Non-Standard-Datenbanken

Von NoSQL zu NewSQL

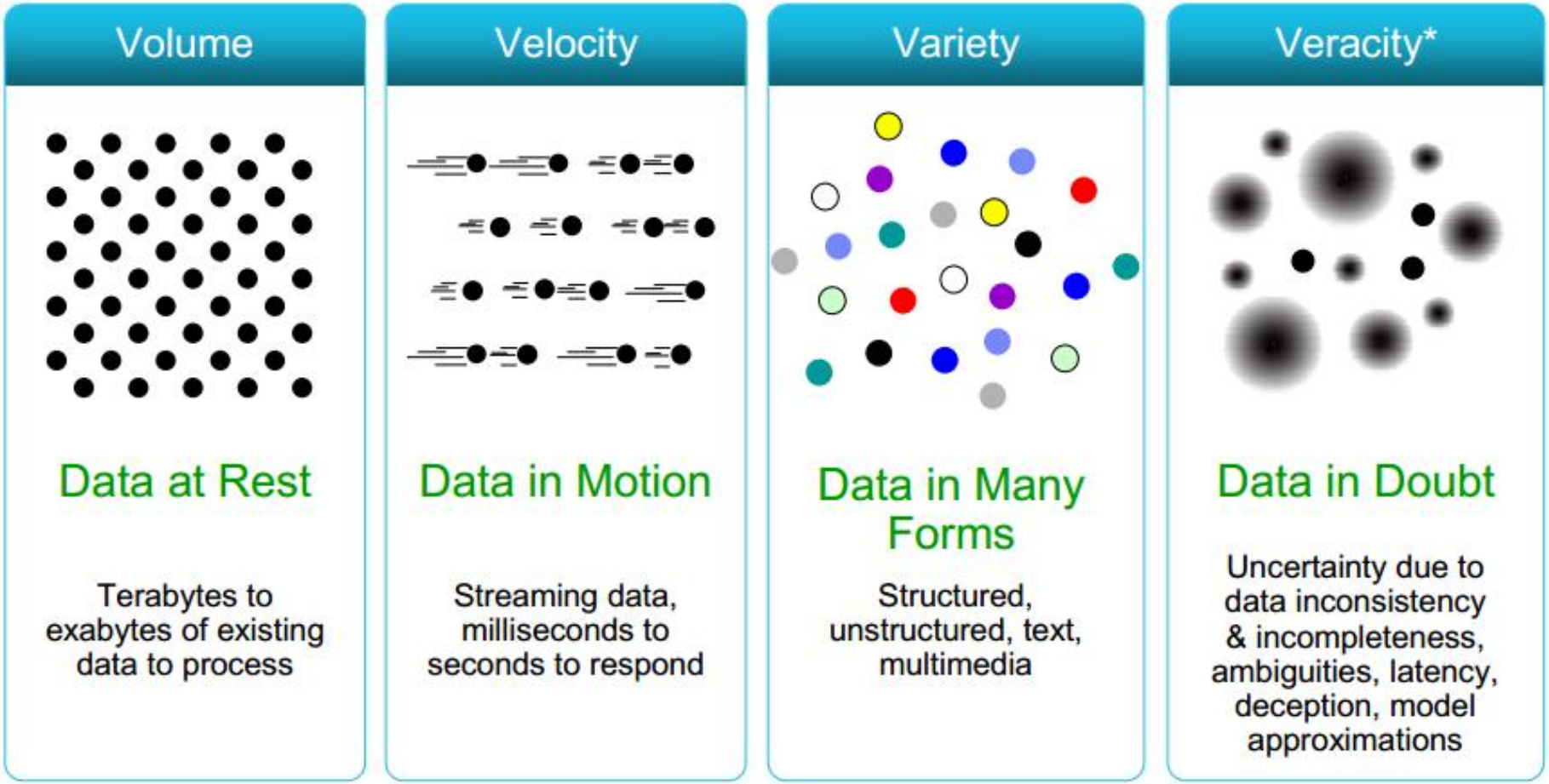
Prof. Dr. Ralf Möller

Universität zu Lübeck

Institut für Informationssysteme

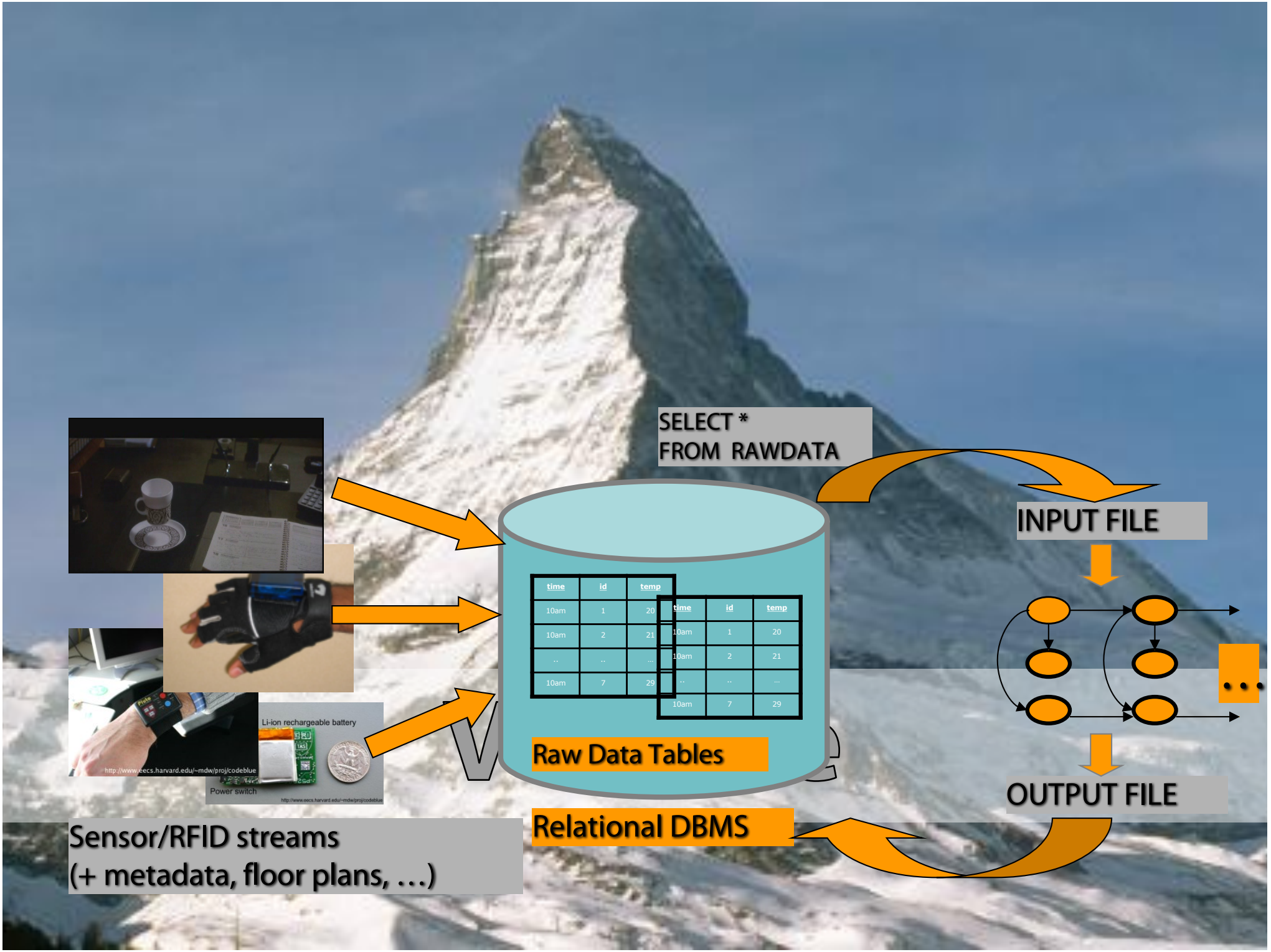
Felix Kuhr (Übungen)

Big Data


Volume
Data at Rest
Terabytes to exabytes of existing data to process
Velocity
Data in Motion
Streaming data, milliseconds to seconds to respond
Variety
Data in Many Forms
Structured, unstructured, text, multimedia
Veracity*
Data in Doubt
Uncertainty due to data inconsistency & incompleteness, ambiguities, latency, deception, model approximations

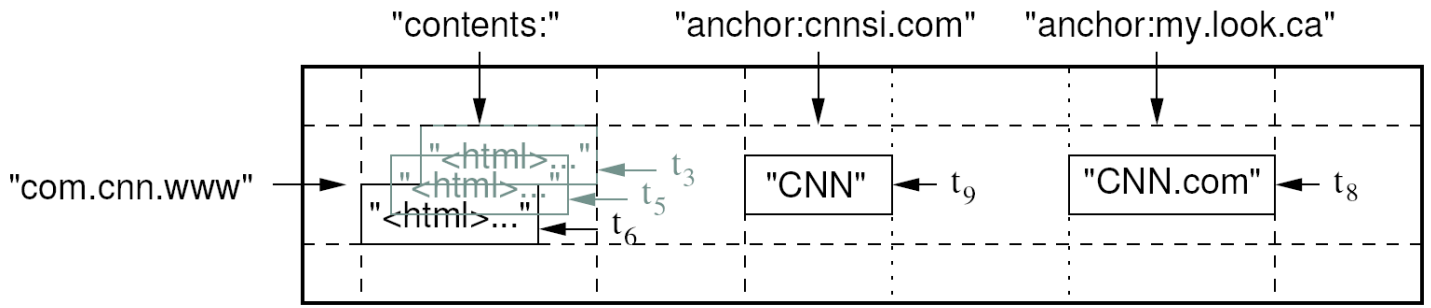
NoSQL: Not Only SQL?

- 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, Paritioning 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


SELECT *
FROM RAWDATA
INPUT FILE
time id temp
10am 1 20
10am 2 21
.. .. ..
10am 7 29
Raw Data Tables
Relational DBMS
OUTPUT FILE
Li-ion rechargeable battery
Power switch
http://www.eecs.harvard.edu/~mdw/proj/codeblue
Sensor/RFID streams
(+ metadata, floor plans, …)

Bigtable (Google) und Dynamo (Amazon)

- Eine Tabelle in Bigtable ist eine dünn besetzte, multidimensionale, persistente und verteilte Hashtabelle
- Abbildung ist indexiert durch Zeilenschlüssel, Spaltenschlüssel und Zeitstempel
  - (row:string, column:string, time:int64) → uninterpretiertes Bytefeld
- Unterstützung von Suchen, Einfügen, Löschen
  - Transaktionen nur zeilenweise etablierbar



F. Chang, J. Dean, S. Ghemawat, W.C. Hsieh, D.A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R.E. Gruber. Bigtable: A Distributed Storage System for Structured Data. *ACM Trans. Comput. Syst.* 26, 2, Article 4, **2008**

Image Source: Chang et al., OSDI 2006

Map Reduce

- Technik zur Indizierung und Verarbeitung von großen Datenmengen
- Implementierung durch Anwendungsentwickler, kein deklaratives DBMS
- Zwei Phasen: Map und Reduce
  - Map
    - Extraktion von Mengen von Key-Value-Paaren aus Daten
    - Potentiell auf vielen Maschinen parallel
  - Reduce
    - Mischung und Sortierung von Key-Value-Paaren
    - Resultat in anderen Verarbeitungskontexten weiterverwendet

Map-Reduce-Patent

Google granted US Patent 7,650,331, January 2010

System and method for efficient large-scale data processing

A large-scale data processing system and method includes one or more application-independent map modules configured to read input data and to apply at least one **application-specific map operation** to the input data to produce intermediate data values, wherein the map operation is automatically parallelized across multiple processors in the parallel processing environment. A plurality of intermediate data structures are used to store the intermediate data values. One or more application-independent reduce modules are configured to retrieve the intermediate data values and to apply at least one **application-specific reduce operation** to the intermediate data values to provide output data.

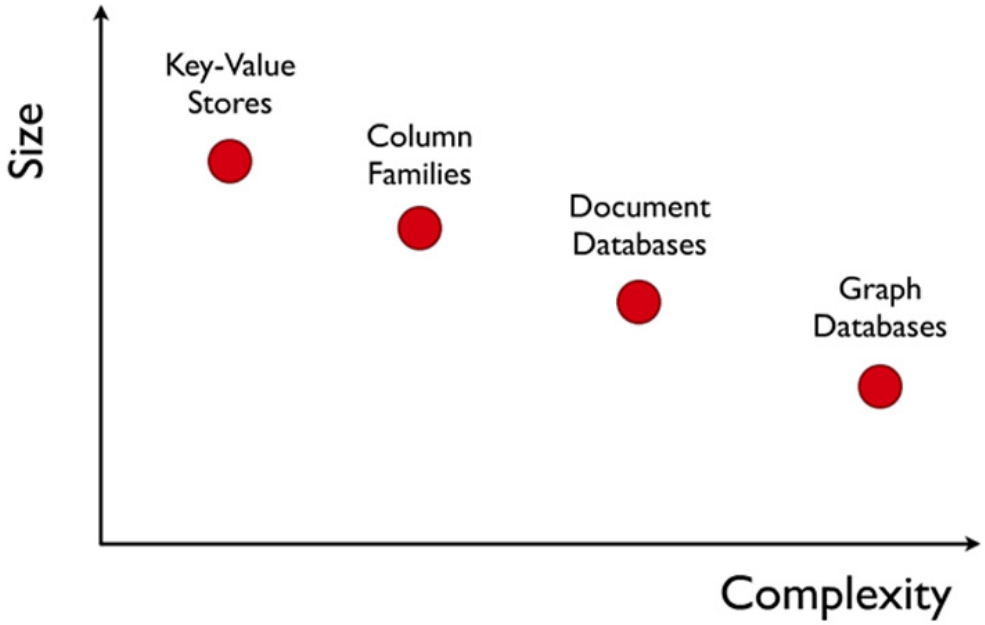
NoSQL-Beispiel: Key-Value Store

- Hash-Tabelle von Schlüsseln
- Werte mit Schlüsseln zusammen gespeichert
- Schneller Zugriff auf (kleine) Datenwerte
- Voldemort
  - http://www.project-voldemort.com/
- MemCacheDB
  - http://memcachedb.org/
  - Backend-Speicher ist eingebettete DB (Berkeley-DB)

NoSQL-Datenbanktypen

Die Diskussion von NoSQL-Datenbanken wird erschwert durch Vielzahl verschiedener Typen:

- Key-Value Store – Hashtabelle von Schlüsseln
- Column Store – Jeder Speicherblock enthält Daten aus nur einer Spalte
- Document Store
  "Dokumente" als Menge ausgezeichneter (tagged) Elemente
- Graphdatenbanken



Andere Nicht-SQL-Datenbanken

- Hierarchische Datenbanken (IBM, 60er Jahre)
- Netzwerk-Datenbanken (CODASYL, 70er Jahre)
- Objekt-orientierte Datenbanken (80er Jahre)
- XML-Datenbanken (90er Jahre)
- Triple Stores (2000er Jahre → Sven Groppes Vorlesung)
- …

NoSQL-Beispiel: Column Store

- Jeder Speicherblock enthält Daten aus nur einer Spalte
  - Effizienzgewinn, wenn nur einige Spalten relevant für Anfrage
- MonetDB
  - https://www.monetdb.org
  - MonetDB (seit 1993)
- Hadoop/Hbase
  - http://hadoop.apache.org/
  - Yahoo, Facebook
- Actian Vortex (ex: VectorWise, ex: Vector)
  - Column Store integriert in SQL-Datenbank Ingres
  - http://www.actian.com
  - Vertreibt auch Versant, ein objektorientiertes DBMS

NoSQL-Beispiel: Document Store

- Verwendung von JSON – JavaScript Object Notation
  - Geschachtelte Objekte
  - Wie XML aber "bequemer" für Java-Programmierer
  - Redundanzvermeidung?
- CouchDB
  - http://couchdb.apache.org/
- MongoDB
  - http://www.mongodb.org/

CouchDB JSON Beispiel

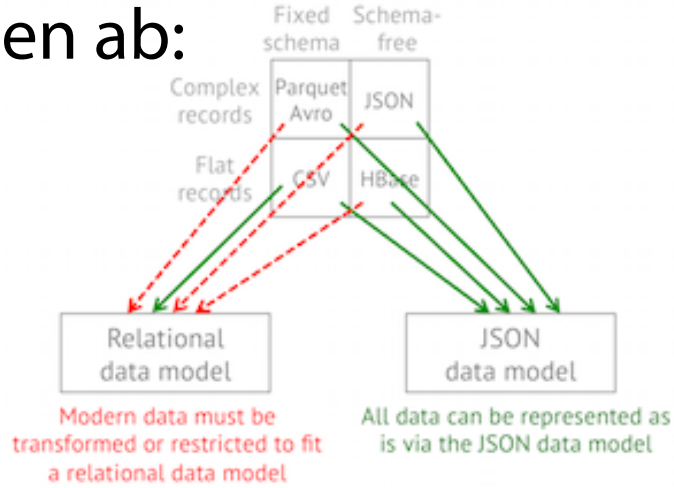
```
{
  "_id": "guid goes here",
  "_rev": "314159",

  "type": "abstract",

  "author": "Keith W. Hare"

  "title": "SQL Standard and NoSQL Databases",

  "body": "NoSQL databases (either no-SQL or Not Only SQL)
           are currently a hot topic in some parts of
           computing.",
  "creation_timestamp": "2011/05/10 13:30:00 +0004"
}
```

