Last Lecture

Defined *concurrent objects* using linearizability or sequential consistency

Implemented *linearizable object* (Two thread FIFO Queue) in read-write memory without mutual exclusion

*Hardware* does not provide linearizable read-write memory
Fundamentals

What is the *weakest* form of communication that supports mutual exclusion?

What is the *weakest* shared object that allows shared-memory computation?
Alan Turing

Showed what is and is not computable on a sequential machine

Still the only game in town
Turing Computability

Mathematical model of computation

What is (and is not) computable

Efficiency (mostly) irrelevant

Art of Multiprocessor Programming
Turing Computability

Mathematical model of computation

What is (and is not) computable

Efficiency (mostly) irrelevant

Art of Multiprocessor Programming
Mathematical model of *concurrent* computation

What is (and is not) *concurrently* computable

Efficiency (mostly) irrelevant
Foundations of Shared Memory

To understand modern multicores we need to ask some basic questions …
Foundations of Shared Memory

To understand modern multicores we need to ask some basic questions...

What can it do?

What is the *weakest* useful form of shared memory?
Register*

Holds a (binary) value

10011

* Register == memory location, name is historical
Register

Can be read

10011
Register

Can be written

01100

10011
public interface Register<T> {
    public T read();
    public void write(T v);
}

Registers

Type of register
(usually Boolean or m-bit Integer)
Single-Reader/Single-Writer Register
Multi-Reader/Single-Writer Register

01100
Multi-Reader/Multi-Writer Register

Art of Multiprocessor Programming
Jargon Watch

SRSW

Single-reader single-writer

MRSW

Multi-reader single-writer

MRMW

Multi-reader multi-writer
Safe Register

OK as long as reads and writes don’t overlap

write(1001)

read(1001)
Safe Register

Some valid value if reads and writes do overlap

write(1001)

read(?????)

0000 1001 1111

Art of Multiprocessor Programming
Regular Register

- write(0)
- write(1)
- read(0)
- read(1)

Only one writer
Readers read:
- No overlap: old value
- Overlap: either old or new
Regular or Not?

write(0)  write(1)

read(1)  read(0)

time
Regular or Not?

Overlap: returns new value

write(0) → write(1) → read(1) → read(0)
Regular or Not?

Write(0) 

Write(1) 

Read(0) 

Overlap: returns old value
Regular or Not?

write(0) → write(1) → read(1) → read(0) → regular
Regular ≠ Linearizable

write(0) → read(1) → write(1) → read(0)

write(1) already happened

can’t justify this!
Atomic Register

write(1001)  read(1001)  read(1010)  read(1010)
write(1010)  read(1010)  read(1010)

Linearizable to sequential safe register
Atomic Register

write(1001) → write(1010) → read(1010) → read(1001) → read(1010)

Art of Multiprocessor Programming
Register Space

MRMW
MRSW
SRSW

Safe
Regular
Atomic

m-valued
Boolean
Weakest Register

Single writer

Safe Boolean register

Single reader
Weakest Register

Single writer

Single reader

Reading correct except during transition
Results

From SRSW safe Boolean register

All the other registers

+Mutual exclusion

But not everything!

Consensus hierarchy
Locking within Registers

Not interesting to rely on mutual exclusion in register constructions

We want registers to implement mutual exclusion!

It’s circular to use mutual exclusion to implement itself!
Definition

An object implementation is *wait-free* if every method call completes in a finite number of steps.

No mutual exclusion
- Thread could halt in critical section
- Build mutual exclusion from registers
From Safe SRSW Boolean to Atomic Snapshots
Road Map

- SRSW safe Boolean
- MRSW safe Boolean
- MRSW regular Boolean
- MRSW regular
- MRSW atomic
- MRMW atomic
- Atomic snapshot
Road Map

- SRSW safe Boolean
- MRSW safe Boolean
- MRSW regular Boolean
- MRSW regular
- MRSW atomic
- MRMW atomic
- Atomic snapshot
public class SafeBoolMRSWRegister implements Register<Boolean> {
    public boolean read() { ... }
    public void write(boolean x) { ... }
}
public class SafeBoolMRSWRegister implements Register<Boolean> {
    public boolean read() { ... }
    public void write(boolean x) { ... }
}
Safe Boolean MRSW from Safe Boolean SRSW

Art of Multiprocessor Programming
Safe Boolean MRSW from Safe Boolean SRSW

I will write 1!
Safe Boolean MRSW from Safe Boolean SRSW

0 or 1

Art of Multiprocessor Programming
Safe Boolean MRSW from Safe Boolean SRSW
Safe Boolean MRSW from Safe Boolean SRSW

Art of Multiprocessor Programming
Safe Boolean MRSW from Safe Boolean SRSW

Whew!

