Conclusion of the lesson

- We have now arrived at the end of this lesson on how to program a distributed system with weak synchronization
  - Synchronization: eventual node-to-node communication
  - Consistency model: strong eventual consistency
- We have shown three important applications of this idea
  - CRDTs, Lasp, and Antidote
- We are convinced that the approach has a promising future
  1. Edge computing
  2. Synchronization-free services

LIGHTKONE

Different consistency models

- Strong consistency: the system obeys linearizability
  - Easy to program but can be very inefficient
- Eventual consistency: the system can support many concurrent operations « in flight »
  - Efficient execution but hard to program because of potential conflicts
- Convergent consistency: the system can support many concurrent operations, plus it obeys strong eventual consistency
  - Both efficient execution and easy to program
  - We cannot do CAP but we can do AP + $\Diamond$C = available, partition-tolerant, and convergent

LIGHTKONE

1. Edge computing

- Distributed systems « at the edge » are omnipresent
  - Internet of Things and mobile devices far outnumber data center nodes
  - Edge networks are highly dynamic for computation and communication
- Synchronization-free programming is well-matched to edge systems
  - Convergent computation layer with a hybrid gossip communication layer
- It is naturally tolerant to faults in edge systems
  - Partitions
  - Message loss and reordering
  - Nodes going offline and online
  - Node crashes

Slows down convergence

Tolerant as long as state exists on at least one node

LIGHTKONE

2. Synchronization-free services (1)

Today

- We are using CRDTs as the basis for a programming framework and a transactional database
  - Lasp and Antidote

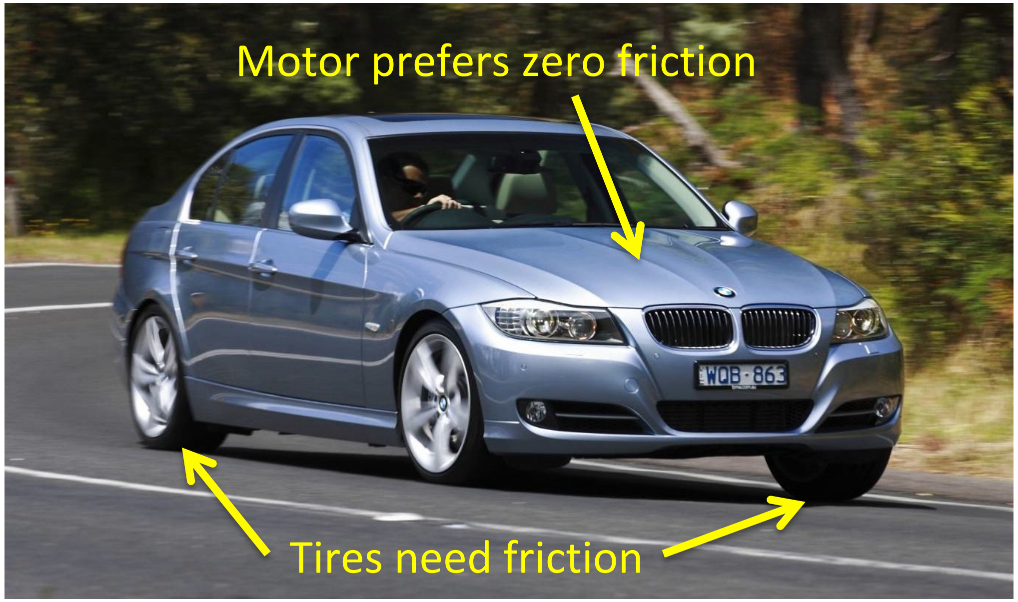
Future

- But the synchronization-free approach can be applied much more generally
  - Let me introduce this with a parable...

LIGHTKONE

Parable of the car (1)

Synchronization is like friction

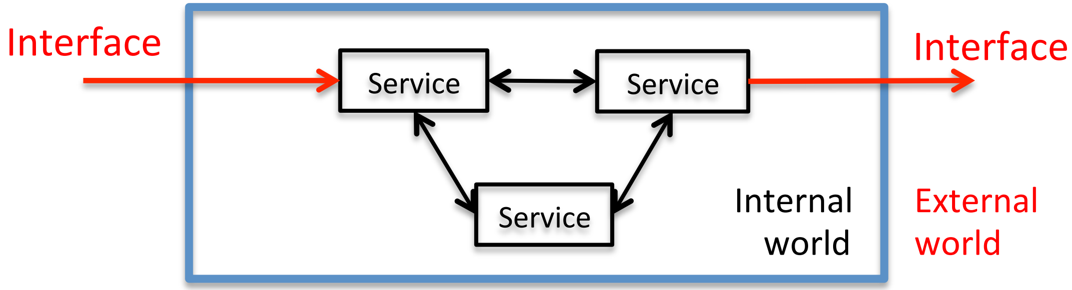


- Like friction, synchronization is both desirable and undesirable
- Consider a car on a highway
- The car needs friction: it moves because the tires grip the road
- But the car's motor avoids friction: the motor should be as frictionless as possible, otherwise it will heat up and wear out

LIGHTKONE

Parable of the car (2)

Distributed computing system



Consider a distributed computing system made of services connected together

- Synchronization is only needed at the system's interface with the external world

Friction is only needed externally, so the tires can grip the road

- Internally the services should avoid synchronization

Internally, the motor avoids friction

LIGHTKONE

Synchronization-free services (2)

- The system has a synchronization boundary
  - Inside this boundary, all services are synchronization-free
  - Synchronization is only needed at the boundary
- Services are inside this boundary
  - Internal state of each service obeys SEC
  - Service API has asynchronous streams, in and out

LIGHTKONE

Going forward!

- In this lesson have introduced the basic concepts of programming with weak synchronization
  - We presented data structures (CRDTs), a programming framework (Lasp), and a transactional database (Antidote)
- Our future work will focus on edge computing and synchronization-free services
  - LightKone H2020 project (lightkone.eu)
  - This project is working on both Lasp and Antidote

LIGHTKONE

Lasp and Antidote resources

- Documentation
  - https://lasp-lang.org
  - http://antidotedb.org
- Code repository
  - https://github.com/lasp-lang
  - https://github.com/SyncFree/antidote

LIGHTKONE

Bibliography

- Marc Shapiro, Nuno Preguiça, Carlos Baquero, and Marek Zawirski. *Conflict-free replicated data types*. Technical Report RR-7687. INRIA (July 2011).
- Christopher Meiklejohn and Peter Van Roy. Lasp: A language for distributed, coordination-free programming. In *PPDP*. ACM, 184–195 (2015).
- SyncFree: Large-scale computation without synchronisation. European FP7 project, 2013–2016. syncfree.lip6.fr
- LightKone: Lightweight computation for networks at the edge. European H2020 project, 2017–2019. lightkone.eu
- Deepthi Devaki Akkoorath, Alejandro Z. Tomsic, Manuel Bravo, Zhongmiao Li, Tyler Crain, Annette Bieniusa, Nuno M. Preguiça, and Marc Shapiro. Cure: strong semantics meets high availability and low latency. In *ICDCS*. 405–414 (2016).
- Peter Bailis, Aaron Davidson, Alan Fekete, Ali Ghodsi, Joseph M. Hellerstein, and Ion Stoica. Highly available transactions: virtues and limitations. In *PVLDB 7(3)*. 181–192 (2013).

LIGHTKONE