Non-Standard-Datenbanken

Von NoSQL zu NewSQL

Prof. Dr. Ralf Möller

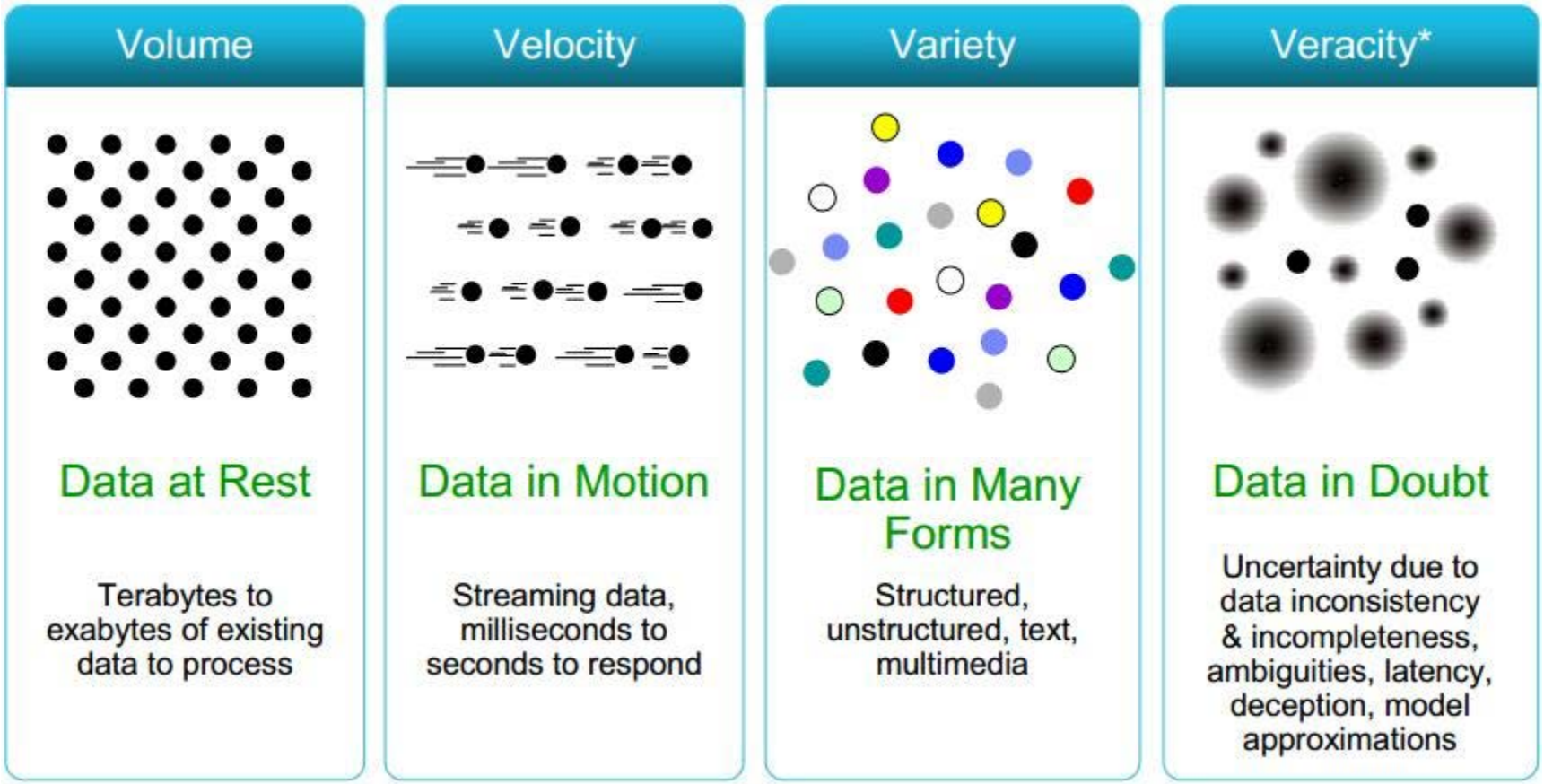
Universität zu Lübeck

Institut für Informationssysteme

Übersicht

- Semistrukturierte Datenbanken (JSON, XML) und Volltextsuche
- Information Retrieval
- Mehrdimensionale Indexstrukturen
- Cluster-Bildung
- Einbettungstechniken
- First-n-, Top-k-, und Skyline-Anfragen
- Probabilistische Datenbanken, Anfragebeantwortung, Top-k-Anfragen und Open-World-Annahme
- Probabilistische Modellierung, Bayes-Netze, Anfragebeantwortungsalgorithmen, Lernverfahren,
- Temporale Datenbanken und das relationale Modell, SQL:2011
- Probabilistische Temporale Datenbanken
- SQL: neue Entwicklungen (z.B. JSON-Strukturen und Arrays), Zeitreihen (z.B. TimeScaleDB)
- Stromdatenbanken, Prinzipien der Fenster-orientierten inkrementellen Verarbeitung
- Approximationstechniken für Stromdatenverarbeitung, Stream-Mining
- Probabilistische raum-zeitliche Datenbanken und Stromdatenverarbeitungsssysteme: Anfragen und Indexstrukturen, Raum-zeitliches Data Mining, Probabilistische Skylines
- Von NoSQL- zu NewSQL-Datenbanken, CAP-Theorem, Blockchain-Datenbanken

Big Data



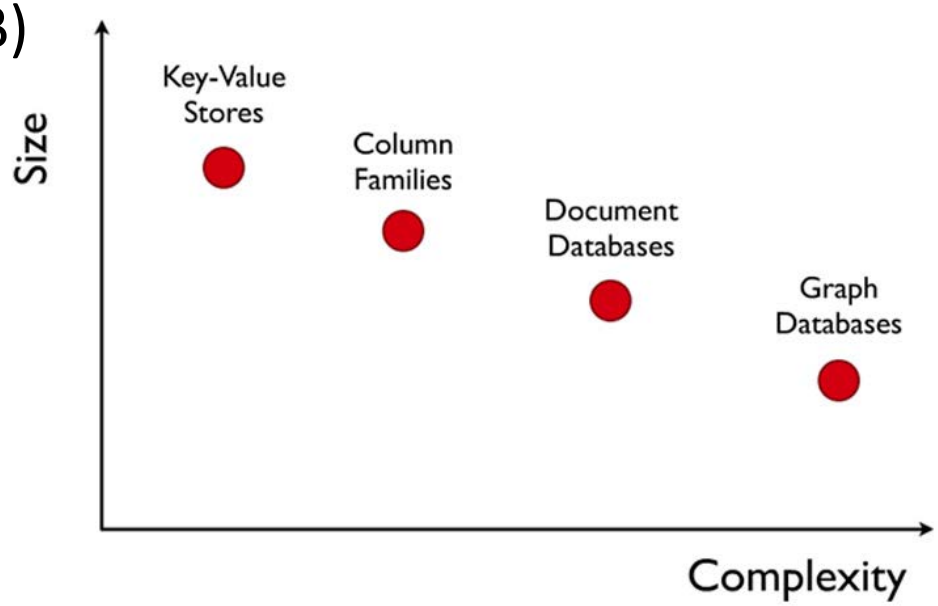
*Veracity = Aufrichtigkeit, Wahrhaftigkeit

NoSQL: Not Only SQL?

- Datenmengen werden groß (Big Data)
  - Horizontale Skalierung notwendig (→ Elastizität)
  - Virtualisierung (Cloud Computing) geht damit einher
- Heterogenität von Daten, Datenintegration
  - Schema-Freiheit (relationales Modell scheint zu starr)
  - Algorithmische Datenverarbeitung wird gewünscht
- Verteilung der Datenhaltung
  - CAP Theorem (Consistency, Availability, Partitioning tolerant: Eric Brewer)
    - Nur 2 aus 3 Kriterien erfüllbar
  - Abschwächung des Serialisierbarkeits-Konsistenzkriteriums: BASE
    - **B**asically **A**vailable
    - **S**oft-state (or scalable)
    - **E**ventually consistent

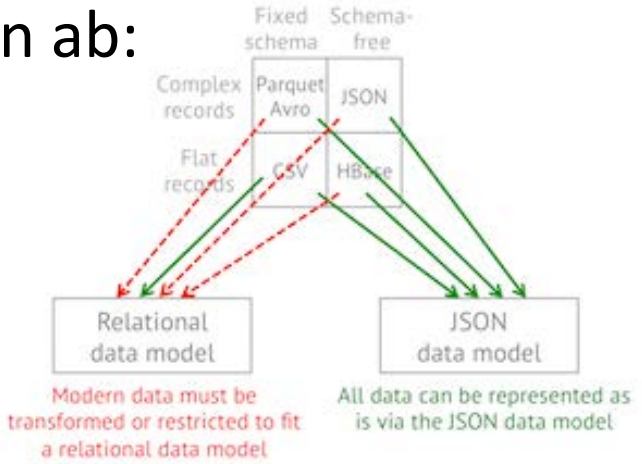
NoSQL-Datenbanktypen

- Key-Value Stores – Hashtabelle von Schlüsseln
  - Google Bigtable, Amazon Dynamo
- Column Stores – Block nur Daten aus einer Spalte
  - Supernormalisierung
- Document Stores (MongoDB)
  - XML-Strukturen
  - JSON-Strukturen
- Graphdatenbanken
  - RDF
    - SPARQL
  - Property Graphs
    - Cypher (Neo4J)
    - GQL als neuer ISO-Standard neben SQL (seit 2017 entwickelt)



NoSQL: Zusammenfassung

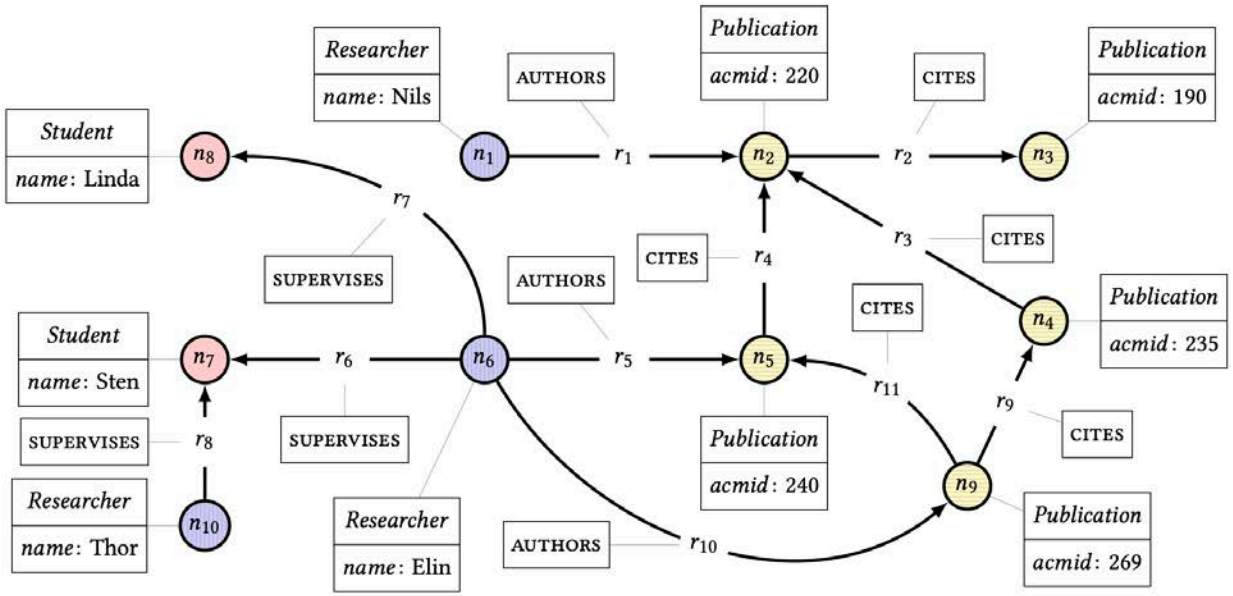
- Nutzer von NoSQL-Datenbanken lehnen ab:
  - Aufwand von ACID-Transaktionen
  - “Komplexität” von SQL
  - Aufwand des Designs von Schemata
  - Deklarative Anfrageformulierung
  - Ein-Prozessor-Technologie
- Programmierer wird verantwortlich für
  - Prozedurale Formulierung von Zugriffsalgorithmen
  - und Navigation zu den benötigten Daten




```
SqlString sql = select(ARTICLE.OID)
                .from(ARTICLE, ARTICLE_COLOR)
                .where(ARTICLE.OID.eq(ARTICLE_COLOR.ARTICLE_OID)
                .and(ARTICLE.ARTICLE_NO.is_not(NULL)));
```

Konkrete Ausprägung variiert

Cypher




```
MATCH (svc:Service)<-[:DEPENDS_ON*]-(dep:Service)
RETURN svc, count(DISTINCT dep) AS dependents
ORDER BY dependents DESC
LIMIT 1
```

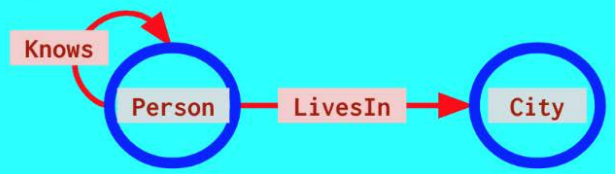
Nadime Francis, Alastair Green, Paolo Guagliardo, Leonid Libkin, Tobias Lindaaker, Victor Marsault, Stefan Plantikow, Mats Rydberg, Petra Selmer, and Andrés Taylor. 2018. Cypher: An Evolving Query Language for Property Graphs. In Proceedings of the **2018** International Conference on Management of Data (SIGMOD '18).

```
MATCH (accHolder:AccountHolder)-[:HAS]->(pInfo)
WHERE pInfo:SSN OR pInfo:PhoneNumber
                OR pInfo:Address
WITH pInfo,
    collect(accHolder.uniqueId) AS accountHolders,
    count(*) AS fraudRingCount
WHERE fraudRing > 1
RETURN accountHolders,
    labels(pInfo) AS personalInformation,
    fraudRingCount
```

GQL

Inspired by
- Cypher
- SQL/PGQ
- SPARQL

```
CREATE GRAPH SocialNetwork {
    (Person {name STRING, dob DATE}),
    (City {name STRING}),
    (Person)-[LivesIn]->(City),
    (Person)-[Knows]->(Person)
}
```



WG3:JCJ-015
DM32.2-2019-00999

ISO/IEC JTC 1/SC 32

Date: 2019-05-09

IWD 39075:202y(E)

ISO/IEC JTC 1/SC 32/WG 3

The United States of America (ANSI)

Information technology — Database languages — GQL

Technologies de l'information — Langages de base de données — GQL

SQL/PGQ querying – Example

Get the creationDate and content of the messages created by one person ("email1") and commented on by another person ("email2").

```
SELECT GT.creationDate, GT.content
FROM myGraph GRAPH_TABLE (
  MATCH
   (Creator IS Person WHERE Creator.email = :email1)
      -[ IS Created ]->
   (M IS Message)
      <-[ IS Commented ]-
   (Commenter IS Person WHERE Commenter.email = :email2)
      WHERE ALL_DIFFERENT (Creator, Commenter)
  COLUMNS (
      M.creationDate,
      M.content )
 ) AS GT
```

Mehr zu Graphdaten-banken in Sven Groppes Vorlesung

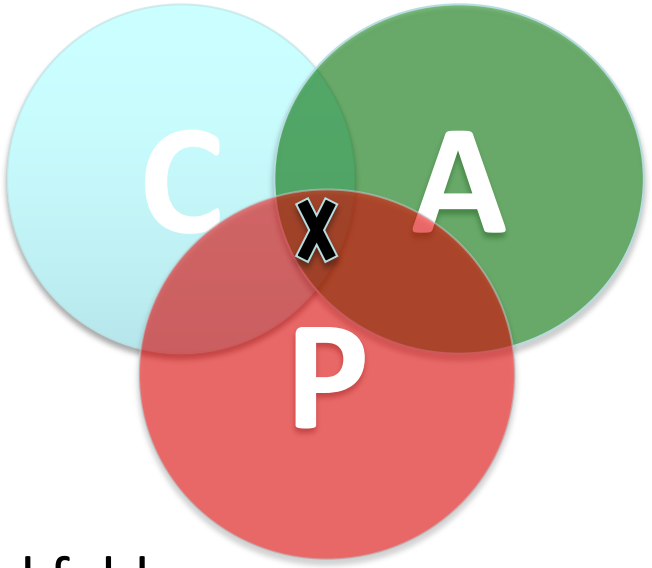
Standardisierung von Graph Datenbanken

SQL/GQL Implementierung werden erscheinen

- Indexstrukturen
- Approximative Anfragebeantwortung
- Ströme von Graphstrukturen