Art of Multiprocessor Programming
Safe Boolean MRSW from Safe Boolean SRSW

public class SafeBoolMRSWRegister
    implements Register<Boolean> {

    private SafeBoolSRSWRegister[] r =
        new SafeBoolSRSWRegister[N];

    public void write(boolean x) {
        for (int j = 0; j < N; j++)
            r[j].write(x);
    }

    public boolean read() {
        int i = ThreadID.get();
        return r[i].read();
    }

}
public class SafeBoolMRSWRegister implements Register<Boolean> {
    private SafeBoolSRSWRegister[] r =
        new SafeBoolSRSWRegister[N];

    public void write(boolean x) {
        for (int j = 0; j < N; j++)
            r[j].write(x);
    }

    public boolean read() {
        int i = ThreadID.get();
        return r[i].read();
    }
}
public class SafeBoolMRSWRegister implements Register<Boolean> {
    private SafeBoolSRSWRegister[] r = new SafeBoolSRSWRegister[N];
    public void write(boolean x) {
        for (int j = 0; j < N; j++)
            r[j].write(x);
    }
    public boolean read() {
        int i = ThreadID.get();
        return r[i].read();
    }
}

Write each thread’s register one at a time
Safe Boolean MRSW from Safe Boolean SRSW

```java
public class SafeBoolMRSWRegister implements Register<Boolean> {
    private SafeBoolSRSWRegister[] r = new SafeBoolSRSWRegister[N];
    public void write(boolean x) {
        for (int j = 0; j < N; j++)
            r[j].write(x);
    }
    public boolean read() {
        int i = ThreadID.get();
        return r[i].read();
    }
}
```

read method
public class SafeBoolMRSWRegister implements Register<Boolean> {
    private SafeBoolSRSWRegister[] r = new SafeBoolSRSWRegister[N];
    public void write(boolean x) {
        for (int j = 0; j < N; j++)
            r[j].write(x);
    }
    public boolean read() {
        int i = ThreadID.get();
        return r[i].read();
    }
}
Safe Multi-Valued MRSW from Safe Multi-Valued SRSW?

Yes, it works!

any value in range

Art of Multiprocessor Programming
Road Map

- SRSW safe Boolean
- MRSW safe Boolean
- MRSW regular Boolean
- MRSW regular
- MRSW atomic
- MRMW atomic
- Atomic snapshot

Questions?
Road Map

- SRSW safe Boolean
- MRSW safe Boolean
- MRSW regular Boolean
- MRSW regular
- MRSW atomic
- MRMW atomic
- Atomic snapshot

Art of Multiprocessor Programming
Regular Boolean MRSW from Safe Boolean MRSW
Regular Boolean MRSW from Safe Boolean MRSW

Uh, oh!

Art of Multiprocessor Programming
Regular Boolean MRSW from Safe Boolean MRSW

```java
public class RegBoolMRSWRegister implements Register<Boolean> {
    private boolean old;
    private SafeBoolMRSWRegister value;
    public void write(boolean x) {
        if (old != x) {
            value.write(x);
            old = x;
        }
    }
    public boolean read() {
        return value.read();
    }
}
```
Regular Boolean MRSW from Safe Boolean MRSW

public class RegBoolMRSWRegister implements Register<Boolean> {
    private boolean old;
    private SafeBoolMRSWRegister value;
    public void write(boolean x) {
        if (old != x) {
            value.write(x);
            old = x;
        }
    }
    public boolean read() {
        return value.read();
    }
}
public class RegBoolMRSWRegister implements Register<Boolean> {
    private boolean old;
    private SafeBoolMRSWRegister value;
    public void write(boolean x) {
        if (old != x) {
            value.write(x);
            old = x;
        }
    }
    public boolean read() {
        return value.read();
    }
}
Regular Boolean MRSW from Safe Boolean MRSW

public class RegBoolMRSWRegister
    implements Register<Boolean> {
    private boolean old;
    private SafeBoolMRSWRegister value;
    public void write(boolean x) {
        if (old != x) {
            value.write(x);
            old = x;
        }
    }
    public boolean read() {
        return value.read();
    }
}
Regular Boolean MRSW from Safe Boolean MRSW

public class RegBoolMRSWRegister implements Register<Boolean> {
    private boolean old;
    private SafeBoolMRSWRegister value;
    public void write(boolean x) {
        if (old != x) {
            value.write(x);
            old = x;
        }
    }
    public boolean read() {
        return value.read();
    }
}
Regular Boolean MRSW from Safe Boolean MRSW

public class RegBoolMRSWRegister implements Register<Boolean> {
    private boolean old;
    private SafeBoolMRSWRegister value;
    public void write(boolean x) {
        if (old != x) {
            value.write(x);
            old = x;
        }
    }
    public boolean read() {
        return value.read();
    }
}

Overlap? What overlap? Either Boolean value works

Art of Multiprocessor Programming
Regular Multi-Valued MRSW from Safe Multi-Valued MRSW?