CouchDB JSON Tags

- "_id"
  - GUID – Global Unique Identifier
  - Passed in or generated by CouchDB
- "_rev"
  - Revision number
  - Versioning mechanism
- "type", "author", "title", etc.
  - Arbitrary tags
  - Schema-less
  - Could be validated after the fact by user-written routine

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



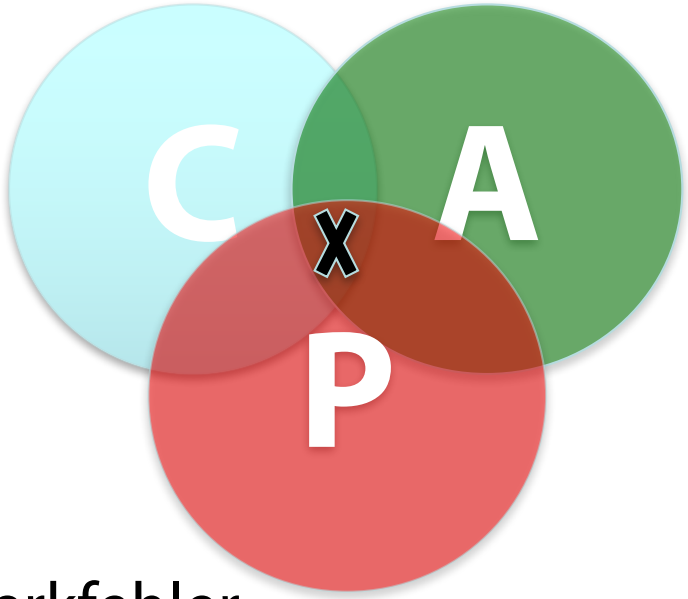
- Programmierer wird verantwortlich für
  - Prozedurale Formulierung von Zugriffsalgorithmen
  - und Navigation zu den benötigten Daten

```
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

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

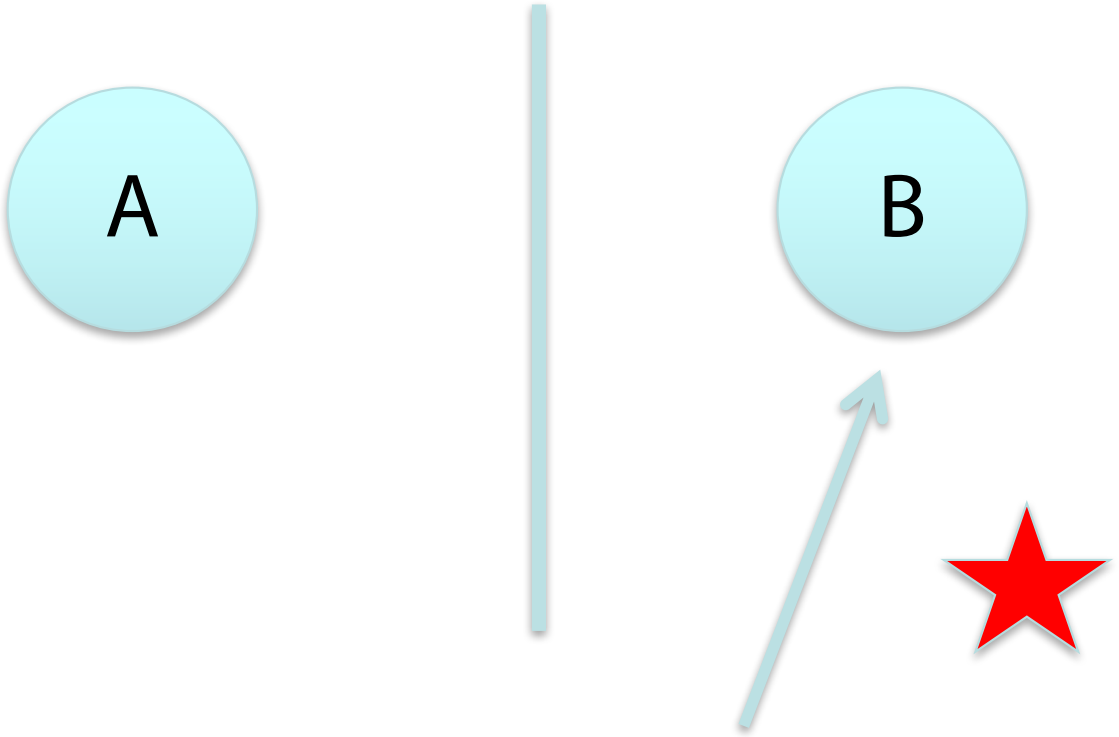


- Theorem: Ein verteiltes System kann nur jeweils zwei Kriterien erfüllen, nicht aber alle drei

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

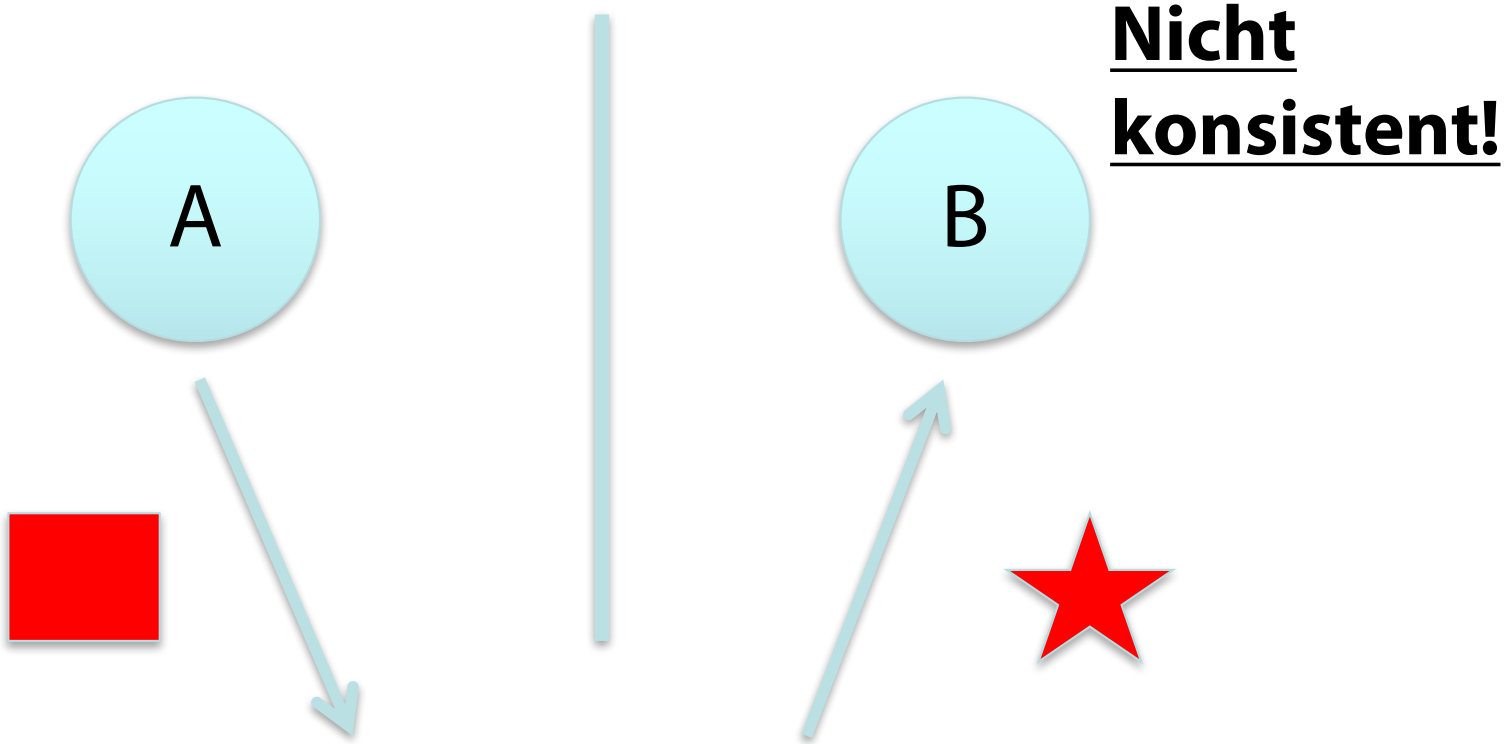
CAP Theorem: Argumente

- Nehmen wir an, es gibt zwei Knoten:



CAP Theorem: Argumente

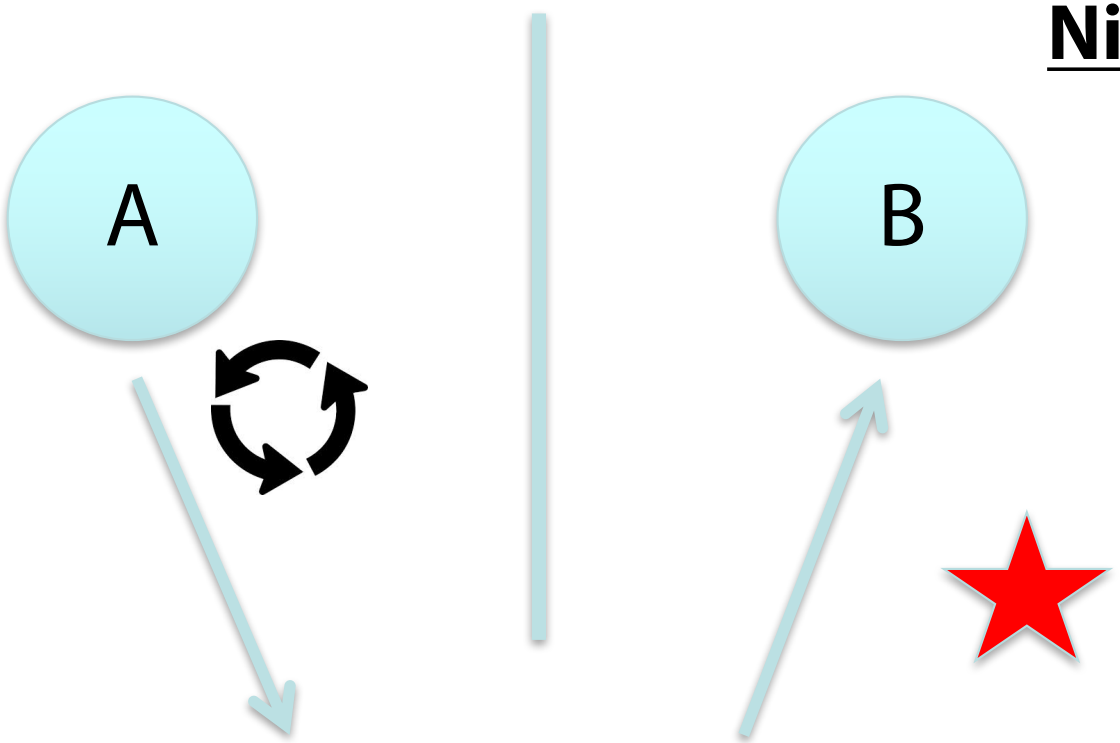
- Nehmen wir an, es gibt zwei Knoten:



CAP Theorem: Argumente

- Nehmen wir an, es gibt zwei Knoten:

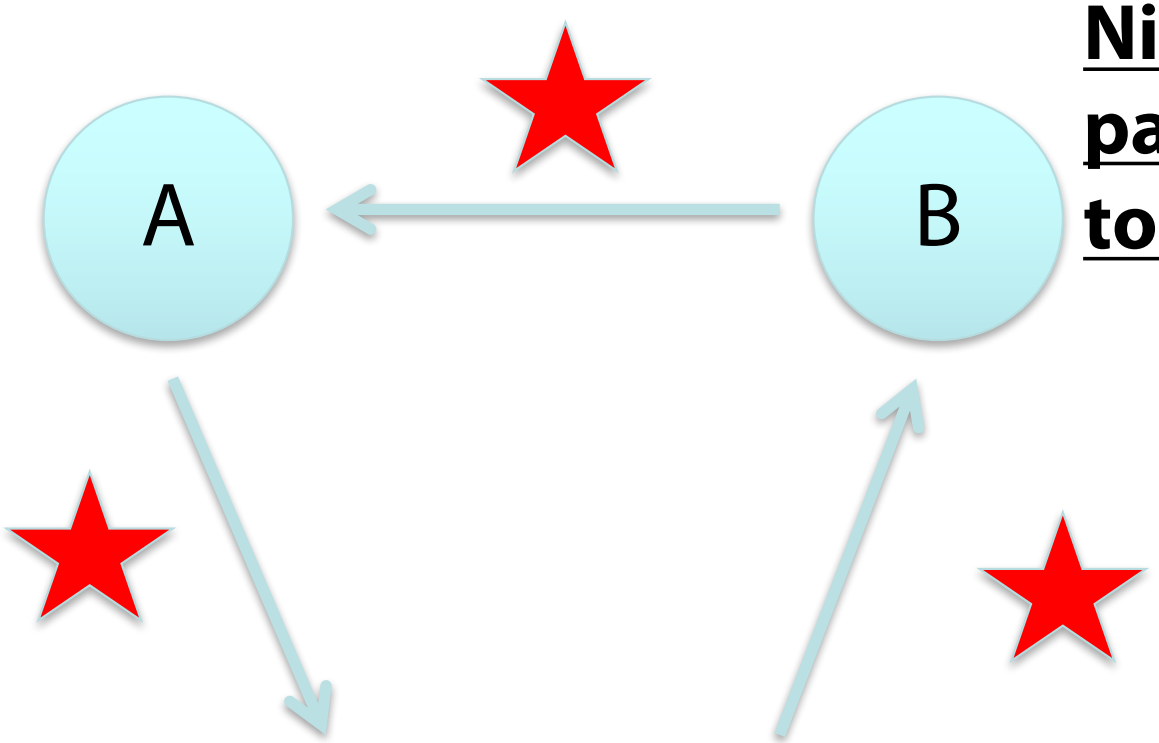
Nicht verfügbar!



Warten auf Aktualisierung

CAP Theorem: Argumente

- Nehmen wir an, es gibt zwei Knoten:



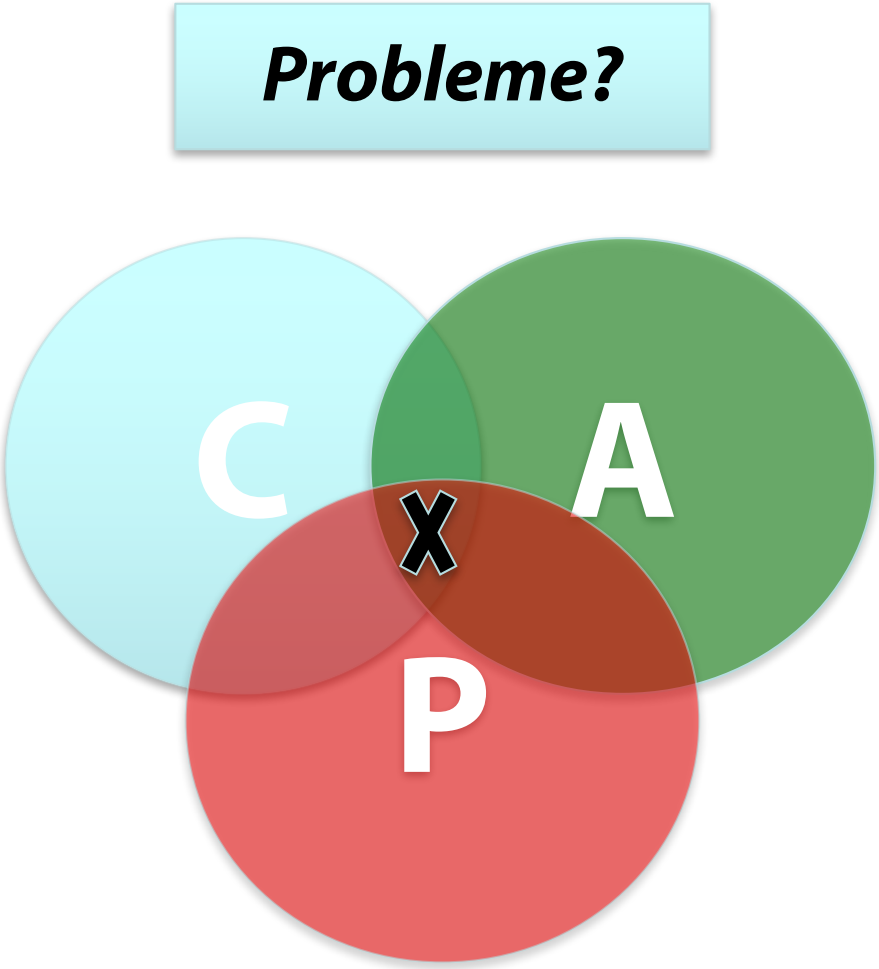
Nicht partitionierungs-tolerant!

A erhält Aktualisierung von B

Partitionierung:
System zerfällt in (mindestens) zwei unverbundenen Teile

