Confiabilidade de Sistemas Distribuídos
Dependable Distributed Systems

DI-FCT-UNL, Henrique Domingos, Nuno Preguiça

Lect. 8
Use Cases

2015/2016, 2nd SEM

MIEI
Mestrado Integrado em Engenharia Informática

BYZANTIUM
Efficient Middleware for Byzantine Fault Tolerant Database Replication

Rui Garcia : CITI / DI - FCT - Universidade Nova de Lisboa
Rodrigo Rodrigues : MPI-SWS
Nuno Preguiça : CITI / DI - FCT - Universidade Nova de Lisboa

Databases and non fail-stop faults

Database systems are central in many software infrastructures

Database systems incur in non fail-stop faults

- Software bugs
  - Large fraction of non fail-stop bugs
- Hardware faults
- Malicious intrusions
- Incorrect configurations

Goals

Middleware for database replication

- Tolerate non-fail stop faults (Byzantine fault model)
- No centralized component
- Performance
  - Circumvent expensive BFT protocols when possible
  - Exploit Snapshot Isolation

Outline

Motivation

Background

Basic solution

The devil is in the details

- Avoiding deadlock
- Improving read-only transactions

Final remarks

Background: snapshot isolation (SI)

A transaction is processed as follows:

- Begin: get database snapshot
- Read/write: execute in snapshot
- Commit: abort if write-write conflict

Properties:

- A read-only transaction does not block nor abort
- No read-write conflicts - increased concurrency

Addressing Byzantine Faults

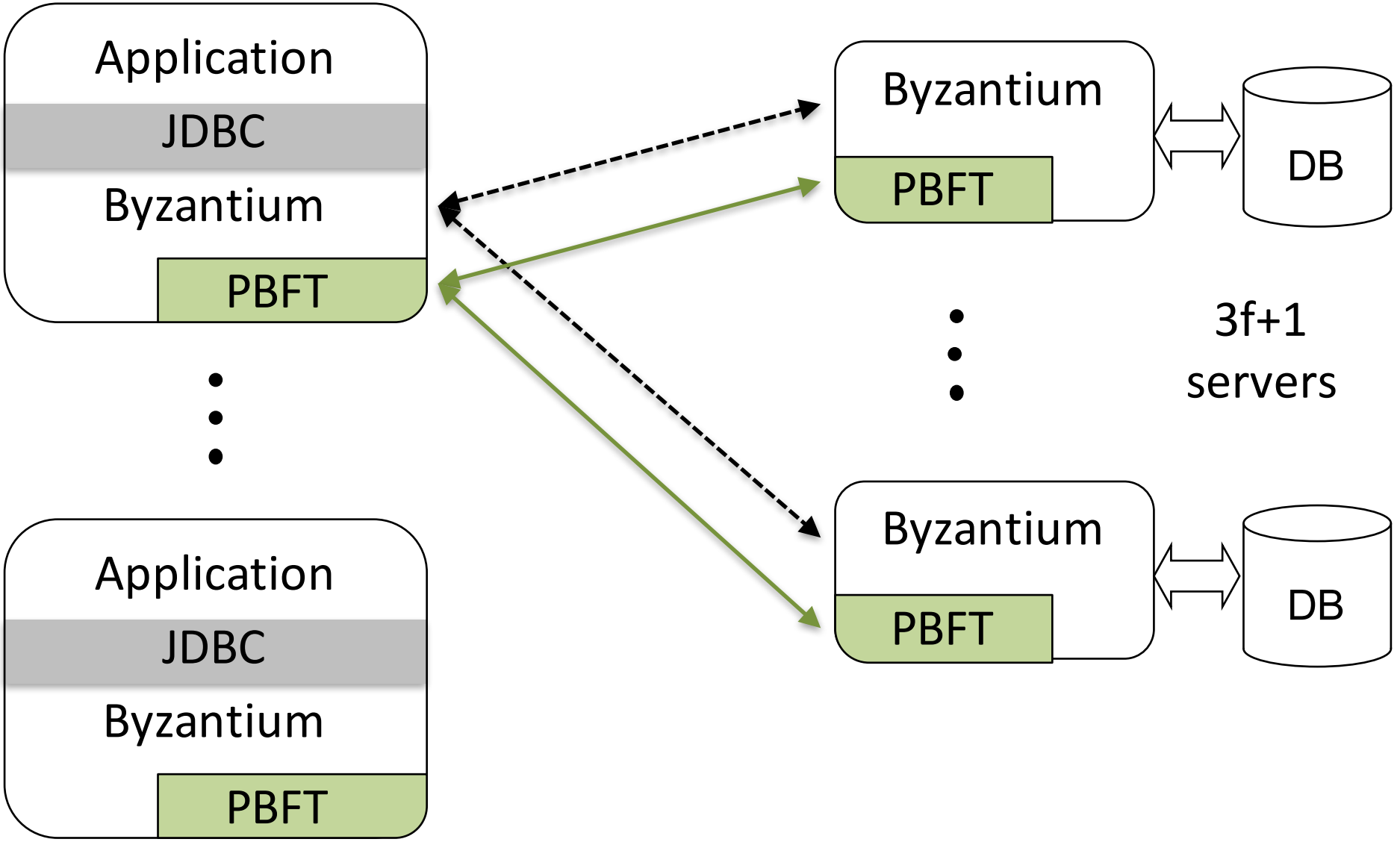
Byzantine Fault Tolerant (BFT) systems

- Tolerate arbitrary faults
- Good performance (batching, speculation, etc.)

State-machine BFT replication

- Replicate arbitrary deterministic service
- All replicas agree on operation ordering
- All replicas execute one operation at a time

