PURDUE UNIVERSITY

Principles of Concurrency

Lecture 18
Deterministic Concurrency and Safe Futures

Starting Point: Futures

- Concurrency abstraction first proposed for MultiLisp
  - Transparent annotations on calls
  - Annotated method may be executed asynchronously (concurrently)
  - Result claimed later in execution
- Effectively serves as a speculation context
- Available in Java 1.5

Future API


```
public interface Future<V> {
    V get()
      throws InterruptedException,
             ExecutionException;
}

public class FutureTask<V>
  implements Future<V>, Runnable {
    FutureTask(Callable<V> callable)
      throws NullPointerException
    { ... }
    V get()
      throws InterruptedException,
             ExecutionException
    { ... }
    void run() { ... }
}

public interface Callable<V> {
    V call() throws Exception;
}
```

```
public class SafeFuture<V>
  implements Future<V>, Runnable {
    SafeFuture(Callable<V> callable)
      throws NullPointerException
    { ... }
    V get()
      throws InterruptedException,
             ExecutionException
    { ... }
    void run() { ... }
}
```

Equivalence


```
Callable<V> c = ...;
```

```
...
V v = c.call();
...
```

```
...
Future<V> f
  = new SafeFuture<V>(c);
f.run();
...
V v = f.get();
```

How should this equivalence be preserved?

Futures - Safety

If sequential program P is annotated with futures to yield concurrent program P_F, then the observable behavior of P is equivalent to P_F

- Easy to satisfy when no side-effects
- Problems arise with mutation of shared data
- Consider programs that only make use of futures – no explicit threads or locks

Consider: mutable Integer


```
class INT {
    int value;
    INT(int i) { value = i; }
    boolean eq(INT i) { return value == i.value; }
    INT sum(INT i) { return new INT(value + i.value); }
    INT add(INT i) { value += i.value; }
    INT mul(INT i) { value *= i.value; }
}
```

Example

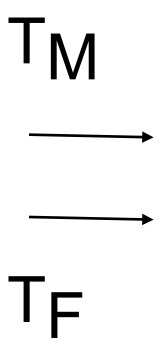

```
Future f = F[foo(i1, i2)];
boolean test =  bar(i1, i2);
if (test) global.add(f.get() );
```

```
static INT foo (INT i1, INT i2) {
    INT res = i1.sum(i2);
    return res; }

static boolean bar (INT i1, INT i2) {
    boolean res = i1.eq(i2);
    return res; }
```

Futures - Concurrent


```
Future f = F[foo(i1, i2)];
boolean test =  bar(i1, i2);
if (test) global.add(f.get() );
```

```
static INT foo (INT i1, INT i2) {
    INT res = i1.sum(i2);
    return res; }

static boolean bar (INT i1, INT i2) {
    boolean res = i1.eq(i2);
    return res; }
```

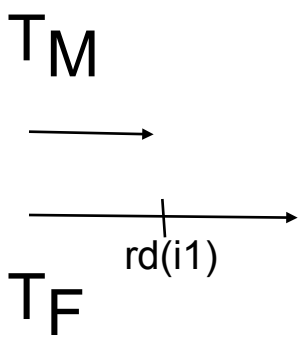


i1 = 5
i2 = 5
global = 0

Futures - Concurrent


```
Future f = F[foo(i1, i2)];
boolean test =  bar(i1, i2);
if (test) global.add(f.get() );
```

```
static INT foo (INT i1, INT i2) {
    INT res = i1.sum(i2);
    return res; }

static boolean bar (INT i1, INT i2) {
    boolean res = i1.eq(i2);
    return res; }
```

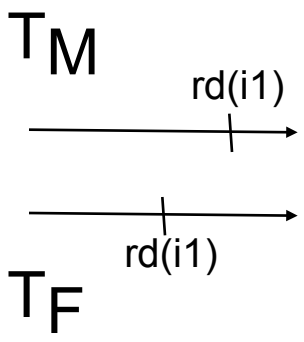


i1 = 5
i2 = 5
global = 0

Futures - Concurrent


```
Future f = F[foo(i1, i2)];
boolean test =  bar(i1, i2);
if (test) global.add(f.get() );
```

```
static INT foo (INT i1, INT i2) {
    INT res = i1.sum(i2);
    return res; }

static boolean bar (INT i1, INT i2) {
    boolean res = i1.eq(i2);
    return res; }
```

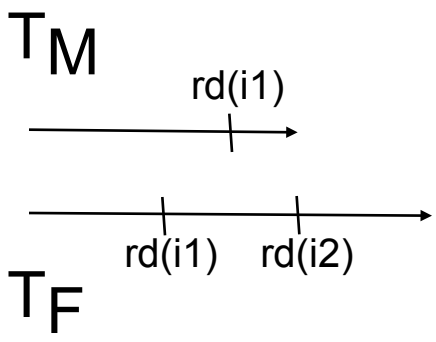


i1 = 5
i2 = 5
global = 0

Futures - Concurrent


```
Future f = F[foo(i1, i2)];
boolean test =  bar(i1, i2);
if (test) global.add(f.get() );
```

```
static INT foo (INT i1, INT i2) {
    INT res = i1.sum(i2);
    return res; }

static boolean bar (INT i1, INT i2) {
    boolean res = i1.eq(i2);
    return res; }
```

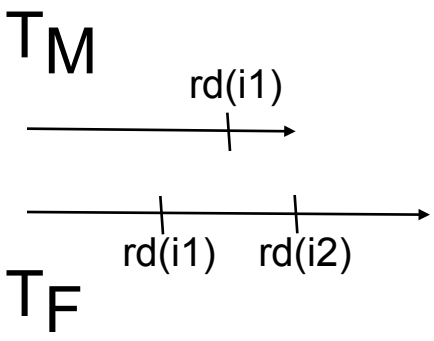


i1 = 5
i2 = 5
global = 0

Futures - Concurrent


```
Future f = F[foo(i1, i2)];
boolean test =  bar(i1, i2);
if (test) global.add(f.get() );
```

```
static INT foo (INT i1, INT i2) {
    INT res = i1.sum(i2); → 10
    return res; }

static boolean bar (INT i1, INT i2) {
    boolean res = i1.eq(i2);
    return res; }
```

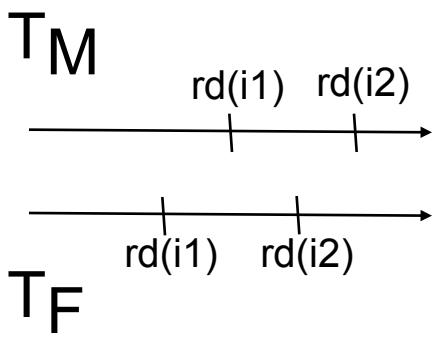


i1 = 5
i2 = 5
global = 0

Futures - Concurrent


```
Future f = F[foo(i1, i2)];
boolean test =  bar(i1, i2);
if (test) global.add(f.get() );
```

```
static INT foo (INT i1, INT i2) {
    INT res = i1.sum(i2); → 10
    return res; }

static boolean bar (INT i1, INT i2) {
    boolean res = i1.eq(i2);
    return res; }
```

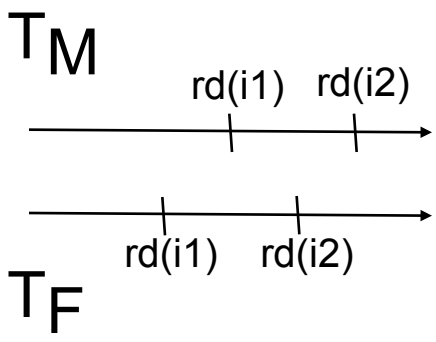


i1 = 5
i2 = 5
global = 0

Futures - Concurrent


```
Future f = F[foo(i1, i2)];
boolean test =  bar(i1, i2);
if (test) global.add(f.get() );
```

```
static INT foo (INT i1, INT i2) {
    INT res = i1.sum(i2); → 10
    return res; }

static boolean bar (INT i1, INT i2) {
    boolean res = i1.eq(i2); → true
    return res; }
```

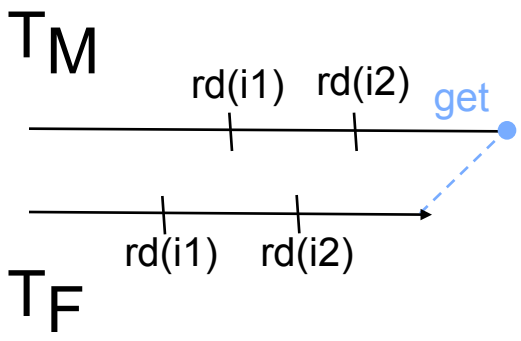


i1 = 5
i2 = 5
global = 0

Futures - Concurrent


```
Future f = F[foo(i1, i2)];
boolean test =  bar(i1, i2);
if (test) global.add(f.get() );
```

```
static INT foo (INT i1, INT i2) {
    INT res = i1.sum(i2); → 10
    return res; }

static boolean bar (INT i1, INT i2) {
    boolean res = i1.eq(i2); → true
    return res; }
```

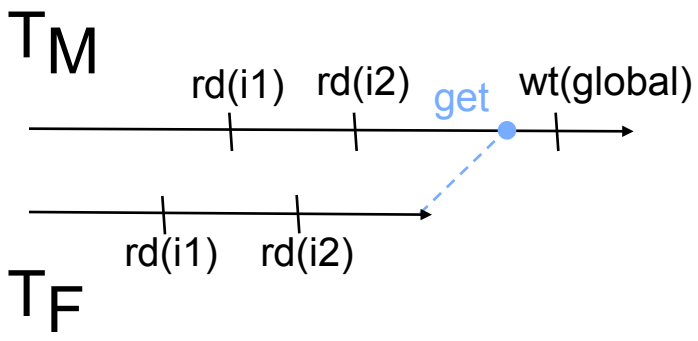


i1 = 5
i2 = 5
global = 0

Futures - Concurrent


```
Future f = F[foo(i1, i2)];
boolean test =  bar(i1, i2);
if (test) global.add(f.get() );
```

```
static INT foo (INT i1, INT i2) {
    INT res = i1.sum(i2); → 10
    return res; }

static boolean bar (INT i1, INT i2) {
    boolean res = i1.eq(i2); → true
    return res; }
```



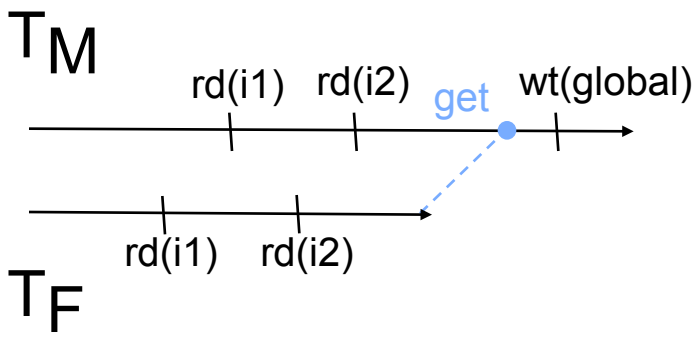
i1 = 5
i2 = 5
global = 10

Futures - Concurrent


```
Future f = F[foo(i1, i2)];
boolean test =  bar(i1, i2);
if (test) global.add(f.get() );
```

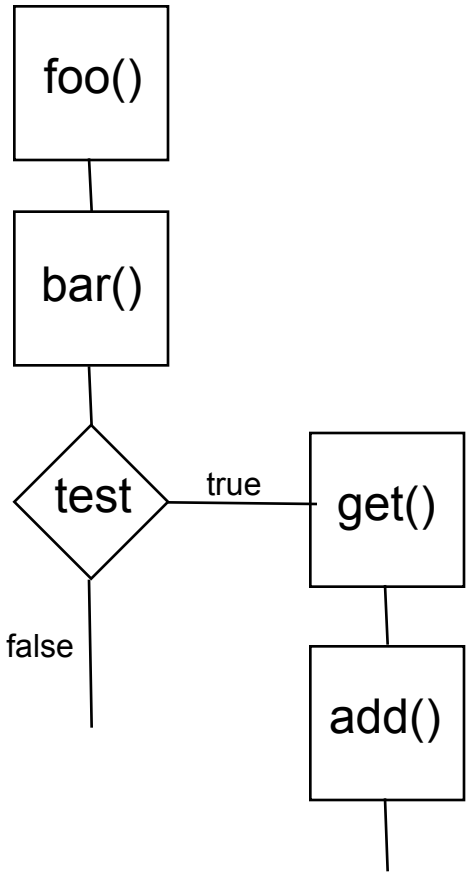
i1 = 5 i2 = 5 global = 10

```
static INT foo (INT i1, INT i2) {
    INT res = i1.sum(i2); → 10
    return res; }

static boolean bar (INT i1, INT i2) {
    boolean res = i1.eq(i2); → true
    return res; }
```