CAP Theorem

- **C**onsistency:
  - Alle Knoten sehen die gleichen Daten zur gleichen Zeit
- **A**vailability (Verfügbarkeit):
  - Fehler von Knoten beeinträchtigen nicht die Funktionsfähigkeit der Überlebenden
- **P**artitionierungstoleranz:
  - System arbeitet weiter, auch bei Partitionierung durch Netzwerkfehler

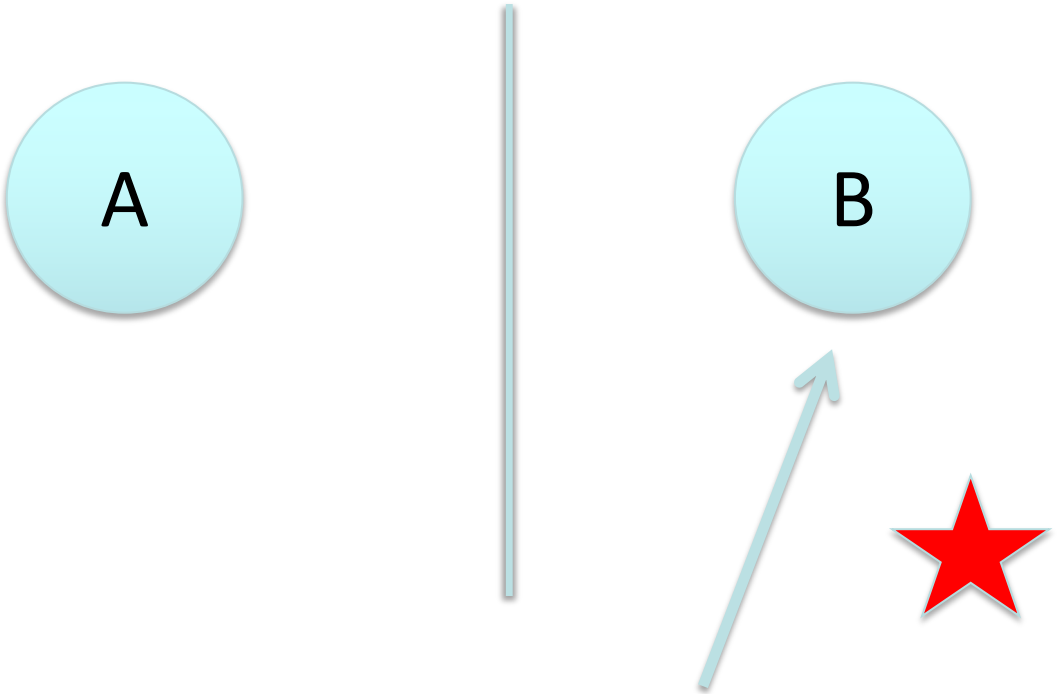


- Theorem: Ein verteiltes System kann nur jeweils zwei Kriterien erfüllen, nicht aber alle drei

Seth Gilbert and Nancy Lynch. "Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services." ACM SIGACT News 33.2 (**2002**): 51-59

CAP Theorem: Argumente

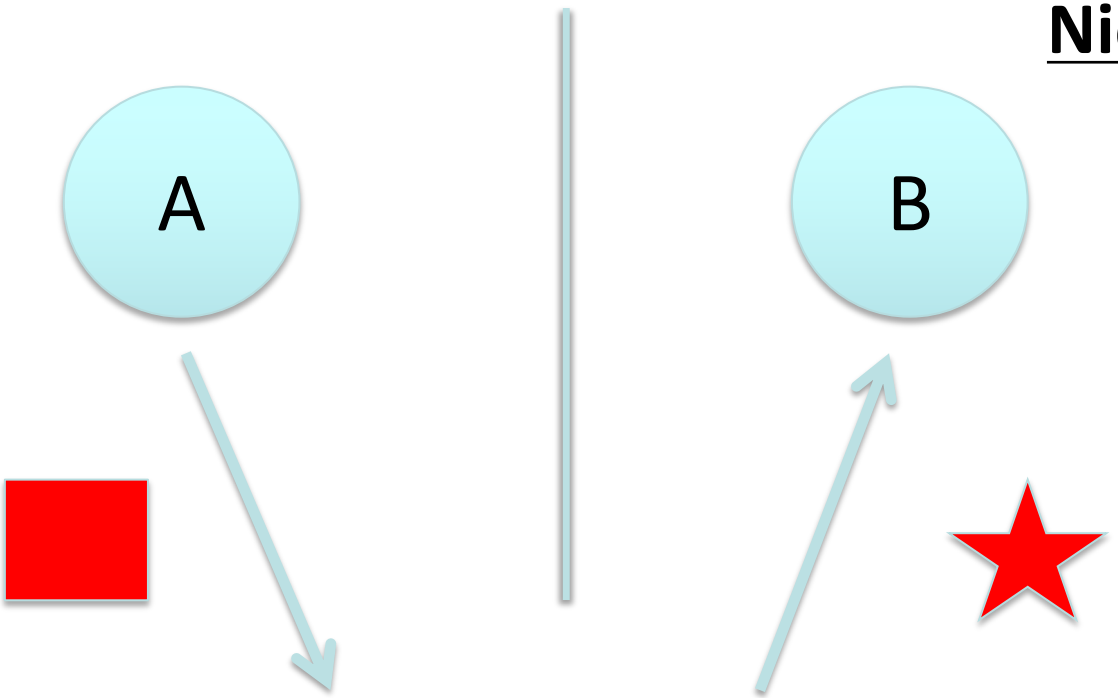
- Nehmen wir an, es gibt zwei Knoten:



CAP Theorem: Argumente

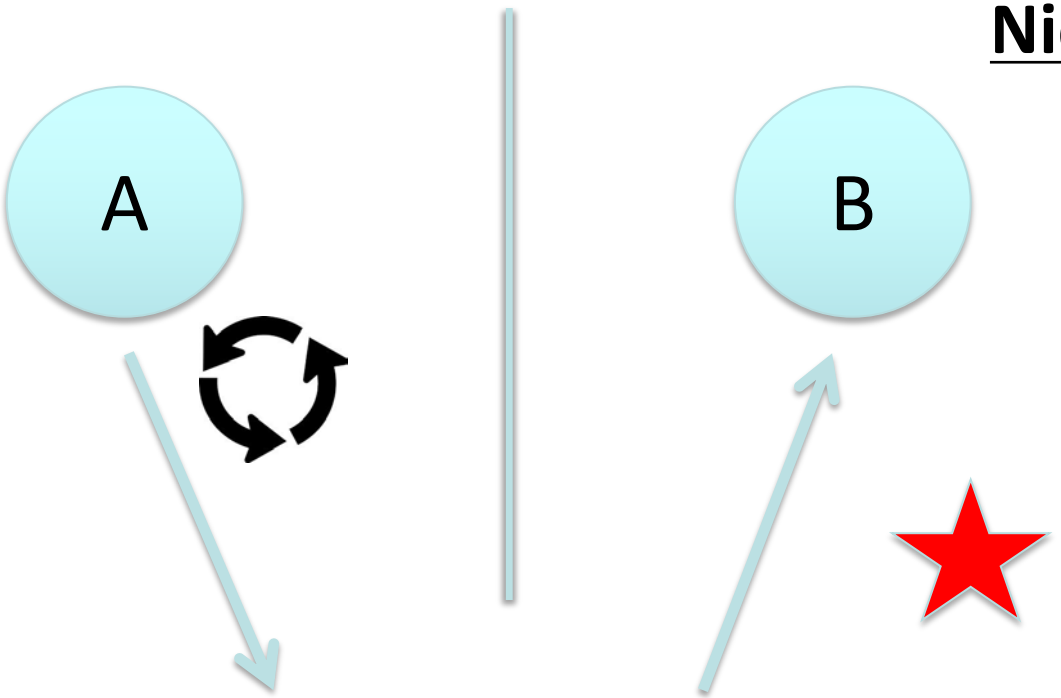
- Nehmen wir an, es gibt zwei Knoten:

Nicht konsistent!



CAP Theorem: Argumente

- Nehmen wir an, es gibt zwei Knoten:

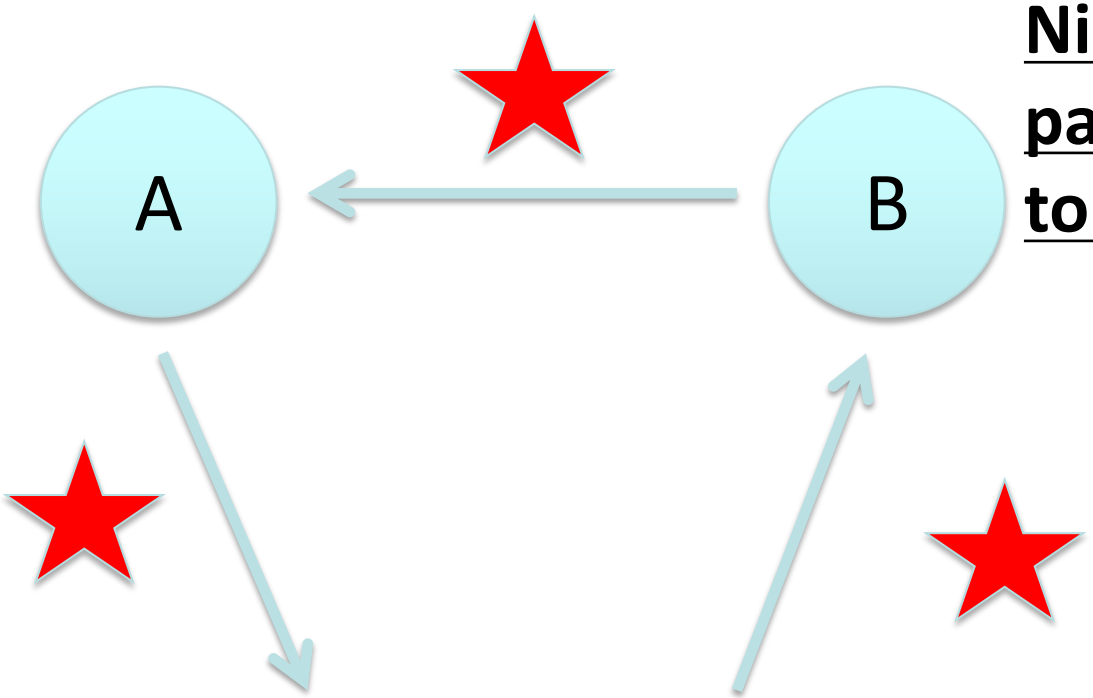


Nicht verfügbar!

Warten auf Aktualisierung

CAP Theorem: Argumente

- Nehmen wir an, es gibt zwei Knoten:



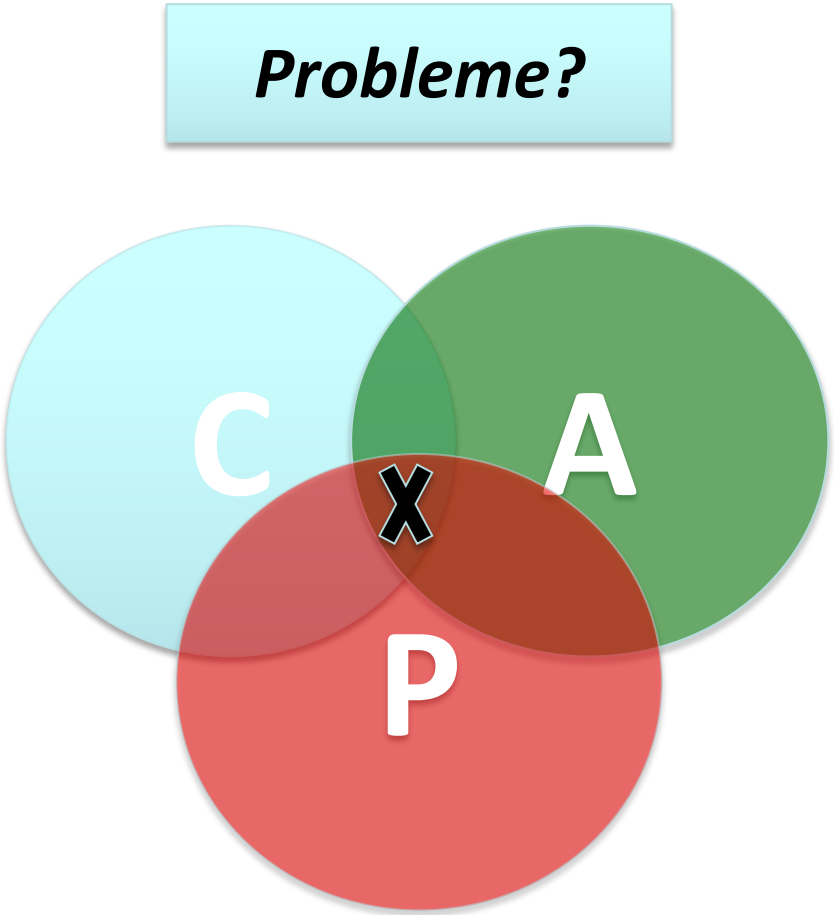
Nicht partitionierungs-tolerant!

A erhält Aktualisierung von B

Partitionierung:
System zerfällt in (mindestens) zwei unverbundenen Teile

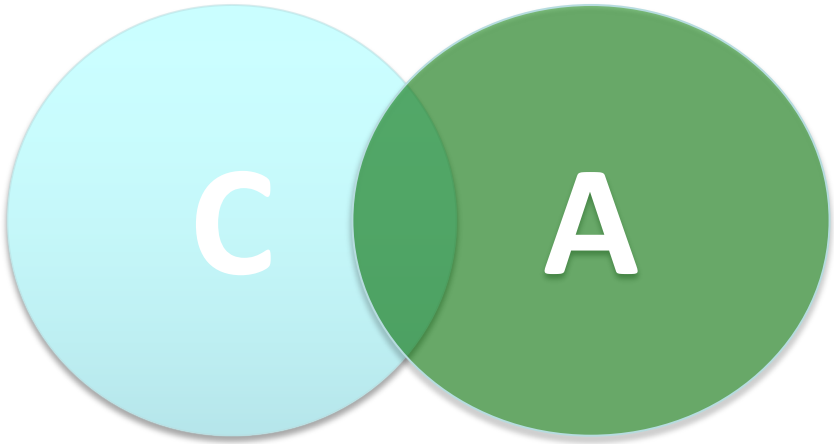
Neubetrachtung CAP Theorem

- Von den folgenden Garantien angeboten von verteilten Systemen:
  - Consistency
  - Availability
  - Partition tolerance
- Wähle zwei
- Drei Möglichkeiten :
  - CP
  - AP
  - CA (relationales DMBS)



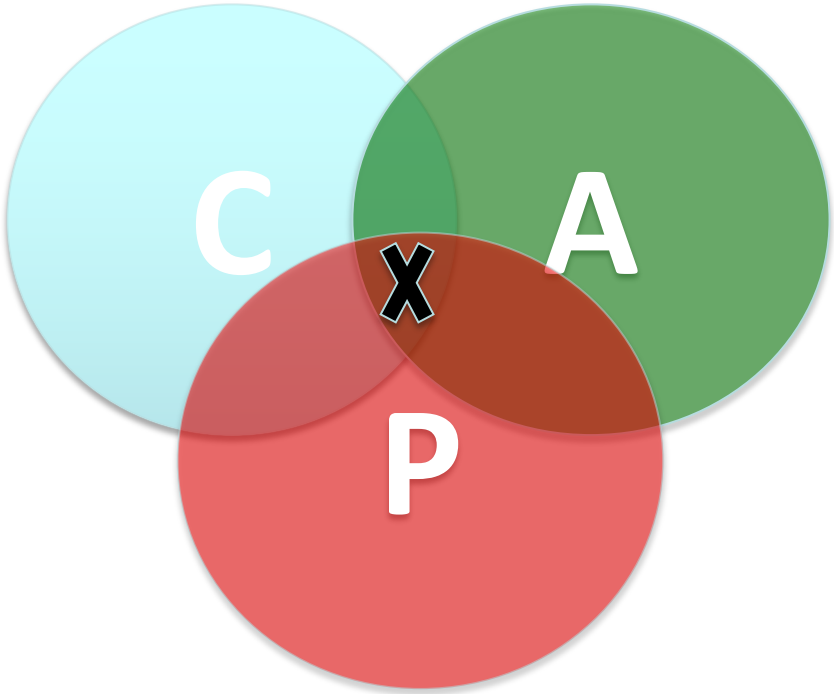
Häufiges Missverständnis: 2 von 3

- Was ist mit CA?
- Kann ein verteiltes System (mit unzuverlässigem Netzwerk) wirklich nicht partitionierungstolerant sein?



