The ROAD to AKKA CLUSTER and BEYOND...

JONAS BONÉR
CTO TYPESAFE
@JBONER

What is a DISTRIBUTED SYSTEM?

What is a DISTRIBUTED SYSTEM?

AND WHY WOULD YOU NEED ONE?

DISTRIBUTED COMPUTING IS THE NEW NORMAL

DISTRIBUTED COMPUTING IS THE NEW NORMAL

YOU ALREADY HAVE A DISTRIBUTED SYSTEM, WHETHER YOU WANT IT OR NOT

DISTRIBUTED COMPUTING IS THE NEW NORMAL

YOU ALREADY HAVE A DISTRIBUTED SYSTEM, WHETHER YOU WANT IT OR NOT

Mobile

NOSQL Databases

SQL Replication

Cloud & REST Services

What is the ESSENCE OF DISTRIBUTED COMPUTING?

What is the ESSENCE OF DISTRIBUTED COMPUTING?

It's to try to OVERCOME

1. Information travels at THE SPEED OF LIGHT
2. Independent things FAIL INDEPENDENTLY

WHY DO WE NEED IT?

WHY DO WE NEED IT?

ELASTICITY

When you outgrow the resources of a single node

WHY DO WE NEED IT?

ELASTICITY

When you outgrow the resources of a single node

AVAILABILITY

Providing resilience if one node fails

WHY DO WE NEED IT?

ELASTICITY

When you outgrow the resources of a single node

AVAILABILITY

Providing resilience if one node fails

RICH STATEFUL CLIENTS

So, what's the problem?

So, what's the problem?

IT IS STILL

Very Hard

The network is

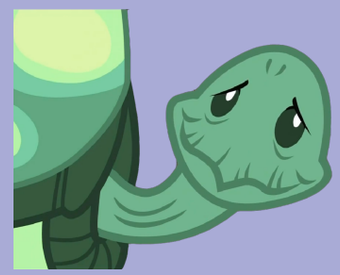
INHERENTLY

UNRELIABLE

YOU CAN'T TELL THE DIFFERENCE

Between a

Slow NODE



and a

Dead NODE

FALLACIES

PETER DEUTSCH'S 8 FALLACIES OF DISTRIBUTED COMPUTING

FALLACIES

PETER DEUTSCH'S 8 FALLACIES OF DISTRIBUTED COMPUTING

1. The network is reliable
2. Latency is zero
3. Bandwidth is infinite
4. The network is secure
5. Topology doesn't change
6. There is one administrator
7. Transport cost is zero
8. The network is homogeneous

So, oh yes . . .

So, oh yes . . .

IT IS STILL

Very Hard

GRAVEYARD OF DISTRIBUTED SYSTEMS

1. Guaranteed Delivery
2. Synchronous RPC
3. Distributed Objects
4. Distributed Shared Mutable State
5. Serializable Distributed Transactions