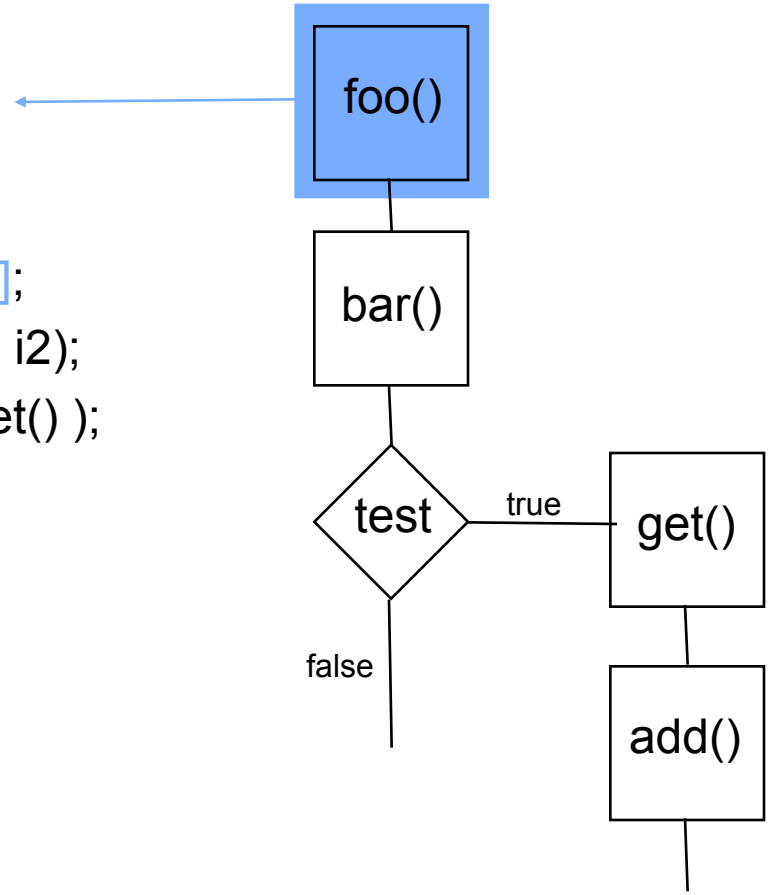
Terminology


```
Future f = F[foo(i1, i2)];
boolean test =  bar(i1, i2);
if (test) global.add(f.get() );
```



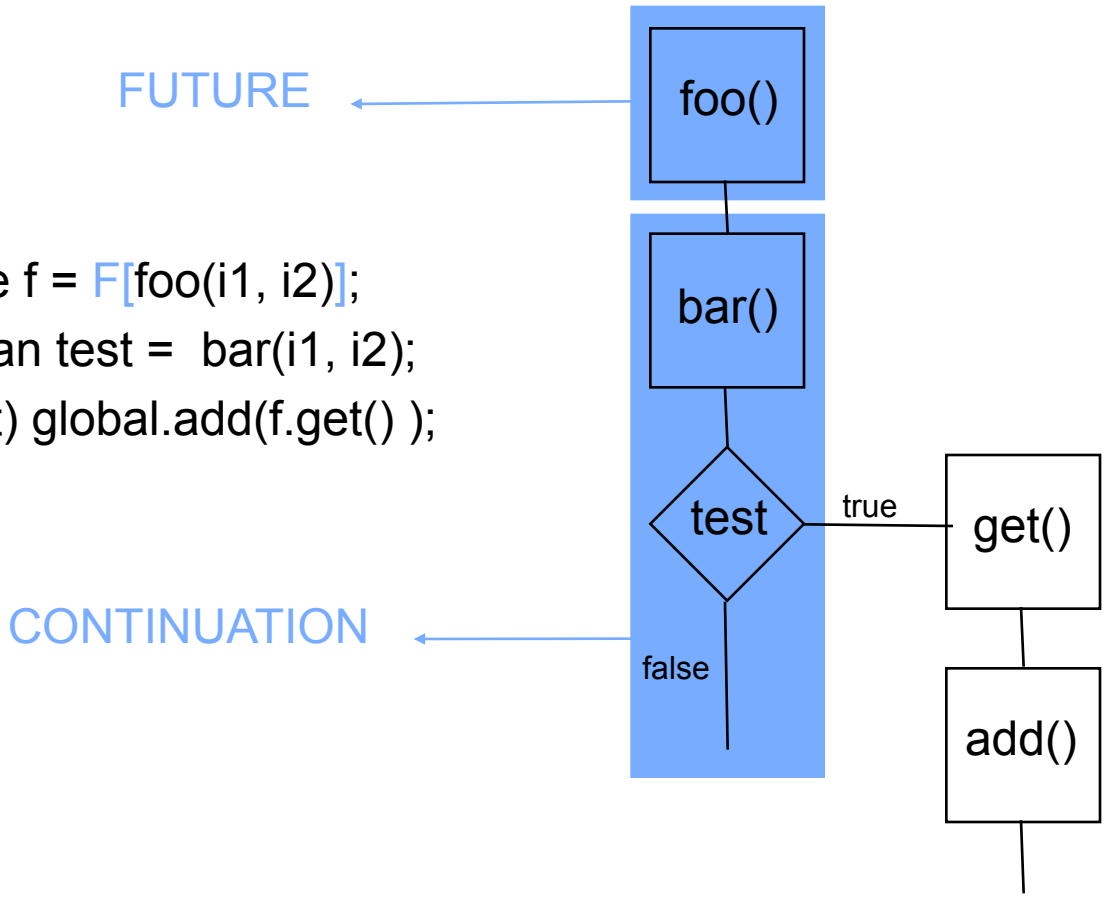
Terminology


```
Future f = F[foo(i1, i2)];
boolean test =  bar(i1, i2);
if (test) global.add(f.get() );
```



Terminology


```
Future f = F[foo(i1, i2)];
boolean test =  bar(i1, i2);
if (test) global.add(f.get() );
```



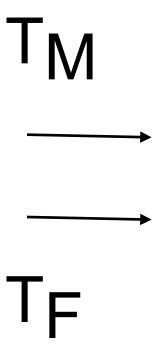
Observations

- No data dependency between future and its continuation
- Concurrent execution trivially equivalent to a serial one, an execution in which the future annotation is erased
- What if we introduce side-effects?

Example: Concurrent


```
Future f = F[foo(i1, i2)];
boolean test =  bar(i1, i2);
if (test) global.add(f.get() );
```

```
static INT foo (INT i1, INT i2) {
    INT res = i1.sum(i2);
    i1.sub(1); i2.sub(1);
    return res; }

static boolean bar (INT i1, INT i2) {
    boolean res = i1.eq(i2);
    i1.mul(2); i2.mul(2);
    return res; }
```

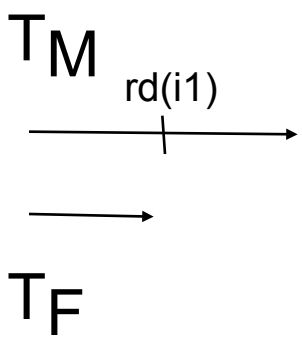


i1 = 5
i2 = 5
global = 0

Example: Concurrent


```
Future f = F[foo(i1, i2)];
boolean test =  bar(i1, i2);
if (test) global.add(f.get() );
```

```
static INT foo (INT i1, INT i2) {
    INT res = i1.sum(i2);
    i1.sub(1); i2.sub(1);
    return res; }

static boolean bar (INT i1, INT i2) {
    boolean res = i1.eq(i2);
    i1.mul(2); i2.mul(2);
    return res; }
```

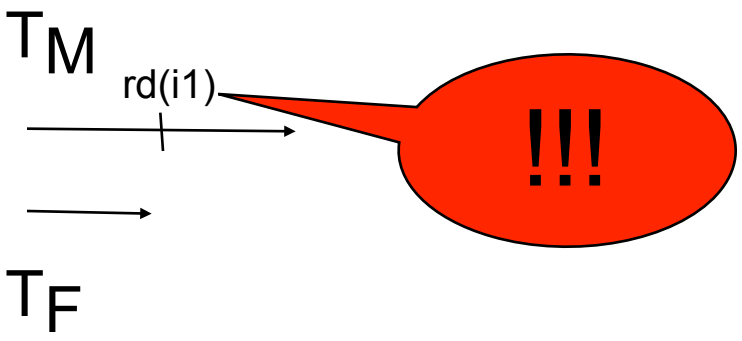


i1 = 5
i2 = 5
global = 0

Example: Concurrent


```
Future f = F[foo(i1, i2)];
boolean test =  bar(i1, i2);
if (test) global.add(f.get() );
```

```
static INT foo (INT i1, INT i2) {
    INT res = i1.sum(i2);
    i1.sub(1); i2.sub(1);
    return res; }

static boolean bar (INT i1, INT i2) {
    boolean res = i1.eq(i2);
    i1.mul(2); i2.mul(2);
    return res; }
```

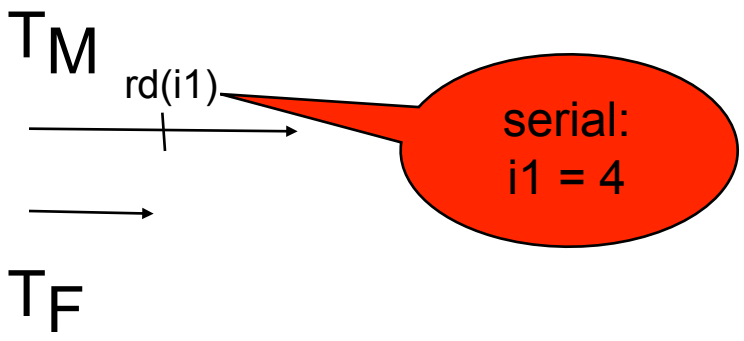


i1 = 5
i2 = 5
global = 0

Example: Concurrent


```
Future f = F[foo(i1, i2)];
boolean test =  bar(i1, i2);
if (test) global.add(f.get() );
```

```
static INT foo (INT i1, INT i2) {
    INT res = i1.sum(i2);
    i1.sub(1); i2.sub(1);
    return res; }

static boolean bar (INT i1, INT i2) {
    boolean res = i1.eq(i2);
    i1.mul(2); i2.mul(2);
    return res; }
```

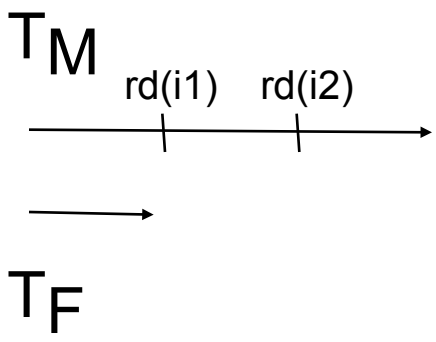


i1 = 5
i2 = 5
global = 0

Example: Concurrent


```
Future f = F[foo(i1, i2)];
boolean test =  bar(i1, i2);
if (test) global.add(f.get() );
```

```
static INT foo (INT i1, INT i2) {
    INT res = i1.sum(i2);
    i1.sub(1); i2.sub(1);
    return res; }

static boolean bar (INT i1, INT i2) {
    boolean res = i1.eq(i2);
    i1.mul(2); i2.mul(2);
    return res; }
```

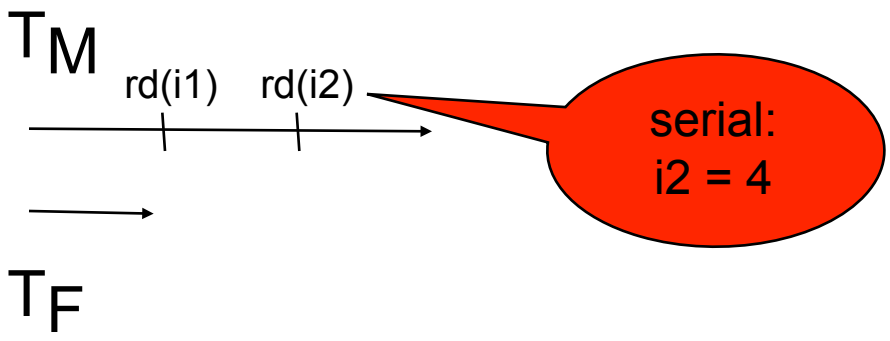


i1 = 5
i2 = 5
global = 0

Example: Concurrent


```
Future f = F[foo(i1, i2)];
boolean test =  bar(i1, i2);
if (test) global.add(f.get() );
```

```
static INT foo (INT i1, INT i2) {
    INT res = i1.sum(i2);
    i1.sub(1); i2.sub(1);
    return res; }

static boolean bar (INT i1, INT i2) {
    boolean res = i1.eq(i2);
    i1.mul(2); i2.mul(2);
    return res; }
```