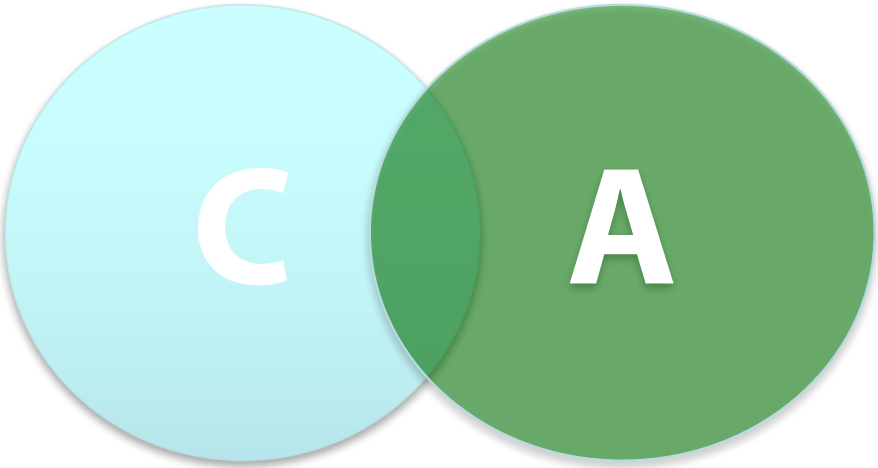
Neubetrachtung CAP Theorem

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



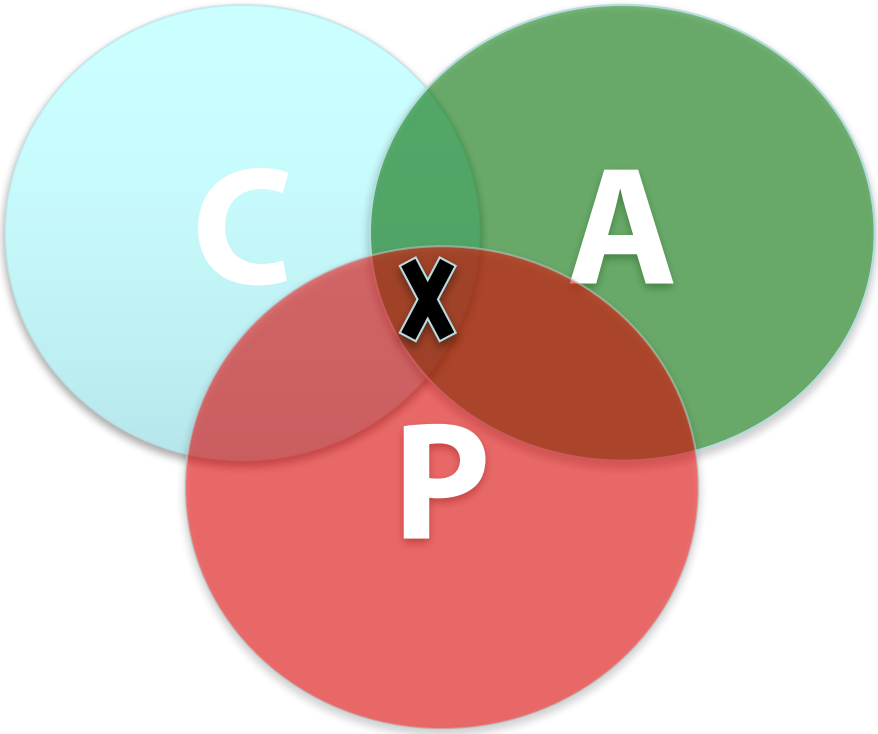
Häufiges Missverständnis: 2 von 3

- 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.

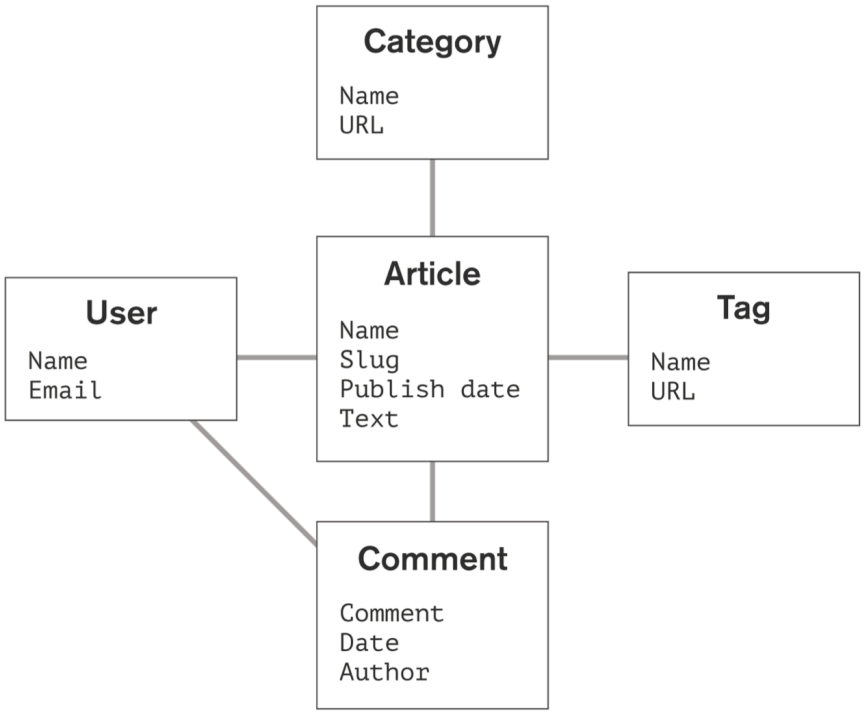
AP, CP greift zu kurz: CAP/ELC

- **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

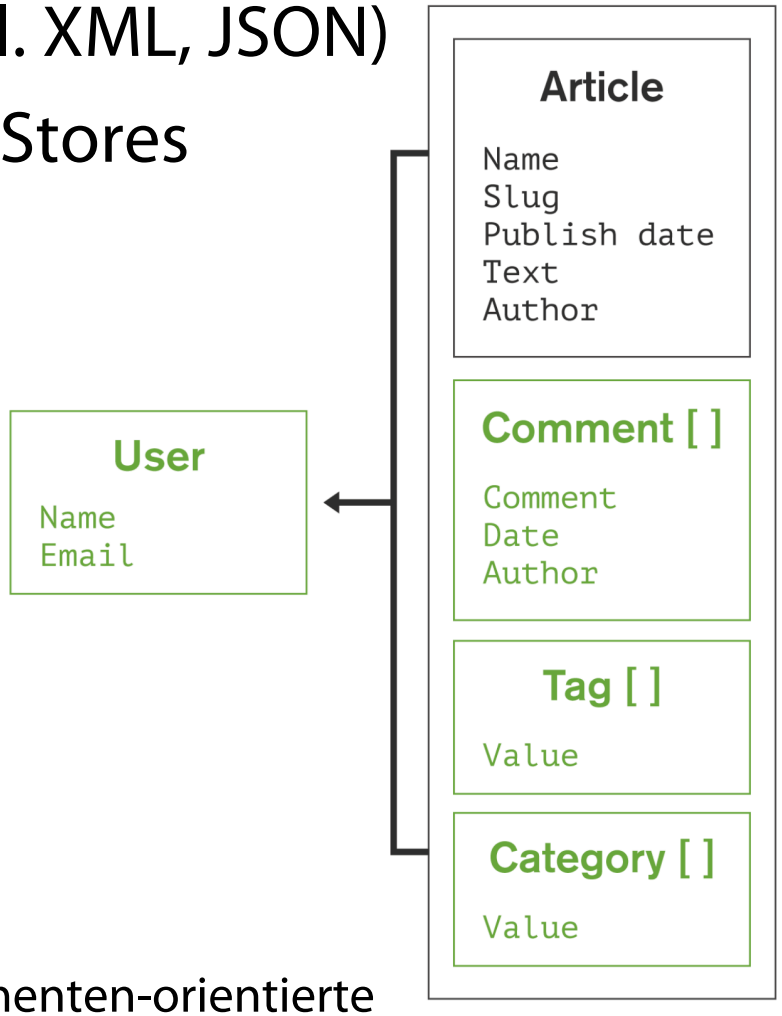
E=ELSE
L=Latency
C=Consistency

Dokument-orientierte DB: Beispiel MongoDB

- Daten als "Dokumente" (vgl. XML, JSON)
- Unterklasse von Key-Value-Stores



Relationale Darstellung



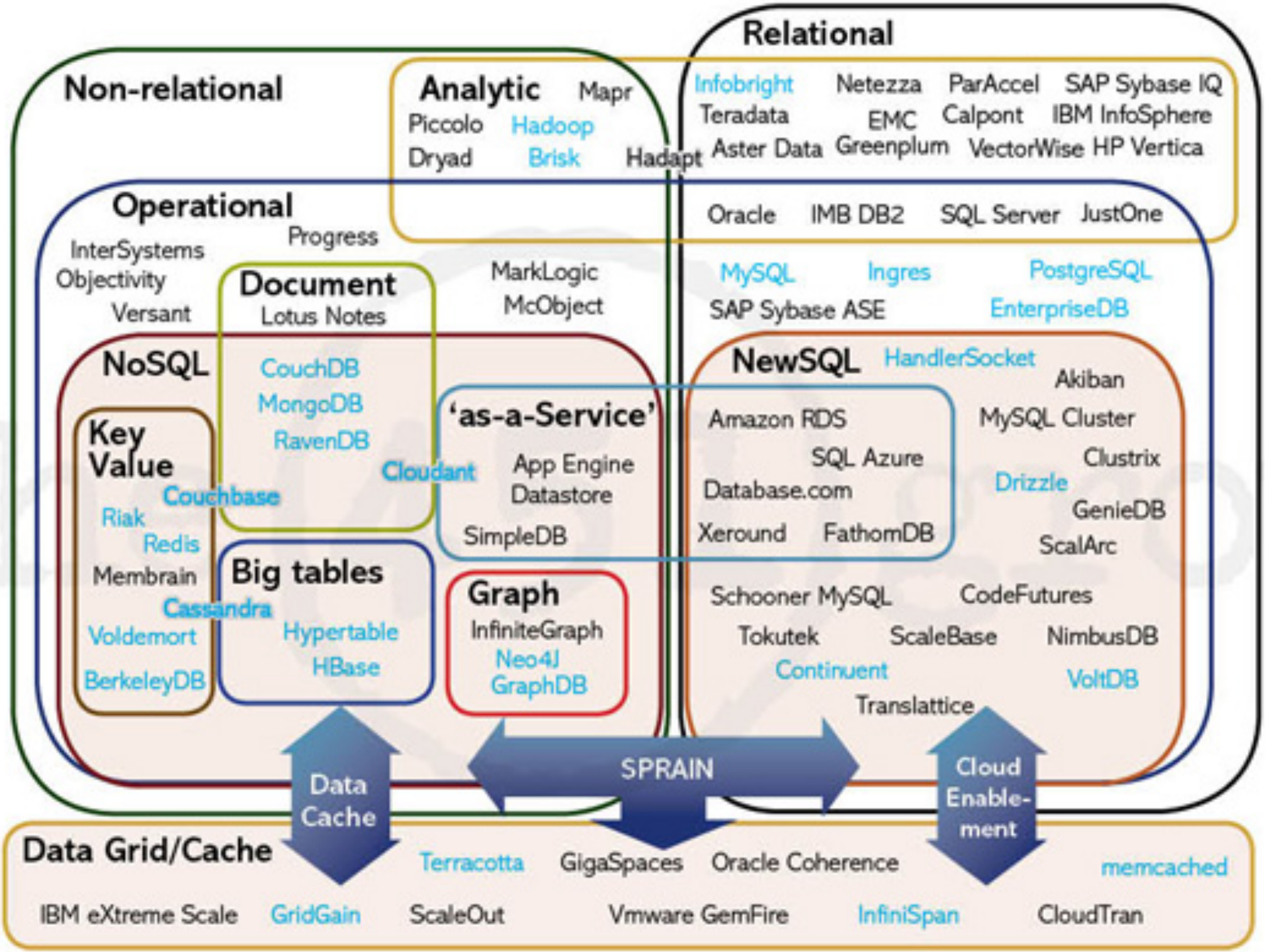
Dokumenten-orientierte Darstellung

Datenmanagement in MongoDB

- MongoDB stellt horizontale Skalierung für Datenbanken auf kostengünstigen, allgemein verfügbaren Rechensystemen bereit (mittels der sog. Sharding-Technik)
- Sharding verteilt Daten über mehrere physikalische Partitionen (shards) zur Vermeidung der Begrenzungen einzelner Rechner
- MongoDB sieht eine automatische Verteilung von Daten im Cluster vor, wenn Datenvolumina über Kapazität eines Einzelsystems hinaus gehen

NewSQL

- Neue Generation von relationalen DBMS
  - Stellen Performanz von NoSQL-Systemen bereit …
  - … für Online-Transaktionsverarbeitung (OLTP) mit Lese- und Schreiboperationen, und zwar unter Beibehaltung von ACID-Garantien relationaler DB-Systems
    - Kurze Transaktionen