GENERAL STRATEGIES

DIVIDE & CONQUER

PARTITION

for scale

REPLICATE

for resilience

GENERAL STRATEGIES

WHICH REQUIRES

SHARE NOTHING ASYNCHRONOUS

Designs Message-Passing

GENERAL STRATEGIES

WHICH REQUIRES

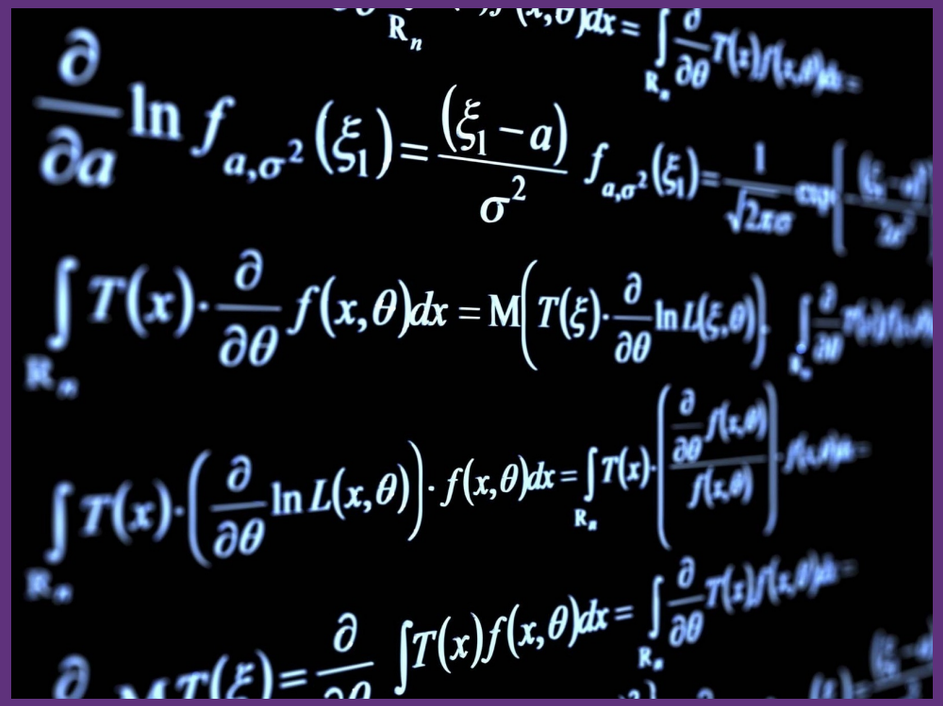
SHARE NOTHING Designs

ASYNCHRONOUS Message-Passing

ISOLATION & Containment

LOCATION Transparency

THEORETICAL
Models



A MODEL FOR DISTRIBUTED COMPUTATION

SHOULD ALLOW EXPLICIT REASONING ABOUT

1. Concurrency
2. Distribution
3. Mobility

CARLOS VARELA 2013

LAMBDA CALCULUS

ALONZO CHURCH 1930

LAMBDA CALCULUS

ALONZO CHURCH 1930

STATE

Immutable state

Managed through functional application

Referential transparent

LAMBDA CALCULUS

ALONZO CHURCH 1930

ORDER

β-reduction—can be performed in ANY order

Normal order
Applicative order
Call-by-name order
Call-by-value order
Call-by-need order

STATE

Immutable state

Managed through functional application

Referential transparent

LAMBDA CALCULUS

ALONZO CHURCH 1930

ORDER

β-reduction—can be performed in ANY order

Even in parallel

Normal order

Applicative order

Call-by-name order

Call-by-value order

Call-by-need order

STATE

Immutable state

Managed through functional application

Referential transparent

LAMBDA CALCULUS

ALONZO CHURCH 1930

ORDER	STATE
β-red	nmutable state
	naged through
	unctional application
Eval	Referential transparent
ative order	
Call-by-name order	
Call-by-value order	
Call-by-need order	