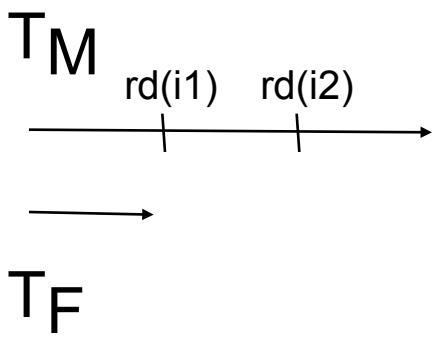


i1 = 5
i2 = 5
global = 0

Example: Concurrent


```
Future f = F[foo(i1, i2)];
boolean test =  bar(i1, i2);
if (test) global.add(f.get() );
```

```
static INT foo (INT i1, INT i2) {
    INT res = i1.sum(i2);
    i1.sub(1); i2.sub(1);
    return res; }

static boolean bar (INT i1, INT i2) {
    boolean res = i1.eq(i2); → true
    i1.mul(2); i2.mul(2);
    return res; }
```

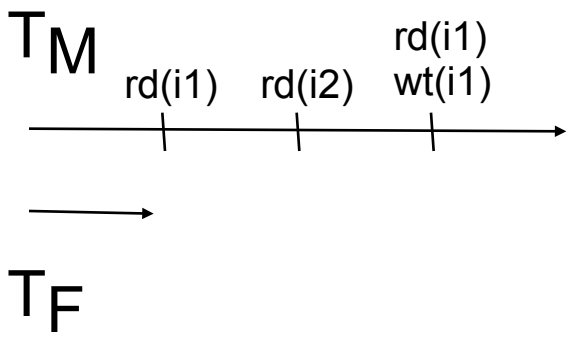


i1 = 5
i2 = 5
global = 0

Example: Concurrent


```
Future f = F[foo(i1, i2)];
boolean test =  bar(i1, i2);
if (test) global.add(f.get() );
```

```
static INT foo (INT i1, INT i2) {
    INT res = i1.sum(i2);
    i1.sub(1); i2.sub(1);
    return res; }

static boolean bar (INT i1, INT i2) {
    boolean res = i1.eq(i2); → true
    i1.mul(2); i2.mul(2);
    return res; }
```

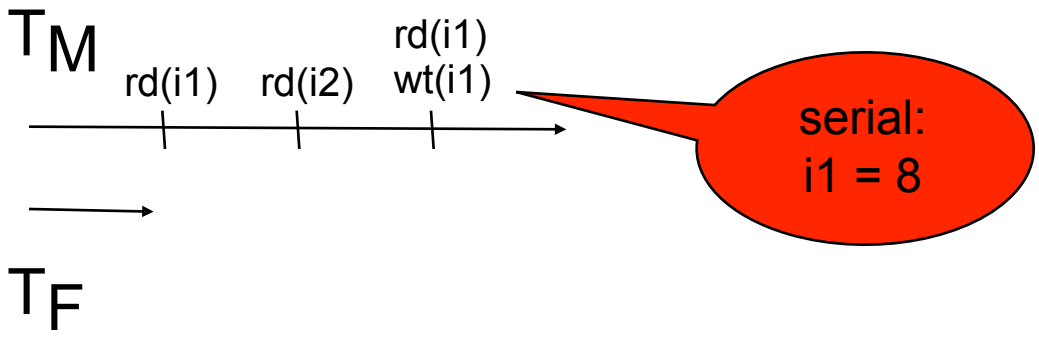


i1 = 10
i2 = 5
global = 0

Example: Concurrent


```
Future f = F[foo(i1, i2)];
boolean test =  bar(i1, i2);
if (test) global.add(f.get() );
```

```
static INT foo (INT i1, INT i2) {
    INT res = i1.sum(i2);
    i1.sub(1); i2.sub(1);
    return res; }

static boolean bar (INT i1, INT i2) {
    boolean res = i1.eq(i2); → true
    i1.mul(2); i2.mul(2);
    return res; }
```

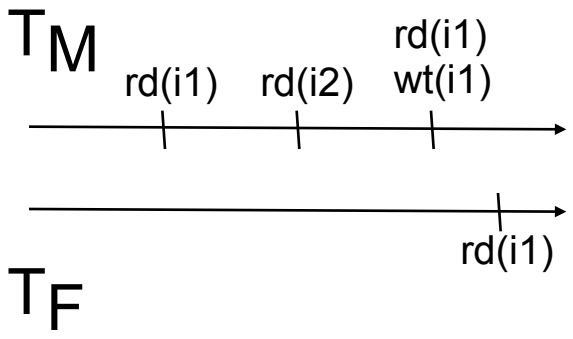


i1 = 10
i2 = 5
global = 0

Example: Concurrent


```
Future f = F[foo(i1, i2)];
boolean test =  bar(i1, i2);
if (test) global.add(f.get() );
```

```
static INT foo (INT i1, INT i2) {
    INT res = i1.sum(i2);
    i1.sub(1); i2.sub(1);
    return res; }

static boolean bar (INT i1, INT i2) {
    boolean res = i1.eq(i2); → true
    i1.mul(2); i2.mul(2);
    return res; }
```

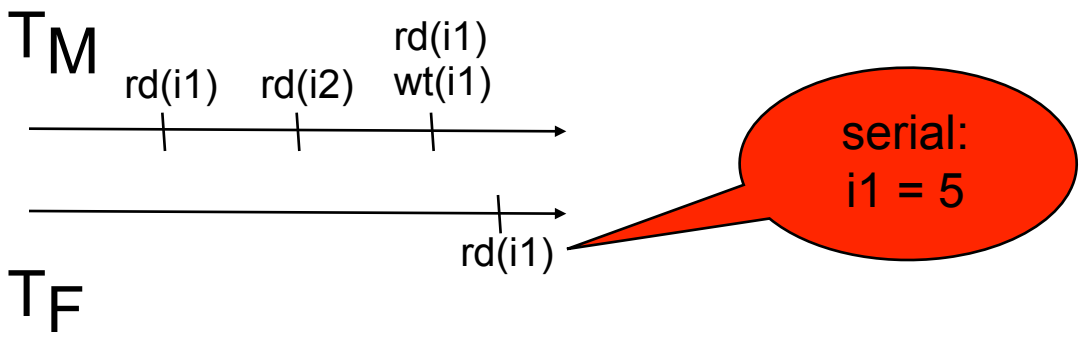


i1 = 10
i2 = 5
global = 0

Example: Concurrent


```
Future f = F[foo(i1, i2)];
boolean test =  bar(i1, i2);
if (test) global.add(f.get() );
```

```
static INT foo (INT i1, INT i2) {
    INT res = i1.sum(i2);
    i1.sub(1); i2.sub(1);
    return res; }

static boolean bar (INT i1, INT i2) {
    boolean res = i1.eq(i2); → true
    i1.mul(2); i2.mul(2);
    return res; }
```

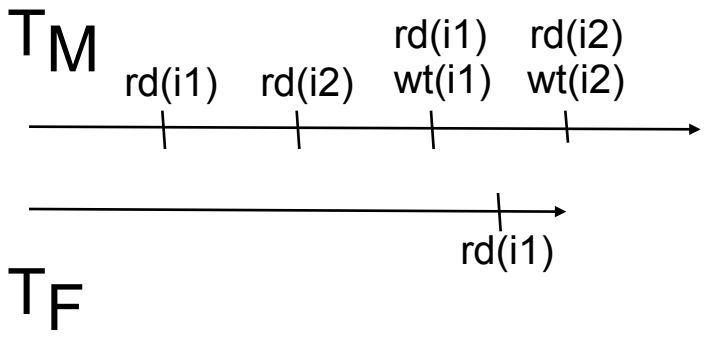


i1 = 10
i2 = 5
global = 0

Example: Concurrent


```
Future f = F[foo(i1, i2)];
boolean test =  bar(i1, i2);
if (test) global.add(f.get() );
```

```
static INT foo (INT i1, INT i2) {
    INT res = i1.sum(i2);
    i1.sub(1); i2.sub(1);
    return res; }

static boolean bar (INT i1, INT i2) {
    boolean res = i1.eq(i2); → true
    i1.mul(2); i2.mul(2);
    return res; }
```

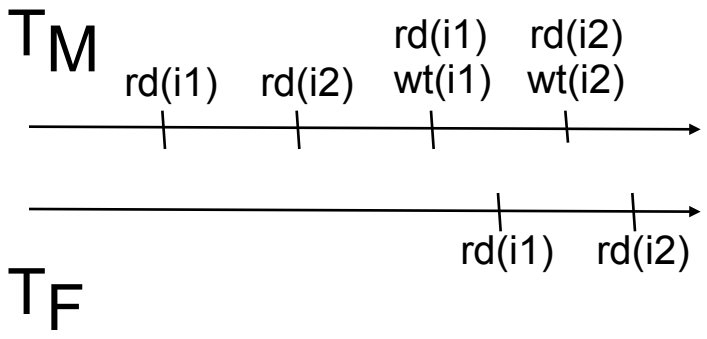


i1 = 10
i2 = 10
global = 0

Example: Concurrent


```
Future f = F[foo(i1, i2)];
boolean test =  bar(i1, i2);
if (test) global.add(f.get() );
```

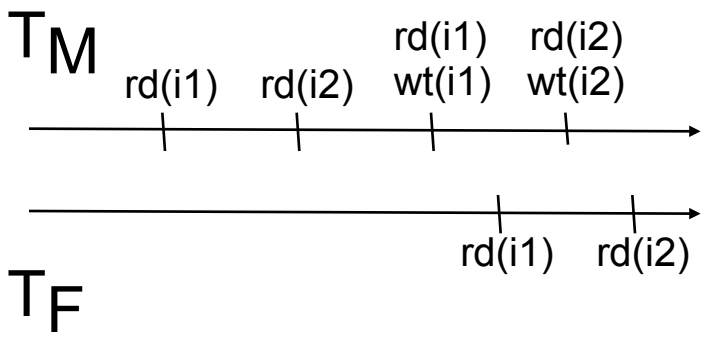
```
static INT foo (INT i1, INT i2) {
    INT res = i1.sum(i2);
    i1.sub(1); i2.sub(1);
    return res; }

static boolean bar (INT i1, INT i2) {
    boolean res = i1.eq(i2); → true
    i1.mul(2); i2.mul(2);
    return res; }
```



Example: Concurrent


```
Future f = F[foo(i1, i2)];
boolean test =  bar(i1, i2);
if (test) global.add(f.get() );
```

```
static INT foo (INT i1, INT i2) {
    INT res = i1.sum(i2); → 20
    i1.sub(1); i2.sub(1);
    return res; }

static boolean bar (INT i1, INT i2) {
    boolean res = i1.eq(i2); → true
    i1.mul(2); i2.mul(2);
    return res; }
```



i1 = 10
i2 = 10
global = 0

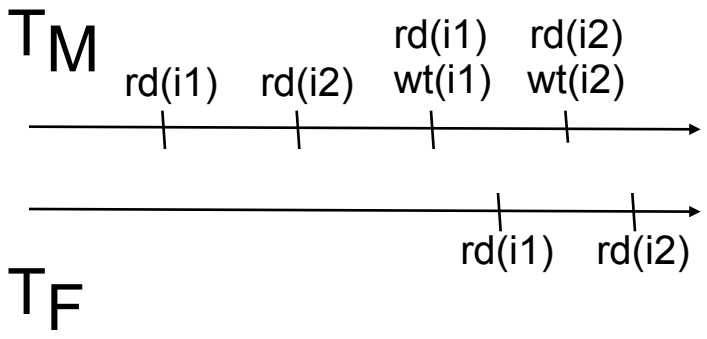
Example: Concurrent


```
Future f = F[foo(i1, i2)];
boolean test =  bar(i1, i2);
if (test) global.add(f.get() );
```

```
static INT foo (INT i1, INT i2) {
    INT res = i1.sum(i2); → 20
    i1.sub(1); i2.sub(1);
    return res; }
```

serial: 10

```
static boolean bar (INT i1, INT i2) {
    boolean res = i1.eq(i2); → true
    i1.mul(2); i2.mul(2);
    return res; }
```

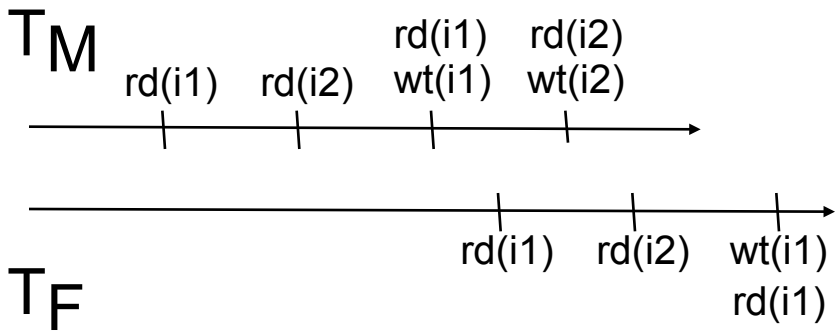


i1 = 10
i2 = 10
global = 0

Example: Concurrent


```
Future f = F[foo(i1, i2)];
boolean test =  bar(i1, i2);
if (test) global.add(f.get() );
```

```
static INT foo (INT i1, INT i2) {
   INT res = i1.sum(i2); → 20
   i1.sub(1); i2.sub(1);
   return res; }

static boolean bar (INT i1, INT i2) {
   boolean res = i1.eq(i2); → true
   i1.mul(2); i2.mul(2);
   return res; }
```

