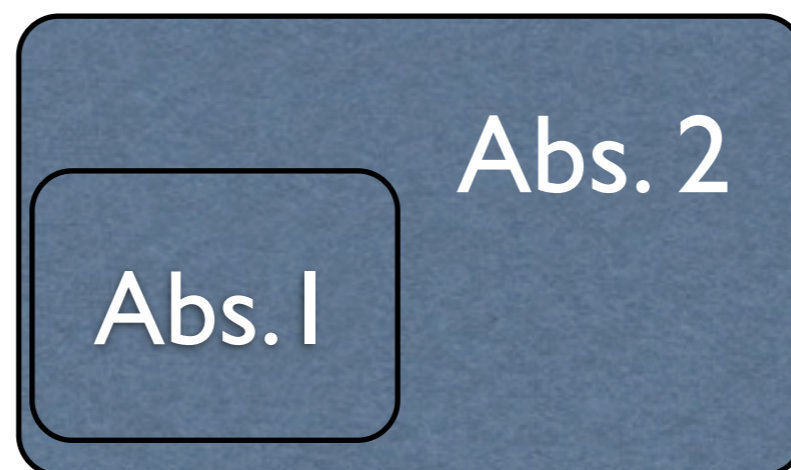
How can Bob code
Abstraction 2?

Motivation

Extensibility



Encapsulation



```
removeAll(C) {  
  for (x in C)  
    remove(x)  
}
```

Problem

Sequential programming is easy

One can reuse an existing library (API)

e.g., [Java](#)

Problem

Sequential programming is easy

One can reuse an existing library (API)
e.g., [Java](#)

Concurrent programming is hard

There is no extensible concurrent type

Let's take a look at [java.util.concurrent](#)

Java.util.concurrent

Invaluable low-level atomic tool box

j.u.c.{atomic, locks}.*

Java.util.concurrent

Invaluable low-level atomic tool box

j.u.c.{atomic, locks}.*

Non-extensible types

j.u.c.ConcurrentSkipListSet

j.u.c.ConcurrentSkipListMap

j.u.c.ConcurrentLinkedQueue

...

ConcurrentSkipList

Known issues (cf. the doc of JDK6)

- keySet() and values() are “*weakly consistent*”

ConcurrentSkipList

Known issues (cf. the doc of JDK6)

- keySet() and values() are “*weakly consistent*”
- putAll(), retainsAll(), containsAll() are *not atomic*

ConcurrentSkipList

Known issues (cf. the doc of JDK6)

- keySet() and values() are “*weakly consistent*”
- putAll(), retainsAll(), containsAll() are *not atomic*
- size() is “*not very useful in concurrent applications*”