    - Berühren kleiner Mengen von Daten durch Indexzugriffe (keine ganzen Tabellen gelesen oder große verteilte Joins)
    - Anfragen sind wiederkehrend (auf neuen Daten)
  - Vermeidung von schwergewichtigen Reparaturmechanismen (wie z.B. Write-Ahead-Logs)
- Unterstützung von verteilten Clustern von Rechnern ohne gemeinsamen Speicher


Relational
Non-relational
Analytic
Mapr
Infobright
Netezza
ParAccel
SAP Sybase IQ
Teradata
EMC
Calpont
IBM InfoSphere
Piccolo
Hadoop
Aster Data
Greenplum
VectorWise
HP Vertica
Dryad
Brisk
Hadapt
Operational
Oracle
IMB DB2
SQL Server
JustOne
Progress
InterSystems
Objectivity
Versant
Document
Lotus Notes
MarkLogic
McObject
MySQL
Ingres
PostgreSQL
SAP Sybase ASE
EnterpriseDB
NoSQL
CouchDB
MongoDB
RavenDB
Cloudant
Key Value
Couchbase
Riak
Redis
Membrain
Cassandra
Voldemort
BerkeleyDB
'as-a-Service'
App Engine Datastore
SimpleDB
Big tables
Hypertable
HBase
Graph
InfiniteGraph
Neo4J
GraphDB
NewSQL
HandlerSocket
Akiban
Amazon RDS
MySQL Cluster
SQL Azure
Clustrix
Drizzle
Database.com
GenieDB
Xeround
FathomDB
ScalArc
Schooner MySQL
CodeFutures
Tokutek
ScaleBase
NimbusDB
Continuent
VoltDB
Translattice
Data Cache
SPRAIN
Cloud Enable-ment
Data Grid/Cache
Terracotta
GigaSpaces
Oracle Coherence
memcached
IBM eXtreme Scale
GridGain
ScaleOut
Vmware GemFire
InfiniSpan
CloudTran

Danksagung

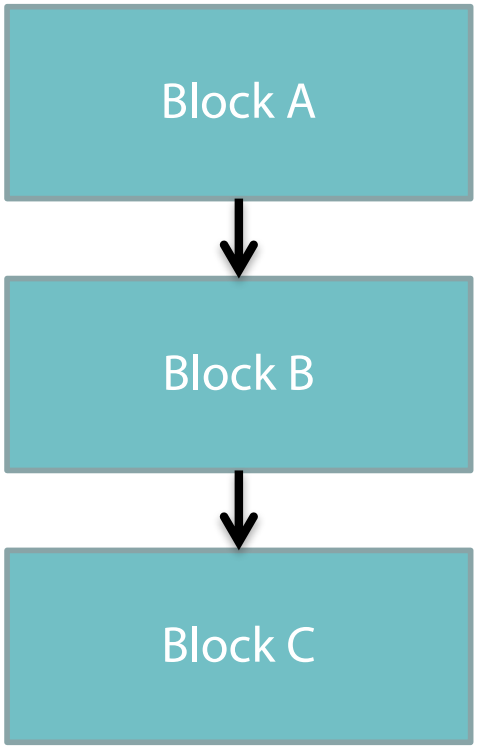
- Blockchain-Präsentationen in Zusammenarbeit mit Dennis Heinrich (IFIS)

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!

