

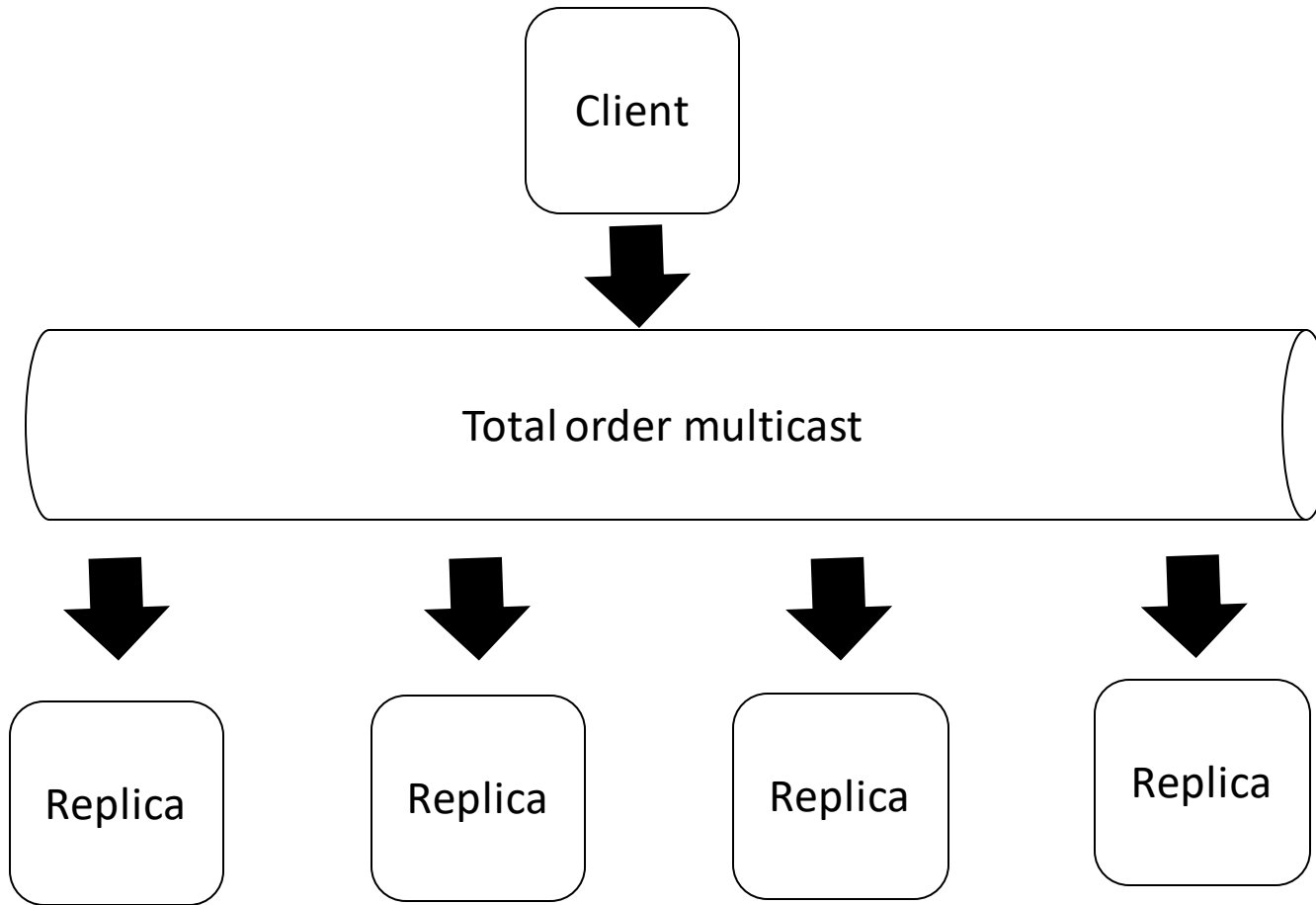
BFT-controllers for Intrusion-tolerant systems

Miguel Garcia
Navtalk'2018

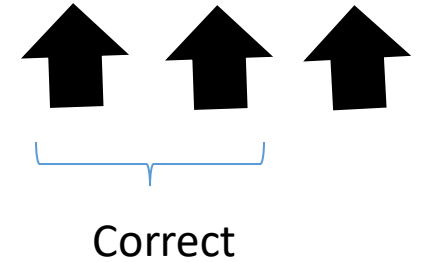
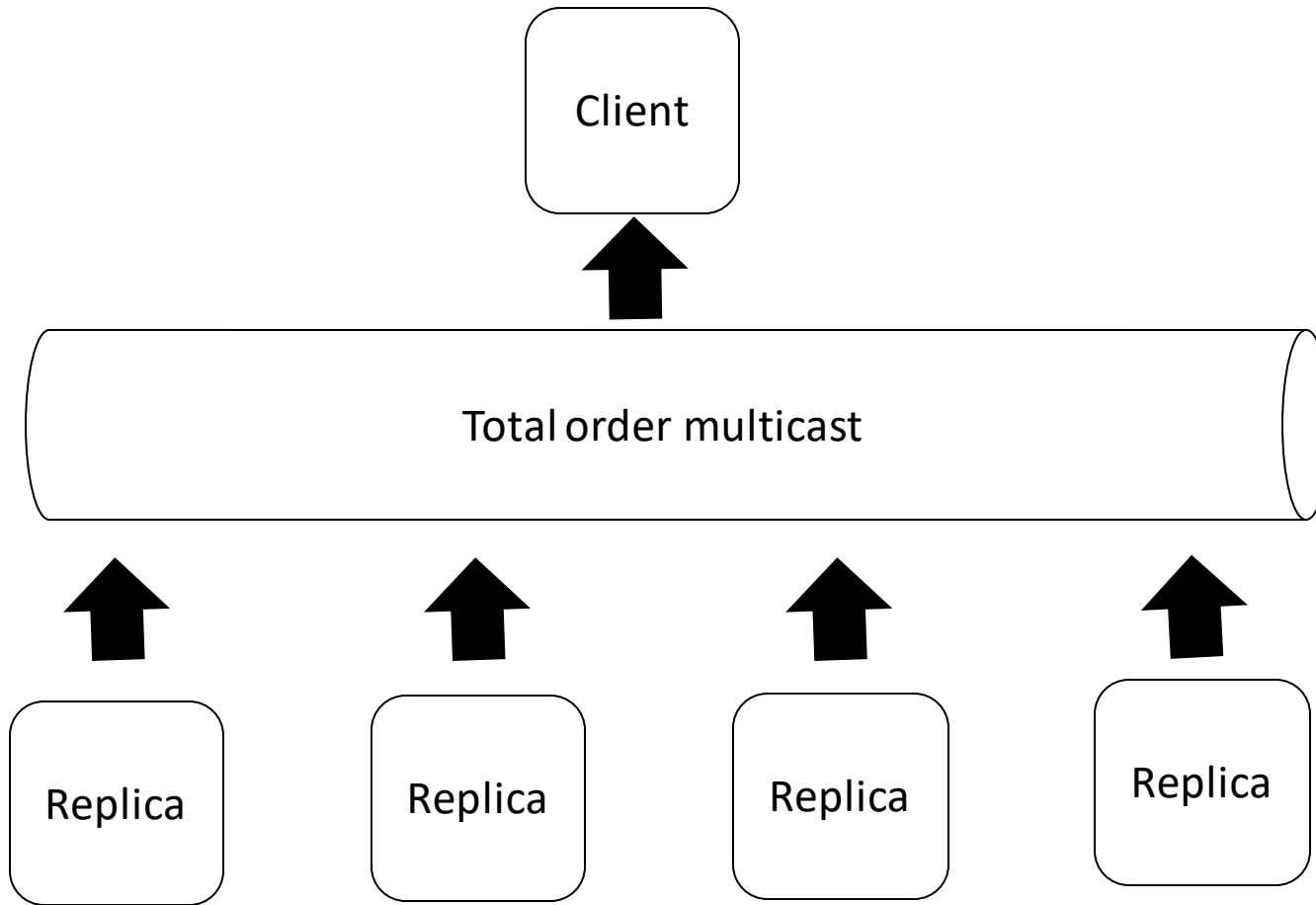
Byzantine fault tolerance in a nutshell

It is a technique that allows services to execute correctly even in the presence of faults

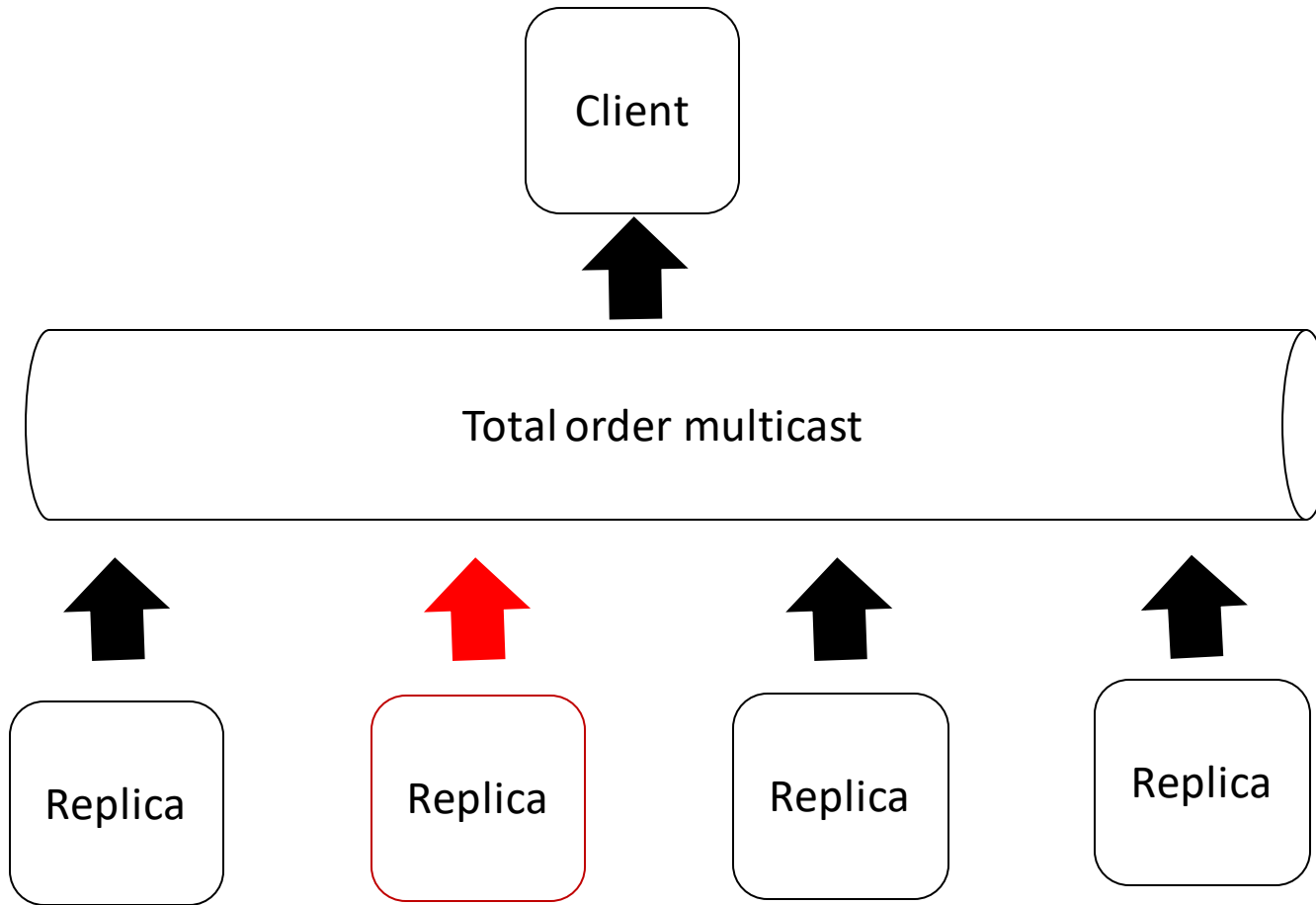
Byzantine fault tolerance in a nutshell



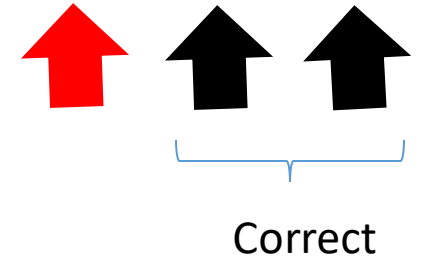
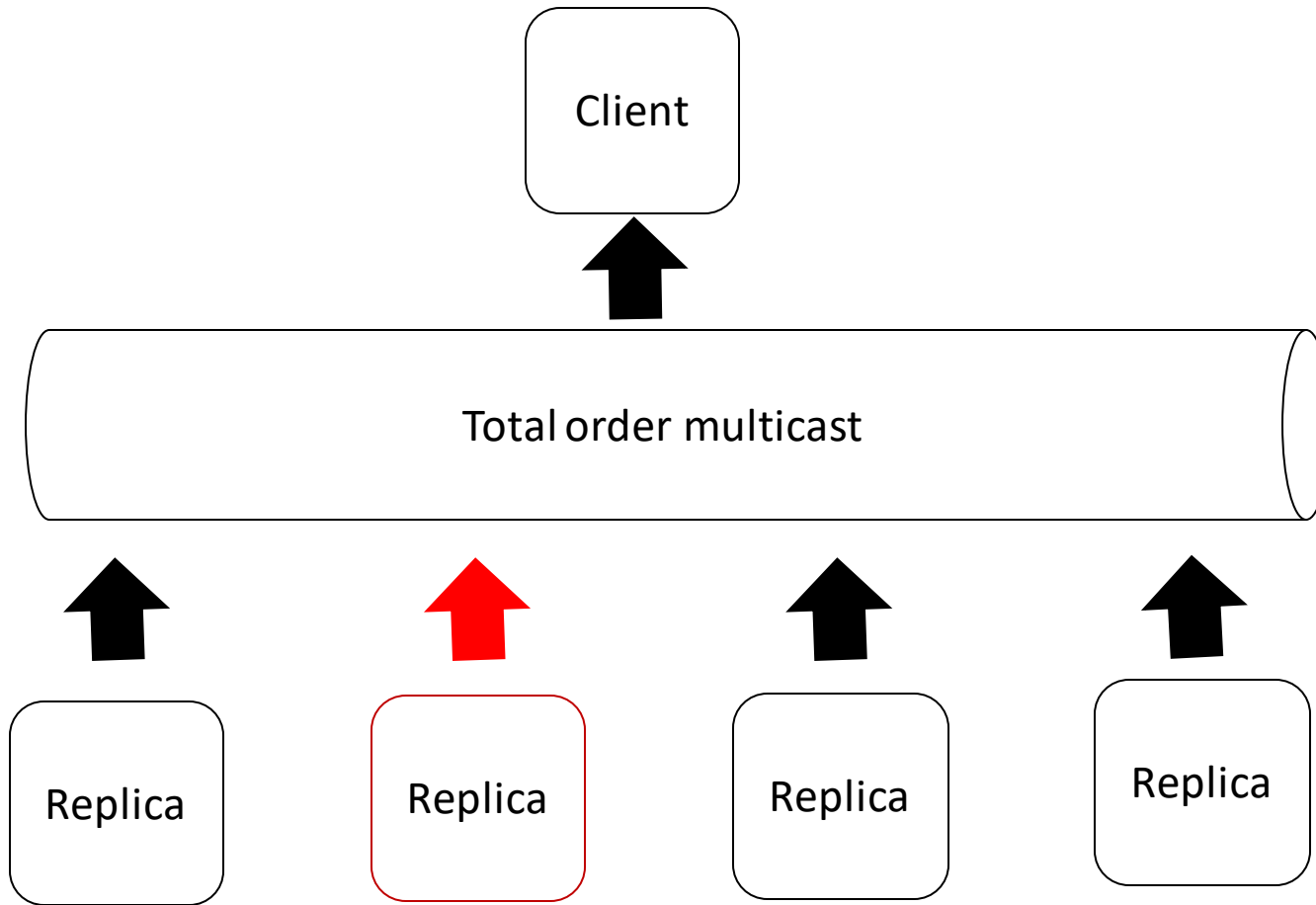
Byzantine fault tolerance in a nutshell