Konsistenz oder Verfügbarkeit

- Konsistenz und Verfügbarkeit sind keine binären Entscheidungen
- AP-Systeme schwächen Konsistenz zugunsten von Verfügbarkeit (sind aber nicht notwendigerweise inkonsistent)
- CP-Systeme opfern Verfügbarkeit zugunsten der Konsistenz (sind aber durchaus meist verfügbar)
- Quintessenz: AP- und CP-Systeme können Konsistenz, Verfügbarkeit und Partitionierungstoleranz graduell zur Verfügung stellen



Arten der Konsistenz

- Starke Konsistenz
  - Nach einer Datenaktualisierung muss jeder nachfolgende Zugriff (egal auf welcher Kopie) den neuen Wert liefern
- Schwache Konsistenz
  - Es wird nicht garantiert, dass nachfolgende Zugriffe auf Kopien den aktualisierten Wert liefern
- Letztendliche Konsistenz (Eventual consistency)
  - Spezielle Form der schwachen Konsistenz
  - Es wir garantiert, dass schließlich alle Prozesse den letztmalig aktualisierten Wert sehen, wenn keine neuen Änderungen erfolgen (verzögerte Propagierung von Änderungen in die Kopien)

Abadi, Daniel J. "Consistency tradeoffs in modern distributed database system design." Computer-IEEE Computer Magazine 45.2 (**2012**): 37.

AP, CP greift zu kurz: CAP/ELC

- **AP/EL-System:** Gib Konsistenz C auf zugunsten der Verfügbarkeit A und kleinerer Latenz L
  - Dynamo, Cassandra, Riak
- **CP/EC-System:** Verweigere, die Konsistenz C aufzugeben, und zahle die Kosten von Verfügbarkeit und Latenz
  - BigTable, Hbase, VoltDB/H-Store
- **AP/EC-System:** Gib Konsistenz auf, wenn Partitionierung erfolgt, und erhalte Konsistenz im normalen Betrieb
  - MongoDB
- **CP/EL-System:** Erhalte Konsistenz, wenn Partitionierung erfolgt, gebe Konsistenz für kleine Latenz im normalen Betrieb auf
  - Yahoo! PNUTS

E=ELSE
L=Latency
C=Consistency

CAP Theorem – Bad News: Complexity

- Ignoring Complexity: Anything can happen. Be afraid.
- Avoiding complexity: No way. Trivial examples occur with read-write and write-write conflicts.
- Encapsulating complexity: Depends who you ask. Coordinated memory systems (e.g. transactional databases, various cache coherency protocols) can do this for you, but at the expense of availability/latency.

Beyond Imperative Computing

- Koordination ist das letzte wirklich teure Gut
  - Anwendungsprogrammierung baut auf mutierbarem Zustand und geordneten Strukturen
  - Nicht sehr gut für Cloud-Architekturen geeignet
  - Sperren und Kommunikation verlangsamen Berechnungen
- Wie kann man Koordination vermeiden, ohne Inkonsistenzen zu erzeugen
  - Geht nicht mit klassischen Read/Write-Betrachtungen (Transaktionen)
  - Vermeidung auf Applikationsebene mit neuen Programmiertechniken
    - Z.B. Bloom http://bloom-lang.net

Bloom Programming Language

- Can my programming environment (language, tools) help me write simple, efficient distributed code, and avoid complicated, expensive distributed protocols?

```
state do
  table :link, [:from, :to]
  scratch :path, [:from, :to]
end

bootstrap {link <+ [['a', 'b'], ['a', 'c'], ['b', 'd']]}

bloom do
  path <= link
  path <= (path * link).pairs(:to=>:from) {|p,l| [p.from, l.to]}
end
```

- Datalog:
  - monotonic (LFP, PTIME complete)
  - w/ stratified negation (still PTIME)
    - Sealing, LFP per stratum bottom up
- Sealing in distributed systems: Coodination

Theory of Distributed Systems

- Notion of Causality,
  - which enables programmers and programs to avoid doing unusual things like executing instructions in orders that could not have been specified by the program that generated them.
  - Grandfather paradox
- Causality is established by distributed clock protocols.
  - These protocols are often used to enforce causal orderings – i.e. to make machines wait for messages. And waiting for messages, as we know, is bad.
- The CRON Principle: To Hell with Distributed Clocks

CALM

- Conjecture: Consistency and Logical Monotonicity
- LM => C. Monotonic logic produces consistent results regardless of the order of messaging. Proof seems pretty easy: construction via pipelined semi-naive
- C => LM? All consistent programs are monotonic? Well, not on their face anyway.
- LM + Coord => C. If we introduce coordination at the stratification boundaries of a distributed Datalog program, we get an order-insensitive program.
- And we can check LM syntactically! (Well, we can check it conservatively that way.)

Alvaro, P., Conway, N., Hellerstein, J.M., & Marczak, W.R.. Consistency Analysis in Bloom: a CALM and Collected Approach. CIDR. **2011**.

CRON

- CRON Hypothesis:
  - Causality Required Only for Non-Monotonic logic.

- Pretty confident conjecture:
  - Monotonic programs don't require causal orders. (Proved by Ameloot and Van den Bussche.)

- Conjecture: Non-monotonic programs require casual ordering. (Or equivalently, "every program that tolerates non-causality is equivalent to a positive program".
  - Disproved by Ameloot and Van den Bussche.)

Bloom, CALM, CRON

- Ignoring Complexity: Suppose I ignore the problems of concurrency and failure. What's the worst that can happen?
  - Answer: no sweat if your program is monotonic. We can also auto-inject coordination into a user program that is non-monotonic. Or we can identify the potentially non-monotonic "points of order", and you can augment your program to carry that "taint" if you like.
- Avoiding complexity: Can I rewrite my programs in a way that avoids the need for coordination? Always?
  - Answer: It appears that we should not need any of those techniques, in theory, for any polynomial-time (i.e. reasonable) tasks. Open question whether that's practical.
- Encapsulating complexity: Can I solve my distributed systems problems once in a reusable library and hide the complexity?
  - *Answer: Well, you can guarantee that a module guards all its non-monotonicity with some coordination protocol. You can similarly build a coordination service of some kind and defer to it, if you call out to it intelligently.

Hellerstein, Joseph & Alvaro, Peter. Keeping CALM: When Distributed Consistency is Easy. **2019**.

Danksagung

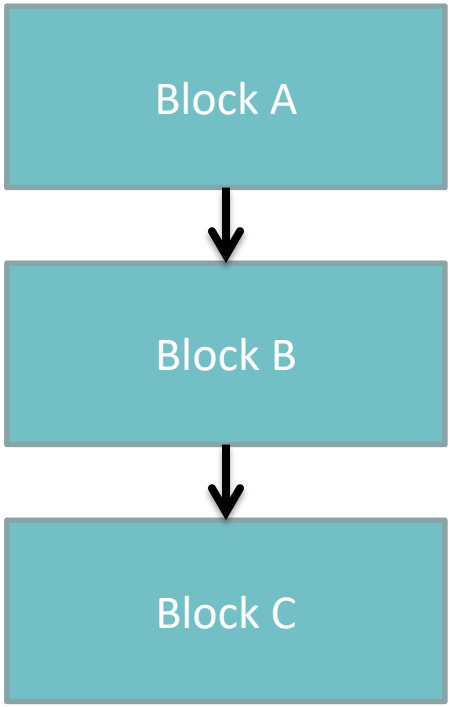
- Blockchain-Präsentationen in Zusammenarbeit mit Dennis Heinrich (IFIS)

Was ist eine Blockchain?

Wie der Name schon sagt, eine **Kette** von **Blöcken**

Was enthalten die **Blöcke**?

- Prinzipiell alle möglichen Daten
  - Währungen
  - Verträge
  - Grundbücher
  - Wikipedia Einträge
  - Stammbäume



Was ist mit der **Kette**?

- Dokumentiert eine Erweiterung der Daten

Ziel: **Konsistente** und **nachvollziehbare** Speicherung der Erweiterung von Daten!

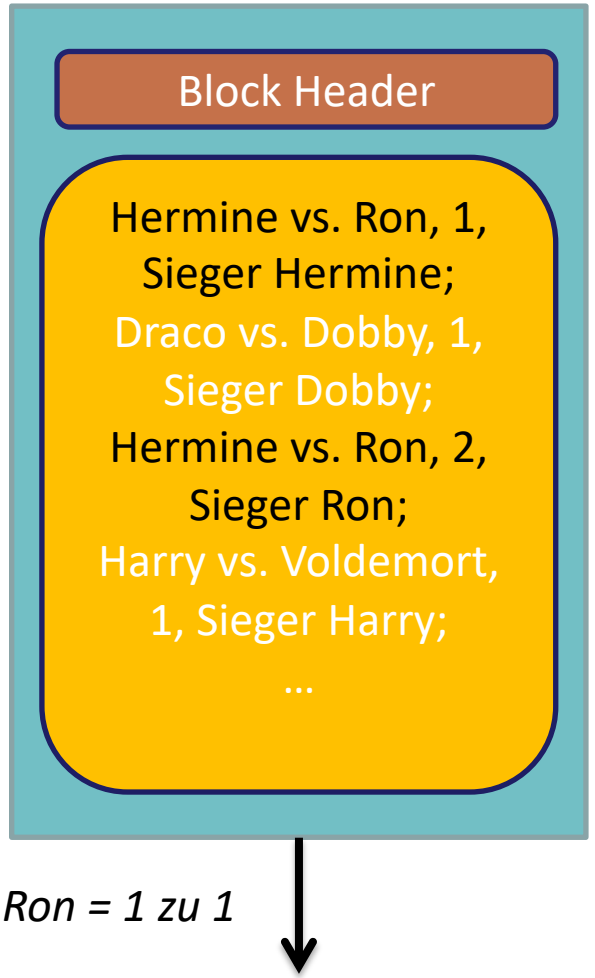
Beispiel Zaubererduell

Was enthält unserer **Block**?

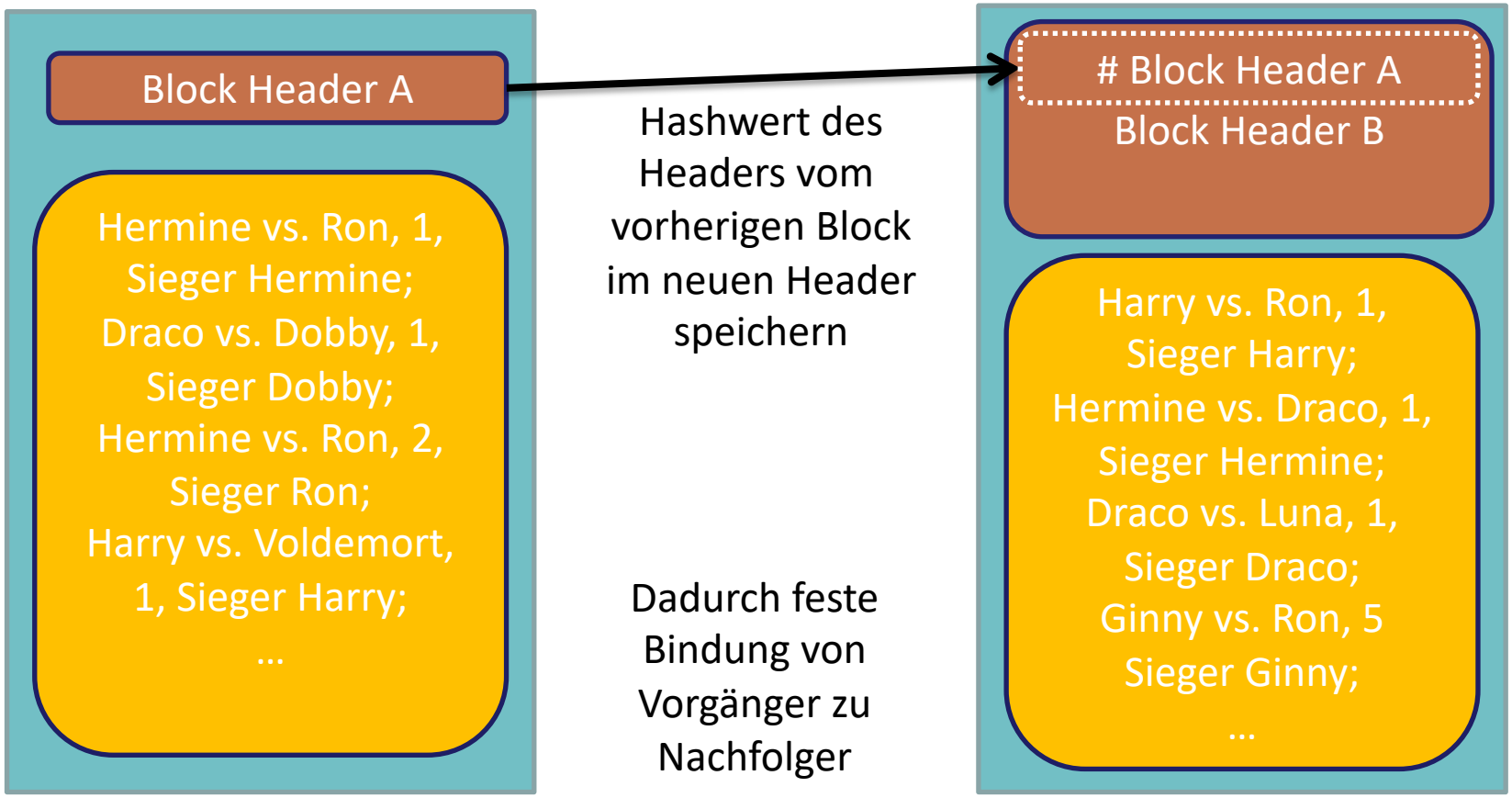
- Duelle zwischen 2 Zaubernden
- Sieger des Duells
- (Duellant A, Duellant B, DuellNr. zwischen Beteiligten, Sieger)

Was können wir damit **nachweisen**?

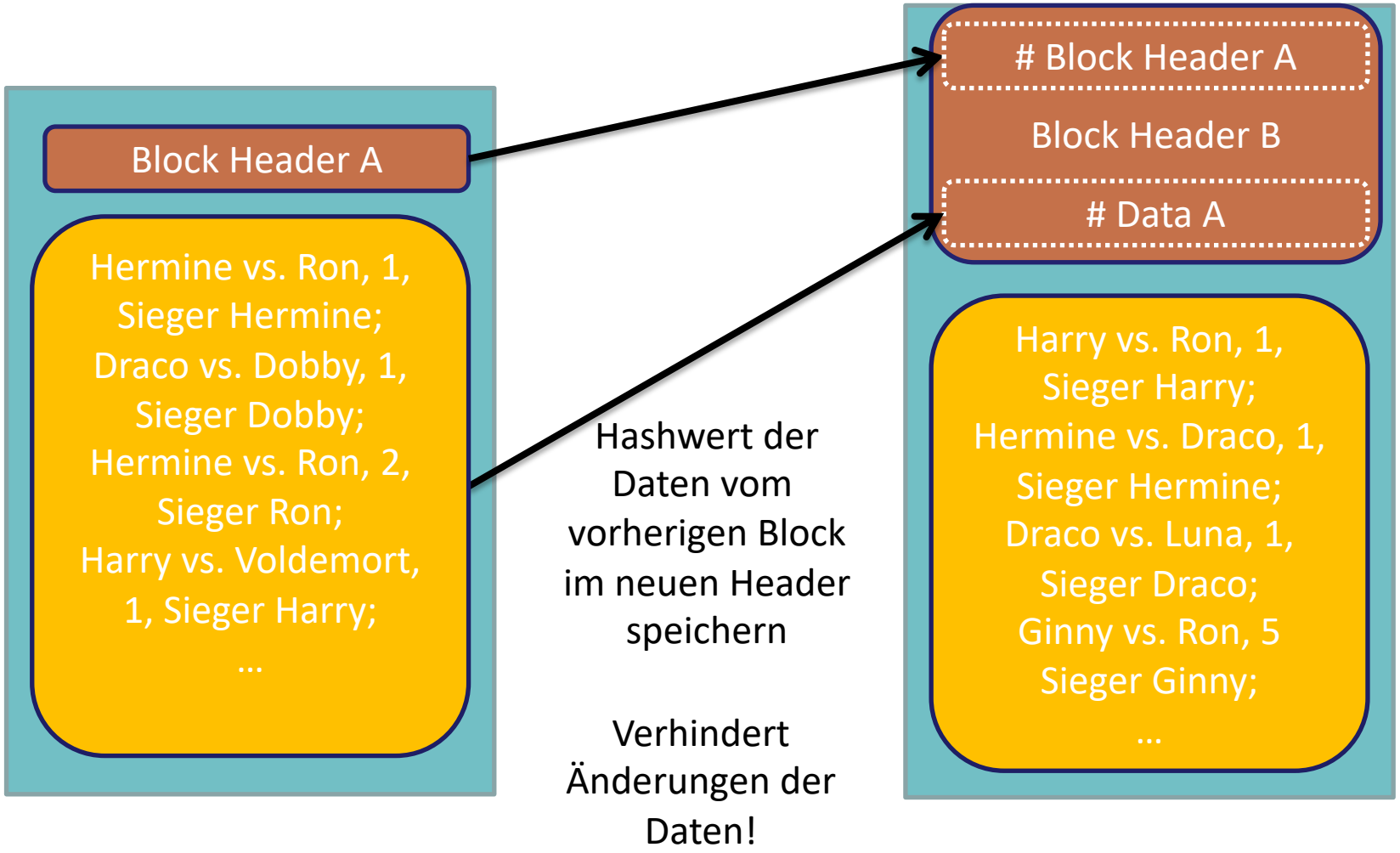
- Wer gegen Wen
- Häufigkeit Sieger/Verlierer über gesamte Kette



Wie sind die Blöcke verbunden?