Safe register can return *any* value in range when value changes.

Regular register can return only old or new when value changes.

*Does not work!*

Art of Multiprocessor Programming
Road Map

- SRSW safe Boolean
- MRSW safe Boolean
- MRSW regular Boolean
  - MRSW regular
  - MRSW atomic
  - MRMW atomic
- Atomic snapshot

Questions?

Art of Multiprocessor Programming
Road Map

- SRSW safe Boolean
- MRSW safe Boolean
- MRSW regular Boolean
- MRSW regular
- MRSW atomic
- MRMW atomic
- Atomic snapshot
Representing $m$ Values

Unary representation: bit[$i$] means value $i$

Initially

0 1 2 3 4 5 6 7
Writing $m$-Valued Register

Write 5

1 0 0 0 0
0 1 2 3 4 5 6 7
Writing $m$-Valued Register

Initially 0

Write 5

0 0 0 0 1

0 1 2 3 4 5 6 7
Writing \( m \)-Valued Register

Write 5

0 1 2 3 4 5 6 7
public class RegMRSWRegister implements Register{
    RegBoolMRSWRegister[M] bit;
    public void write(int x) {
        this.bit[x].write(true);
        for (int i=x-1; i>=0; i--)
            this.bit[i].write(false);
    }
    public int read() {
        for (int i=0; i < M; i++)
            if (this.bit[i].read())
                return i;
    }
}
public class RegMRSWRegister implements Register{

    RegBoolMRSWRegister[M] bit;
    public void write(int x) {
        this.bit[x].write(true);
        for (int i=x-1; i>=0; i--)
            this.bit[i].write(false);
    }

    public int read() {
        for (int i=0; i < M; i++)
            if (this.bit[i].read())
                return i;
    }
}

Unary representation: bit[i] means value i
public class RegMRSWRegister implements Register{
    RegBoolMRSWRegister[M] bit;
    public void write(int x) {
        this.bit[x].write(true);
        for (int i=x-1; i>=0; i--)
            this.bit[i].write(false),
    }

    public int read() {
        for (int i=0; i < M; i++)
            if (this.bit[i].read())
                return i;
    }
}

Art of Multiprocessor Programming
public class RegMRSWRegister implements Register{
    RegBoolMRSWRegister[M] bit;
    public void write(int x) {
        this.bit[x].write(true);
        for (int i=x-1; i>=0; i--)
            this.bit[i].write(false);
    }

    public int read() {
        for (int i=0; i < M; i++)
            if (this.bit[i].read())
                return i;
    }
}

Clear bits from higher to lower

MRSW Regular $m$-valued from MRSW Regular Boolean

Art of Multiprocessor Programming
MRSW Regular $m$-valued from MRSW Regular Boolean

```java
public class RegMRSWRegister implements Register{
    RegBoolMRSWRegister[M] bit;

    public void write(int x) {
        this.bit[x].write(true);
        for (int i=x-1; i>=0; i--)
            this.bit[i].write(false);
    }

    public int read() {
        for (int i=0; i < M; i++)
            if (this.bit[i].read())
                return i;
    }
}
```

Scan from lower to higher & return first bit set
Road Map

- SRSW safe Boolean
- MRSW safe Boolean
- MRSW regular Boolean
- MRSW regular
- MRSW atomic
- MRMWW atomic
- Atomic snapshot

Questions?
Road Map

- SRSW safe Boolean
- MRSW safe Boolean
- MRSW regular Boolean
- MRSW regular
- MRSW atomic
- MRMW atomic
- Atomic snapshot
Road Map (Slight Detour)

- SRSW safe Boolean
- MRSW safe Boolean
- MRSW regular Boolean
- MRSW regular
- MRSW atomic
- MRMW atomic
- Atomic snapshot

Art of Multiprocessor Programming
SRSW Atomic From SRSW
Regular

Regular writer

Concurrent Reading

5678

1234

Instead of 5678…

Regular reader

When is this a problem?