Byzantine fault tolerance in a nutshell



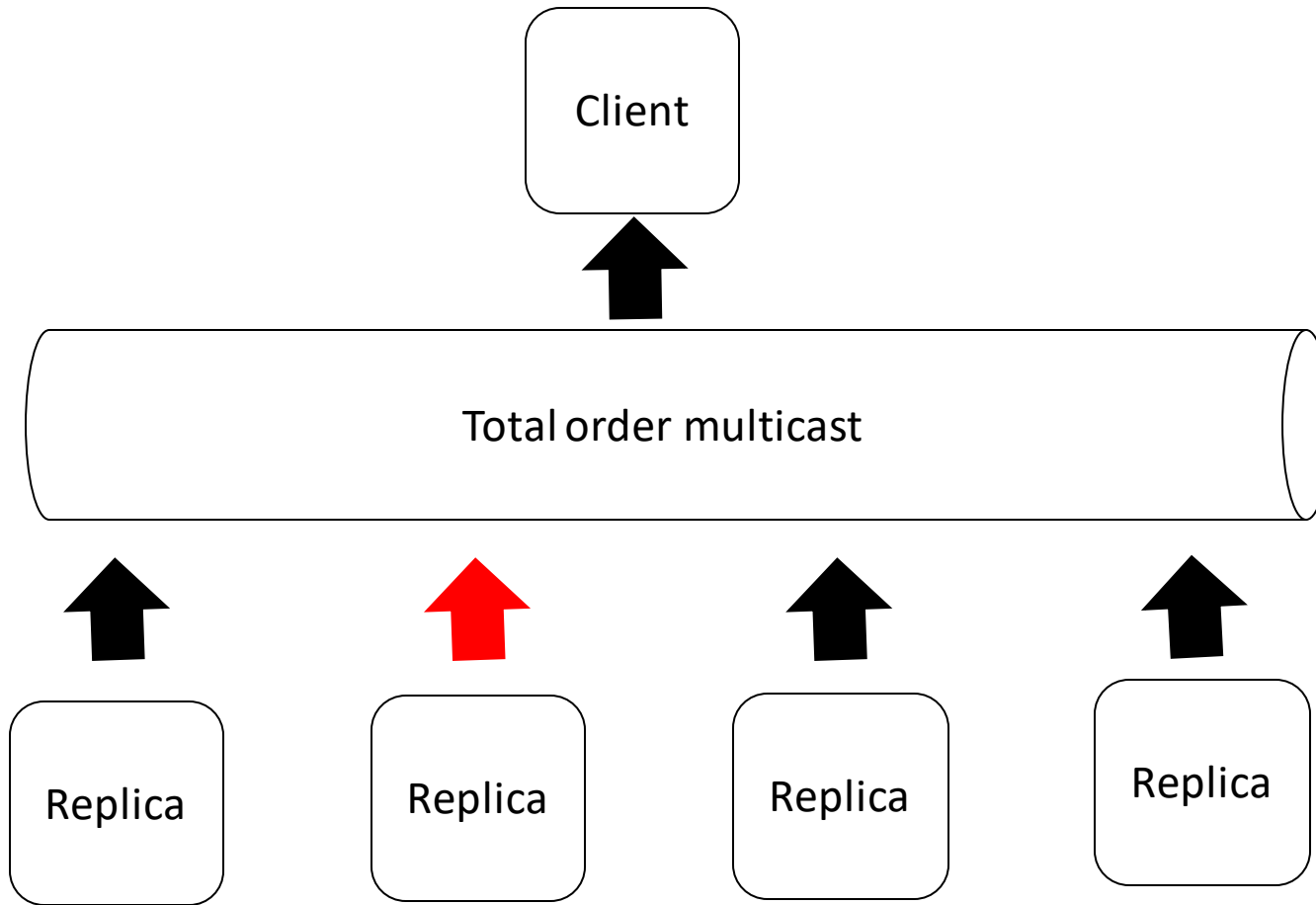
Byzantine fault tolerance in a nutshell



BFT-Intrusion tolerance

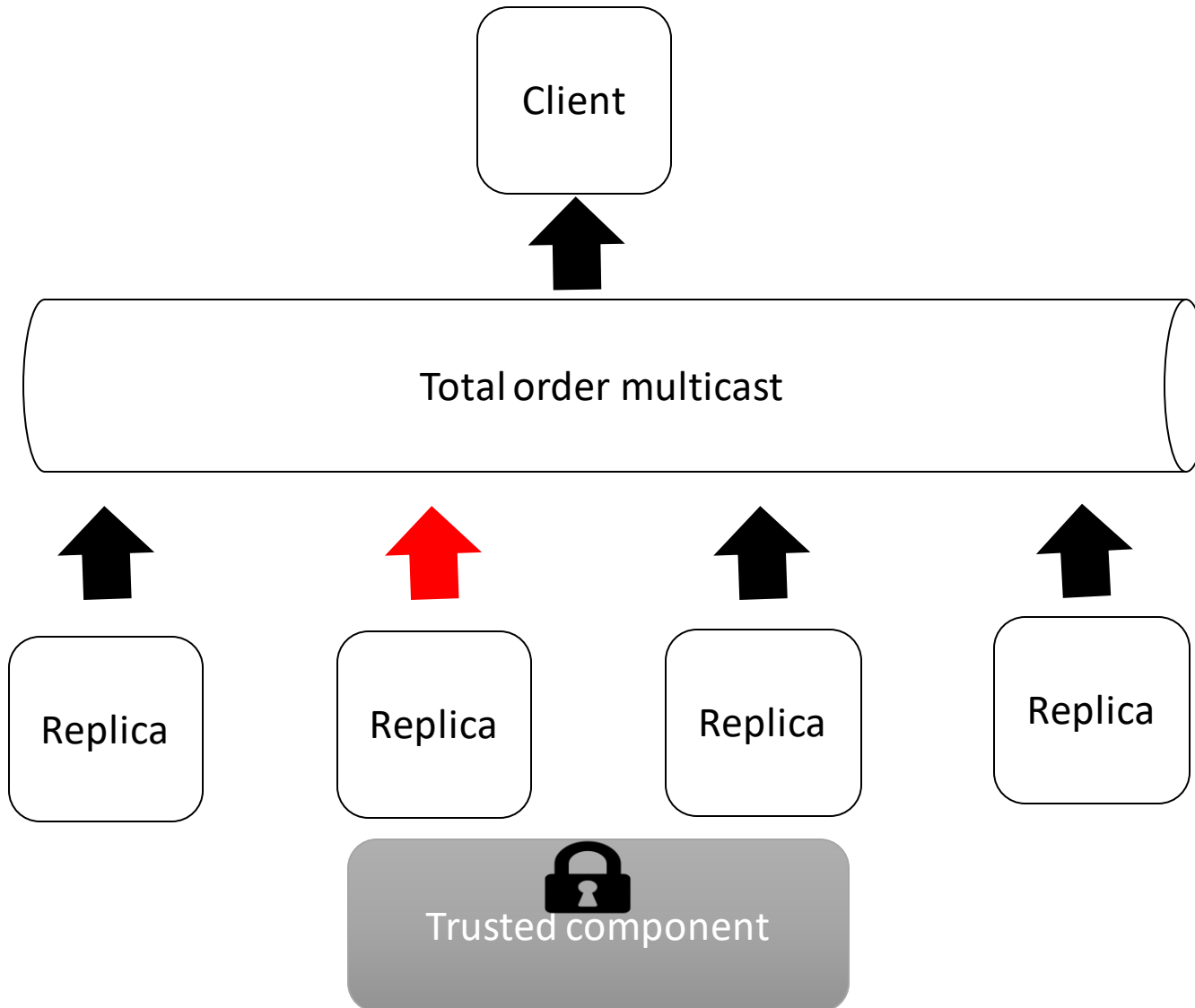
- Clean the replicas faulty state – **recovery techniques**
- Create replicas' fault independence – **diversity mechanisms**

BFT-Intrusion tolerance



Can we trust on a faulty replica to self-recover?

BFT-Intrusion tolerance

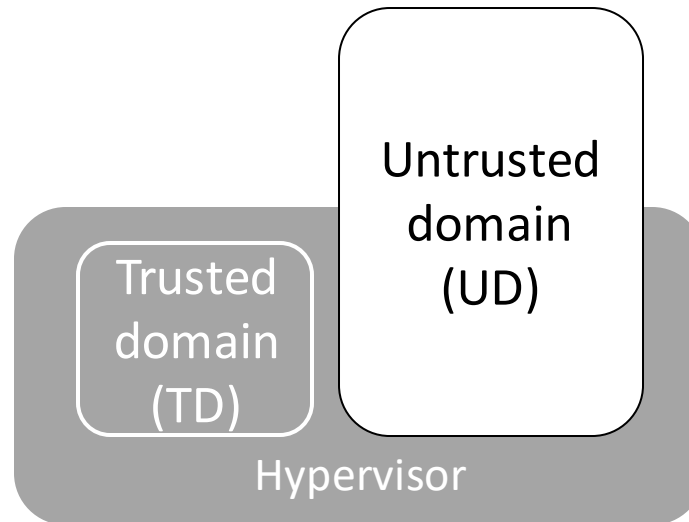


Why do we need to trust some part?

- In the Byzantine model a compromised replica is lost for the attacker.
- We need additional mechanisms to ensure the correct operation:
 - Tamper proof components or software isolation

E.g., Hypervisors

- Hypervisors have been used in several works to provide **isolation between a Byzantine environment and a controlled (trusted) environment**