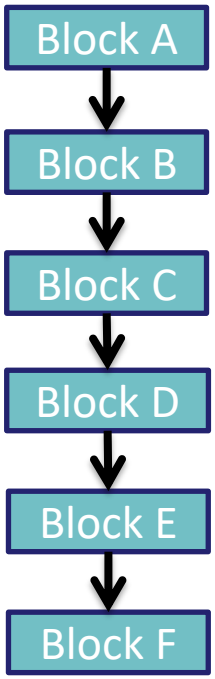
Wie sind die Blöcke verbunden?



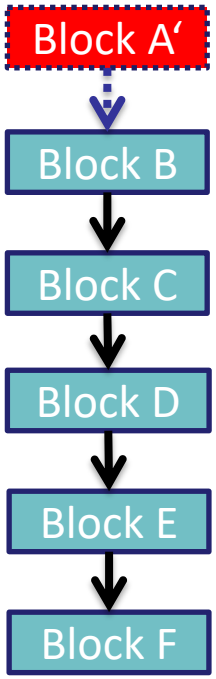
Kann Draco schummeln?

Problem: Verändere die Kette so, dass Draco in der Vergangenheit gewonnen hat.

Aktuelle Kette:



Dracos Kette:



Hashwert anders wegen veränderter Daten!
Block B erkennt seinen Vorgänger nicht an!

Schummeln am **Anfang** oder in der **Mitte** der Kette nicht möglich.

Was ist mit dem **Ende**?

Wer entscheidet über einen neuen Block?

Prinzipell: **JEDER** Teilnehmer an der Blockchain

Heißt in unserem Fall jeder Zaubernde, **ABER** es muss **Konsens** herrschen!

Wir brauchen somit ein dezentrales **Konsensverfahren**!
Häufig wird die sog. **Proof-of-Work-**Methode verwendet:

- Aufwendiger Arbeitsnachweis des Blockerstellers
- Einfache Prüfung für jeden anderen Teilnehmer

BEISPIEL: Ein neuer Block darf nur erstellt werden, wenn eine Abbildung des Blockinhalts auf einen Zauberspruch genannt werden kann, so dass z.B. der:

- Zauberspruch mit A anfängt,
- aus 2 Worten besteht und
- genau 2 Mal ‚v' enthält

Beispiel: **Avada Kedavra (man muss erst einmal viele Zauberspruchbücher durchsehen, um so eine Abbildung zu finden)**
Einfach zu prüfen, aber Abbildung aufwendig zu finden.

Anforderungen wechseln für jeden neuen Block, da neuer Inhalt gegeben

Erstellen eines neuen Blocks

1. **Verteile** alle neuen **Daten (bsp. [Harry, Draco, 3, Harry])** an alle bekannten Zauberer
2. Zauberer sammeln **Daten** in Blöcke und verteilen neue **Daten** weiter
3. Zauberer arbeiten am **Proof of Work**, um ihren Block zu veröffentlichen
4. Falls **Proof of Work** erfüllt, **verteile Block** an alle anderen bekannten Zauberer
5. Andere Zauberer **akzeptieren** den **Block** oder stellen **ungültige*** Daten fest
6. **Akzeptanz** wird durch Erstellung eines **Nachfolgeblocks** an den neuen Block ausgedrückt

** Was als ungültig angesehen wird, hängt auch von den Daten ab. In unserem Beispiel :*

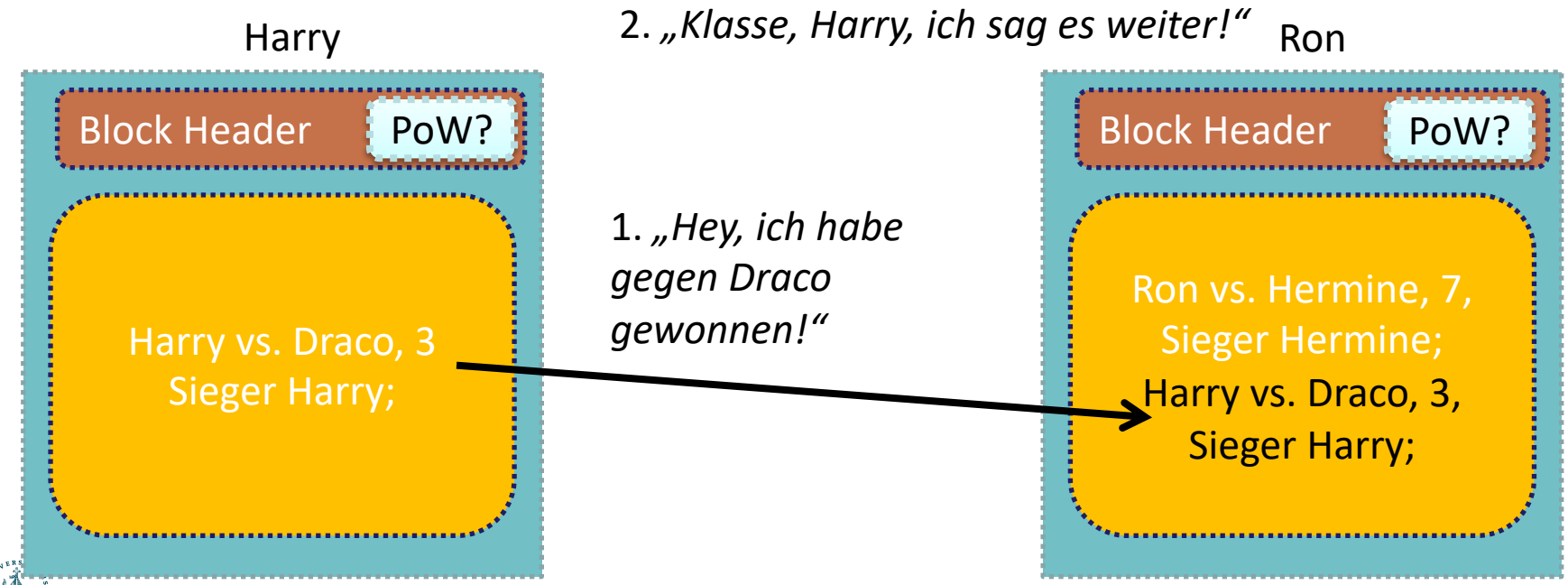
- *Duell gegen sich selbst verboten*
- *Duell mit 2 Siegern/Verlierern (keine doppelte Version eines Duells)*
- *Lücke/Reihenfolge falsch (Duell 3 vor 2, Wo ist Duell 1?)*

Schritte aus https://bitcoin.org/bitcoin.pdf

Erstellen eines neuen Blocks

1. **Verteile** alle neuen **Daten (bsp. [Harry, Draco, 3, Harry])** an alle bekannten Zauberer
2. Zauberer sammeln **Daten** in Blöcke und verteilen neue **Daten** weiter

3. Zauberer arbeiten am **Proof of Work**, um ihren Block zu veröffentlichen
4. Falls **Proof of Work** erfüllt, **verteile Block** an alle anderen bekannten Zauberer
5. Andere Zauberer **akzeptieren** den **Block** oder stellen ungültige Daten fest
6. **Akzeptanz** wird durch Erstellung eines **Nachfolgeblocks** an den neuen Block ausgedrückt

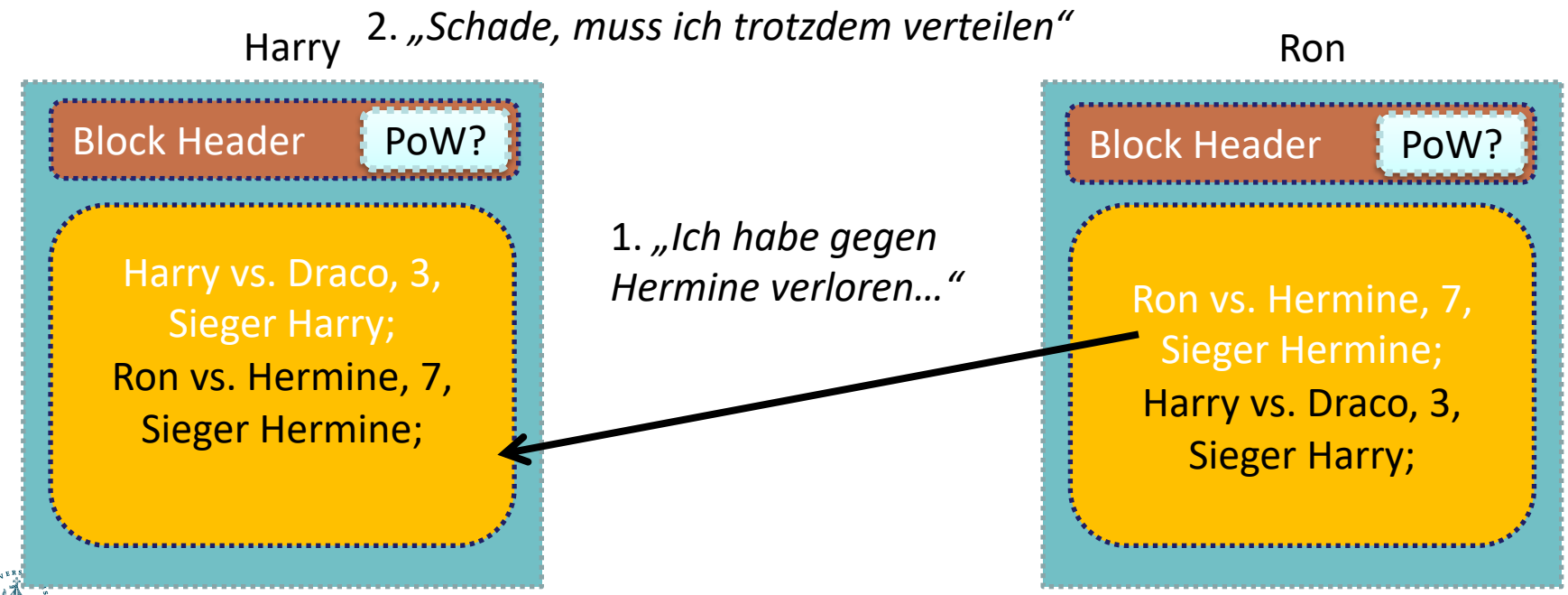


Schritte aus https://bitcoin.org/bitcoin.pdf

Erstellen eines neuen Blocks

1. **Verteile** alle neuen **Daten (bsp. [Harry, Draco, 3, Harry])** an alle bekannten Zauberer
2. Zauberer sammeln **Daten** in Blöcke und verteilen neue **Daten** weiter

3. Zauberer arbeiten am **Proof of Work**, um ihren Block zu veröffentlichen
4. Falls **Proof of Work** erfüllt, **verteile Block** an alle anderen bekannten Zauberer
5. Andere Zauberer **akzeptieren** den **Block** oder stellen invalide Daten fest
6. **Akzeptanz** wird durch Erstellung eines **Nachfolgeblocks** an den neuen Block ausgedrückt



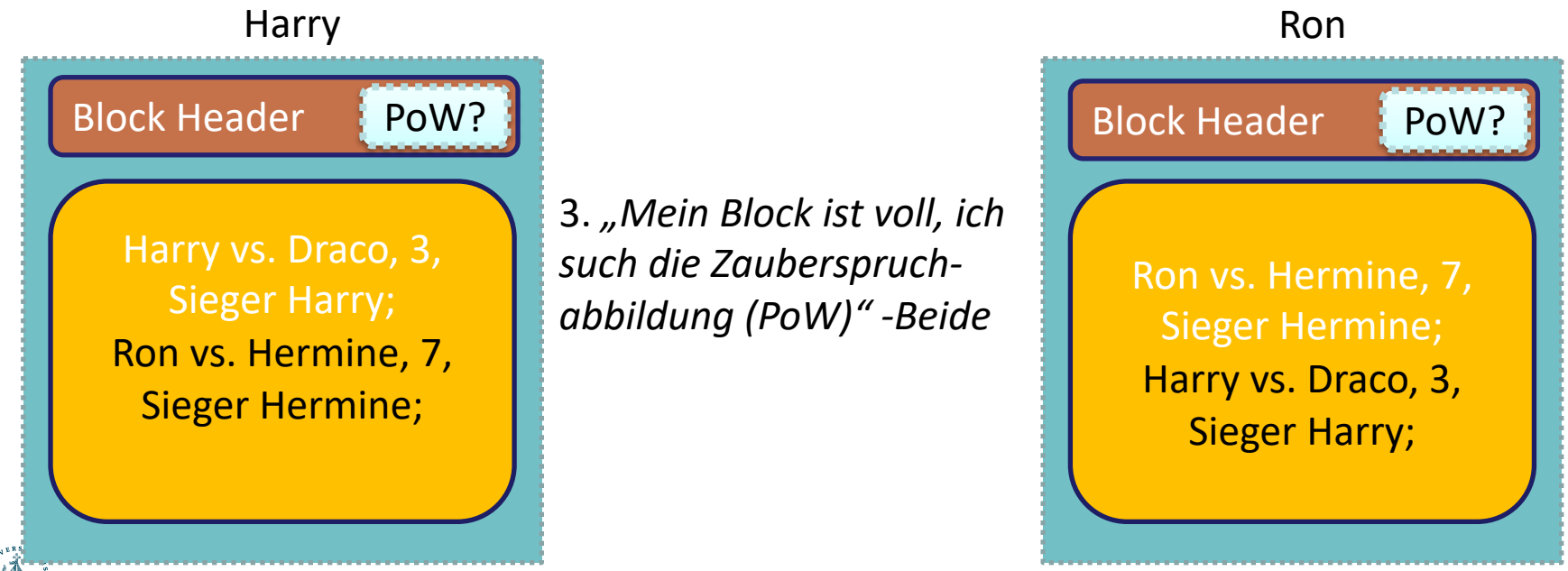
Schritte aus https://bitcoin.org/bitcoin.pdf

Erstellen eines neuen Blocks

1. **Verteile** alle neuen **Daten (bsp. [Harry, Draco, 3, Harry])** an alle bekannten Zauberer
2. Zauberer sammeln **Daten** in Blöcke und verteilen neue **Daten** weiter

3. Zauberer arbeiten am **Proof of Work**, um ihren Block zu veröffentlichen

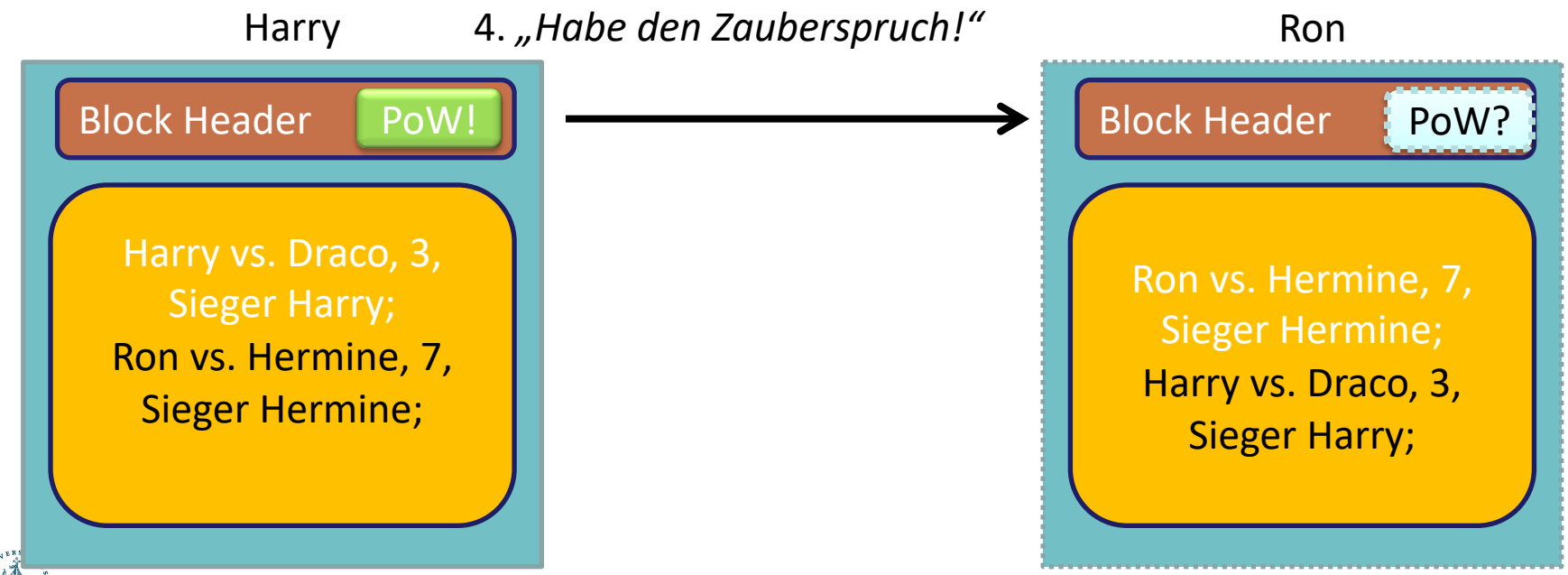
4. Falls **Proof of Work** erfüllt, **verteile Block** an alle anderen bekannten Zauberer
5. Andere Zauberer **akzeptieren** den **Block** oder stellen invalide Daten fest
6. **Akzeptanz** wird durch Erstellung eines **Nachfolgeblocks** an den neuen Block ausgedrückt