Art of Multiprocessor Programming
SRSW Atomic From SRSW Regular

Regular writer

Regular reader

Initially 1234

write(5678)

Same as Atomic

Art of Multiprocessor Programming
SRSW Atomic From SRSW Regular

Regular writer

Initially 1234

write(5678) read(1234)

Same as Atomic

5678...

Regular reader

Art of Multiprocessor Programming
SRSW Atomic From SRSW Regular

Initially
1234

write(5678)
Reg read(5678)
read(1234)

Write 5678 happened

Art of Multiprocessor Programming

Not Atomic!
Timestamped Values

Writer writes value and stamp together

Reader saves last value & stamp read returns new value only if stamp is higher
SRSW Atomic From SRSW Regular

Initially 1:45 1234

write(2:00 5678)
read(2:00 5678)
read(1:45 1234)

Same as Atomic
Atomic Single-Reader to Atomic Multi-Reader

<table>
<thead>
<tr>
<th>stamp</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1:45</td>
<td>1234</td>
</tr>
<tr>
<td>1:45</td>
<td>1234</td>
</tr>
<tr>
<td>1:45</td>
<td>1234</td>
</tr>
<tr>
<td>1:45</td>
<td>1234</td>
</tr>
</tbody>
</table>

One per reader
Another Scenario

Writer starts write...

<table>
<thead>
<tr>
<th>stamp</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>2:00</td>
<td>5678</td>
</tr>
<tr>
<td>1:45</td>
<td>1234</td>
</tr>
<tr>
<td>1:45</td>
<td>1234</td>
</tr>
<tr>
<td>1:45</td>
<td>1234</td>
</tr>
</tbody>
</table>
Yellow was completely after Blue but read earlier value...not linearizable!
### Multi-Reader Redux

One per thread

<table>
<thead>
<tr>
<th></th>
<th>1:45</th>
<th>1234</th>
<th></th>
<th>1:45</th>
<th>1234</th>
<th></th>
<th>1:45</th>
<th>1234</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1:45</td>
<td>1234</td>
<td>2</td>
<td>1:45</td>
<td>1234</td>
<td>3</td>
<td>1:45</td>
<td>1234</td>
</tr>
<tr>
<td>2</td>
<td>1:45</td>
<td>1234</td>
<td>3</td>
<td>1:45</td>
<td>1234</td>
<td>3</td>
<td>1:45</td>
<td>1234</td>
</tr>
<tr>
<td>3</td>
<td>1:45</td>
<td>1234</td>
<td></td>
<td>1:45</td>
<td>1234</td>
<td></td>
<td>1:45</td>
<td>1234</td>
</tr>
<tr>
<td>2:00</td>
<td>5678</td>
<td>1:45</td>
<td>1234</td>
<td>1:45</td>
<td>1234</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>------</td>
<td>------</td>
<td>------</td>
<td>------</td>
<td>------</td>
<td>------</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2:00</td>
<td>5678</td>
<td>1:45</td>
<td>1234</td>
<td>1:45</td>
<td>1234</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2:00</td>
<td>5678</td>
<td>1:45</td>
<td>1234</td>
<td>1:45</td>
<td>1234</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Writer writes column...

Reader reads row

Art of Multiprocessor Programming
Multi Reader Redux

reader writes column to notify others of what it read

Yellow will read new value in column written earlier by Blue reader

Art of Multiprocessor Programming
Can Yellow Miss Blue’s Update? … Only if Readers Overlap…

1:45
1234

write(2:00 5678)
read(2:00 5678)
read(1:45 1234)

In which case it’s OK to read 1234
Bad Case Only When Readers Don’t Overlap

In which case Blue will complete writing 2:00 5678 to its column.
Road Map

- SRSW safe Boolean
- MRSW safe Boolean
- MRSW regular Boolean
- MRSW regular
- MRSW atomic
- MRMW atomic
- Atomic snapshot

Art of Multiprocessor Programming
Multi-Writer Atomic From Multi-Reader Atomic

Each writer reads all then writes Max+1 to its register

Readers read all and take max (Lexicographic like Bakery)

Max is 2:15, return XYZW

Art of Multiprocessor Programming
Atomic Execution
Means it is Linearizable

<table>
<thead>
<tr>
<th>Time</th>
<th>Actions</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>write(1)</td>
</tr>
<tr>
<td></td>
<td>read(1)</td>
</tr>
<tr>
<td></td>
<td>write(2)</td>
</tr>
<tr>
<td></td>
<td>write(3)</td>
</tr>
<tr>
<td></td>
<td>read(3)</td>
</tr>
<tr>
<td></td>
<td>write(4)</td>
</tr>
<tr>
<td></td>
<td>read(4)</td>
</tr>
</tbody>
</table>