LAMBDA CALCULUS

ALONZO CHURCH 1930

ORDER STATE

β-red… …mutable state

E… …nt

Call-by-name order

Call-by-value order

Call-by-need order

Suppor fa…

No model for Concurr' Distribution

LAMBDA CALCULUS

ALONZO CHURCH 1930

No model for

State

Mobility

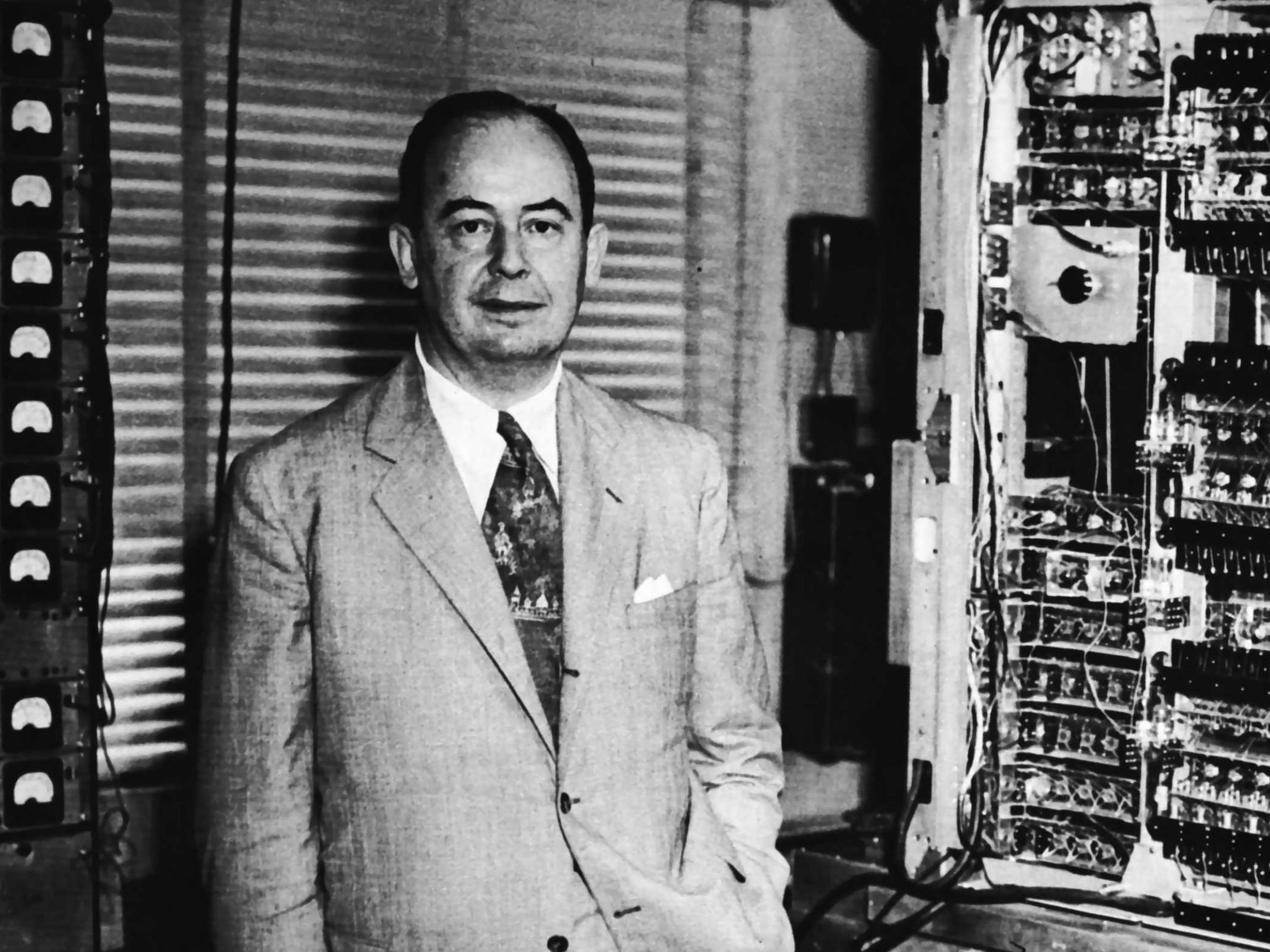
Concurr' Distribution

β-red

Call-by-name order

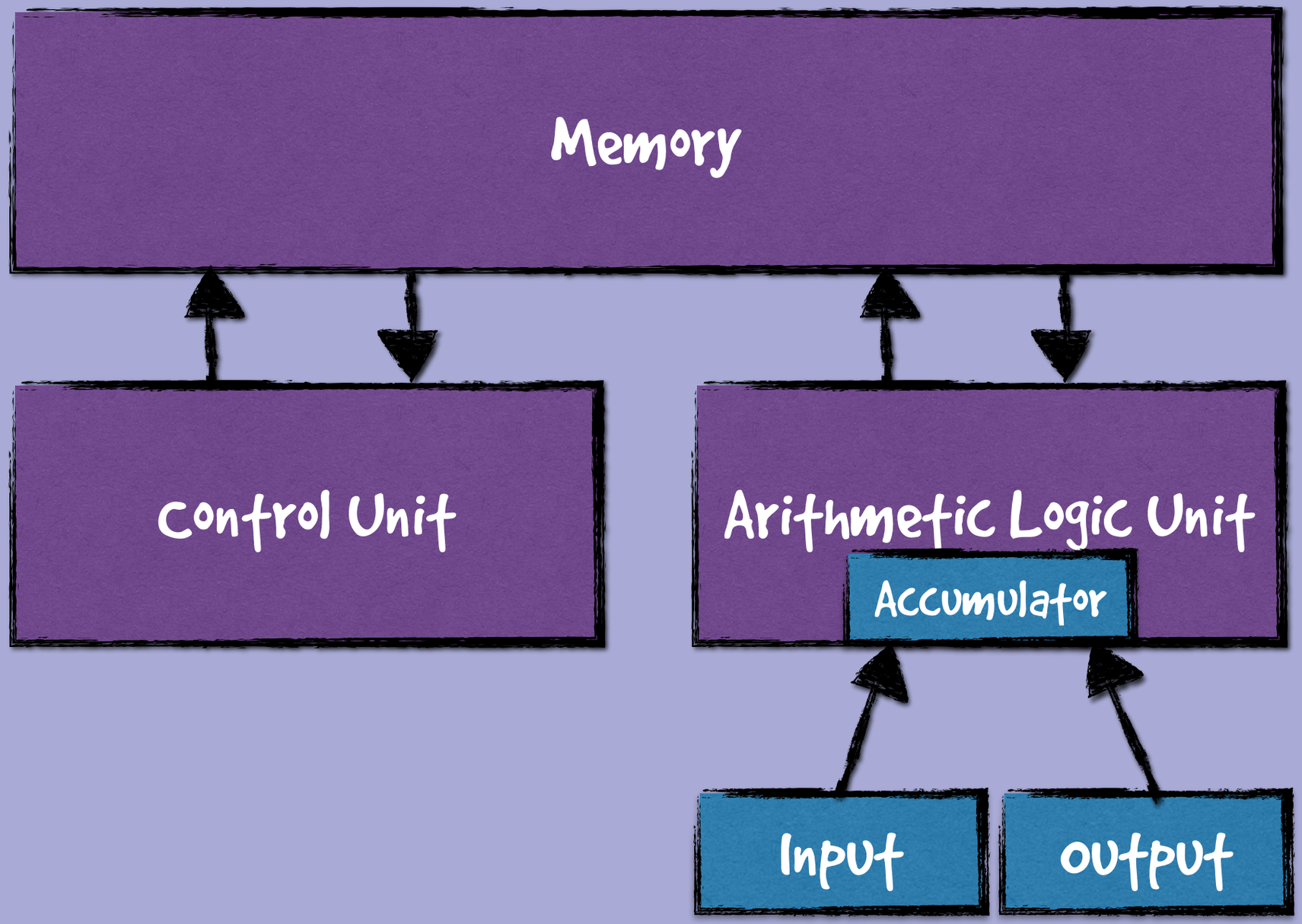
Call-by-value order

Call-by-need order



VON NEUMANN MACHINE

JOHN VON NEUMANN 1945



VON NEUMANN MACHINE

JOHN VON NEUMANN 1945

