Practical Concurrent and Parallel Programming XIII

Lock free coordination

Jørgen Staunstrup, IT University

Friday 2020-11-20

Start at 8:00

Just for fun

Running `TestCountFactors` on my Android phone:

```
final int range = 10000000;
```

Results:

Samsung A20 - Lassen P+
4 cores
`Time in s:  7.24`
`# factors:  37861235`

Lenovo ThinkBook Intel i5
8 cores (hyperthreading)
`Time in s:  2.57`
`# factors:  37861235`

Apologies

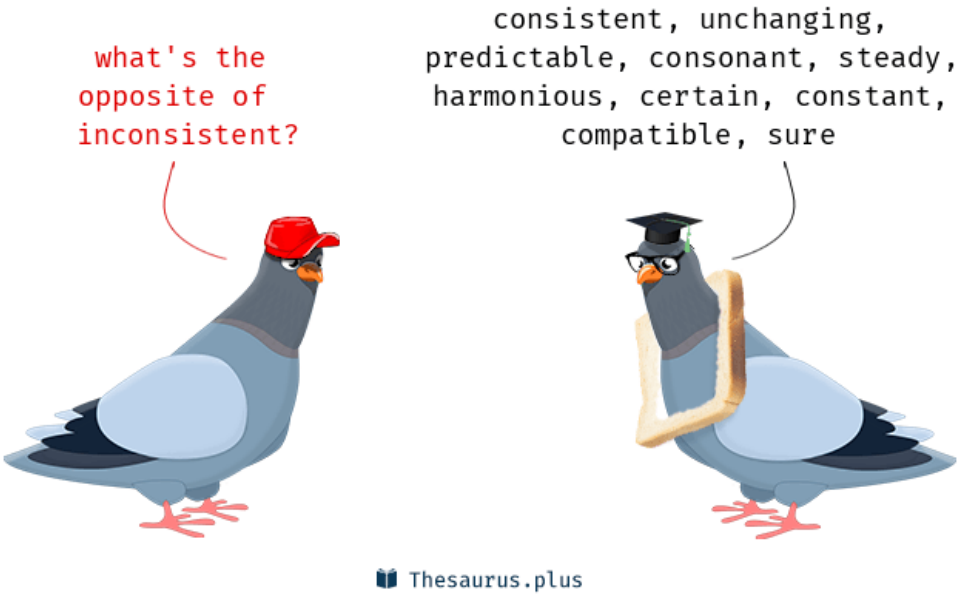
Last week you asked questions about terminology:

concurrency, parallelism, multiprogramming, multithreading, multiprocessing, ...

There is no commonly agreed consistent terminology !! The same is unfortunately the case for many other computer science topics.
What if the same was the case in medicine? elbow, armbender, armlink, ...

Learning and using the agreed terminology is a keep part of becoming a professional in may other fields

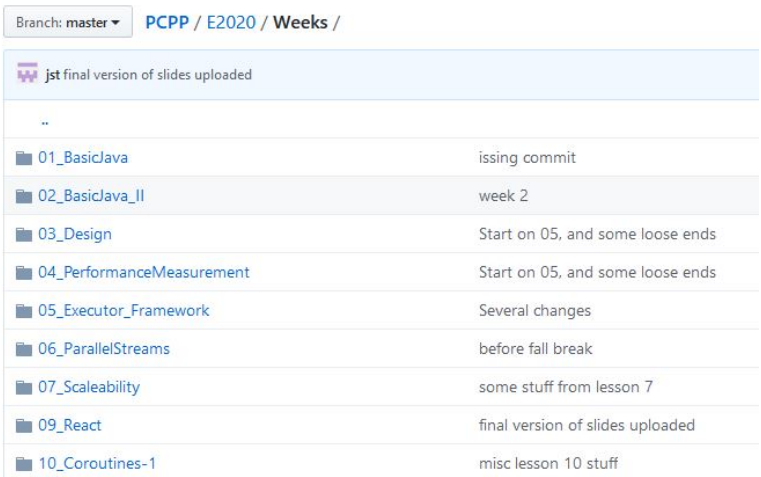
There is a similar terminology question to this weeks material on LearnIT. I will return to this.



Agenda

- Git
- Optimistic concurrency control
- Operational transform
- Linearization
- Realm (MongoDB) database
- Atomicity

File sharing with Git



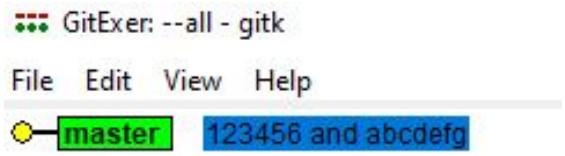
Workflow:

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

Works because Kasper and I modify different files !!