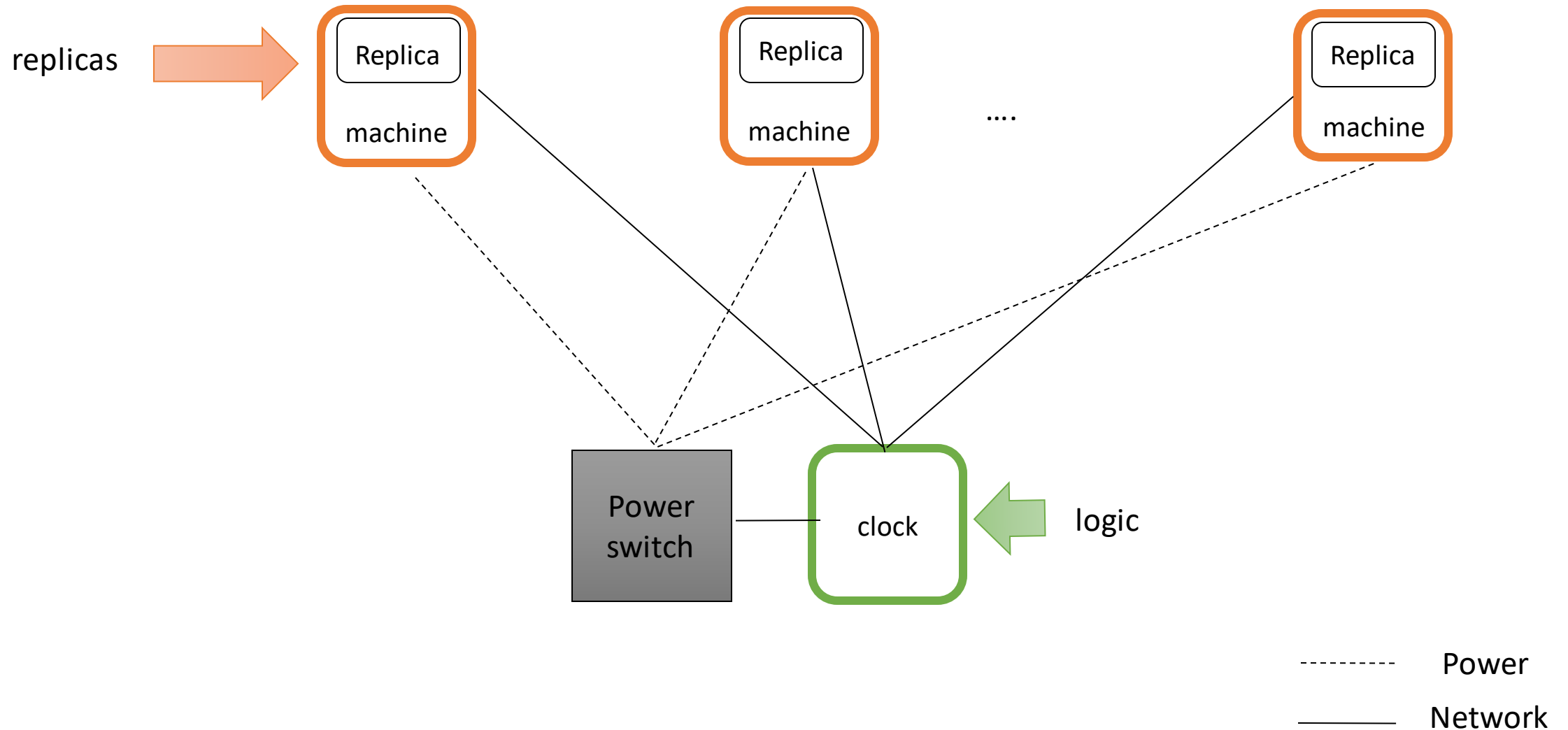


Existent solutions

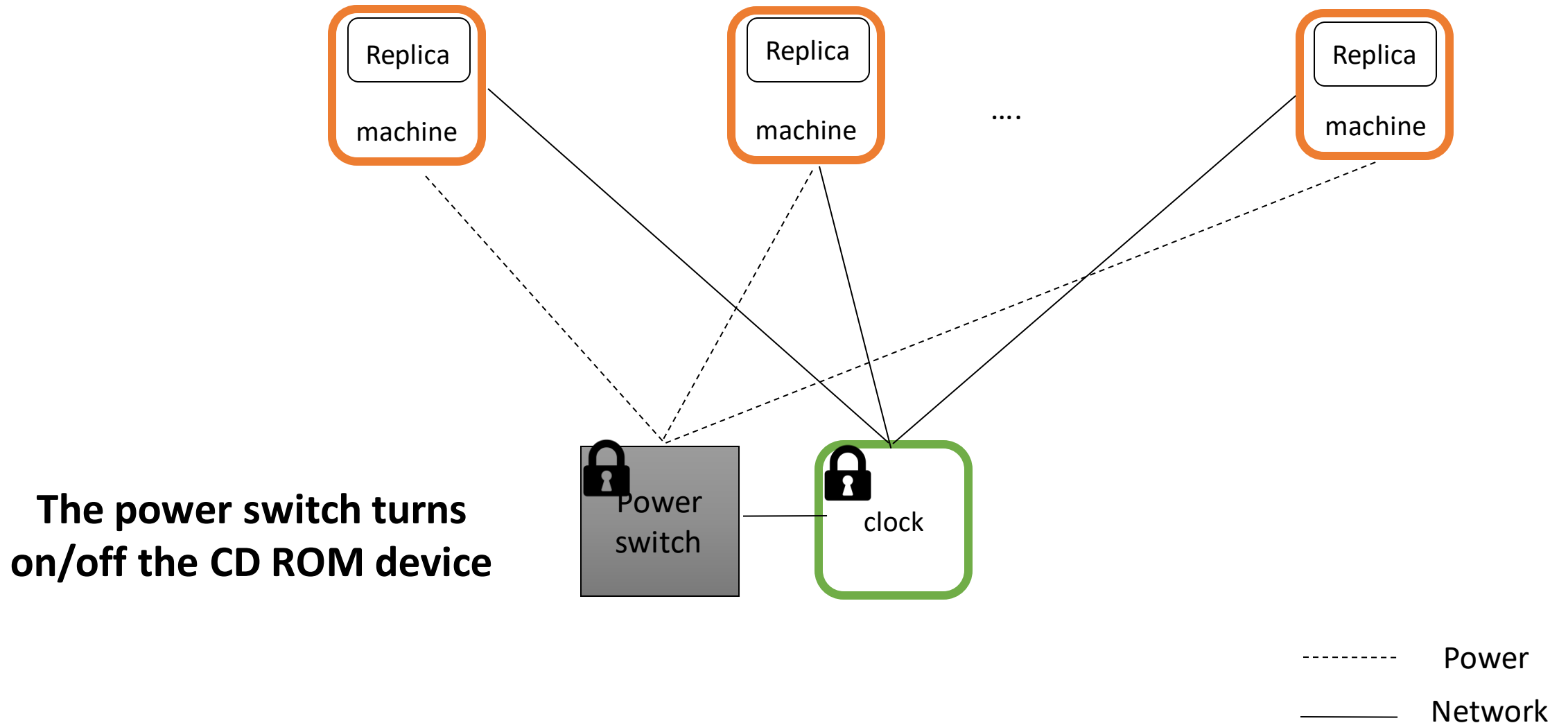
Existent solutions

- There are two types of solutions:
 - **Bare metal**: difficult to implement recovery/diversity techniques
 - **Virtualized**: make it easier to recover and diversify replicas

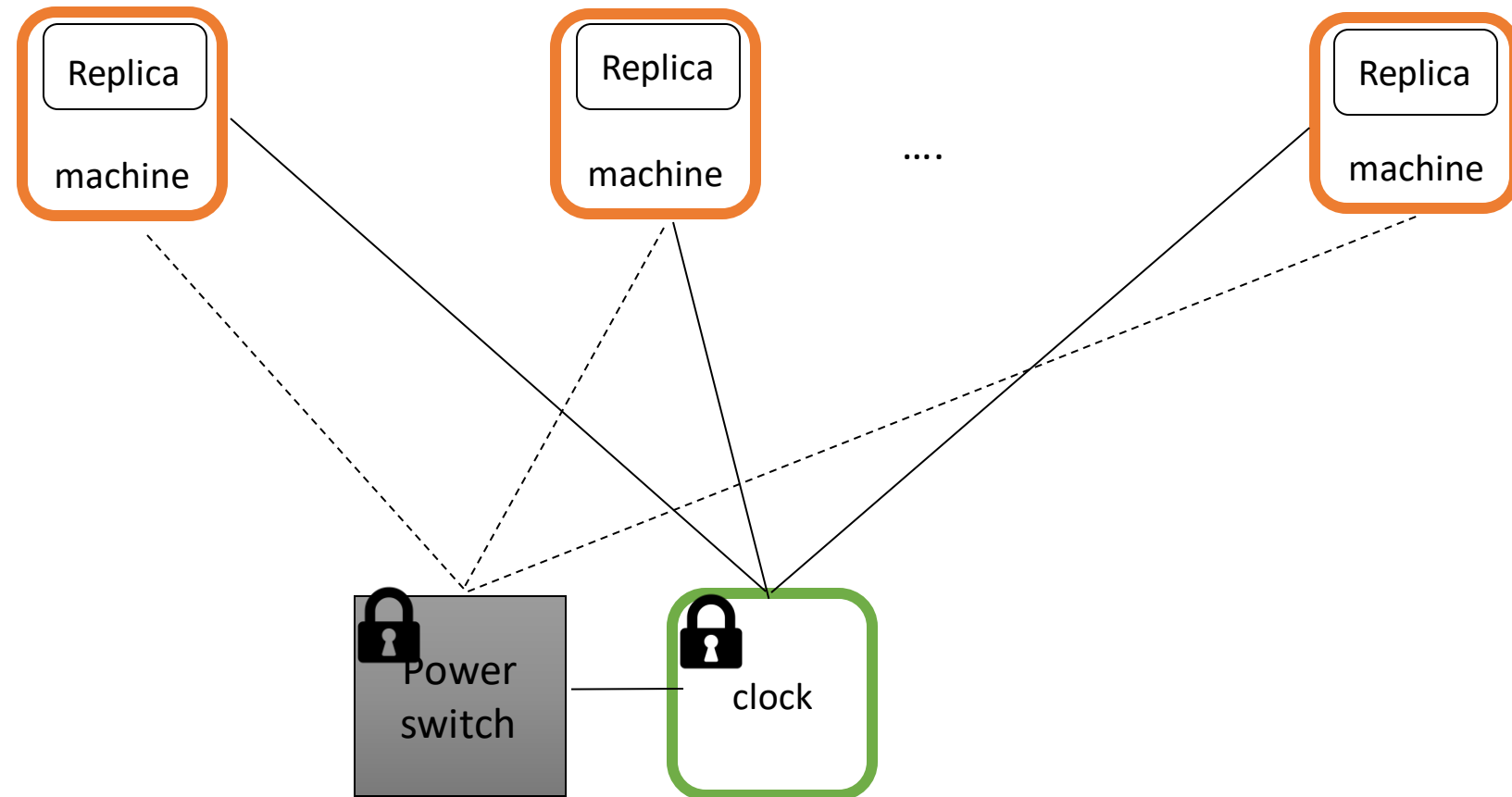
Bare metal: Roeder 2010



Bare metal: Roeder 2010



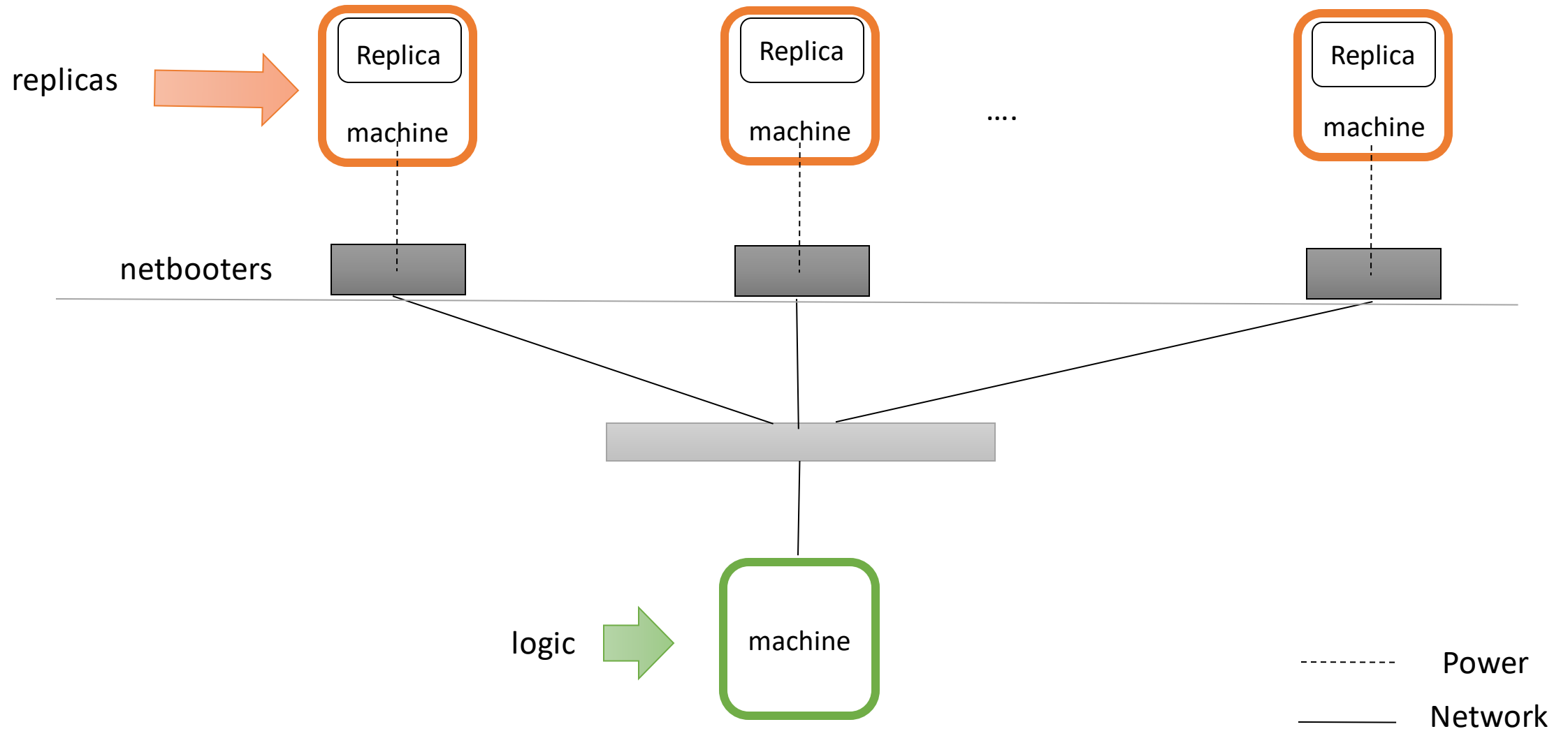
Bare metal: Roeder 2010



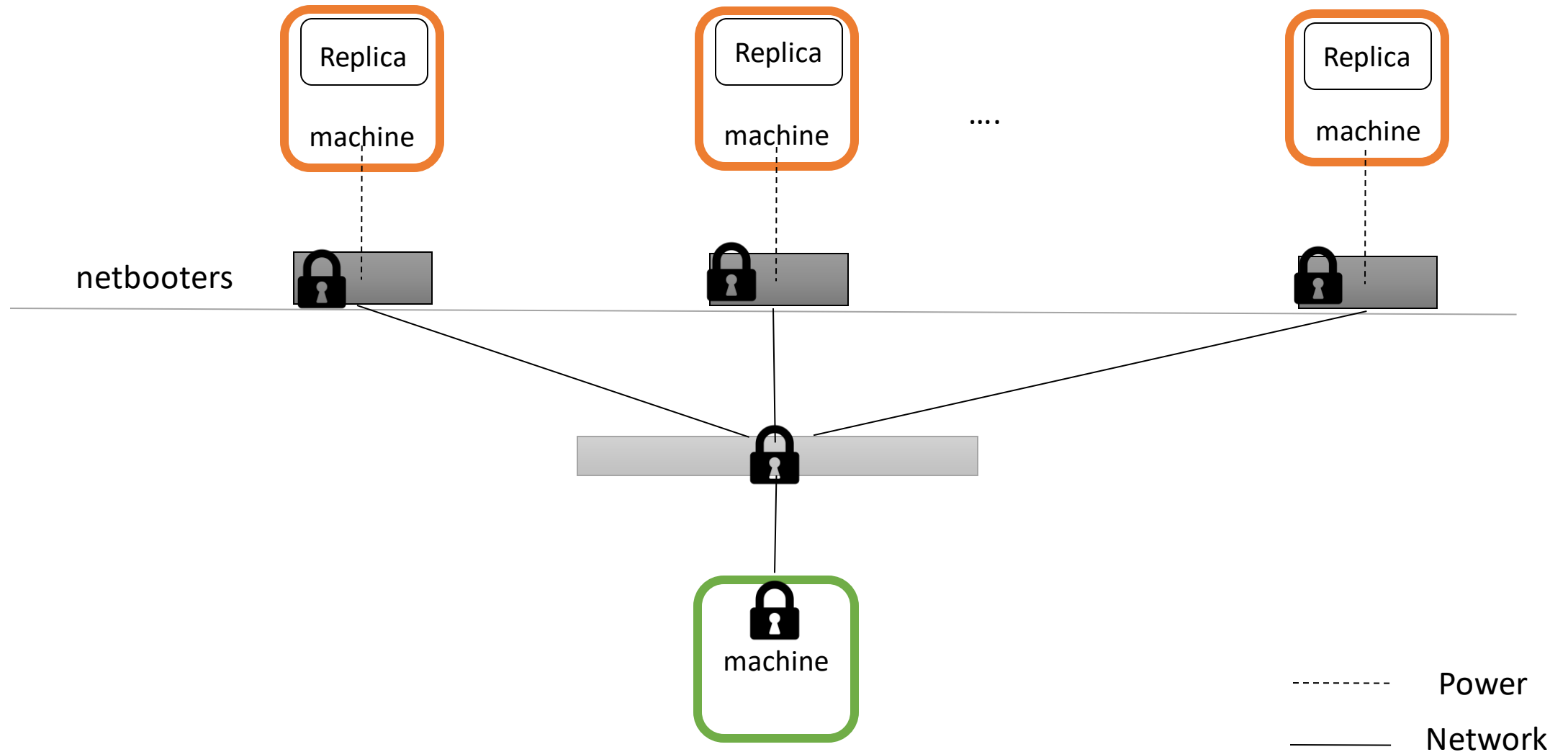
On: recovery mode install the OS from the CD Rom
Off: normal mode , to run the OS

