Replication and Consistency Models

Pedro F. Souto (pfs@fe.up.pt)

November 8, 2021

Data Replication

Replicate data at many nodes

Performance local reads

Reliability no data-loss unless data is lost in all replicas

Availability data available unless all replicas fail or become unreachable

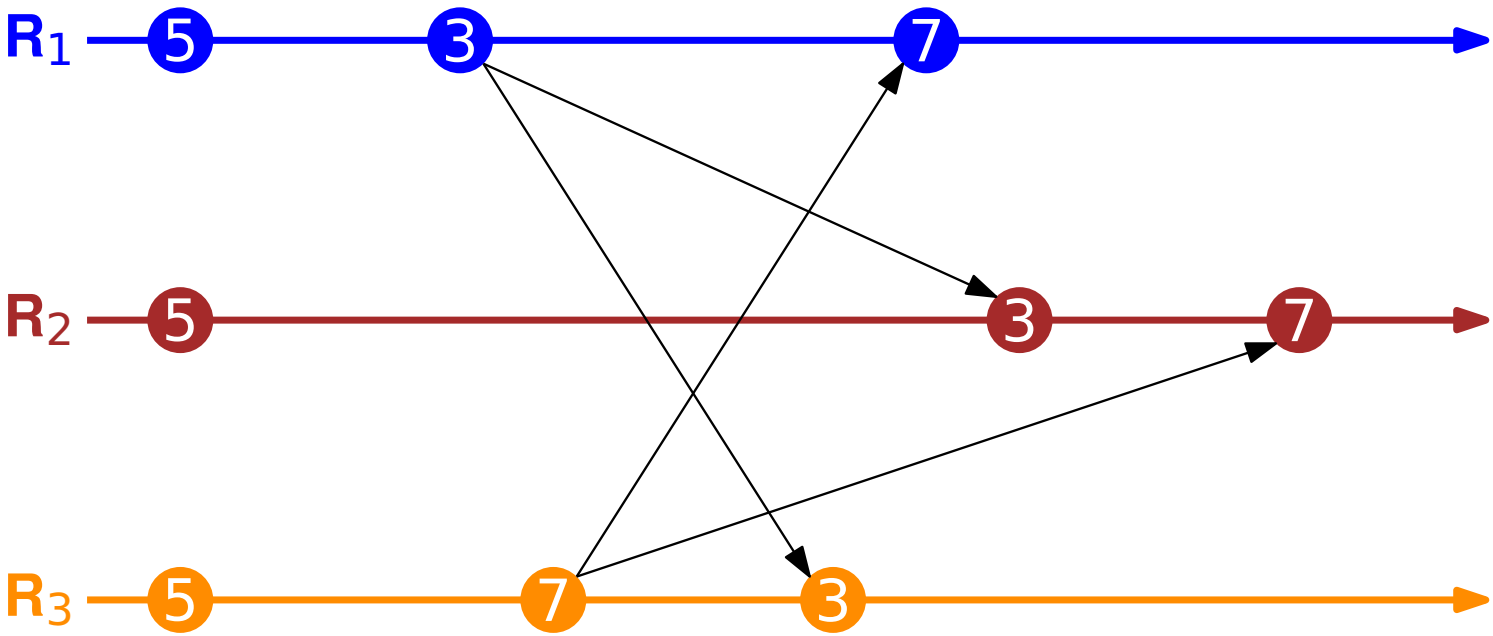
Scalability balance load across nodes for reads

