# Themen für Abschlussarbeiten

Mail an **berger@cs.fau.de**

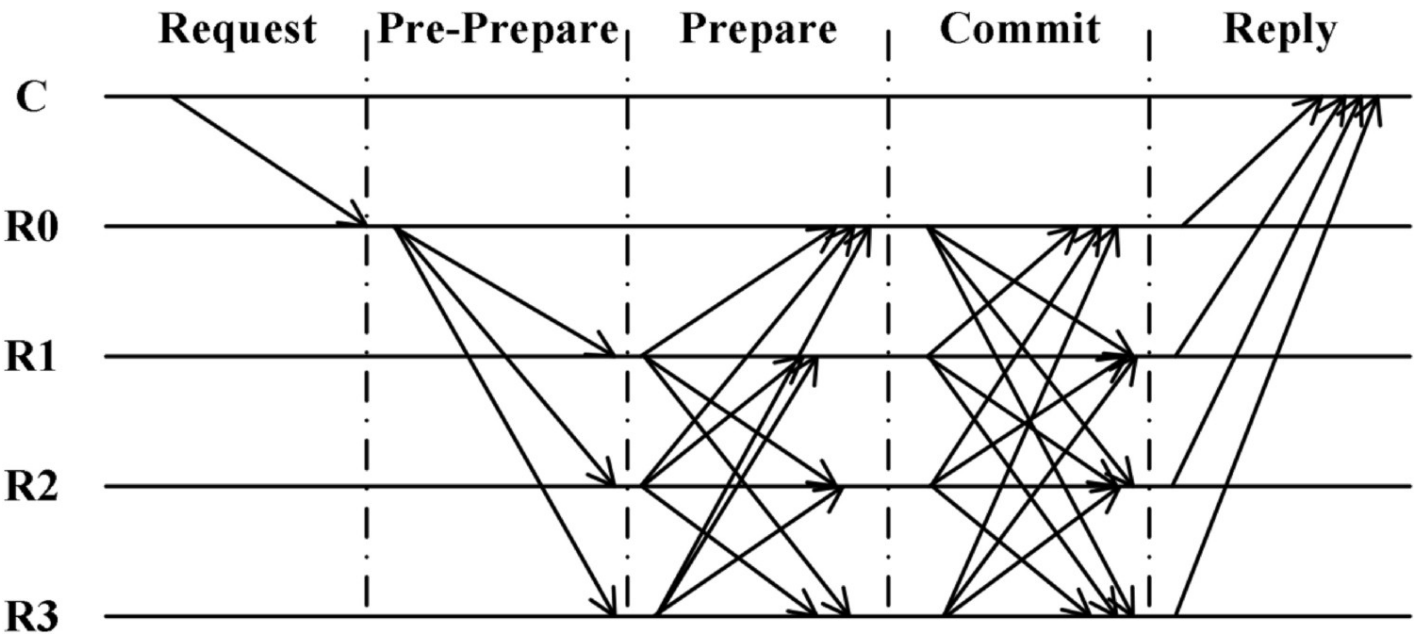

**Lehrstuhl für Informatik 4**
Systemsoftware

Friedrich-Alexander-Universität
Technische Fakultät

# Byzantine Fault Tolerance

- **Byzantinisch** = “Willkürliche” Fehler im verteilten System
- Oft wird ein **Angreifer** im System modelliert

# PBFT (Practical Byzantine Fault Tolerance)




- Erstes **praktische** BFT Protokol
- Korrekt und bewährt
- Für **Blockchains** nur begrenzt geeignet

# Hintergrund: BFT Forschung am i4

- Modernes **Rust-basiertes** BFT Framework namens “Themis”
  - Implementiert z.B. PBFT
  - multi-threaded und memory-safe

