Practical Concurrent and Parallel Programming XIII
Atomicity ?

Raúl Pardo and
Jørgen Staunstrup

Agenda

- **Follow-up on Exercise from week 10**
- Git
- Optimistic concurrency control
- Operational transform
- Consistency
- Atomicity
- Examination
- Course evaluation survey

Exercise 10.2

Performance measurement of CASHistogram vs. Histogram2 (lock-based)

```
casHistogram test                39501337,5 ns 3520507,69          8
lockHistogram test               42774195,0 ns 7058273,01          8
```

"Yes, it performed as expected. The more contention we introduce, the better CasHisgram performed. ... "

Hm !

Benchmark code

```
Mark7("casHistogram test", i -> {
    countParallel(i, 512, casHistogram);
    return (double) casHistogram.getCount(0);
});

Mark7("lockHistogram test", i -> {
    countParallel(i, 512, lockHistogram);
    return (double) lockHistogram.getCount(0);
});
```

Benchmark code

```
Mark7("casHistogram test", i -> {
    countParallel(i, 512, casHistogram);
    return (double) casHistogram.getCount(0);
});

Mark7("lockHistogram test", i -> {
    countParallel(i, 512, lockHistogram);
    return (double) lockHistogram.getCount(0);
});
```

?

```
… countParallel(int range, int threadCount, Histogram h)
```

```
public static double Mark7(String msg, IntToDoubleFunction f) {
          ...
          do {
               count *= 2;
                    ...
                    for (int i = 0; i < count; i++)
                         dummy += f.applyAsDouble(i);
                    ...
               }
          }
```

Benchmark code


```
Mark7("casHistogram test", i -> {
    countParallel(range, 512, casHistogram);
    return (double) casHistogram.getCount(0);
});

Mark7("lockHistogram test", i -> {
    countParallel(range, 512, lockHistogram);
    return (double) lockHistogram.getCount(0);
});
```

```
casHistogram test                 5602045.8 ns  115070.68          64
lockHistogram test               19914000.0 ns  109678.50          16
```

Always be suspicious about your code !!!!

Agenda

- Follow-up on Exercise from week 10
- **Git**
- Optimistic concurrency control
- Operational transform
- Consistency
- Atomicity
- Examination

Synchronization errors

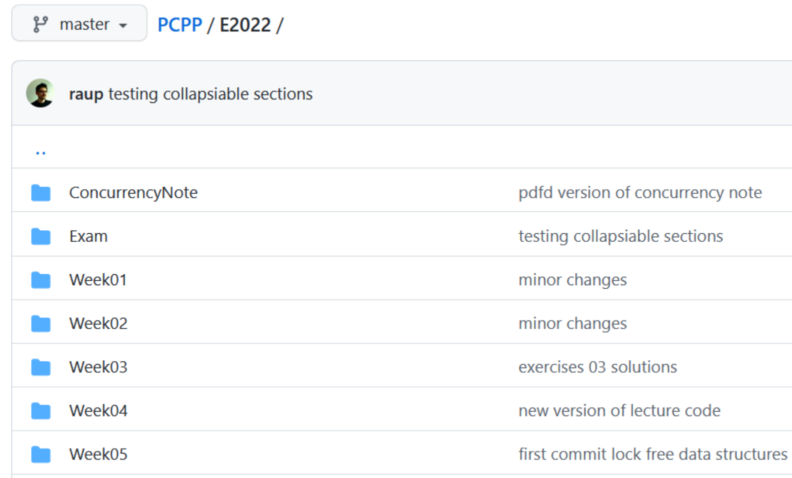
Some strategies

Avoid them — Atomicity (synchronized)

Fix them ← This week

In Danish "pyt" (Live with them)