Upon an update

- Push data to all replicas
- Challenge: ensure **data consistency**

Conflicts

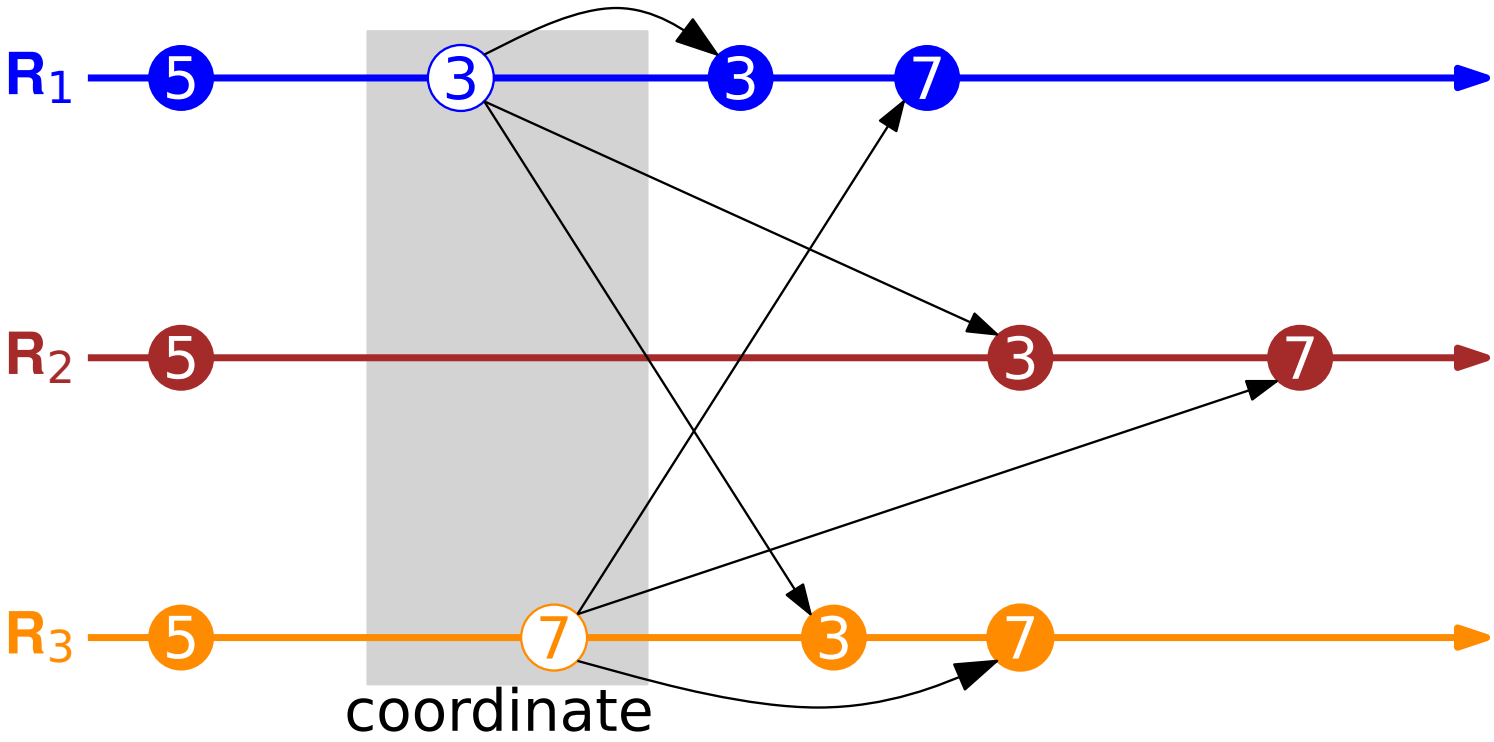
Observation Updating at different replicas may lead to different results, i.e. **inconsistent** data



Strong Consistency

All replicas execute updates in the same order

- Deterministic updates: **same initial state** leads to **same result**



Actually, total order is not enough: it must be sensible

Strong Consistency Models

Sequential Consistency

Serializability

Linearizability

Sequential Consistency Model (Lamport 79)

Definition An execution is **sequential consistent iff** it is identical to a sequential execution of all the operations in that execution such that

- all operations executed by any thread, appear in the order in which they were executed by the corresponding thread

Observation This is the model provided by a multi-threaded system on a uniprocessor

Counter-example Consider the following operations executed on two replicas of variables x and y, whose initial values are 2 and 3, respectively

```
Répl. 1          Répl. 2
(2,3)            (2,3)          /* Initial values */
x = y + 2;       y = x + 3;
(5,5)            (5,5)          /* Final values */
```

If the two operations are executed sequentially, the final result cannot be $(5, 5)$