VON NEUMANN MACHINE

JOHN VON NEUMANN 1945

STATE

Mutable state

In-place updates

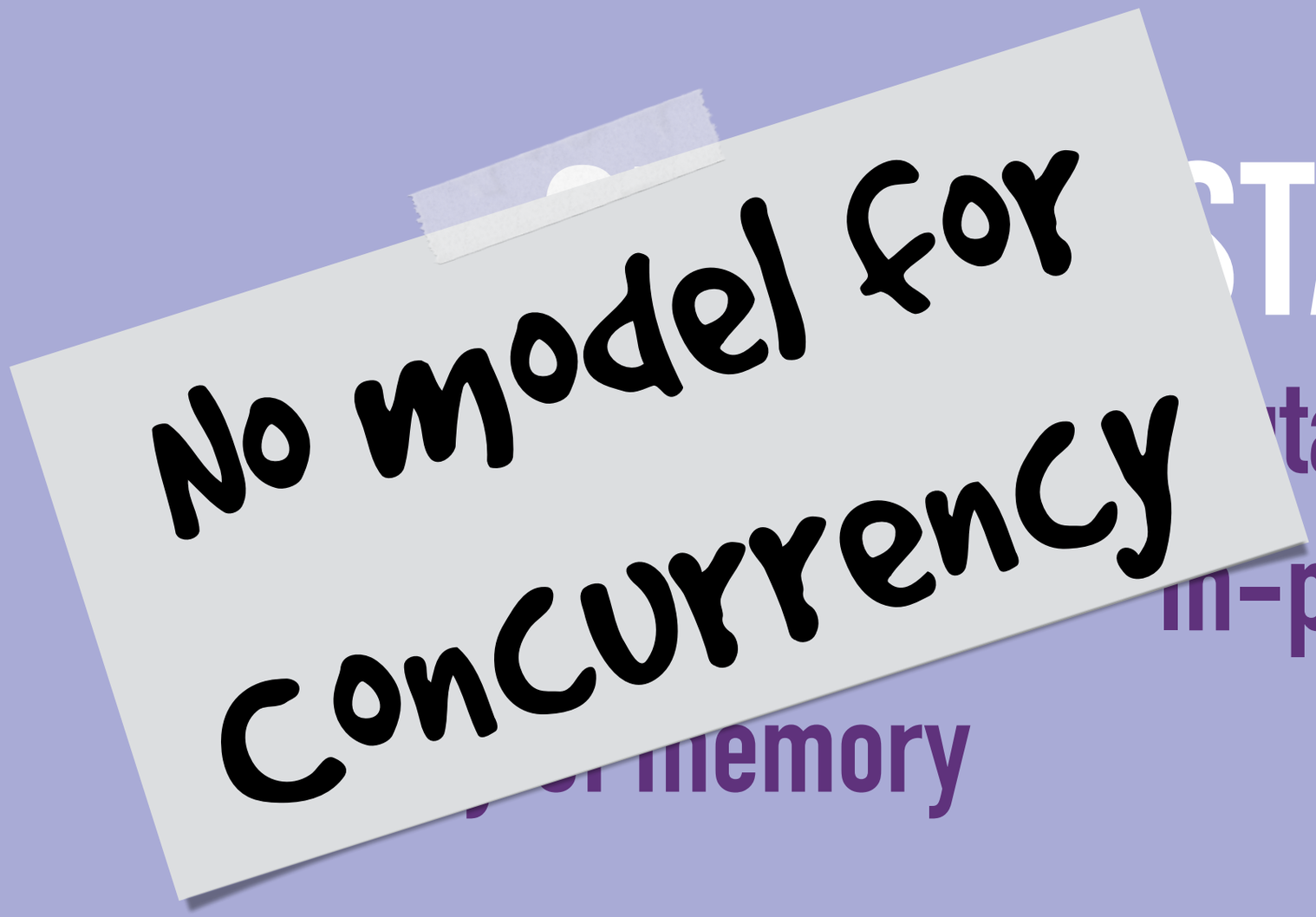
VON NEUMANN MACHINE

JOHN VON NEUMANN 1945

ORDER	STATE
Total order	Mutable state
List of instructions	In-place updates
Array of memory	