File sharing with Git



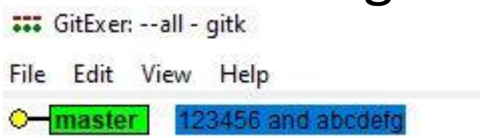
Workflow:

```
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



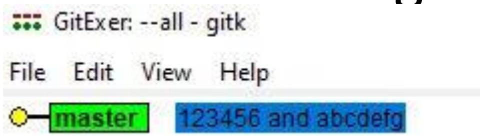

```
git branch newnumbers
git checkout newnumbers
```

change file numbers.txt: 1234


```
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




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

change file numbers.txt: 12xy4q


```
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

- Follow-up on Exercise from week 10
- Git
- **Optimistic concurrency control**
- Operational transform
- Consistency
- Atomicity
- Examination

Pessimistic concurrency control


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

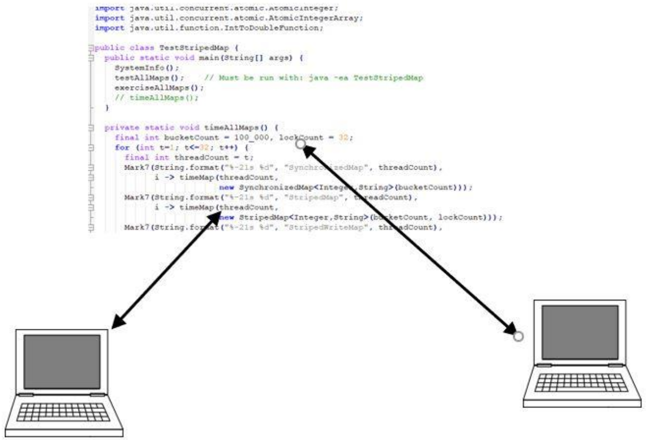
Optimistic concurrency control


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

Google Wave, Realm (MongoDB), …

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

Concurrent text editing



Google wave https://youtu.be/p6pgxLaDdQw

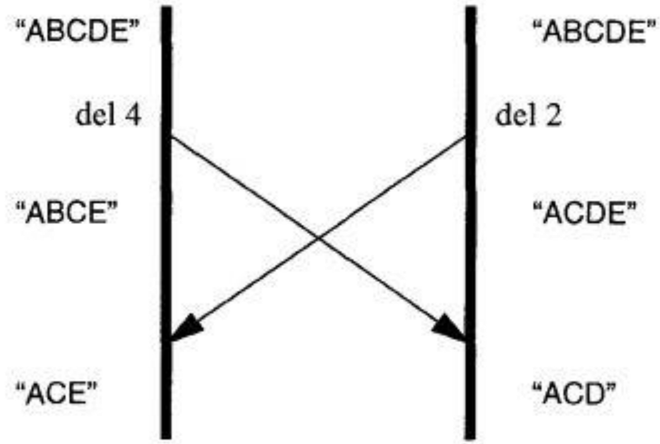
Concurrent editing survived in Google Docs, MS Office, …

Agenda

- Follow-up on Exercise from week 10
- Git
- Optimistic concurrency control
- **Operational transform**
- Consistency
- Atomicity
- Examination
- Course Evaluation

