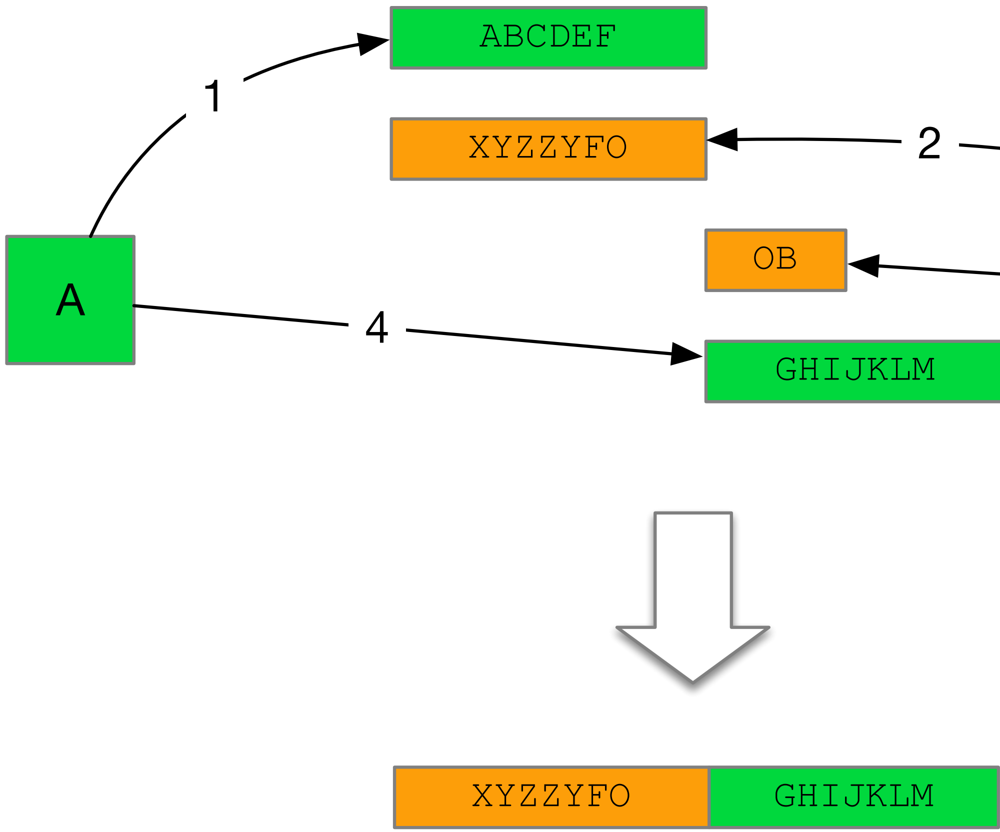
Aalto University
School of Electrical
Engineering

Things never to do, That you might have to do

14.3.2019
Santeri Paavolainen

Why concurrency?

- **Locally: performance**
- **Distributed systems (including microservice architectures) are by definition concurrent systems**
  - Independent and autonomous randomly failing machines communicating over asynchronous and lossy networks
- **Completely sequenced systems can work efficiently**
  - Very limited situations, not often found in real life
  - Still does not solve redundancy
- **Redundant systems (>1 node) always need some distributed coordination**



General problem on any asynchronous sequence of operations that rely on earlier steps

e.g. locking, mutexes, sequencing, …

What are these?

- **Standard concepts used for concurrency control**
  - Prevent concurrent access (mutual exclusion)
  - Coordinate operations (barriers)
  - Restrict resource usage (semaphores) or access type (rw locks)
  - Atomic operations (compare-and-swap)
- **Very important concepts!!!**
  - Critical in creating truly concurrent systems

Why should not be used?

- **Concurrent programming is <u>devilishly difficult</u>**
- **Lot of design in many systems, languages and programming frameworks has gone into <u>hiding concurrency</u>**
  - "Looks mostly sequential" plus exceptions for corner cases: <u>most databases</u> (SQL and NoSQL alike)
  - Parallel single-threaded applications with sequential interfaces: <u>Erlang</u>, <u>AKKA</u>, all "event loop" frameworks
  - Idempotent or functional interfaces: <u>anything with retries</u> (queues, WfS, Hadoop)
- **<u>Avoid explicit concurrency management if possible!</u>**