Sequential Consistent Replication Protocol

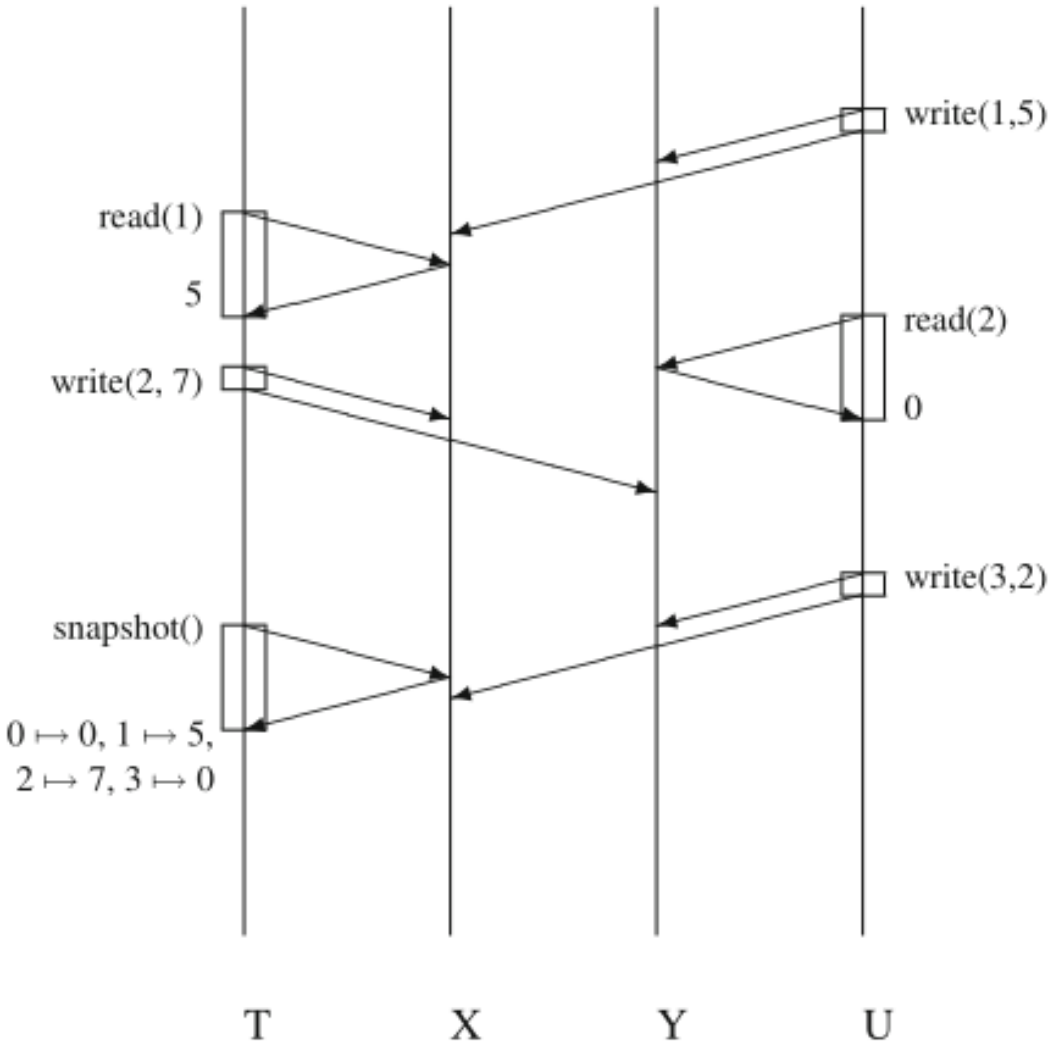
Data type array of 4 elements

- **Read(a)** read value of array's element/index *a*
- **Write(a, v)** write value *v* to array's element/index *a*
- **Snapshot()** read all values stored in the array

Protocol

- **Read** reads from one replica
- **Write** writes to all replicas in same order. Writes have no reply: return after sending the write request messages to all replicas
- **Snapshot** reads from one replica