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

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

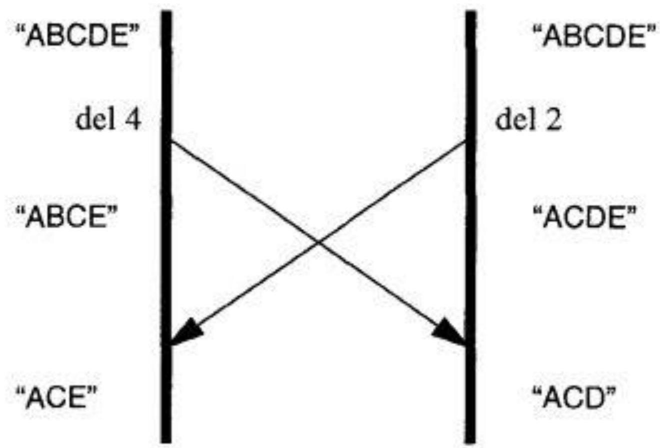
CAP theorem: *impossible* for a distributed data store to simultaneously provide more than two out of the three: consistency, availability and partition tolerance.

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

Strong eventual consistency

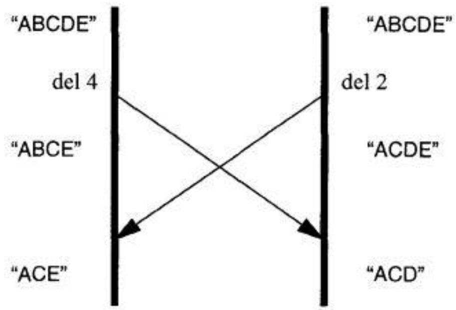
Off-line is default - AP system

When online, requests are merged (operational transform)



Operational transform (example)

Imagine a text editor where many clients can edit without locking

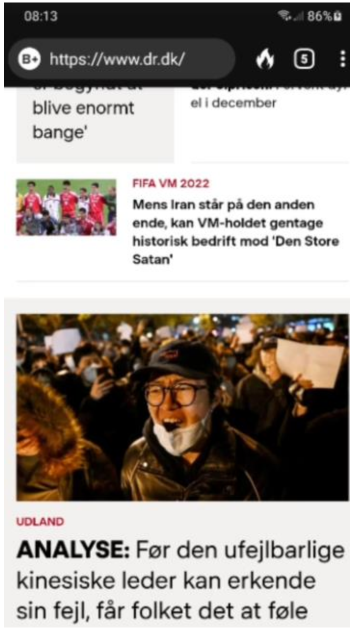


The server makes an opTrans operation on conflicting operations such as: `del4 and del2`.

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

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

Mobile app



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

Realm

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.

- 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)

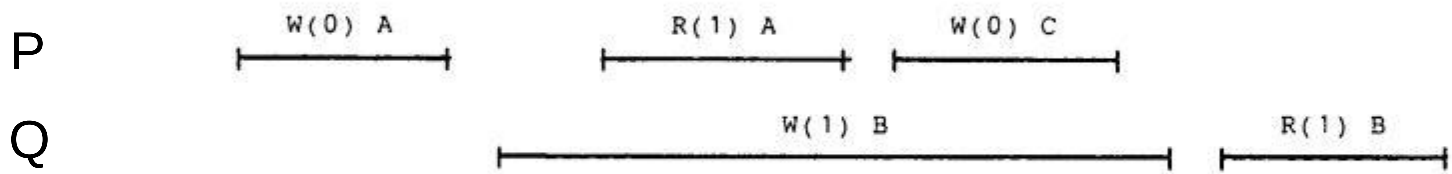
source: https://docs.mongodb.com/realm/sync/protocol/#sync-protocol

Agenda

- Follow-up on Exercise from week 10
- Git
- Optimistic concurrency control
- Operational transform
- **Consistency**