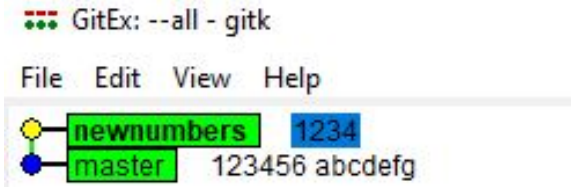
Git merge / rebase

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



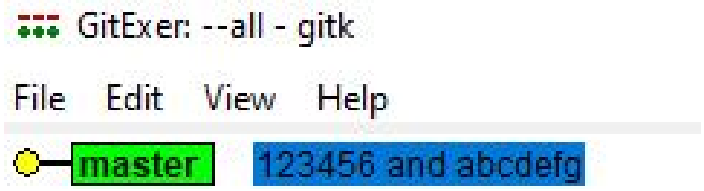

```
git branch newnumbers
git checkout newnumbers
```

change file numbers.txt: 1234

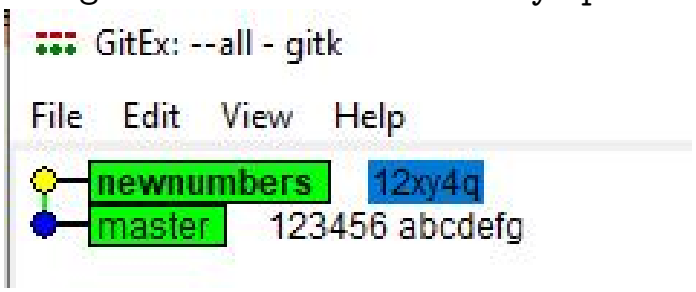



```
git checkout master
git merge newnumbers
Updating dd2289c..a423cf8
  Fast-forward
   numbers.txt | 2 +-
   1 file changed, 1 insertion(+), 1 deletion(-)
```


Git merge/rebase (2)