Art of Multiprocessor Programming
Linearization Points

- write(1)
- Read(max = 2)
- write(4)
- write(2)
- Read(max = 3)
- write(3)
- Read(max = 4)
- Read (max = 1)
- write(2)

Art of Multiprocessor Programming
Linearization Points

Look at Writes First

write(1) → write(2) → write(2) → write(3) → write(2) → write(4)

time
Linearization Points

Order writes by timestamps

write(1)
write(2)
write(2)
write(3)
write(2)
write(4)

time
Linearization Points

Order reads by max stamp read

- write(1)
- write(2)
- Read(max = 2)
- write(3)
- Read(max = 1)
- Read(max = 2)
- write(4)
- Read(max = 3)
- Read(max = 4)

Art of Multiprocessor Programming
Linearization Points

The linearization point depends on the execution (not a line in the code)!

- write(1)
- write(2)
- write(3)
- write(4)
- Read(max = 2)
- Read(max = 3)
- Read(max = 4)

Art of Multiprocessor Programming
Road Map

- SRSW safe Boolean
- MRSW safe Boolean
- MRSW regular Boolean
- MRSW regular
- MRSW atomic
- MRMW atomic
- Atomic snapshot

Questions?
Road Map

- SRSW safe Boolean
- MRSW safe Boolean
- MRSW regular Boolean
- MRSW regular
- MRSW atomic
- MRMW atomic
- Atomic snapshot

Next
Atomic Snapshot

update

scan
Array of SWMR atomic registers

Take instantaneous snapshot of all

Generalizes to MRMW registers ...
public interface Snapshot {
    public int update(int v);
    public int[][] scan();
}
**Snapshot Interface**

Thread $i$ writes $v$ to its register

```java
public interface Snapshot {
    public int update(int v);
    public int[] scan();
}
```
Snapshot Interface

Instantaneous snapshot of all threads' registers

```java
public interface Snapshot {
    public int update(int v);
    public int[] scan();
}
```
Atomic Snapshot

Collect

Read values one at a time

Problem:

Incomparable concurrent collects
Clean Collects

Clean Collect

Collect during which nothing changed

Can we detect if it happens?

Can we force it to happen?
Simple Snapshot

Put increasing labels on each entry

Scanner collects twice

Agree? Done!

Disagree? Try over …

Problem: view might not be a snapshot!
Claim: We Must Use Labels

But scanner sees $x$ and $z$ together!

Scanner

Updater

$w$

Updater

$x$ and $z$ are never in memory together
Scanner reads x and z with different labels and recognizes collect not clean.

Scanner reads x and z with different labels and recognizes collect not clean.

Scanner reads x and z with different labels and recognizes collect not clean.

Scanner reads x and z with different labels and recognizes collect not clean.

Scanner reads x and z with different labels and recognizes collect not clean.

Scanner reads x and z with different labels and recognizes collect not clean.

Scanner reads x and z with different labels and recognizes collect not clean.

Scanner reads x and z with different labels and recognizes collect not clean.

Scanner reads x and z with different labels and recognizes collect not clean.

Scanner reads x and z with different labels and recognizes collect not clean.

Scanner reads x and z with different labels and recognizes collect not clean.

Scanner reads x and z with different labels and recognizes collect not clean.

Scanner reads x and z with different labels and recognizes collect not clean.

Scanner reads x and z with different labels and recognizes collect not clean.

Scanner reads x and z with different labels and recognizes collect not clean.

Scanner reads x and z with different labels and recognizes collect not clean.

Scanner reads x and z with different labels and recognizes collect not clean.

Scanner reads x and z with different labels and recognizes collect not clean.

Scanner reads x and z with different labels and recognizes collect not clean.

Scanner reads x and z with different labels and recognizes collect not clean.

Scanner reads x and z with different labels and recognizes collect not clean.

Scanner reads x and z with different labels and recognizes collect not clean.

Scanner reads x and z with different labels and recognizes collect not clean.

Scanner reads x and z with different labels and recognizes collect not clean.

Scanner reads x and z with different labels and recognizes collect not clean.

Scanner reads x and z with different labels and recognizes collect not clean.

Scanner reads x and z with different labels and recognizes collect not clean.

Scanner reads x and z with different labels and recognizes collect not clean.

Scanner reads x and z with different labels and recognizes collect not clean.