- Atomicity
- Examination
- Course Evaluation

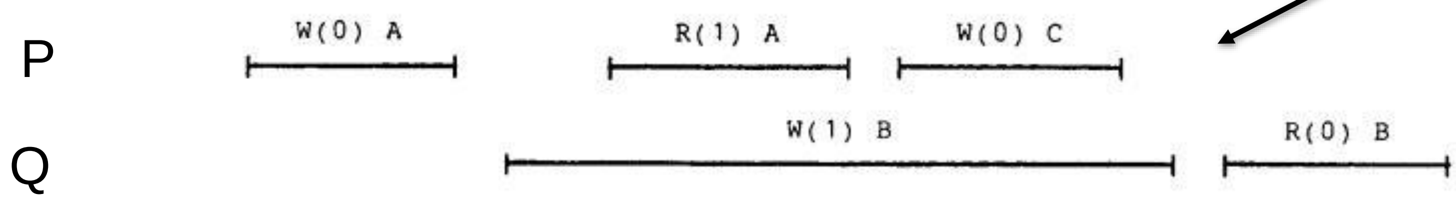
Consistency ?

Define behavior of operations on shared data (P||Q)



Is this consistent ?

Histories interleavings



Is this consistent ?

[Readings week 13:WingHerlihy.pdf]

Linearizable

[Readings week 13:WingHerlihy.pdf]

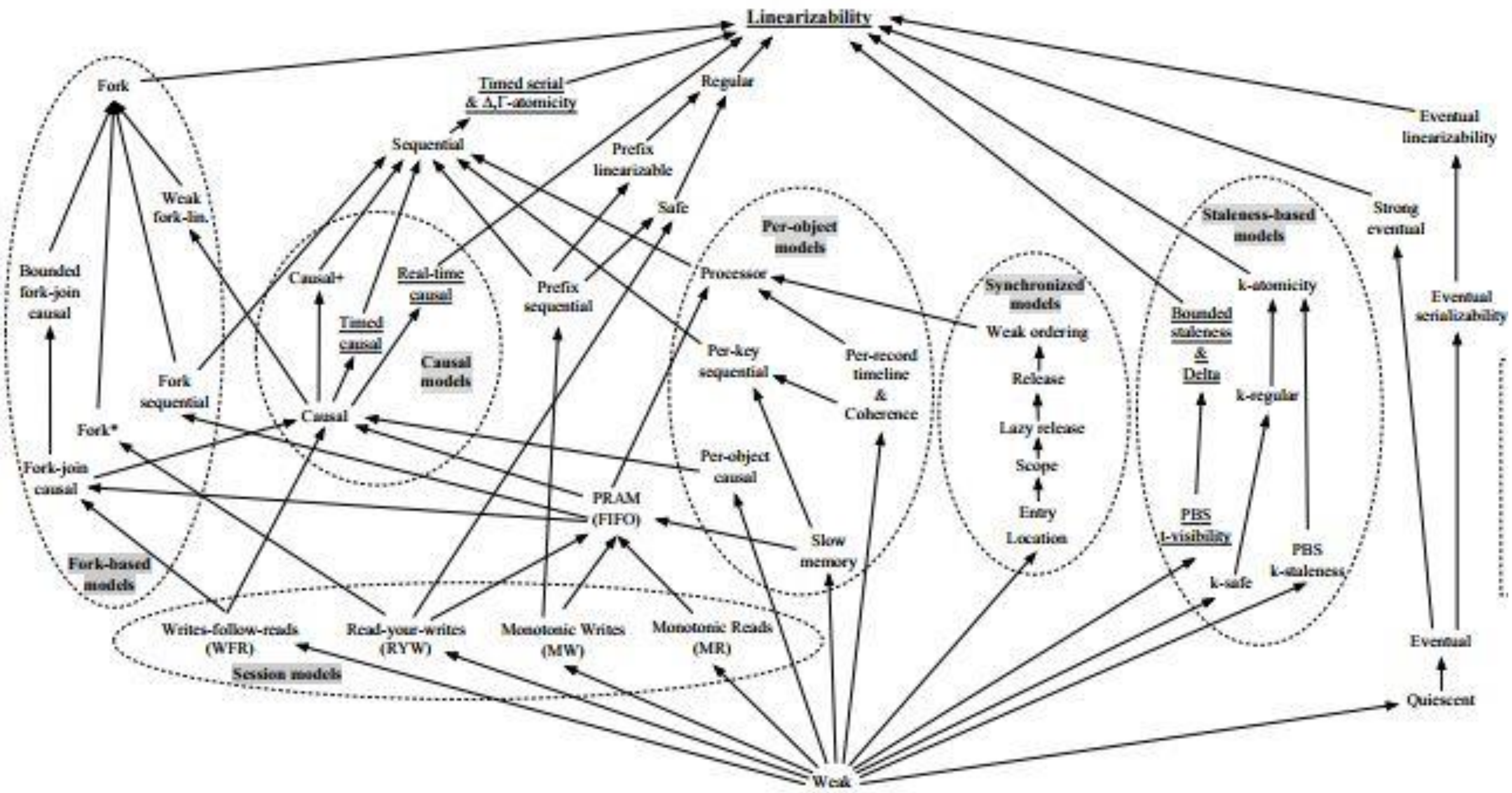
possible interleavings of operation invocations.

A concurrent computation is *linearizable* if it is "equivalent," in a sense formally defined in Section 2, to a legal sequential computation. We interpret a data type's (sequential) axiomatic specification as permitting only linearizable interleavings

A history H is *linearizable* if it can be extended (by appending zero or more response events) to some sequential history

Weaker than thread-safety

Consistency definitions



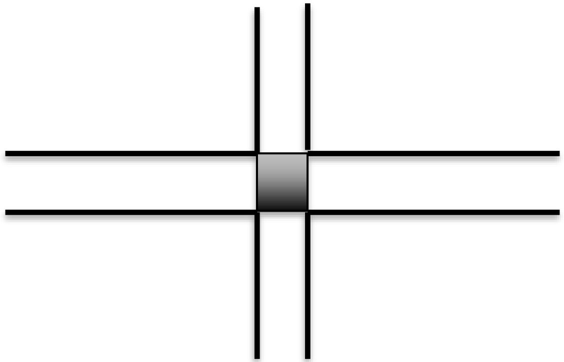
Consistency in Non-Transactional Distributed Storage Systems by Paolo Viotti and Marko Vukolic

Agenda

- Follow-up on Exercise from week 10
- Git
- Optimistic concurrency control
- Operational transform
- Consistency
- **Atomicity**
- Examination

Atomicity?

How to implement atomicity?



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

semaphore

SimpleTryLock, non-blocking (Week10)