VON NEUMANN MACHINE

JOHN VON NEUMANN 1945



STATE

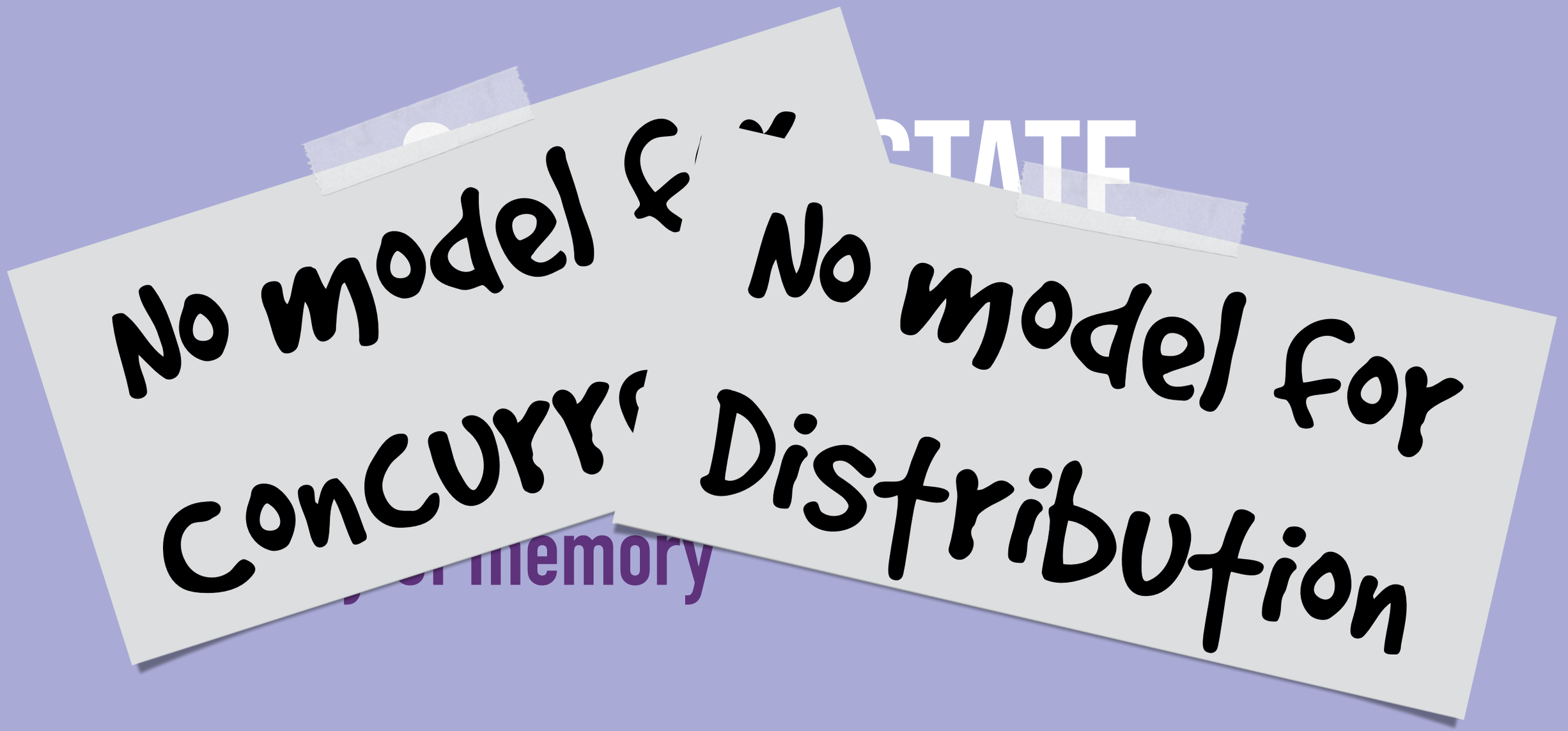
table state

in-place updates

memory

VON NEUMANN MACHINE

JOHN VON NEUMANN 1945



VON NEUMANN MACHINE

JOHN VON NEUMANN 1945

No model for Mobility

Concurr' Distribution

:l for

memory


HENRY LOVD

TRANSACTIONS

JIM GRAY 1981

TRANSACTIONS

JIM GRAY 1981

STATE

Isolation of updates

Atomicity

TRANSACTIONS

JIM GRAY 1981

ORDER	STATE
Serializability	Isolation of updates
Disorder across transactions	Atomicity
Illusion of order within transactions	