Sequential Consistent Execution



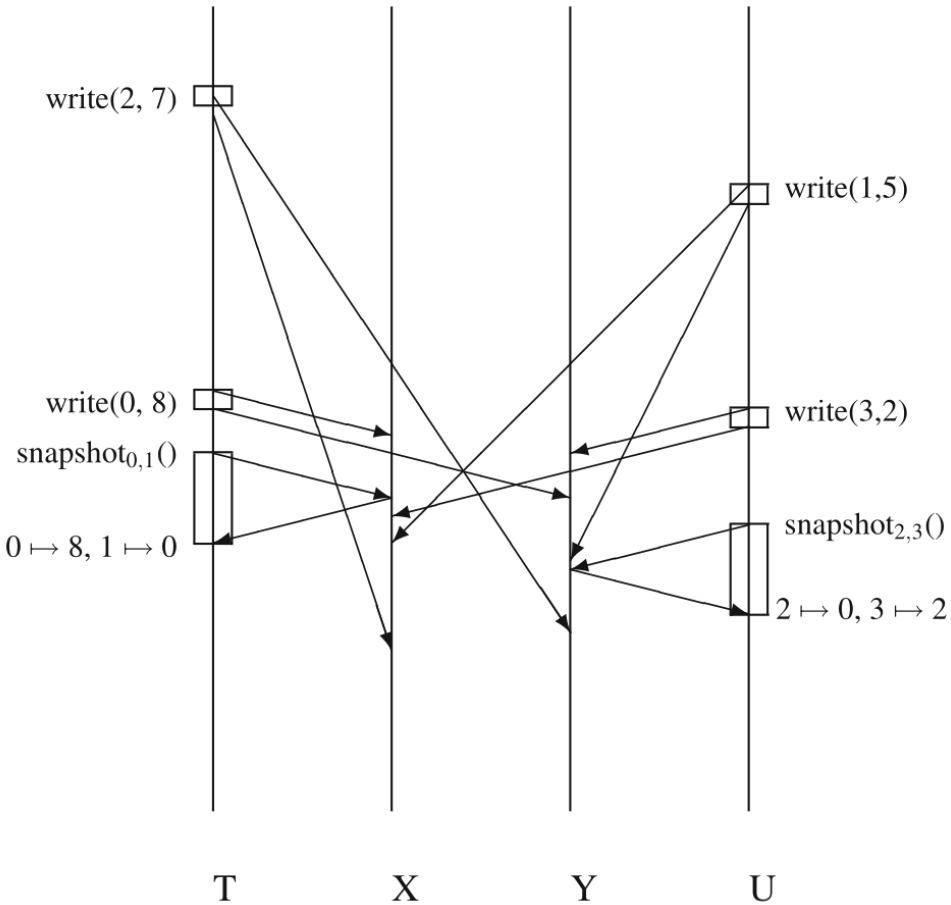
src: Fekete and Ramamritham 09

1. *write*(1, 5)
2. *read*(1)
3. *read*(2)
4. *write*(2, 7)
5. *snapshot*()
6. *write*(3, 2)

What other ordering would be possible?

Sequential Consistency Is Not Composable

- Consider two sub-arrays, each of 2 elements;
- Assume the same algorithm to replicate each of the sub-arrays, and thus ensure sequential consistency on each array
- The combined execution may not be sequential consistent



src: Fekete and Ramamritham 09

First sub-array elements 0 and 1

1. *write*$(0, 8)$
2. *snapshot*$_{0,1}()$
3. *write*$(1, 5)$

Second sub-array elem. 2 and 3

1. *write*$(3, 2)$
2. *snapshot*$_{2,3}()$
3. *write*$(2, 7)$

Can you merge these two orders into a single order such that it is sequential consistent?

Linearizability (Herlihy&Wing90)

Definition An execution is **linearizable** iff it is **sequential consistent** and

- if op_1 **occurs before** op_2, according to one **omniscient observer**, then op_1 appears before op_2

Assumption Operations have:

start time
finish time

measured on some global clock accessible to the omniscient observer

- $op1$ **occurs before** $op2$, if op_1's finish time is smaller than that op_2's start time
- If op_1 and op_2 overlap in time, their relative order may be any