```
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.

Multi-Maren arbiter

Assume that the computer hardware offers an atomic exchange Operation (CAS operation)

a <--> b

How to implement: `a <--> b` ???

```
boolean semaphore= true; //global
```

```
boolean enter= false; //local

repeat
  enter <--> semaphore
until enter

atomic operation

enter <--> semaphore
```

...

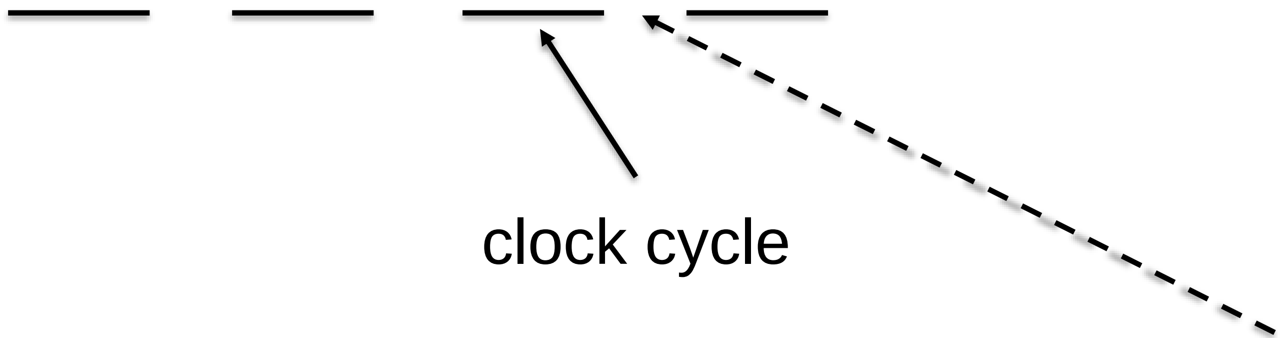
```
boolean enter= false; //local

repeat
  enter <--> semaphore
until enter

atomic operation

enter <--> semaphore
```