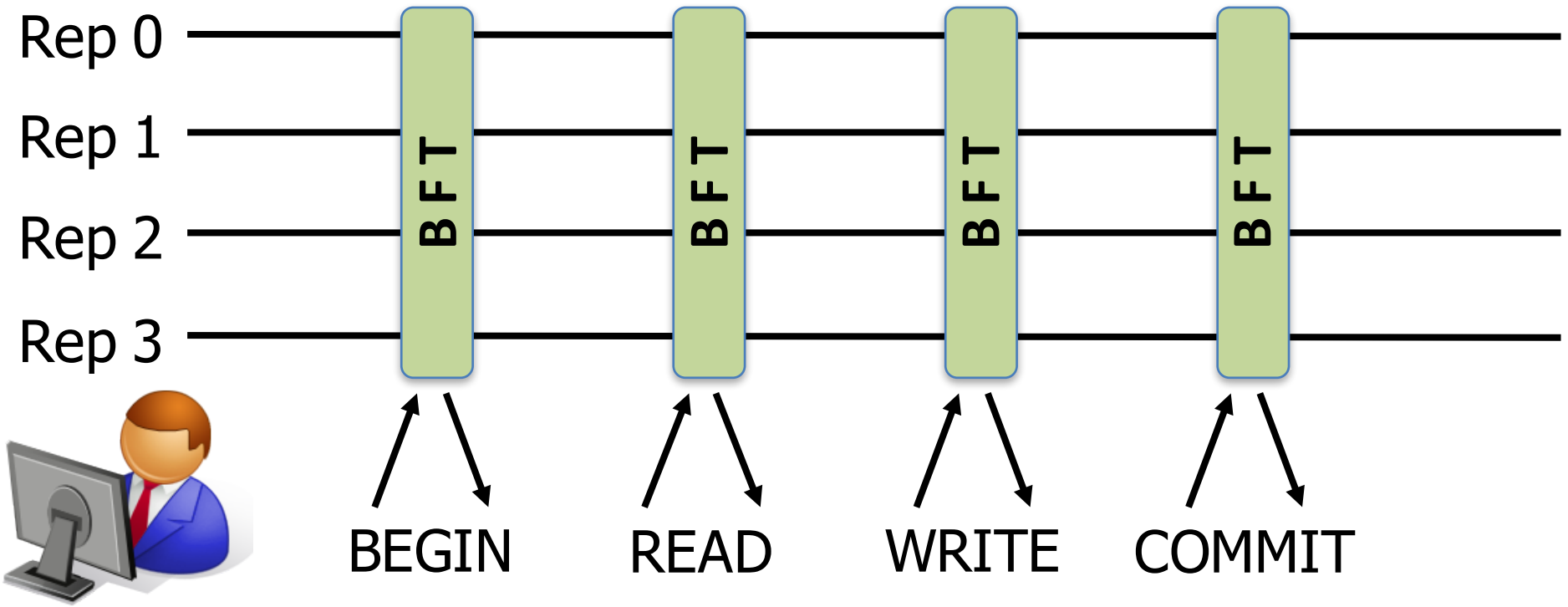
Byzantium Architecture



Mapping transactions and state-machine BFT

Each DB operation as one BFT operation

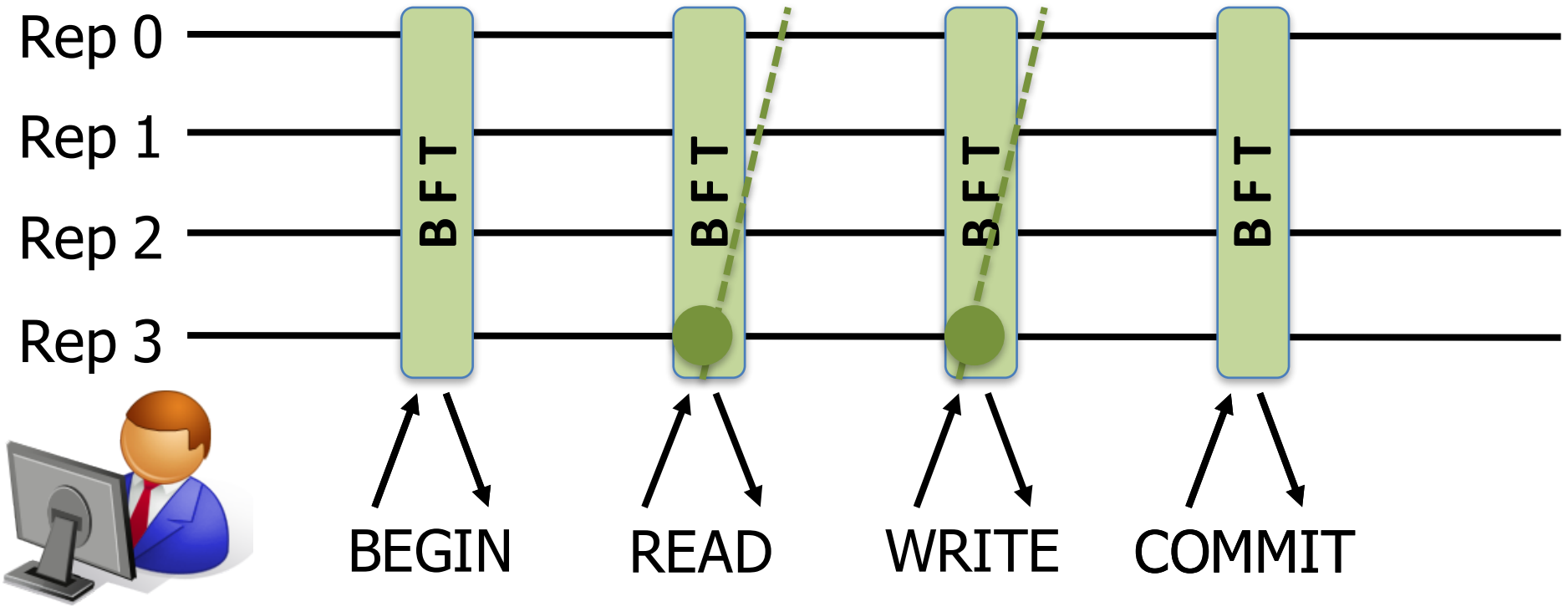
- Limits concurrency on database servers
- BFT overhead for each operation



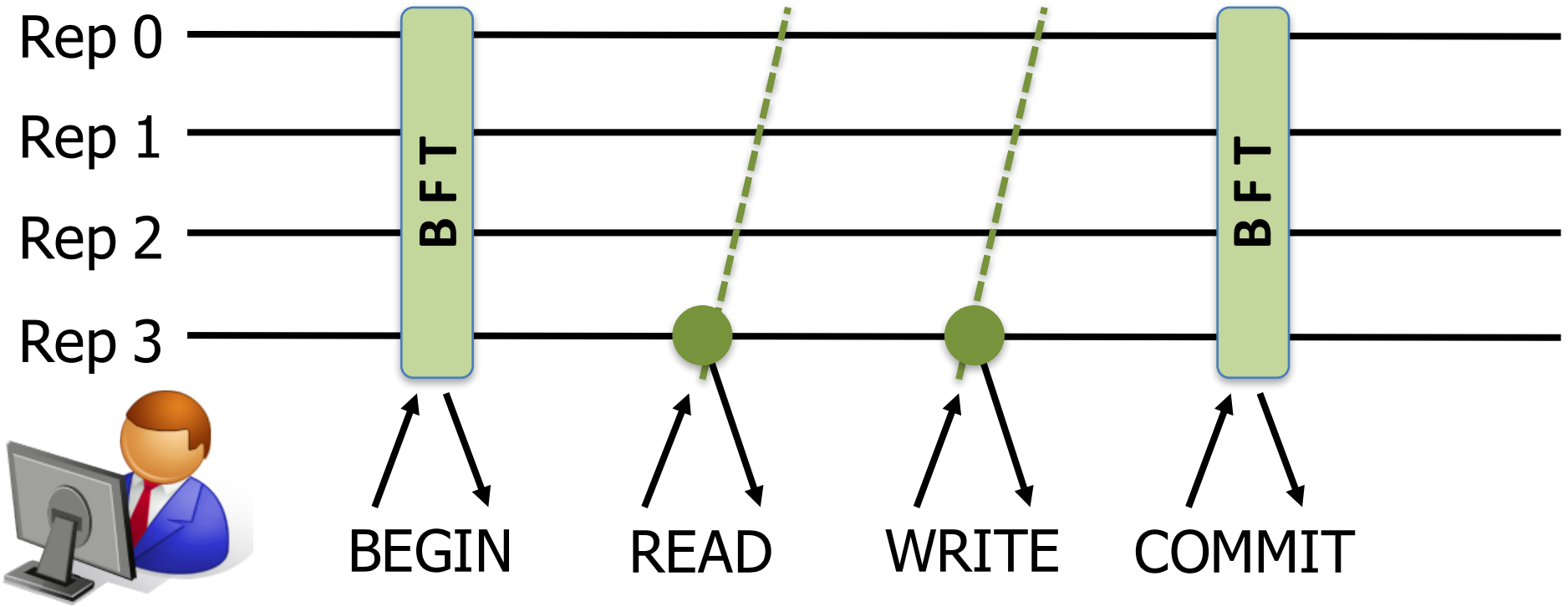
Mapping transactions and state-machine BFT

Our key idea: minimize the number of BFT operations

- Operations execute concurrently
- BFT overhead only for a small fraction of operations