Schritte aus https://bitcoin.org/bitcoin.pdf

Erstellen eines neuen Blocks

1. **Verteile** alle neuen **Daten (bsp. [Harry, Draco, 3, Harry])** an alle bekannten Zauberer
2. Zauberer sammeln **Daten** in Blöcke und verteilen neue **Daten** weiter
3. Zauberer arbeiten am **Proof of Work**, um ihren Block zu veröffentlichen
4. Falls **Proof of Work** erfüllt, **verteile Block** an alle anderen bekannten Zauberer
5. Andere Zauberer **akzeptieren** den **Block** oder stellen invalide Daten fest
6. **Akzeptanz** wird durch Erstellung eines **Nachfolgeblocks** an den neuen Block ausgedrückt

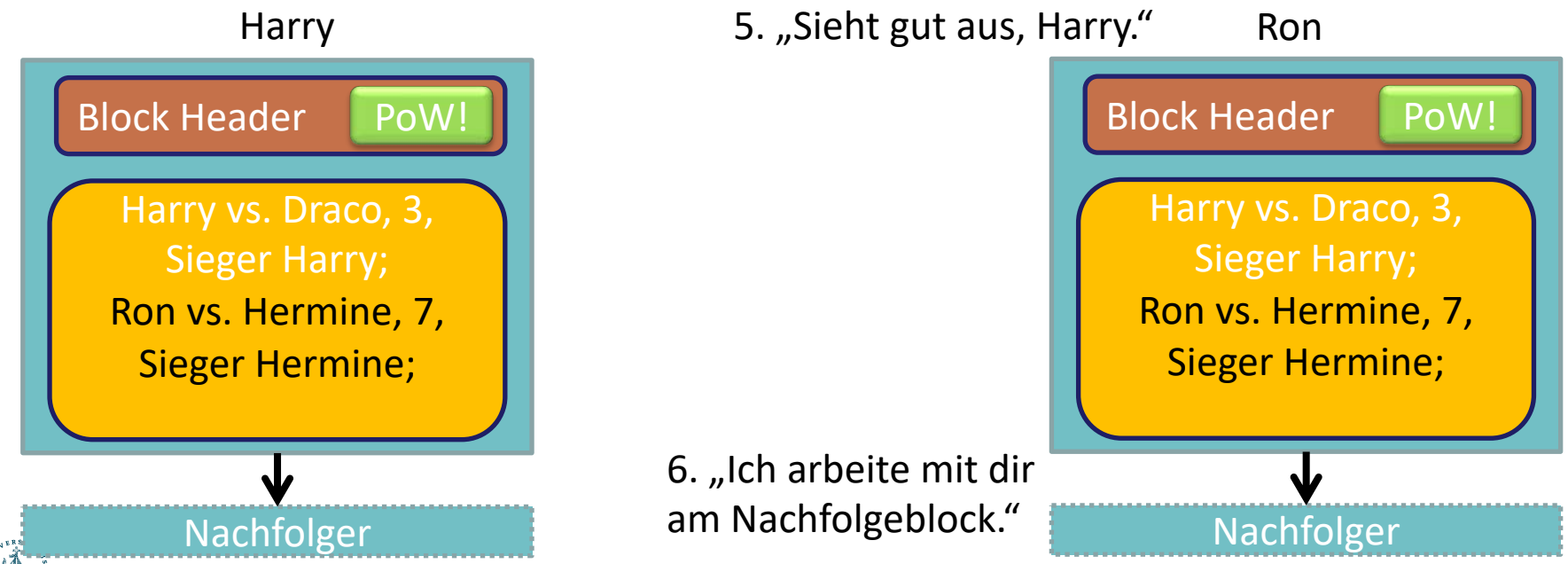


Schritte aus https://bitcoin.org/bitcoin.pdf

Erstellen eines neuen Blocks

1. **Verteile** alle neuen **Daten (bsp. [Harry, Draco, 3, Harry])** an alle bekannten Zauberer
2. Zauberer sammeln **Daten** in Blöcke und verteilen neue **Daten** weiter
3. Zauberer arbeiten am **Proof of Work**, um ihren Block zu veröffentlichen
4. Falls **Proof of Work** erfüllt, **verteile Block** an alle anderen bekannten Zauberer

5. Andere Zauberer **akzeptieren** den **Block** oder stellen invalide Daten fest
6. **Akzeptanz** wird durch Erstellung eines **Nachfolgeblocks** an den neuen Block ausgedrückt



Schritte aus https://bitcoin.org/bitcoin.pdf

Wie sieht ein realer Proof of Work aus?

Es wird ein Ziel festgelegt, das vom Hashwert erfüllt werden muss,
beispielweise:

Hashfunktion(Header, Daten, RandomNumber) < Ziel

Da **Header** und **Daten** vorgegeben sind, kann nur die **RandomNumber** verändert werden.

Nach längerem ‚**Raten**' kann eine Lösung gefunden werden.

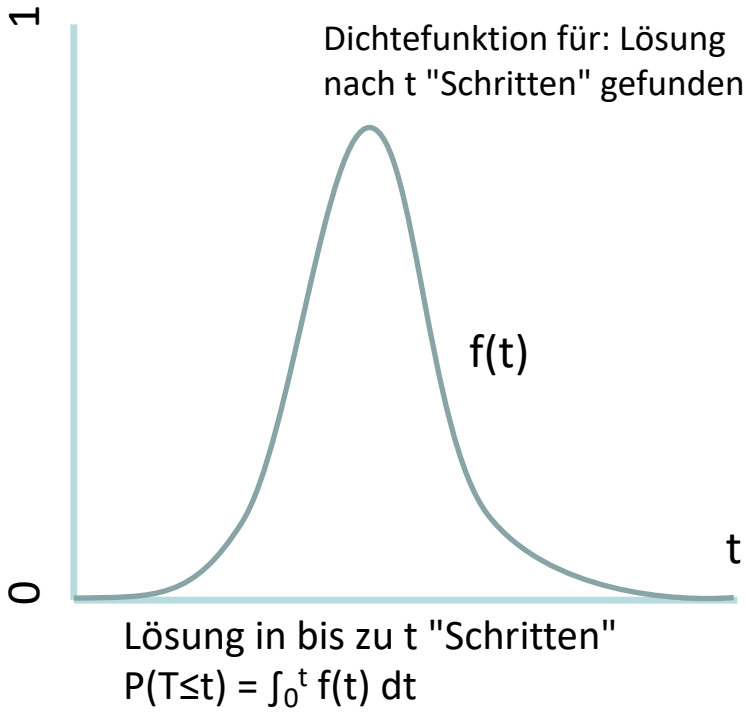
*Hashfunktion(4EF..., 3GH..., **1**) < 100...*

*Hashfunktion(4EF..., 3GH..., **2**) < 100...*

*Hashfunktion(4EF..., 3GH..., **3**) < 100...*

...

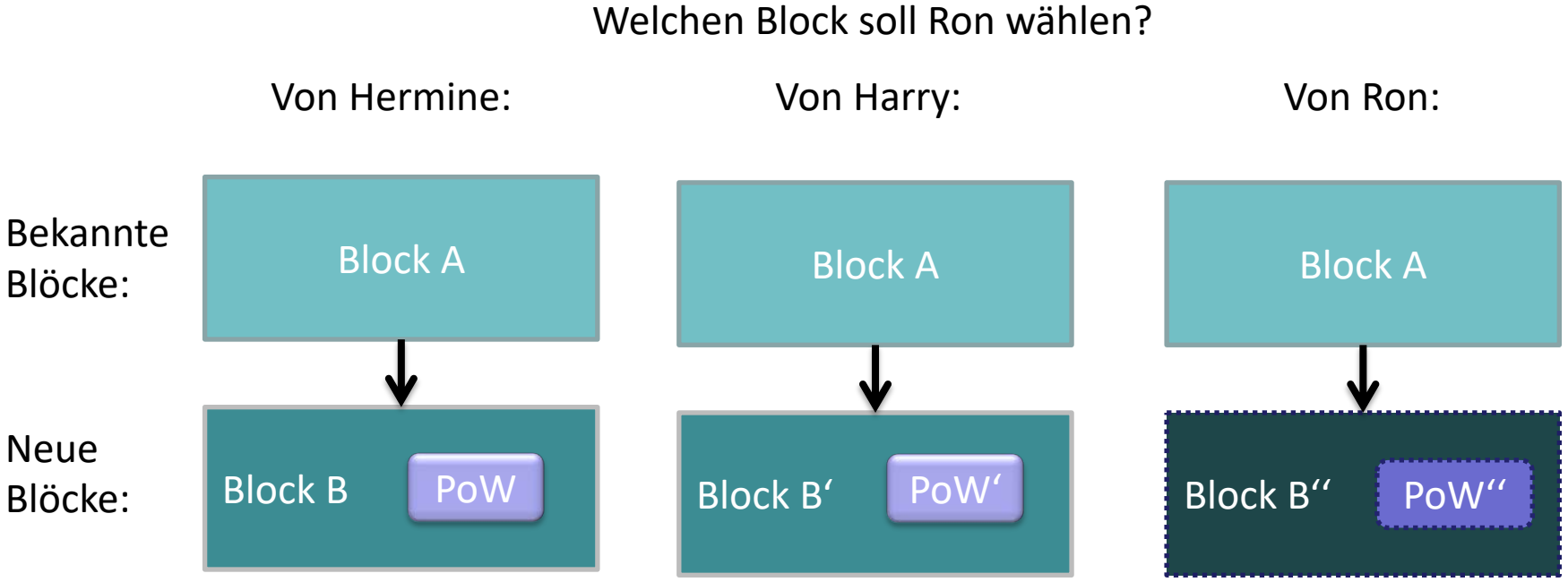
*Hashfunktion(4EF..., 3GH..., **578.321**) < 100...*



Welche Ketten werden übernommen?

Problem: Zauberer sind verteilt, d.h. Ketten können am Ende variieren, wenn viele Zaubernde einen neuen Block erstellen.

Welchen Block soll Ron wählen?

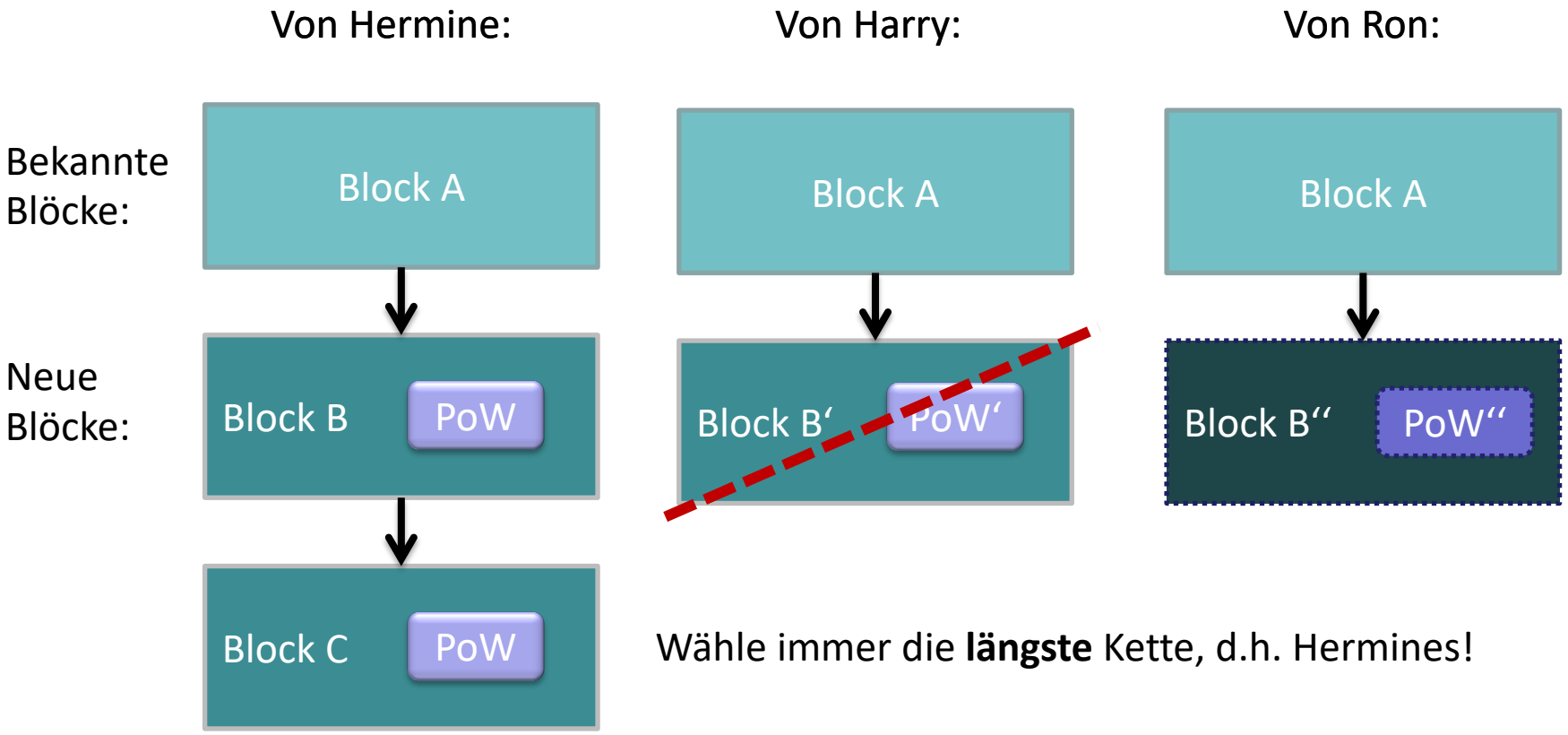


- PoW kann für beide Blöcke getestet werden
- Duelle sind konsistent und Blöcke gefüllt

→ Ron kann sich für einen von beiden entscheiden, denn noch ist keiner wirklich besser als der andere Block

Welche Ketten werden übernommen?

Problem: Zauberer sind verteilt, d.h. die Kette kann am Ende variieren, wenn viele Zaubernde einen neuen Block erstellen.



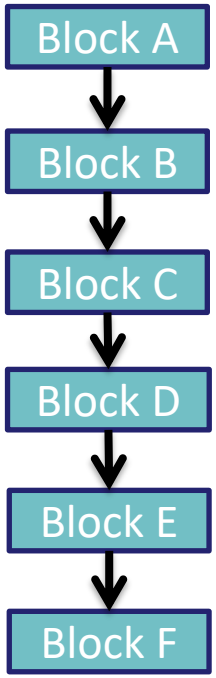
Wähle immer die **längste** Kette, d.h. Hermines!

Kann Draco schummeln? Revisited

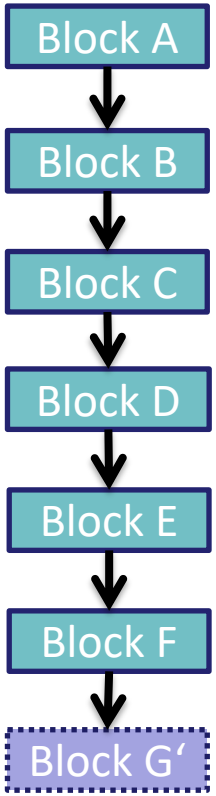
Draco kämpft gegen Harry, Harry gewinnt. Beide tragen Event in letzten Block ein

Problem: Verändere die Kette so, dass Draco in der Gegenwart gewonnen hat.