Atomic ⇔ One clock cycle



clock cycle

a <--> b

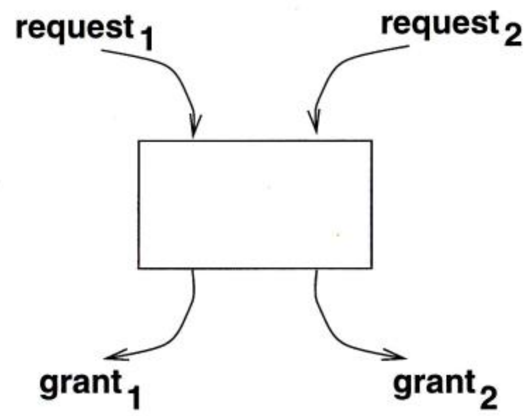
The values of a and b have been swapped

But what about I/O ?

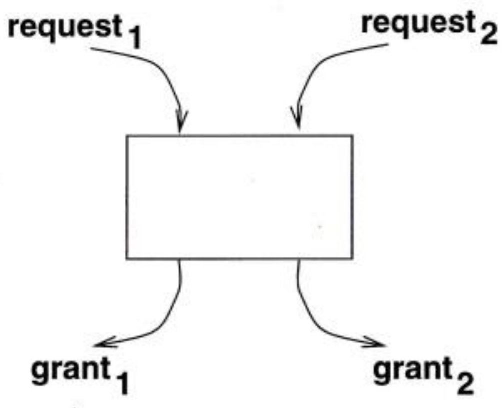
external events: e.g. pushing a key on the keyboard?

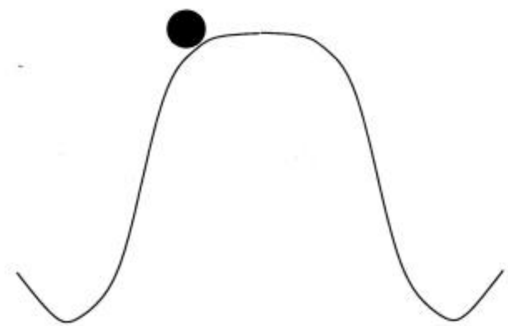
What happens ?