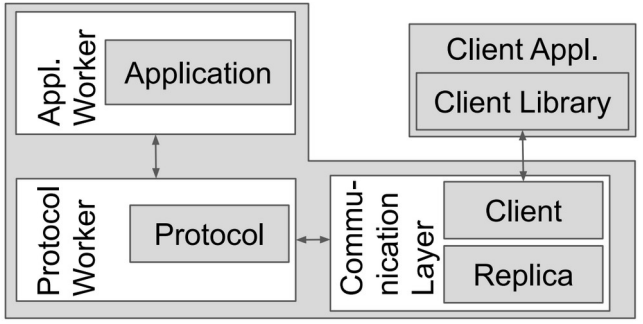
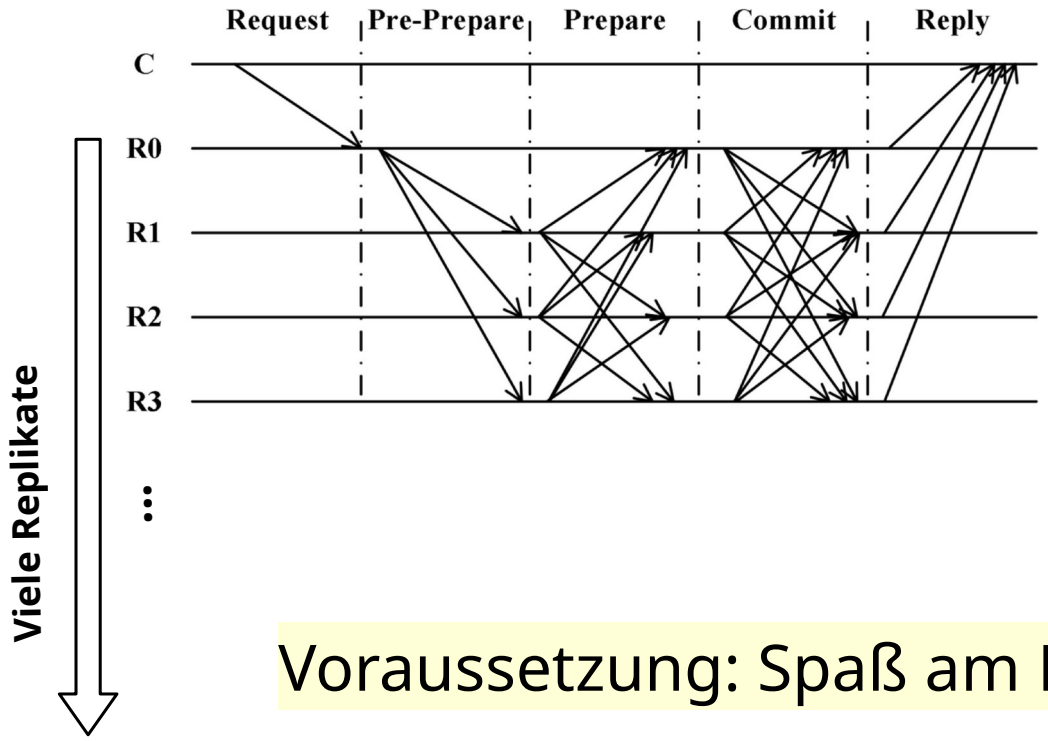



**Figure 1.** THEMIS framework components.

- BFT2Chain Forschungsprojekt
  - **Thema: BFT in Blockchains**
  - Skalierbarkeit, Validierung, Modellierung

# Skalierbarkeit (#Replikate) von BFT für Blockchain

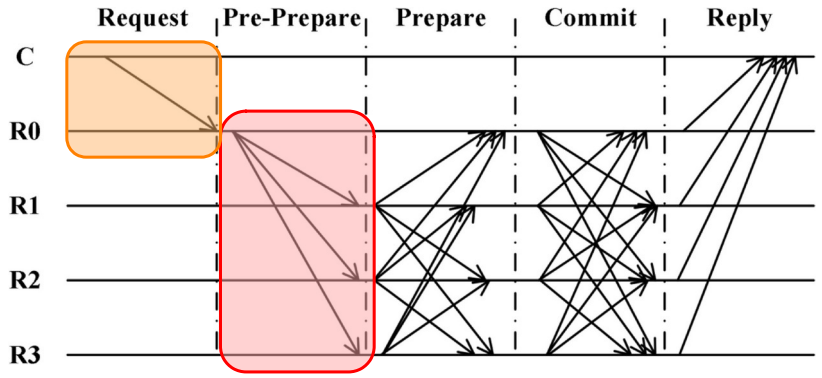



Voraussetzung: Spaß am Programmieren

# Thema 1: Verbesserung der Skalierbarkeit von Themis

- **Problem**: Leader ist Flaschenhals für Performance in Blockchains
  - Alle Clients schicken Requests an Leader
  - Leader muss Batches an alle Replikate verteilen

    → Leader hat am meisten Arbeit
- **Ansatz:**
  - “Load Balancing” der Client Requests
  - Trennung von “Daten” und “Konsensus”



# Thema 1: Verbesserung der Skalierbarkeit von Themis

- “Load Balancing” der Client Requests
  - Einführung eines “Mempools”
    - Requests können an verschiedene Replikate geschickt werden
- Trennung von “Daten” und “Konsensus”
  - Der Mempool verteilt Daten der Requests konsistent an alle Replikate
    - Konsensus ordnet nur “Metadaten” (Hashes von Batches)
    - Protokolllogik bleibt unverändert
    - Integration von Schwellwertsignaturen (Threshold-Signatures)