----- Power
——— Network

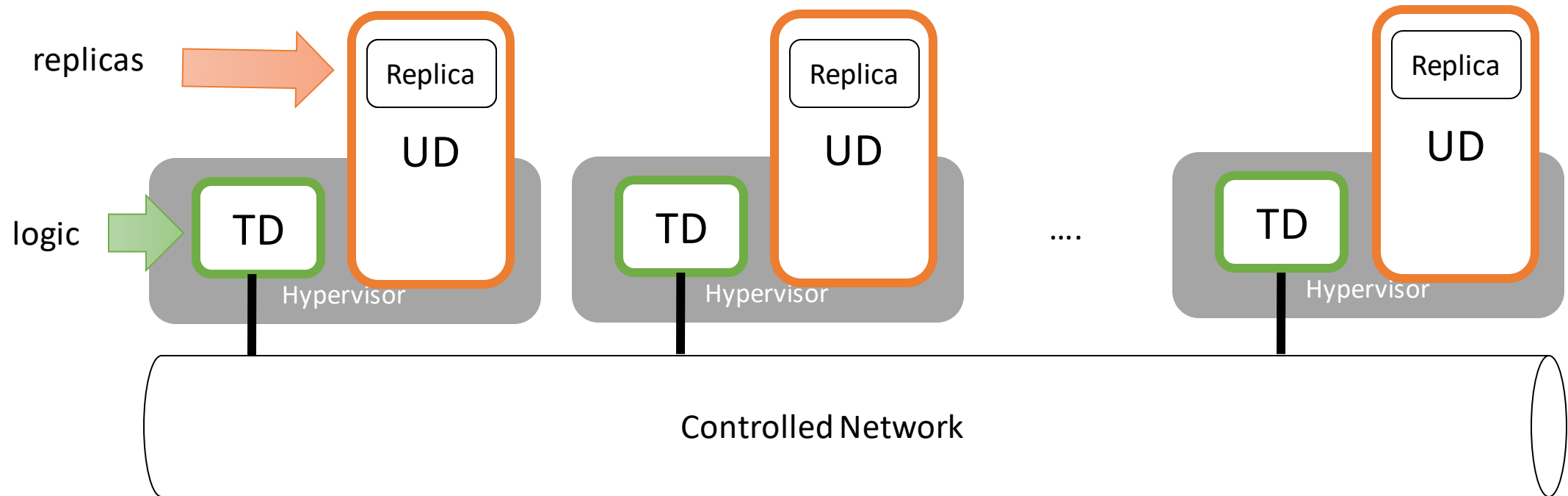
Bare metal: Platania 2014



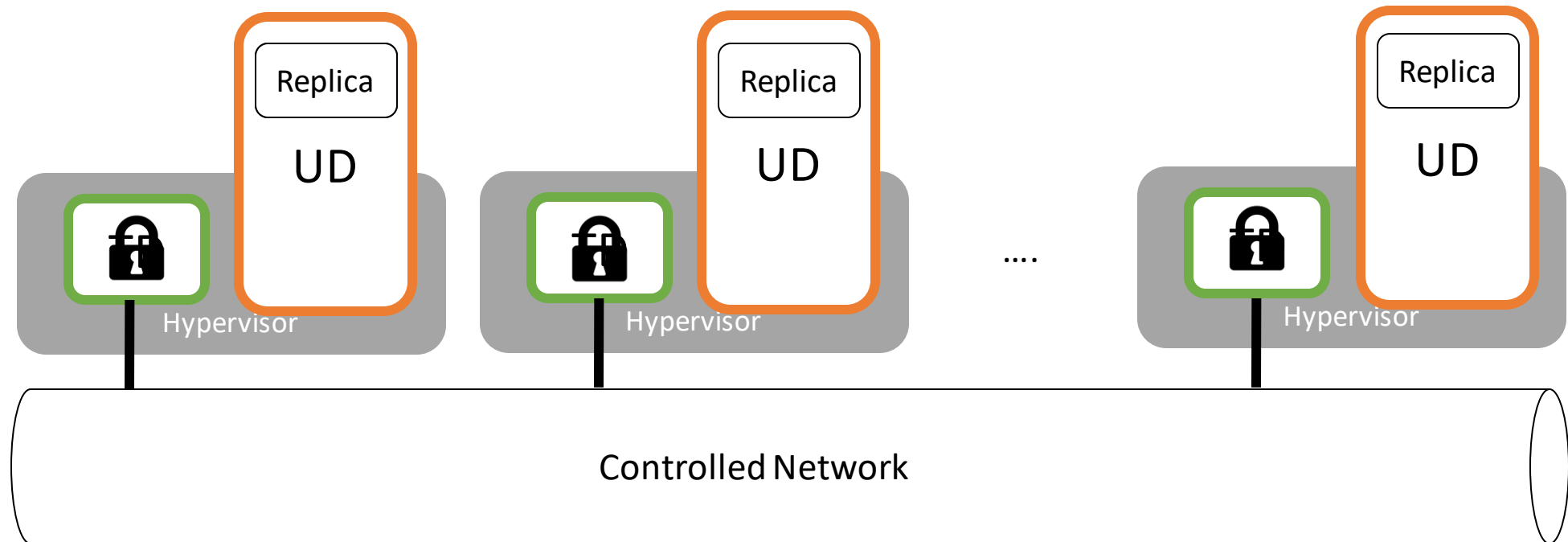
Bare metal: Platania 2014



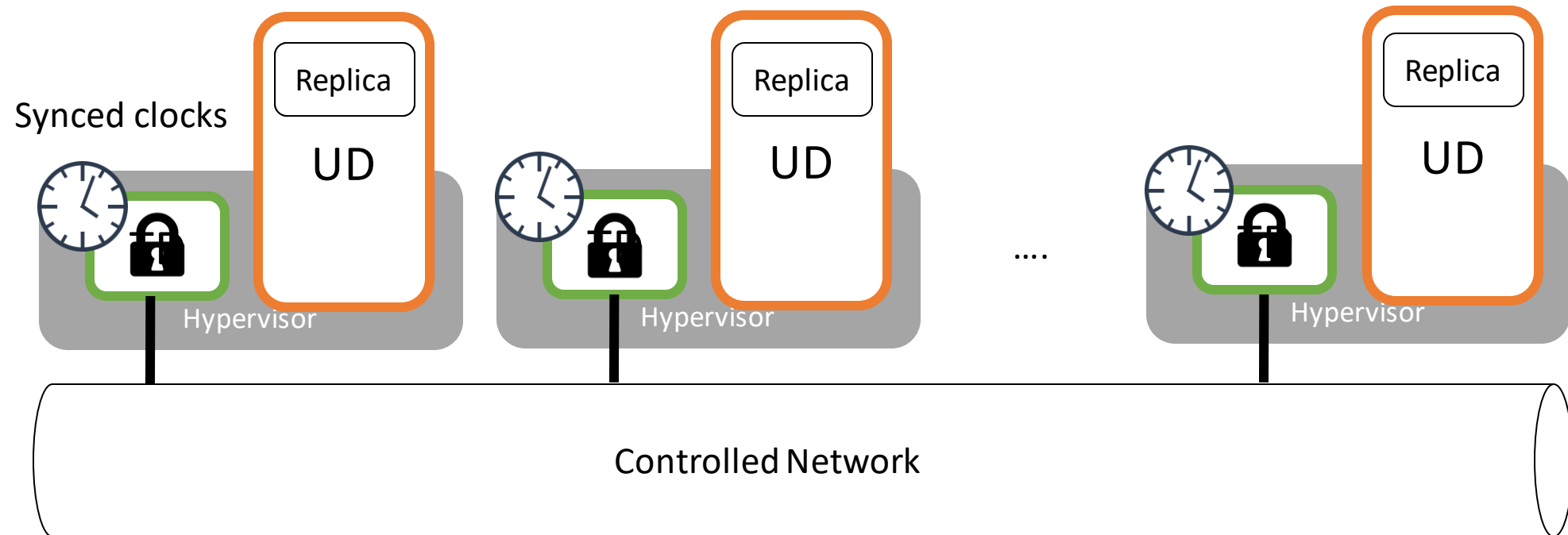
Virtualized: Sousa 2007



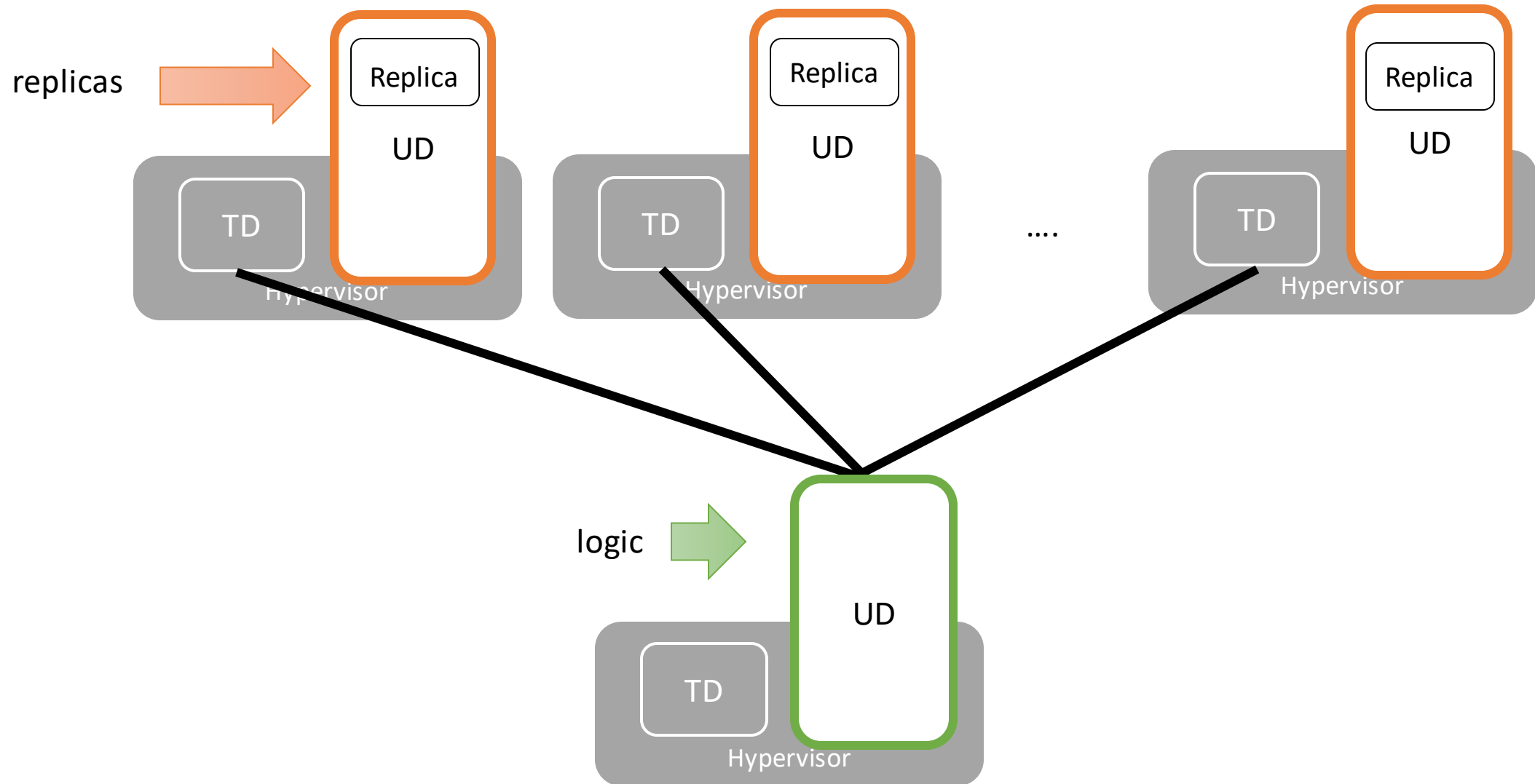
Virtualized: Sousa 2007



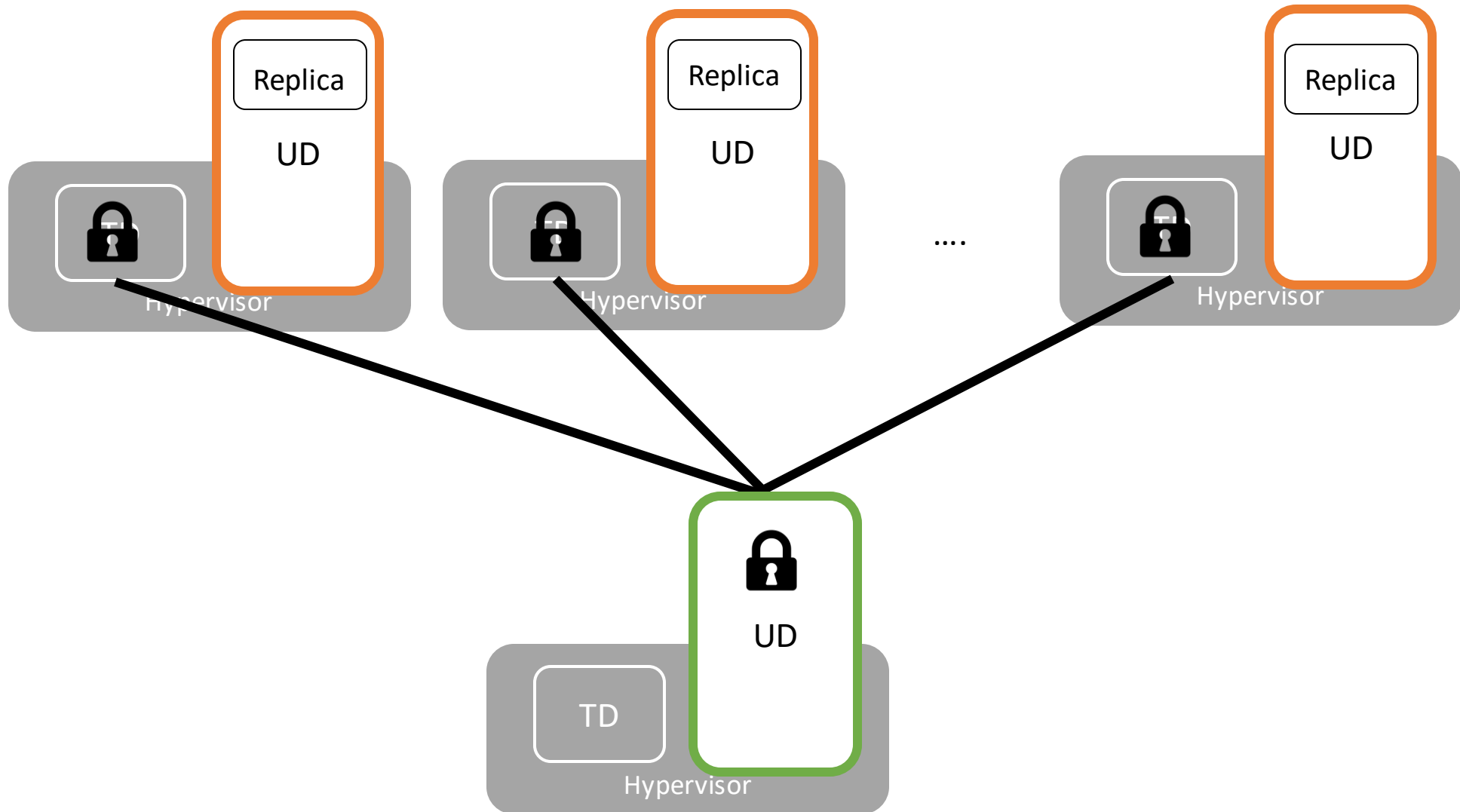
Virtualized: Sousa 2007



Virtualized: Platania 2014



Virtualized: Platania 2014



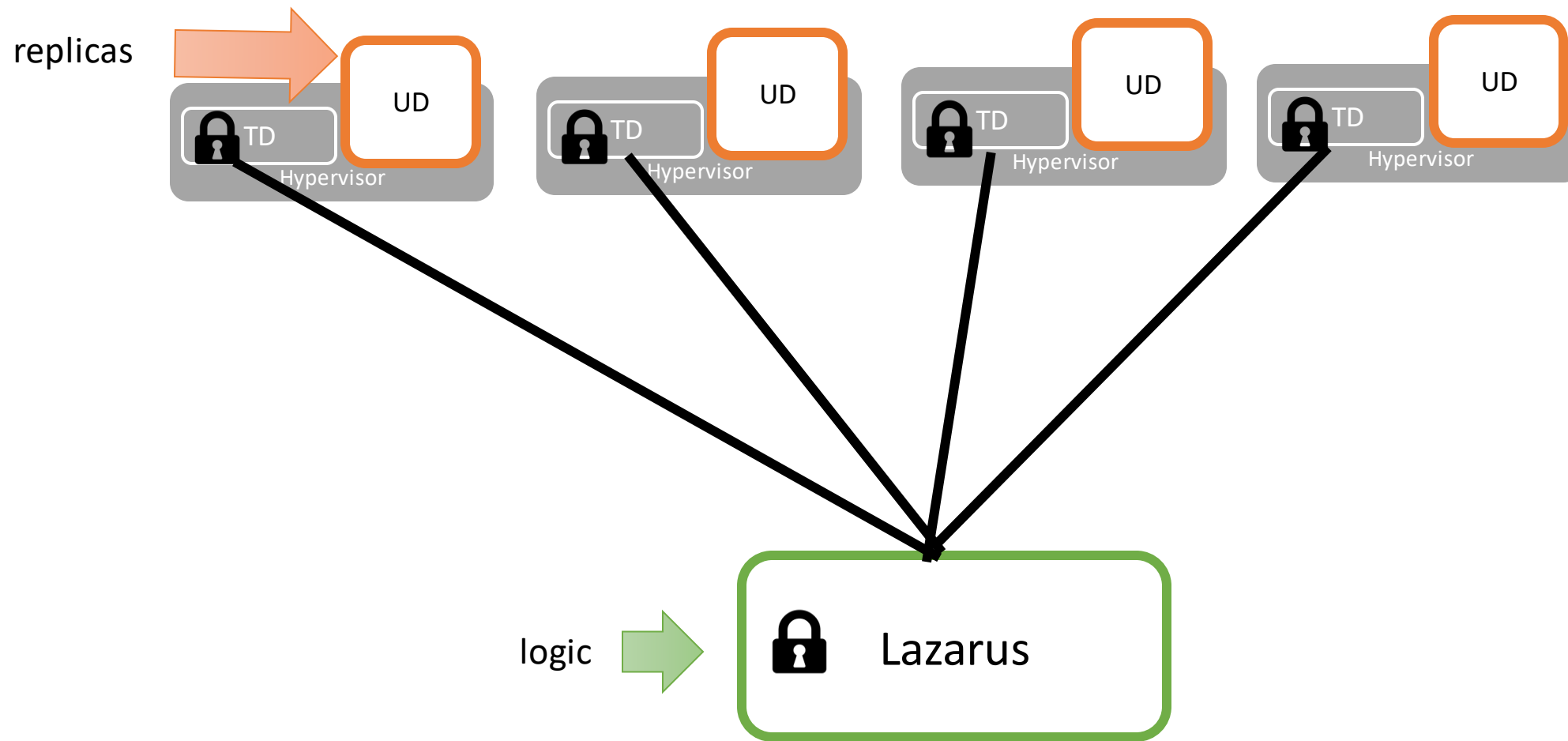
Can we do better?

- Can we reduce the assumptions?