TRANSACTIONS

JIM GRAY 1981

ATE

ion of updates

ity

Illusi

transactions

TRANSACTIONS

JIM GRAY 1981

Concurr Distribution

Wo

Does Not

Wor Work Well

Illusi

transactions

ACTORS

CARL HEWITT 1973

ACTORS

CARL HEWITT 1973

STATE

Share nothing

Atomicity within the actor

ACTORS

CARL HEWITT 1973

ORDER

Async message passing

Non-determinism in message delivery

STATE

Share nothing

Atomicity within the actor

ACTORS

CARL HEWITT 1973

Great model for concurrency

micity within the actor

ACTORS

CARL HEWITT 1973

ACTORS

CARL HEWITT 1973

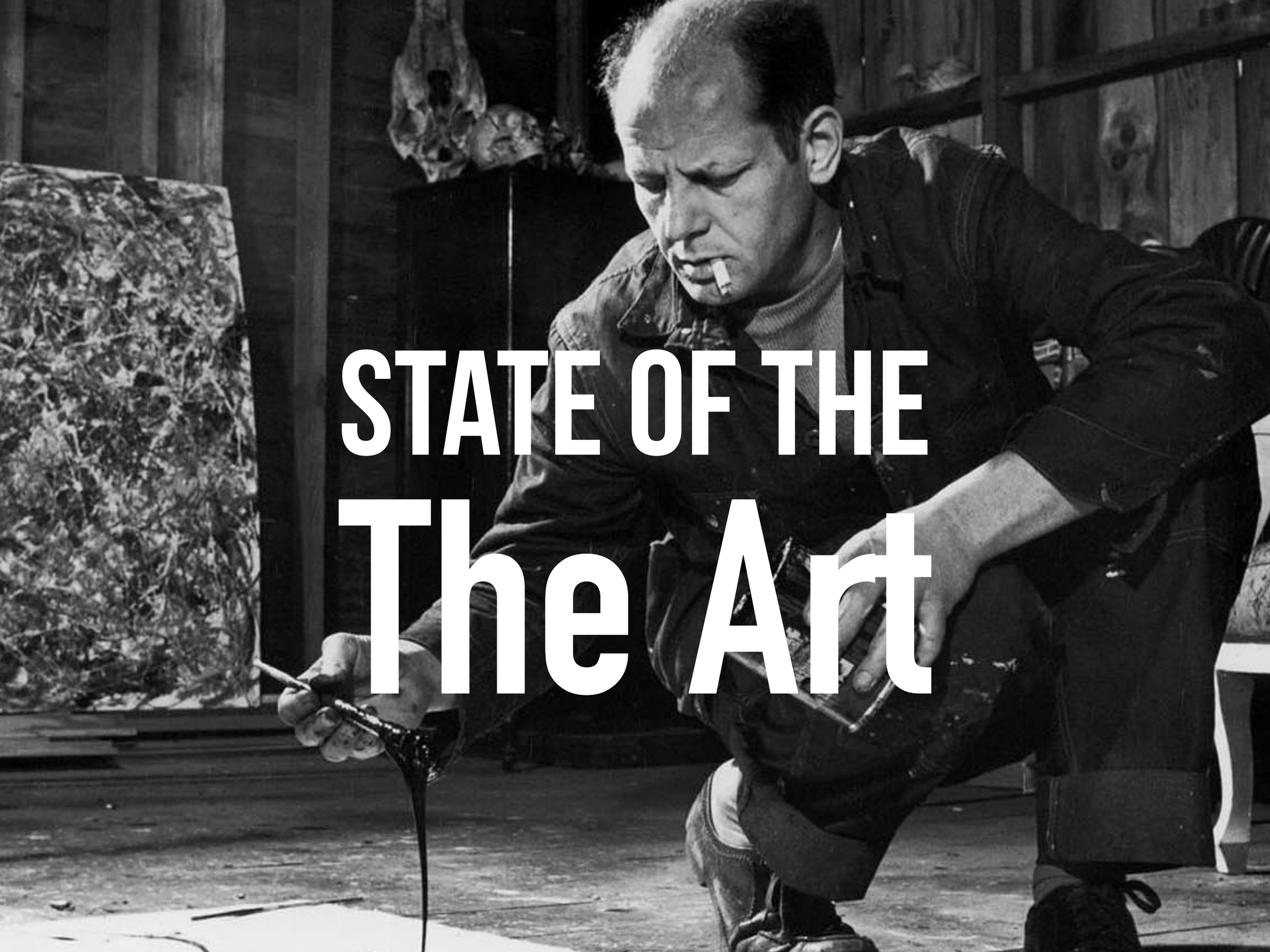
Great model for

Mobility

OTHER INTERESTING MODELS

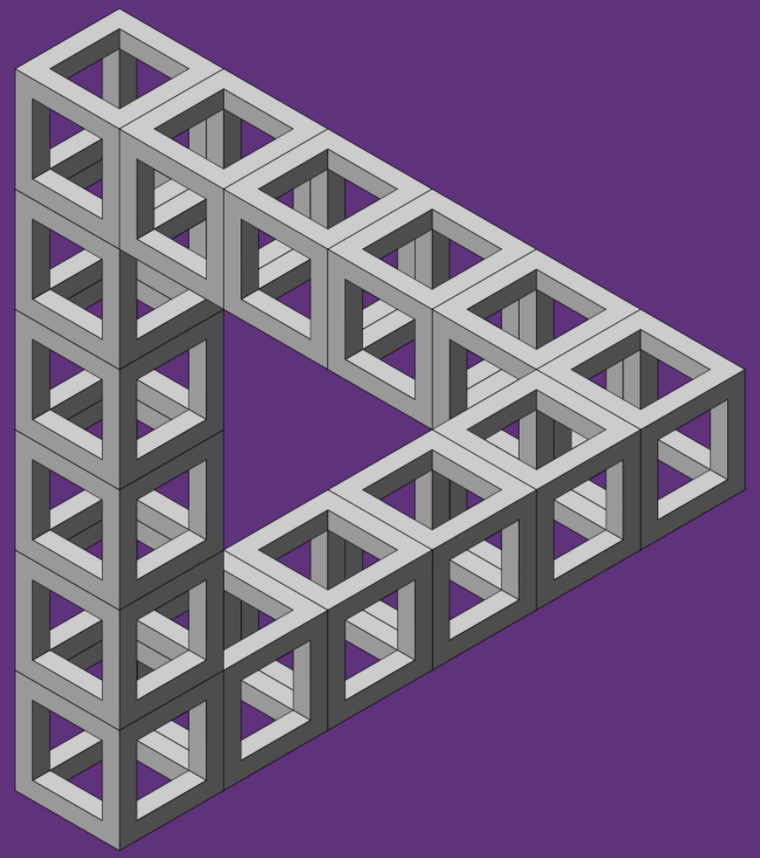
THAT ARE SUITABLE FOR DISTRIBUTED SYSTEMS

1. Pi Calculus
2. Ambient Calculus
3. Join Calculus



STATE OF THE
The Art

IMPOSSIBILITY Theorems



IMPOSSIBILITY OF DISTRIBUTED

CONSENSUS WITH ONE FAULTY PROCESS

IMPOSSIBILITY OF DISTRIBUTED CONSENSUS WITH ONE FAULTY PROCESS

FLP

FISCHER

LYNCH

PATERSON

1985

IMPOSSIBILITY OF DISTRIBUTED CONSENSUS WITH ONE FAULTY PROCESS

FLP

FISCHER

LYNCH

PATERSON

1985

Consensus IS IMPOSSIBLE

IMPOSSIBILITY OF DISTRIBUTED CONSENSUS WITH ONE FAULTY PROCESS

FLP

FISCHER

LYNCH

PATERSON

1985