Scanner reads x and z with different labels and recognizes collect not clean.

Scanner reads x and z with different labels and recognizes collect not clean.

Scanner reads x and z with different labels and recognizes collect not clean.

Scanner reads x and z with different labels and recognizes collect not clean.

Scanner reads x and z with different labels and recognizes collect not clean.

Scanner reads x and z with different labels and recognizes collect not clean.

Scanner reads x and z with different labels and recognizes collect not clean.

Scanner reads x and z with different labels and recognizes collect not clean.

Scanner reads x and z with different labels and recognizes collect not clean.

Scanner reads x and z with different labels and recognizes collect not clean.

Scanner reads x and z with different labels and recognizes collect not clean.

Scanner reads x and z with different labels and recognizes collect not clean.

Scanner reads x and z with different labels and recognizes collect not clean.

Scanner reads x and z with different labels and recognizes collect not clean.

Scanner reads x and z with different labels and recognizes collect not clean.

Scanner reads x and z with different labels and recognizes collect not clean.

Scanner reads x and z with different labels and recognizes collect not clean.

Scanner reads x and z with different labels and recognizes collect not clean.

Scanner reads x and z with different labels and recognizes collect not clean.

Scanner reads x and z with different labels and recognizes collect not clean.

Scanner reads x and z with different labels and recognizes collect not clean.

Scanner reads x and z with different labels and recognizes collect not clean.

Scanner reads x and z with different labels and recognizes collect not clean.

Scanner reads x and z with different labels and recognizes collect not clean.

Scanner reads x and z with different labels and recognizes collect not clean.

Scanner reads x and z with different labels and recognizes collect not clean.

Scanner reads x and z with different labels and recognizes collect not clean.

Scanner reads x and z with different labels and recognizes collect not clean.

Scanner reads x and z with different labels and recognizes collect not clean.

Scanner reads x and z with different labels and recognizes collect not clean.

Scanner reads x and z with different labels and recognizes collect not clean.

Scanner reads x and z with different labels and recognizes collect not clean.

Scanner reads x and z with different labels and recognizes collect not clean.

Scanner reads x and z with different labels and recognizes collect not clean.

Scanner reads x and z with different labels and recognizes collect not clean.

Scanner reads x and z with different labels and recognizes collect not clean.

Scanner reads x and z with different labels and recognizes collect not clean.

Scanner reads x and z with different labels and recognizes collect not clean.

Scanner reads x and z with different labels and recognizes collect not clean.

Scanner reads x and z with different labels and recognizes collect not clean.

Scanner reads x and z with different labels and recognizes collect not clean.

Scanner reads x and z with different labels and recognizes collect not clean.

Scanner reads x and z with different labels and recognizes collect not clean.

Scanner reads x and z with different labels and recognizes collect not clean.

Scanner reads x and z with different labels and recognizes collect not clean.

Scanner reads x and z with different labels and recognizes collect not clean.

Scanner reads x and z with different labels and recognizes collect not clean.

Scanner reads x and z with different labels and recognizes collect not clean.

Scanner reads x and z with different labels and recognizes collect not clean.

Scanner reads x and z with different labels and recognizes collect not clean.

Scanner reads x and z with different labels and recognizes collect not clean.

Scanner reads x and z with different labels and recognizes collect not clean.

Scanner reads x and z with different labels and recognizes collect not clean.

Scanner reads x and z with different labels and recognizes collect not clean.

Scanner reads x and z with different labels and recognizes collect not clean.

Scanner reads x and z with different labels and recognizes collect not clean.

Scanner reads x and z with different labels and recognizes collect not clean.

Scanner reads x and z with different labels and recognizes collect not clean.

Scanner reads x and z with different labels and recognizes collect not clean.

Scanner reads x and z with different labels and recognizes collect not clean.

Scanner reads x and z with different labels and recognizes collect not clean.

Scanner reads x and z with different labels and recognizes collect not clean.

Scanner reads x and z with different labels and recognizes collect not clean.

Scanner reads x and z with different labels and recognizes collect not clean.

Scanner reads x and z with different labels and recognizes collect not clean.

Scanner reads x and z with different labels and recognizes collect not clean.

Scanner reads x and z with different labels and recognizes collect not clean.

Scanner reads x and z with different labels and recognizes collect not clean.

Scanner reads x and z with different labels and recognizes collect not clean.

Scanner reads x and z with different labels and recognizes collect not clean.

Scanner reads x and z with different labels and recognizes collect not clean.

Scanner reads x and z with different labels and recognizes collect not clean.