The arbiter



The arbiter





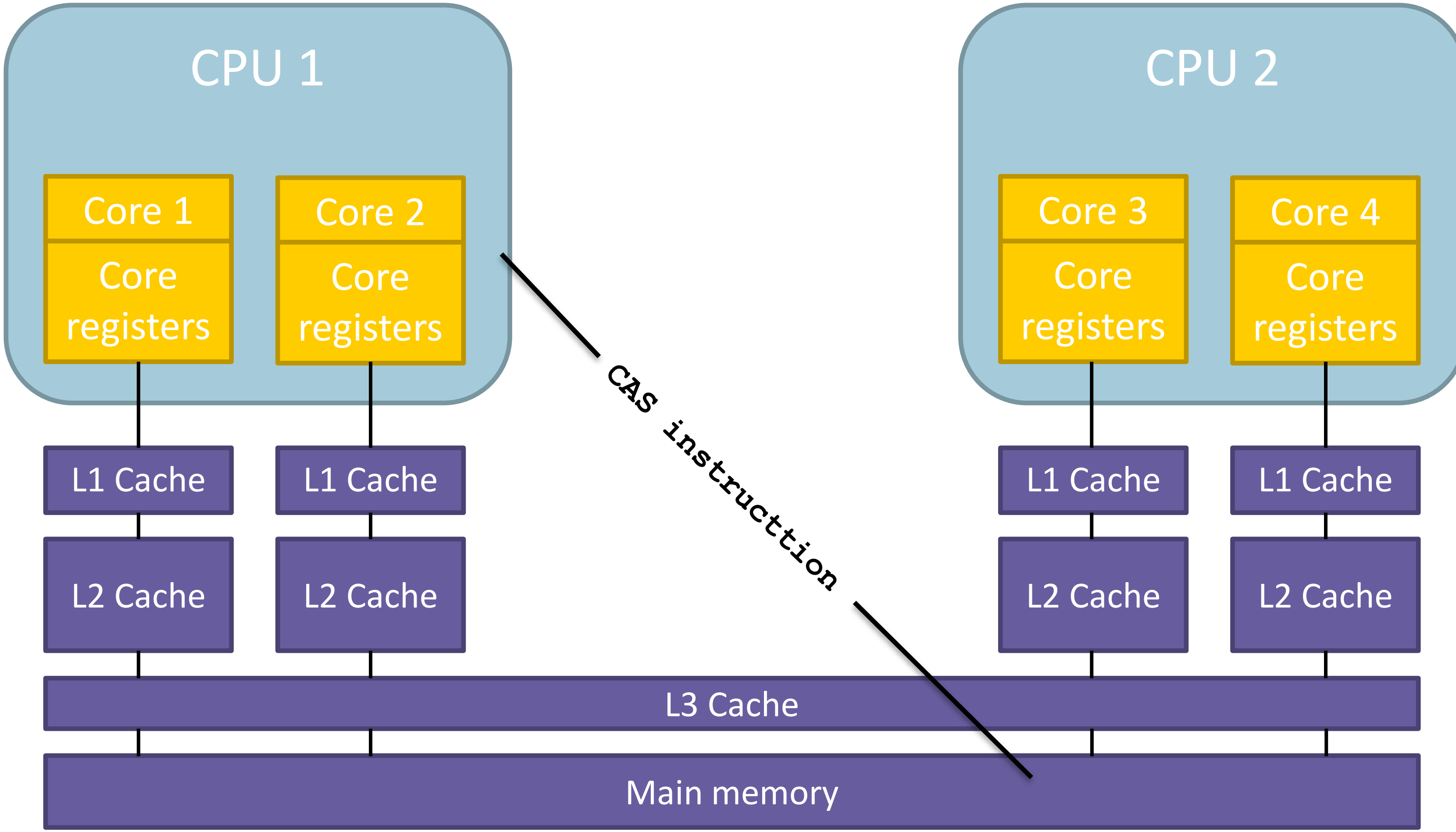
Anomalous Behavior of Synchronizer and Arbiter Circuits. Thomas J. Chaney and Charles E. Molnar, IEEE TC 22, April 1973

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

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

Atomicity is an abstraction !!!

Architecture of today's processors



Drawing and explanation inspired from: https://www.youtube.com/watch?v=nNXkzDS6dOg

Agenda

- Follow-up on Exercise from week 10
- Git
- Optimistic concurrency control
- Operational transform
- Consistency
- Atomicity
- **Examination**

Examination – Material

- The folder `exam` in the GitHub repository contains the mandatory reading
  - Although *the list is preliminary and subject to change*, you can consider this an *almost final version*

- Please **read the list with mandatory reading carefully** and ask for any clarifications/comments
  - **Send questions (e.g. to repeat something) to Raúl and Jørgen before Wednesday Dec 7th**

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

- Questions and answers in the LearnIT forum are not part of the mandatory reading

Examination – Preparation

- Try solving the challenging exercises
- Revisit the mandatory exercises
  - Reflect on whether your answers could be more precise and self-contained
- Example exam answer in `exam` in the GitHub repository
  - Send us questions for next week (see previous slide)

Exercise 10.1.1 revisited

- *"Write a class CasHistogram implementing the above interface. <u>Explain</u> why the methods increment, getBins, getSpan and getAndClear are thread-safe."*

- Answers of the type below are not sufficient:
  - It is thread-safe because I use AtomicInteger
  - It is thread-safe because I use CAS
  - …

DISCLAIMER: None of these are real answers to the exercise. Choosing this question was not motivated by any particular solution to this exercise. But due to how appropriate the question is to discuss what we mean by good answers.

