Practical Concurrent and Parallel Programming XIII
Atomicity ?

Raúl Pardo and
Jørgen Staunstrup

Agenda

- **Atomicity ?**
  - Git
  - Optimistic concurrency control
    - Operational transform
  - Consistency
  - Atomicity in real life
- Work-stealing queues
- Examination

Errors

Errors

Some strategies

Errors

Some strategies

Avoid them

Errors

Some strategies

Avoid them

Fix them

Errors

Some strategies

Avoid them

Fix them

In Danish "pyt" (Live with them)

Errors

Some strategies

Avoid them   Atomicity (synchronized)

Fix them

In Danish "pyt" (Live with them)

Errors

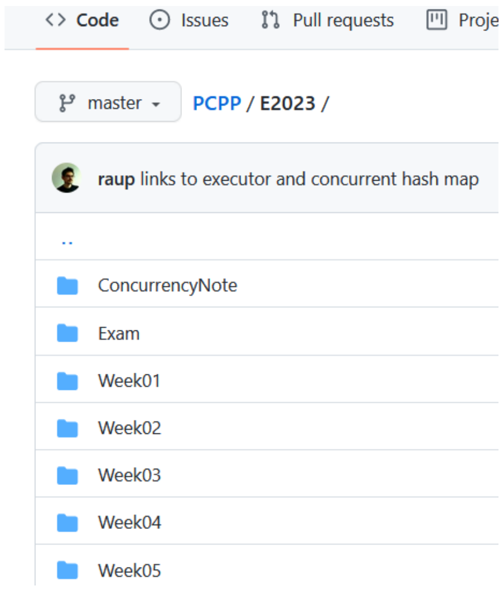
Some strategies

Avoid them — Atomicity (synchronized)

Fix them ← This week

In Danish "pyt" (Live with them)

File sharing with Git



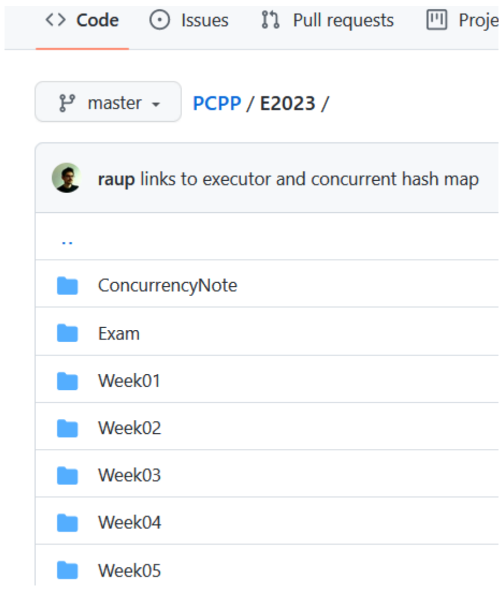
PCPP / E2023 /

raup links to executor and concurrent hash map

..
ConcurrencyNote
Exam
Week01
Week02
Week03
Week04
Week05

File sharing with Git

Potential race condition ?

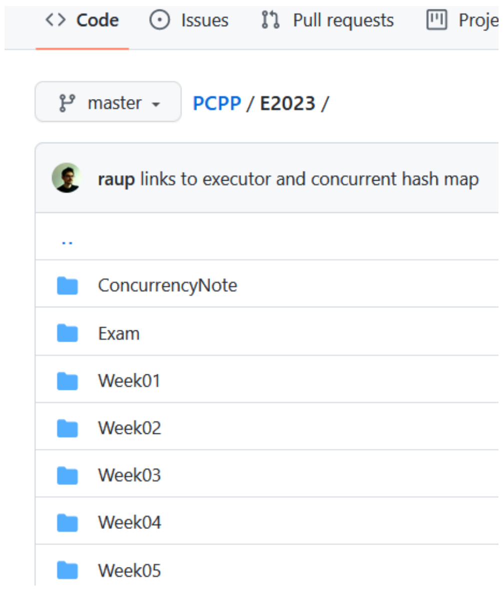


PCPP / E2023 /

raup links to executor and concurrent hash map

..
ConcurrencyNote
Exam
Week01
Week02
Week03
Week04
Week05

File sharing with Git



PCPP / E2023 /

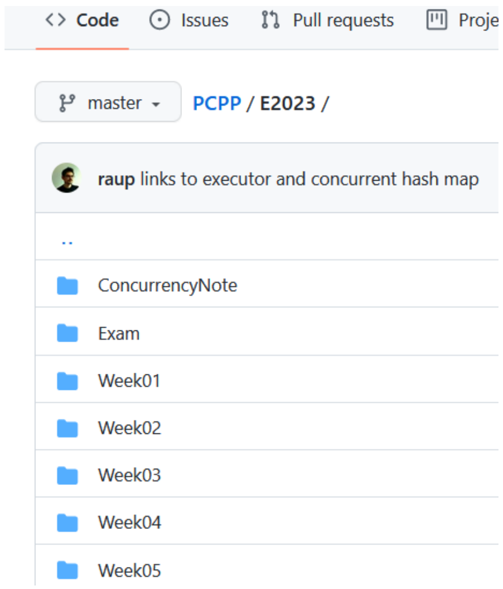
raup links to executor and concurrent hash map

..
ConcurrencyNote
Exam
Week01
Week02
Week03
Week04
Week05

Potential race condition ?

Workflow:

File sharing with Git



PCPP / E2023 /

raup links to executor and concurrent hash map

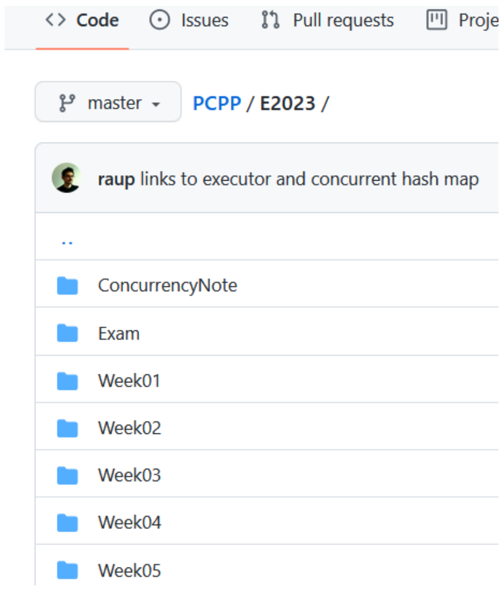
..
ConcurrencyNote
Exam
Week01
Week02
Week03
Week04
Week05

Potential race condition ?

Workflow:

```
git pull % modifications from collaborator
```

File sharing with Git



PCPP / E2023 /

raup links to executor and concurrent hash map

..
ConcurrencyNote
Exam
Week01
Week02
Week03
Week04
Week05

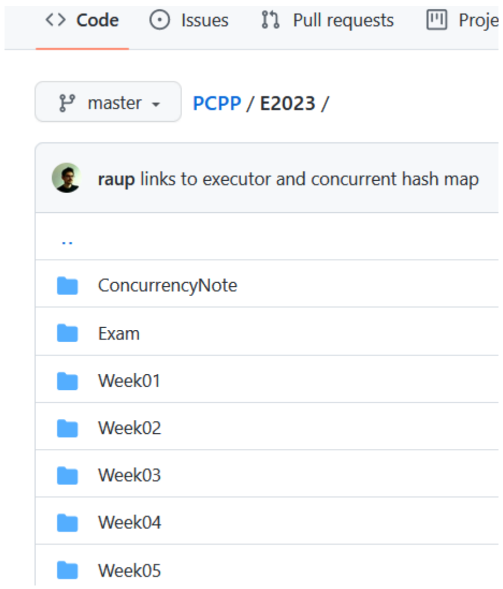
Potential race condition ?

Workflow:

```
git pull % modifications from collaborator
git stage -A
git commit ...
git push
```

File sharing with Git

Potential race condition ?



PCPP / E2023 /

raup links to executor and concurrent hash map

..
ConcurrencyNote
Exam
Week01
Week02
Week03
Week04
Week05

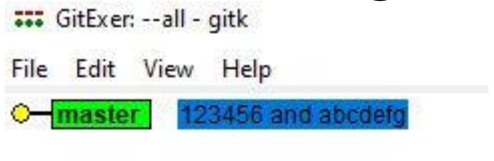
Workflow:

```
git pull % modifications from collaborator
git stage -A
git commit ...
git push
```

Works because Raúl and I modify different files !!

Git merge / rebase (1)

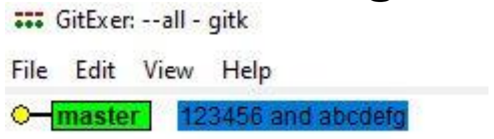
file abc.txt: abcdefg and file numbers.txt: 123456



123456 and abcdefg

Git merge / rebase (1)

file abc.txt: abcdefg and file numbers.txt: 123456

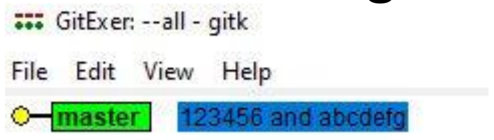


123456 and abcdefg

```
git branch newnumbers
```

Git merge / rebase (1)

file abc.txt: abcdefg and file numbers.txt: 123456

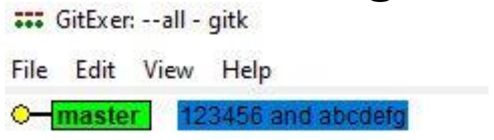


123456 and abcdefg

```
git branch newnumbers
git checkout newnumbers
```

Git merge / rebase (1)

file abc.txt: abcdefg and file numbers.txt: 123456

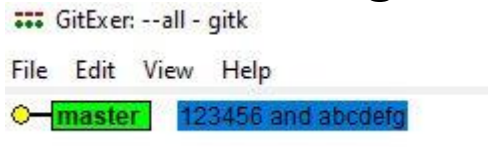


```
git branch newnumbers
git checkout newnumbers
```

change file numbers.txt: 1234

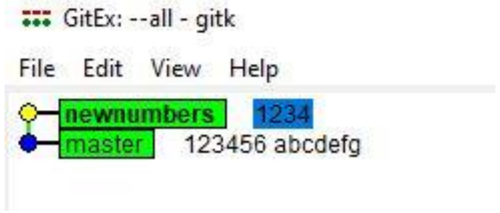
Git merge / rebase (1)

file abc.txt: abcdefg and file numbers.txt: 123456



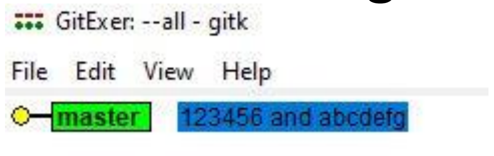
```
git branch newnumbers
git checkout newnumbers
```

change file numbers.txt: 1234



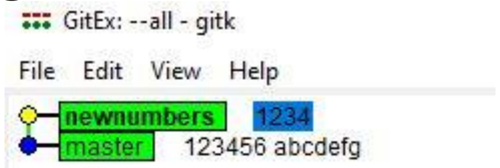
Git merge / rebase (1)

file abc.txt: abcdefg and file numbers.txt: 123456



```
git branch newnumbers
git checkout newnumbers
```

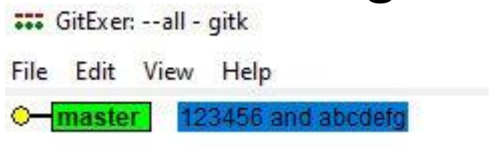
change file numbers.txt: 1234



```
git checkout master
git merge newnumbers
```

Git merge / rebase (1)

file abc.txt: abcdefg and file numbers.txt: 123456

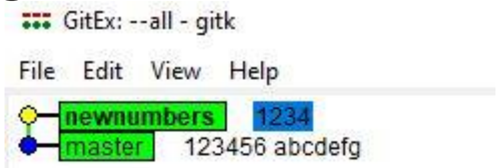


GitExer: --all - gitk

123456 and abcdefg

```
git branch newnumbers
git checkout newnumbers
```

change file numbers.txt: 1234



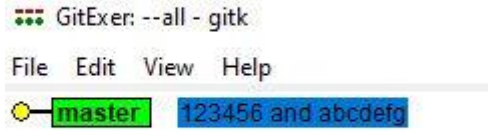
GitEx: --all - gitk

1234
123456 abcdefg

```
git checkout master
git merge newnumbers
Updating dd2289c..a423cf8
  Fast-forward
   numbers.txt | 2 +-
   1 file changed, 1 insertion(+), 1 deletion(-)
```

Git merge / rebase (2)

file abc.txt: abcdefg and file numbers.txt: 123456


GitExer: --all - gitk

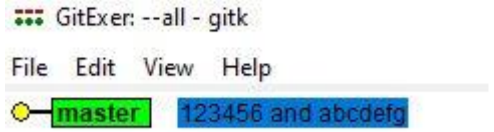
123456 and abcdefg

```
git branch newnumbers
git checkout newnumbers
```

change file numbers.txt: 12xy4q

Git merge / rebase (2)