- Can we make Byzantine fault tolerant controllers?
- Can we live without trusted parts?
- Is it heavier?

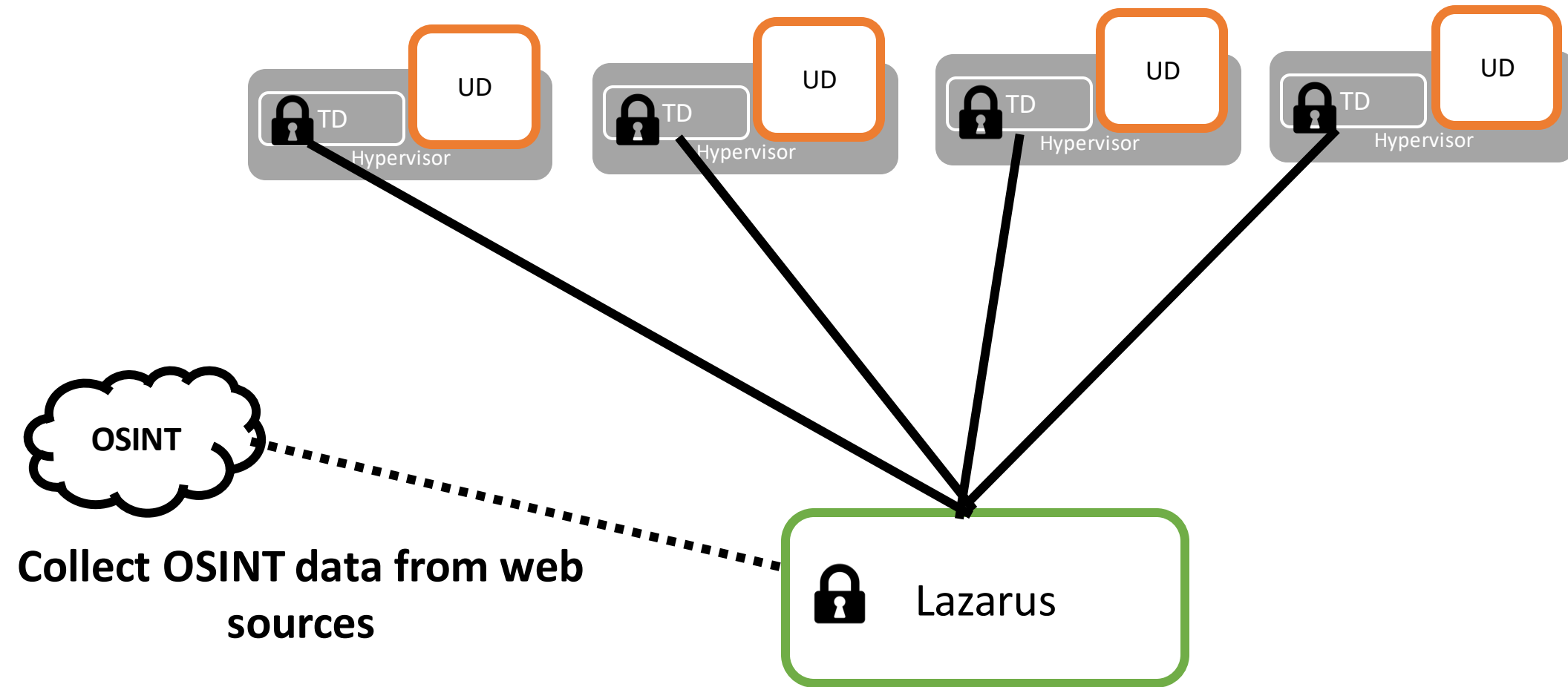
Lazarus backend

The main Lazarus contributions are waiting for re-submission
The following ideas are work in progress

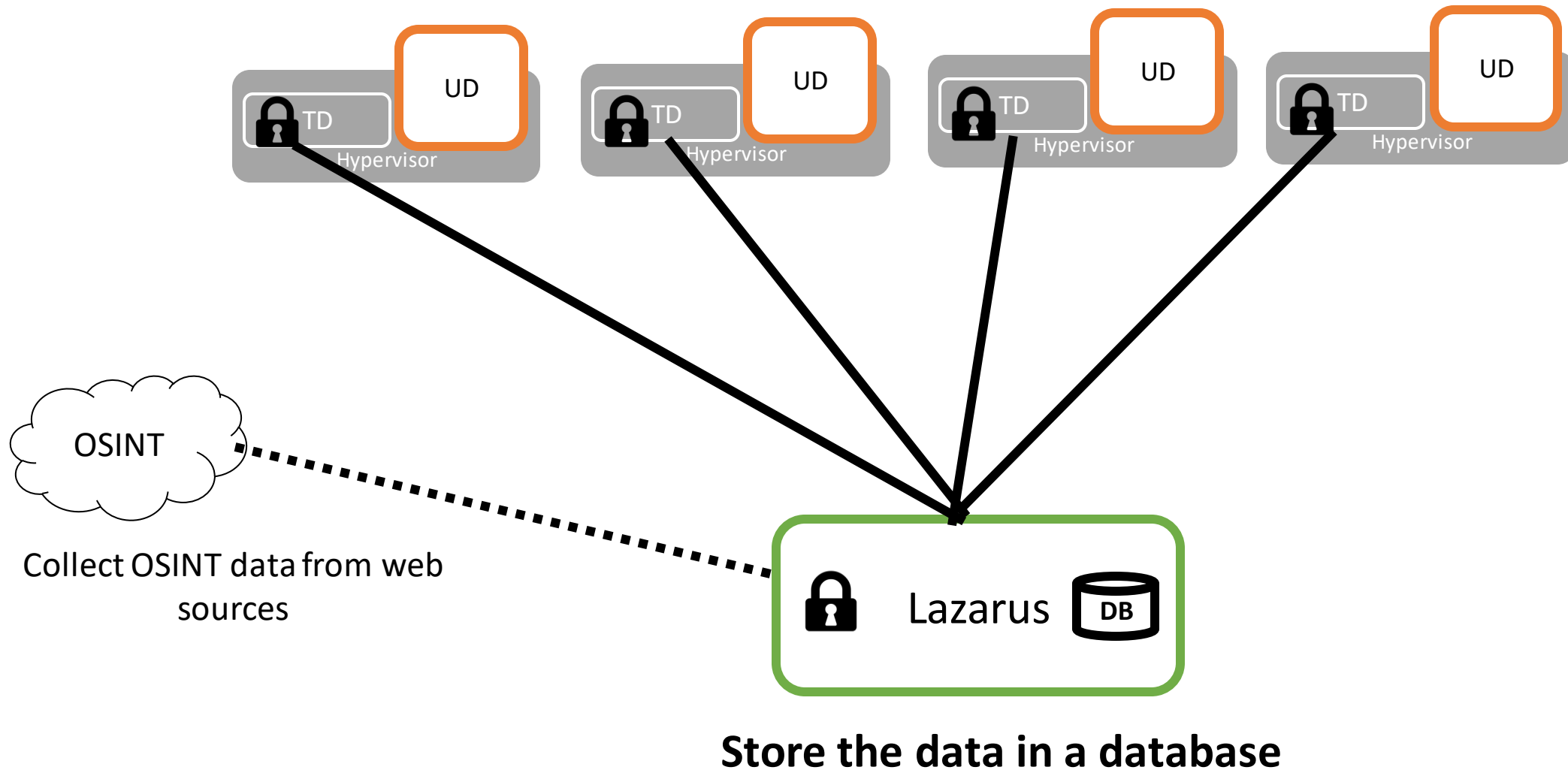
Lazarus 2018 (centralized)



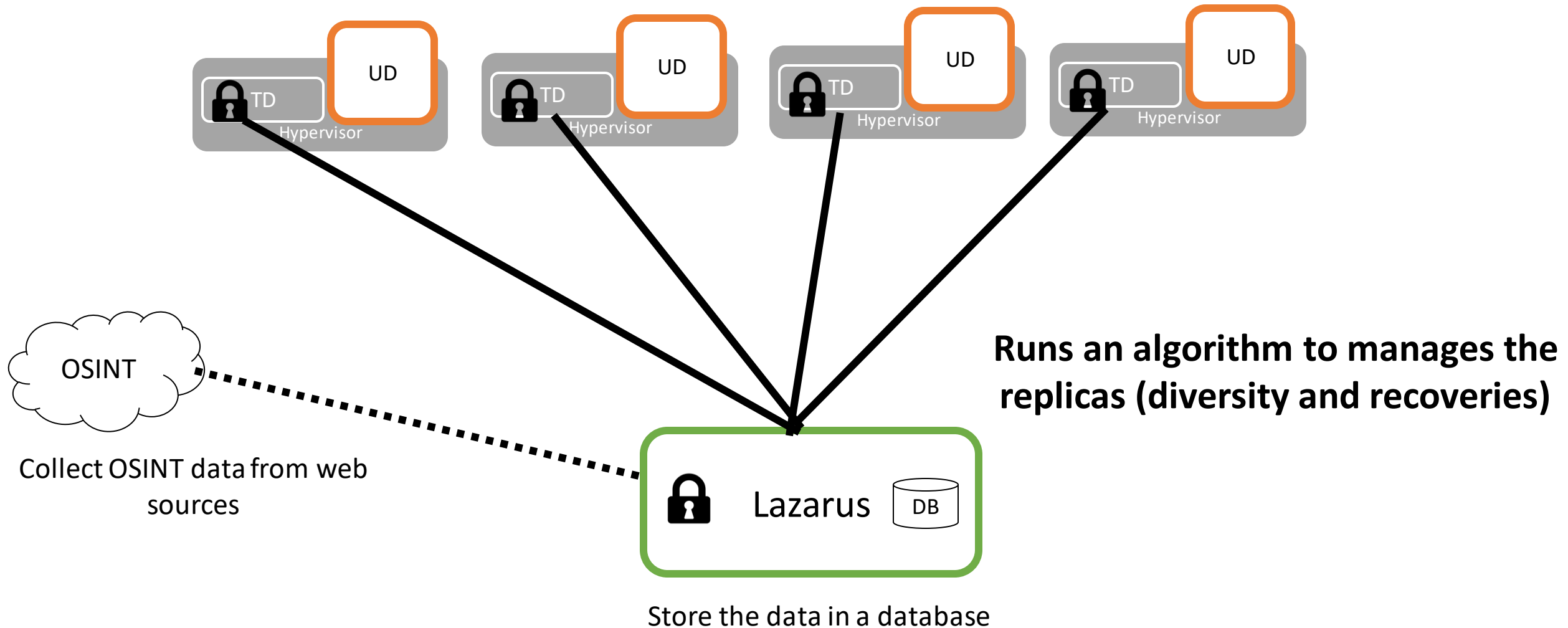
Lazarus 2018 (centralized)



Lazarus 2018 (centralized)



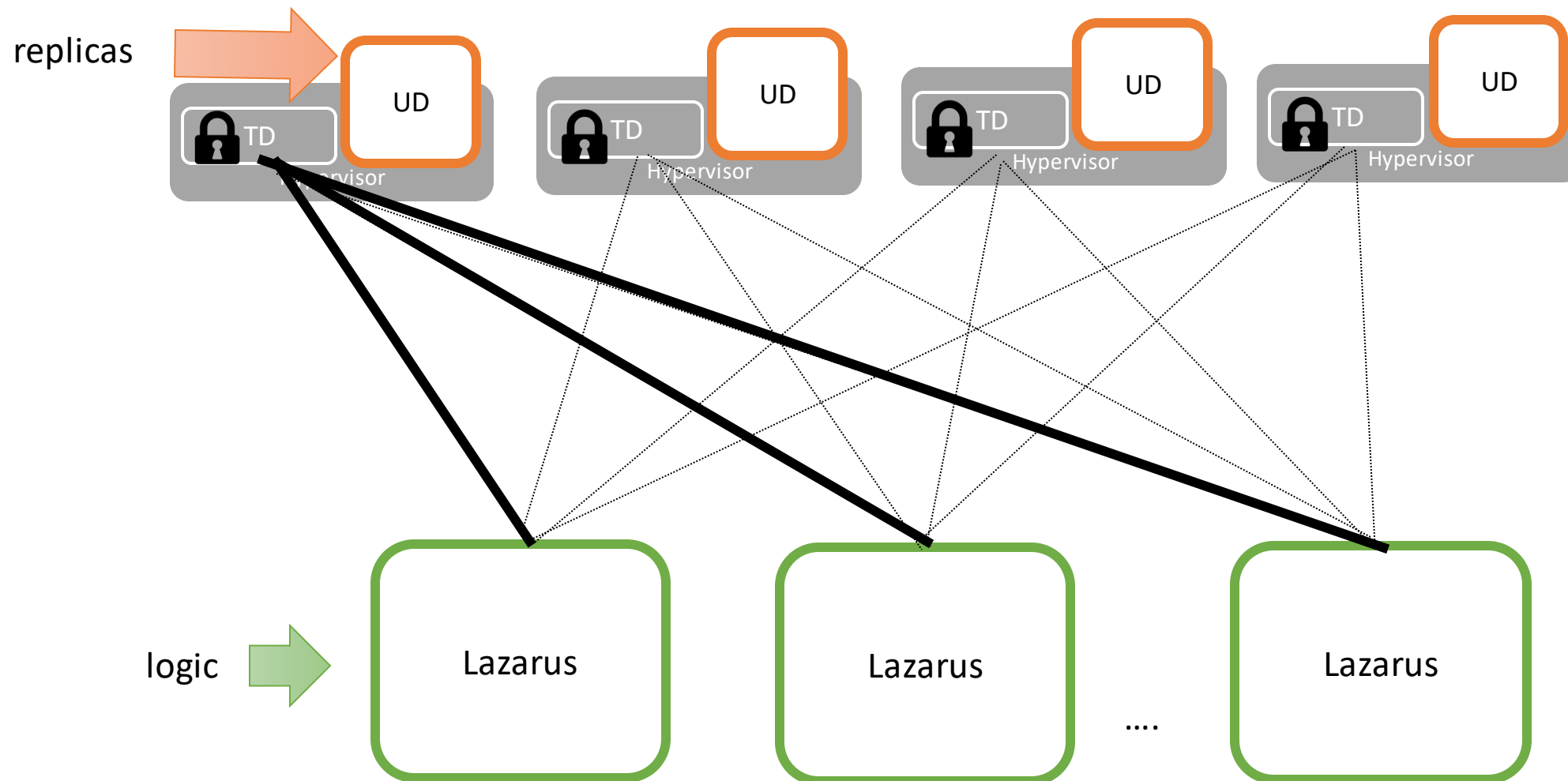
Lazarus 2018 (centralized)