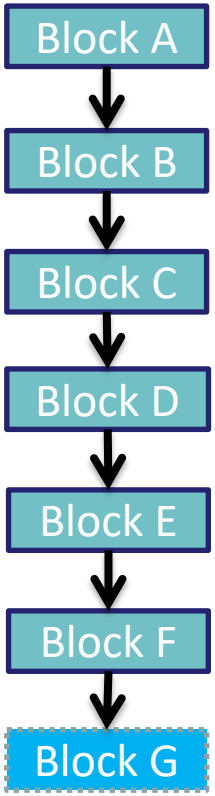
Aktuelle Kette:



Dracos Kette:



Harrys Kette:



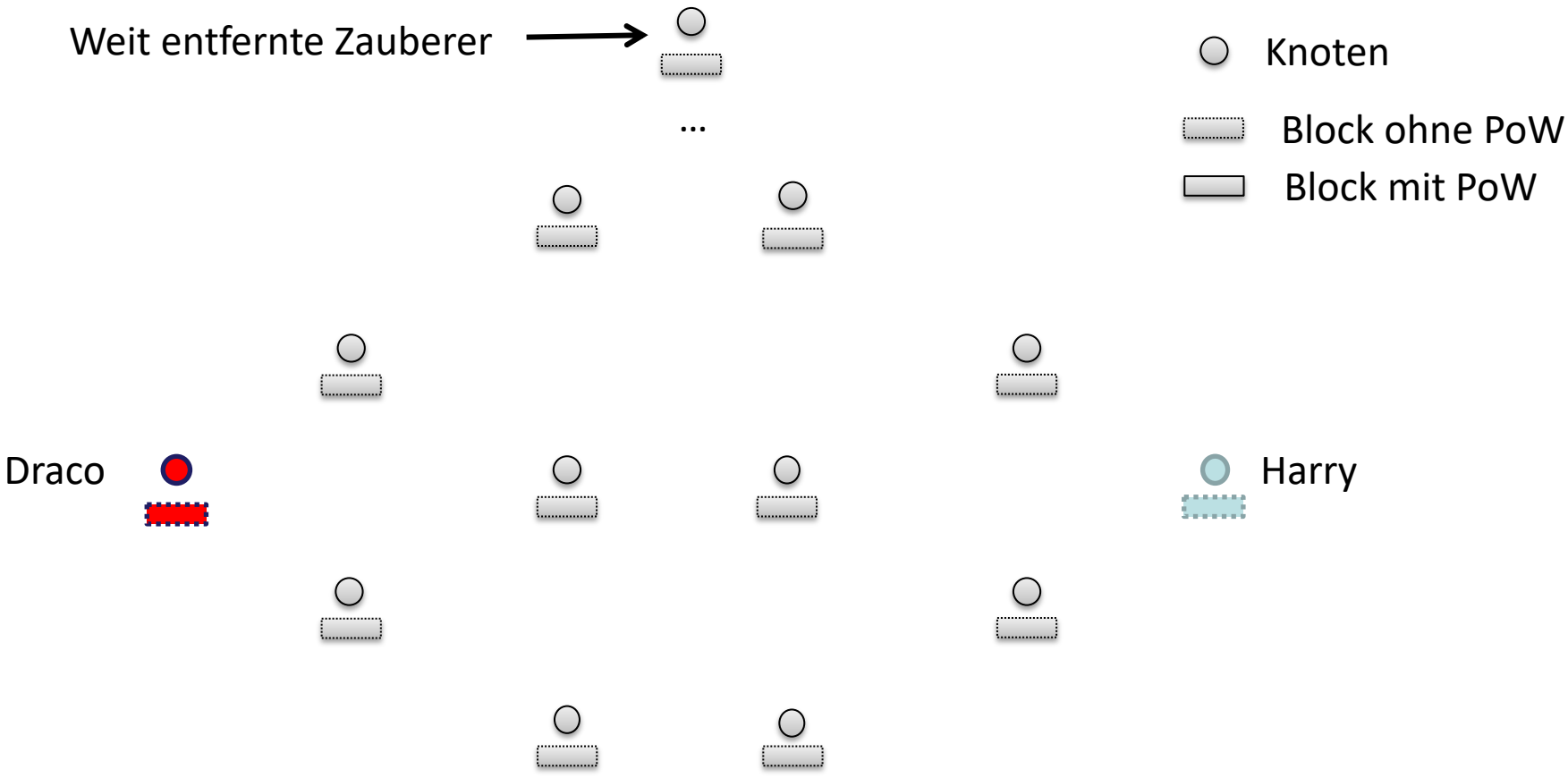
Draco muss erstmal einen PoW leisten, damit er seinen Block verteilen kann.

Draco muss schneller sein als Harry!

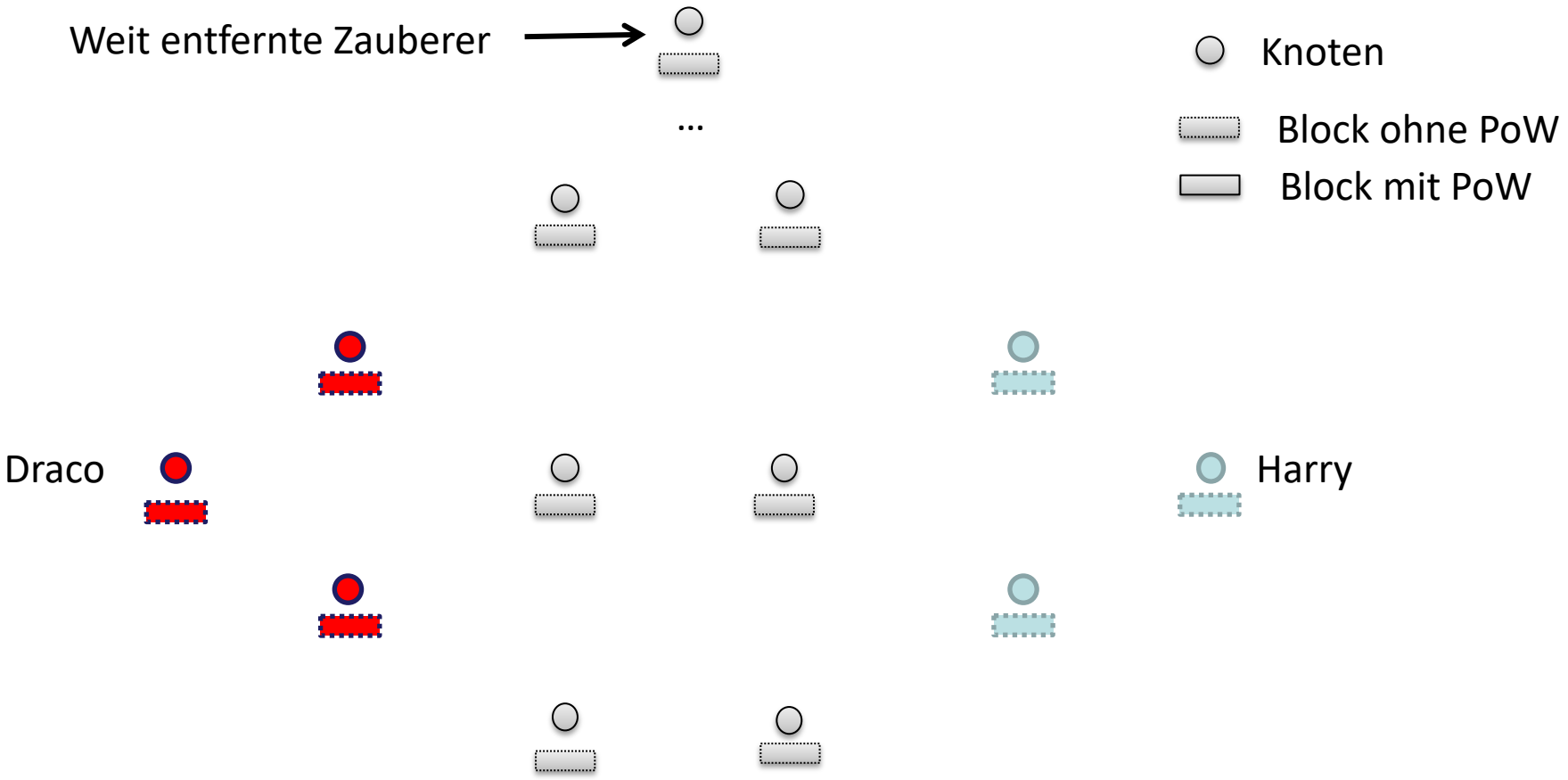
Gewinnchance: 50 zu 50*

*bei gleicher Reichweite im Netzwerk

Kann Draco schummeln? Revisited



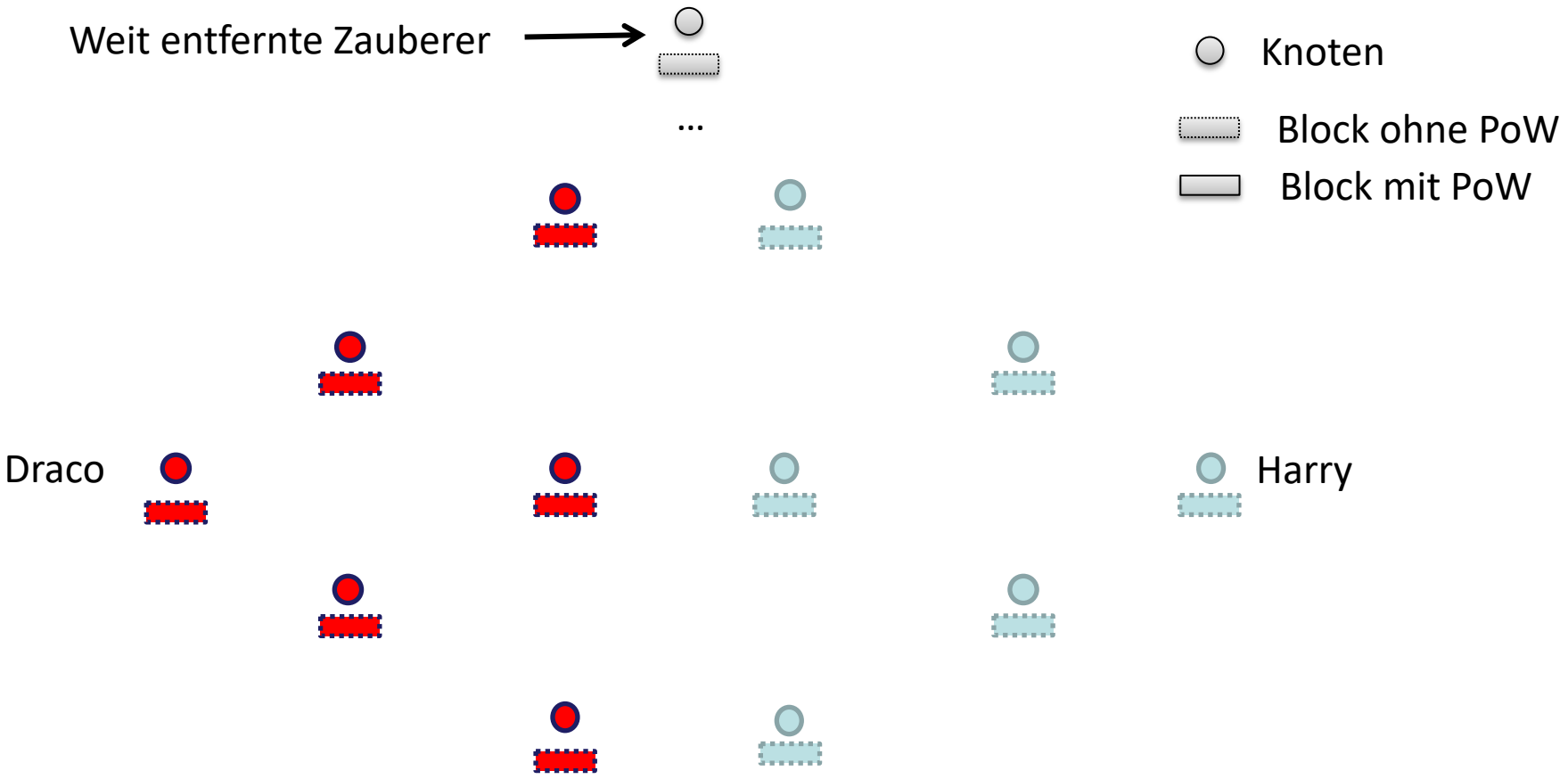
Kann Draco schummeln? Revisited



Draco verbreitet, dass er gewonnen hat

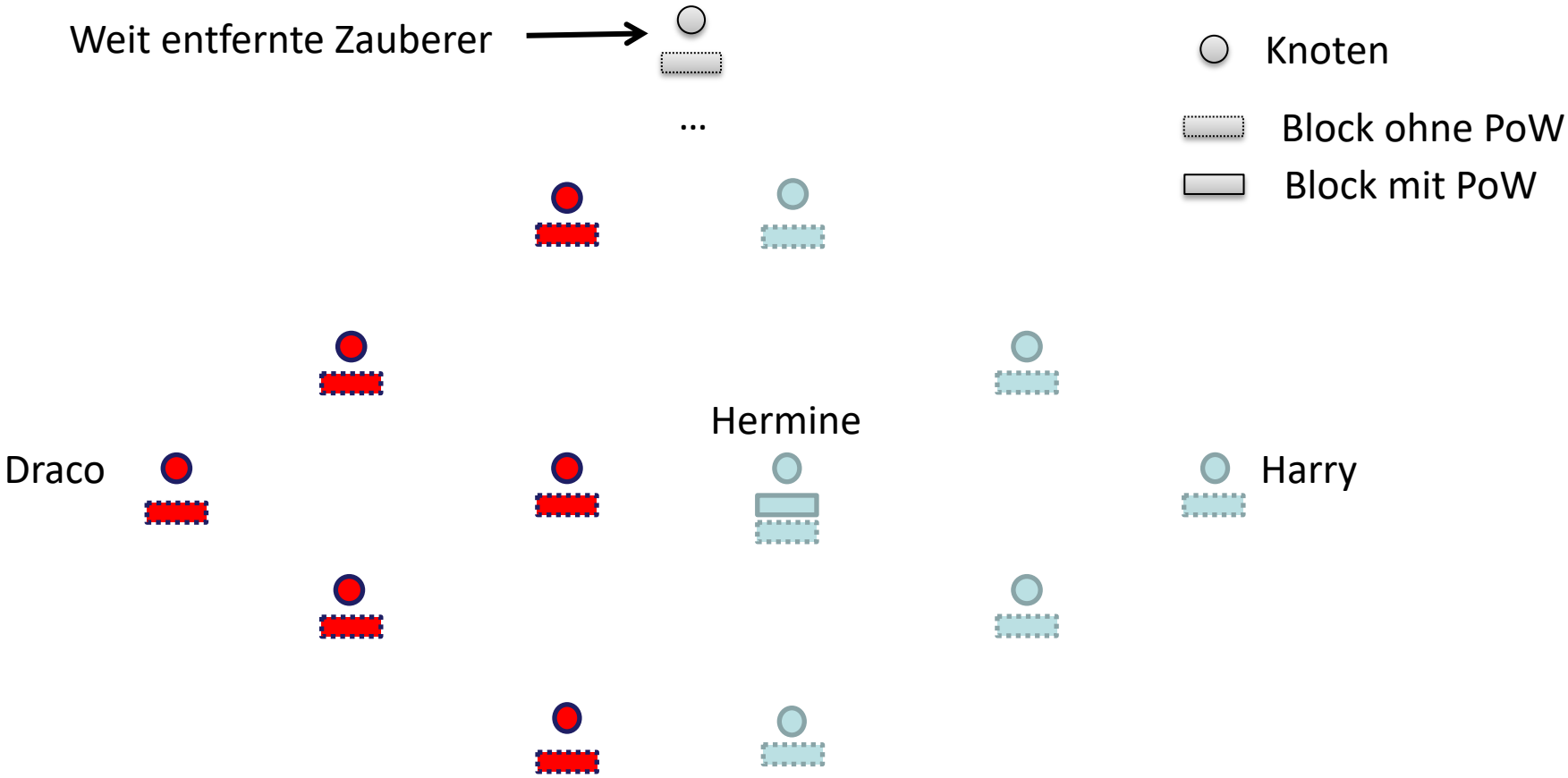
Harry verbreitet, dass er gewonnen hat

Kann Draco schummeln? Revisited



Es kann nur einen Sieger pro Duell geben!
Kein Knoten wird beide als Sieger in seinen Block aufnehmen.