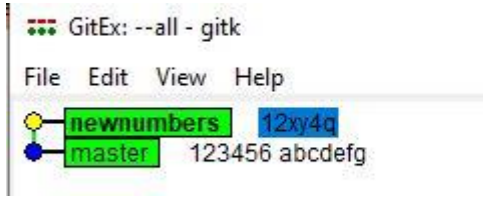
file abc.txt: abcdefg and file numbers.txt: 123456



123456 and abcdefg

```
git branch newnumbers
git checkout newnumbers
```

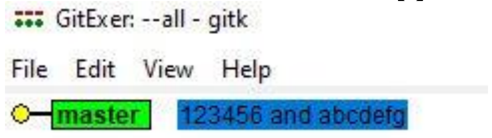
change file numbers.txt: 12xy4q



12xy4q
123456 abcdefg

Git merge / rebase (2)

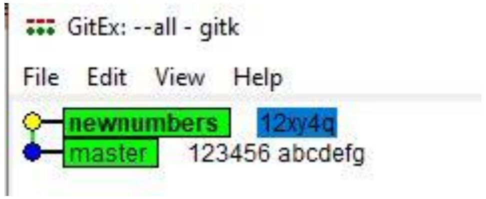
file abc.txt: abcdefg and file numbers.txt: 123456



master 123456 and abcdefg

```
git branch newnumbers
git checkout newnumbers
```

change file numbers.txt: 12xy4q

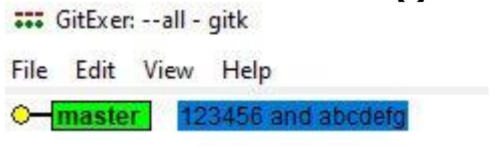


newnumbers 12xy4q
master 123456 abcdefg

```
git checkout master
git merge newnumbers
```

Git merge / rebase (2)

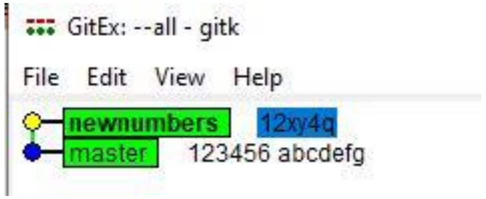
file abc.txt: abcdefg and file numbers.txt: 123456



123456 and abcdefg

```
git branch newnumbers
git checkout newnumbers
```

change file numbers.txt: 12xy4q

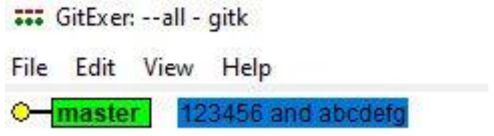


12xy4q
123456 abcdefg

```
git checkout master
git merge newnumbers
Auto-merging numbers.txt
CONFLICT (content): Merge conflict in numbers.txt
Automatic merge failed; fix conflicts and then commit the
result.
```

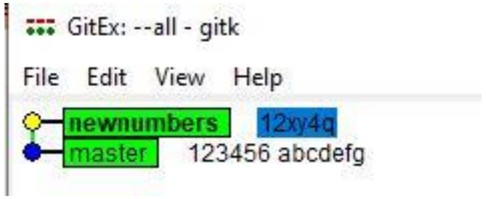
Git merge / rebase (2)

file abc.txt: abcdefg and file numbers.txt: 123456



```
git branch newnumbers
git checkout newnumbers
```

change file numbers.txt: 12xy4q



Manual fix
of data race

```
git checkout master
git merge newnumbers
Auto-merging numbers.txt
CONFLICT (content): Merge conflict in numbers.txt
Automatic merge failed; fix conflicts and then commit the
result.
```

Agenda

- **Atomicity ?**
  - Git
  - **Optimistic concurrency control**
    - Operational transform
  - Consistency
  - Atomicity in real life
- Work-stealing queues
- Examination

Concurrency control

Pessimistic:

```
public void synchronized modify(Something s){
...
}
```

Concurrency control

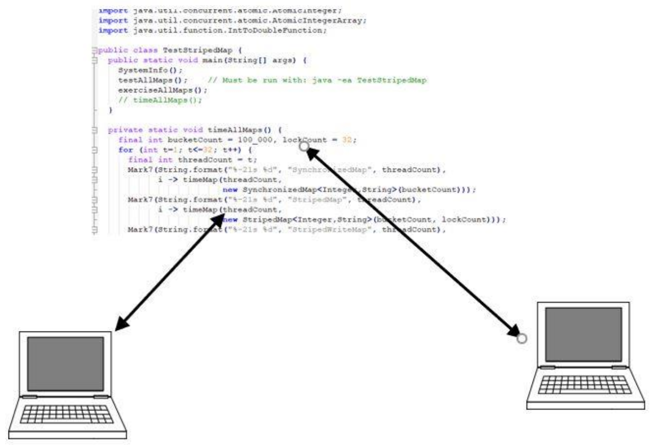
Pessimistic:

```
public void synchronized modify(Something s){
...
}
```

Optimistic: Discover and fix data races at runtime

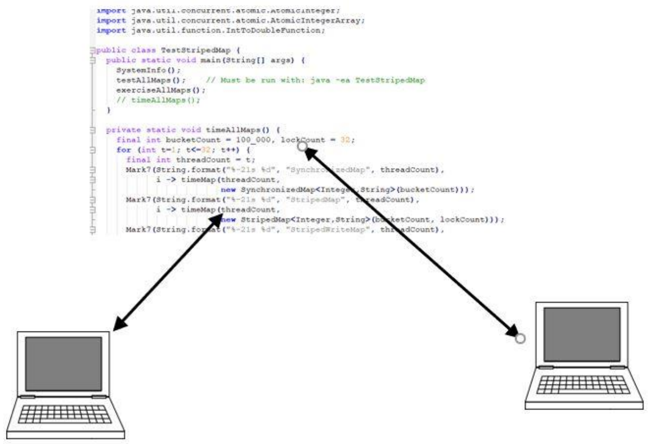
```
public void modify(Something s){
...
}
```

Concurrent text editing



```
import java.util.concurrent.atomic.AtomicInteger;
import java.util.concurrent.atomic.AtomicIntegerArray;
import java.util.function.IntToDoubleFunction;

public class TestStripedMap {
  public static void main(String[] args) {
    SystemInfo();
    testAllMaps();    // Must be run with: java -ea TestStripedMap
    exerciseAllMaps();
    // timeAllMaps();
  }

  private static void timeAllMaps() {
    final int bucketCount = 100_000, lockCount = 32;
    for (int t=1; t<=32; t++) {
      final int threadCount = t;
      Mark7(String.format("%-21s %d", "SynchronizedMap", threadCount),
            i -> timeMap(threadCount,
                         new SynchronizedMap<Integer,String>(bucketCount)));
      Mark7(String.format("%-21s %d", "StripedMap", threadCount),
            i -> timeMap(threadCount,
                         new StripedMap<Integer,String>(bucketCount, lockCount)));
      Mark7(String.format("%-21s %d", "StripedWriteMap", threadCount),
```

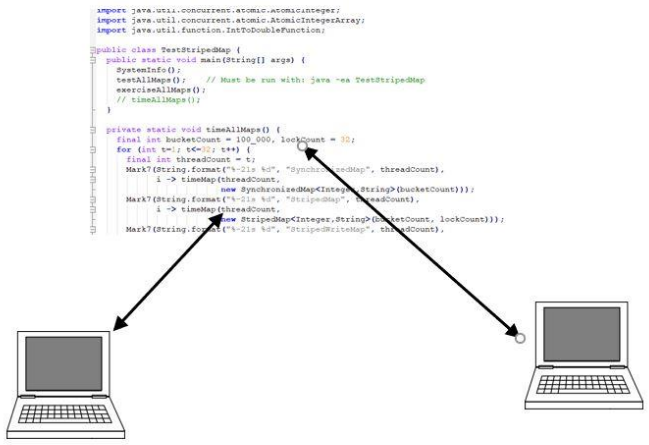
Concurrent text editing



```
import java.util.concurrent.atomic.AtomicInteger;
import java.util.concurrent.atomic.AtomicIntegerArray;
import java.util.function.IntToDoubleFunction;

public class TestStripedMap {
  public static void main(String[] args) {
    SystemInfo();
    testAllMaps();    // Must be run with: java -ea TestStripedMap
    exerciseAllMaps();
    // timeAllMaps();
  }

  private static void timeAllMaps() {
    final int bucketCount = 100_000, lockCount = 32;
    for (int t=1; t<=32; t++) {
      final int threadCount = t;
      Mark7(String.format("%-21s %d", "SynchronizedMap", threadCount),
            i -> timeMap(threadCount,
                         new SynchronizedMap<Integer,String>(bucketCount)));
      Mark7(String.format("%-21s %d", "StripedMap", threadCount),
            i -> timeMap(threadCount,
                         new StripedMap<Integer,String>(bucketCount, lockCount)));
      Mark7(String.format("%-21s %d", "StripedWriteMap", threadCount),
```

Discover and fix
data races at runtime

Concurrent text editing

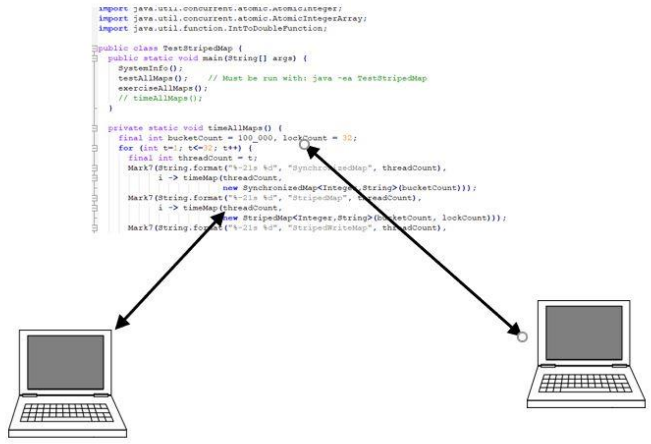


```
import java.util.concurrent.atomic.AtomicInteger;
import java.util.concurrent.atomic.AtomicIntegerArray;
import java.util.function.IntToDoubleFunction;

public class TestStripedMap {
  public static void main(String[] args) {
    SystemInfo();
    testAllMaps();    // Must be run with: java -ea TestStripedMap
    exerciseAllMaps();
    // timeAllMaps();
  }

  private static void timeAllMaps() {
    final int bucketCount = 100_000, lockCount = 32;
    for (int t=1; t<=32; t++) {
      final int threadCount = t;
      Mark7(String.format("%-21s %d", "SynchronizedMap", threadCount),
            i -> timeMap(threadCount,
                         new SynchronizedMap<Integer,String>(bucketCount)));
      Mark7(String.format("%-21s %d", "StripedMap", threadCount),
            i -> timeMap(threadCount,
                         new StripedMap<Integer,String>(bucketCount, lockCount)));
      Mark7(String.format("%-21s %d", "StripedWriteMap", threadCount),
```

Discover and fix
data races at runtime

Google wave https://youtu.be/p6pgxLaDdQw

Concurrent text editing

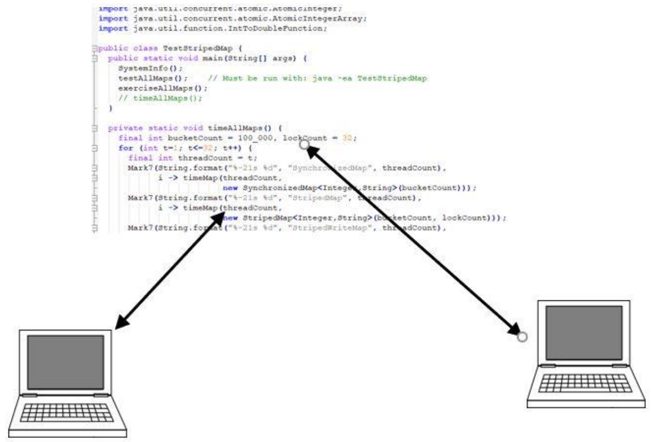


```
import java.util.concurrent.atomic.AtomicInteger;
import java.util.concurrent.atomic.AtomicIntegerArray;
import java.util.function.IntToDoubleFunction;

public class TestStripedMap {
  public static void main(String[] args) {
    SystemInfo();
    testAllMaps();    // Must be run with: java -ea TestStripedMap
    exerciseAllMaps();
    // timeAllMaps();
  }

  private static void timeAllMaps() {
    final int bucketCount = 100_000, lockCount = 32;
    for (int t=1; t<=32; t++) {
      final int threadCount = t;
      Mark7(String.format("%-21s %d", "SynchronizedMap", threadCount),
            i -> timeMap(threadCount,
                         new SynchronizedMap<Integer,String>(bucketCount)));
      Mark7(String.format("%-21s %d", "StripedMap", threadCount),
            i -> timeMap(threadCount,
                         new StripedMap<Integer,String>(bucketCount, lockCount)));
      Mark7(String.format("%-21s %d", "StripedWriteMap", threadCount),
```