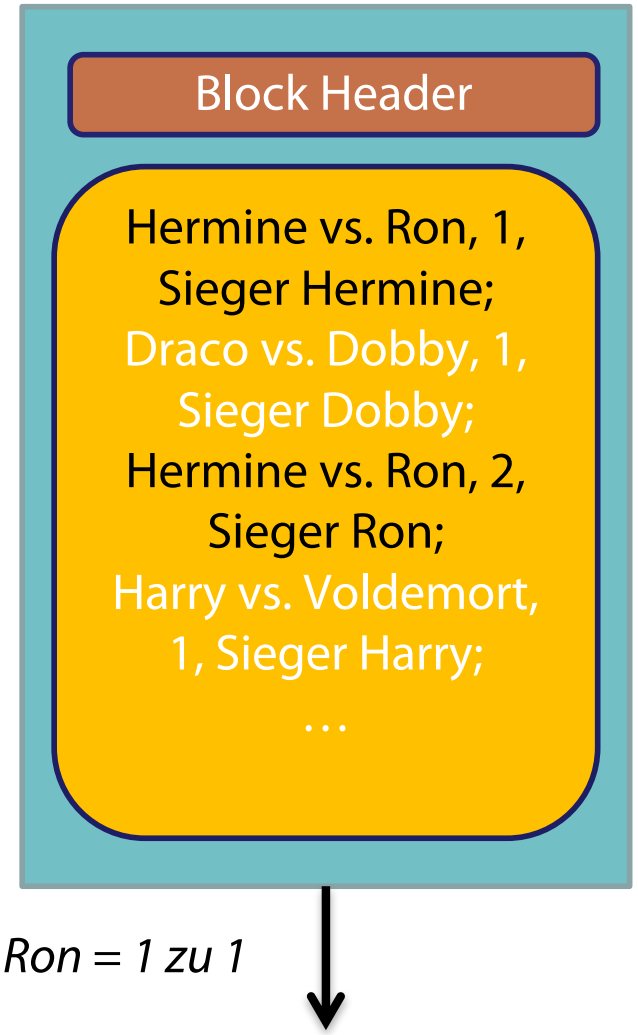
Beispiel Zaubererduell

Was enthält unserer **Block**?

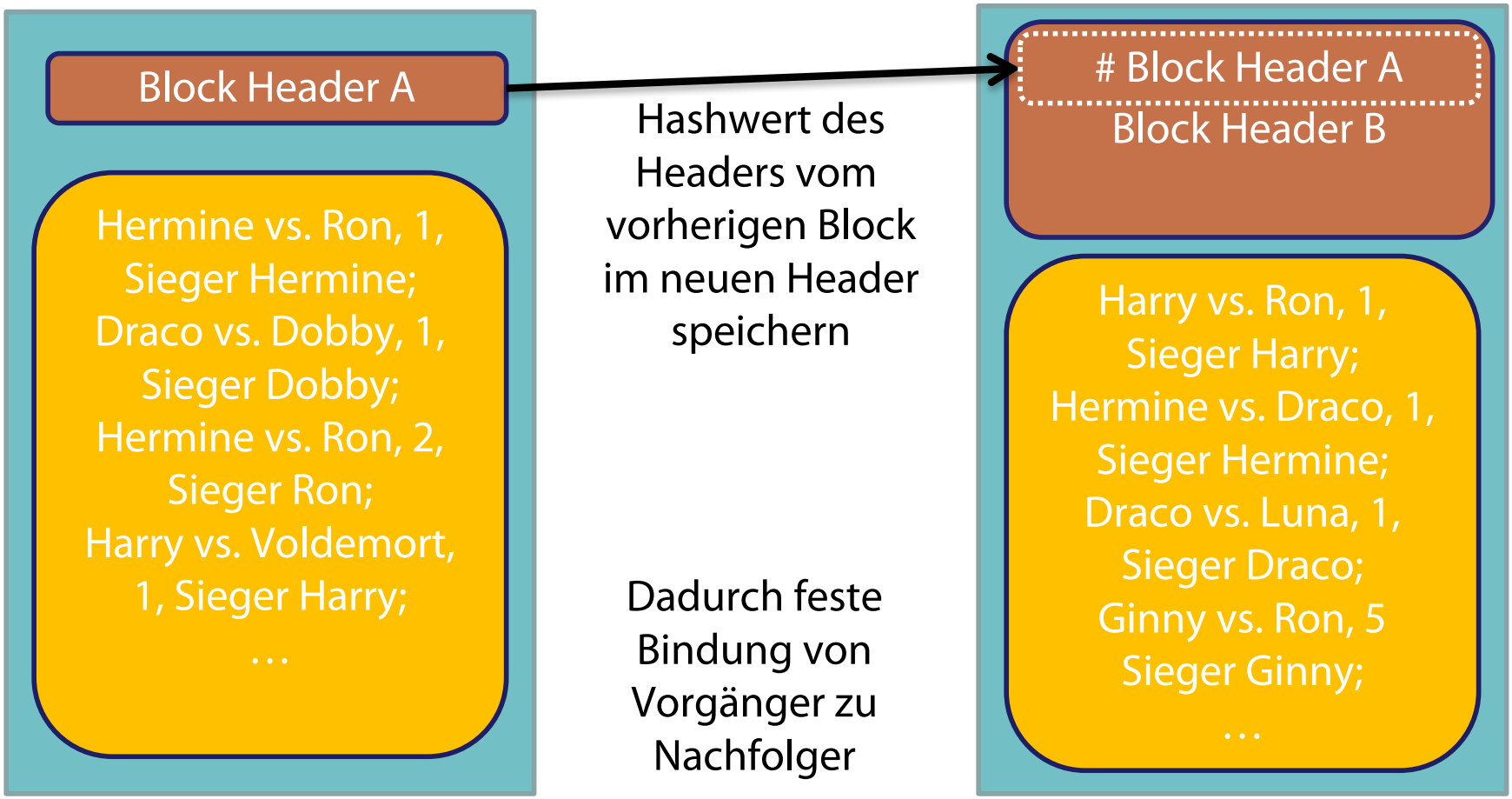
- Duelle zwischen 2 Zaubernden
- Sieger des Duells
- (Duellant A, Duellant B, DuellNr. zwischen Beteiligten, Sieger)

Was können wir damit **nachweisen**?

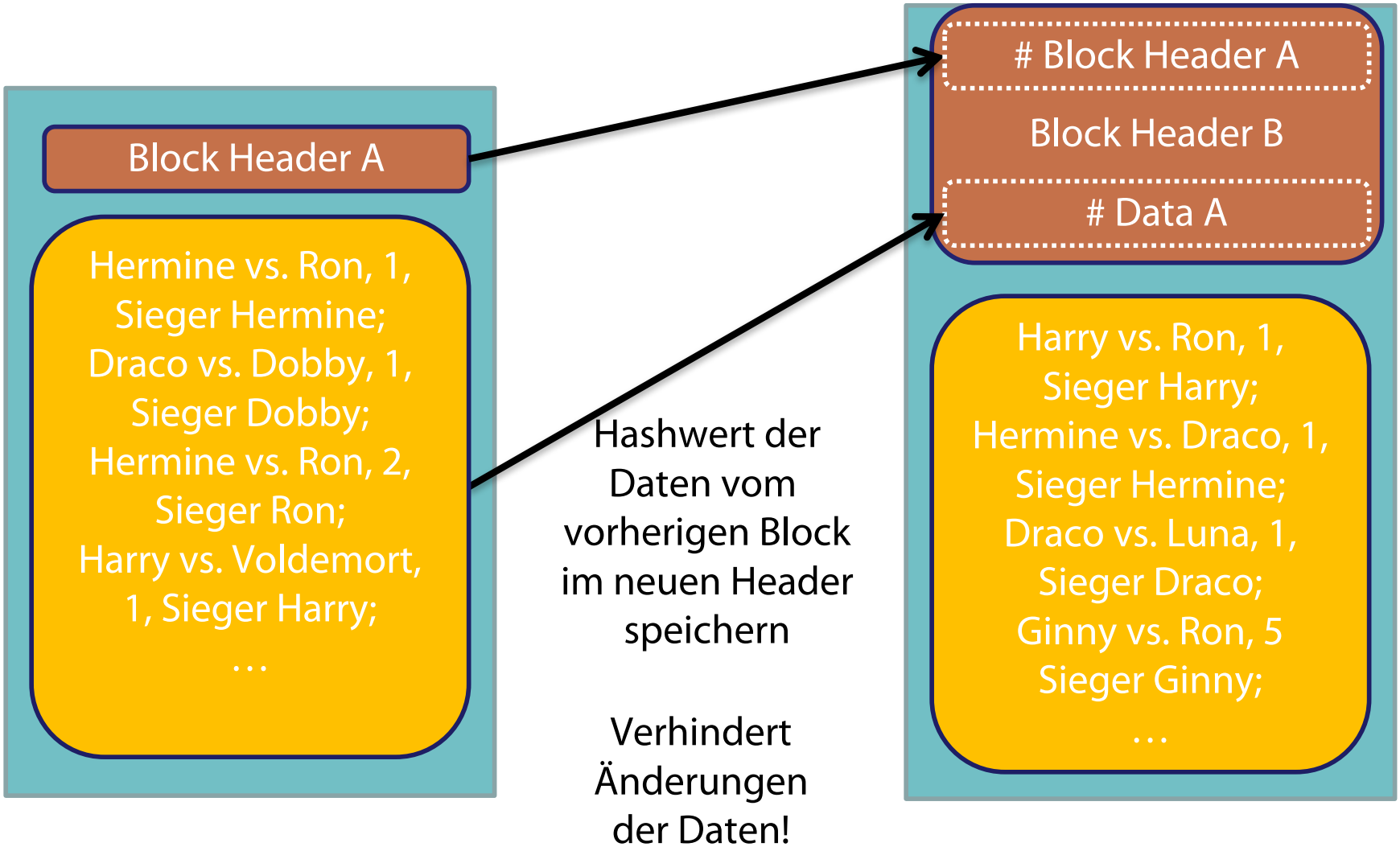
- Wer gegen Wen
- Häufigkeit Sieger/Verlierer über gesamte Kette



Wie sind die Blöcke verbunden?



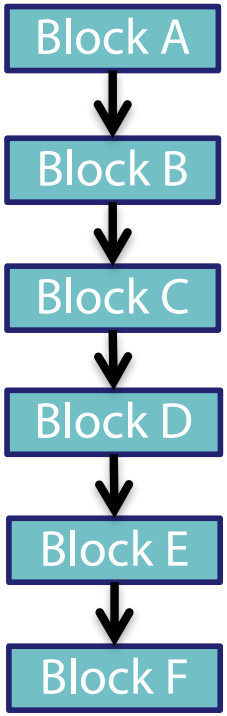
Wie sind die Blöcke verbunden?



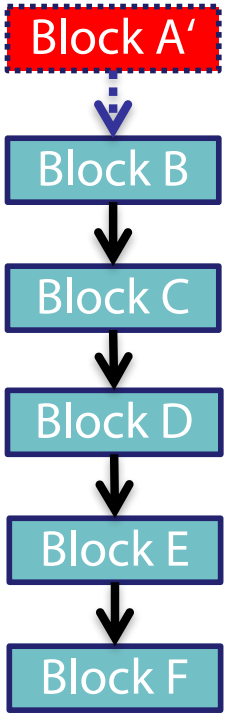
Kann Draco schummeln?

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 **Konsenz** herrschen!

Wir brauchen somit ein dezentrales **Konsenzverfahren**!
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 :*

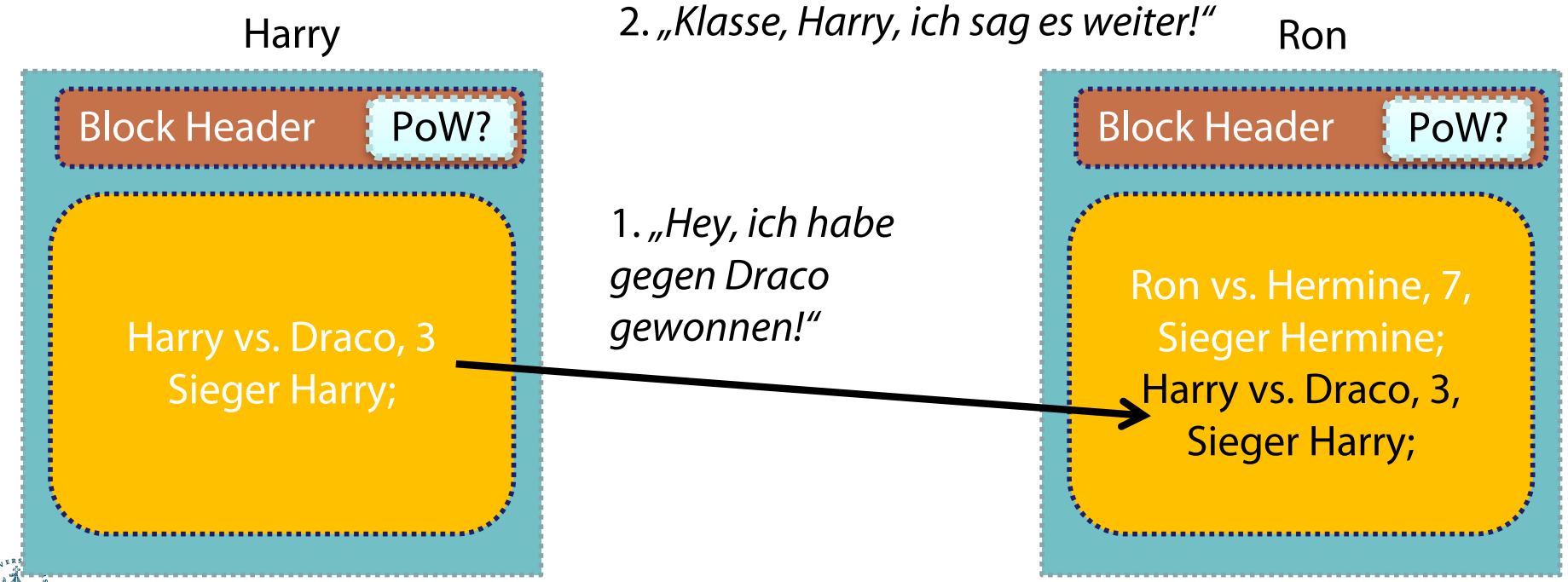
- *Duell gegen sich selbst verboten*
- *Duell mit 2 Siegern/Verlierern (keine doppelte Version eines Duells)*
- *Lücke/Reihenfolge falsch (Duell 3 vor 2, Wo ist Duell 1?)*

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

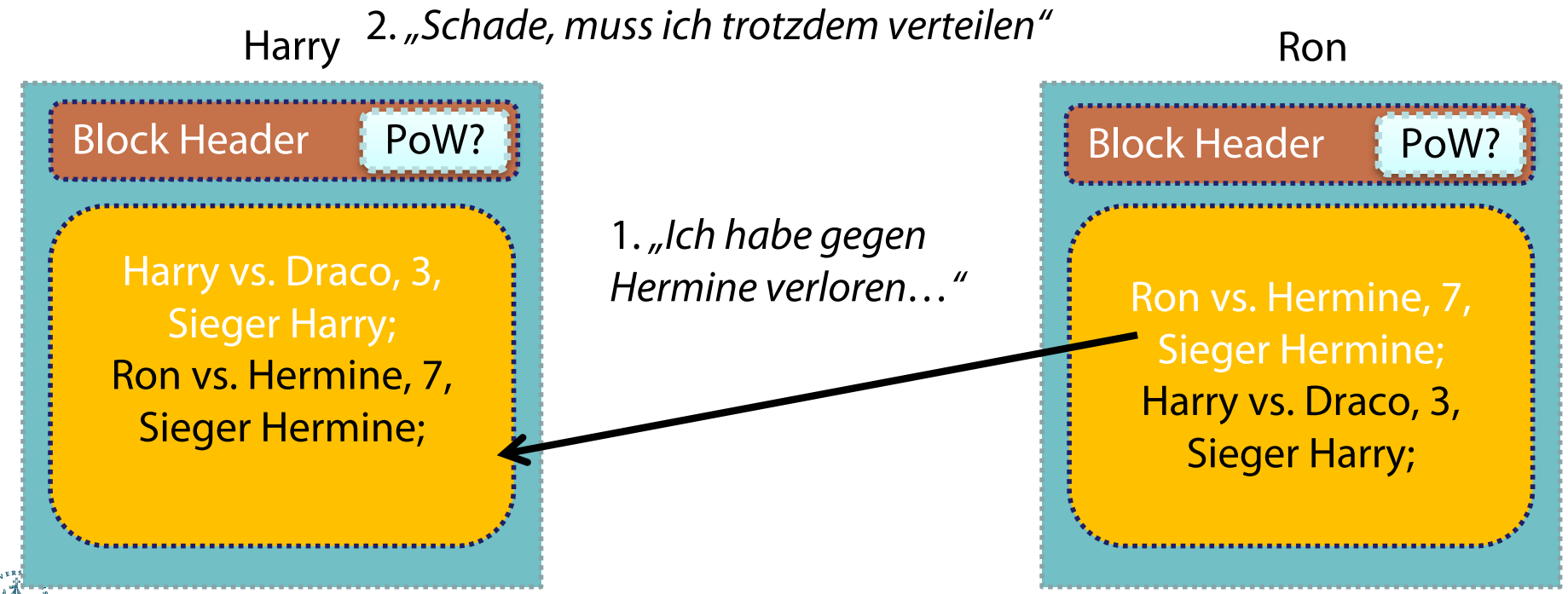


Schritte aus https://bitcoin.org/bitcoin.pdf

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

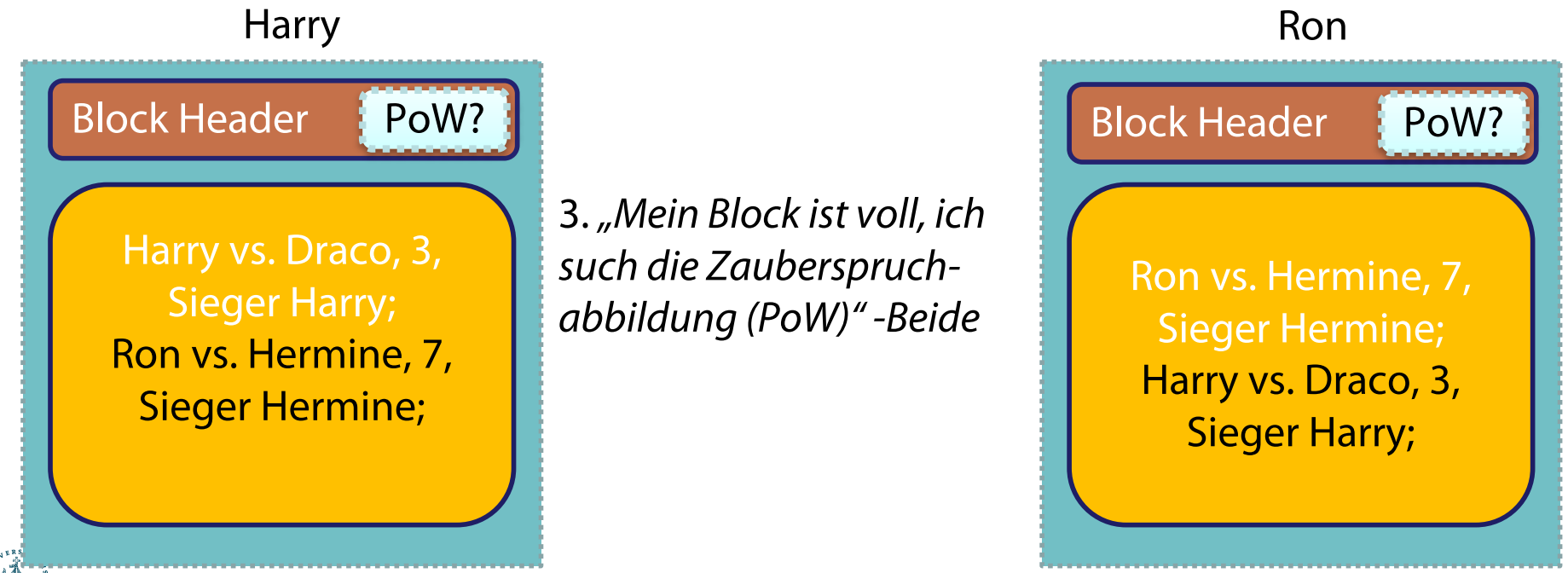
3. Zauberer arbeiten am **Proof of Work**, um ihren Block zu veröffentlichen
4. Falls **Proof of Work** erfüllt, **verteile Block** an alle anderen bekannten Zauberer