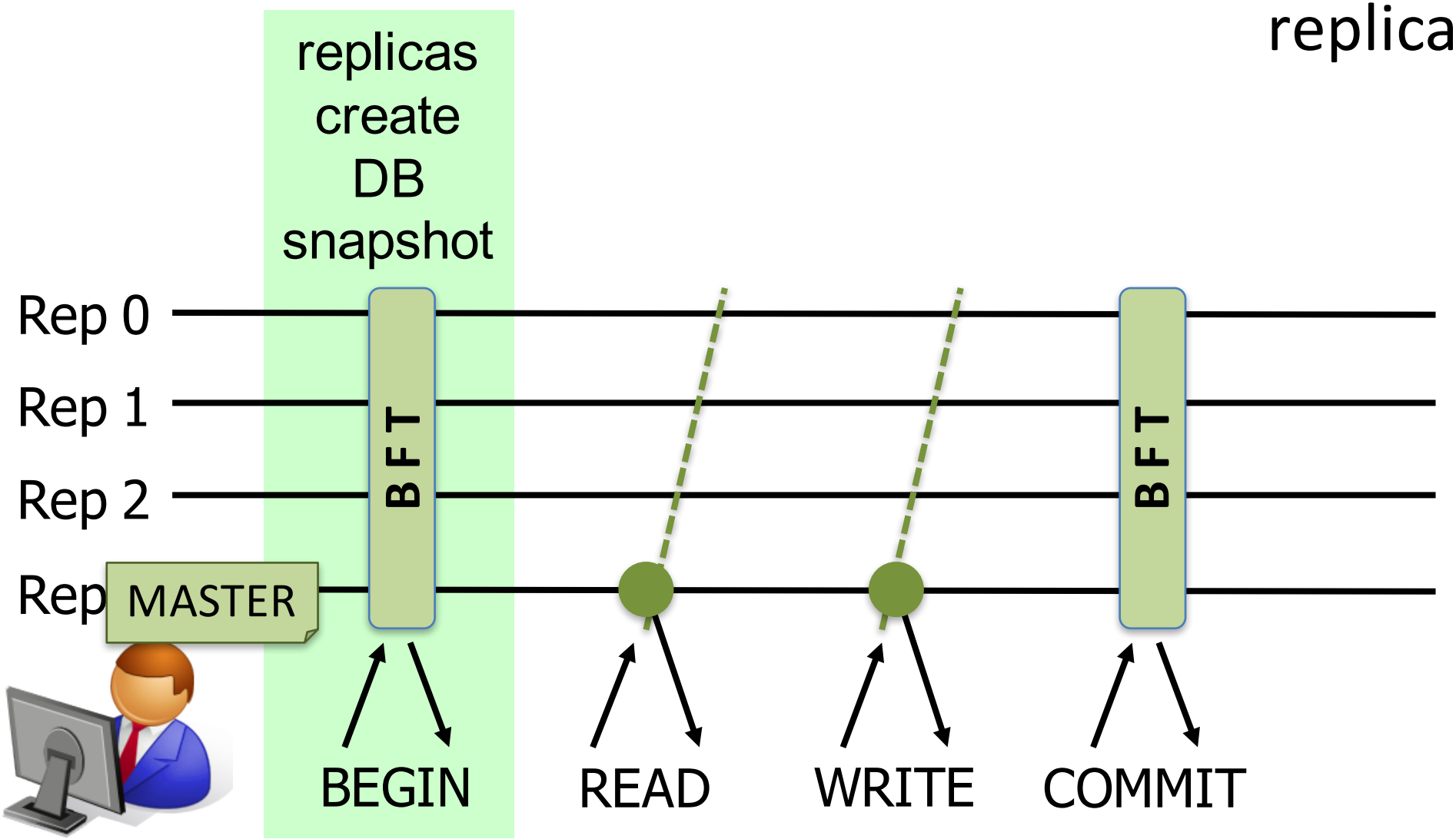


Basic solution



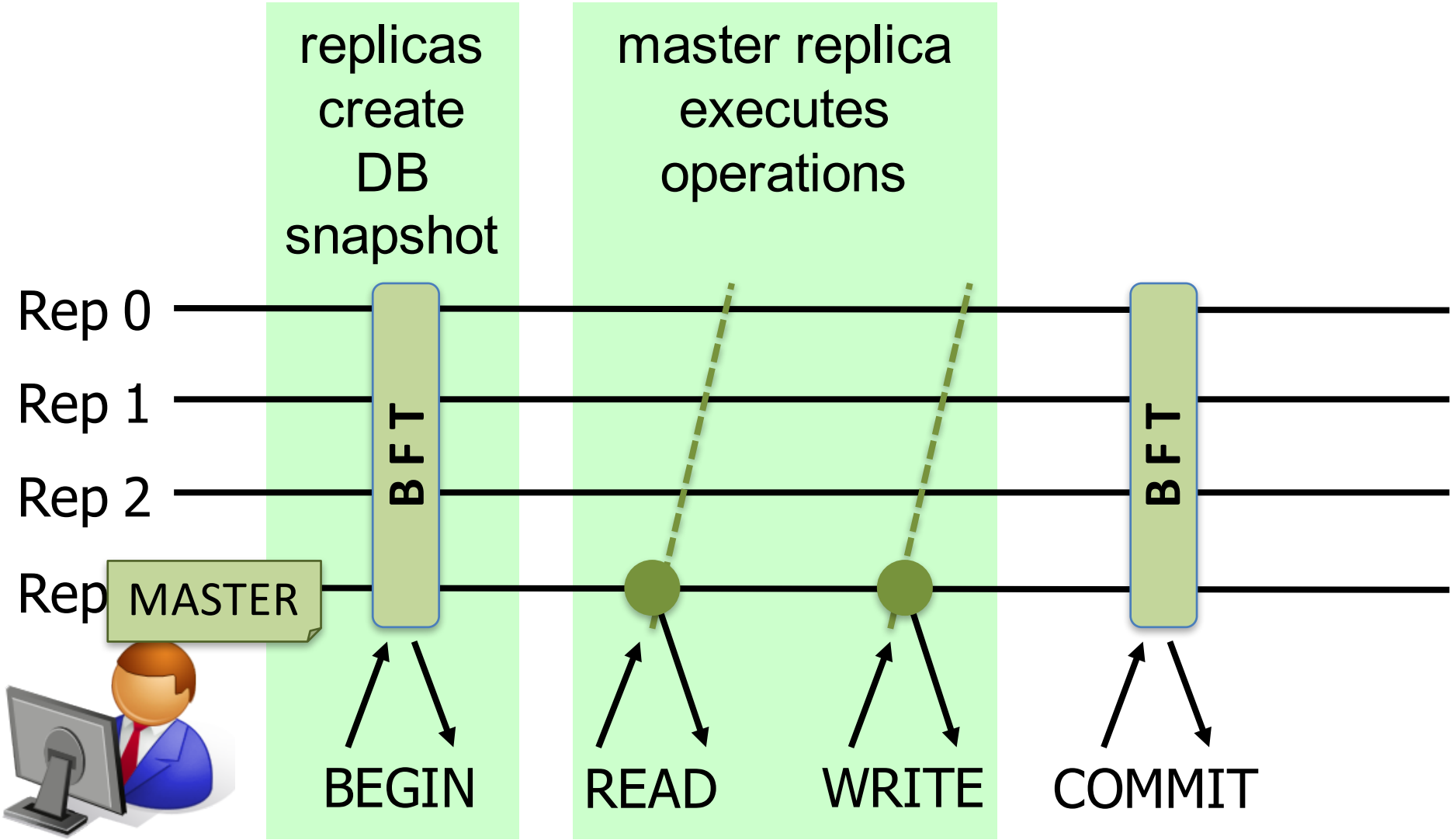
Basic solution

Transaction must execute in the same state in all replicas



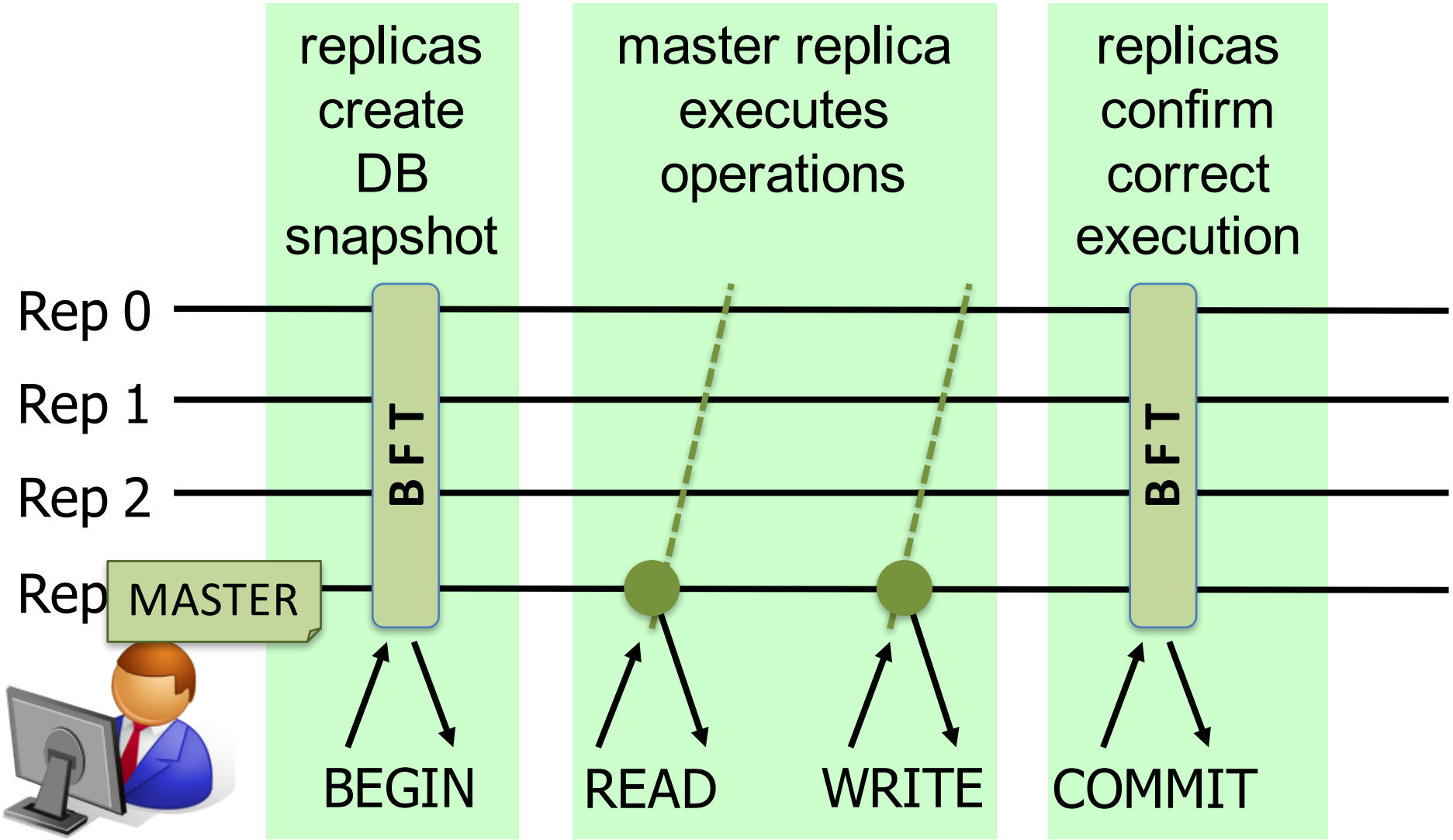
Basic solution

Operations execute tentatively in a master replica



Basic solution

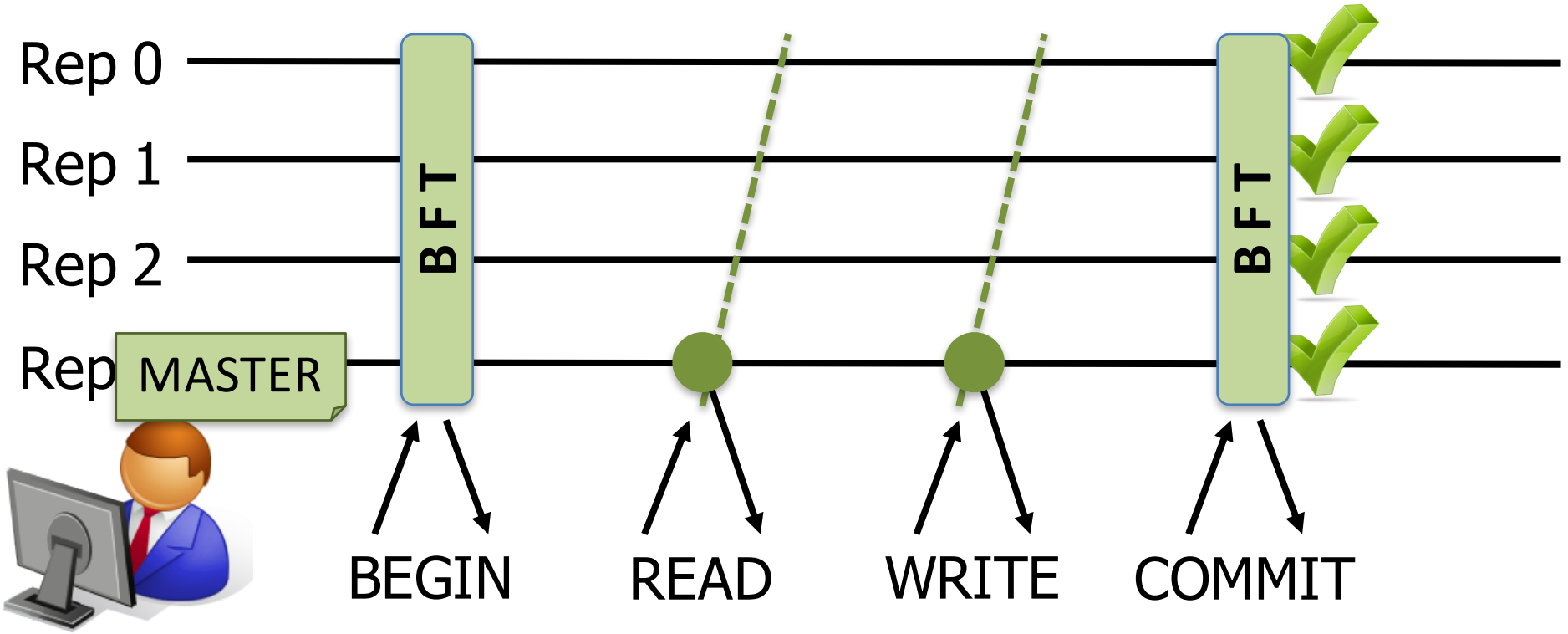
Replicas need to confirm tentative execution