Discover and fix
data races at runtime

Google wave https://youtu.be/p6pgxLaDdQw

Concurrent text editing



```
import java.util.concurrent.atomic.AtomicInteger;
import java.util.concurrent.atomic.AtomicIntegerArray;
import java.util.function.IntToDoubleFunction;

public class TestStripedMap {
  public static void main(String[] args) {
    SystemInfo();
    testAllMaps();    // Must be run with: java -ea TestStripedMap
    exerciseAllMaps();
    // timeAllMaps();
  }

  private static void timeAllMaps() {
    final int bucketCount = 100_000, lockCount = 32;
    for (int t=1; t<=32; t++) {
      final int threadCount = t;
      Mark7(String.format("%-21s %d", "SynchronizedMap", threadCount),
            i -> timeMap(threadCount,
                         new SynchronizedMap<Integer,String>(bucketCount)));
      Mark7(String.format("%-21s %d", "StripedMap", threadCount),
            i -> timeMap(threadCount,
                         new StripedMap<Integer,String>(bucketCount, lockCount)));
      Mark7(String.format("%-21s %d", "StripedWriteMap", threadCount),
```

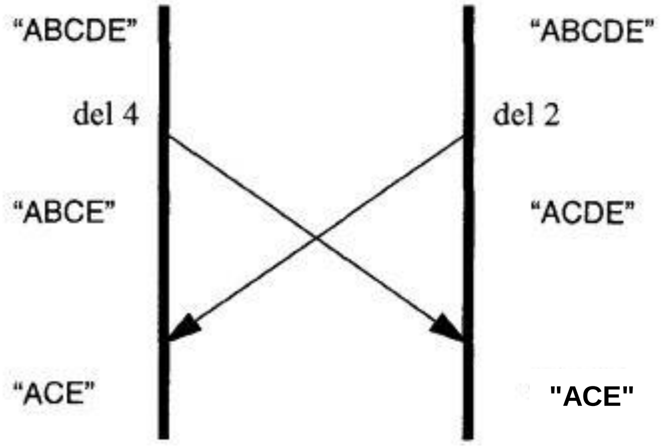
Discover and fix
data races at runtime

Google wave https://youtu.be/p6pgxLaDdQw

Concurrent editing survived in Google Docs, MS Office, …

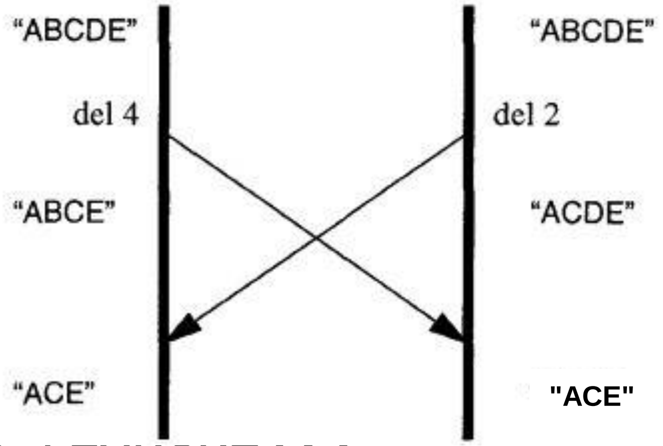
Operational transform

The key concept behind Google Wave (and many similar systems)



Operational transform

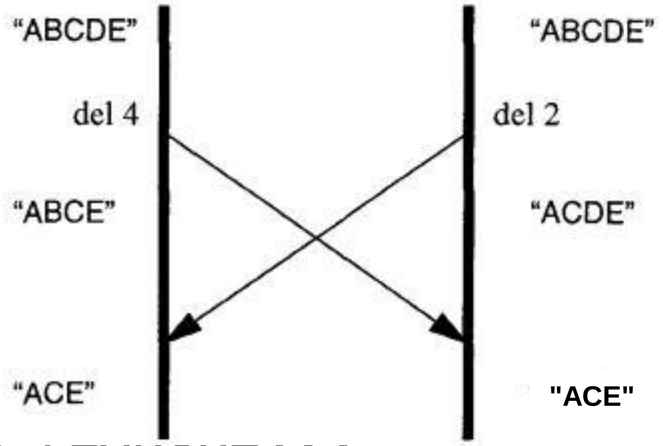
The key concept behind Google Wave (and many similar systems)



https://youtu.be/3ykZYKCK7AM

Operational transform

The key concept behind Google Wave (and many similar systems)

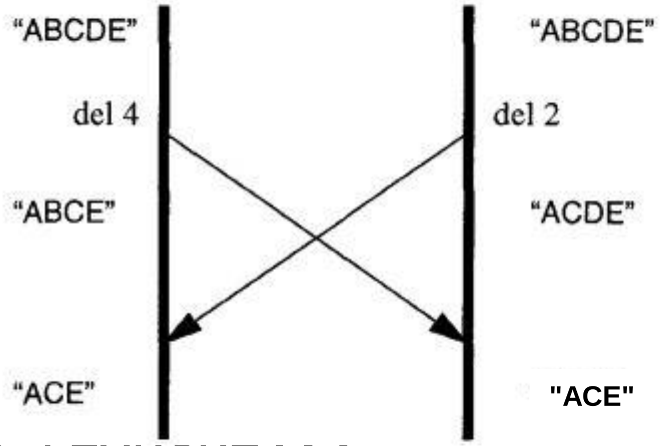


https://youtu.be/3ykZYKCK7AM

Find a way to resolve conflicts for **all** pairs of operations o1 and o2 where: o1;o2 ≠ o2;o1

Operational transform

The key concept behind Google Wave (and many similar systems)



https://youtu.be/3ykZYKCK7AM

Find a way to resolve conflicts for **all** pairs of operations o1 and o2 where: o1;o2 ≠ o2;o1

This is not so difficult for text operations like insert and delete

CAP theorem

CAP theorem

Consistency
Every read receives the most recent write or an error

CAP theorem

Consistency
Every read receives the most recent write or an error

Availability
Every request receives a (non-error) response { without guarantee that it contains the most recent write}

CAP theorem

Consistency
Every read receives the most recent write or an error

Availability
Every request receives a (non-error) response { without guarantee that it contains the most recent write}

Partition tolerance
The system continues to operate despite an arbitrary number of messages being dropped (or delayed) by the network between node

CAP theorem

Consistency
Every read receives the most recent write or an error

Availability
Every request receives a (non-error) response { without guarantee that it contains the most recent write}

Partition tolerance
The system continues to operate despite an arbitrary number of messages being dropped (or delayed) by the network between node

CAP theorem: *impossible* for a distributed data store to simultaneously provide more than two out of the three: consistency, availability and partition tolerance.

CAP theorem

Consistency
Every read receives the most recent write or an error

Availability
Every request receives a (non-error) response { without guarantee that it contains the most recent write}

Partition tolerance
The system continues to operate despite an arbitrary number of messages being dropped (or delayed) by the network between node

CAP theorem: *impossible* for a distributed data store to simultaneously provide more than two out of the three: consistency, availability and partition tolerance.

Gilbert and Nancy Lynch, "Brewer’s conjecture and the feasibility of consistent, available, partition-tolerant web services", ACM SIGACT News, Volume 33 Issue 2 (2002), pg. 51{59. https://dl.acm.org/doi/10.1145/564585.564601

CAP theorem

Consistency
Every read receives the most recent write or an error

Availability
Every request receives a (non-error) response { without guarantee that it contains the most recent write}

Partition tolerance
The system continues to operate despite an arbitrary number of messages being dropped (or delayed) by the network between node

CAP theorem

~~**Consistency**~~
~~Every read receives the most recent write or an error~~

Availability
Every request receives a (non-error) response { without guarantee that it contains the most recent write}

Partition tolerance
The system continues to operate despite an arbitrary number of messages being dropped (or delayed) by the network between node

CAP theorem

~~**Consistency**~~
~~Every read receives the most recent write or an error~~

Availability
Every request receives a (non-error) response { without guarantee that it contains the most recent write}

Partition tolerance
The system continues to operate despite an arbitrary number of messages being dropped (or delayed) by the network between node

Operational transform: accept temporary inconsistencies

Strong eventual consistency

When off-line: accept temporary inconsistencies

Strong eventual consistency

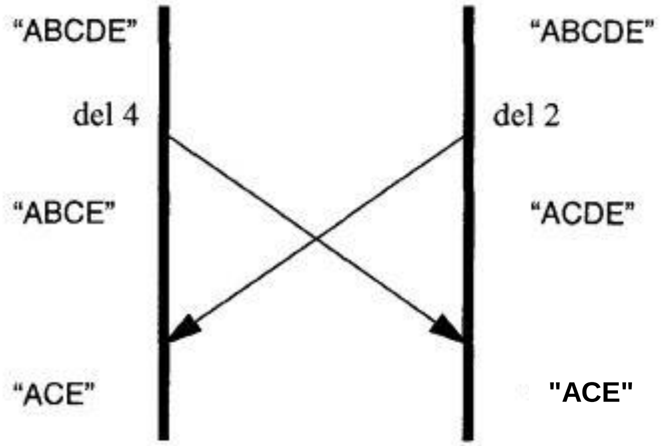
When off-line: accept temporary inconsistencies

When on-line, requests are merged (operational transform)

Strong eventual consistency

When off-line: accept temporary inconsistencies

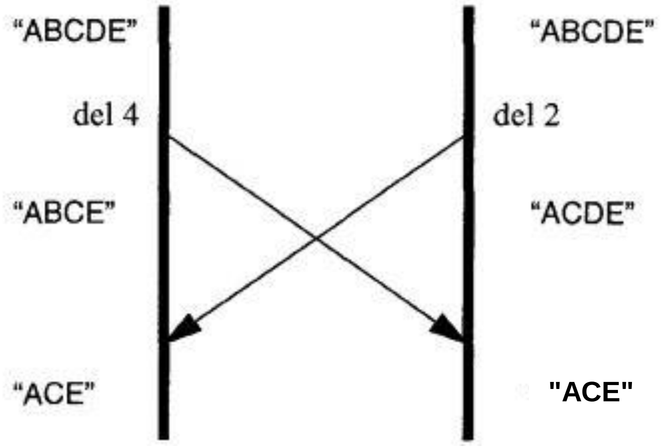
When on-line, requests are merged (operational transform)



Strong eventual consistency

When off-line: accept temporary inconsistencies

When on-line, requests are merged (operational transform)



Consistent

Operational transform (example)

Imagine a text editor where many clients can edit without locking

Operational transform (example)

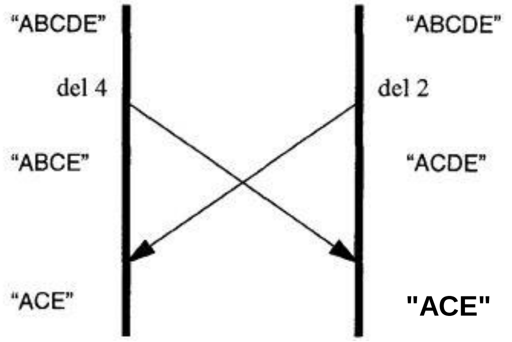
Imagine a text editor where many clients can edit without locking

The server makes an opTrans operation on conflicting operations such as: `del4 and del2`.

```
opTrans(del x, del y) =
                        {delx-1, dely}if x>y
                        {delx, dely-1)if x<y
                        {no-op, no-op} if x = y
```

Operational transform (example)

Imagine a text editor where many clients can edit without locking

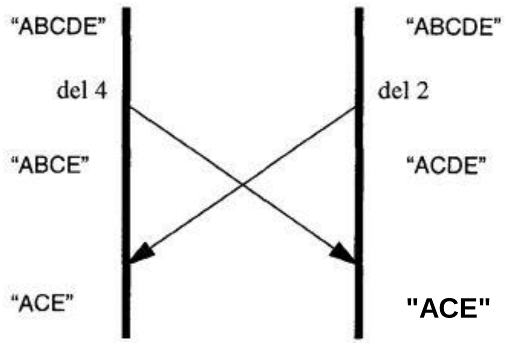


The server makes an opTrans operation on conflicting operations such as: `del4 and del2`.

```
opTrans(del x, del y) =
                        {delx-1, dely}if x>y
                        {delx, dely-1)if x<y
                        {no-op, no-op} if x = y
```

Operational transform (example)

Imagine a text editor where many clients can edit without locking

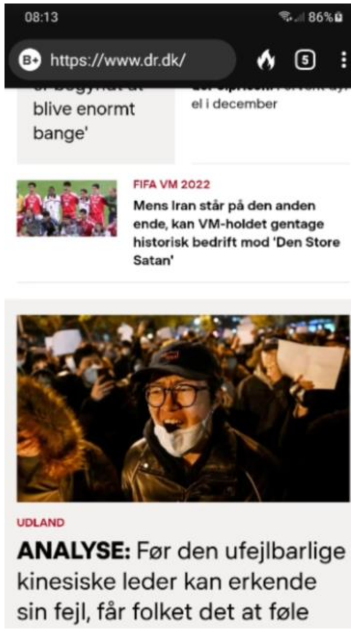


The server makes an opTrans operation on conflicting operations such as: `del4 and del2`.

```
opTrans(del x, del y) =
                        {delx-l, dely}if x>y
                        {delx, dely-l)if x<y
                        {no-op, no-op} if x = y
```