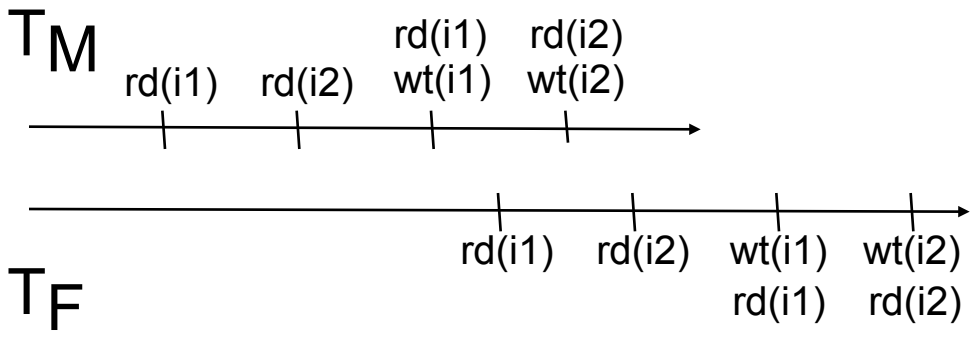


i1 = 9
i2 = 10
global = 0

Example: Concurrent


```
Future f = F[foo(i1, i2)];
boolean test =  bar(i1, i2);
if (test) global.add(f.get() );
```

```
static INT foo (INT i1, INT i2) {
    INT res = i1.sum(i2); → 20
    i1.sub(1); i2.sub(1);
    return res; }

static boolean bar (INT i1, INT i2) {
    boolean res = i1.eq(i2); → true
    i1.mul(2); i2.mul(2);
    return res; }
```

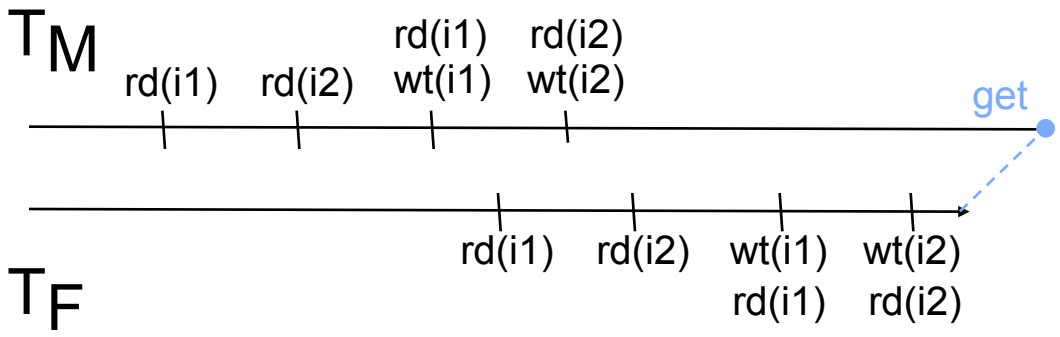


i1 = 9
i2 = 9
global = 0

Example: Concurrent


```
Future f = F[foo(i1, i2)];
boolean test =  bar(i1, i2);
if (test) global.add(f.get() );
```

```
static INT foo (INT i1, INT i2) {
    INT res = i1.sum(i2); → 20
    i1.sub(1); i2.sub(1);
    return res; }

static boolean bar (INT i1, INT i2) {
    boolean res = i1.eq(i2); → true
    i1.mul(2); i2.mul(2);
    return res; }
```



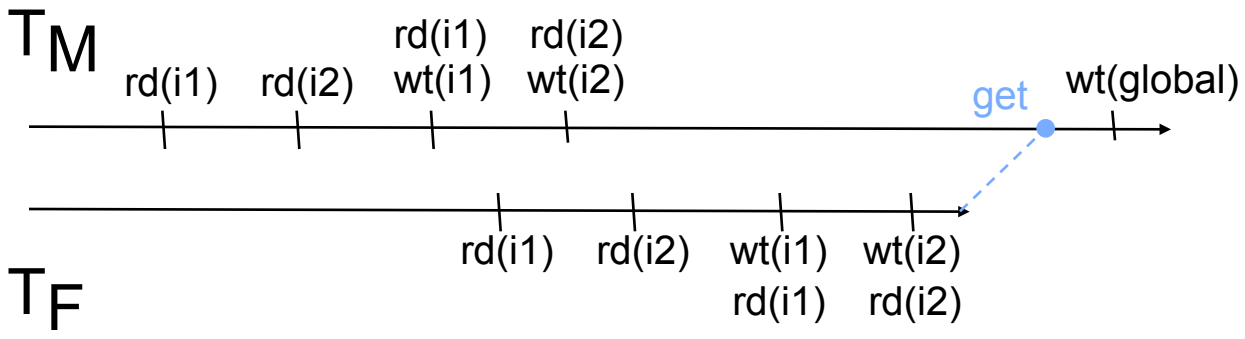
i1 = 9
i2 = 9
global = 0

Example: Concurrent


```
Future f = F[foo(i1, i2)];
boolean test =  bar(i1, i2);
if (test) global.add(f.get() );
```

i1 = 8 i2 = 8 global = 10

```
static INT foo (INT i1, INT i2) {
    INT res = i1.sum(i2); → 20
    i1.sub(1); i2.sub(1);
    return res; }

static boolean bar (INT i1, INT i2) {
    boolean res = i1.eq(i2); → true
    i1.mul(2); i2.mul(2);
    return res; }
```



i1 = 9
i2 = 9
global = 20

What Happened?

- Concurrency of shared updates led to unexpected behavior
- Updates from continuation leaked into future: foo() should not see results of bar()
- Results computed by future were not available for continuation: bar() supposed to see results of foo()

Safety Guarantees

- Future never reads a location written by its continuation.
- Continuation never reads a location written by the future.
- Unfortunately, cannot trivially apply static race detection techniques:
  - No locks or atomic sections
    - Even with locks, could only guarantee serializability, not serial order
  - No syntactic synchronization boundary on continuation
- Futures and their associated claims are decoupled

Mediating Conflicts

Two distinct questions:

- Does a conflict exist between a future and a continuation?
  - Where should barriers be inserted?
- What should the scope of a continuation be?
  - Must guarantee that the continuation scope encapsulates all potential conflict points.
  - Syntactic claim or touch points insufficient.

Dependency Violations

C_f

```
int i = o.bar;
o.foo = 0;
```

C_f	C_c
read(o)	
	write(o)
	read(o)
write(o)	

(a) Forward

C_c

```
o.bar = 0;
int j = o.foo;
```

C_f	C_c
	write(o)
read(o)	
write(o)	
	read(o)

(b) Backward

- Forward dependency violations can be handled by tracking data dependencies.
- Backward dependency violations can be handled by versioning updates. Future never sees a premature update by its continuation.

Safe Futures

Runtime system maintains appearance of serial execution

- Versions shared data
- Tracks shared data accesses
- Revokes non-serial executions

Implementation

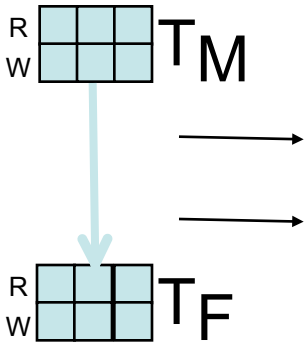
- Data accesses hashed into read and write maps
- Maps used by continuation to detect conflicts for accesses from its future
- Versions used by future to prevent seeing updates by its continuation
- Rollback when conflict detected

Safe Futures


```
Future f = F[foo(i1, i2)];
boolean test =  bar(i1, i2);
if (test) global.add(f.get() );
```

```
static INT foo (INT i1, INT i2) {
    INT res = i1.sum(i2);
    i1.sub(1); i2.sub(1);
    return res; }

static boolean bar (INT i1, INT i2) {
    boolean res = i1.eq(i2);
    i1.mul(2); i2.mul(2);
    return res; }
```

i1 = 5 i2 = 5 global = 0

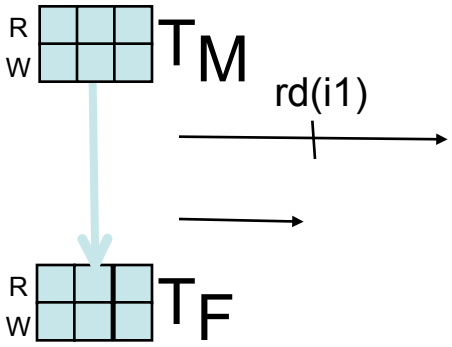


Safe Futures


```
Future f = F[foo(i1, i2)];
boolean test =  bar(i1, i2);
if (test) global.add(f.get() );
```

```
static INT foo (INT i1, INT i2) {
    INT res = i1.sum(i2);
    i1.sub(1); i2.sub(1);
    return res; }

static boolean bar (INT i1, INT i2) {
    boolean res = i1.eq(i2);
    i1.mul(2); i2.mul(2);
    return res; }
```

i1 = 5 i2 = 5 global = 0

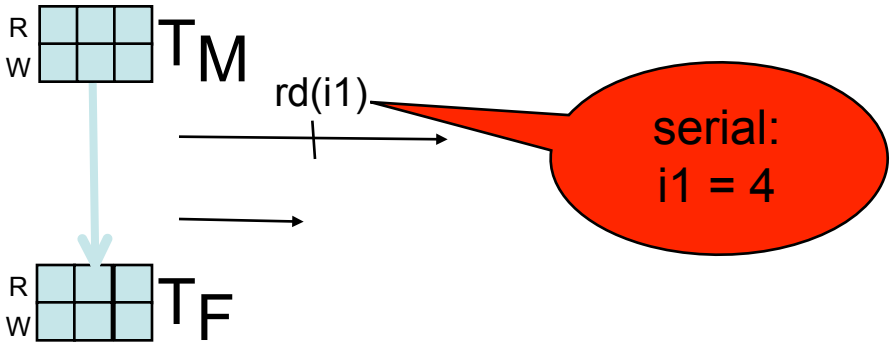


Safe Futures


```
Future f = F[foo(i1, i2)];
boolean test =  bar(i1, i2);
if (test) global.add(f.get() );
```

```
static INT foo (INT i1, INT i2) {
    INT res = i1.sum(i2);
    i1.sub(1); i2.sub(1);
    return res; }

static boolean bar (INT i1, INT i2) {
    boolean res = i1.eq(i2);
    i1.mul(2); i2.mul(2);
    return res; }
```

i1 = 5 i2 = 5 global = 0

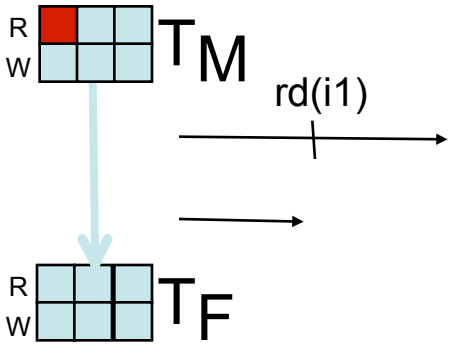


Safe Futures


```
Future f = F[foo(i1, i2)];
boolean test =  bar(i1, i2);
if (test) global.add(f.get() );
```

```
static INT foo (INT i1, INT i2) {
    INT res = i1.sum(i2);
    i1.sub(1); i2.sub(1);
    return res; }

static boolean bar (INT i1, INT i2) {
    boolean res = i1.eq(i2);
    i1.mul(2); i2.mul(2);
    return res; }
```

i1 = 5 i2 = 5 global = 0

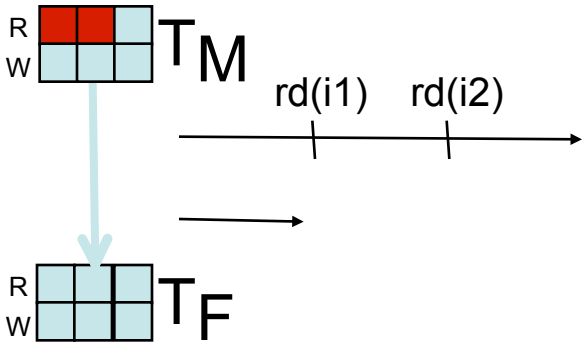


Safe Futures


```
Future f = F[foo(i1, i2)];
boolean test =  bar(i1, i2);
if (test) global.add(f.get() );
```

```
static INT foo (INT i1, INT i2) {
    INT res = i1.sum(i2);
    i1.sub(1); i2.sub(1);
    return res; }

static boolean bar (INT i1, INT i2) {
    boolean res = i1.eq(i2);
    i1.mul(2); i2.mul(2);
    return res; }
```