Ways to avoid explicit concurrency control

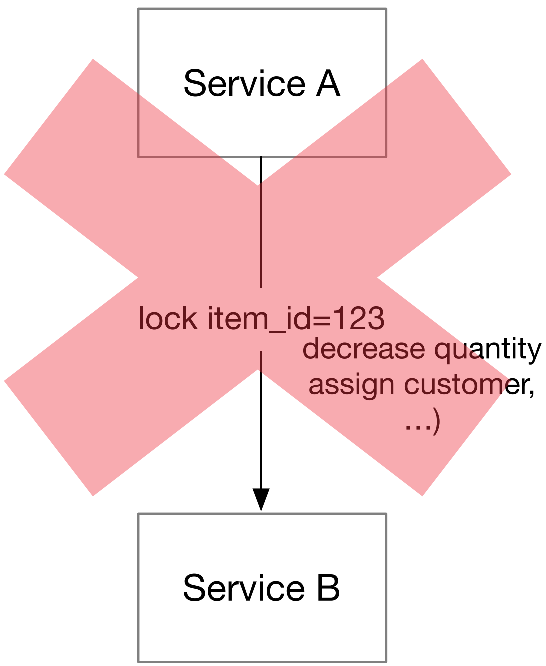
- **Idempotent or functional interfaces**
  - Doing the same operation twice has no effect
  - Keep track of progress (database?), return result of first operation
  - Just re-do but ensure always same result (like image thumbnails): functional e.g. same input = same output
- **Make it someone else's problem**
  - Although you'll have to deal with corner cases anyway
    - *Usually simpler, though – db rollback? → 503 Service Temporarily Unavailable or 500 Internal Server Error and hope upstream retries*
- **Use languages and frameworks that simplifies things**
  - Erlang, AKKA, Elm, ...: Explicit message-passing across actors, no shared state
  - JavaScript: Everything is asynchronous and callback-based (promises etc.)
    - *Brings its own problems*
- **Single sequencer aka leader (but now have leader selection problem…)**
- **Write excessively and clean up later**
  - Unique filenames in S3, FCFS for DB update, later GC unused files

However …

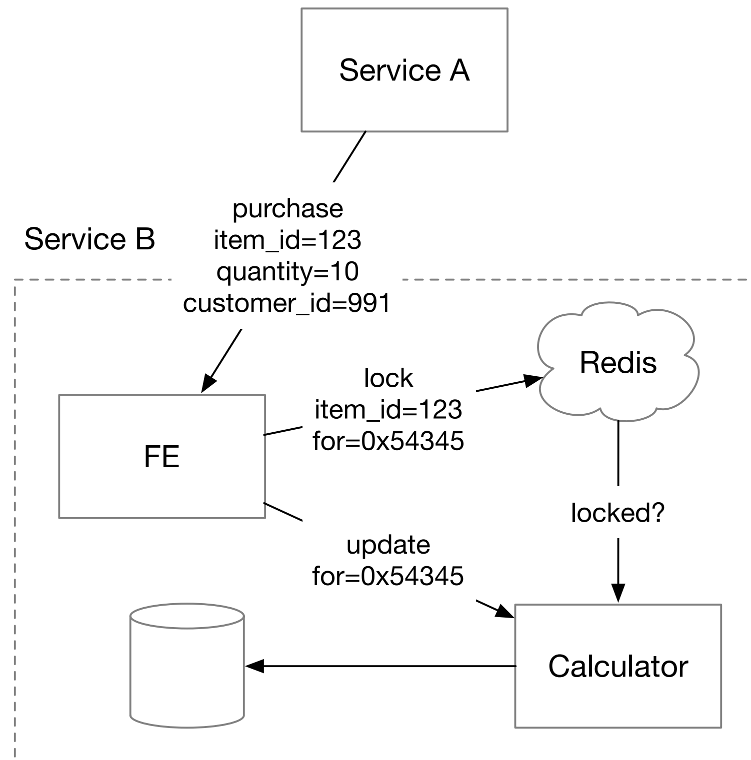
- **For some problems there may not be a good ready-to-go solution**
  - Performance impact
  - Unacceptable complexity

- **NEVER EVER ACROSS MICROSERVICES!**
  - Keep any explicit concurrency control inside your service's boundary!
  - (Of course there are corner cases … there always are)

NO


Service A
lock item_id=123
decrease quantity
assign customer,
...)
Service B

YES
(maybe)


Service A
purchase
item_id=123
quantity=10
customer_id=991
Service B
FE
lock
item_id=123
for=0x54345
Redis
locked?
update
for=0x54345
Calculator

If you really insist

- **Use some distributed system to start with**
  - Memcache, Redis, etcd, ZooKeeper (esp. Curator)
  - Understand network partitioning behavior! (CAP again)
- **Atomic operations**
  - Increment (increment and return value)
  - Compare-and-swap (CAS), e.g. read old value, CAS, if fails, retry
- **CAS basis for**
  - Leader election: Try to become leader, fails if someone already is
  - Lock: CAS 0 → 1, if fail, wait and retry
- **Always consider what happens if "client" dies**
  - TTL, lock refreshing, check before commit, … (corner cases)

Still

- **Try to avoid**
- **Try to rewrite the problem so won't have to do**
  - Or use less error-prone primitives (even DB update and transaction abort are useful)
- **Example: Users uploading files to S3, thumbnail workflow**
  - Thumbnail: user-id / filename_resolution → multiple writes?
  - Thumbnail: hash(filename)_resolution → same content?
  - Thumbnail: monotonic sequence (atomic incr) → contention
  - Thumbnail: random string (locally) → no conflict
    - *With sufficiently large string and good source of randomness unlikely conflict*

Too much time on something you should not be doing…

Aalto University
School of Electrical
Engineering