More details: *High-Latency, Low-Bandwidth Windowing in the Jupiter Collaboration System*, see Nichols.pdf

Mobile app


08:13 86%

https://www.dr.dk/

blive enormt bange'

el i december

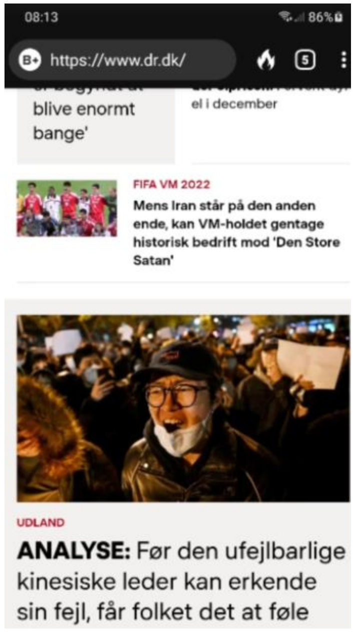
FIFA VM 2022

Mens Iran står på den anden ende, kan VM-holdet gentage historisk bedrift mod 'Den Store Satan'

UDLAND

ANALYSE: Før den ufejlbarlige kinesiske leder kan erkende sin fejl, får folket det at føle

Mobile app


08:13 86%

https://www.dr.dk/

blive enormt bange'

el i december

FIFA VM 2022

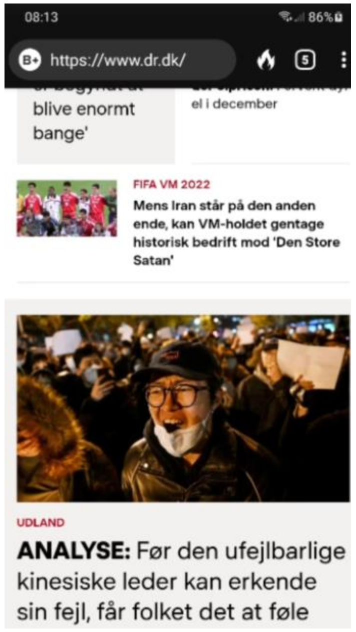
Mens Iran står på den anden ende, kan VM-holdet gentage historisk bedrift mod 'Den Store Satan'

UDLAND

ANALYSE: Før den ufejlbarlige kinesiske leder kan erkende sin fejl, får folket det at føle

- Local storage: on client device

Mobile app


08:13 86%

https://www.dr.dk/

blive enormt bange'

el i december

FIFA VM 2022

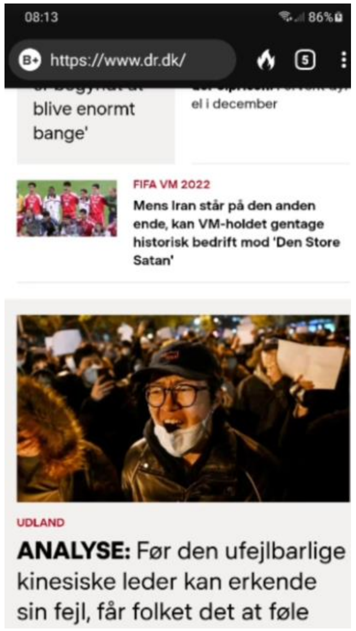
Mens Iran står på den anden ende, kan VM-holdet gentage historisk bedrift mod 'Den Store Satan'

UDLAND

ANALYSE: Før den ufejlbarlige kinesiske leder kan erkende sin fejl, får folket det at føle

- Local storage: on client device
- Network unreliable

Mobile app


08:13
86%
https://www.dr.dk/
blive enormt bange'
el i december
FIFA VM 2022
Mens Iran står på den anden ende, kan VM-holdet gentage historisk bedrift mod 'Den Store Satan'
UDLAND
ANALYSE: Før den ufejlbarlige kinesiske leder kan erkende sin fejl, får folket det at føle

- Local storage: on client device
- Network unreliable
- Reactive UI: Live objects always reflect the latest data stored

Realm: Mobile app database

Database that can be synchronized with multiple client in real-time

Realm: Mobile app database

Database that can be synchronized with multiple client in real-time

- Local storage: local copy (of relevant parts)

Realm: Mobile app database

Database that can be synchronized with multiple client in real-time

- Local storage: local copy (of relevant parts)
- Offline-first: you always read from and write to the local database

Realm: Mobile app database

Database that can be synchronized with multiple client in real-time

- Local storage: local copy (of relevant parts)
- Offline-first: you always read from and write to the local database
- Synchronizes data with central database in a background thread using operational transform

Realm: Mobile app database

Database that can be synchronized with multiple client in real-time

- Local storage: local copy (of relevant parts)
- Offline-first: you always read from and write to the local database
- Synchronizes data with central database in a background thread using operational transform
- Reactive UI: Live objects always reflect the latest data stored (on device)

Realm: Mobile app database

Database that can be synchronized with multiple client in real-time

- Local storage: local copy (of relevant parts)
- Offline-first: you always read from and write to the local database
- Synchronizes data with central database in a background thread using operational transform
- Reactive UI: Live objects always reflect the latest data stored (on device)
- Object oriented: Database stores Java objects directly

Realm: Mobile app database

Database that can be synchronized with multiple client in real-time

- Local storage: local copy (of relevant parts)
- Offline-first: you always read from and write to the local database
- Synchronizes data with central database in a background thread using operational transform
- Reactive UI: Live objects always reflect the latest data stored (on device)
- Object oriented: Database stores Java objects directly

The Realm SDK: Android, iOS, Node.js, React Native, and UWP (Windows)

Realm is now part of MongoDB

Realm: Mobile app database

Database that can be synchronized with multiple client in real-time

- Local storage: local copy (of relevant parts)
- Offline-first: you always read from and write to the local database
- Synchronizes data with central database in a background thread using operational transform
- Reactive UI: Live objects always reflect the latest data stored (on device)
- Object oriented: Database stores Java objects directly

The Realm SDK: Android, iOS, Node.js, React Native, and UWP (Windows)

Realm is now part of MongoDB

source: https://docs.mongodb.com/realm/get-started/introduction-mobile/

Realm synchronization protocol

Goal: correctly and efficiently sync data changes in real time across multiple clients that each maintain their own local Realm database.

Realm synchronization protocol

Goal: correctly and efficiently sync data changes in real time across multiple clients that each maintain their own local Realm database.

- Changeset: list of write operations to database objects

Realm synchronization protocol

Goal: correctly and efficiently sync data changes in real time across multiple clients that each maintain their own local Realm database.

- Changeset: list of write operations to database objects
- Operational transformation: operational transformation is used to resolve conflicts between changesets from different clients

Realm synchronization protocol

Goal: correctly and efficiently sync data changes in real time across multiple clients that each maintain their own local Realm database.

- Changeset: list of write operations to database objects
- Operational transformation: operational transformation is used to resolve conflicts between changesets from different clients
- Off-line first: any device may perform offline writes and upload the corresponding changesets when there is network connectivity

Realm synchronization protocol

Goal: correctly and efficiently sync data changes in real time across multiple clients that each maintain their own local Realm database.

- Changeset: list of write operations to database objects
- Operational transformation: operational transformation is used to resolve conflicts between changesets from different clients
- Off-line first: any device may perform offline writes and upload the corresponding changesets when there is network connectivity
- *Realm objects: Some restrictions on field types (to enable operational transform)*

Realm synchronization protocol

Goal: correctly and efficiently sync data changes in real time across multiple clients that each maintain their own local Realm database.

- Changeset: list of write operations to database objects
- Operational transformation: operational transformation is used to resolve conflicts between changesets from different clients
- Off-line first: any device may perform offline writes and upload the corresponding changesets when there is network connectivity
- *Realm objects: Some restrictions on field types (to enable operational transform)*

source: https://docs.mongodb.com/realm/sync/protocol/#sync-protocol

Optimistic concurrency control

```
public void modify(Something s){
...
}
```

Google Wave, Realm (MongoDB),   …

Optimistic concurrency control

```
public void modify(Something s){
...
}
```

Google Wave, Realm (MongoDB), …

Compromise on consistency: *Strong eventual consistency*

Optimistic concurrency control

```
public void modify(Something s){
...
}
```

Google Wave, Realm (MongoDB), …

Compromise on consistency: *Strong eventual consistency*
and many more

Agenda

- **Atomicity ?**
  - Git
  - Optimistic concurrency control
    - Operational transform
  - **Consistency**
  - Atomicity in real life
- Work-stealing queues
- Examination

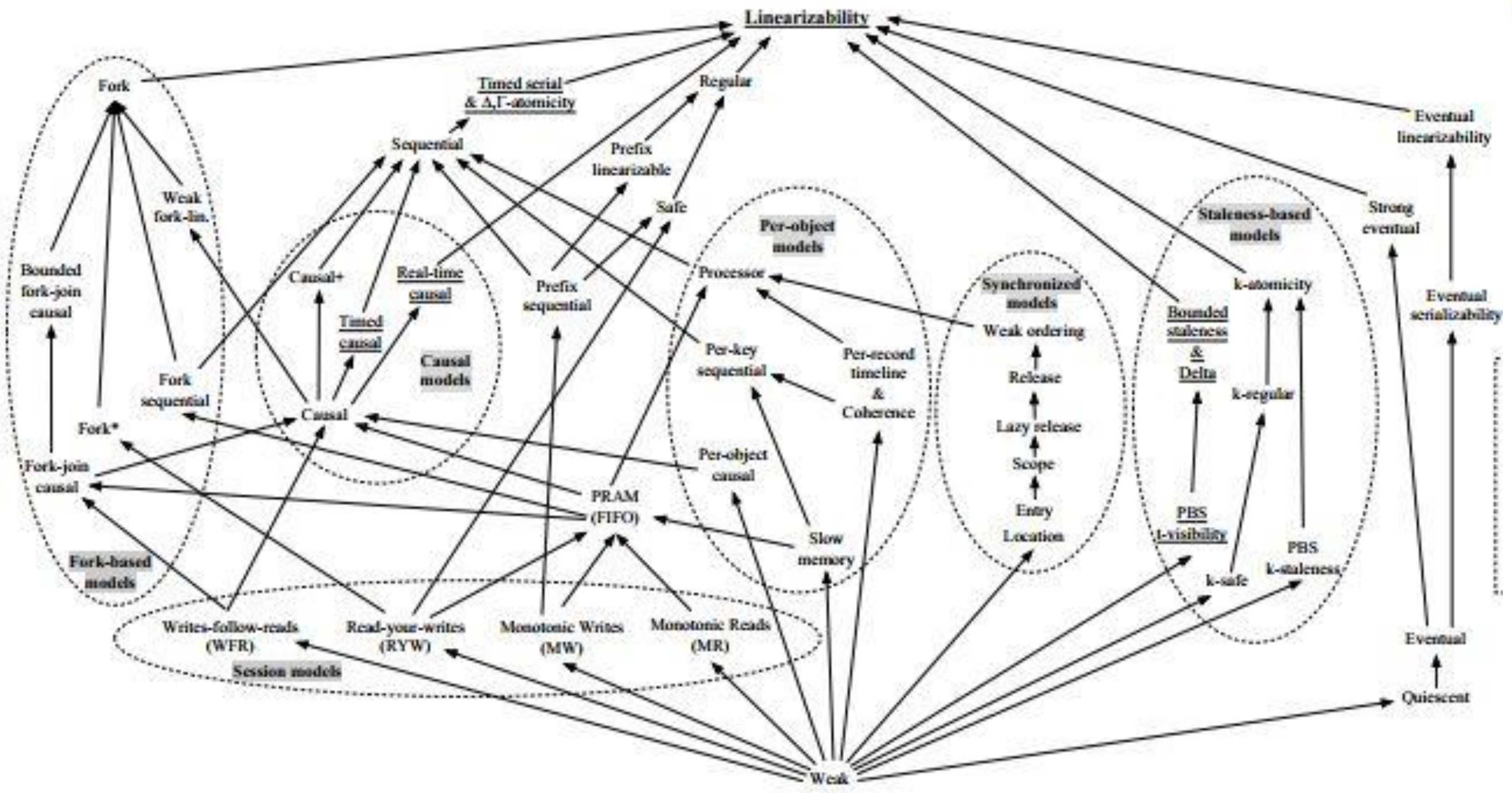
From atomicity to consistency

Linearizability (from week8)

- Linearizability extends sequential consistency by requiring that the real time order of the execution is preserved

- Linearizability extends sequential consistency with the following condition:
  1. Each method call should appear to take effect instantaneously at some moment between its invocation and response

Consistency definitions



Consistency in Non-Transactional Distributed Storage Systems by Paolo Viotti and Marko Vukolic

Agenda

- **Atomicity ?**
  - Git
  - Optimistic concurrency control
    - Operational transform
  - Consistency
  - **Atomicity in real life**
- Work-stealing queues
- Examination

Atomicity in real life

Timesharing (1969)

RC 4000
Operating system written in Algol + message passing primitives

Via a number of terminals several users shared the computer

Timesharing (1969)

RC 4000
Operating system written in Algol + message passing primitives

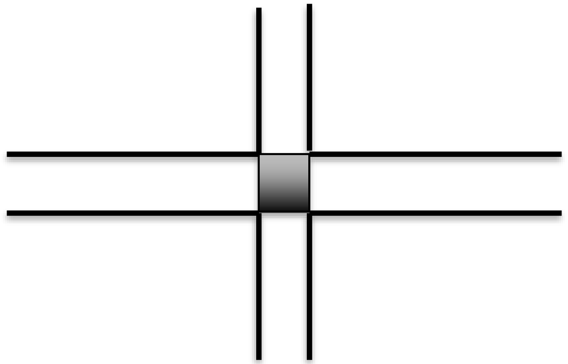
Via a number of terminals several users shared the computer

My first (and best) question:

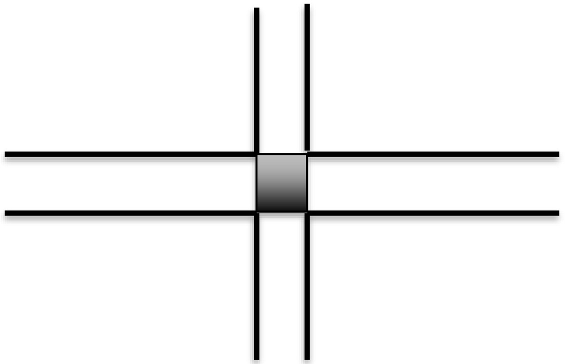
What happens if two users print simultaneously?