Replication Protocol For Linearizability

Data type array of 4 elements

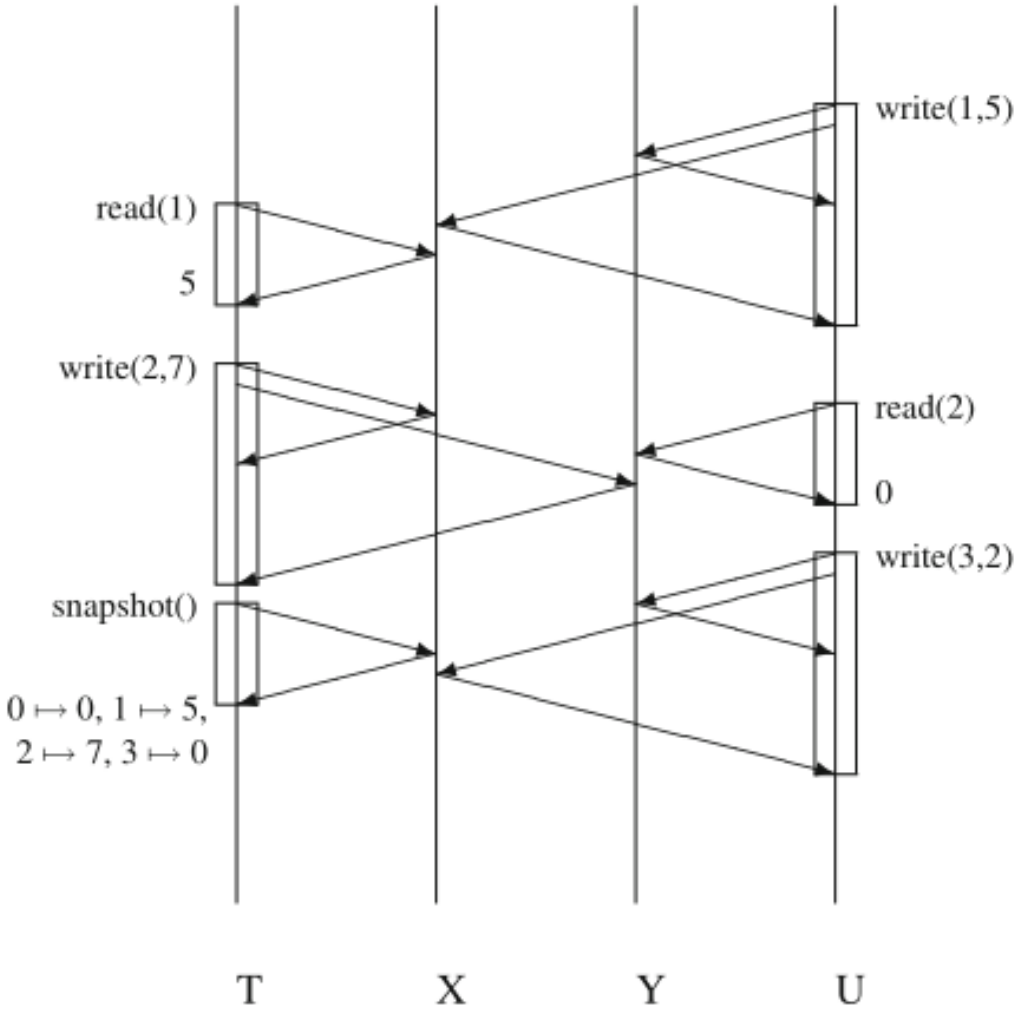
- Read(a) read value of array's element/index *a*
- Write(a, v) write value *v* to array's element/index *a*
- Snapshot() read all values stored in the array

Protocol

- Read reads from one replica
- Write writes to all replicas in same order. **Wait for ack from all replicas before returning**
- Snapshot reads from one replica