5. Andere Zauberer **akzeptieren** den **Block** oder stellen invalide Daten fest
6. **Akzeptanz** wird durch Erstellung eines **Nachfolgeblocks**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



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

Erstellen eines neuen Blocks

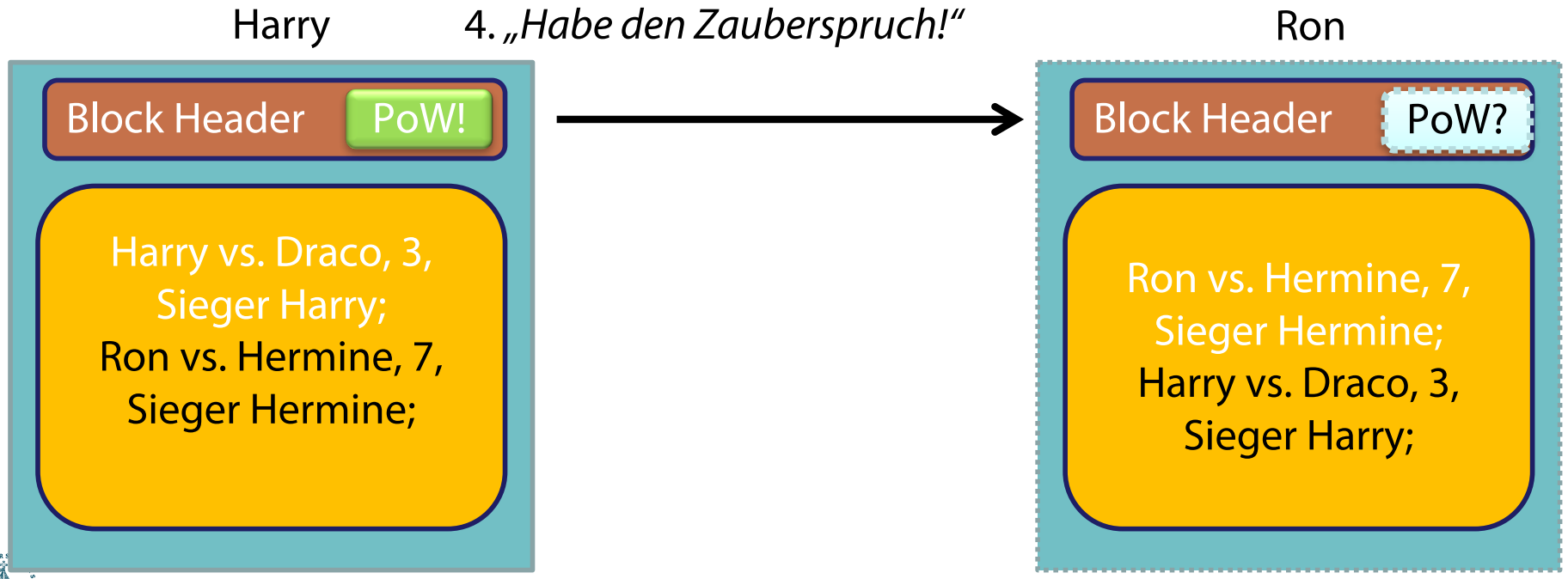
1. **Verteile** alle neuen **Daten (bsp. [Harry, Draco, 3, Harry])** an alle bekannten Zauberer
2. Zauberer sammeln **Daten** in Blöcke und verteilen neue **Daten** weiter
3. Zauberer arbeiten am **Proof of Work**, um ihren Block zu veröffentlichen
4. Falls **Proof of Work** erfüllt, **verteile Block**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

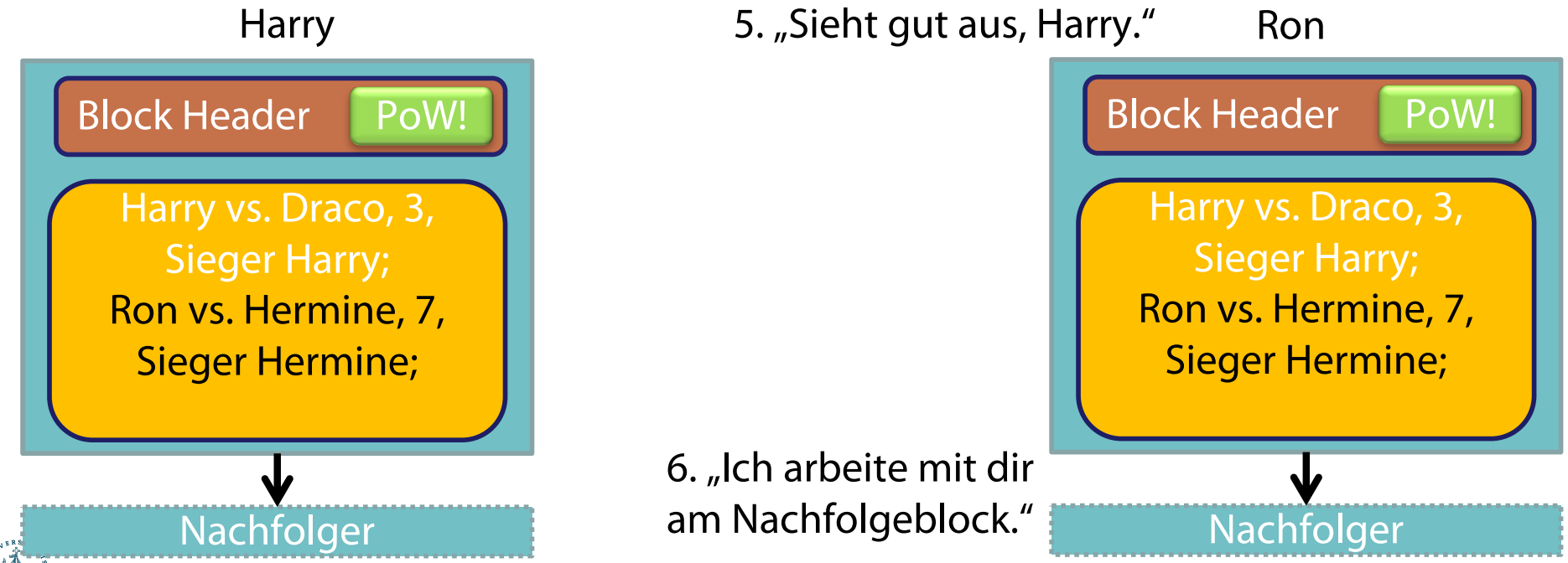


Schritte aus https://bitcoin.org/bitcoin.pdf

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.

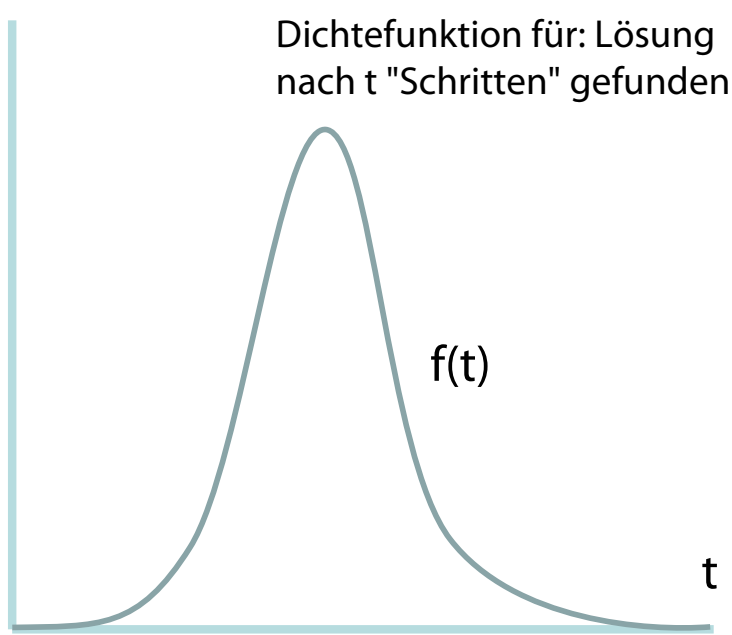
*Hashfunktion(4EF…, 3GH…, **1**) < 100…*

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

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

…

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



Dichtefunktion für: Lösung nach t "Schritten" gefunden

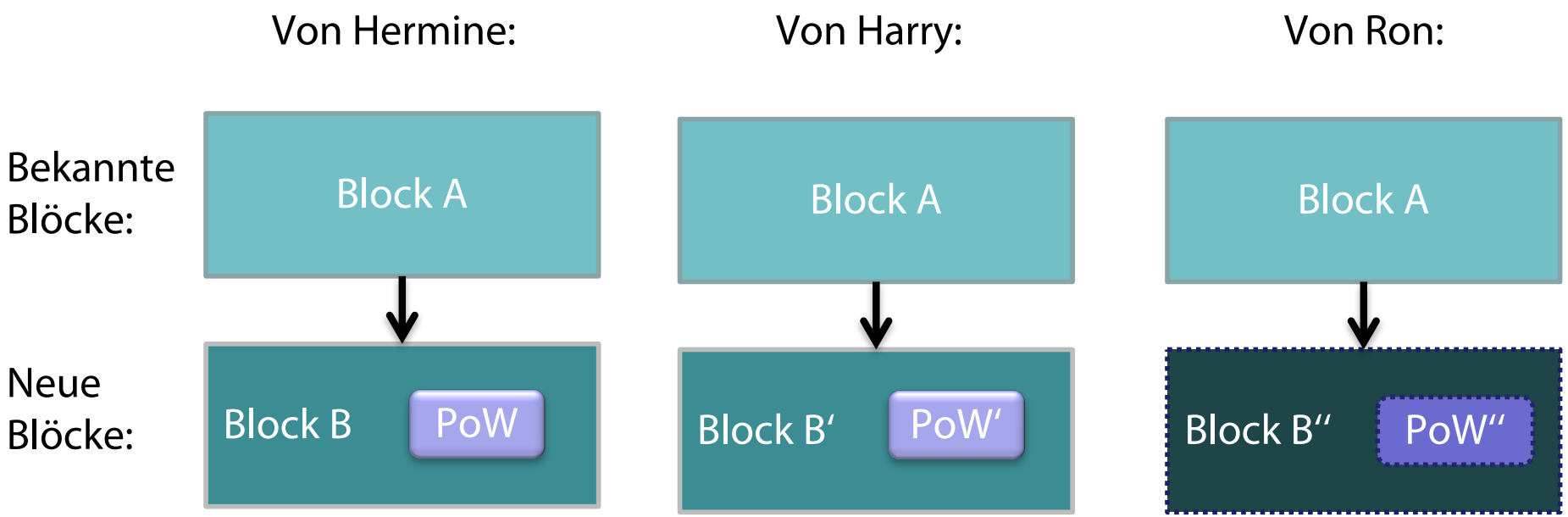
Lösung in bis zu t "Schritten"

$P(T \leq t) = \int_0^t f(t)\, dt$

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?

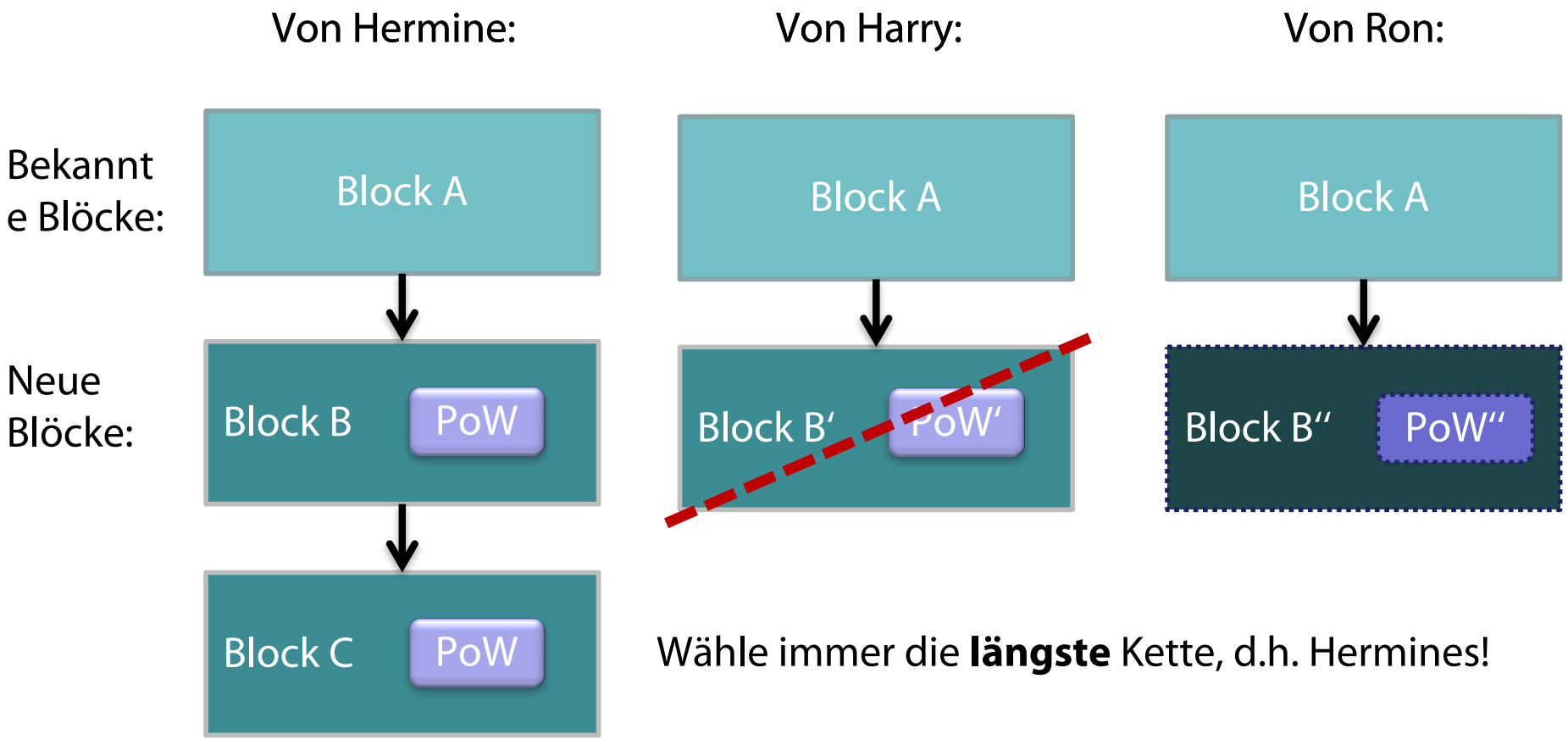


- PoW kann für beide Blöcke getestet werden
- Duelle sind konsistent und Blöcke gefüllt

→ 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.