How to implement atomicity in real life?

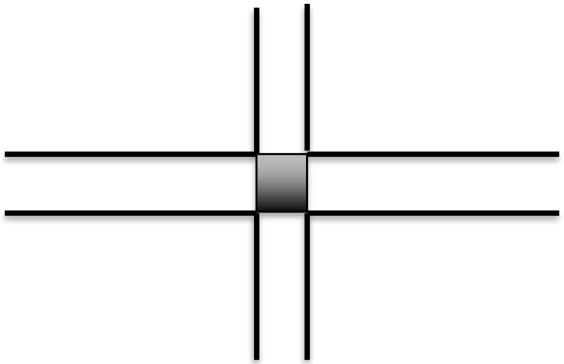
How to implement atomicity in real life?



How to implement atomicity in real life?

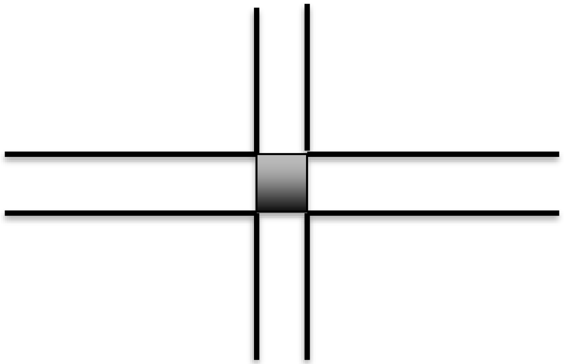


How to implement atomicity in real life?



semaphore

How to implement atomicity in real life?



```
wait(s);
  atomic operation;
signal(s);
```

semaphore

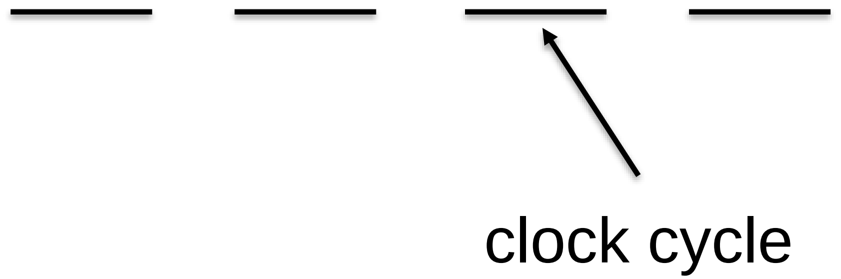
SimpleTryLock, non-blocking (Week7)

```
class SimpleTryLock {

    // Refers to holding thread, null iff unheld
    private final AtomicReference<Thread> holder = new AtomicReference<Thread>();

    public boolean tryLock() {
      final Thread current = Thread.currentThread();
      return holder.compareAndSet(null, current);
    }


    public void unlock() {
      final Thread current = Thread.currentThread();
      if (!holder.compareAndSet(current, null))
          throw new RuntimeException("Not lock holder");
    }

}
```

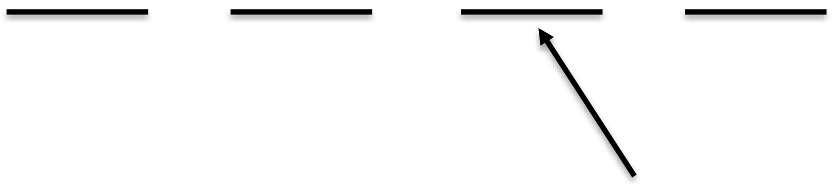
If the lock is free (holder == null), takes it and return true. Otherwise, holder is unmodified and returns false.

Sets holder to null. If CAS returns false throws an exception indicating that this thread is not holding the lock.

Atomic ⇔ One clock cycle



Atomic ⇔ One clock cycle



clock cycle

compare and set
must be done in
one clock cycle

But what about I/O ?

___ ___ ___ ___

But what about I/O ?

external events: e.g. pushing a key on the keyboard?
Not controlled by the clock

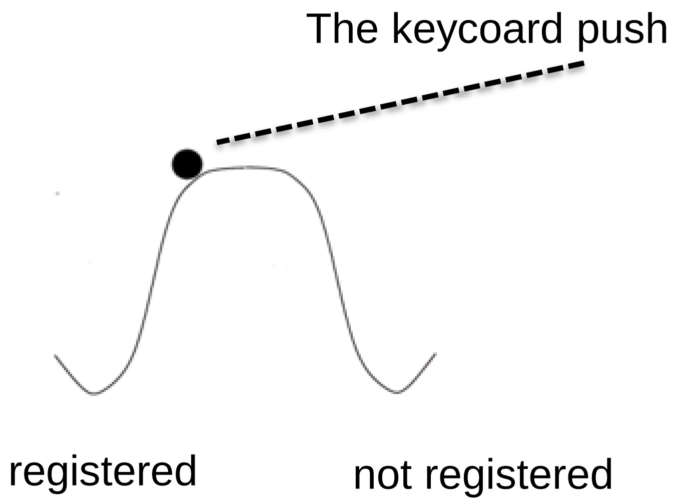
But what about I/O ?

____ ____ ____ ____

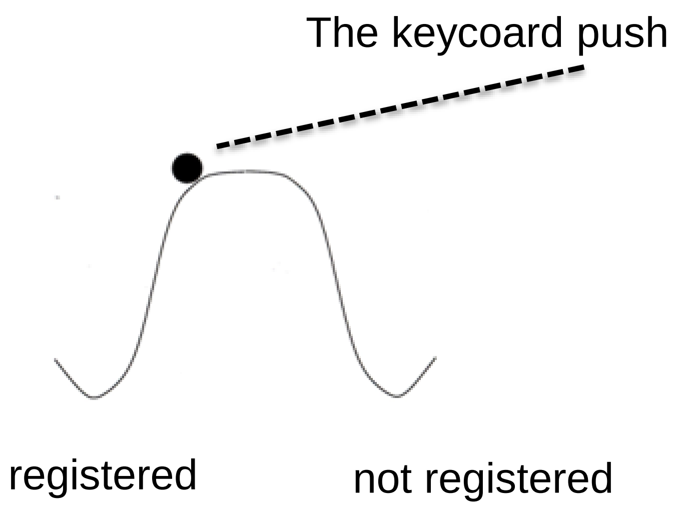
external events: e.g. pushing a key on the keyboard?
Not controlled by the clock

What happens if CAS is used to register a key push ?

The CAS hardware

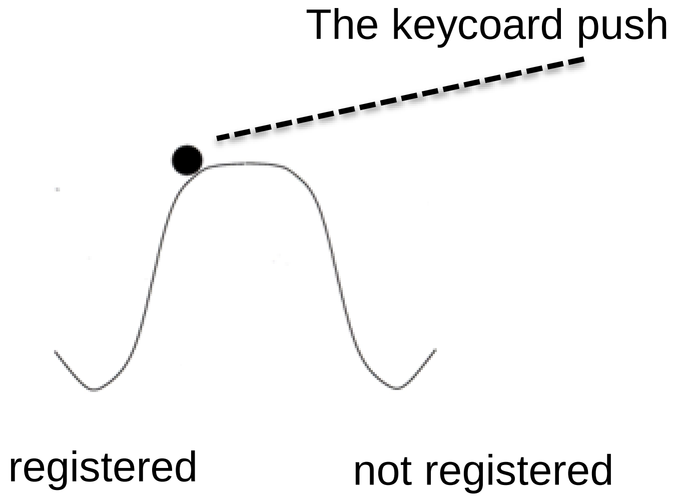


The CAS hardware



will ultimately settle

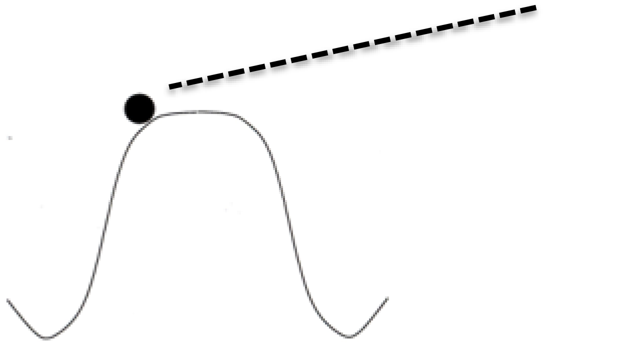
The CAS hardware



will ultimately
settle

but **no bound** on when !!!

The CAS hardware



will ultimately
settle

but **no bound** on when !!!

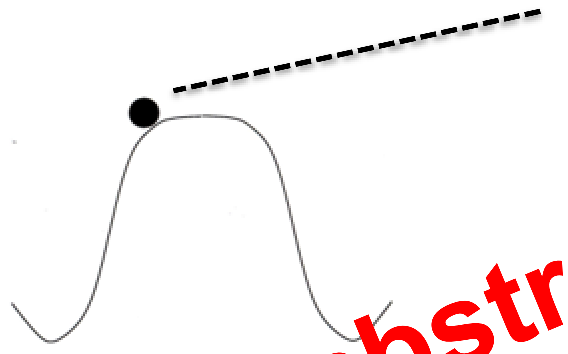
registered not registered

Anomalous Behavior of Synchronizer and Arbiter Circuits. Thomas J. Chaney and Charles E. Molnar, IEEE TC 22, April 1973

General Theory of Metastable Operations, Leonard Marino, IEEE TC 30, February 1981

Buridans donkey ~1230 https://en.wikipedia.org/wiki/Buridan's_ass

The CAS hardware



will ultimately settle

but **no bound** on when !!!

Atomicity is an abstraction !!!