Scanner reads x and z with different labels and recognizes collect not clean.

Scanner reads x and z with different labels and recognizes collect not clean.

Scanner reads x and z with different labels and recognizes collect not clean.

Scanner reads x and z with different labels and recognizes collect not clean.

Scanner reads x and z with different labels and recognizes collect not clean.

Scanner reads x and z with different labels and recognizes collect not clean.

Scanner reads x and z with different labels and recognizes collect not clean.

Scanner reads x and z with different labels and recognizes collect not clean.

Scanner reads x and z with different labels and recognizes collect not clean.

Scanner reads x and z with different labels and recognizes collect not clean.

Scanner reads x and z with different labels and recognizes collect not clean.

Scanner reads x and z with different labels and recognizes collect not clean.

Scanner reads x and z with different labels and recognizes collect not clean.

Scanner reads x and z with different labels and recognizes collect not clean.

Scanner reads x and z with different labels and recognizes collect not clean.

Scanner reads x and z with different labels and recognizes collect not clean.

Scanner reads x and z with different labels and recognizes collect not clean.

Scanner reads x and z with different labels and recognizes collect not clean.

Scanner reads x and z with different labels and recognizes collect not clean.

Scanner reads x and z with different labels and recognizes collect not clean.

Scanner reads x and z with different labels and recognizes collect not clean.

Scanner reads x and z with different labels and recognizes collect not clean.

Scanner reads x and z with different labels and recognizes collect not clean.

Scanner reads x and z with different labels and recognizes collect not clean.

Scanner reads x and z with different labels and recognizes collect not clean.

Scanner reads x and z with different labels and recognizes collect not clean.

Scanner reads x and z with different labels and recognizes collect not clean.
Simple Snapshot

Collect twice

Agree? Done!

Disagree? Try over …

Collect 1

Collect 2

1
22
1
7
13
18
12

= 

1
22
1
7
13
18
12

Art of Multiprocessor Programming
public class SimpleSnapshot implements Snapshot {
    private AtomicMRSWRegister[] register;

    public void update(int value) {
        int i = Thread.myIndex();
        LabeledValue oldValue = register[i].read();
        LabeledValue newValue = new LabeledValue(oldValue.label + 1, value);
        register[i].write(newValue);
    }
}
public class SimpleSnapshot implements Snapshot {
    private AtomicMRSWRegister[] register;

    public void update(int value) {
        int i = Thread.myIndex();
        LabeledValue oldValue = register[i].read();
        LabeledValue newValue = new LabeledValue(oldValue.label+1, value);
        register[i].write(newValue);
    }

    One single-writer register per thread
public class SimpleSnapshot implements Snapshot {
    private AtomicMRSWRegister[] register;

    public void update(int value) {
        int i = Thread.myIndex();
        LabeledValue oldValue = register[i].read();
        LabeledValue newValue =
            new LabeledValue(oldValue.label + 1, value);
        register[i].write(newValue);
    }
}
Simple Snapshot: Collect

private LabeledValue[] collect() {
    LabeledValue[] copy =
    new LabeledValue[n];
    for (int j = 0; j < n; j++)
        copy[j] = this.register[j].read();
    return copy;
}
private LabeledValue[] collect() {
    LabeledValue[] copy =
    new LabeledValue[n];
    for (int j = 0; j < n; j++)
        copy[j] = this.register[j].read();
    return copy;
}
public int[] scan() {
    LabeledValue[] oldCopy, newCopy;
    oldCopy = collect();
    collect: while (true) {
        newCopy = collect();
        if (!equals(oldCopy, newCopy)) {
            oldCopy = newCopy;
            continue collect;
        }
    }
    return getValues(newCopy);
}
public int[] scan() {
    LabeledValue[] oldCopy, newCopy;
    oldCopy = collect();
    collect: while (true) {
        newCopy = collect();
        if (!equals(oldCopy, newCopy)) {
            oldCopy = newCopy;
            continue collect;
        }
    }

    return getValues(newCopy);
}
Simple Snapshot

- Linearizable
- Update is wait-free
- No unbounded loops
- But Scan can starve
- If interrupted by concurrent updates
Wait-Free Snapshot

Add a scan before every update

Write resulting snapshot together with update value

If scan is continuously interrupted by updates, scan can take the update’s snapshot
Wait-free Snapshot

If A’s scan observes that B moved *twice*, then B completed an update while A’s scan was in progress.
Wait-free Snapshot

Art of Multiprocessor Programming
Wait-free Snapshot

Art of Multiprocessor Programming
Wait-free Snapshot

B’s 1st update must have written during 1st collect

So scan of B’s second update must be within interval of A’s scan

So A can steal result of B’s scan

Art of Multiprocessor Programming
Wait-free Snapshot

But no guarantee that scan of B’s 1\textsuperscript{st} update can be used… Why?