i1 = 5 i2 = 5 global = 0

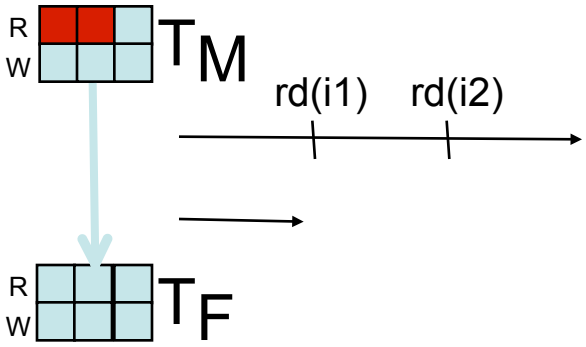


Safe Futures


```
Future f = F[foo(i1, i2)];
boolean test =  bar(i1, i2);
if (test) global.add(f.get() );
```

```
static INT foo (INT i1, INT i2) {
    INT res = i1.sum(i2);
    i1.sub(1); i2.sub(1);
    return res; }

static boolean bar (INT i1, INT i2) {
    boolean res = i1.eq(i2); → true
    i1.mul(2); i2.mul(2);
    return res; }
```

i1 = 5 i2 = 5 global = 0

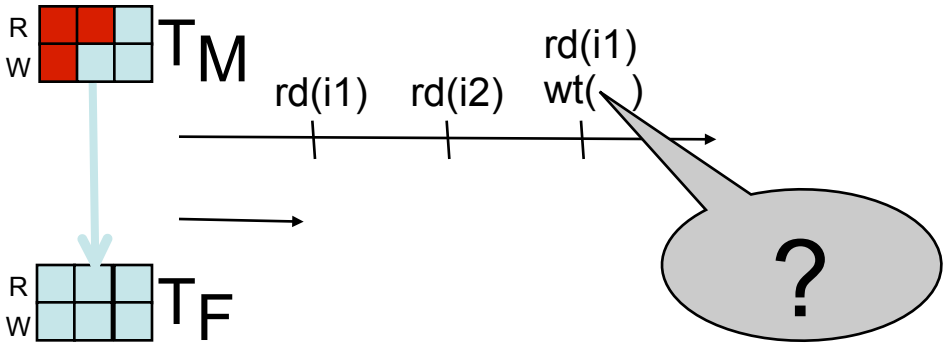


Safe Futures


```
Future f = F[foo(i1, i2)];
boolean test =  bar(i1, i2);
if (test) global.add(f.get() );
```

```
static INT foo (INT i1, INT i2) {
    INT res = i1.sum(i2);
    i1.sub(1); i2.sub(1);
    return res; }

static boolean bar (INT i1, INT i2) {
    boolean res = i1.eq(i2); → true
    i1.mul(2); i2.mul(2);
    return res; }
```

i1 = 5 i2 = 5 global = 0



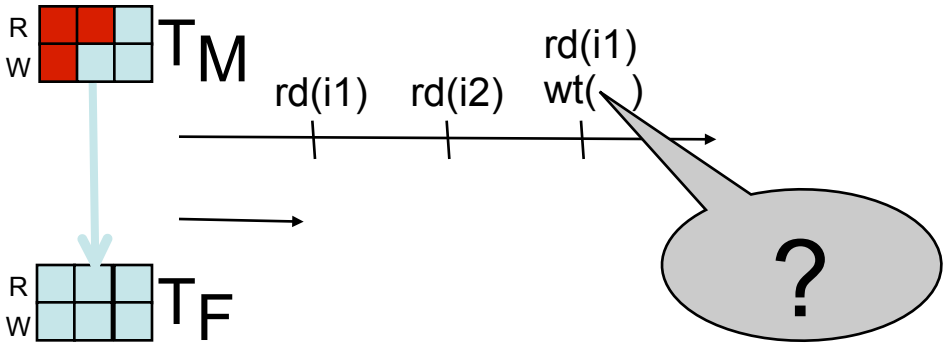
Safe Futures


```
Future f = F[foo(i1, i2)];
boolean test =  bar(i1, i2);
if (test) global.add(f.get() );
```

```
static INT foo (INT i1, INT i2) {
    INT res = i1.sum(i2);
    i1.sub(1); i2.sub(1);
    return res; }

static boolean bar (INT i1, INT i2) {
    boolean res = i1.eq(i2); → true
    i1.mul(2); i2.mul(2);
    return res; }
```

$i1_M$

i1 = 5 i2 = 5 global = 0

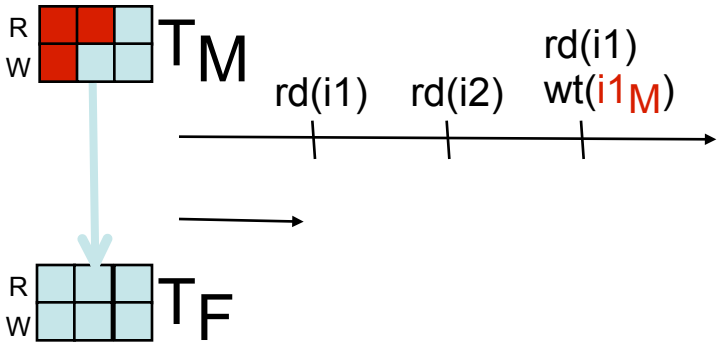


Safe Futures


```
Future f = F[foo(i1, i2)];
boolean test =  bar(i1, i2);
if (test) global.add(f.get() );
```

$i1_M = 10$

i1 = 5 i2 = 5 global = 0



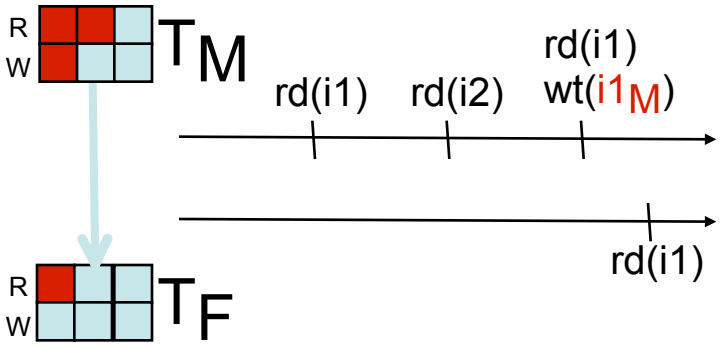

```
static INT foo (INT i1, INT i2) {
    INT res = i1.sum(i2);
    i1.sub(1); i2.sub(1);
    return res; }

static boolean bar (INT i1, INT i2) {
    boolean res = i1.eq(i2); → true
    i1.mul(2); i2.mul(2);
    return res; }
```

Safe Futures


```
Future f = F[foo(i1, i2)];
boolean test =  bar(i1, i2);
if (test) global.add(f.get() );
```

$i1_M = 10$

i1 = 5 i2 = 5 global = 0




```
static INT foo (INT i1, INT i2) {
    INT res = i1.sum(i2);
    i1.sub(1); i2.sub(1);
    return res; }

static boolean bar (INT i1, INT i2) {
    boolean res = i1.eq(i2); → true
    i1.mul(2); i2.mul(2);
    return res; }
```

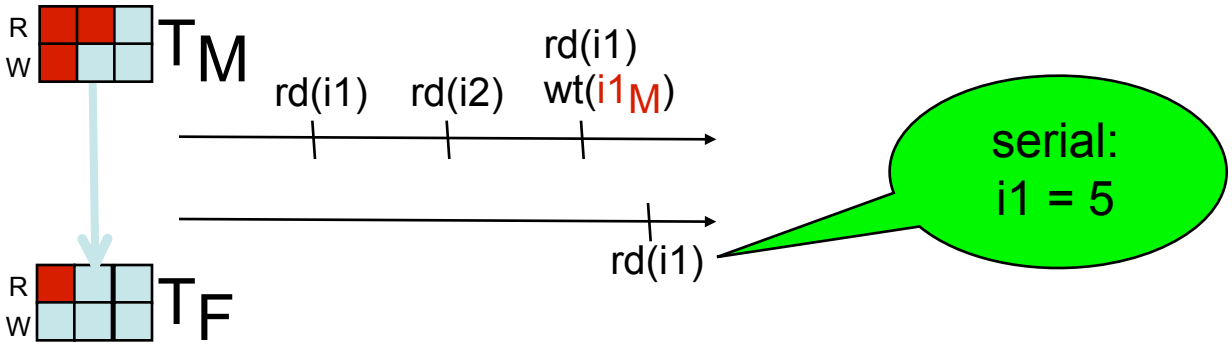
Safe Futures


```
Future f = F[foo(i1, i2)];
boolean test =  bar(i1, i2);
if (test) global.add(f.get() );
```

```
static INT foo (INT i1, INT i2) {
    INT res = i1.sum(i2);
    i1.sub(1); i2.sub(1);
    return res; }

static boolean bar (INT i1, INT i2) {
    boolean res = i1.eq(i2); → true
    i1.mul(2); i2.mul(2);
    return res; }
```

$i1_M = 10$

i1 = 5 i2 = 5 global = 0

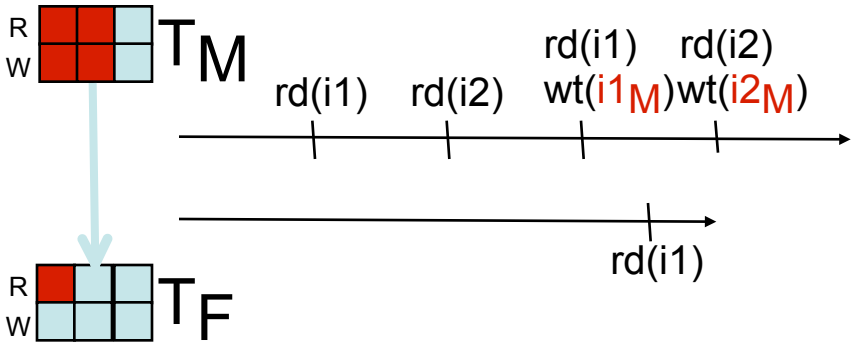


Safe Futures


```
Future f = F[foo(i1, i2)];
boolean test =  bar(i1, i2);
if (test) global.add(f.get() );
```

$i1_M = 10$ $i2_M = 10$

$i1 = 5$ $i2 = 5$ global = 0




```
static INT foo (INT i1, INT i2) {
    INT res = i1.sum(i2);
    i1.sub(1); i2.sub(1);
    return res; }

static boolean bar (INT i1, INT i2) {
    boolean res = i1.eq(i2); → true
    i1.mul(2); i2.mul(2);
    return res; }
```

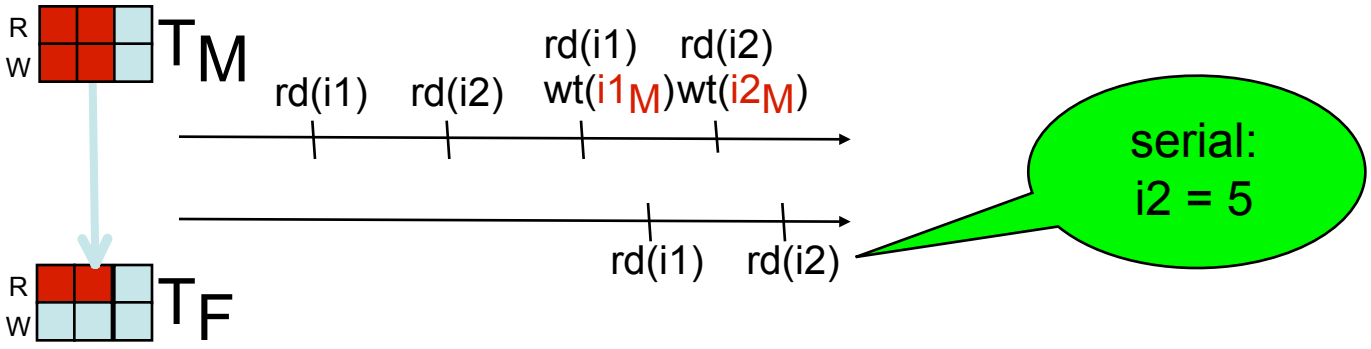
Safe Futures


```
Future f = F[foo(i1, i2)];
boolean test =  bar(i1, i2);
if (test) global.add(f.get() );
```

```
static INT foo (INT i1, INT i2) {
    INT res = i1.sum(i2);
    i1.sub(1); i2.sub(1);
    return res; }

static boolean bar (INT i1, INT i2) {
    boolean res = i1.eq(i2); → true
    i1.mul(2); i2.mul(2);
    return res; }
```