Anomalous Behavior of Synchronizer and Arbiter Circuits. Thomas J. Chaney and Charles E. Molnar, IEEE TC 22, April 1973

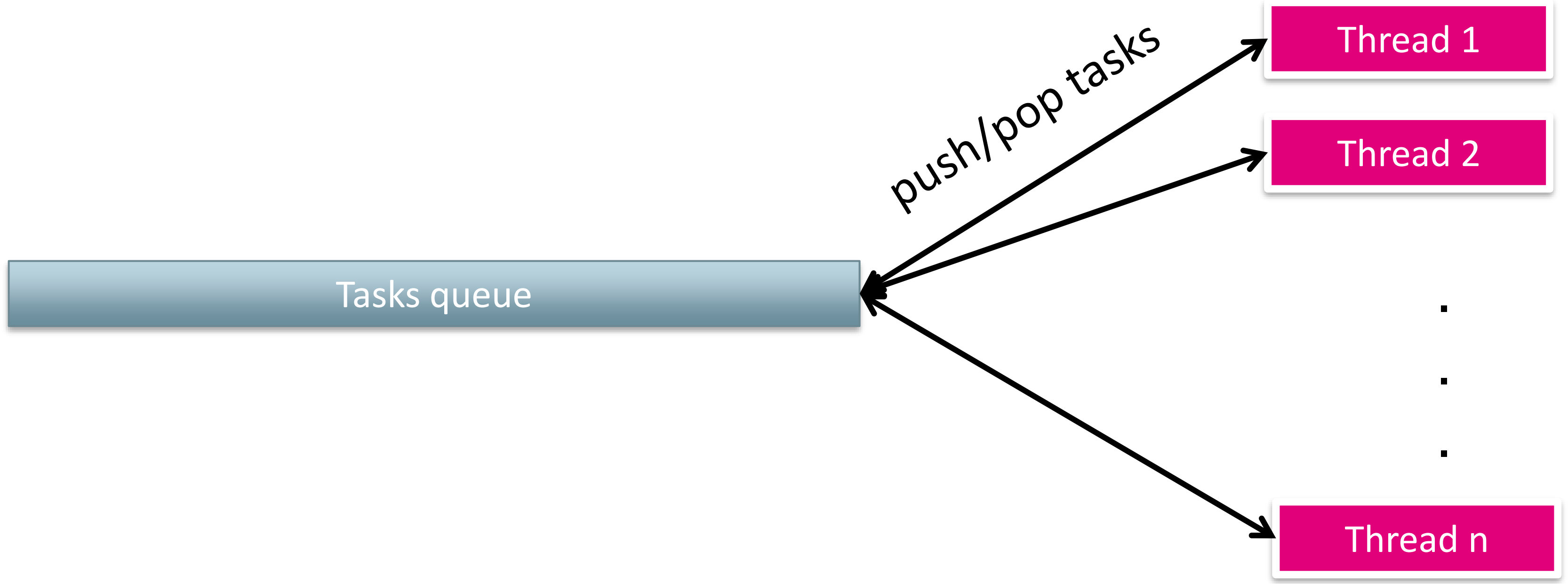
General Theory of Metastable Operations, Leonard Marino, IEEE TC 30, February 1981

Buridans donkey ~1230 https://en.wikipedia.org/wiki/Buridan's_ass

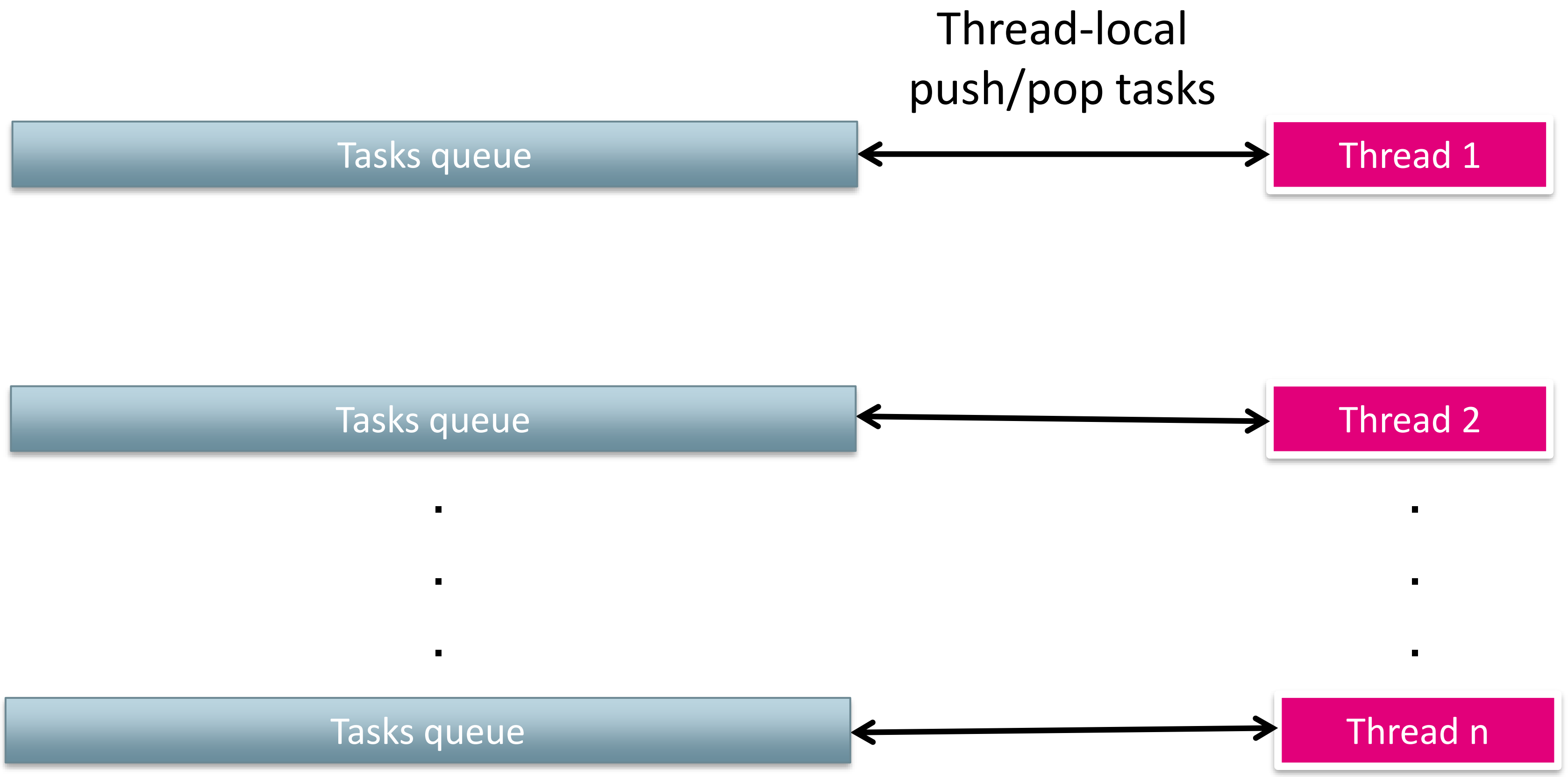
Agenda

- **Atomicity ?**
    - Git
    - Optimistic concurrency control
        - Operational transform
    - Consistency
    - Atomicity in real life
- **Work-stealing queues**
- Examination

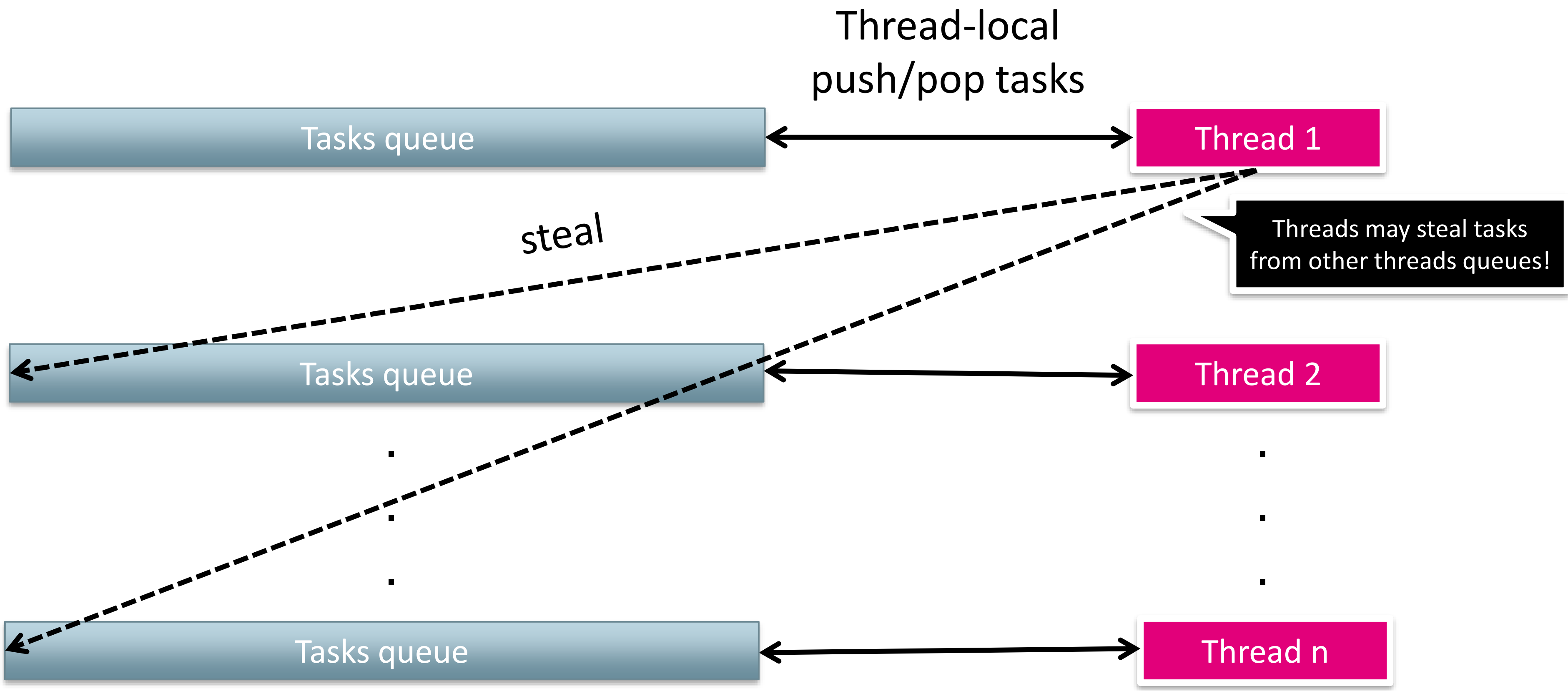
Recall the executor framework



Executor framework work-stealing task queue

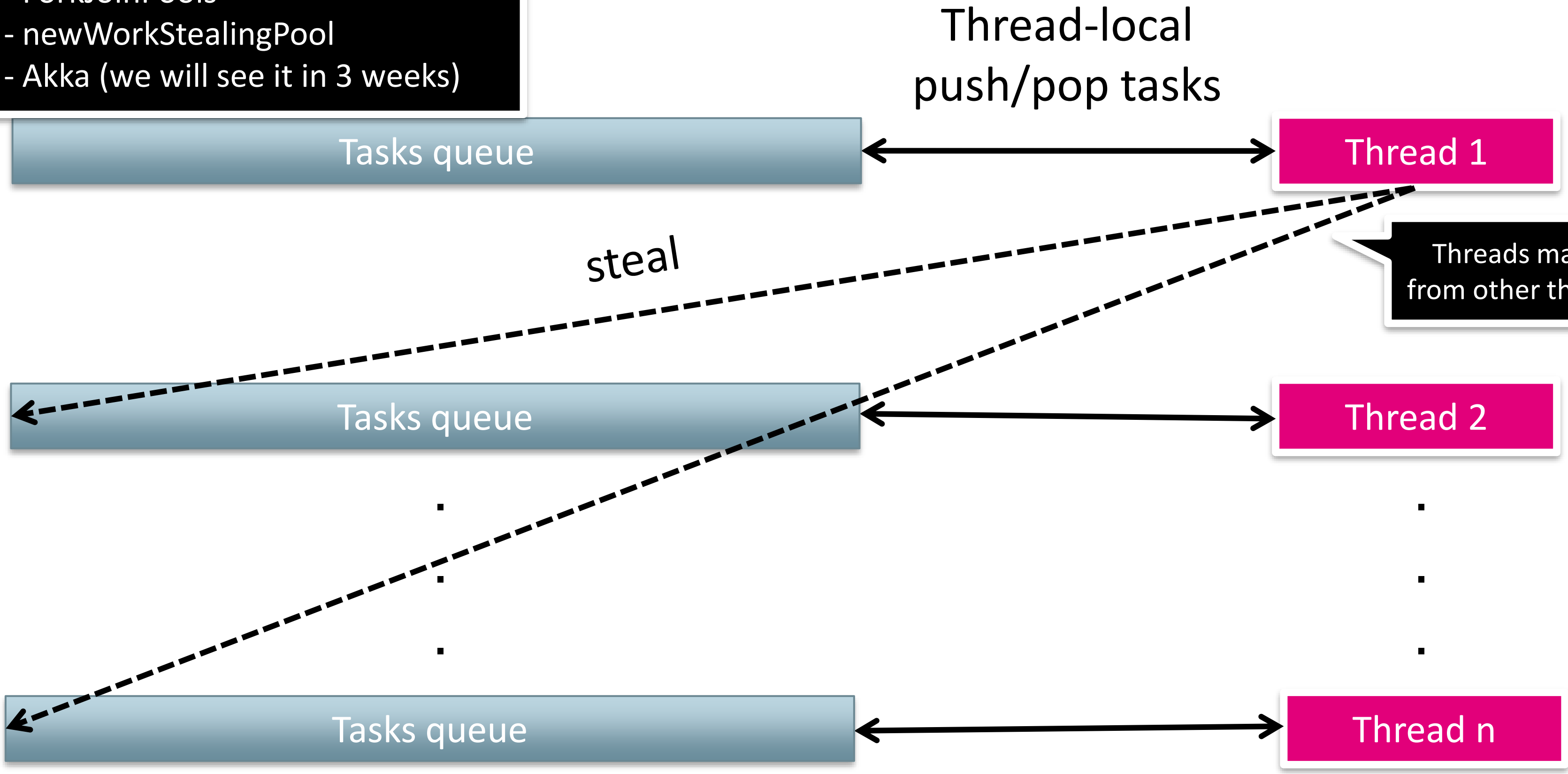


Executor framework work-stealing task queue



Executor framework work-stealing task queue

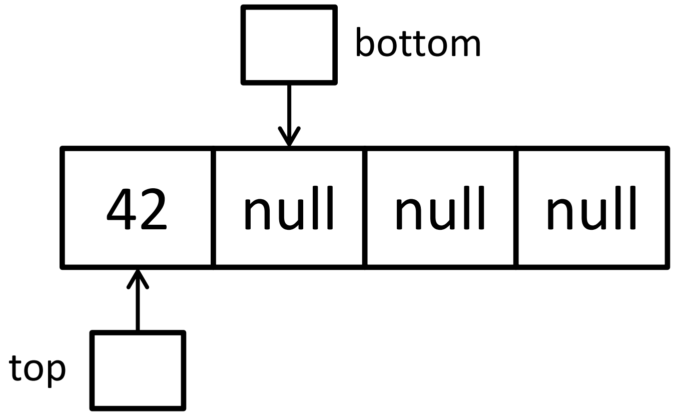
It is used in the implementation of:
- ForkJoinPools
- newWorkStealingPool
- Akka (we will see it in 3 weeks)



Threads may steal tasks from other threads queues!