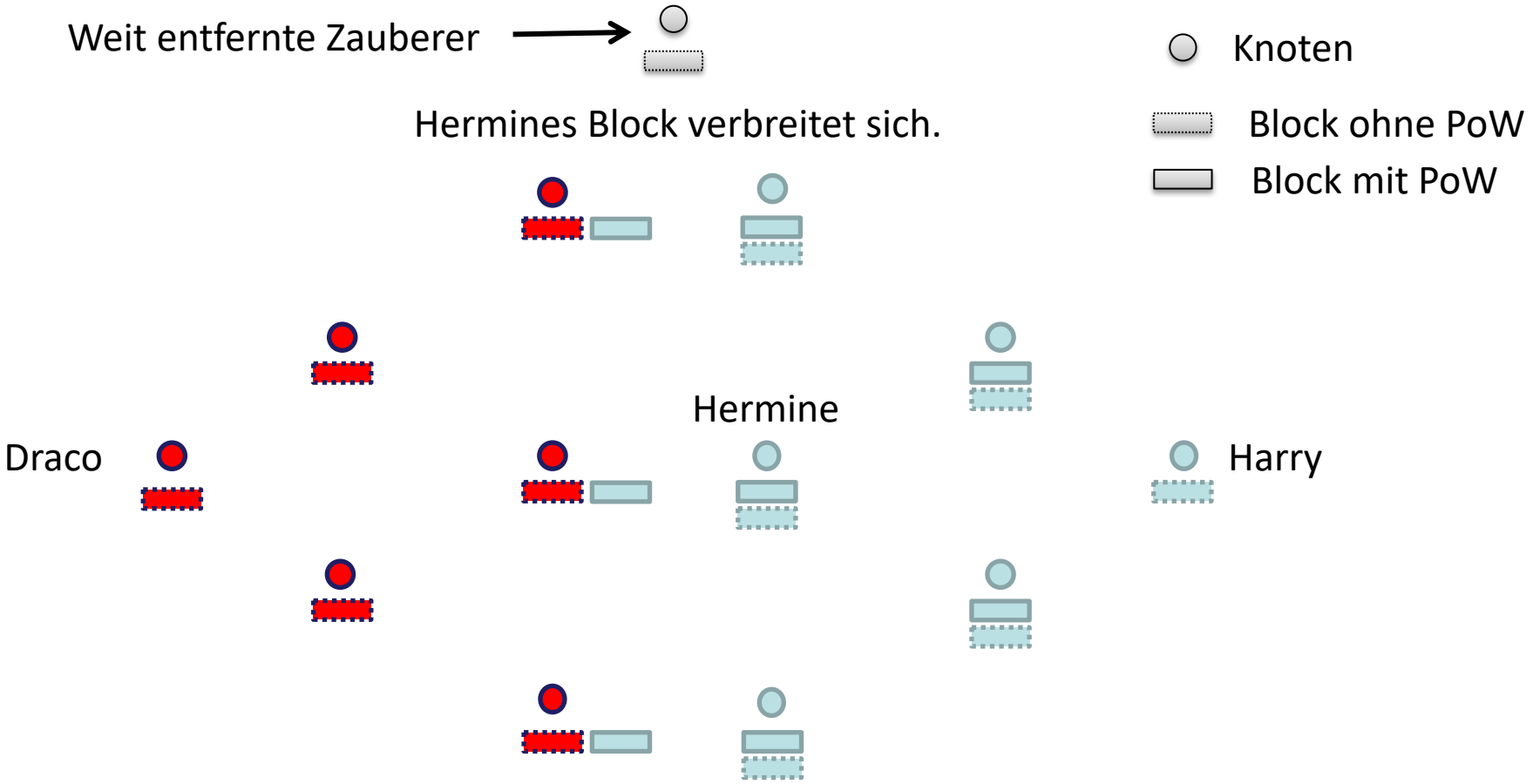
Kann Draco schummeln? Revisited



Der Knoten Hermine erfüllt den **Proof of Work** für Harrys Version.

Hermine arbeitet nun an Nachfolgeblock

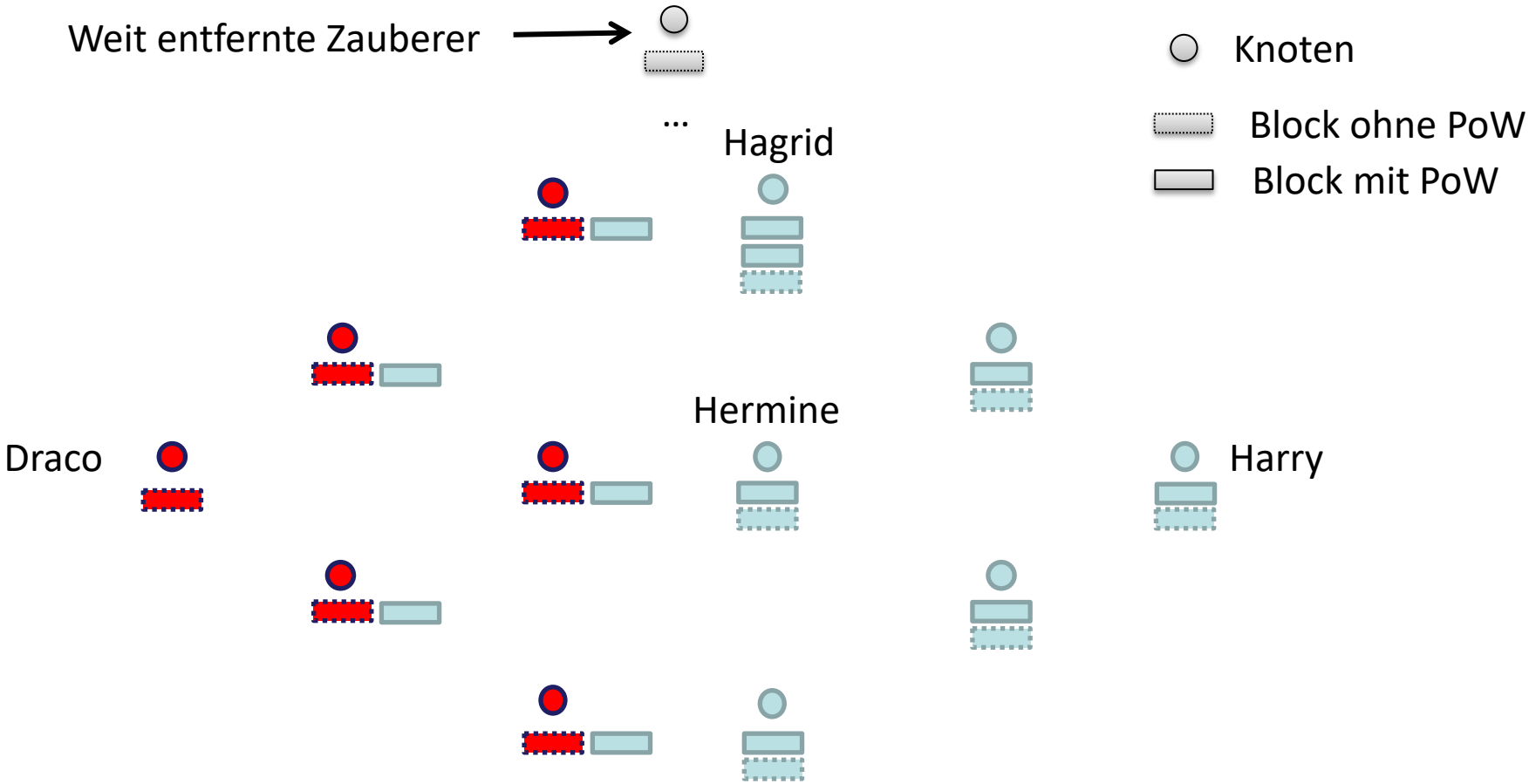
Kann Draco schummeln? Revisited



Gruppe Draco erhält Hermines Block, akzeptiert diesen aber nicht.

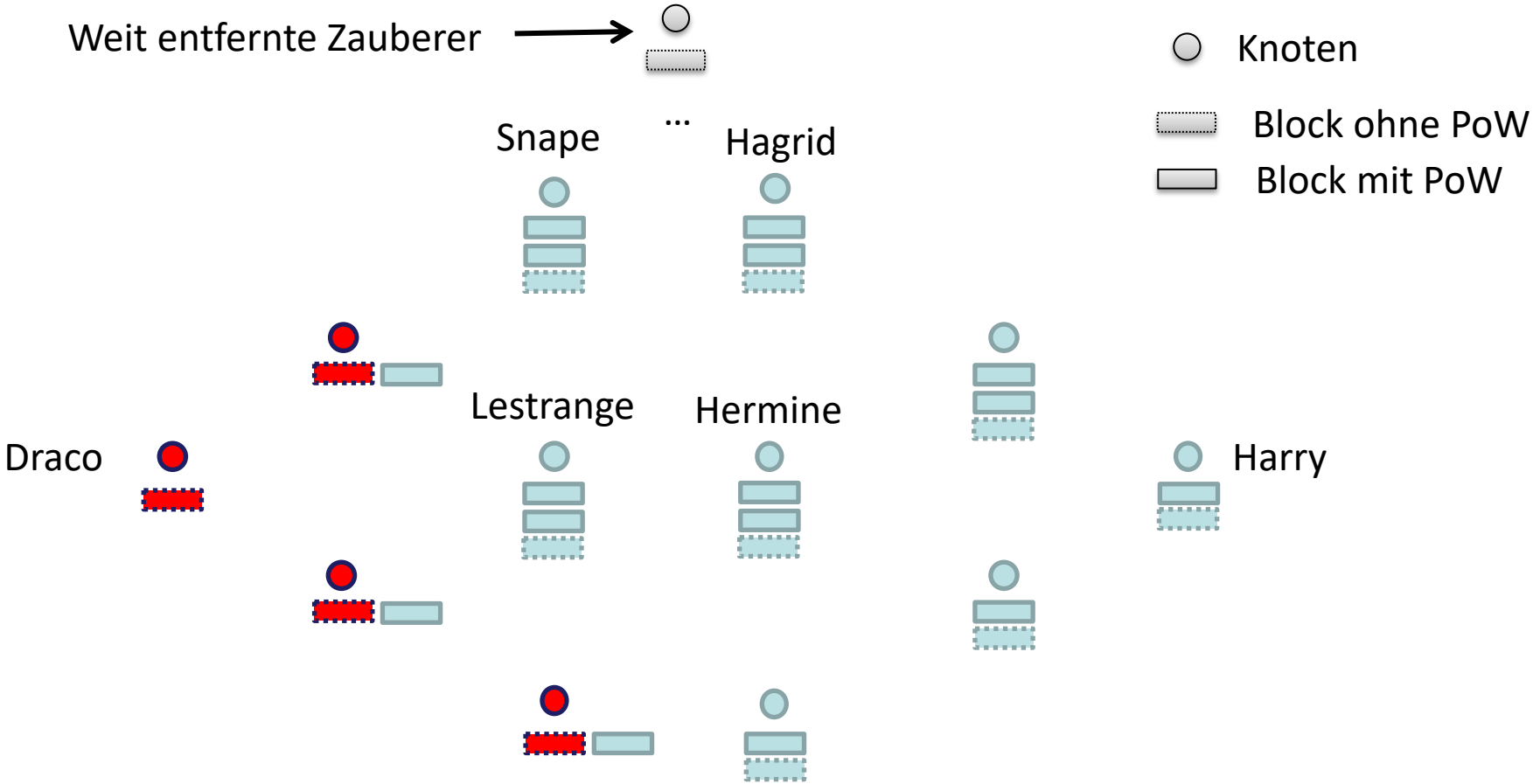
Gruppe Harry arbeitet nun an Nachfolgeblock

Kann Draco schummeln? Revisited



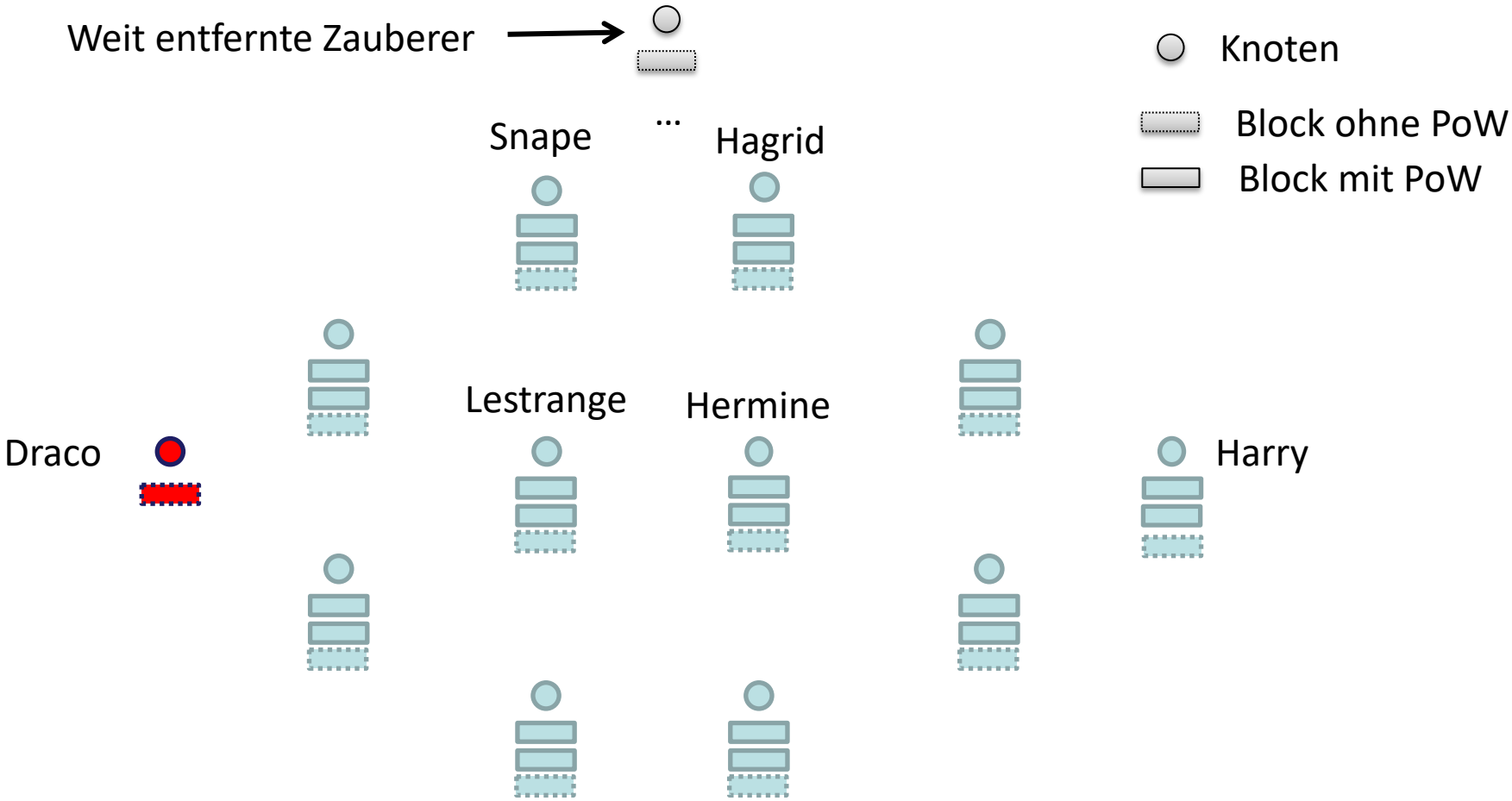
Hagrid erfüllt **Proof of Work** mit einem Block der auf Hermines aufbaut.

Kann Draco schummeln? Revisited



Snape und Lestrange erkennen längere Kette an und arbeiten nun an der Version in der Harry gewonnen hat.

Kann Draco schummeln? Revisited

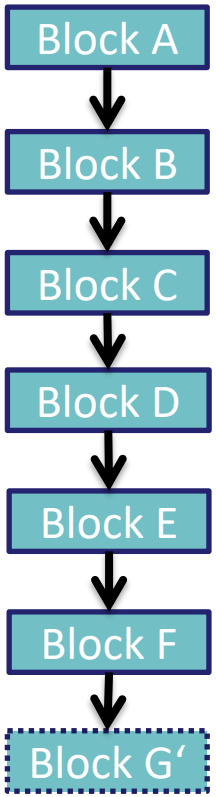


Es sieht schlecht aus für Draco…

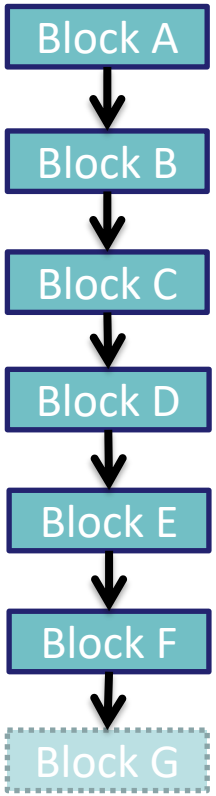
Kann Draco schummeln? Revisited

Problem: Verändere die Kette so, dass Draco in der Gegenwart gewonnen hat.