$i1_M = 10$ $i2_M = 10$

$i1 = 5$ $i2 = 5$ global = 0

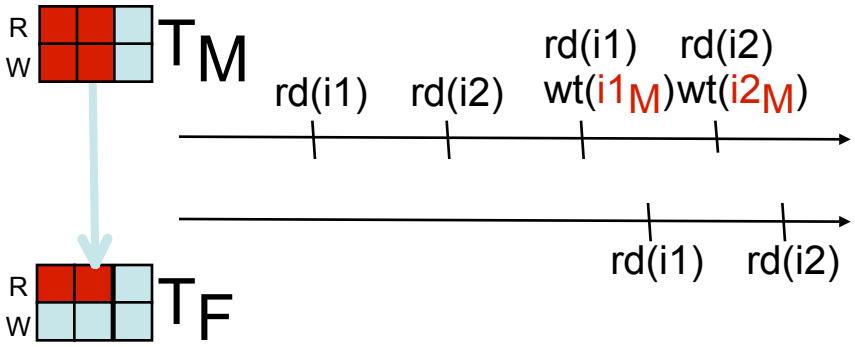


Safe Futures


```
Future f = F[foo(i1, i2)];
boolean test =  bar(i1, i2);
if (test) global.add(f.get() );
```

$i1_M = 10$ $i2_M = 10$

i1 = 5 i2 = 5 global = 0




```
static INT foo (INT i1, INT i2) {
   INT res = i1.sum(i2); → 10
   i1.sub(1); i2.sub(1);
   return res; }
```

serial: 10

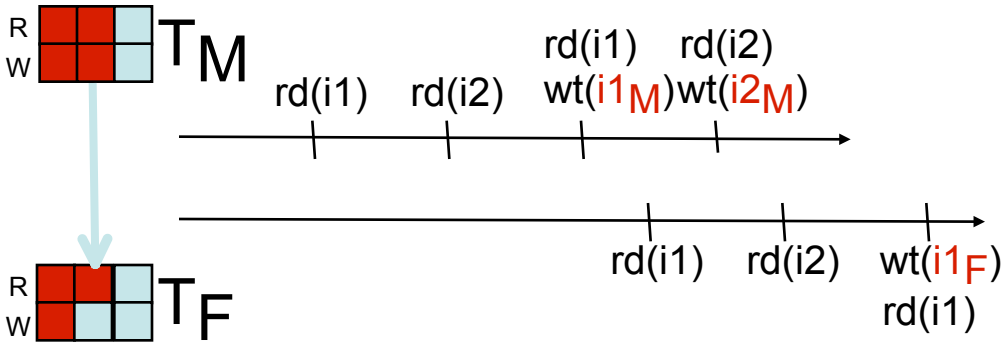
```
static boolean bar (INT i1, INT i2) {
   boolean res = i1.eq(i2); → true
   i1.mul(2); i2.mul(2);
   return res; }
```

Safe Futures


```
Future f = F[foo(i1, i2)];
boolean test =  bar(i1, i2);
if (test) global.add(f.get() );
```

$i1_F = 4$
$i1_M = 10$ $i2_M = 10$
$i1 = 5$ $i2 = 5$ global = 0

```
static INT foo (INT i1, INT i2) {
   INT res = i1.sum(i2); → 10
   i1.sub(1); i2.sub(1);
   return res; }

static boolean bar (INT i1, INT i2) {
   boolean res = i1.eq(i2); → true
   i1.mul(2); i2.mul(2);
   return res; }
```

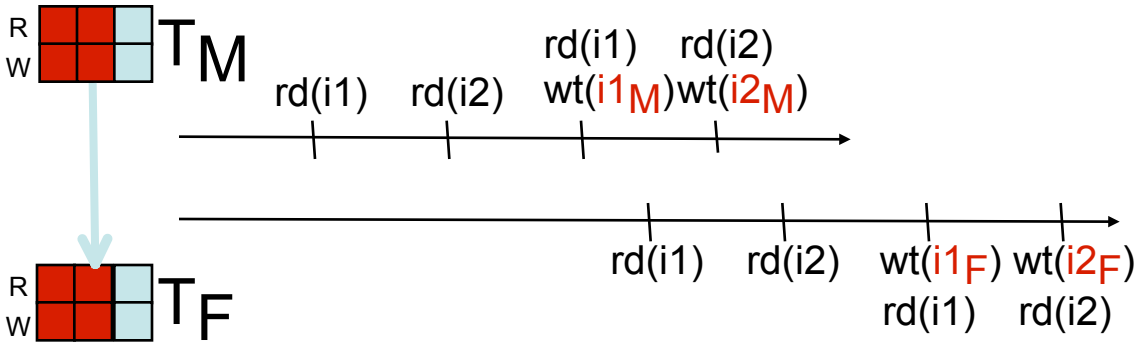


Safe Futures


```
Future f = F[foo(i1, i2)];
boolean test =  bar(i1, i2);
if (test) global.add(f.get() );
```

$i1_F = 4$ $i2_F = 4$
$i1_M = 10$ $i2_M = 10$
$i1 = 5$ $i2 = 5$ $global = 0$

```
static INT foo (INT i1, INT i2) {
    INT res = i1.sum(i2); → 10
    i1.sub(1); i2.sub(1);
    return res; }

static boolean bar (INT i1, INT i2) {
    boolean res = i1.eq(i2); → true
    i1.mul(2); i2.mul(2);
    return res; }
```

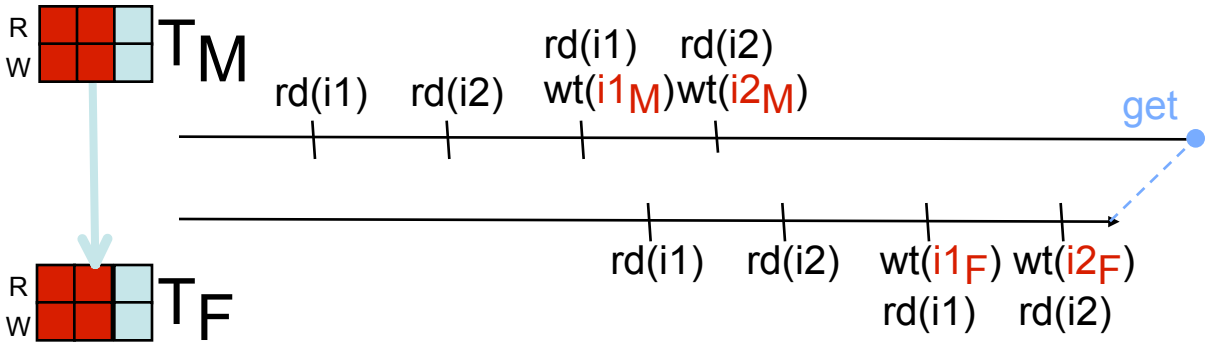


Safe Futures


```
Future f = F[foo(i1, i2)];
boolean test =  bar(i1, i2);
if (test) global.add(f.get() );
```

$i1_F = 4$ $i2_F = 4$
$i1_M = 10$ $i2_M = 10$
$i1 = 5$ $i2 = 5$ global = 0

```
static INT foo (INT i1, INT i2) {
    INT res = i1.sum(i2); → 10
    i1.sub(1); i2.sub(1);
    return res; }

static boolean bar (INT i1, INT i2) {
    boolean res = i1.eq(i2); → true
    i1.mul(2); i2.mul(2);
    return res; }
```



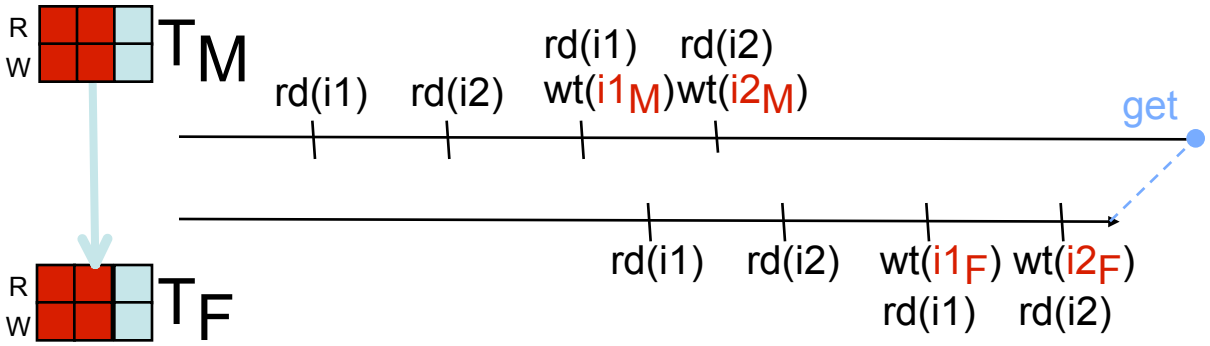
Safe Futures


```
Future f = F[foo(i1, i2)];
boolean test =  bar(i1, i2);
if (test) global.add(f.get() );
```

```
static INT foo (INT i1, INT i2) {
    INT res = i1.sum(i2); → 10
    i1.sub(1); i2.sub(1);
    return res; }

static boolean bar (INT i1, INT i2) {
    boolean res = i1.eq(i2); → true
    i1.mul(2); i2.mul(2);
    return res; }
```

$i1_M = 10$ $i2_M = 10$

$i1_F = 4$ $i2_F = 4$ global = 0



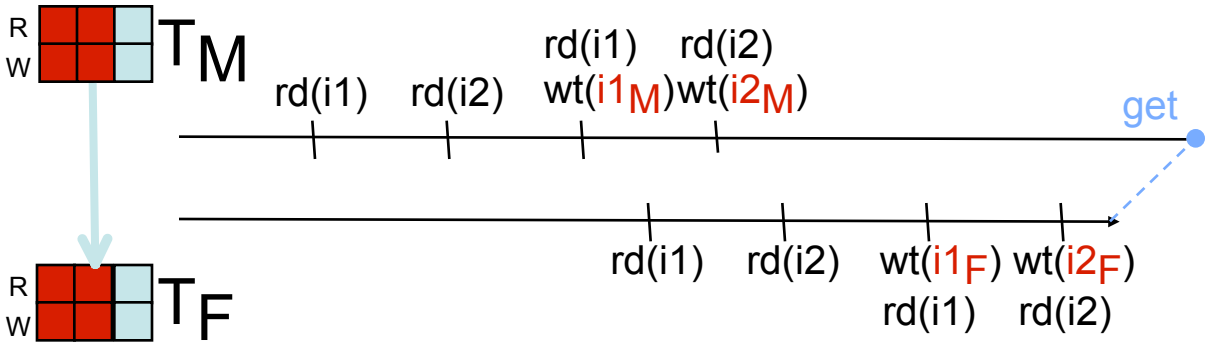
Safe Futures


```
Future f = F[foo(i1, i2)];
boolean test =  bar(i1, i2);
if (test) global.add(f.get() );
```

```
static INT foo (INT i1, INT i2) {
    INT res = i1.sum(i2); → 10
    i1.sub(1); i2.sub(1);
    return res; }

static boolean bar (INT i1, INT i2) {
    boolean res = i1.eq(i2); → true
    i1.mul(2); i2.mul(2);
    return res; }
```

$i1_M = 10$ $i2_M = 10$

$i1 = 4$ $i2 = 4$ global = 0



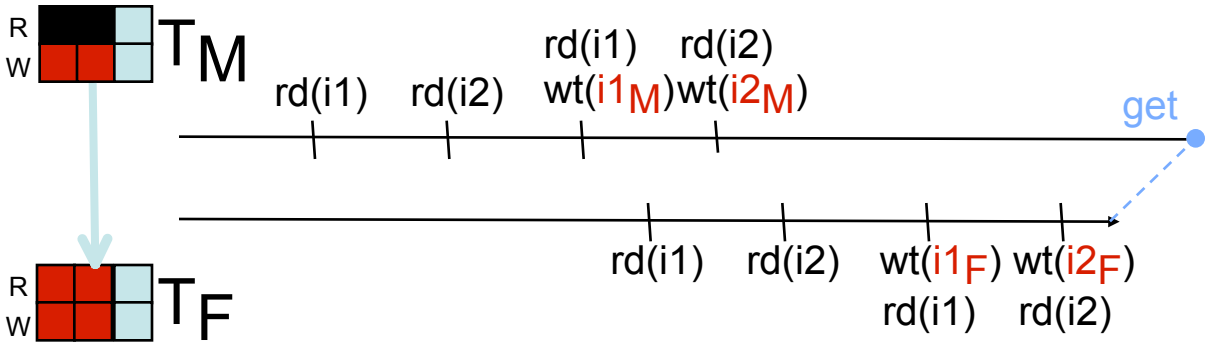
Safe Futures


```
Future f = F[foo(i1, i2)];
boolean test =  bar(i1, i2);
if (test) global.add(f.get() );
```

```
static INT foo (INT i1, INT i2) {
    INT res = i1.sum(i2); → 10
    i1.sub(1); i2.sub(1);
    return res; }

static boolean bar (INT i1, INT i2) {
    boolean res = i1.eq(i2); → true
    i1.mul(2); i2.mul(2);
    return res; }
```