# Thema 1: Verbesserung der Skalierbarkeit von Themis

- **Forschungsfragen**:
  1) Wie weit lässt sich die Skalierbarkeit von PBFT verbessern ohne die Protokolllogik zu modifizieren (*Performance messen*)?
  2) Wie modular lassen sich diese Veränderungen realisieren (i.e., eigene Kommunikationsschicht für das Verteilen von Daten)?
- **Methodik**:
  1) Konzeptuelle Ausarbeitung des Ansatzes in der Thesis
  2) Implementierung eines Prototypen im bestehenden Rust Framework
  3) Evaluation durch Experimente (mithilfe Netzwerksimulators)

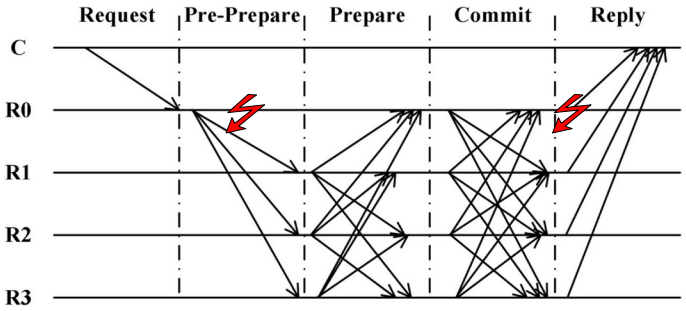
# Thema 1: Literatur und weiterführende Links

- Der PBFT Algorithmus erklärt im Orginalpapier hier

  (Auch relevant ist die Umsetzung der Technik “big request” optimization, hier erklärt auf Seite 64 § 5.1.5)
- Mempool mithilfe eines DAG und Consistent Broadcasts:
  - Narwhal and Tusk: a DAG-based mempool and efficient BFT consensus
- Kurze Einführung ins Themis Framework hier
- Repo zum Framework:
  - https://github.com/ibr-ds/themis

# Validierung von BFT für Blockchain

- 2 interessante Themen im Bereich Validierung
  - Jeweils Integration einer bestimmten Testmethodik **{Fuzzing, Twins}** in ein existierendes Rust-basiertes BFT Framework (Themis) zur Härtung




Voraussetzung: Spaß am Programmieren

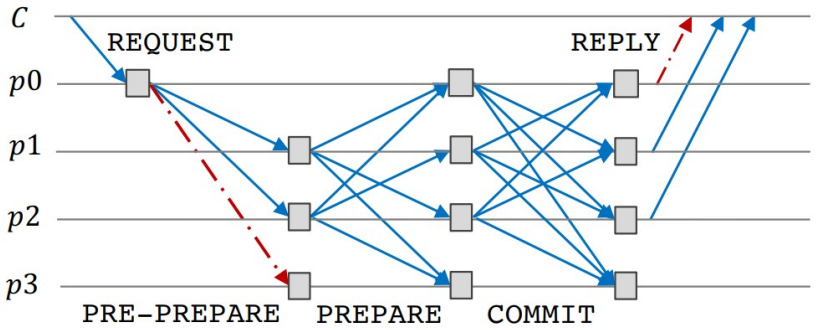
# Thema 2: Fuzzing des Themis BFT Frameworks

- **Problem**:
  - BFT Protokolle sind sehr komplex
  - Implementierungen oft buggy
  - “Fehlertoleranz”-Validierung herausfordernd da Fehler oftmals “maskiert” werden

- **Ansatz**:
  - *Fuzzing* ist eine **randomisierte Technik** für Fehlerinduktion um Schwachstellen zu finden

# Thema 2: Fuzzing des Themis BFT Frameworks

- **Fuzzing BFT Protocols**
  - Zufällige Auswahl Byzantinischer Replikate
  - Lokale Mutationen von Nachrichten
    - Gezielt, entsprechend “Protokolllogik”
  - Message Handler, Deserialisierung




- Testszenarien und Fehleranalyse
  - **Härtung** des Themis Frameworks

# Thema 2: Fuzzing des Themis BFT Frameworks

- **Forschungsfragen**:
  1) Wie kann Fuzzing angepasst werden um speziell “realistisches” Byzantinisches Fehlverhalten nachzubilden? (i.e. Konstruktion der Mutatoren)
  2) Themis ist bereits memory-safe. Aber ist das genug? Welche Fehlertypen findet man mithilfe von Fuzzing (i.e., Concurrency, Message Handling, Deserialization, ..)
- **Methodik**:
  1) Konzeptuelle Ausarbeitung der Fuzzing Technik in der Thesis
  2) Implementierung eines Werkzeugs für bestehenden Rust Framework
  3) Evaluation durch eine automatisierte Fehleranalyse (z.B. Test Coverage bei absichtlich eingebauten Fehlern im Code)