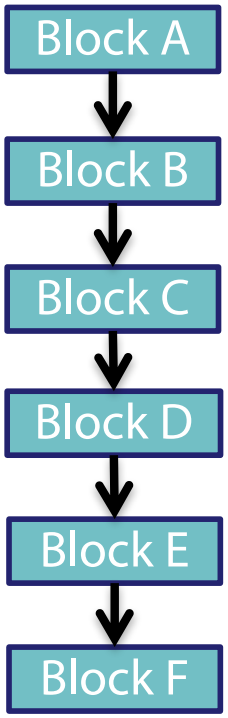
Wähle immer die **längste** Kette, d.h. Hermines!

Kann Draco schummeln? Revisited

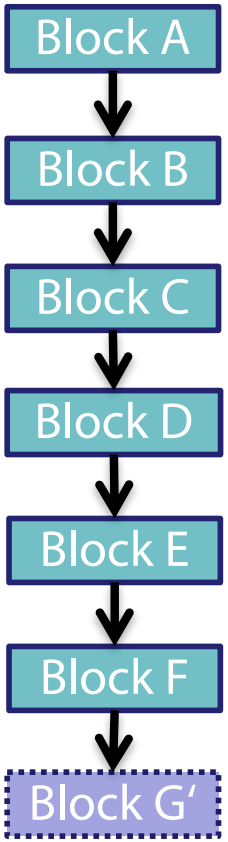
Draco kämpft gegen Harry, Harry gewinnt. Beide tragen Event in letzten Block ein

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

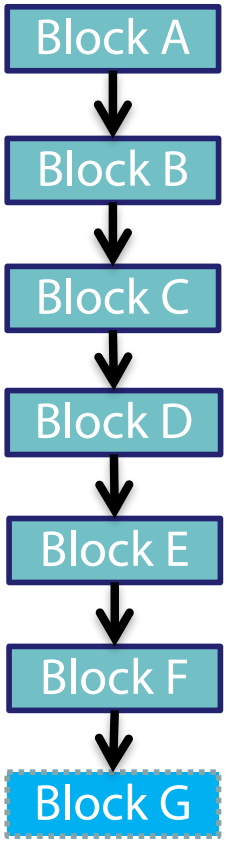
Aktuelle Kette:



Dracos Kette:



Harrys Kette:



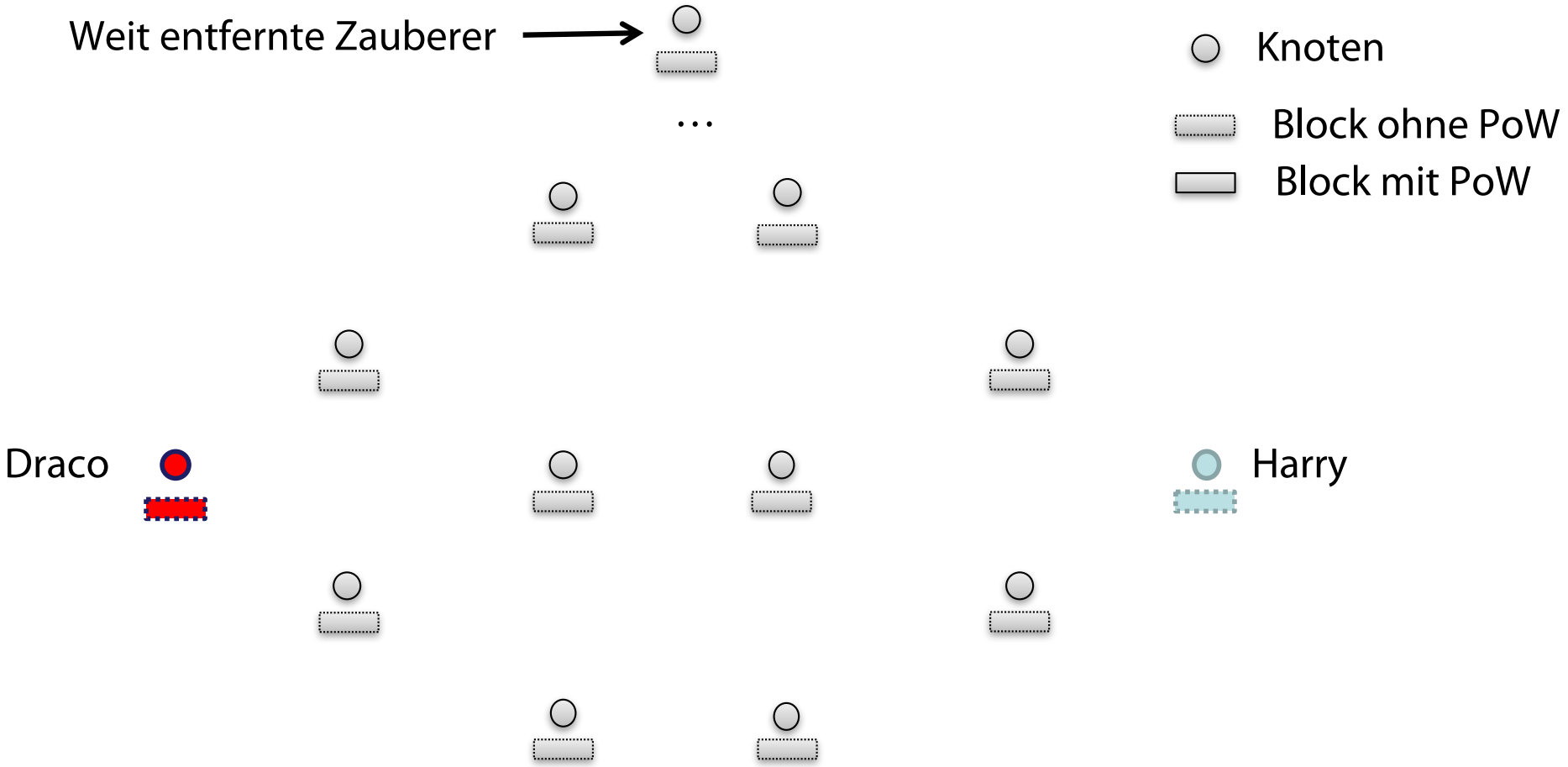
Draco muss erstmal einen PoW leisten, damit er seinen Block verteilen kann.

Draco muss schneller sein als Harry!

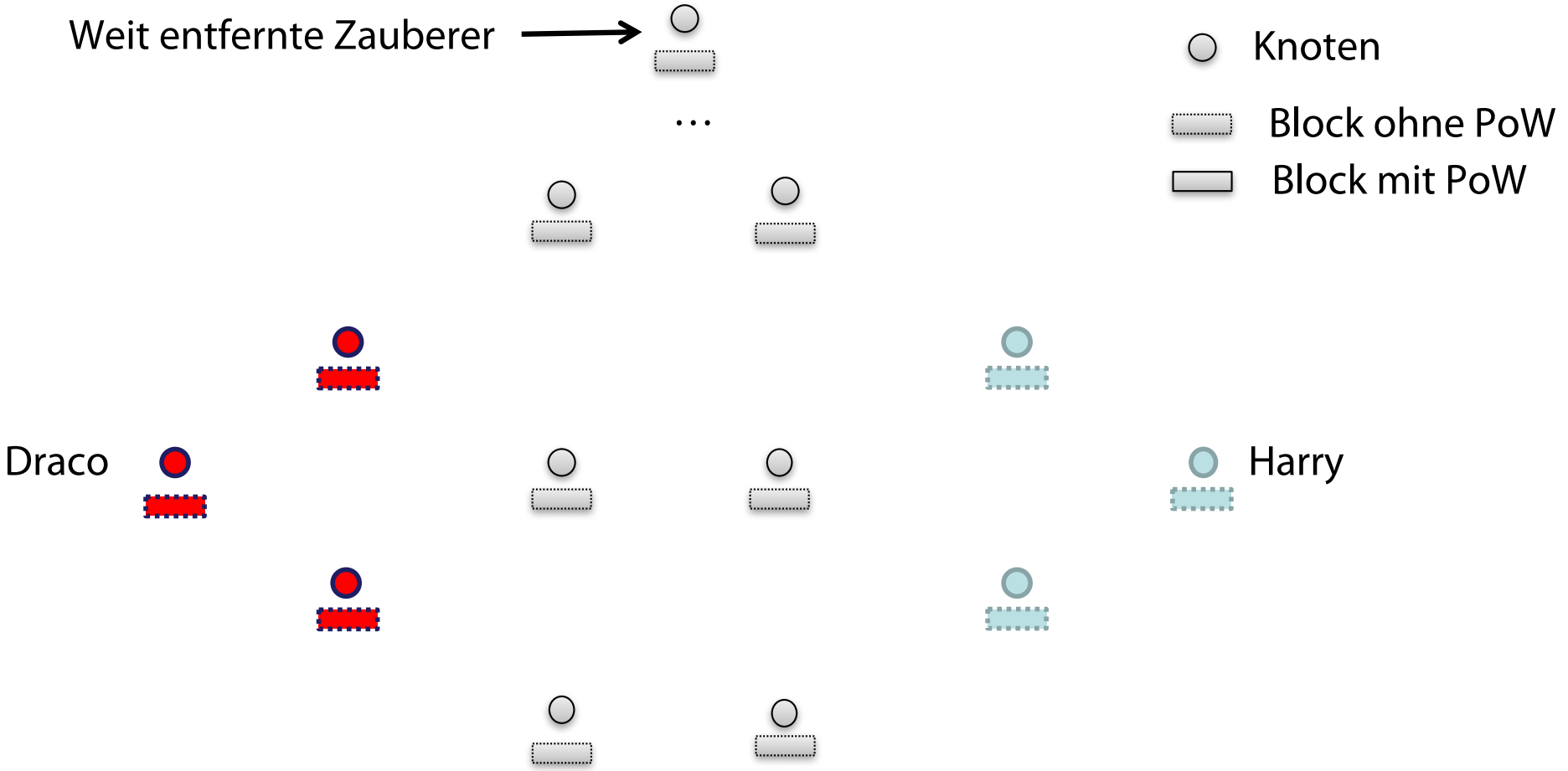
Gewinnchance: 50 zu 50*

*bei gleicher Reichweite im Netzwerk

Kann Draco schummeln? Revisited



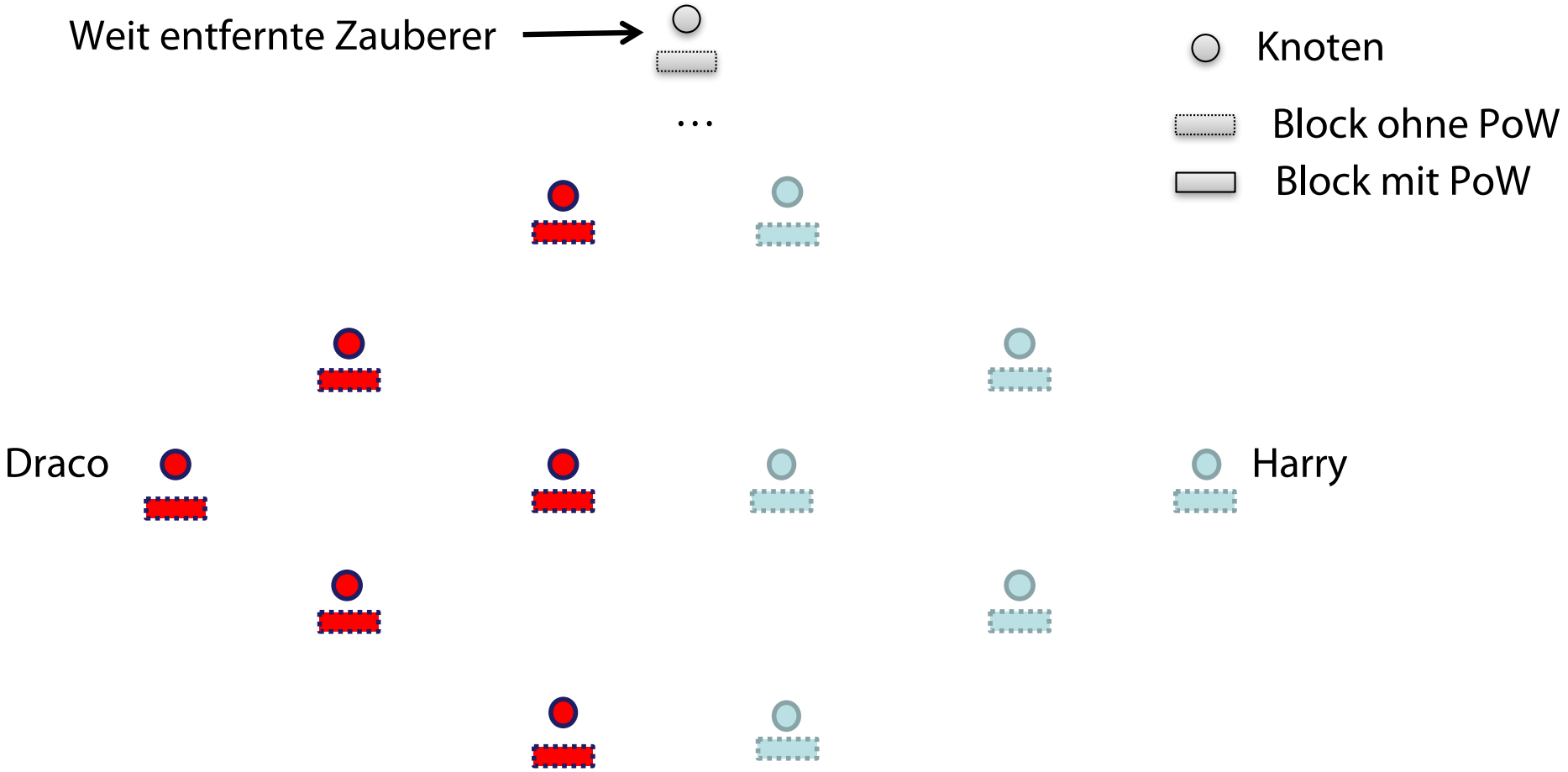
Kann Draco schummeln? Revisited



Draco verbreitet, dass er gewonnen hat

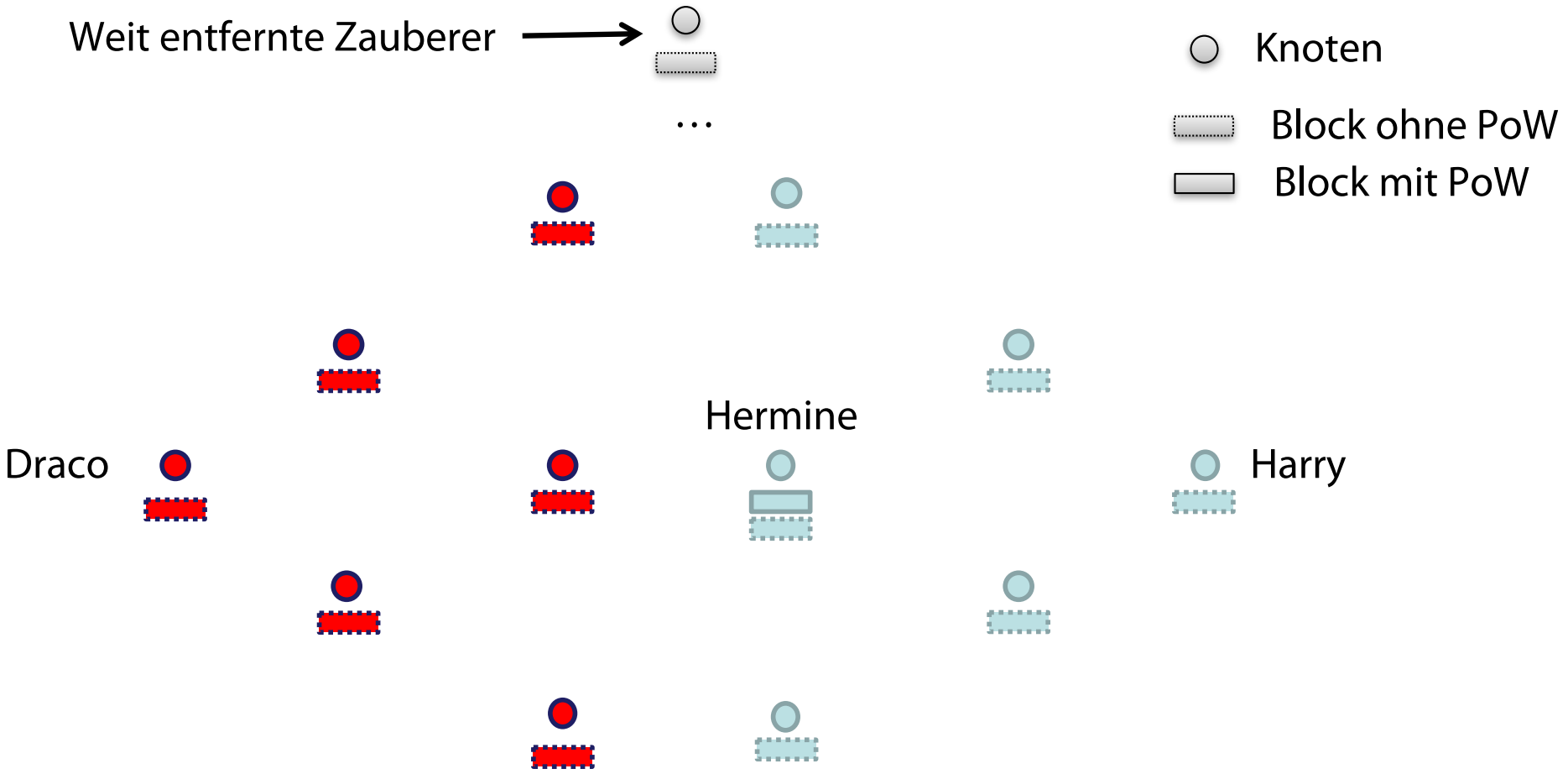
Harry verbreitet, dass er gewonnen hat

Kann Draco schummeln? Revisited



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

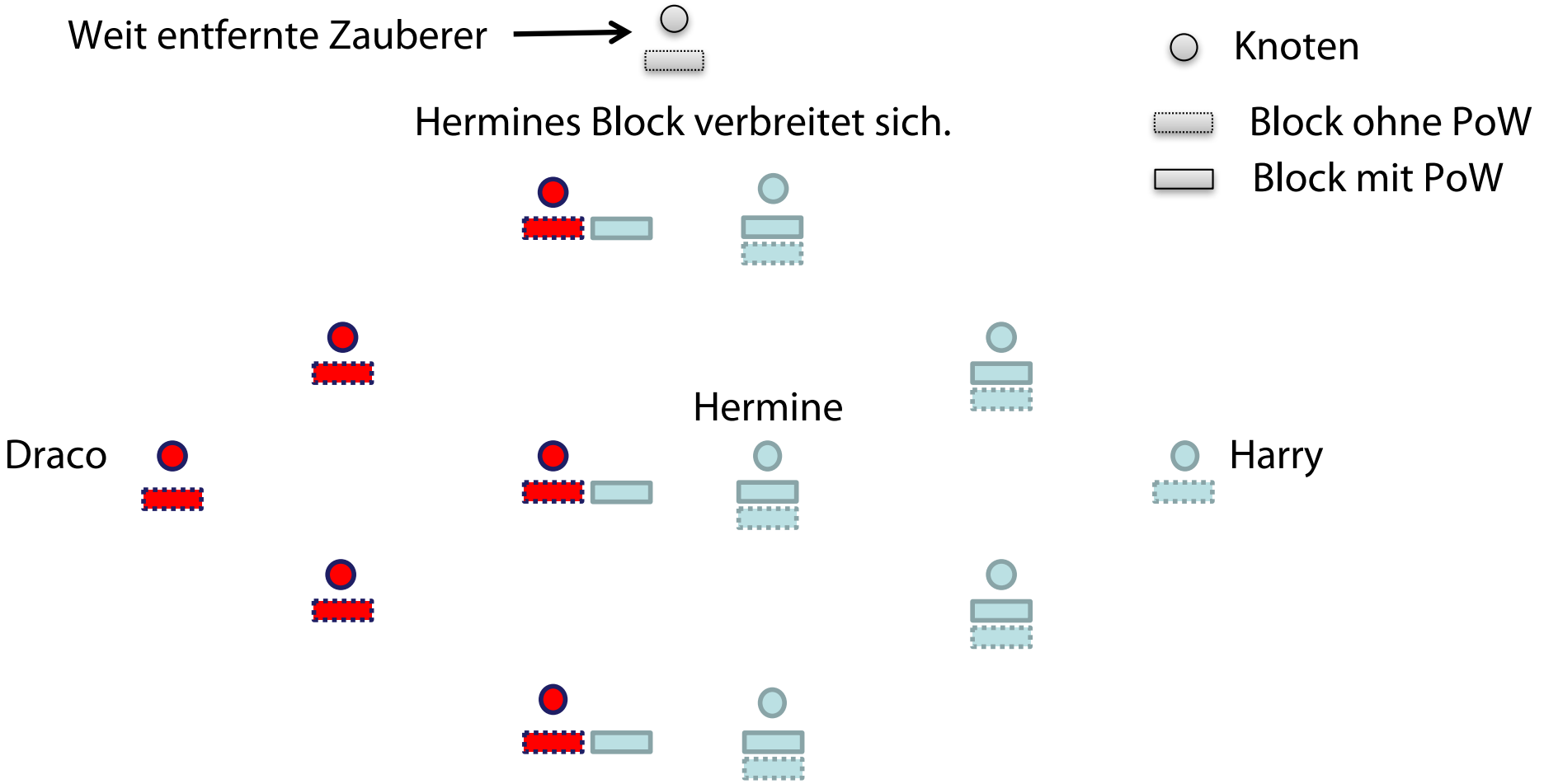
Kann Draco schummeln? Revisited



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

Hermine arbeitet nun an Nachfolgeblock

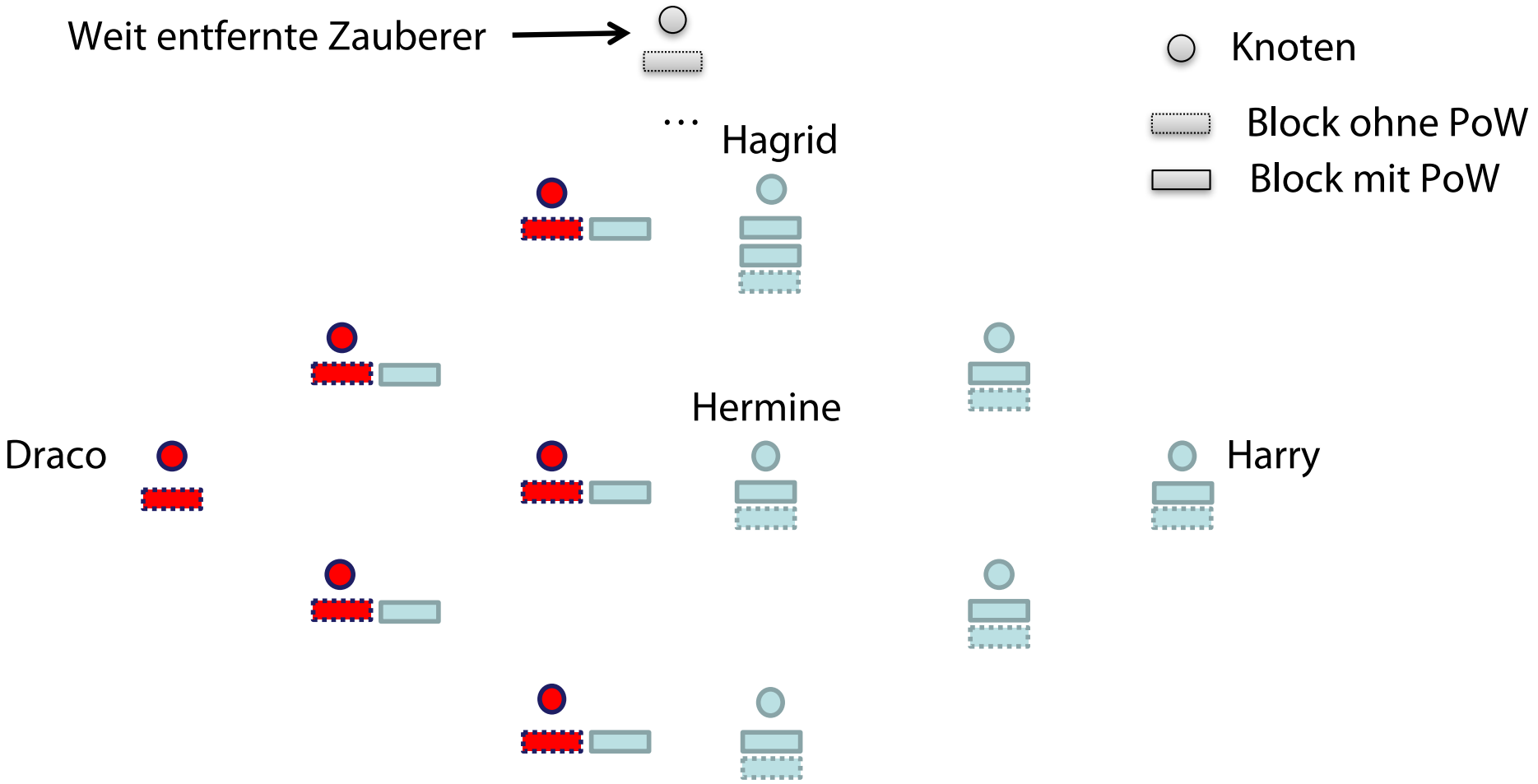
Kann Draco schummeln? Revisited



Gruppe Draco erhält Hermines Block, akzeptiert diesen aber nicht.

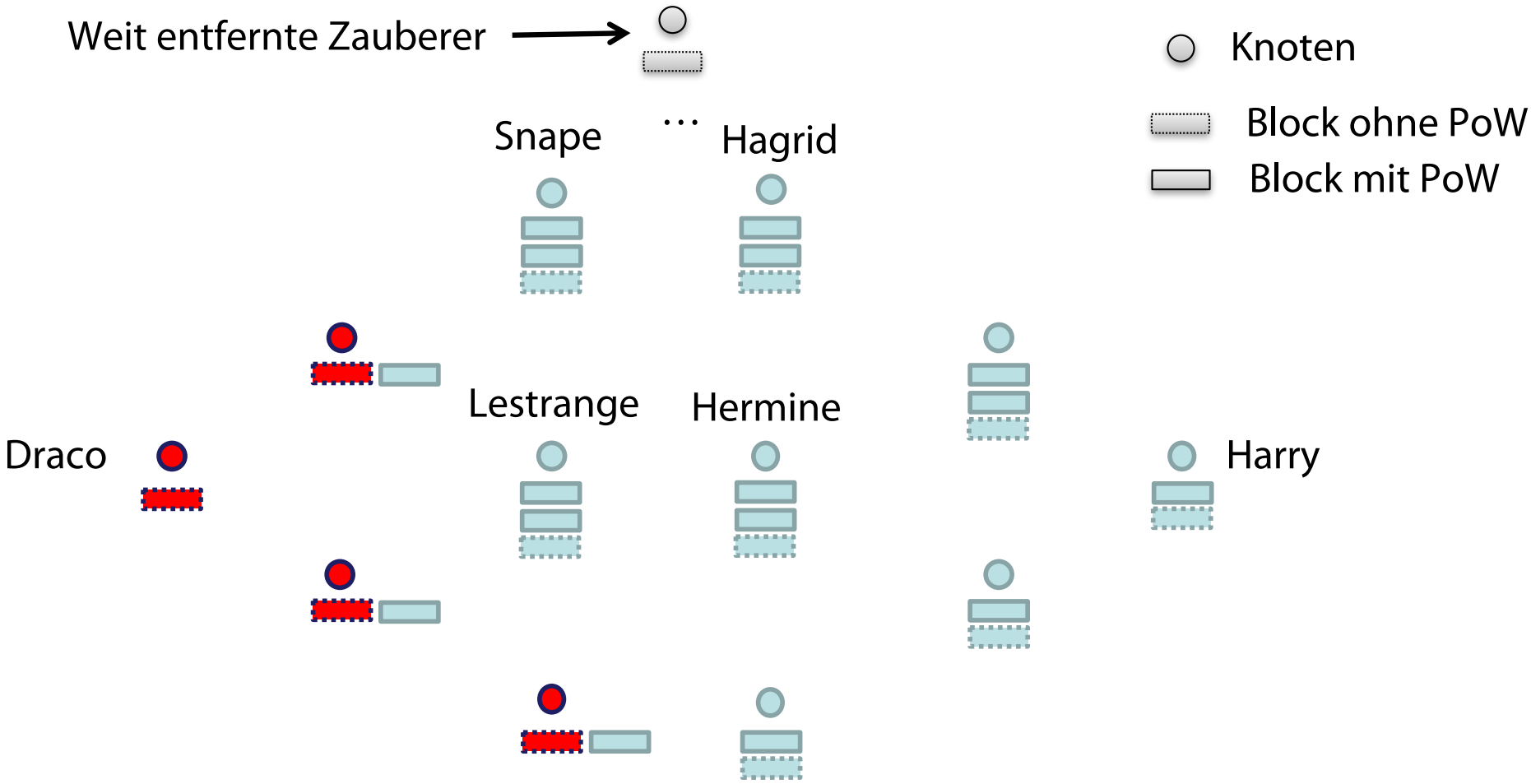
Gruppe Harry arbeitet nun an Nachfolgeblock

Kann Draco schummeln? Revisited