Exercise 10.1.1 revisited

- A good answer must first state what thread-safety means in the context of the question
- We have seen two definitions in the course for thread-safety

*A **class** is said to be **thread-safe** if and only if no concurrent execution of method calls or field accesses (read/write) result in race conditions*

*A concurrent **program** is said to be **thread-safe** if and only if it is race condition free*

- Note that this exercise could have been better formulated, as it asks for thread-safety of methods (which does not match exactly any of the definitions above)
  - Here you should indicate/clarify what definition you will apply

Exercise 10.1.1 revisited

A concurrent ***program*** *is said to be* ***thread-safe*** *if and only if it is race condition free*

- Here, as an example, we focus on this definition and adapt it for methods
  - *"We will argue that the […] methods are thread-safe by showing that no concurrent execution of said methods results in race conditions"*
- Note that the definition of thread-safety depends on the definition on race condition (recall "A race condition occurs when the result of the computation depends on the interleavings of the operations"), which in turn requires us to reason about interleavings
  - So we need to argue that for interleavings involving these methods there are no race conditions
- *For example, consider an interleaving where two threads execute increment on the same bin (counts[i]) concurrently. This case aims to established thread-safety of the increment method.*
  - *First we establish that because the type of count[i] is AtomicInteger, we know that the operations get() and compareAndSet are atomic (this is enforced at the hardware level)*
  - *When threads increment sequentially, it is trivially race condition free. This corresponds to the interleaving* `T1(get)T1(CAS)T2(get)T2(CAS)` *(the computation ends with count[i] incremented twice)*
  - *When two threads concurrently execute we have that only one CAS operation will succeed* `T1(get)T2(get)T1(CAS)`~~`T2(CAS)`~~ *because T1 updated the value (* ~~`T2(CAS)`~~ *denotes a CAS operation that returns false). So T2 needs to retry resulting in* `T1(get)T2(get)T1(CAS)`~~`T2(CAS)`~~`T2(get)T2(CAS)`*. This interleaving results in count[i] being incremented twice, as required for thread-safety.*
- Similarly for other cases (perhaps not in so much detail as they may be analogous, e.g., getAndClear)
  - Also, note that different methods can execute concurrently, e.g., increment and getAndClear

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"*

- *<u>It is your responsibility to let us know if there are any errors in grading</u>*
  - For instance, missing grades, ungraded assignment, etc.

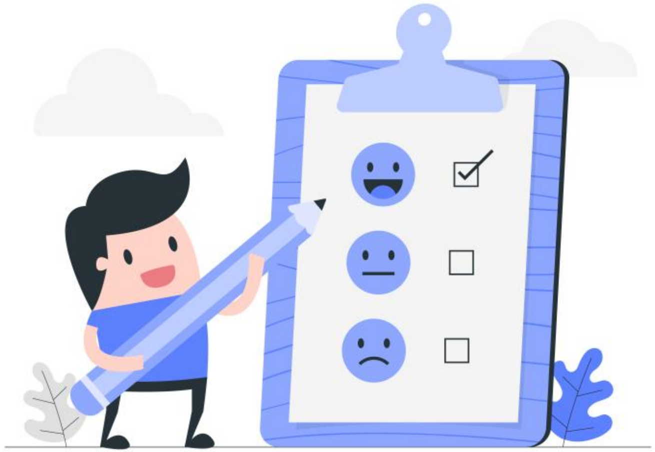
- There will be a final extra deadline in Dec 14th to hand-in assignments that have not yet been approved
  - With no possibility of re-submission and written feedback

Examination – Dates & guidelines

- Exam hand-out: Dec 19th
- Exam hand-in deadline: Dec 20th
- Random fraud control
  - 20% of students will be randomly selected
    – We will publish a list with selected students
  - Performed 30 min after scheduled deadline
    – That is, Dec 20th
  - Takes 30 min
  - Conducted via Zoom
- Exam guidelines in `exam` in the GitHub repository
  - Read the guidelines and send questions for next week

Course Evaluation Survey

Please participate in the course evaluation



Questions ?