Basic solution: normal case

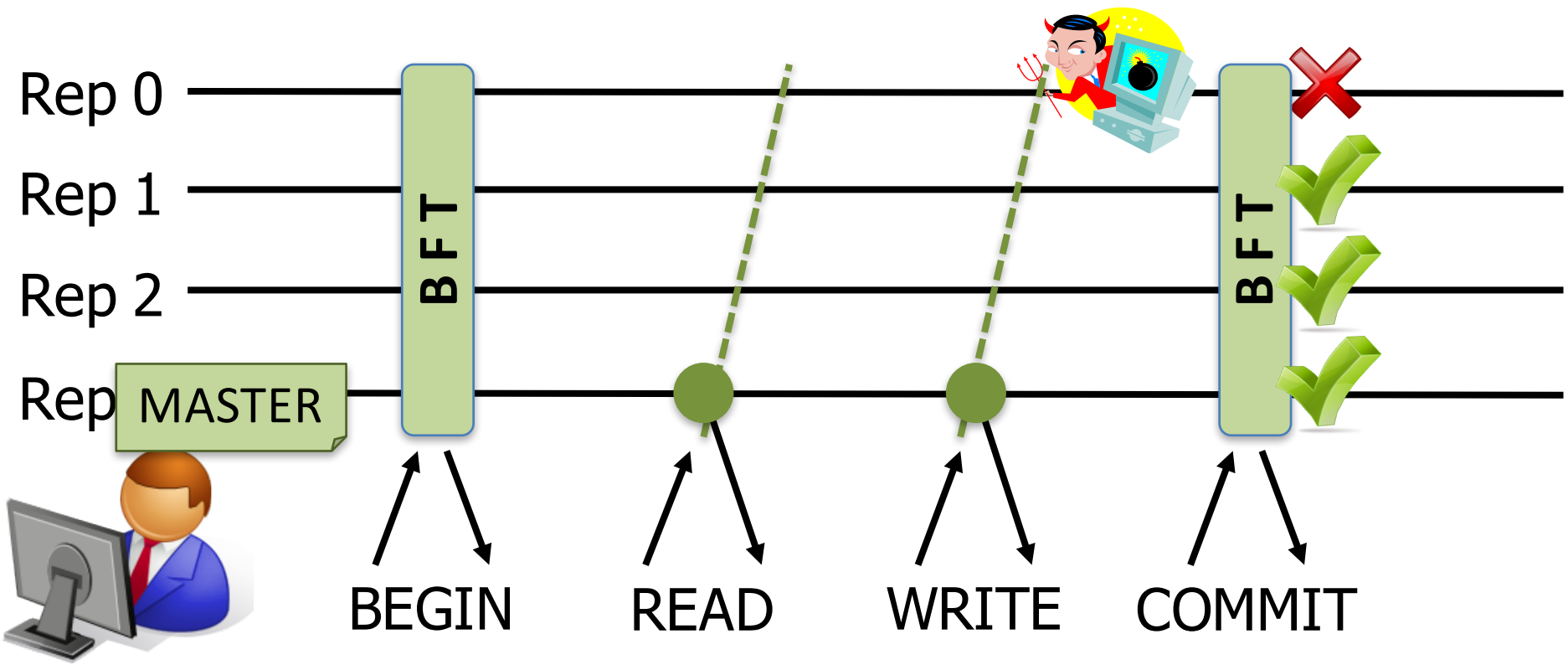
Correct replicas compute the same results

- Execute in the same snapshot
  - BEGIN & COMMIT are totally ordered
- Deterministic



Basic solution: Byzantine replica

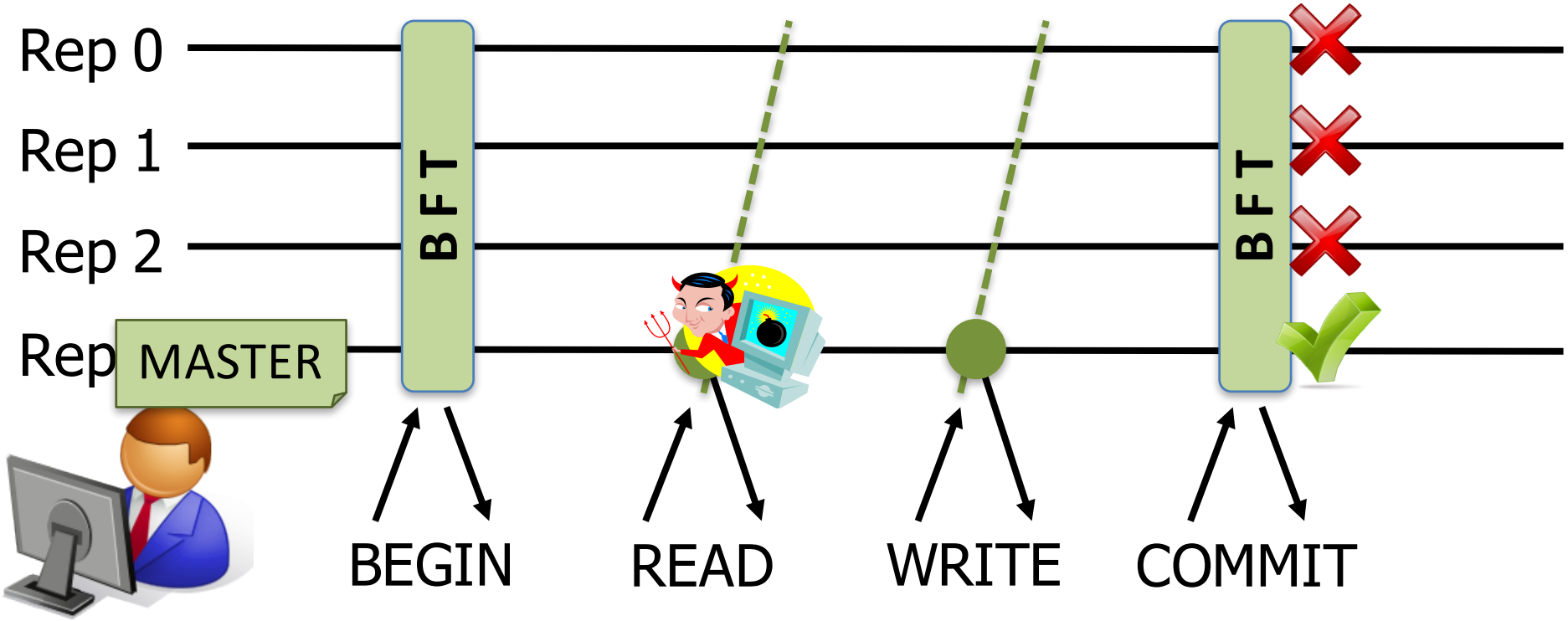
With up to f non-master Byzantine replicas, a quorum of correct replicas will commit



Basic solution: Byzantine master

In the presence of a Byzantine master, correct replicas will abort on incorrect result

- Client sends hash of observed results with COMMIT



Outline

Motivation

Background

Basic solution

The devil is in the details

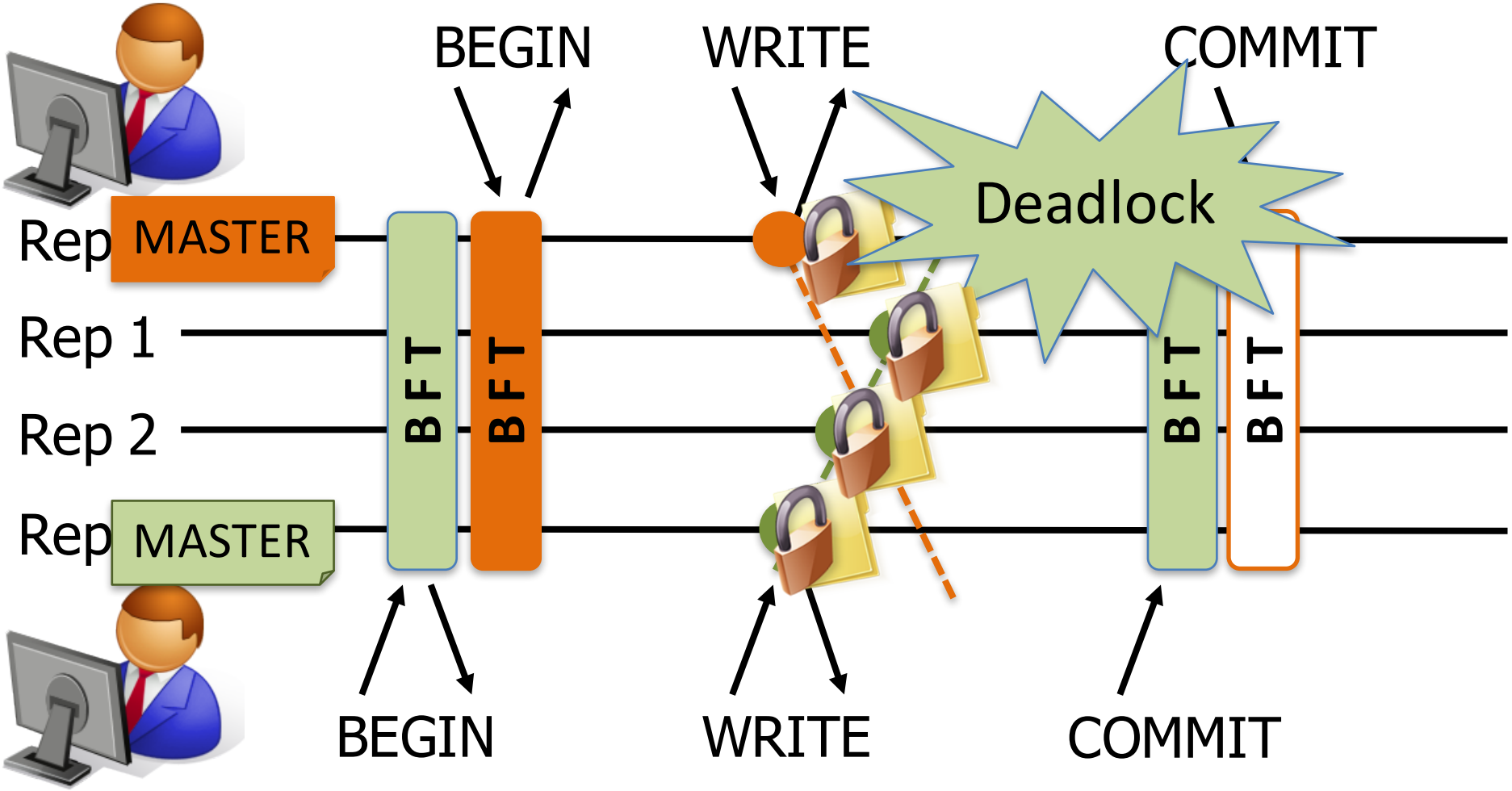
- Avoiding deadlock
- Improving read-only transactions