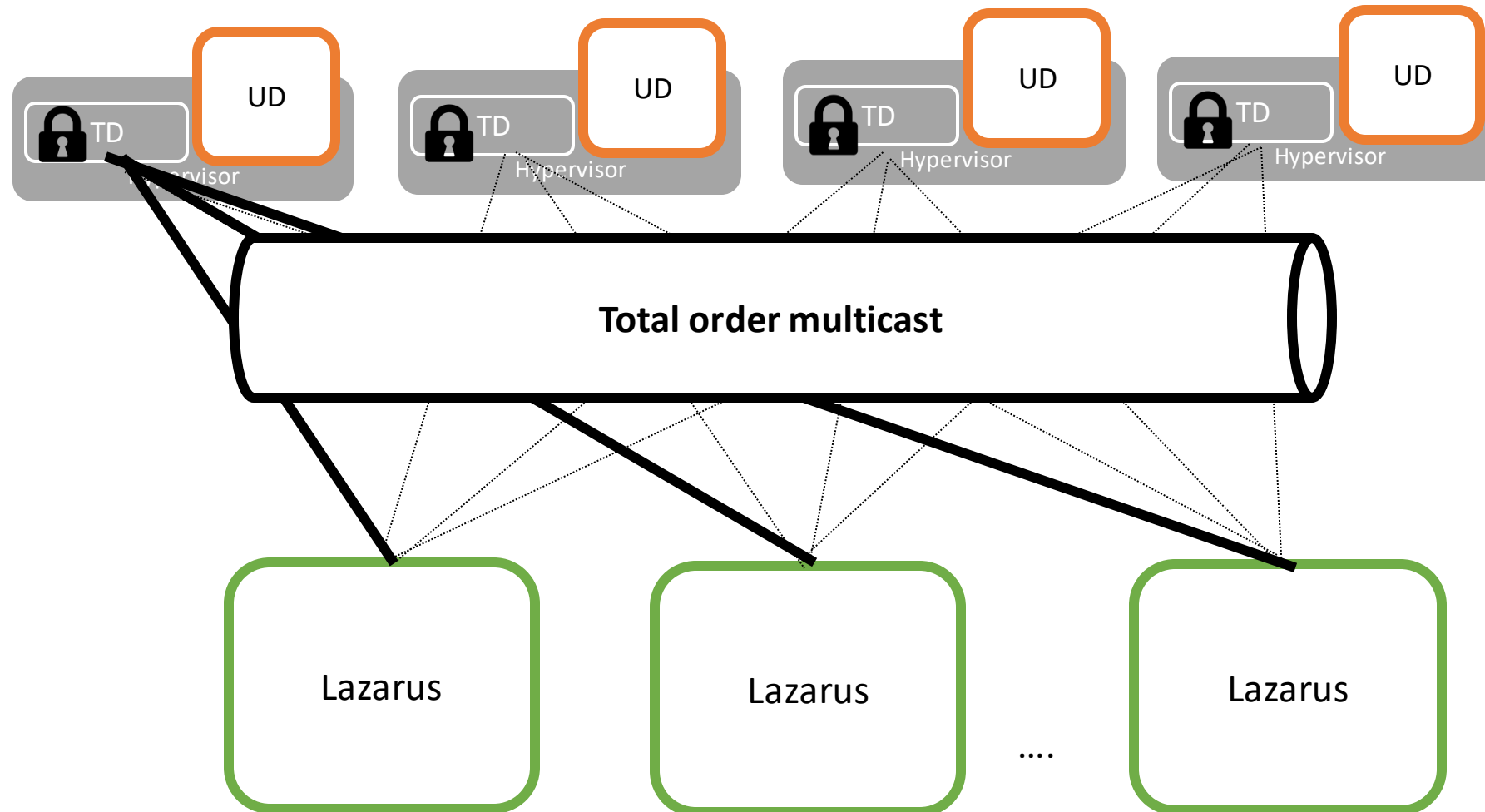
Some challenges

- Make it Byzantine fault tolerant
- Replicate OSINT database
- Implement distributed random generator

Lazarus 2018 –BFT(distributed)



Lazarus 2018 –BFT(distributed)



Distributed random generator

- Lazarus selects different OS to run in the replicated system. We developed an algorithm to select OS. **The algorithm uses some randomness**
- Lazarus needs a distributed random generator that offers:
 - Unpredictability
 - Unbiasability
 - Verifiability

Distributed random generator

Scalable Bias-Resistant Distributed Randomness

Ewa Syta*, Philipp Jovanovic[†], Eleftherios Kokoris Kogias[†], Nicolas Gailly[†],
Linus Gasser[†], Ismail Khoffi[‡], Michael J. Fischer[§], Bryan Ford[†]

*Trinity College, USA

[†]École Polytechnique Fédérale de Lausanne, Switzerland

[‡]University of Bonn, Germany

[§]Yale University, USA

In 2017 IEEE Symposium on Security and Privacy

Distributed random generator

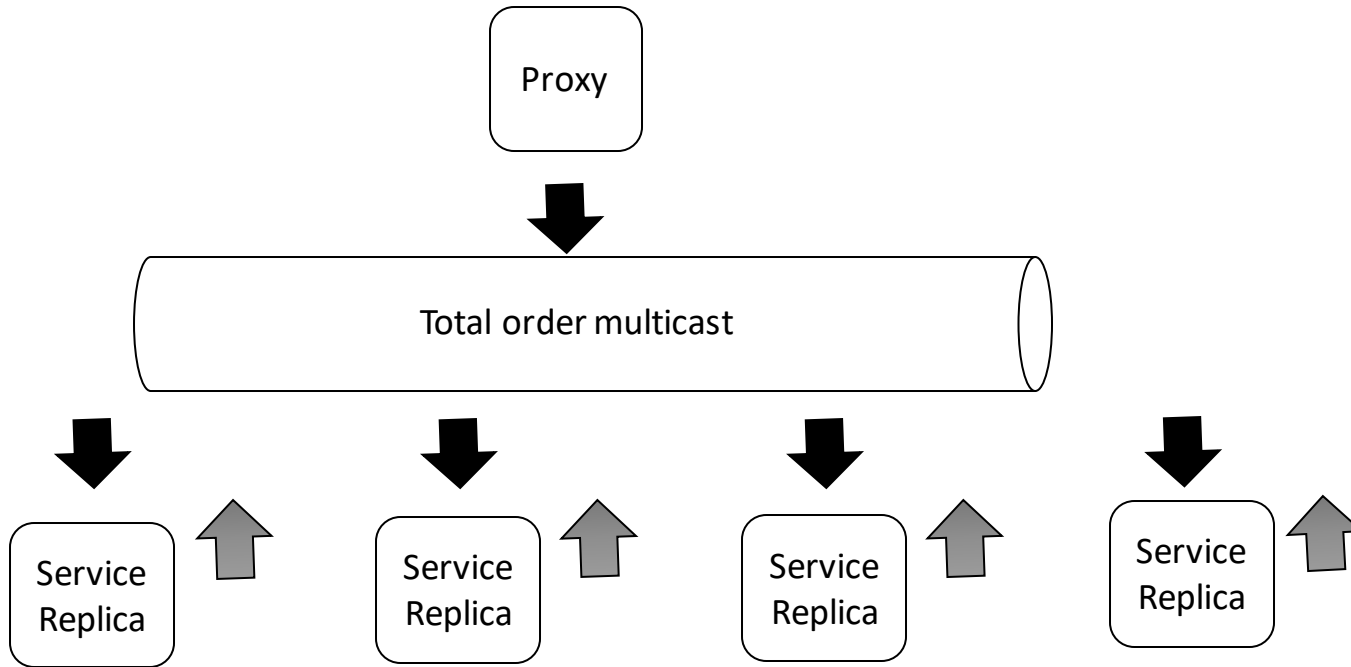
- This paper proposes a distributed random generator that provides the previous properties.
- The protocol is cooperative as all the replicas contribute to generate the random value

Replicate OSINT DB

- In the current version there is a crawler that collect vulnerability data from OSINT.
- It is expected that updates on this data flow are not so frequent
 - For example, two consecutive reads will produce the same vulnerability data.

Replicate OSINT DB

- State machine replication BFT protocols are client-driven, i.e., a client sends a request the server replicas respond to the client.



Replicate OSINT DB

Survivable SCADA Via Intrusion-Tolerant Replication

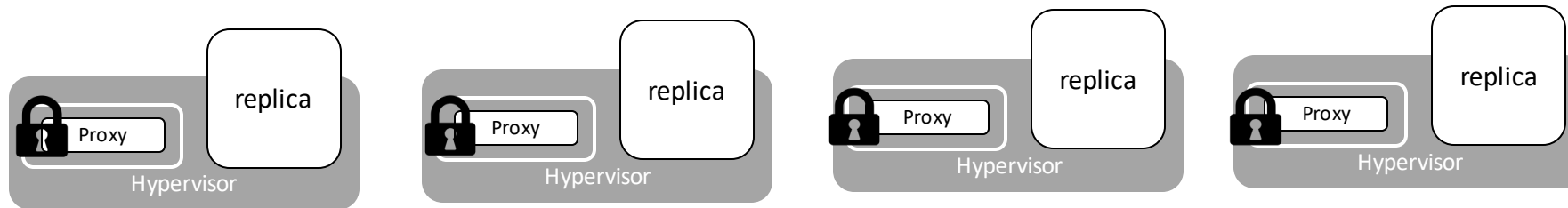
Jonathan Kirsch, Stuart Goose, Yair Amir, *Member, IEEE*, Dong Wei, *Member, IEEE*, and
Paul Skare, *Member, IEEE*

In 2014 IEEE Transactions on Smart Grid

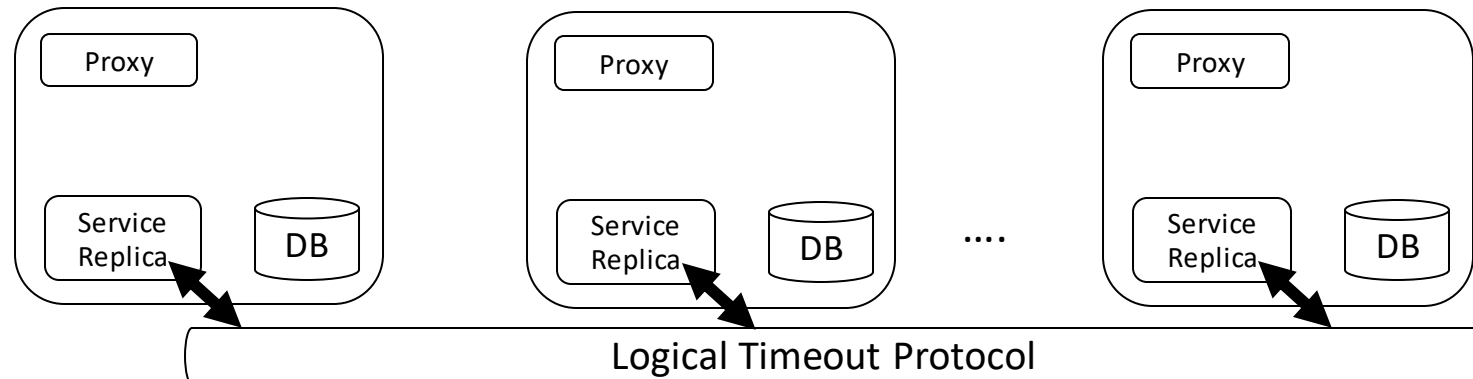
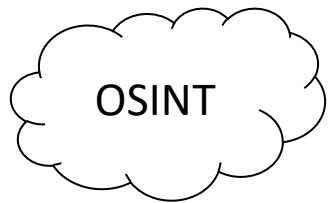
Replicate OSINT DB

- One of the paper contributions is a logical timeout protocol (LTP).
- LTP allows, without strict clock synchrony assumptions, that different replicas trigger a timeout at the same logical time
 - The protocol minimizes the differences between real clock time
- This allow the replicas to poll the OSINT sources at the same (logical) time

Replicate OSINT DB (implementation)



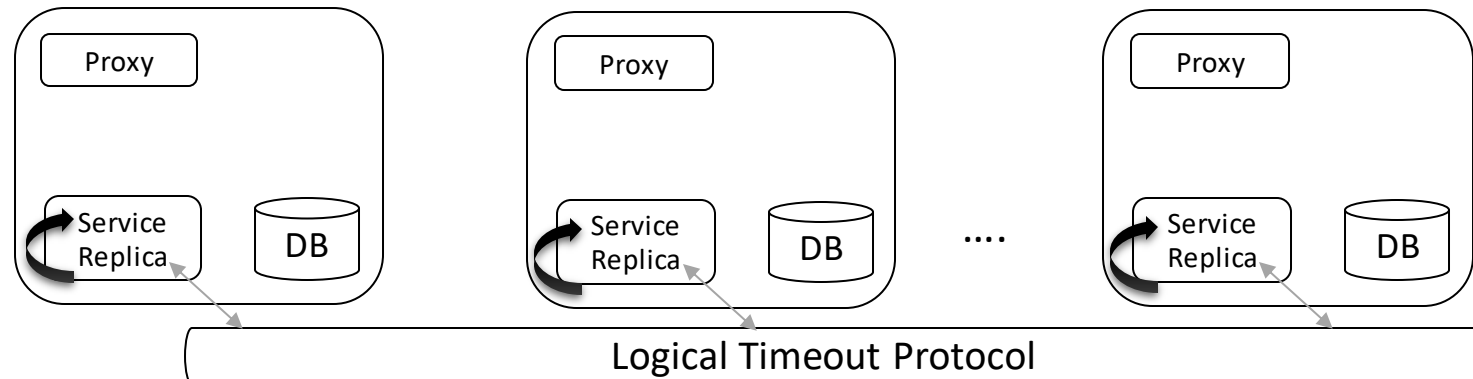
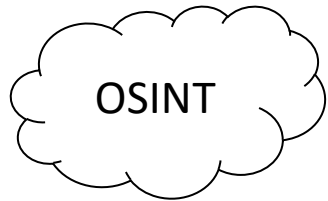
The LTP is running all the time to keep the controller replicas synchronized