$i1_M = 10$ $i2_M = 10$

$i1 = 4$ $i2 = 4$ global = 0



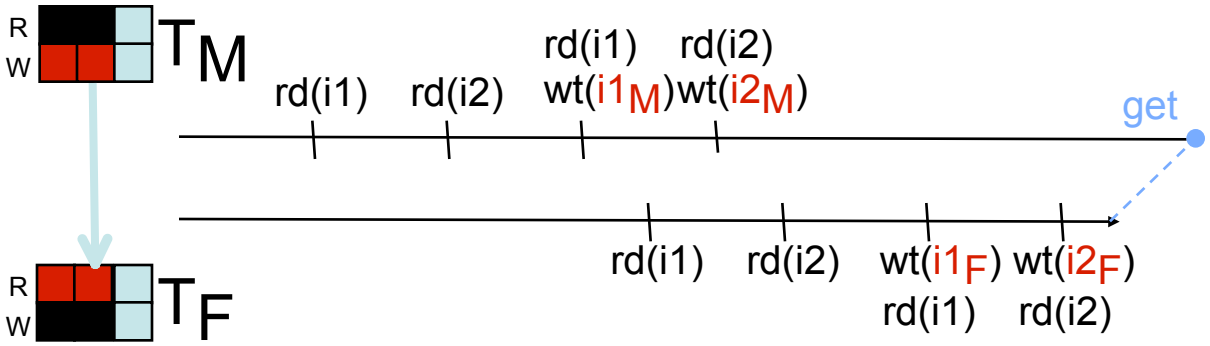
Safe Futures


```
Future f = F[foo(i1, i2)];
boolean test =  bar(i1, i2);
if (test) global.add(f.get() );
```

```
static INT foo (INT i1, INT i2) {
    INT res = i1.sum(i2); → 10
    i1.sub(1); i2.sub(1);
    return res; }

static boolean bar (INT i1, INT i2) {
    boolean res = i1.eq(i2); → true
    i1.mul(2); i2.mul(2);
    return res; }
```

$i1_M = 10$ $i2_M = 10$

$i1 = 4$ $i2 = 4$ global = 0

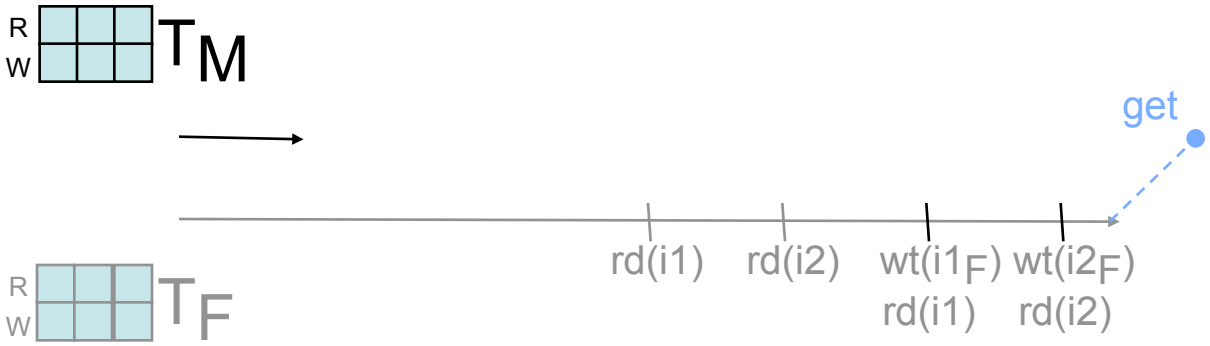


Safe Futures


```
Future f = F[foo(i1, i2)];
boolean test =  bar(i1, i2);
if (test) global.add(f.get() );
```

```
static INT foo (INT i1, INT i2) {
    INT res = i1.sum(i2); → 10
    i1.sub(1); i2.sub(1);
    return res; }

static boolean bar (INT i1, INT i2) {
    boolean res = i1.eq(i2);
    i1.mul(2); i2.mul(2);
    return res; }
```

i1 = 4 i2 = 4 global = 0

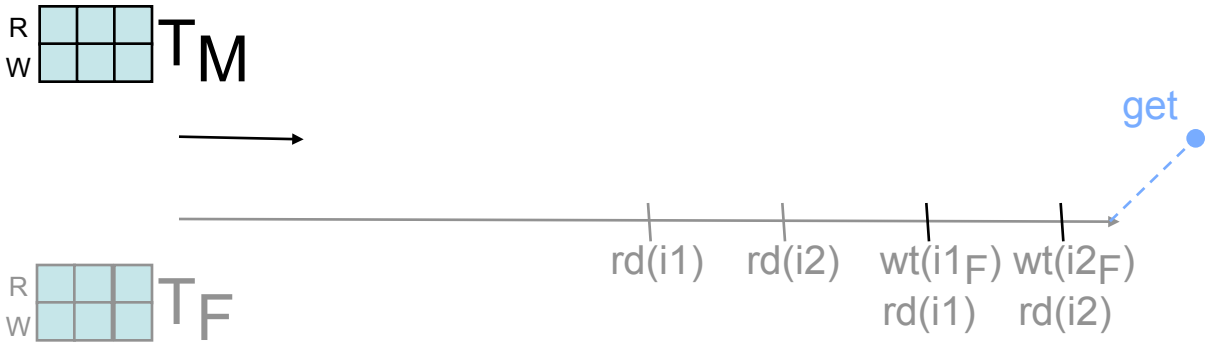


Safe Futures


```
Future f = F[foo(i1, i2)];
boolean test =  bar(i1, i2);
if (test) global.add(f.get() );
```

```
static INT foo (INT i1, INT i2) {
    INT res = i1.sum(i2); → 10
    i1.sub(1); i2.sub(1);
    return res; }

static boolean bar (INT i1, INT i2) {
    boolean res = i1.eq(i2);
    i1.mul(2); i2.mul(2);
    return res; }
```

i1 = 4 i2 = 4 global = 0

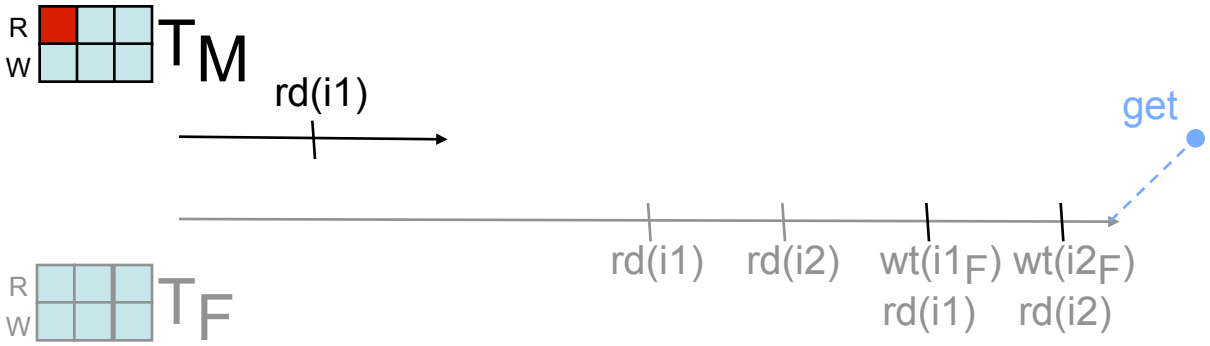


Safe Futures


```
Future f = F[foo(i1, i2)];
boolean test =  bar(i1, i2);
if (test) global.add(f.get() );
```

```
static INT foo (INT i1, INT i2) {
    INT res = i1.sum(i2); → 10
    i1.sub(1); i2.sub(1);
    return res; }

static boolean bar (INT i1, INT i2) {
    boolean res = i1.eq(i2);
    i1.mul(2); i2.mul(2);
    return res; }
```

i1 = 4 i2 = 4 global = 0

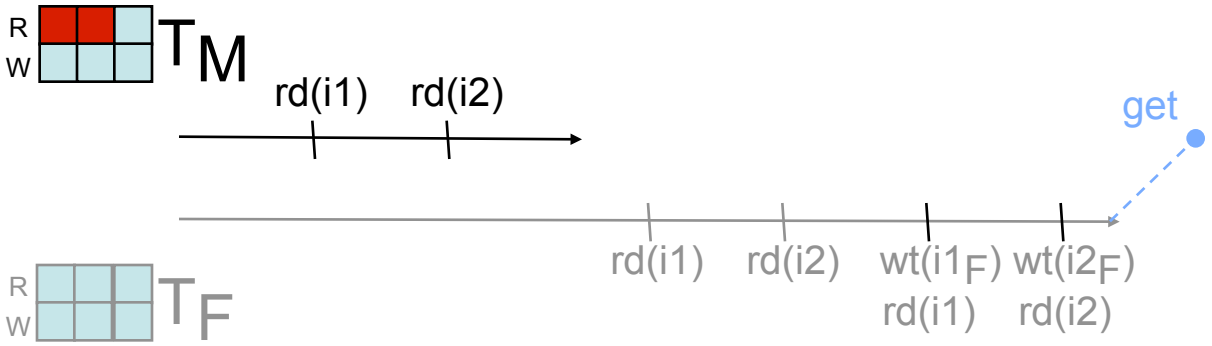


Safe Futures


```
Future f = F[foo(i1, i2)];
boolean test =  bar(i1, i2);
if (test) global.add(f.get() );
```

```
static INT foo (INT i1, INT i2) {
    INT res = i1.sum(i2); → 10
    i1.sub(1); i2.sub(1);
    return res; }

static boolean bar (INT i1, INT i2) {
    boolean res = i1.eq(i2);
    i1.mul(2); i2.mul(2);
    return res; }
```

i1 = 4 i2 = 4 global = 0

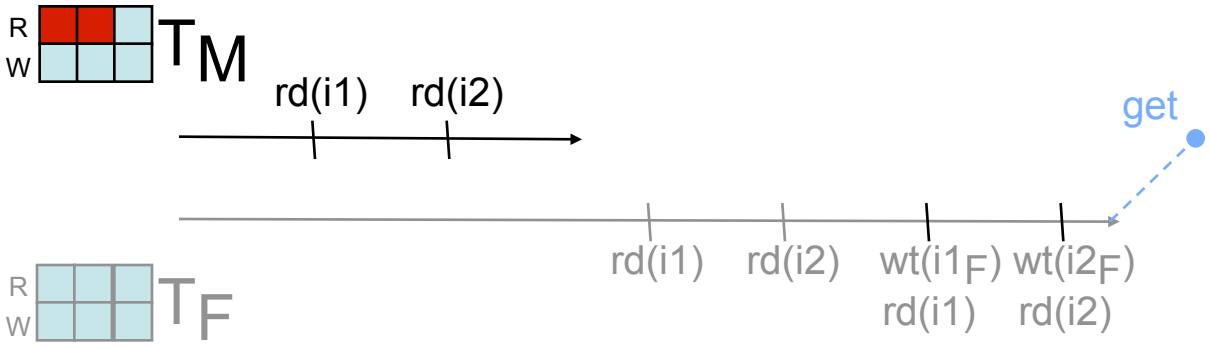


Safe Futures


```
Future f = F[foo(i1, i2)];
boolean test =  bar(i1, i2);
if (test) global.add(f.get() );
```

```
static INT foo (INT i1, INT i2) {
    INT res = i1.sum(i2); → 10
    i1.sub(1); i2.sub(1);
    return res; }

static boolean bar (INT i1, INT i2) {
    boolean res = i1.eq(i2); → true
    i1.mul(2); i2.mul(2);
    return res; }
```

i1 = 4 i2 = 4 global = 0



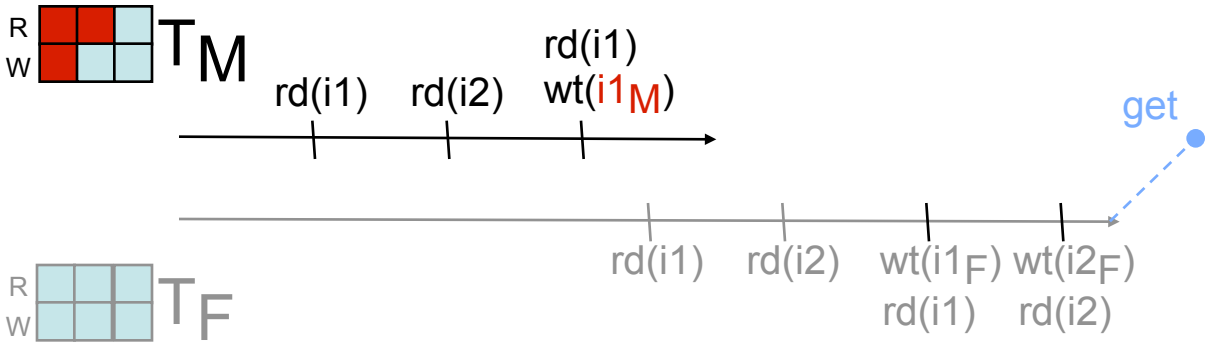
Safe Futures


```
Future f = F[foo(i1, i2)];
boolean test =  bar(i1, i2);
if (test) global.add(f.get() );
```

```
static INT foo (INT i1, INT i2) {
    INT res = i1.sum(i2); → 10
    i1.sub(1); i2.sub(1);
    return res; }

static boolean bar (INT i1, INT i2) {
    boolean res = i1.eq(i2); → true
    i1.mul(2); i2.mul(2);
    return res; }
```