# Thema 2: Literatur und weiterführende Links

- Der PBFT Algorithmus erklärt im Orginalpapier hier
- Eine Fuzzing Methode für BFT hier erklärt:
  - Randomized Testing of Byzantine Fault Tolerant Algorithms
  - (Adaptierung dieser Methode für Themis & Fehleranalyse)
  - Repo zu einem Java-basierten Software Artifakt: https://github.com/burcuku/byzzfuzz-pbft
- Kurze Einführung ins Rust-basierte Themis Framework hier
  - Repo zum Framework: https://github.com/ibr-ds/themis

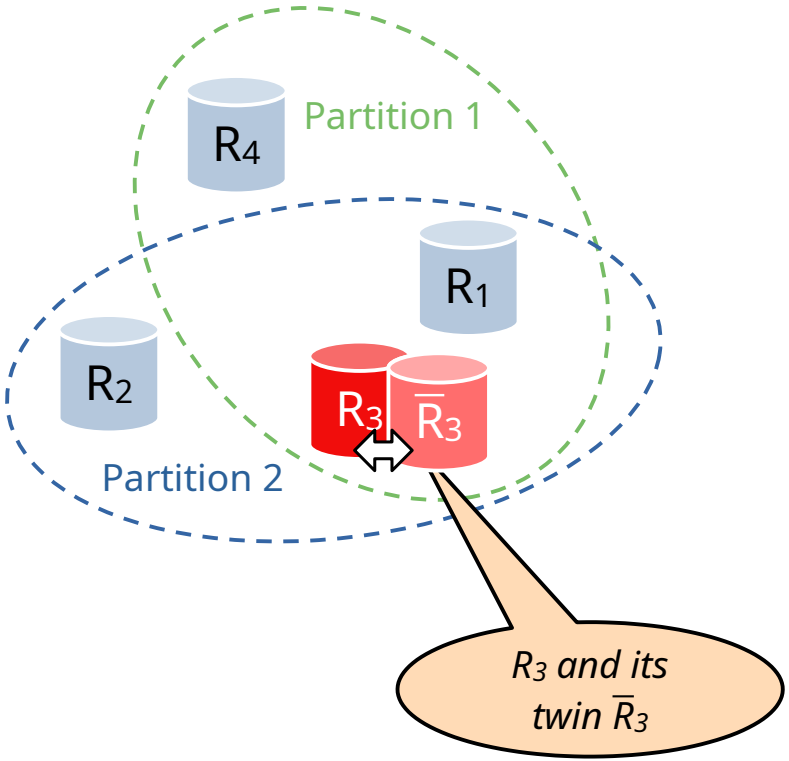
# Thema 3: Twins Testing im Themis BFT Framework

- **Problem**:
  - BFT Protokolle haben manchmal Schwachstellen
  - Byzantinische Angriffe können komplex (und schwer zu finden) sein

- **Ansatz**:
  - *Twins* ist ein automatisierter **Testgenerator** um Byzantinische Angriffe zu finden

# Thema 3: Twins Testing im Themis BFT Framework

- **Twins Testing BFT Protocols**
  - Auswahl von Partitionen des Systems
  - Einfügen eines “Zwillings” mit identischer ID und (priv, pub)-Keys
  - Zwilling sorgt für Equivocation, Double Voting, vergessene Protokollzustände




- Testszenarien und Fehleranalyse
  - **Härtung** des Themis Frameworks

# Thema 3: Twins Testing im Themis BFT Framework

- **Forschungsfragen**:
  1) Wie kann das sog. “Twins Testing” (siehe nächste Folie) für das PBFT Protokoll (in Themis) angepasst werden um Byzantinische Angriffe automatisiert zu finden?
  2) Welche Schwachstellen kann man mit Twins finden? Ist der Ansatz auch mit Performance Messungen kombinierbar? (i.e., System performance under attack?)
- **Methodik**:
  1) Konzeptuelle Ausarbeitung der Twins Technik für Themis in der Thesis
  2) Implementierung der Methode für das bestehende Themis Framework
  3) Evaluation z.B. durch eine automatisierte Fehleranalyse (z.B. Test Coverage bei absichtlich eingebauten Fehlern im Code)

# Thema 3: Twins Testing im Themis BFT Framework

- Der PBFT Algorithmus erklärt im Orginalpapier hier
- Die Twins Methodik für BFT hier erklärt:
  - https://malkhi.com/posts/2020/04/making-BFT-systems-robust/
  - Twins: BFT Systems made Robust
  - Repo Link (https://github.com/asonnino/twins-simulator)
- Kurze Einführung ins Themis Framework hier
  - Repo zum Framework: https://github.com/ibr-ds/themis