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.
```

Pessimistic concurrency control

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


Optimistic concurrency control

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

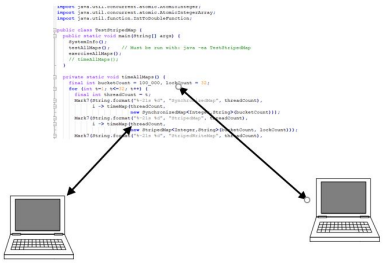
Locking (atomicity) is pessimistic concurrency

Google Wave and Realm (MongoDB) use optimistic concurrency control

Compromise on consistency: *Strong eventual consistency*

Recall discussion about `forEach` in exercise 7.2 (Stribed map)

Concurrent text editing (Google wave)

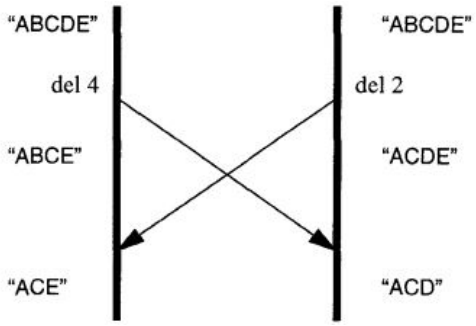


Google wave https://youtu.be/p6pgxLaDdQw

Concurrent editing survided in Google Docs.

Operational transform

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



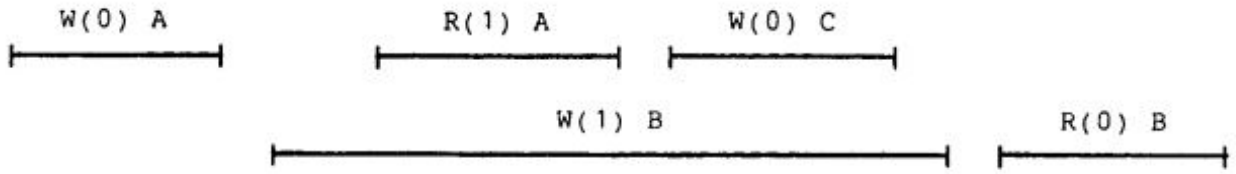
https://youtu.be/3ykZYKCK7AM

Key idea is to find a way to resolve conflicts for all pairs of operations o1 and o2 where: o1;o2 $\neq$ o2;o1

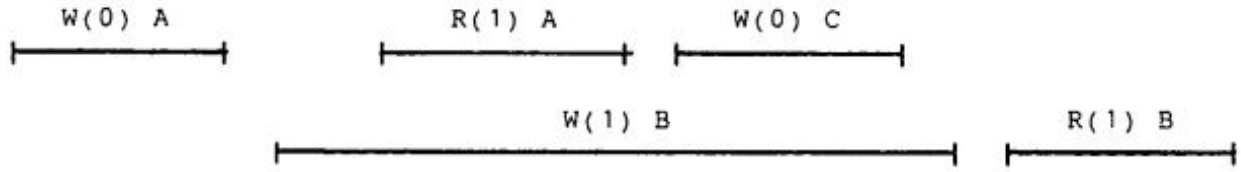
This is not so difficult for text operations like insert and delete.

Consistency

Defining acceptable behavior of shared data.



Is this consistent?



Is this consistent?

LearnIT question

In the paper: Linearizability: A Correctness Condition for Concurrent Objects (WingHerlihyCons.pdf in this weeks reading) the authors write
Unlike alternative correctness conditions such as sequential consistency [31] or serializability [40], linearizability is a local property

Local property ???

Consistency definitions

Strict consistency: *a write to a variable by any thread needs to be seen instantaneously by all other threads.*

Example: a bank account

Note that for some datastructures strict consistency can be ensured even if some operations are not atomic.
Example?

Weak wonsistency: *A write to a variable does not have to be seen instantaneously, however, writes to variables by different threads have to be seen in the same order by all threads.*

Weak wonsistency: (example)

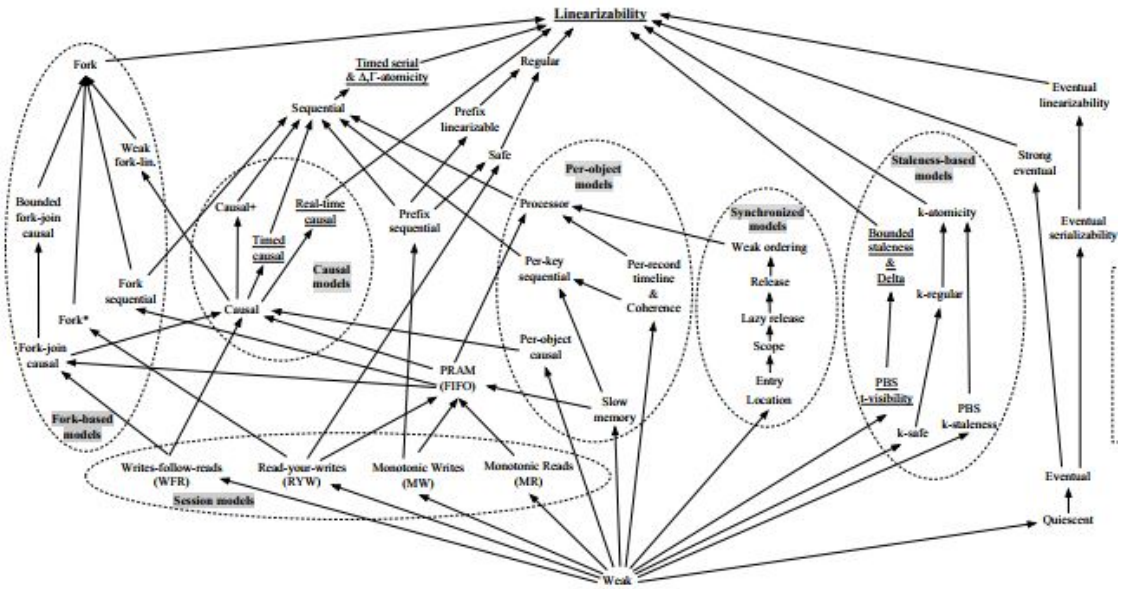
Example: the striped map from week 7