Final remarks

Databases and locks

Most databases use locks to avoid conflicts

Byzantium must avoid deadlocks



Avoiding deadlocks

Multi-master

- Each transaction/client will select its master replica

Single master

- All transactions have the same master

Multi-master: approach and challenges

Each transaction/client will select its master replica

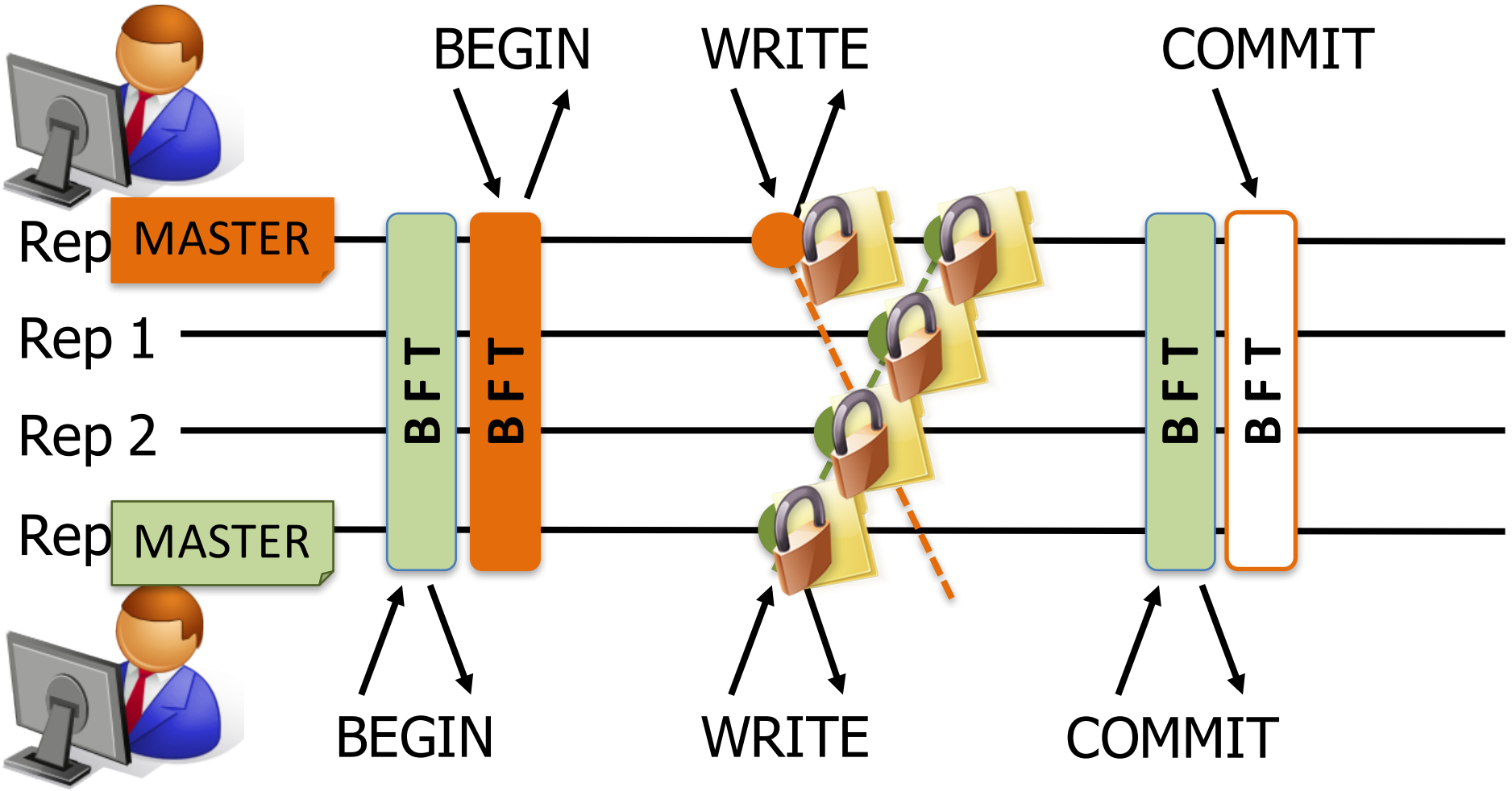
Two conflicting transactions may have different masters and proceed concurrently

Challenge

- Avoid system deadlocks during commit

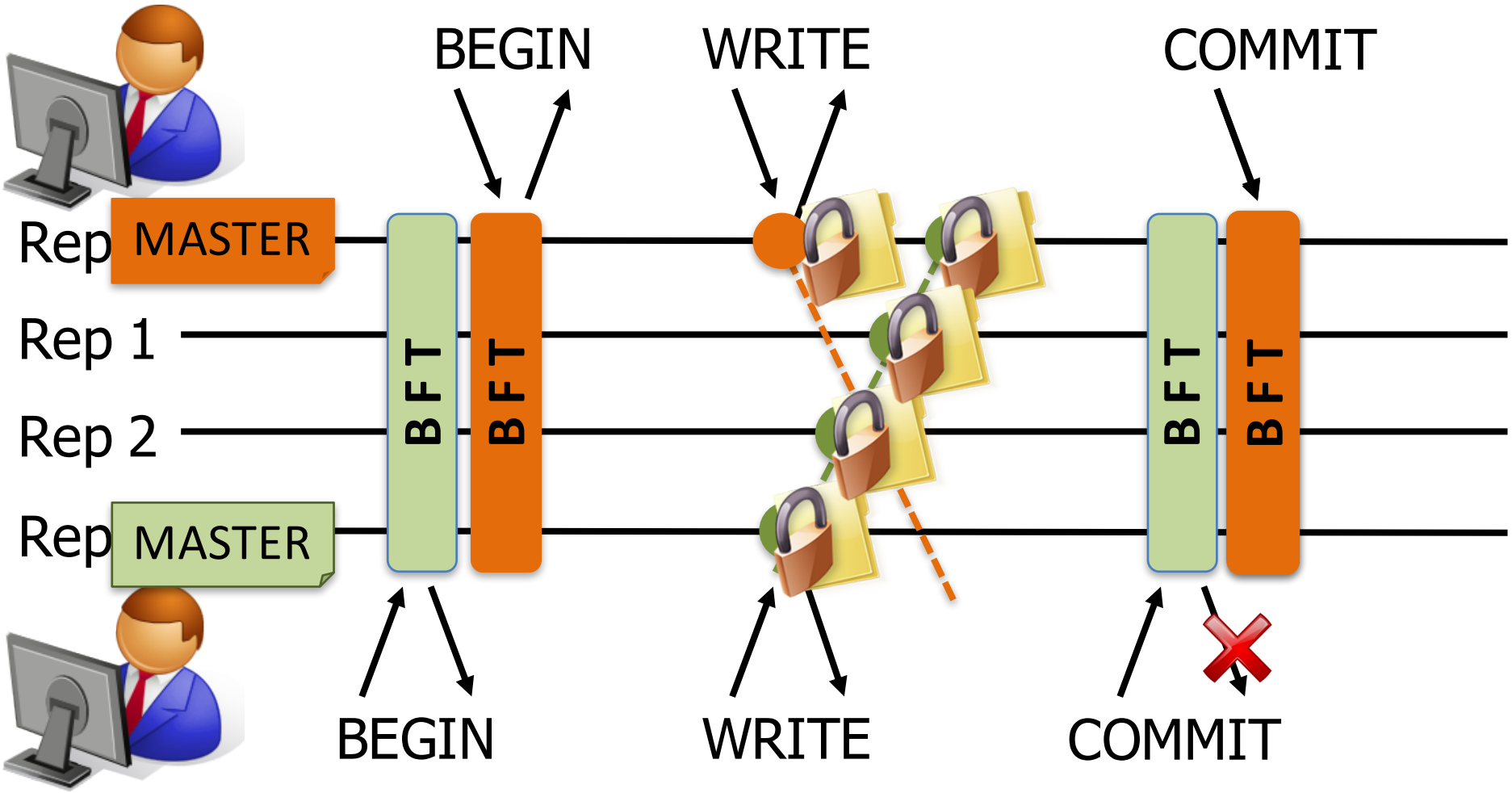
Multi-master: solution

Non-master replicas must undo local transactions to avoid deadlocks



Multi-master: solution

When commit fails, re-executes local transaction from savepoint created on begin



Single master: approach and challenges

A single master exists in the system

All transactions share one master, which manages concurrent transactions

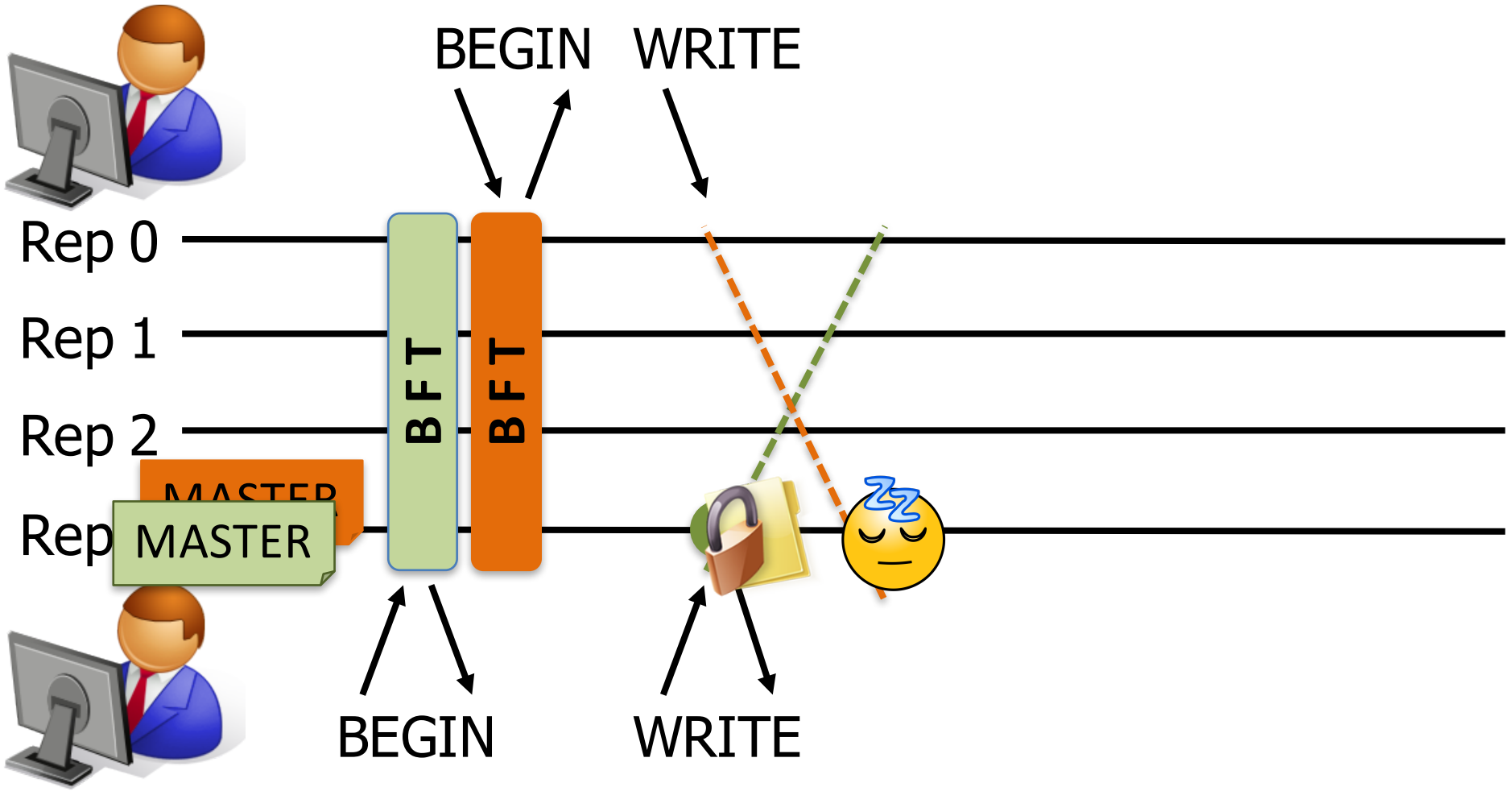
Database server solves deadlocks on master