Work-stealing queues

- A work-stealing queue has the following methods
  - Push – adds an element at the bottom of the queue (thread-local)
  - Pop – removes an element from the bottom of the queue (thread-local)
  - Steal – removes an element from the top of the queue (concurrent)

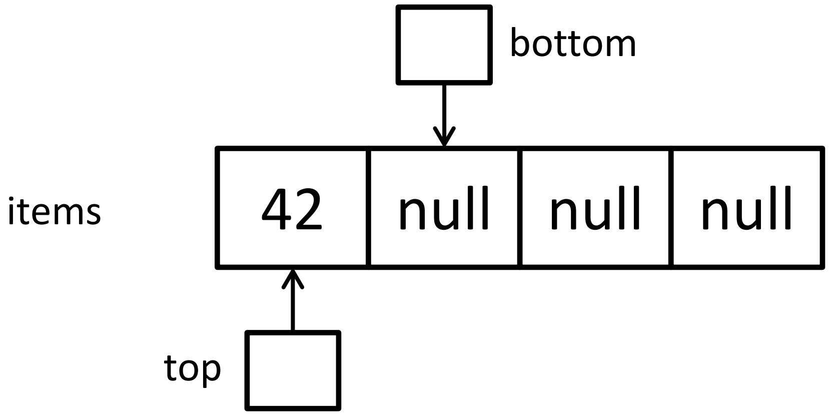


```
interface Deque<T> {
  void push(T item); // at bottom
  T pop();           // from bottom
  T steal();         // from top
}
```

We consider a simplified implementation with a fix size array

Chase-Lev work-stealing queue - state

```
class ChaseLevDeque<T> implements Deque<T> {
  private volatile long bottom = 0;
  private final AtomicLong top = new AtomicLong();
  private final T[] items;
…
}
```

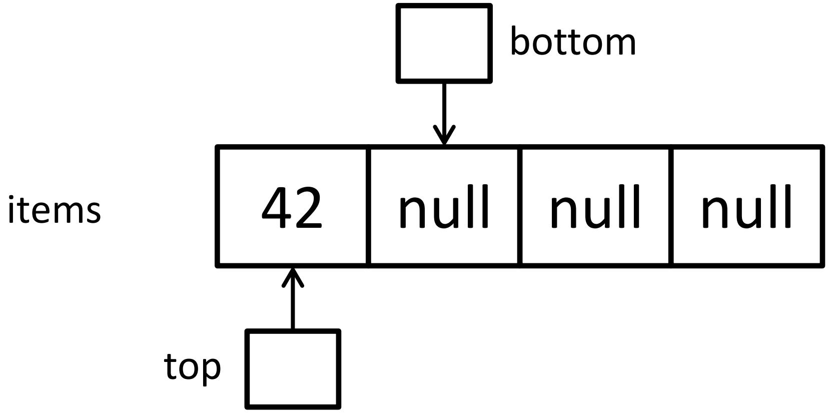


- The variable bottom is thread-local
  - Only the thread assigned to the queue can write it (other threads may read it)
- Any thread can read/write the variable top
  - We need an atomic variable to prevent data races
- For simplicity, we consider a fix-size array to store the elements of the queue
  - The array is used as a circular buffer

Chase-Lev work-stealing queue - state

```
class ChaseLevDeque<T> implements Deque<T> {
  private volatile long bottom = 0;
  private final AtomicLong top = new AtomicLong();
  private final T[] items;
…
}
```

Why is volatile enough for bottom?

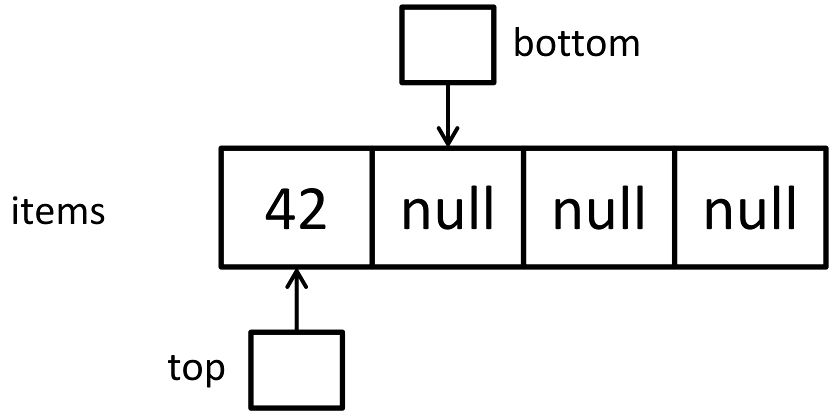


- The variable bottom is thread-local
  - Only the thread assigned to the queue can write it (other threads may read it)
- Any thread can read/write the variable top
  - We need an atomic variable to prevent data races
- For simplicity, we consider a fix-size array to store the elements of the queue
  - The array is used as a circular buffer

Chase-Lev work-stealing queue - push

```
public void push(T item) { // at bottom
  final long b = bottom, t = top.get(), size = b - t;
  if (size == items.length)
    throw new RuntimeException("queue overflow");
  items[index(b, items.length)] = item;
  bottom = b+1;
}
```

- Thread-safe because it is assumed to be thread-local
  - Always the same thread executes this method
  - Only writes bottom

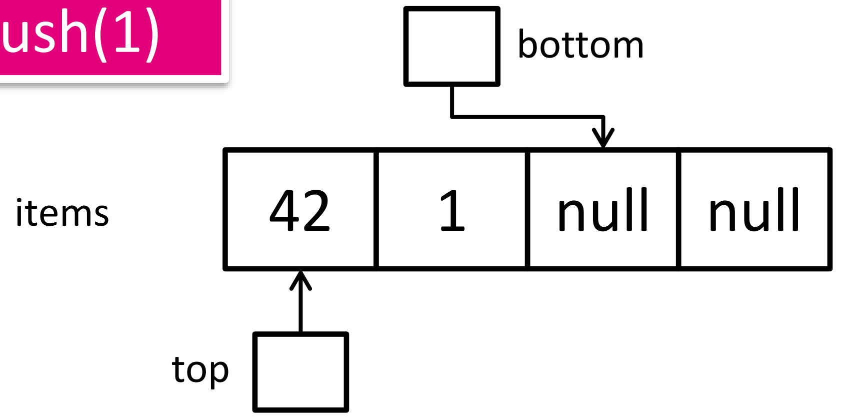


Chase-Lev work-stealing queue - push

```
public void push(T item) { // at bottom
  final long b = bottom, t = top.get(), size = b - t;
  if (size == items.length)
    throw new RuntimeException("queue overflow");
  items[index(b, items.length)] = item;
  bottom = b+1;
}
```

- Always the same thread executes this method
- Thread-safe because only writes bottom are thread-local (see other methods)

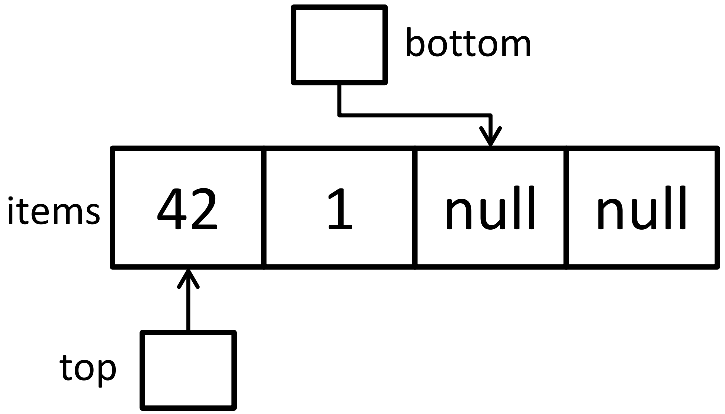
push(1)



Chase-Lev work-stealing queue - steal

```
public T steal() { // from top
  final long t = top.get();
  final long b = bottom;
  final long size = b - t;
  if (size <= 0)
    return null;
  else {
    T result = items[index(t, items.length)];
    if (top.compareAndSet(t, t+1))
      return result;
    else
      return null;
  }
}
```

- It is executed by multiple threads
- Only reads bottom
- Performs a CAS on top to steal the top element
  - Only if not empty

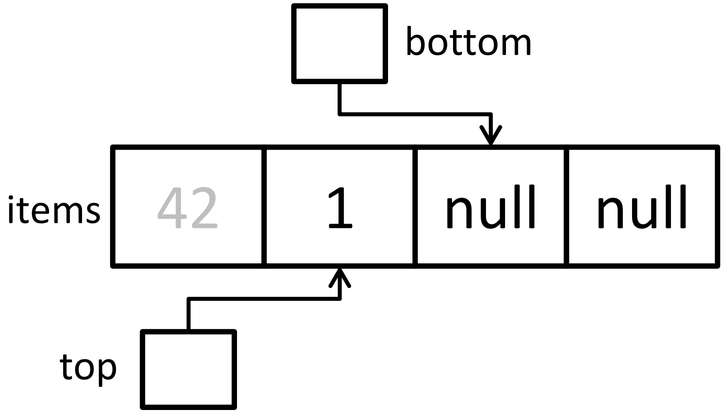


Chase-Lev work-stealing queue - steal

```
public T steal() { // from top
  final long t = top.get();
  final long b = bottom;
  final long size = b - t;
  if (size <= 0)
    return null;
  else {
    T result = items[index(t, items.length)];
    if (top.compareAndSet(t, t+1))
      return result;
    else
      return null;
  }
}
```

steal() -> 42

- It is executed by multiple threads
- Only reads bottom
- Performs a CAS on top to steal the top element
  - Only if not empty



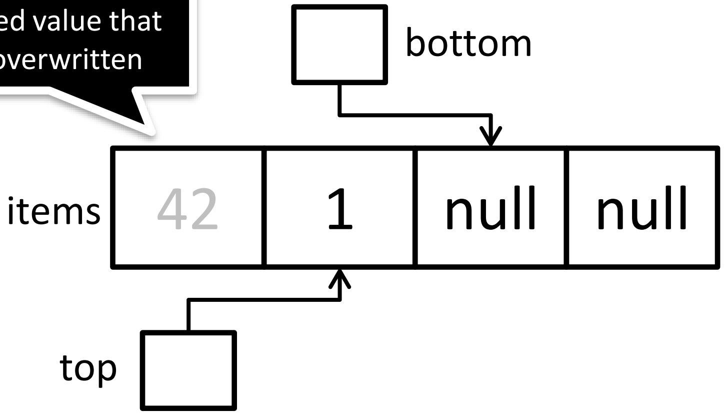
Chase-Lev work-stealing queue - steal

```
public T steal() { // from top
  final long t = top.get();
  final long b = bottom;
  final long size = b - t;
  if (size <= 0)
    return null;
  else {
    T result = items[index(t, items.length)];
    if (top.compareAndSet(t, t+1))
      return result;
    else
      return null;
  }
}
```

steal() -> 42

- It is executed by multiple threads
- Only reads bottom
- Performs a CAS on top to steal the top element
  - Only if not empty

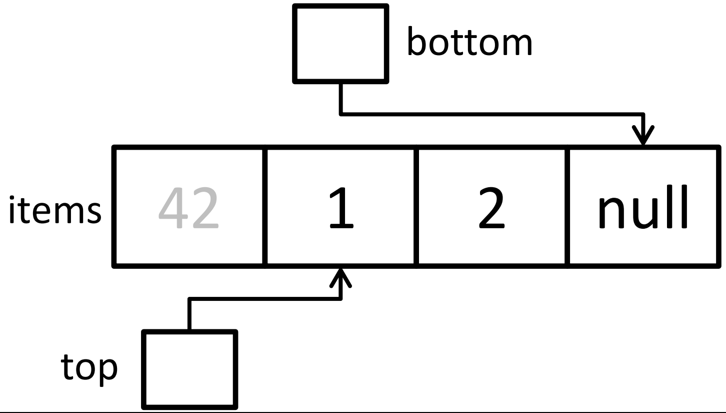
This becomes a deprecated value that will be overwritten