```
public void forEach(Consumer<K,V> consumer) {
  final ItemNode<K,V>[] bs = buckets;
  for (int stripe=0; stripe<lockCount; stripe++)
    synchronized (locks[stripe]) {
      for (int hash=stripe; hash<bs.length; hash+=lockCount) {
        ItemNode<K,V> node = bs[hash];
        while (node != null) {
          consumer.accept(node.k, node.v);
          node = node.next;
        }
      }
    }
}
```

t_1: `forEach(bs[1])` — assume it takes a while
t_2: `remove(bs[0])` and a little later `insert(bs[2])` — before t_1 moves on
Now t_1 will have iterated (forEach) over `bs[0]`, `bs[1]`, `bs[2]`, `...`
!!!

Is this consistent?

Consistency definitions (2)



source: *Consistency in Non-Transactional Distributed Storage Systems* by Paolo Viotti and Marko Vukolic´

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

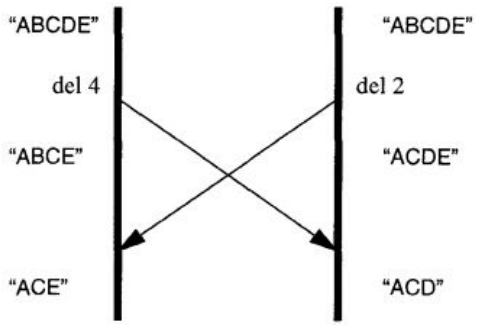
Partition tolerance
The system continues to operate despite an arbitrary number of messages being dropped (or delayed) by the network between nodes

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

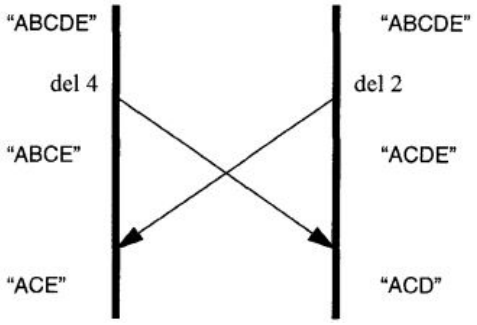
Strong eventual consistency

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 a shared document:



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 In the folder with material for lecture 13).

But what if the operations are more complicated than insert and delete?

E.g. in a database?

Realm database

Mobile apps: network reliability, local storage, and UIs reactive.

- Local storage: on client devices i.e. fast.
- Network reliability: Realm Database is offline-first: you always read from and write to the local database. Synchronizes data with central database in a background thread. The sync protocol resolves conflicts using operational transform.
- Reactive UI: Live objects always reflect the latest data stored.
- 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 files.

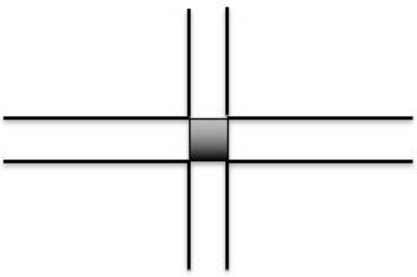
- Changeset: list of write operations to database objects.
- Operational transformation: operational transformation is used to resolve conflicts between changesets from different clients.
- Off-line first: any device may perform offline writes and upload the corresponding changesets when there is network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

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

Atomicity

Throughout the course we have relied on locks to ensure atomicity. Let us take a little closer look at how locks work.

How to implement atomicity



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

How to implement a semaphore/lock on a computer

Assume that the computer hardware offers an atomic exchange operation: `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
```

Modern hardware has such an operation e.g. on Intel `cmpxchg`

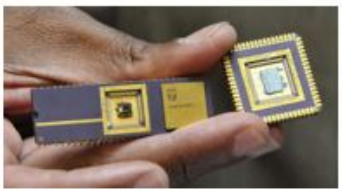
Multi Maren 1980

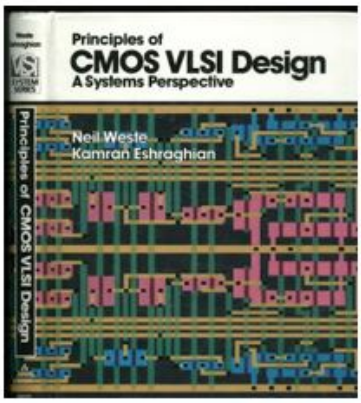
`a <--> b` implemented by a hardware arbiter

Hardware implementation of `a <--> b`

Gust professor at UW Washington 1984-85

Introduced to VLSI (chip) design





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

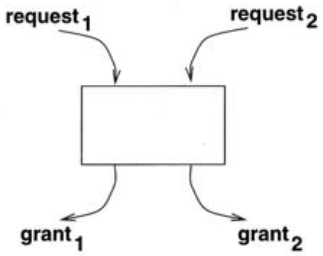
???

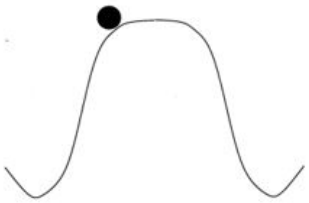
How to implement <--> in hardware?

Clock: step1, step2, step3, ...
a single operation (e.g. `a <--> b`) in each step !!!

BUT what about networked computers?
They have different clocks !!! same challenge with all other i/o

Arbiter





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

Buridans donkey 1230
https://en.wikipedia.org/wiki/Buridan
https://en.wikipedia.org/wiki/Buridan%27s_ass

The end

This (philosophical) detour ends this course.
Next week: Info about examination and course evaluation