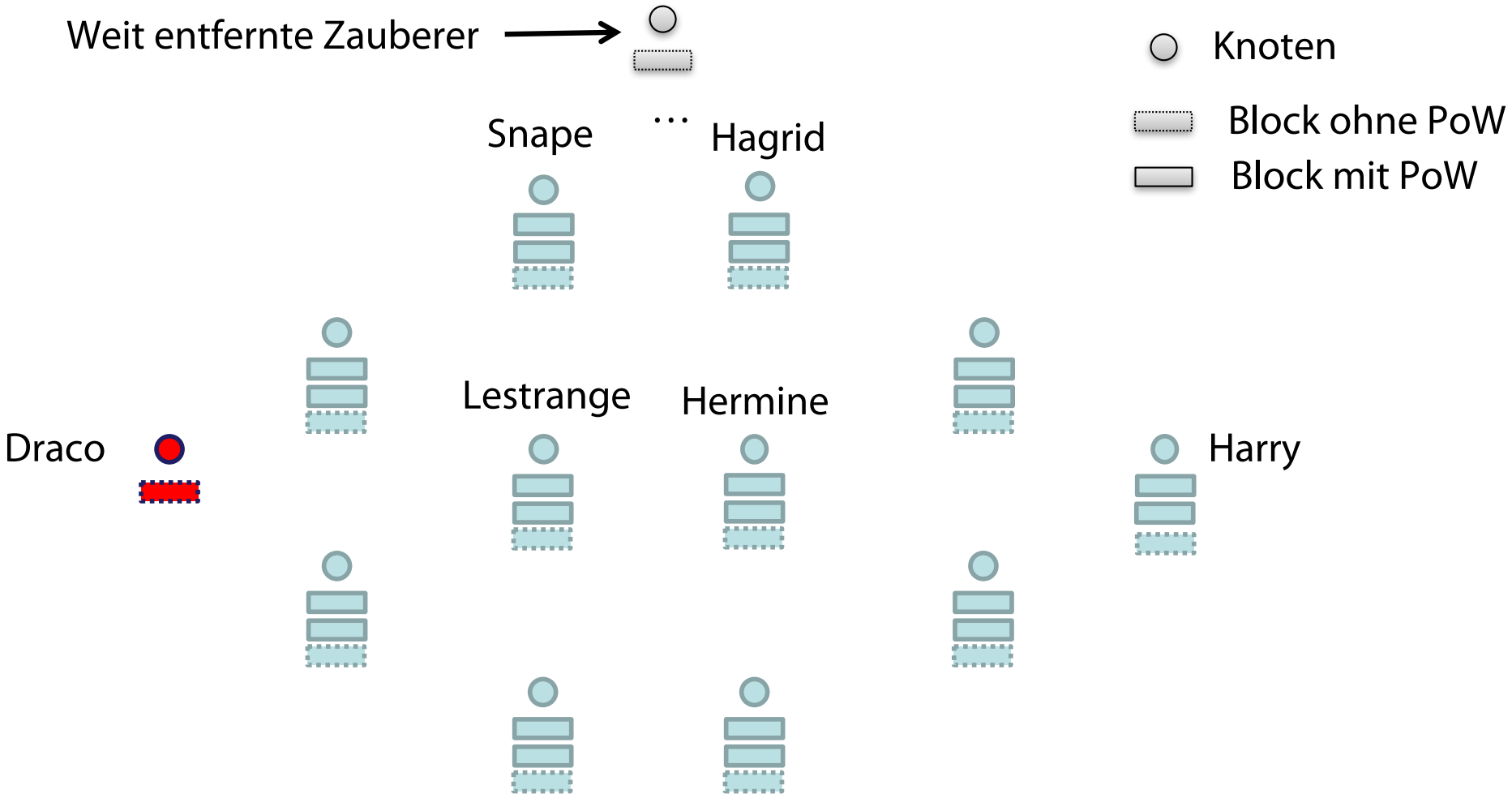
Hagrid erfüllt **Proof of Work** mit einem Block der auf Hermines aufbaut.

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

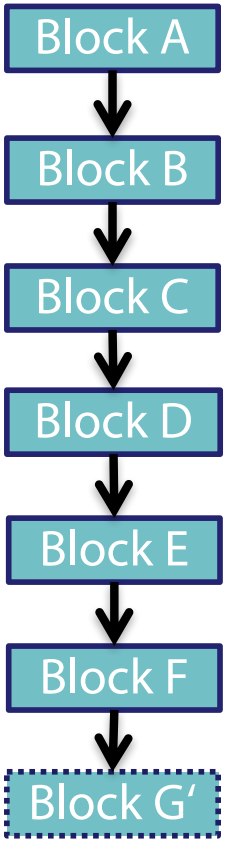


Es sieht schlecht aus für Draco…

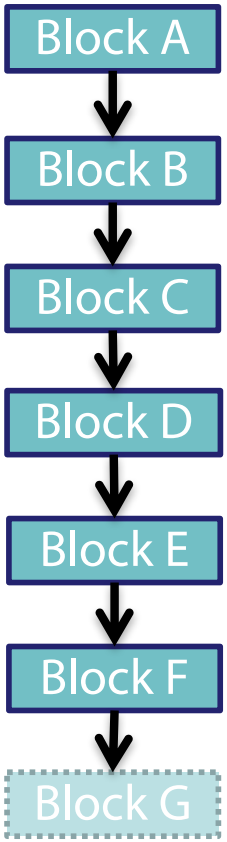
Kann Draco schummeln? Revisited

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

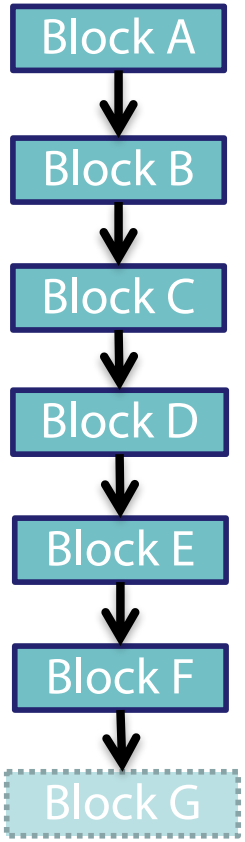
Dracos Kette:



Harrys Kette:



Kette von Dumbledore's Armee:

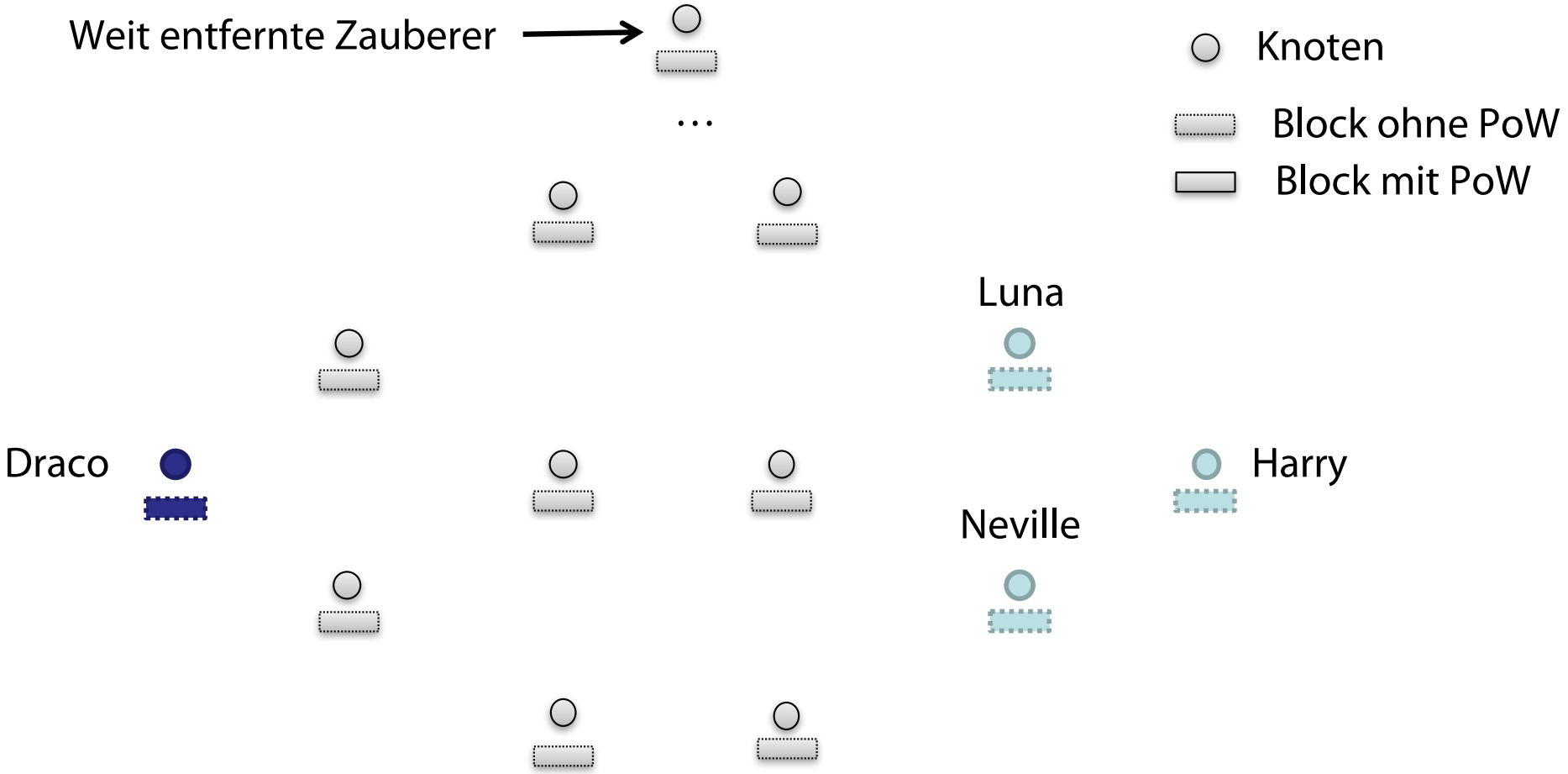


Wahrscheinlichkeit, dass einer der anderen Zauberer oder Harry seine Sicht veröffentlichen kann größer!

Plötzlich Rennen gegen viele andere!

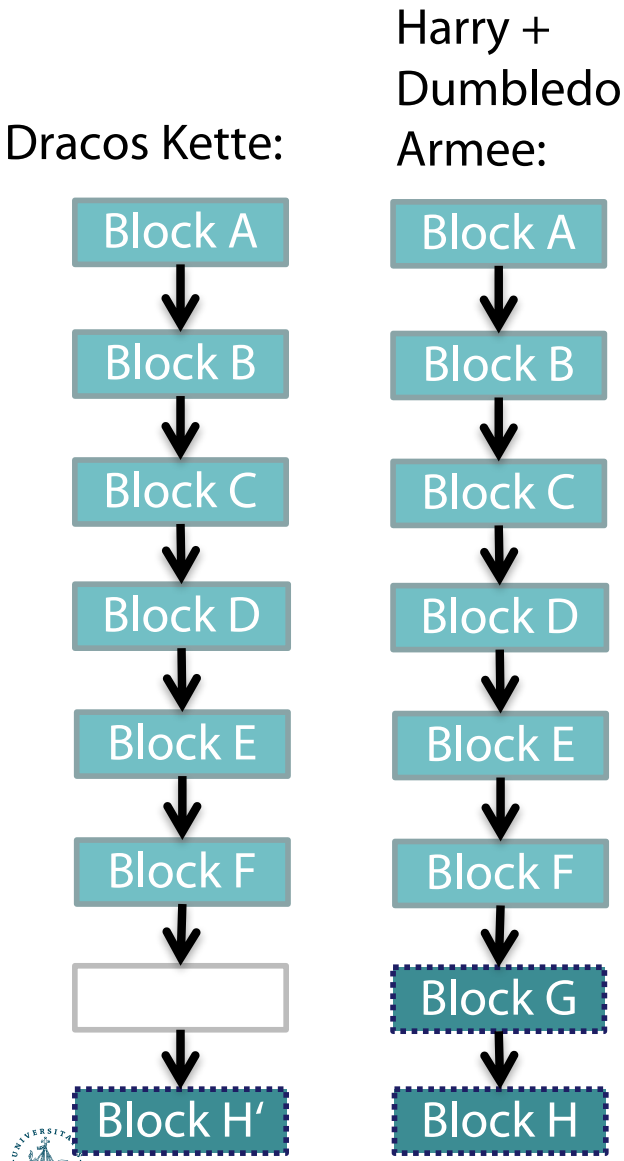
Dracos Chancen, einen neuen Block vor den anderen zu erstellen, sinken mit der Anzahl der anderen Beteiligten.

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.

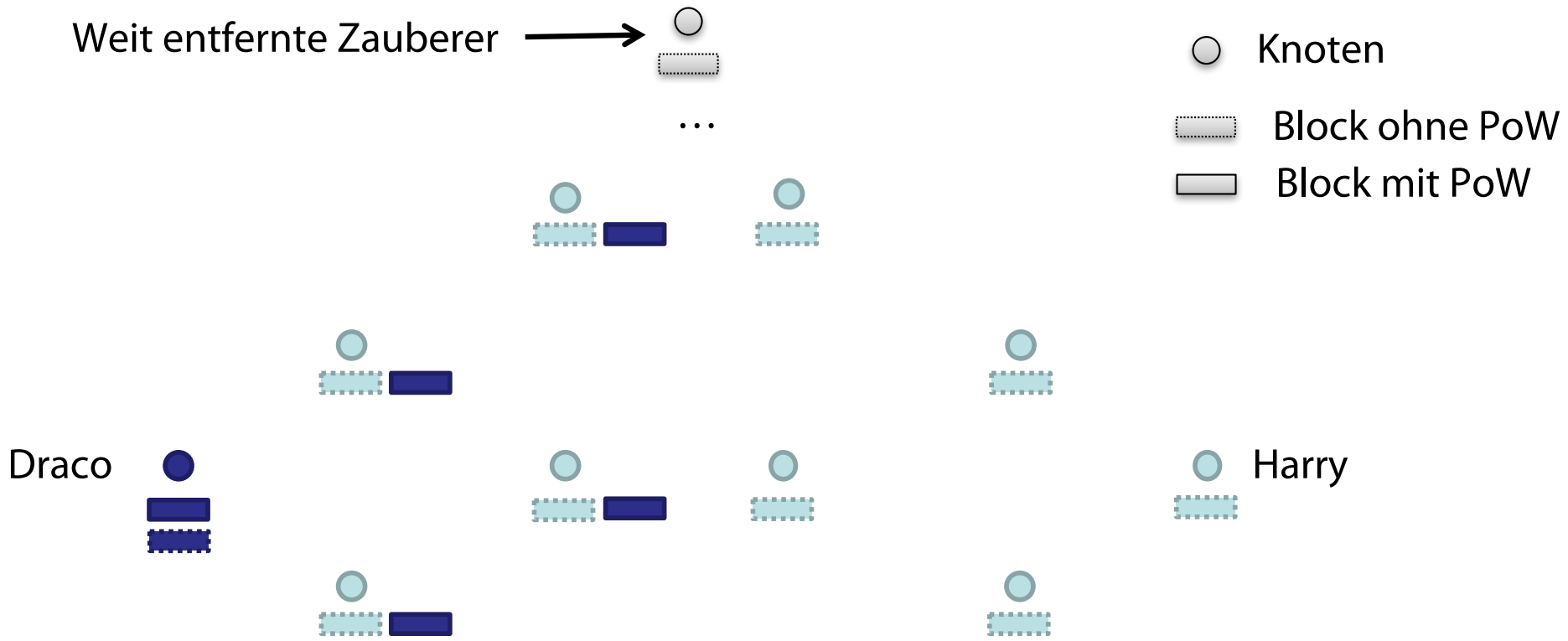
Damit ist der **Block H'** durch den Hash allerdings anders als der von der anderen Gruppe.
Und Draco muss **erneut** seinen **Proof of Work** für diesen Block alleine leisten.

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

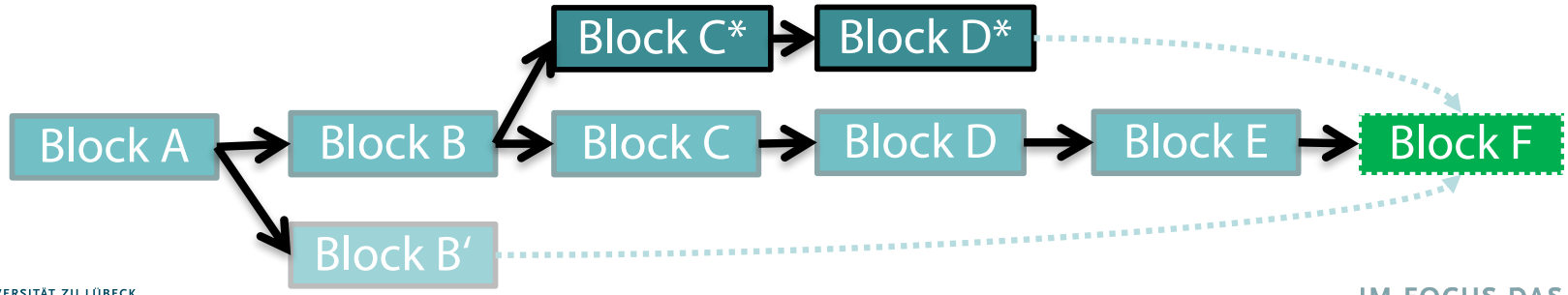
Pro:

- 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

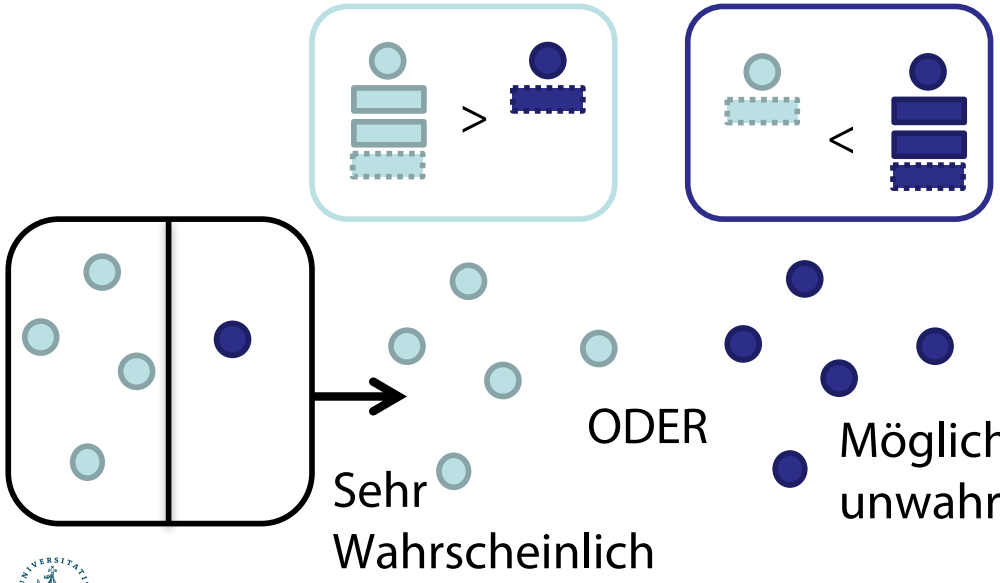
Partitionierungstoleranz

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 Konsenzverfahren 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