Chase-Lev work-stealing queue - pop

```
public T pop() { // from bottom
  final long b = bottom - 1;
  bottom = b;
  final long t = top.get(),
  final long afterSize = b - t;
  if (afterSize < 0) {
    bottom = t;
    return null;
  } else {
    T result = items[index(b, items.length)];
    if (afterSize > 0)
      return result;
    else {
      if (!top.compareAndSet(t, t+1))
          result = null;
      bottom = t+1;
      return result;
    }
  }
}
```

- Thread-local but more subtle than push
- It updates bottom (thread-local) and possibly top (concurrent)

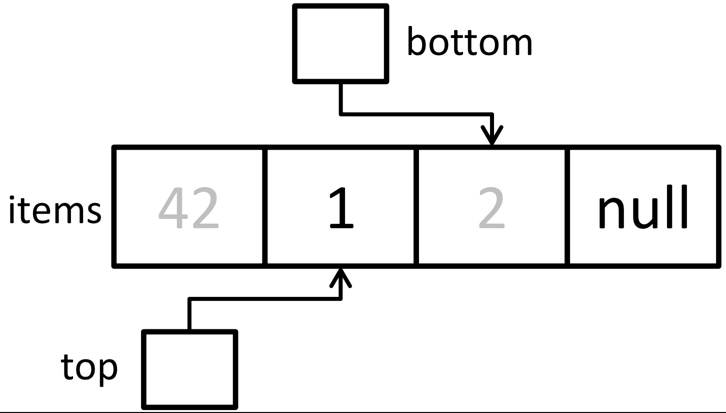


Chase-Lev work-stealing queue - pop

```
public T pop() { // from bottom
  final long b = bottom - 1;
  bottom = b;
  final long t = top.get(),
  final long afterSize = b - t;
  if (afterSize < 0) {
    bottom = t;
    return null;
  } else {
    T result = items[index(b, items.length)];
    if (afterSize > 0)
      return result;
    else {
      if (!top.compareAndSet(t, t+1))
          result = null;
      bottom = t+1;
      return result;
    }
  }
}
```

pop() -> 2

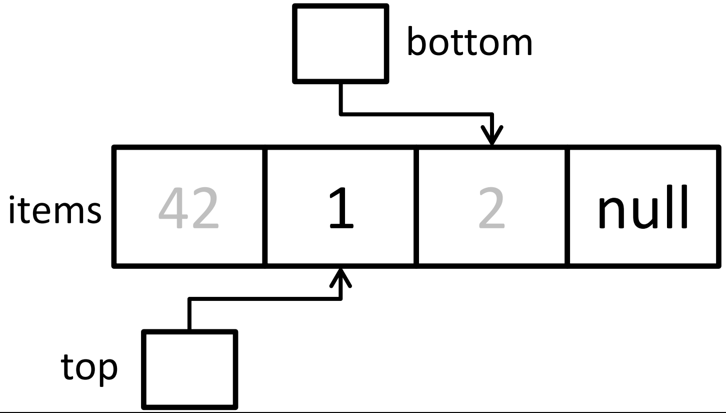
- When only the assign thread executes, then we simply update bottom and return the element



Chase-Lev work-stealing queue - pop

```
public T pop() { // from bottom
  final long b = bottom - 1;
  bottom = b;
  final long t = top.get(),
  final long afterSize = b - t;
  if (afterSize < 0) {
    bottom = t;
    return null;
  } else {
    T result = items[index(b, items.length)];
    if (afterSize > 0)
      return result;
    else {
      if (!top.compareAndSet(t, t+1))
          result = null;
      bottom = t+1;
      return result;
    }
  }
}
```

- What if we had pop() and steal() concurrently?

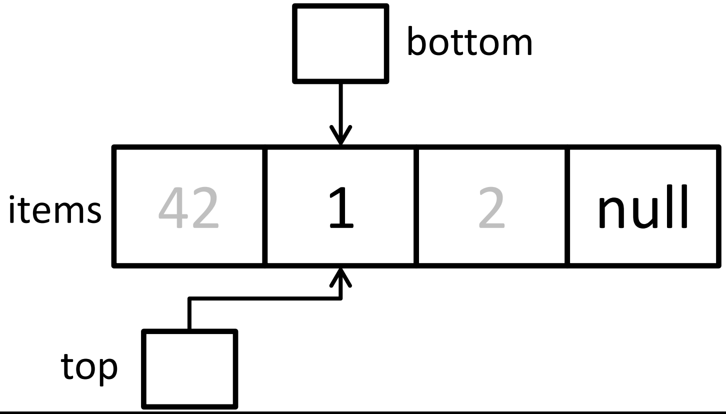


Chase-Lev work-stealing queue - pop

```
public T pop() { // from bottom
  final long b = bottom - 1;
  bottom = b;
  final long t = top.get(),
  final long afterSize = b - t;
  if (afterSize < 0) {
    bottom = t;
    return null;
  } else {
    T result = items[index(b, items.length)];
    if (afterSize > 0)
      return result;
    else {
      if (!top.compareAndSet(t, t+1))
          result = null;
      bottom = t+1;
      return result;
    }
  }
}
```

pop() -> ?

- What if we had pop() and steal() concurrently?



Chase-Lev ue - pop

```
public T steal() { // from top
  final long t = top.get();
  final long b = bottom;
  final long size = b - t;
  if (size <= 0)
    return null;
  else {
    T result = items[index(t, items.length)];
    if (top.compareAndSet(t, t+1))
      return result;
    else
      return null;
  }
}
```

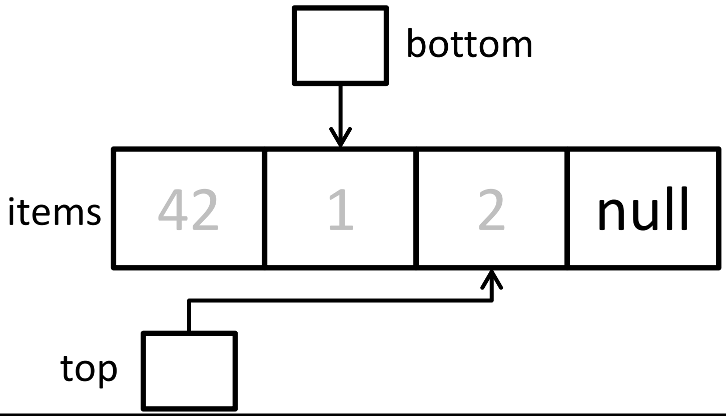
```
public T pop() { /
  final long b = b
  bottom = b;
  final long t = t
  final long afterSize = b - t;
  if (afterSize < 0) {
    bottom = t;
    return null;
  } else {
    T result = items[index(b, items.length)];
    if (afterSize > 0)
      return result;
    else {
      if (!top.compareAndSet(t, t+1))
          result = null;
      bottom = t+1;
      return result;
    }
  }
}
```

pop() -> ?

steal() -> ?

What if we had pop() and steal() concurrently?

- Whatever thread succeeds in the CAS operation gets the element



Chase-Lev ue - pop

```
public T steal() { // from top
  final long t = top.get();
  final long b = bottom;
  final long size = b - t;
  if (size <= 0)
    return null;
  else {
    T result = items[index(t, items.length)];
    if (top.compareAndSet(t, t+1))
      return result;
    else
      return null;
  }
}
```

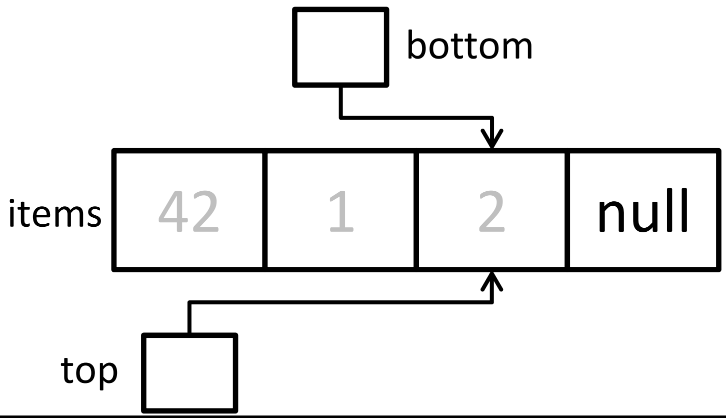
```
public T pop() { /
  final long b = b
  bottom = b;
  final long t = t
  final long afterSize = b - t;
  if (afterSize < 0) {
    bottom = t;
    return null;
  } else {
    T result = items[index(b, items.length)];
    if (afterSize > 0)
      return result;
    else {
      if (!top.compareAndSet(t, t+1))
          result = null;
      bottom = t+1;
      return result;
    }
  }
}
```

pop() -> ?

steal() -> ?

What if we had pop() and steal() concurrently?

- Whatever thread succeeds in the CAS operation gets the element
- Afterwards, pop always fixes the bottom variable



Agenda

- Atomicity ?
  - Git
  - Optimistic concurrency control
    - Operational transform
  - Consistency
  - Atomicity in real life
- Work-stealing queues
- **Examination**

Examination – Material

- The folder `exam` in the GitHub repository contains
  - *The mandatory readings for the exam* (we can ask questions about any of these readings)
  - *Questions for the exam*

  Although *the list is preliminary and subject to change*, you can consider this an *almost final version*

- Please **read the list with mandatory reading and exam questions carefully** and ask for any clarifications/comments
  - **Send questions and/or topics to revisitto Raúl (raup@itu.dk) before Thursday Nov 30th**

- Week 14 will be mostly about addressing your question/comments

- *Questions and answers in the LearnIT forum are not part of the mandatory readings*
  - The Q&A forum will be closed soon after we finish the course

Examination – Preparation

- **Prepare a short presentation for each question**
  - You may find inspiration in this video https://www.youtube.com/watch?v=587aD3tWSGk

- Make a short agenda
  for the answer to each question
  1. Motivation for concept X
  2. Key elements
  3. Challenges/Shortcomings/Alternatives
  4. Code examples
     - **Use code examples from your assignments**

- **Thoroughly study the mandatory readings**

Examination – Process

- The exam starts with a question you draw (at random) from the list of questions in GitHub
- Afterwards, the teachers and examiners may ask you anything from the mandatory readings
- While you answer a question, teachers and examiners may ask about specific details related to the question you are answering

Examination – What you can bring to the exam

- One A4 paper (optional)
  - With the ***<u>agenda</u>*** for the short presentation you may prepare for each question
  - You cannot write full answers to the questions in this page
  - If we see you are reading from the paper, we will probably switch to other topics

- Your laptop or printout of the code
  - To show code example(s)

Mandatory assignments

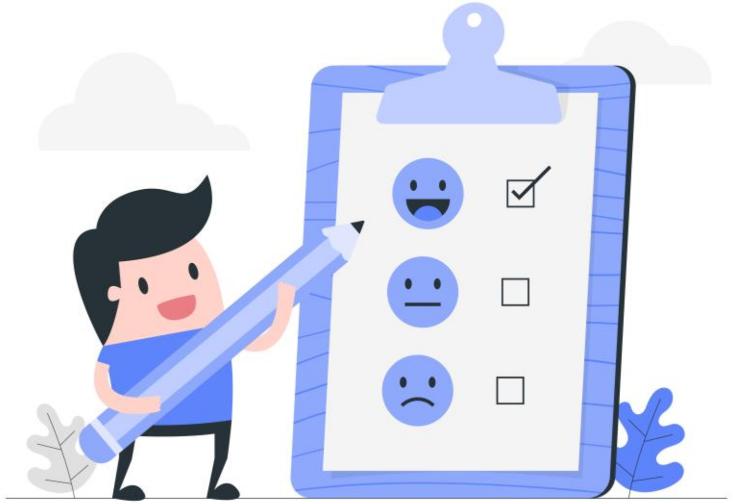
- To be eligible for the exam, 5 (or more) mandatory assignments must be approved
- You will get confirmation in the feedback for assignment 6
  - *"Your assignments have been approved and you may take the exam"*
- *It is your responsibility to let us know if there are any errors in grading*
  - For instance, missing grades, ungraded assignment, etc.
- There will be a final extra deadline on Dec 14th to hand-in assignments that have not yet been approved
  - With no possibility of re-submission and with written feedback

Examination – Dates

- Exam dates:
  - Week 3: January 17, 18 (23 spots)
  - Week 4: January 23, 24, 25 (96 spots)
- If you have constraints (e.g. other exams), please inform Raúl via e-mail (raup@itu.dk) by Dec 14th
  - We cannot guarantee that we will meet all constraints, but we will do our best
  - The more constraints we get, the more difficult it is to meet them
  - Please consider carefully whether your constraint is justified/reasonable
- The final schedule will be available in LearnIT in early January

Course Evaluation Survey

Please participate in the course evaluation



Questions ?