Art of Multiprocessor Programming
Once is not Enough

Why can’t A steal B’s scan?
Because another update might have interfered before the scan.
If we collect $n$ times...some thread must move twice (pigeonhole principle)
Scan is Wait-free

At most n-1 depth

So some thread must have had clean collect
public class SnapValue {
    public int label;
    public int value;
    public int[] snap;
}
public class SnapValue {
    public int label;
    public int value;
    public int[] snap;
}

Counter incremented with each snapshot
public class SnapValue {
  public int label;
  public int value;
  public int[] snap;
}

Wait-Free Snapshot Label

Actual value
public class SnapValue {
    public int label;
    public int value;
    public int[] snap;
}

most recent snapshot
Wait-Free Snapshot Label

110111110110100010110000

label

value

Last snapshot

Art of Multiprocessor Programming 132
public void update(int value) {
    int i = Thread.myIndex();
    int[] snap = this.scan();
    SnapValue oldValue = r[i].read();
    SnapValue newValue =
        new SnapValue(oldValue.label+1, value, snap);
    r[i].write(newValue);
}
public int[] scan() {
    SnapValue[] oldCopy, newCopy;
    boolean[] moved = new boolean[n];
    oldCopy = collect();
    collect: while (true) {
        newCopy = collect();
        for (int j = 0; j < n; j++) {
            if (oldCopy[j].label != newCopy[j].label) {
                ...
            }
        }
    }
    return getValues(
        })
}}

Keep track of who moved
public int[] scan() {
    SnapValue[] oldCopy, newCopy;
    boolean[] moved = new boolean[n];
    oldCopy = collect();
    collect: while (true) {
        newCopy = collect();
        for (int j = 0; j < n; j++) {
            if (oldCopy[j].label != newCopy[j].label) {
                ...
            }
        }
    }
    return getValues(newCopy);
}}

Repeated double collect
Wait-free Scan

```java
public int[] scan() {
    SnapValue[] oldCopy, newCopy;
    boolean[] moved = new boolean[n];
    oldCopy = collect();
    collect: while (true) {
        newCopy = collect();
        for (int j = 0; j < n; j++) {
            if (oldCopy[j].label != newCopy[j].label) {
                ...
            }
        }
        return getValues(newCopy);
    }
}
```

If mismatch detected...
if (oldCopy[j].label != newCopy[j].label) {
    if (moved[j]) {
        return newCopy[j].snap;
    } else {
        moved[j] = true;
        oldCopy = newCopy;
        continue collect;
    }
}}
return getValues(newCopy);
Mismatch Detected

```c
if (oldCopy[j].label != newCopy[j].label) {
    if (moved[j]) { // second move
        return newCopy[j].snap;
    } else {
        moved[j] = true;
        oldCopy = newCopy;
        continue collect;
    }
}
return getValues(newCopy);
```

Remember that thread moved
Observations

Uses unbounded counters

Can be replaced with 2 bits

Assumes SWMR registers

for labels

can be extended to MRMW
Summary

We can implement MRMW multi-valued snapshots

From SRSW binary safe registers (simple flipflops)

Impressive, so what’s next for read-write registers?
Grand Challenge

Snapshot means

Write any one array element

Read multiple array elements

What about writing multiple elements?
Challenge!

Write multiple slots, then snapshot

Writes to 0 and 1

Writes to 1 and 2
This work is licensed under a Creative Commons Attribution-ShareAlike 2.5 License.

- You are free:
  - to Share — to copy, distribute and transmit the work
  - to Remix — to adapt the work

- Under the following conditions:
  - Attribution. You must attribute the work to “The Art of Multiprocessor Programming” (but not in any way that suggests that the authors endorse you or your use of the work).
  - Share Alike. If you alter, transform, or build upon this work, you may distribute the resulting work only under the same, similar or a compatible license.

- For any reuse or distribution, you must make clear to others the license terms of this work. The best way to do this is with a link to
  - http://creativecommons.org/licenses/by-sa/3.0/.

- Any of the above conditions can be waived if you get permission from the copyright holder.

- Nothing in this license impairs or restricts the author's moral rights.
Snapshot means

Write any one array element

Read multiple array elements

What about writing multiple elements?

can be extended to MRMW

Multi-reader multi-writer