Challenges

- Execute operation is non-master replicas as soon as possible
- Avoiding deadlocks in non-master replicas

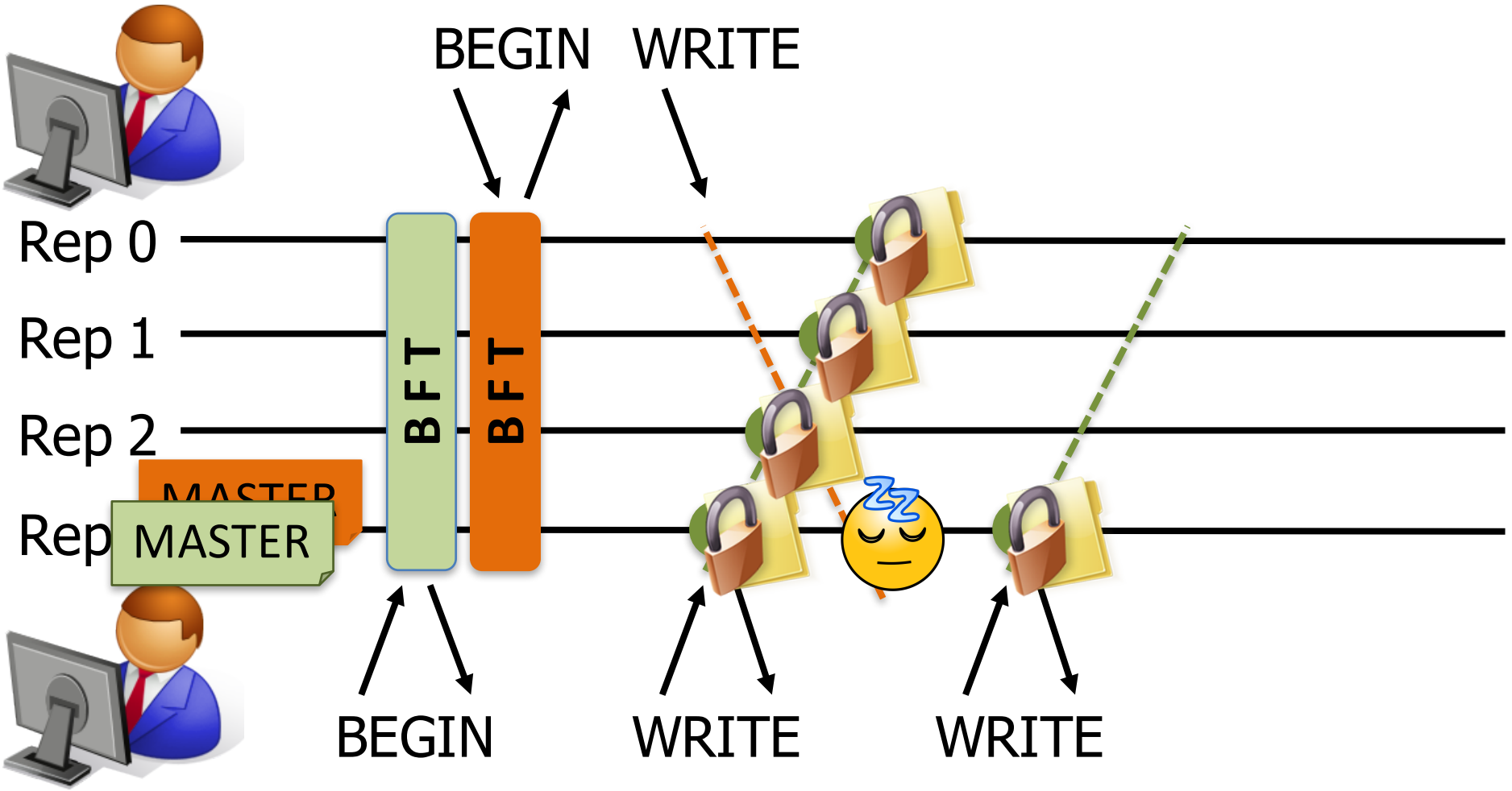
Singe-master: solution

A transaction blocks in the master if it conflicts with another



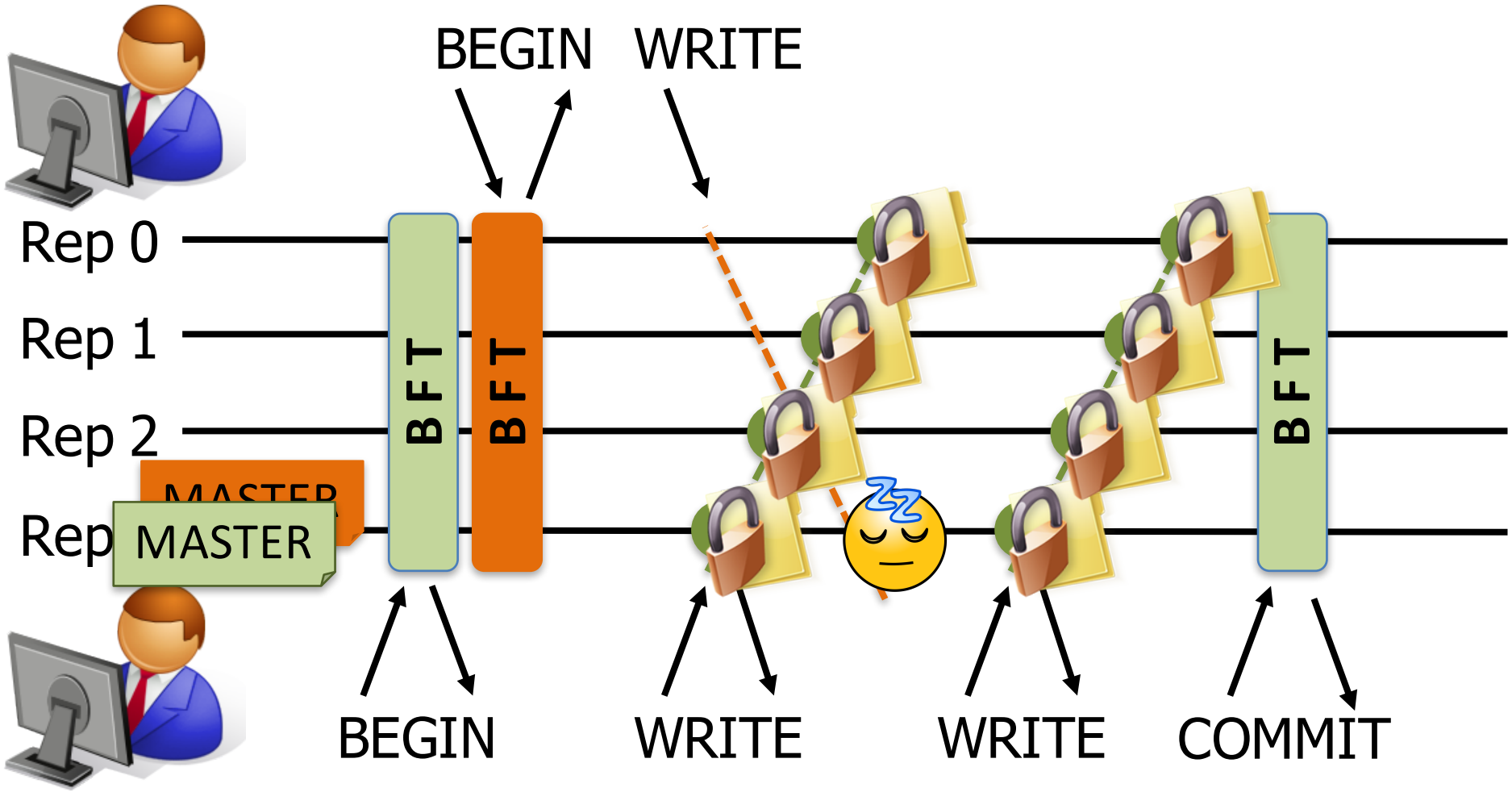
Singe-master: solution

Non-master replicas execute previous operation guaranteeing consistent locking



Singe-master: solution

On commit, all replicas execute last operation



Comparing solutions

Single master

- All transactions proceed in all replicas with one-operation lag
- Faster commits