Guaranteeing linearizability usually requires more synchronization

Linearizable Execution



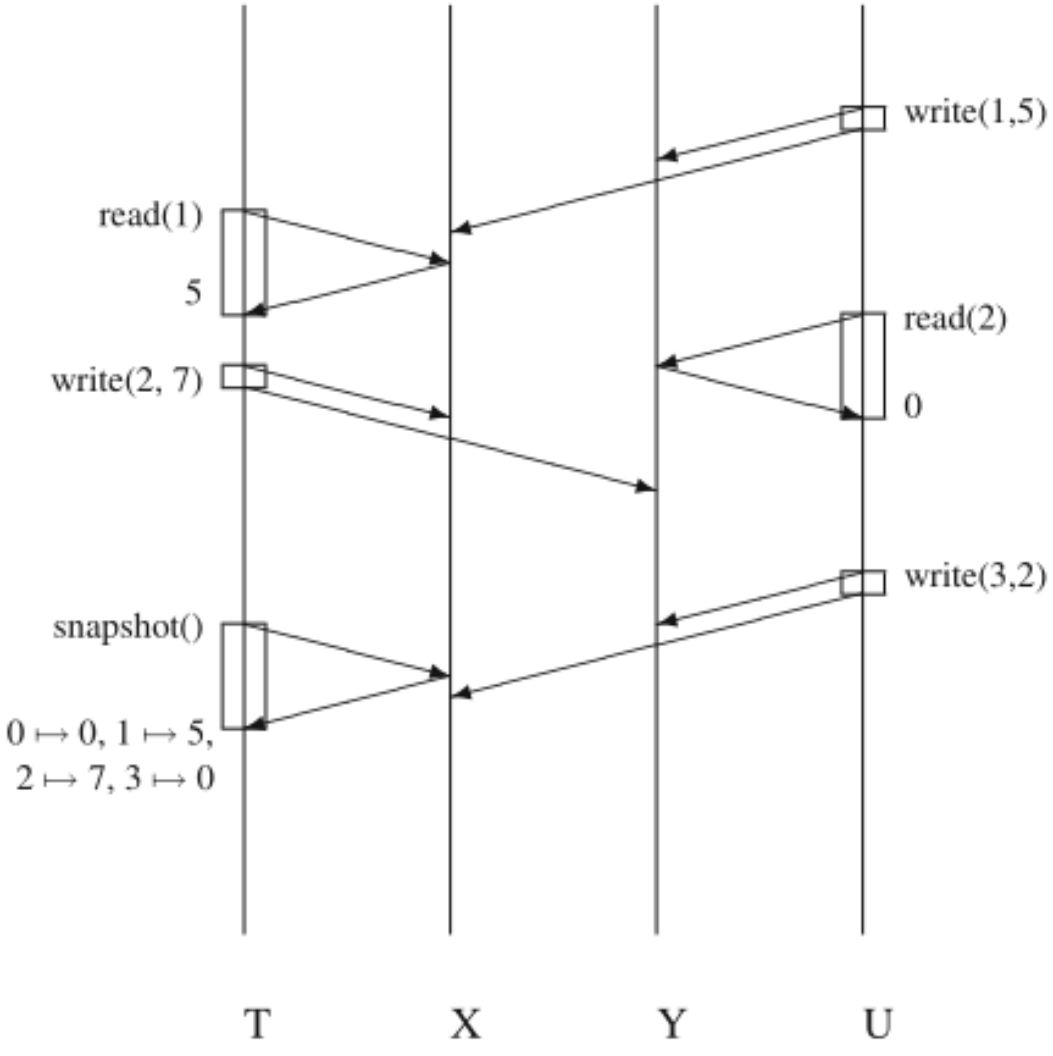
src: Fekete and Ramamritham 09

1. *write*(1, 5)
2. *read*(1)
3. *read*(2)
4. *write*(2, 7)
5. *snapshot*()
6. *write*(3, 2)

Is the other sequentially consistent order also linearizable?

Sequential Consistency vs. Linearizability

Sequential consistent replication protocol execution



src: Fekete and Ramamritham 09

1. *write*(1, 5)
2. *read*(1)
3. *read*(2)
4. *write*(2, 7)
5. *snapshot*()
6. *write*(3, 2)

Is this linearizable?

One-copy Serializability (Transaction-based Systems)

Definition The execution of a set of transactions is **one-copy serializable iff** its outcome is similar to the execution of those transactions in a **single** copy

Observation 1 Serializability used to be the most common consistency model used in transaction-based systems

- DB systems nowadays provide weaker consistency models to achieve higher performance

Observation 2 This is essentially the sequential consistency model, when the operations executed by all processors are transactions

- The isolation property ensures that the outcome of the concurrent execution of a set of transactions is equal to some sequential execution of those transactions

Observation 3 (Herlihy . . . sort of) Whereas

Serializability Was proposed for databases, where there is a need to preserve complex application-specific invariants

Sequential consistency Was proposed for multiprocessing, where programmers are expected to reason about concurrency

Further Reading

- Fekete A.D., Ramamritham K. (2010) *Consistency Models for Replicated Data*. In: Charron-Bost B., Pedone F., Schiper A. (eds) Replication. Lecture Notes in Computer Science, vol 5959. pp. 1-17
- van Steen and Tanenbaum, *Distributed Systems, 3rd Ed.*
  - Section 4.3 *Message-oriented communication*
- Ion Stoica & Ali Ghodsi, *CRDTs and Coordination Avoidance*, Lecture 8, cs262a, UC Berkeley, February 12, 2018