$i1_M = 8$

i1 = 4 i2 = 4 global = 0



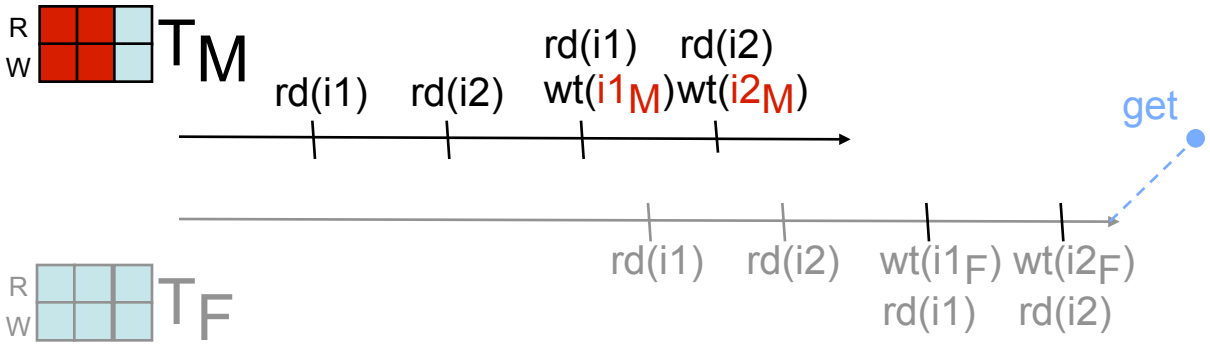
Safe Futures


```
Future f = F[foo(i1, i2)];
boolean test =  bar(i1, i2);
if (test) global.add(f.get() );
```

```
static INT foo (INT i1, INT i2) {
    INT res = i1.sum(i2); → 10
    i1.sub(1); i2.sub(1);
    return res; }

static boolean bar (INT i1, INT i2) {
    boolean res = i1.eq(i2); → true
    i1.mul(2); i2.mul(2);
    return res; }
```

$i1_M = 8$ $i2_M = 8$

$i1 = 4$ $i2 = 4$ $global = 0$



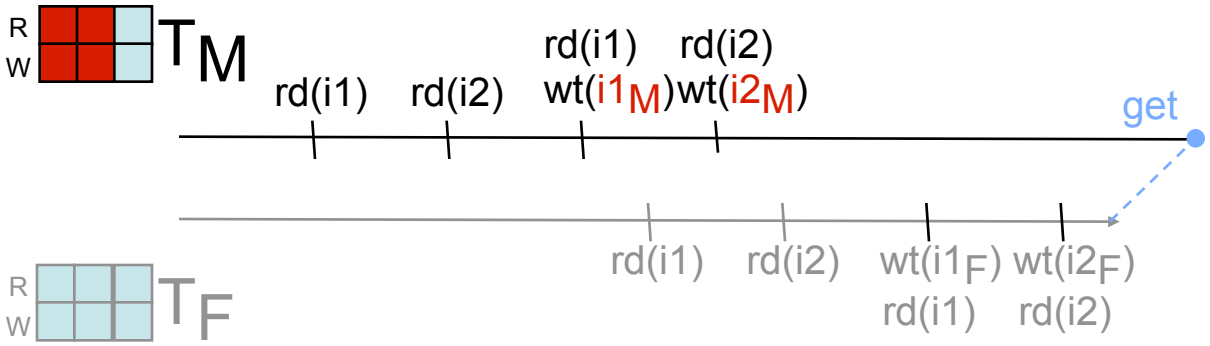
Safe Futures


```
Future f = F[foo(i1, i2)];
boolean test =  bar(i1, i2);
if (test) global.add(f.get() );
```

```
static INT foo (INT i1, INT i2) {
    INT res = i1.sum(i2); → 10
    i1.sub(1); i2.sub(1);
    return res; }

static boolean bar (INT i1, INT i2) {
    boolean res = i1.eq(i2); → true
    i1.mul(2); i2.mul(2);
    return res; }
```

$i1_M = 8$ $i2_M = 8$

$i1 = 4$ $i2 = 4$ $global = 0$

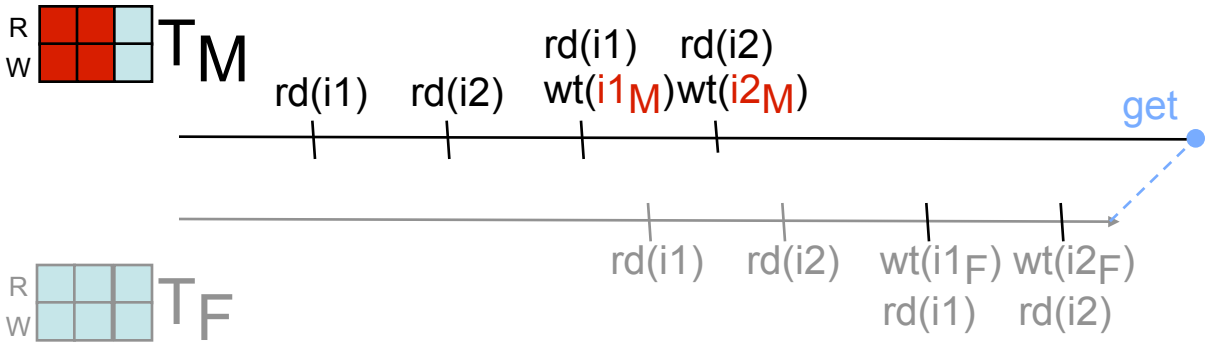


Safe Futures


```
Future f = F[foo(i1, i2)];
boolean test =  bar(i1, i2);
if (test) global.add(f.get() );
```

```
static INT foo (INT i1, INT i2) {
    INT res = i1.sum(i2); → 10
    i1.sub(1); i2.sub(1);
    return res; }

static boolean bar (INT i1, INT i2) {
    boolean res = i1.eq(i2); → true
    i1.mul(2); i2.mul(2);
    return res; }
```

$i1_M = 8$ $i2_M = 8$ global = 0

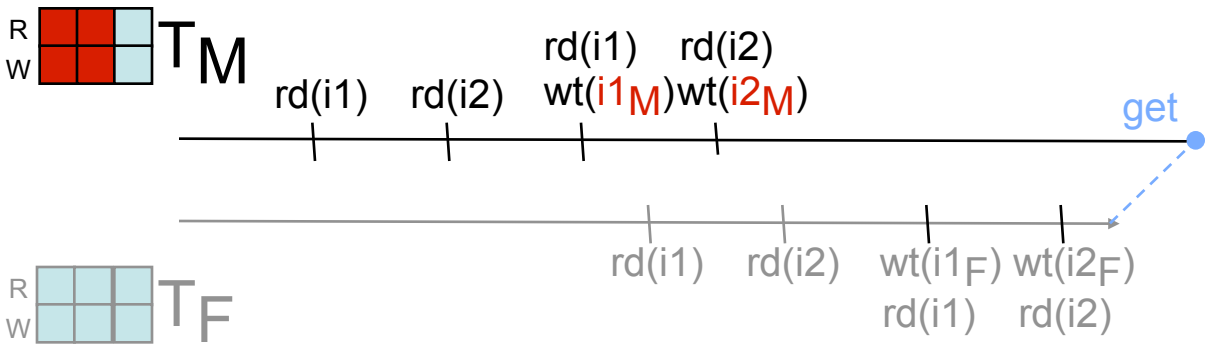


Safe Futures


```
Future f = F[foo(i1, i2)];
boolean test =  bar(i1, i2);
if (test) global.add(f.get() );
```

```
static INT foo (INT i1, INT i2) {
    INT res = i1.sum(i2); → 10
    i1.sub(1); i2.sub(1);
    return res; }

static boolean bar (INT i1, INT i2) {
    boolean res = i1.eq(i2); → true
    i1.mul(2); i2.mul(2);
    return res; }
```

i1 = 8 i2 = 8 global = 0

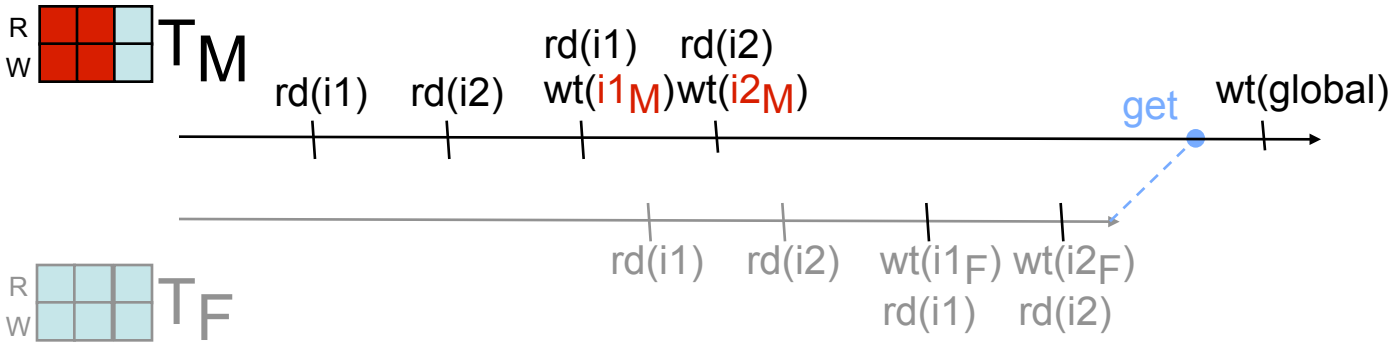


Safe Futures


```
Future f = F[foo(i1, i2)];
boolean test =  bar(i1, i2);
if (test) global.add(f.get() );
```

```
static INT foo (INT i1, INT i2) {
    INT res = i1.sum(i2); → 10
    i1.sub(1); i2.sub(1);
    return res; }

static boolean bar (INT i1, INT i2) {
    boolean res = i1.eq(i2); → true
    i1.mul(2); i2.mul(2);
    return res; }
```

i1 = 8 i2 = 8 global = 10



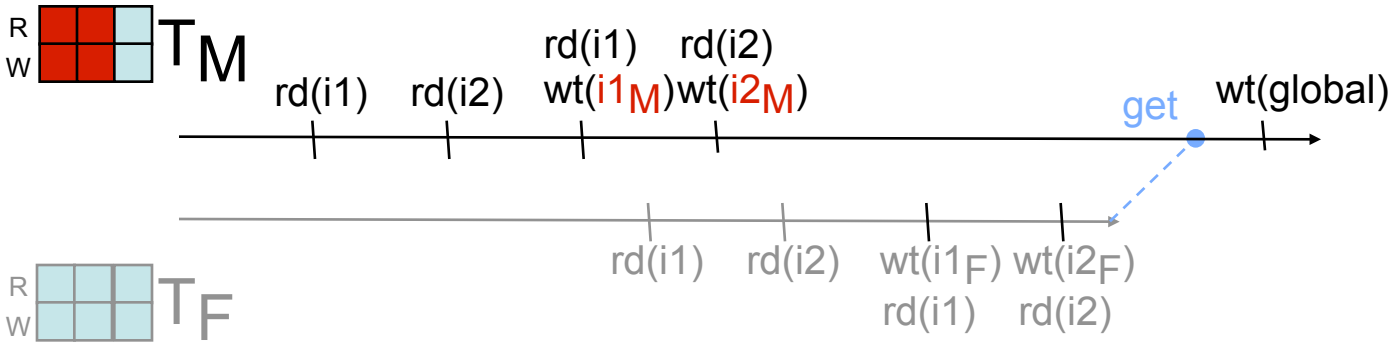
Safe Futures


```
Future f = F[foo(i1, i2)];
boolean test =  bar(i1, i2);
if (test) global.add(f.get() );
```

```
static INT foo (INT i1, INT i2) {
    INT res = i1.sum(i2); → 10
    i1.sub(1); i2.sub(1);
    return res; }

static boolean bar (INT i1, INT i2) {
    boolean res = i1.eq(i2); → true
    i1.mul(2); i2.mul(2);
    return res; }
```

i1 = 8 i2 = 8 global = 10

i1 = 8 i2 = 8 global = 10



Roll-back: Optimistic Transactions

Discard versions

Futures:

- evaluated within separate thread so just re-run

Continuations:

- Rewrite bytecodes to save state at start
- On roll-back throw **revoke** exception
- Modify run-time to unwind **revoke** exceptions without running user handlers
- Handler restores state and restarts continuation