ConcurrentLinkedQueue

Unknown issue

size() is inconsistent

Example

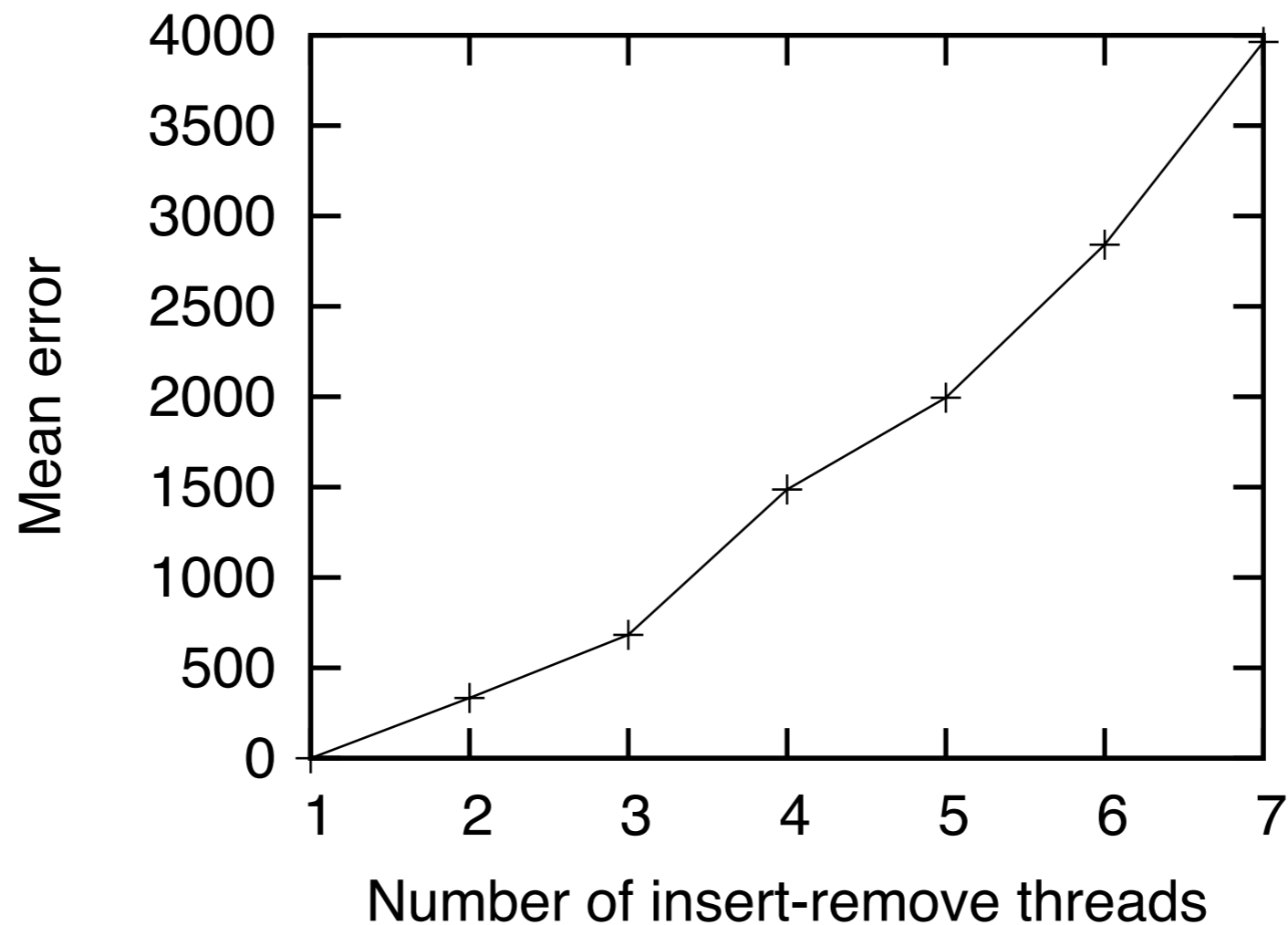
- i threads executes put(x), remove(x) j times
- another thread computes size()

worst-case result:

size() may return with an offset error $i(j-1)$
while the variation should be $\pm i$.

ConcurrentLinkedQueue

Experimental result (1024 elts, j=1000):



Reported to [Concurrency JSR 166](#) list on May 20, 2010.

Solution: Transactions

Relaxed Transactions for efficiency (basic blocks)

Comply with traditional transactions

Can be coded by expert programmers

(if they read the 19 last pages of the doc)

Regular Transactions for extension (customized blocks)

Easy-to-use for newbie programmer

(even if they only read only the 1st page)

XJDK

Reimplement from scratch the library...

Implement your **bimodal STM** (relaxed/regular)

Take a **sequential** put() (resp. remove...)

Encapsulate it into a transaction

You obtain a **concurrent put()** (resp. remove...)

Invoke put() for some set of elements

Encapsulate this loop inside a transaction

You have a **concurrent putAll()** (resp. removeAll...)

XJDK

We need a transactional type

- *efficient*: should not suffer from read-write conflicts
- *extensible*: should not require operations to be invertible
- *concurrent*: use multiversioning for snapshot-based operations.

XJDK

We need a transactional type

- *efficient*: should not suffer from read-write conflicts **No regular transactions**
- *extensible*: should not require operations to be invertible
- *concurrent*: use multiversioning for snapshot-based operations.

Solution

We need a transactional type

- *efficient*: should not suffer from read-write conflicts **No regular transactions**
- *extensible*: should not require operations to be invertible **No open nesting / tx boosting**
- *concurrent*: use multiversioning for snapshot-based operations.

Solution

We need a transactional type

- *efficient*: should not suffer from read-write conflicts **No regular transactions**
- *extensible*: should not require operations to be invertible **No open nesting / tx boosting**
- *concurrent*: use multiversioning for snapshot-based operations. **No single version?**

Summary

Ambitious project: XJDK

implements **concurrent extensible
types**

uses **multiversions elastic
transactions**

Context

Extensible Programming Language

Provide **primitive** types
Rules to extend them

Substitution Principle

A sub-type satisfies the property of its parent type.

All object-oriented languages rely on it

Liskov got the **Turing Award** in part for it