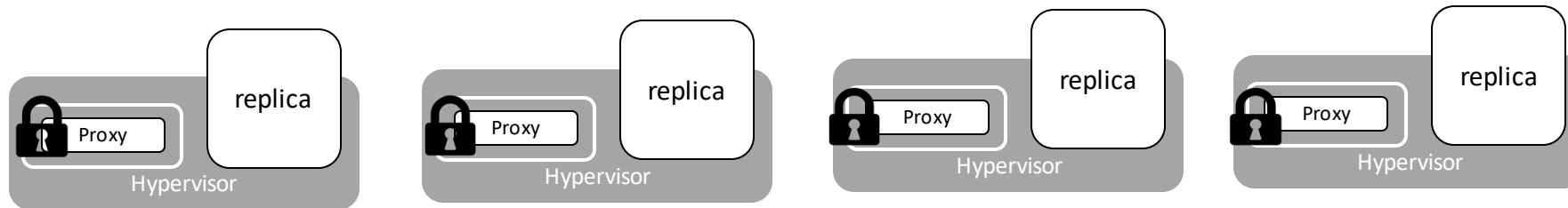
Replicate OSINT DB (implementation)



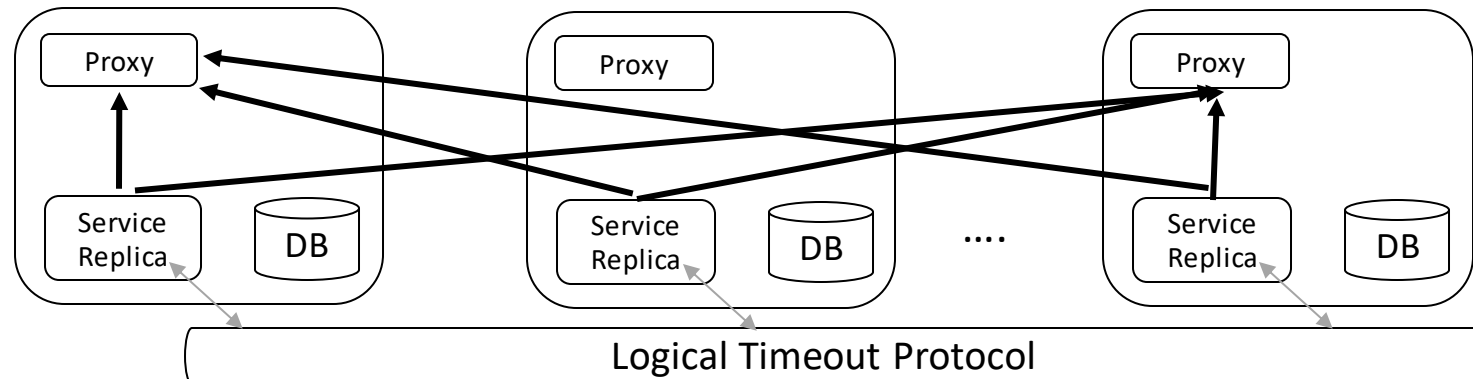
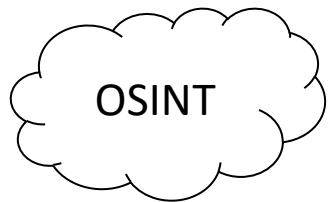
Each replica receives $f+1$ timeouts and triggers an action



Replicate OSINT DB (implementation)



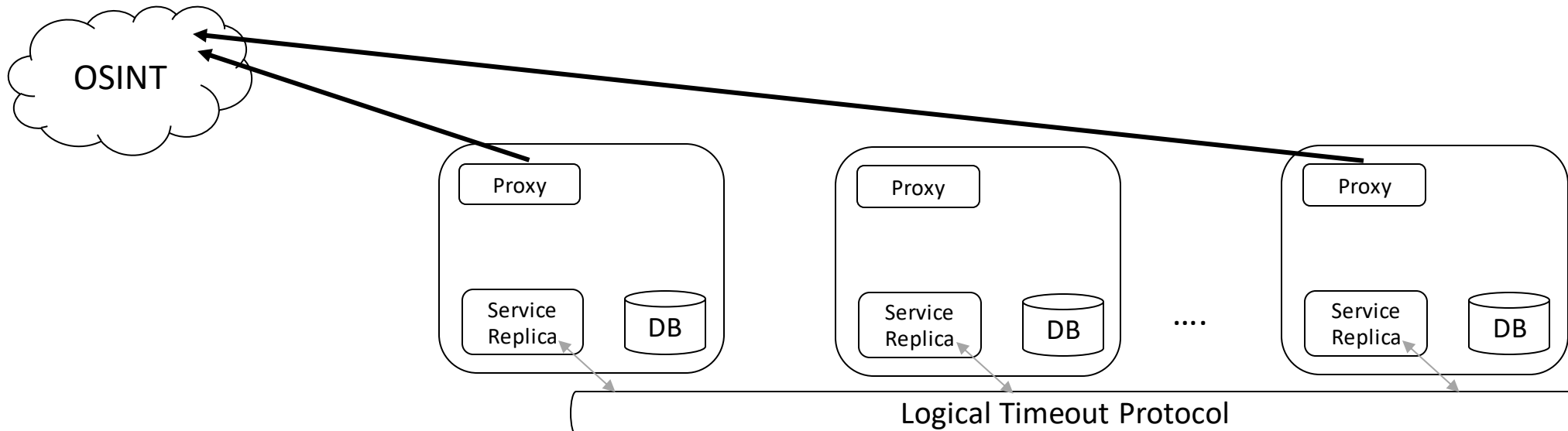
The replicas decide which Proxies will fetch OSINT data



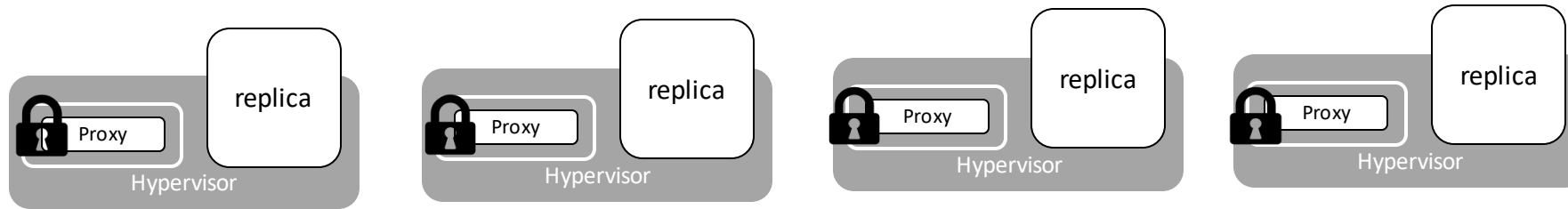
Replicate OSINT DB (implementation)



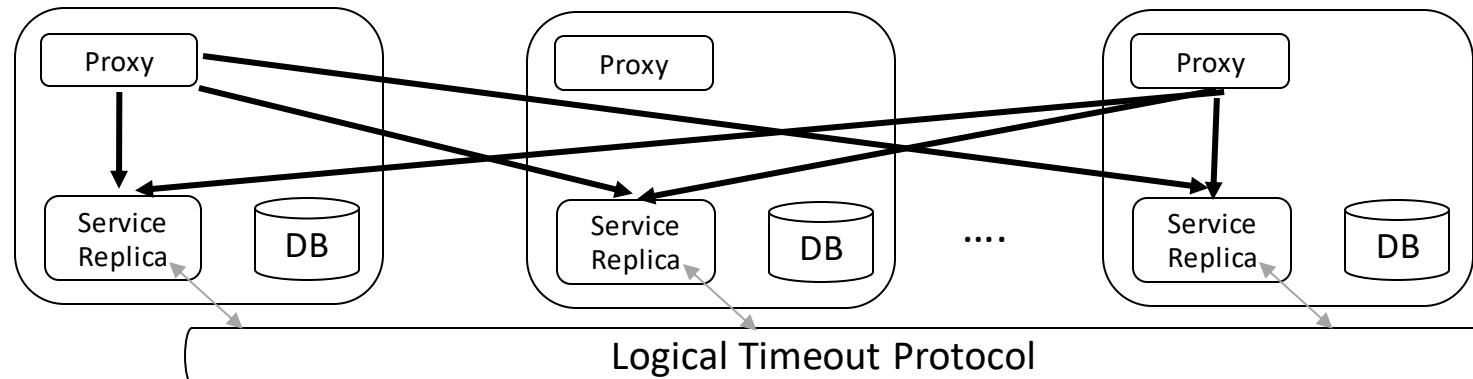
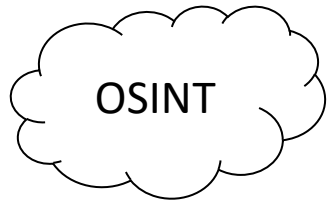
The replicas decide which Proxies will fetch OSINT data



Replicate OSINT DB (implementation)



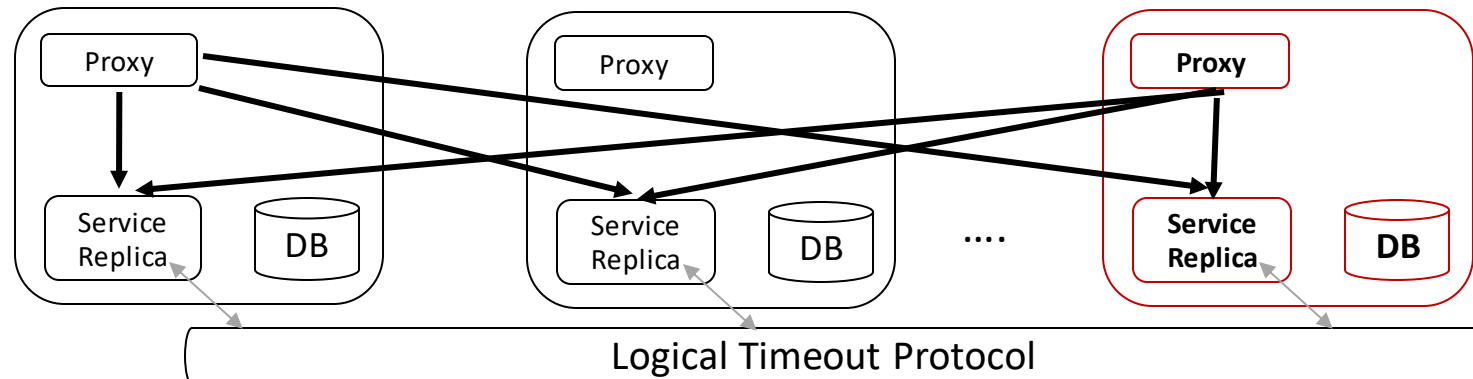
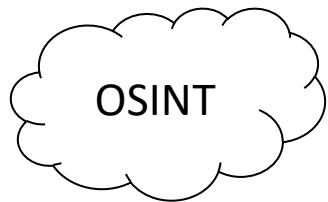
Then each Service Replica waits for $f+1$ equal "data"



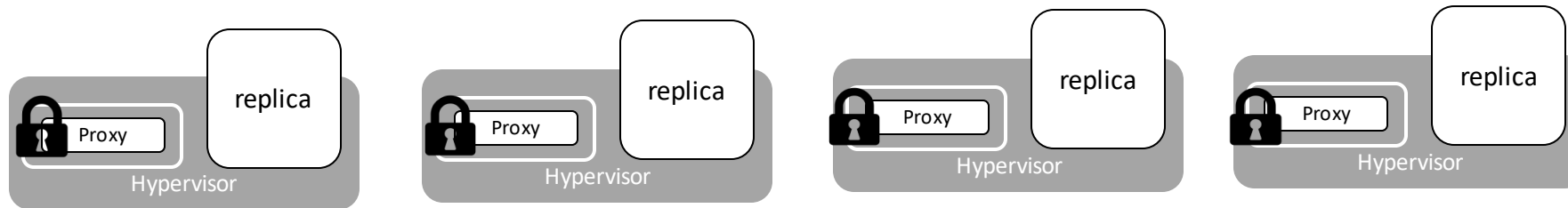
Replicate OSINT DB (implementation)



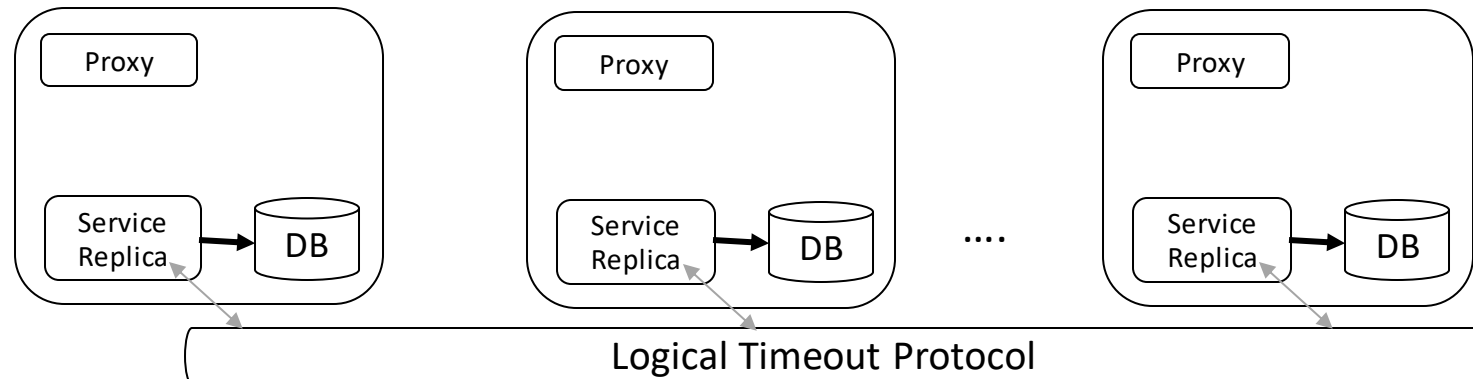
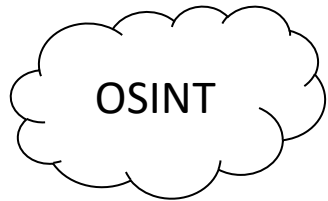
If the data does not match => use another Lazarus controller to decide which data is correct



Replicate OSINT DB (implementation)