Multiple masters

- Non-master replicas execute transaction operations in a burst

Evaluation

Byzantium **single master**

Byzantium **multi-master**

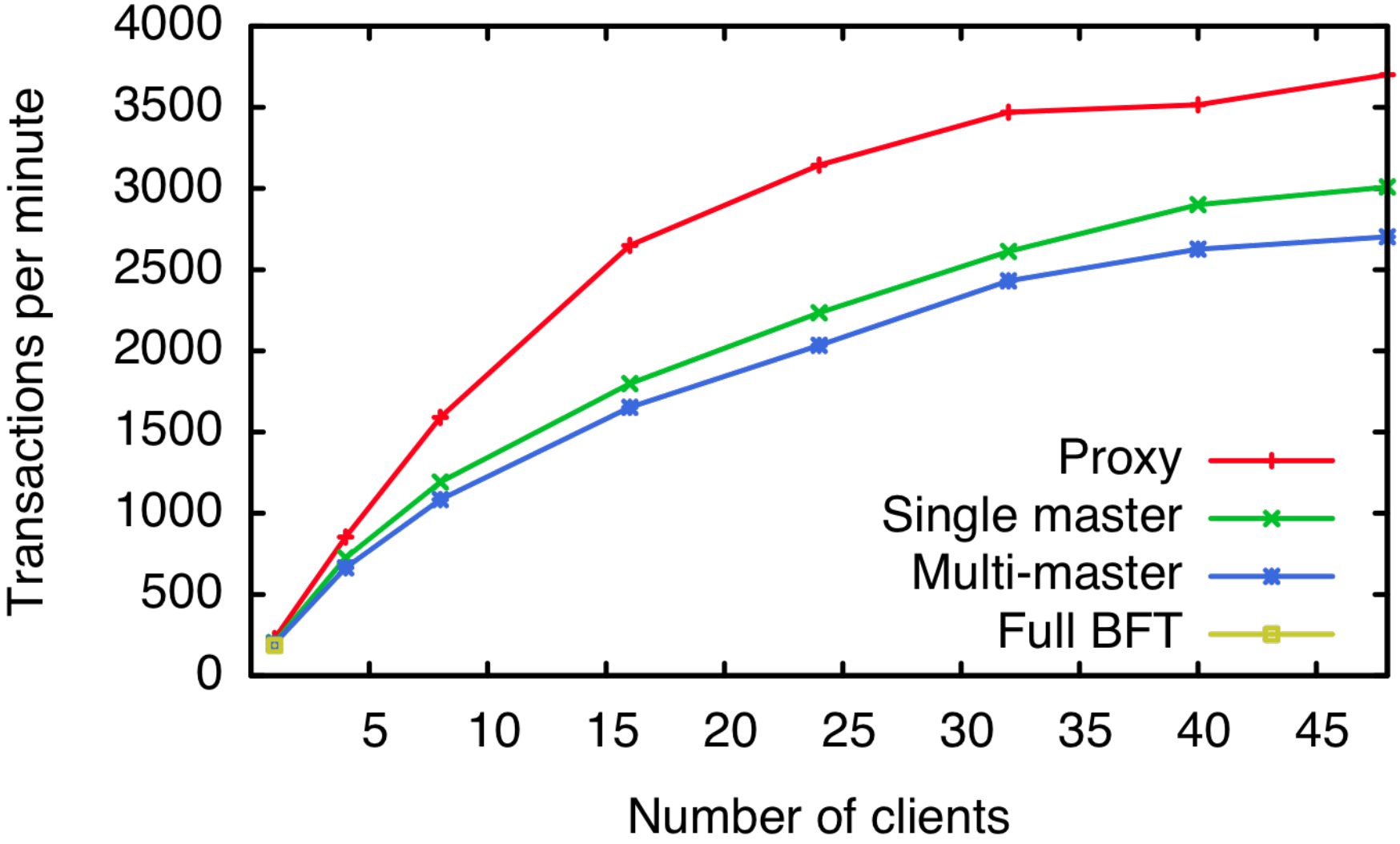
Proxy: single server accessed through proxy

Full BFT: all operations execute as BFT

Benchmark	TPC-C (open source)
Database	PostgreSQL 8.3.4
OS	Linux 2.6.30
Processor	Single-core 2.6 Ghz Opteron 252
Memory	4GB
Network	1Gbit ethernet

Standard TPC-C (92% writes)

Modest overhead compared with non-replicated DB



Outline

Motivation

Background

Basic solution

The devil is in the details

- Avoiding deadlock
- Improving read-only transactions

Final remarks

Optimizing read-only transactions

Key observations

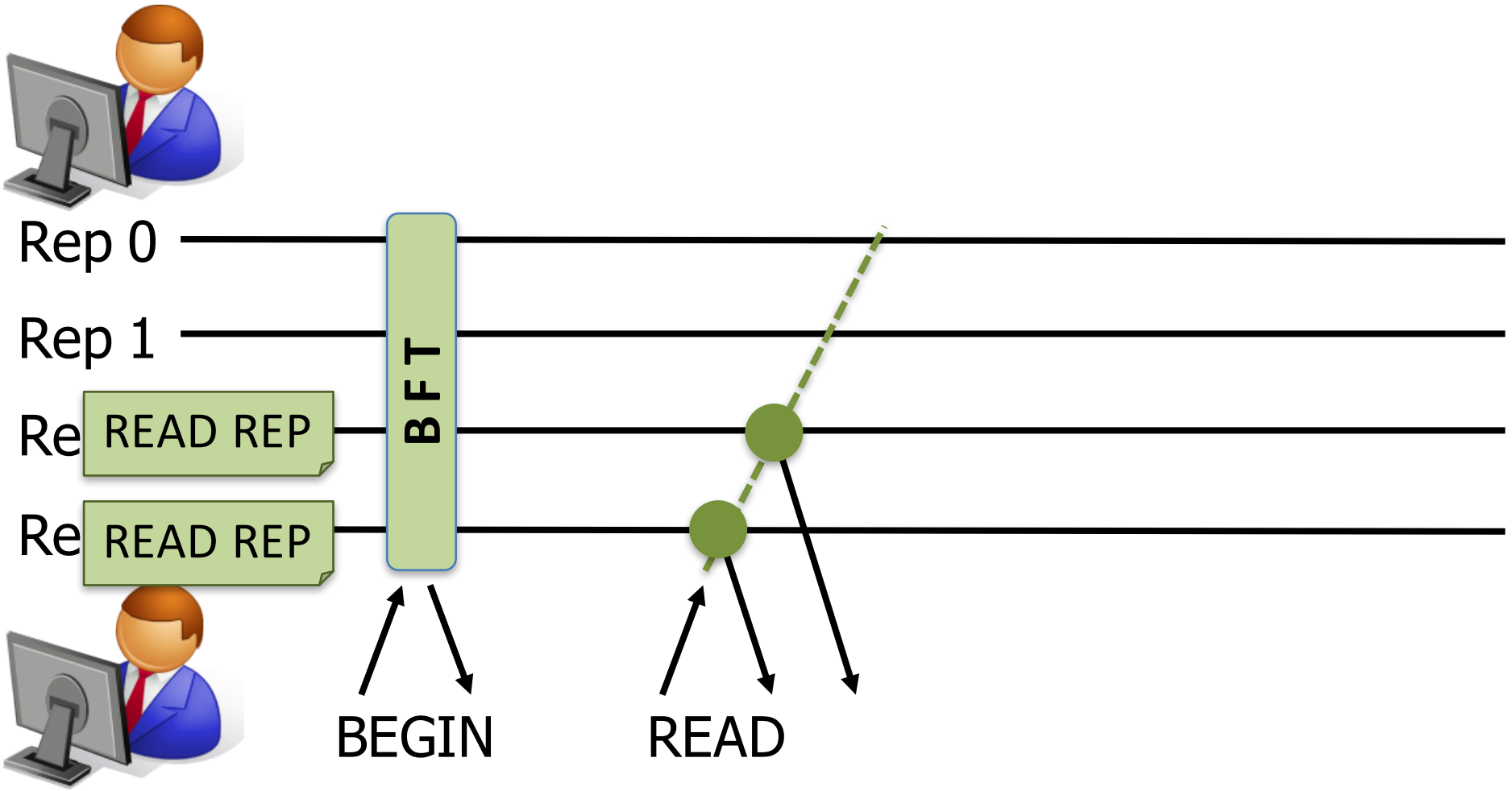
- In snapshot isolation reads never block
- Reads confirmed by f+1 replicas are correct

Key ideas

- Read operations contact f+1 replicas in parallel
- Commit does not require BFT operation