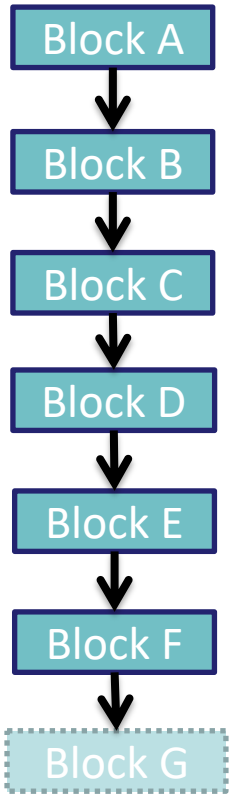
Dracos Kette:



Harrys Kette:



Kette von Dumbledore's Armee:

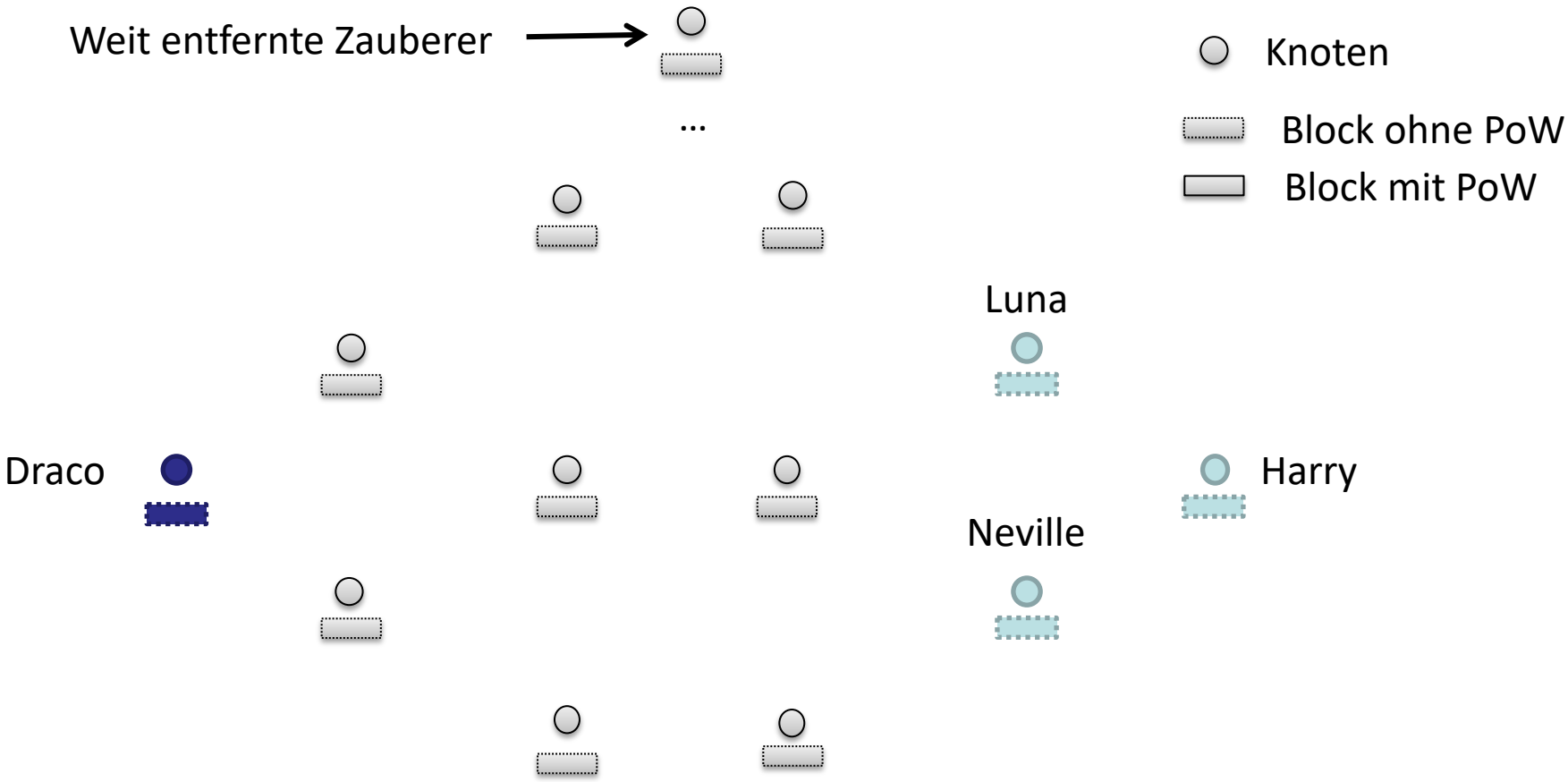


Wahrscheinlichkeit, dass einer der anderen Zauberer oder Harry seine Sicht veröffentlichen kann größer!

Plötzlich Rennen gegen viele andere!

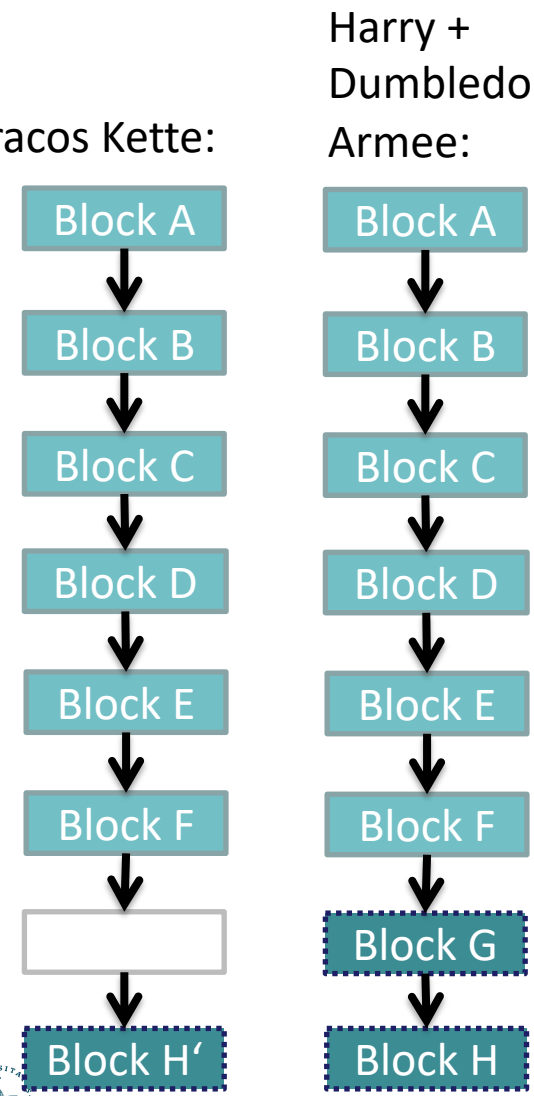
Dracos Chancen, einen neuen Block vor den anderen zu erstellen, sinken mit der Anzahl der anderen Beteiligten.

Kann Draco schummeln? Revisited



Schon von der Ausgangslage hat Draco ein Problem, hat aber trotzdem noch die Möglichkeit zu gewinnen

Kann Draco schummeln? Revisited



Problem: Verändere die Kette so, dass Draco in der Gegenwart gewonnen hat.

Draco hat trotzdem immer noch eine Chance seinen **Block G‘** zu verteilen.

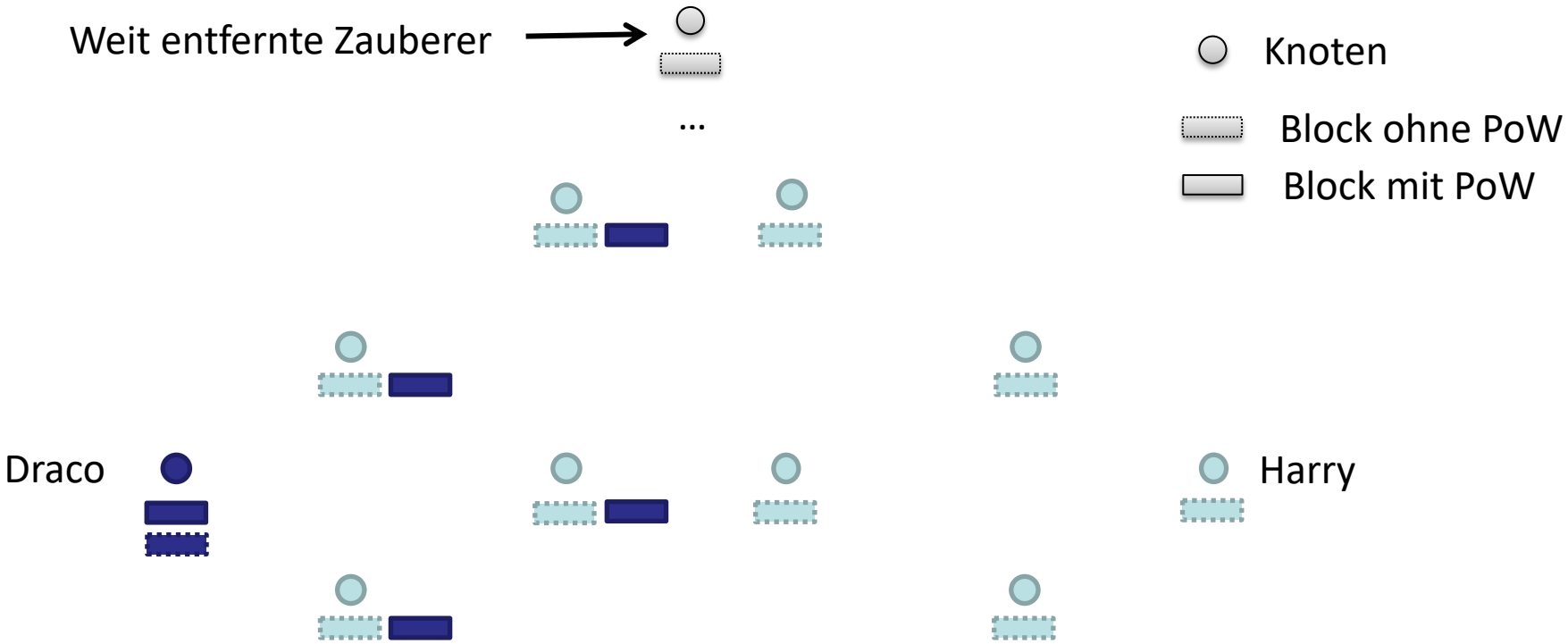
Damit ist der **Block H‘** durch den Hash allerdings anders als der von der anderen Gruppe.
Und Draco muss **erneut** seinen **Proof of Work** für diesen Block alleine leisten.

Die Wahrscheinlichkeit dafür, dass jemand aus Harrys Gruppe den **Proof of Work** für **G** und **H** findet, steigt mit der Anzahl der Teilnehmenden.

Dracos **G‘** würde verworfen, wenn die Blöcke **G** und **H** vor **H‘** veröffentlicht werden.

Mehrheiten haben bessere Chancen!

Kann Draco schummeln? Revisited



Trotz des Vorsprungs muss Draco alleine ein **Proof of Work** für den nächsten Block erfüllen, bevor der Rest umgestimmt wird.

Harrys Gruppe muss nachziehen, hat aber durch die Anzahl eine höhere Chance auf einen **Proof of Work.**

Wie sieht es mit CAP aus?

Konsistenz

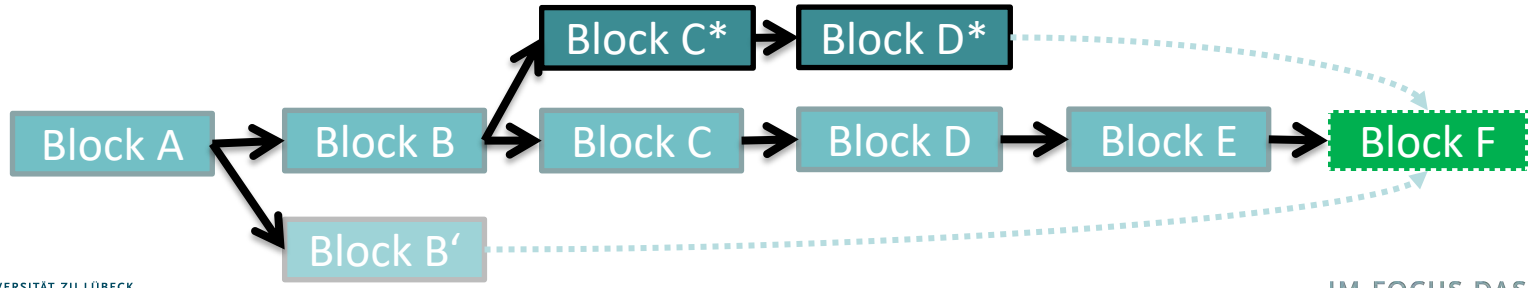
Pro:

- Neue Teilnehmer erhalten **immer** komplette Kette, heißt:
  - Eigenes Nachverfolgen **aller** Daten
  - **Anfang** und **Mitte** der Kette relativ sicher vor Veränderung
- Verworfene Blöcke können in neue Blöcke eingepflegt werden, um Transaktionen zu retten

Contra:

- Verteilung der Transaktionen, Proof of Work und Verteilung der Blöcke brauchen Zeit

Aussagen von unterschiedlichen Teilnehmern am **Ende** der Kette eher **inkonsistent**



Wie sieht es mit CAP aus?

Availability

Jeder Teilnehmende besitzt komplette Kette.
Direkte Antwort immer möglich

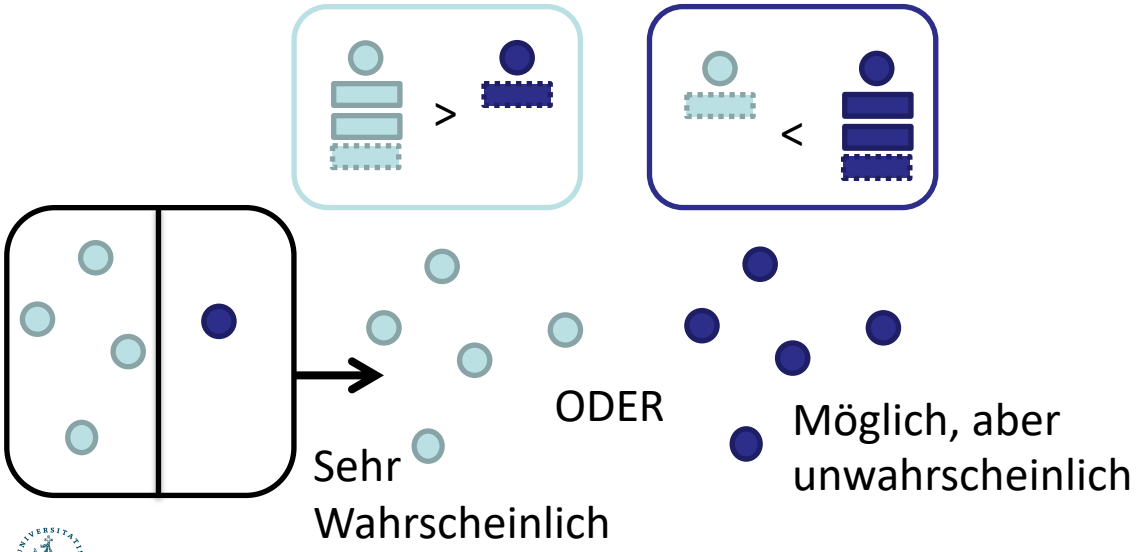
Partitionierungstoleranz

Trennung in Subnetze möglich
Bei Wiedervereinigung setzt sich längste Kette durch

Knoten

Block ohne PoW

Block mit PoW



Sehr Wahrscheinlich

ODER

Möglich, aber unwahrscheinlich

Damit ist eine Blockchain ein **AP**-System (AP/EC)!

Zusammenfassung

- Blockchains bestehen aus verketteten Blöcken, die **beliebige** Daten enthalten können.
- **Proof of Work** ermöglicht Konsensverfahren ohne zentrale Instanz und limitiert die neu generierten Blöcke.
- Die **längste** Kette gewinnt.
- Manipulation durch Hashverfahren innerhalb der Liste ausgeschlossen.
- Manipulation am **Ende** möglich, aber nur mit großer Mehrheit (>50%) erfolgsversprechend

Übersicht

- Semistrukturierte Datenbanken (JSON, XML) und Volltextsuche
- Information Retrieval
- Mehrdimensionale Indexstrukturen
- Cluster-Bildung
- Einbettungstechniken
- First-n-, Top-k-, und Skyline-Anfragen
- Probabilistische Datenbanken, Anfragebeantwortung, Top-k-Anfragen und Open-World-Annahme
- Probabilistische Modellierung, Bayes-Netze, Anfragebeantwortungsalgorithmen, Lernverfahren,
- Temporale Datenbanken und das relationale Modell, SQL:2011
- Probabilistische Temporale Datenbanken
- SQL: neue Entwicklungen (z.B. JSON-Strukturen und Arrays), Zeitreihen (z.B. TimeScaleDB)
- Stromdatenbanken, Prinzipien der Fenster-orientierten inkrementellen Verarbeitung
- Approximationstechniken für Stromdatenverarbeitung, Stream-Mining
- Probabilistische raum-zeitliche Datenbanken und Stromdatenverarbeitungsssysteme: Anfragen und Indexstrukturen, Raum-zeitliches Data Mining, Probabilistische Skylines
- Von NoSQL- zu NewSQL-Datenbanken, CAP-Theorem, Blockchain-Datenbanken