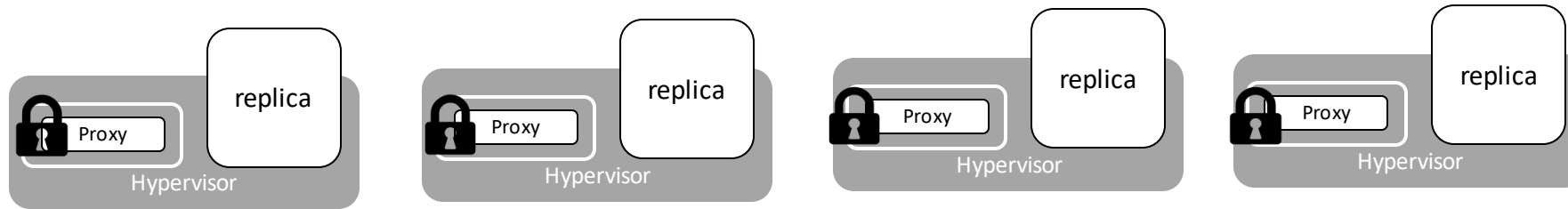
Then it is stored in the DB



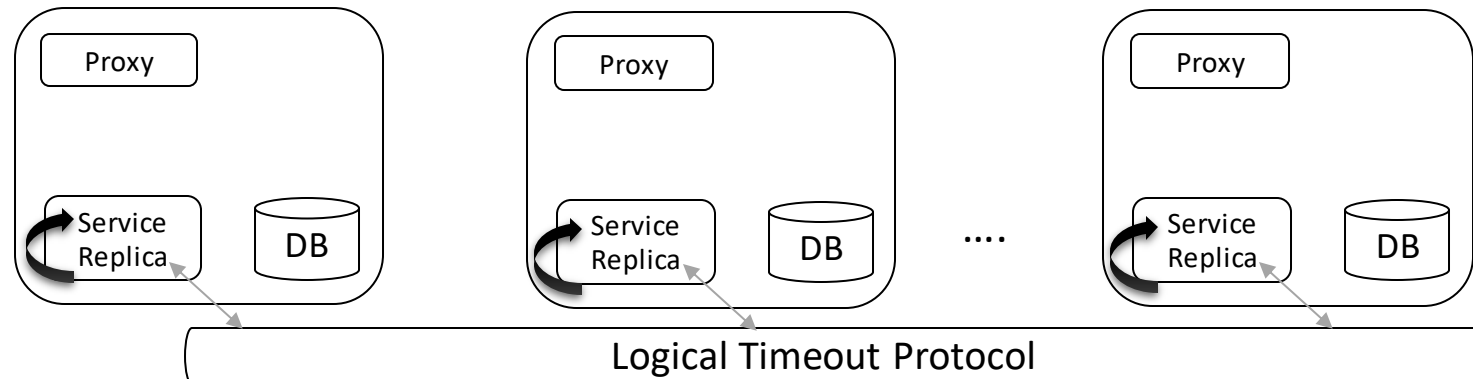
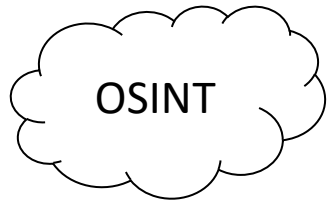
Recoveries

- We use a similar protocol to trigger recoveries.
- In this case, the controller replicas communicate with the trusted proxy to recover its replica.

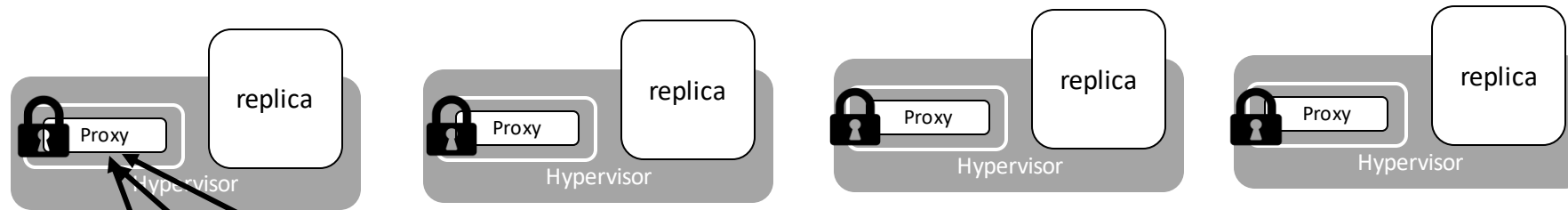
Recoveries



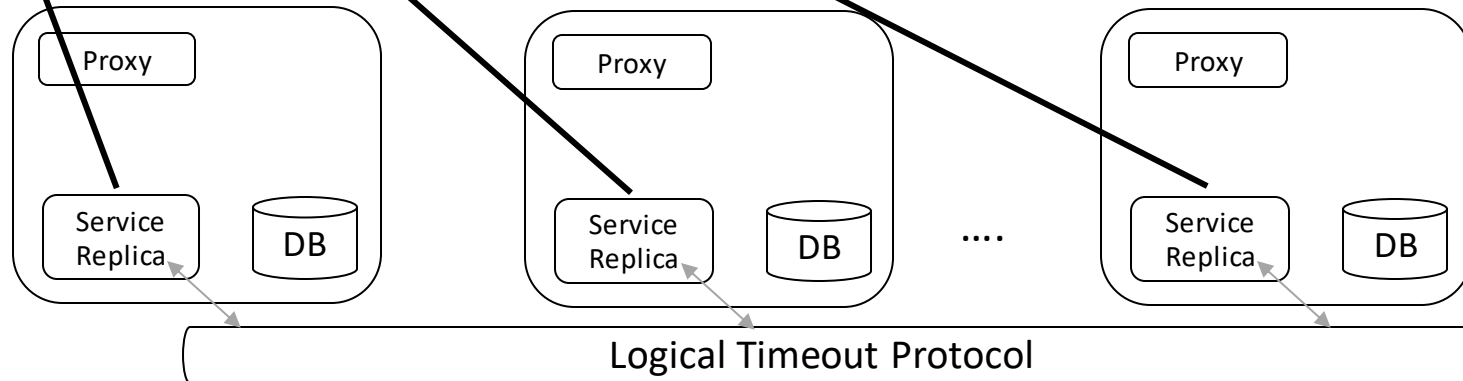
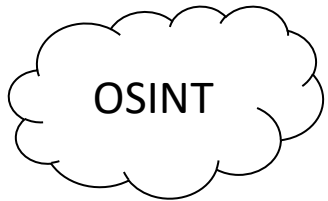
Each replica receives $f+1$ timeouts and it trigger an action



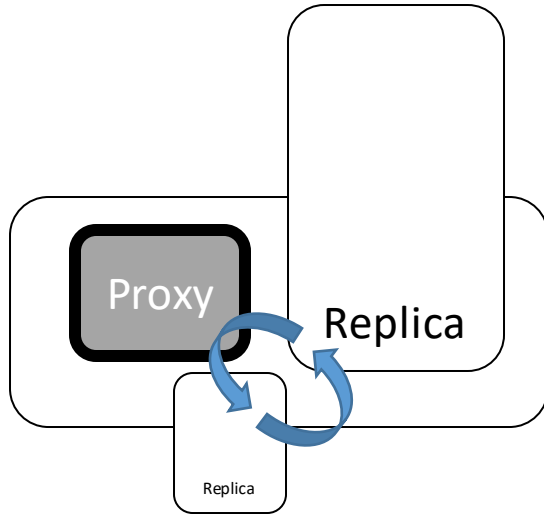
Recoveries



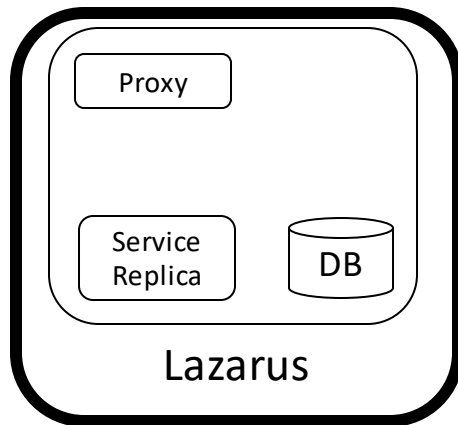
The controller replicas know which trusted proxy need to contact to recover the replica



Recoveries



A replica is restarted only when the **Proxy** receives $f+1$ restart requests from the **Service Replicas**



No malicious **Service Replica** can restart a **Replica**

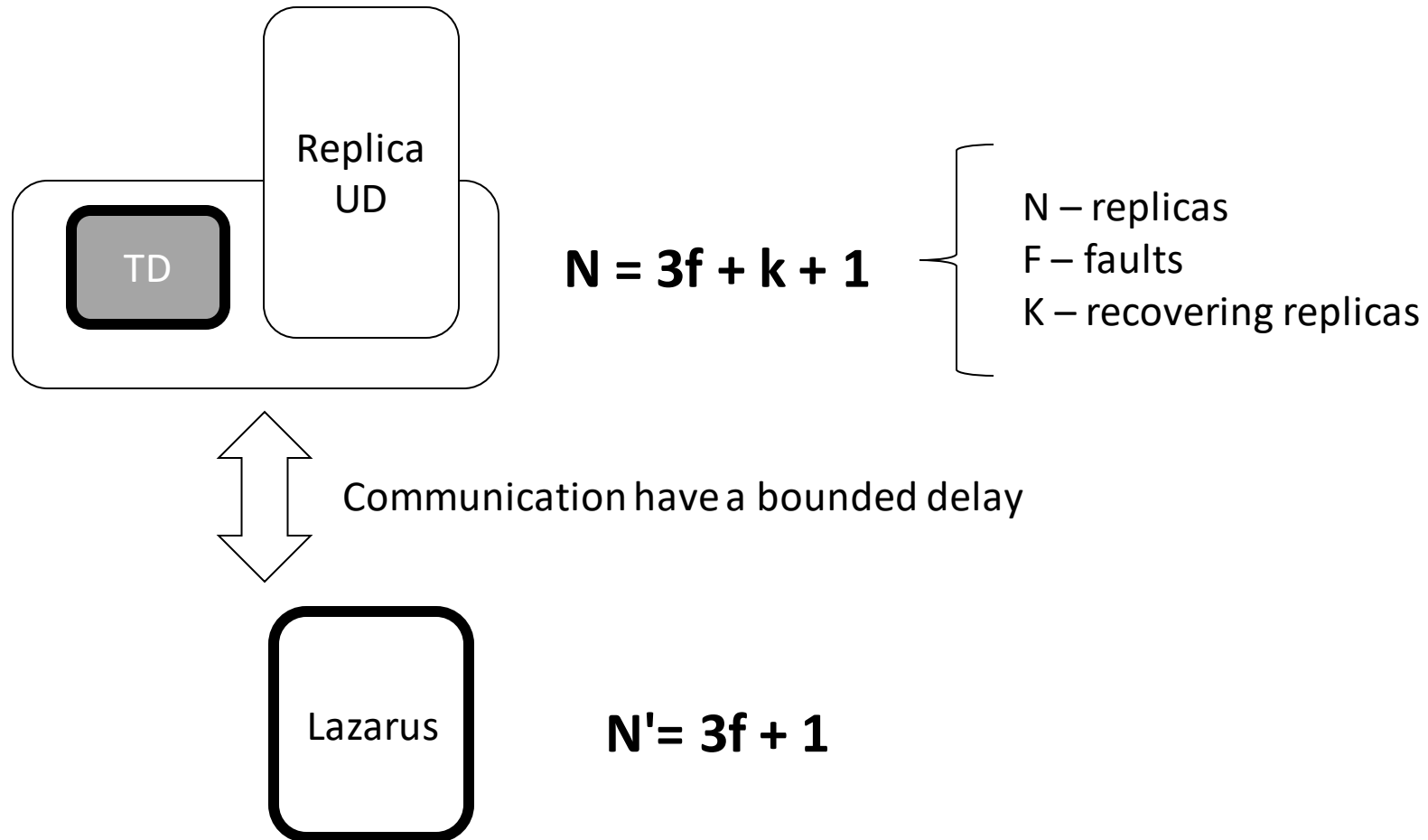
Conclusions

- Can we reduce the assumptions?
 - **Yes**, we no longer assume the whole controller as trusted, no real-time assumptions
- Can we make Byzantine fault tolerant controllers?
 - **Yes**, to some extent
- Can we live without trusted parts?
 - **No**, each node needs a tamper proof component that ensures the correct behavior even if the of the replica is compromised
- Is it heavier?
 - **Yes**, it is the price of BFT replication

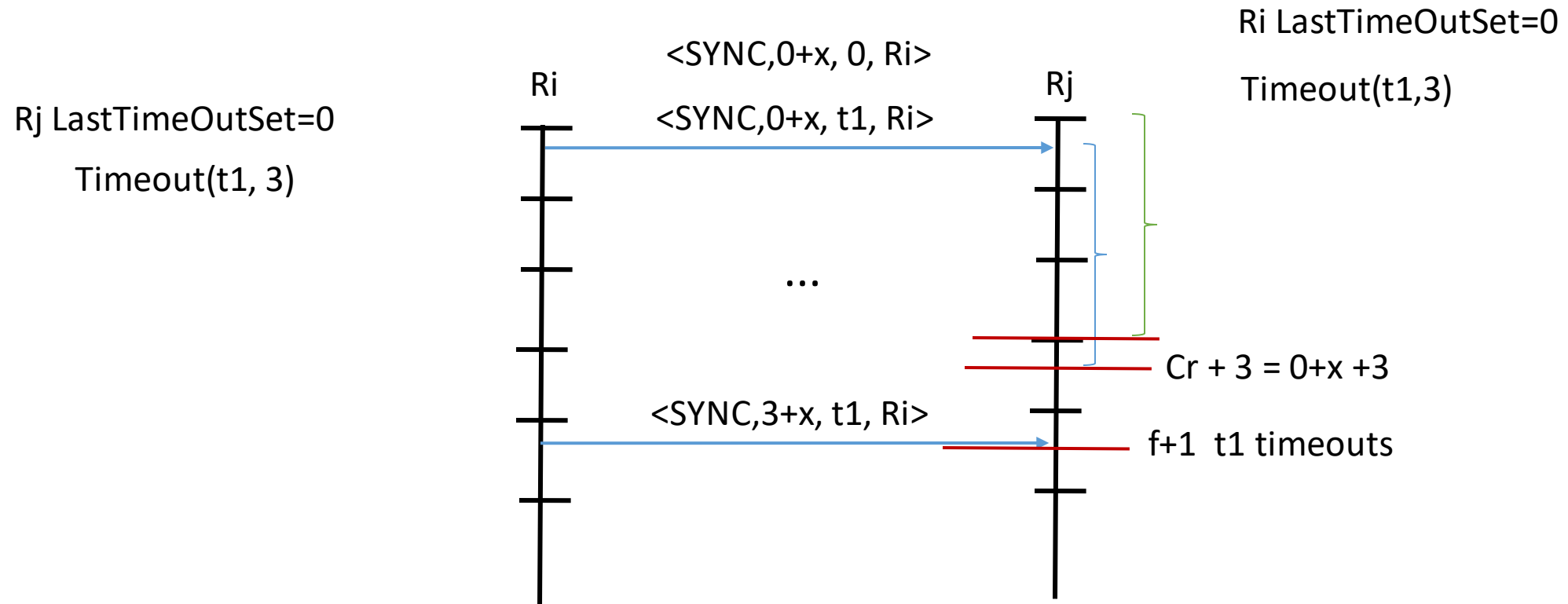
Questions ?

Thanks

Lazarus 2018 –BFT system model



LTP Protocol (overview)



Virtualization advantages

- Fast recoveries, it allows shadow replicas to be ready on time
- Provides security layers with the hypervisor

Wrap up

- The execution system is intrusion-tolerant
- The controller system is Byzantine fault-tolerant
 - It tolerates f Byzantine faults in "one shot"
 - To recover Byzantine node one admin needs to restart the machine properly