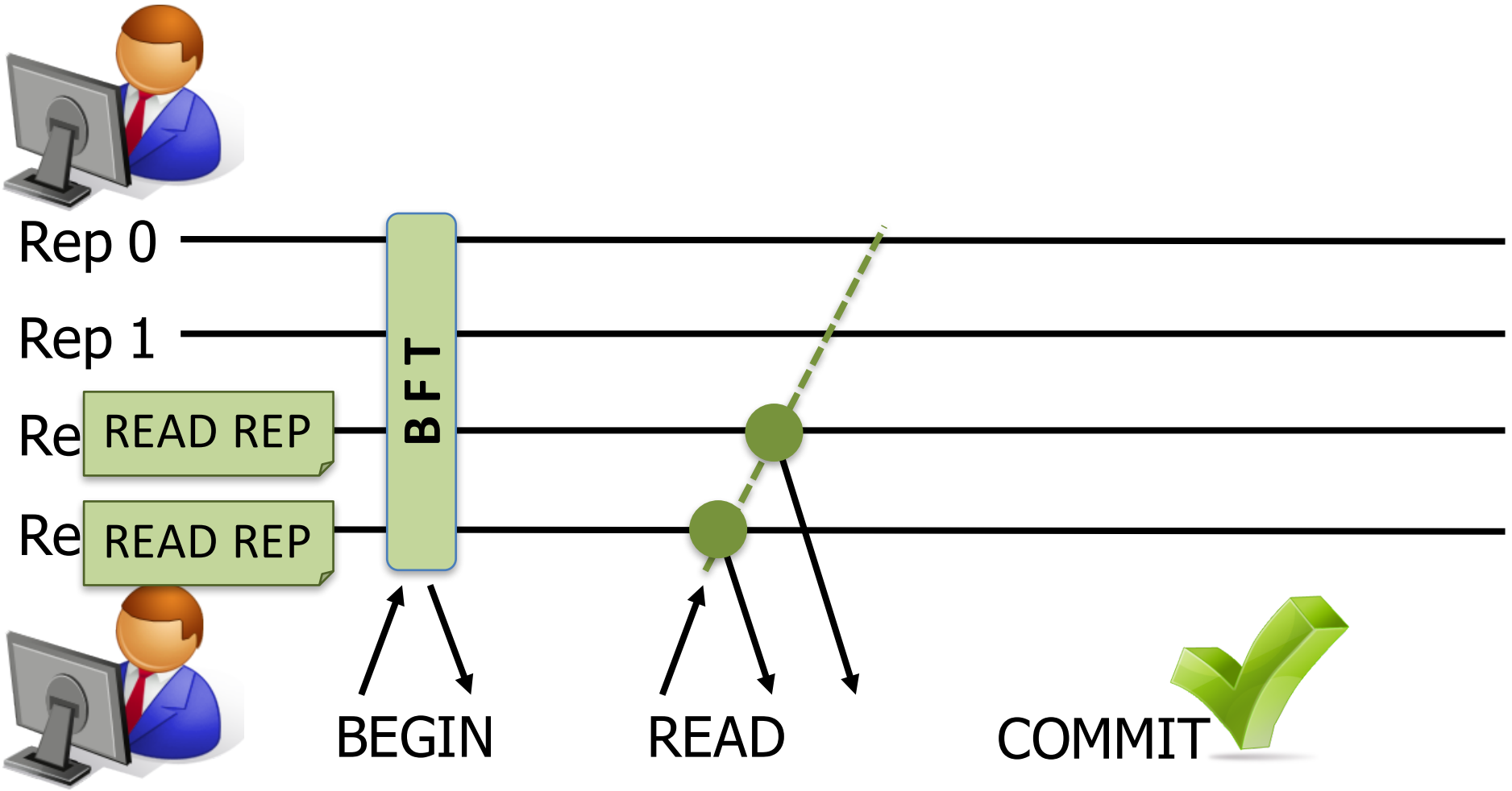
Optimizing read-only transactions

Reads execute tentatively in f+1 read replicas



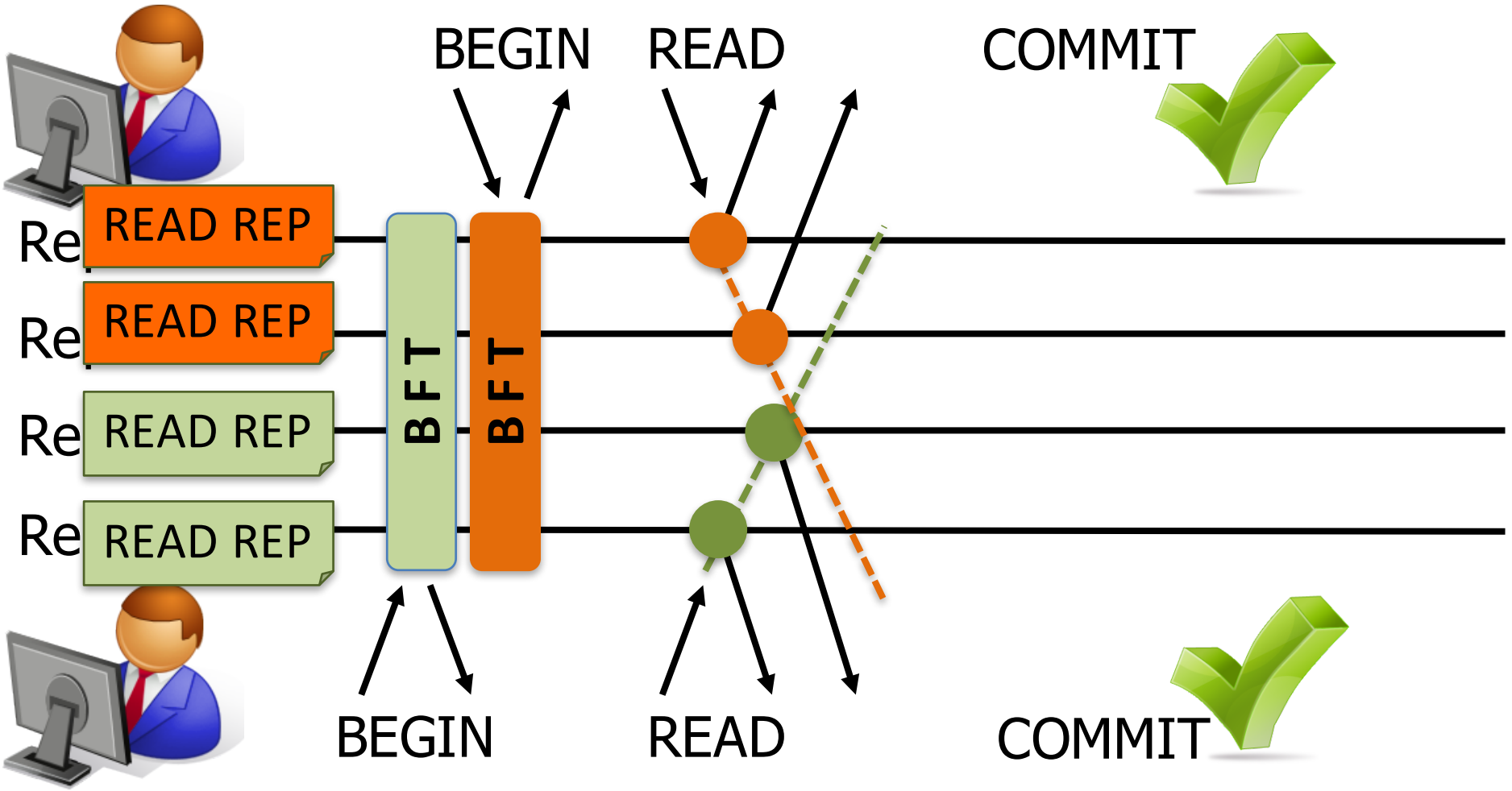
Optimizing read-only transactions

Commit confirmed locally if all reads confirmed



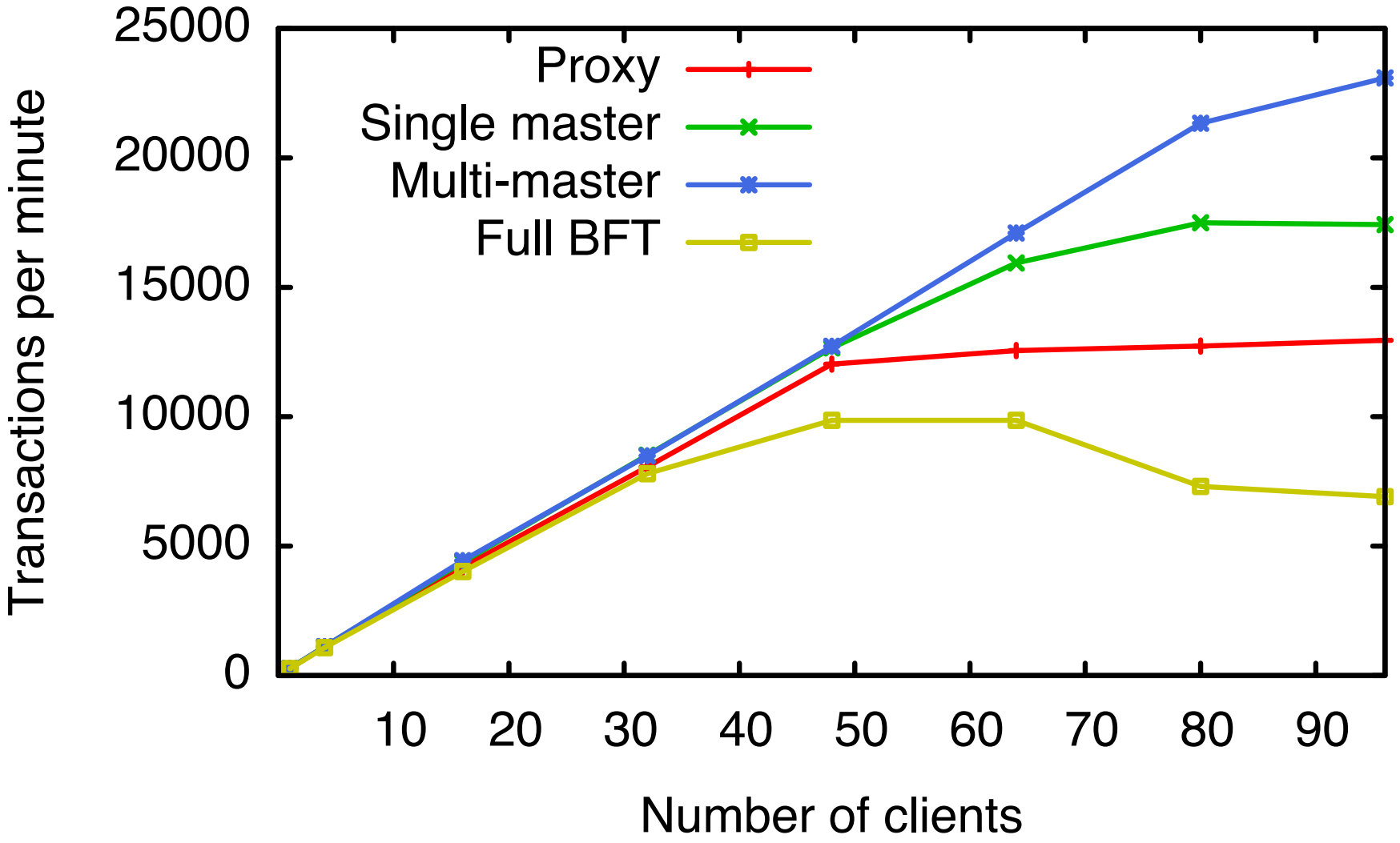
Optimizing read-only transactions

Reads from different clients striped to different replicas => reduced load on each server



Read-only workload

Up to 90% improvement over non-replicated DB



Summary

Middleware solution for tolerating Byzantine faults in database systems

- No trusted component
- Avoid BFT serialization for improved concurrency
- Striping of read operations among replicas

Two solutions

- Single master – better for read-write transactions
- Multi-master – better for read-only transactions