"The FLP result shows that in an asynchronous setting, where only one processor might crash, there is no distributed algorithm that solves the consensus problem" - THE PAPER TRAIL

Consensus IS IMPOSSIBLE

IMPOSSIBILITY OF DISTRIBUTED CONSENSUS WITH ONE FAULTY PROCESS

FLP

FISCHER

LYNCH

PATERSON

1985

IMPOSSIBILITY OF DISTRIBUTED CONSENSUS WITH ONE FAULTY PROCESS

FLP

FISCHER

LYNCH

PATERSON

1985

"These results do not show that such problems cannot be "solved" in practice; rather, they point up the need for more refined models of distributed computing" - FLP PAPER

CAP

THEOREM

CAP THEOREM

Linearizability IS IMPOSSIBLE

CAP THEOREM

Conjecture by

Eric Brewer 2000

Proof by

Lynch & Gilbert 2002

Linearizability

IS IMPOSSIBLE

BREWER'S CONJECTURE AND THE FEASIBILITY OF CONSISTENT, AVAILABLE, PARTITION-TOLERANT WEB SERVICES

CAP THEOREM

Conjecture by

Eric Brewer 2000

Proof by

Lynch & Gilbert 2002

Linearizability IS IMPOSSIBLE

LINEARIZABILITY

LINEARIZABILITY

"Under linearizable consistency, all operations appear to have executed atomically in an order that is consistent with the global real-time ordering of operations."

HERLIHY & WING 1991

LINEARIZABILITY

"Under linearizable consistency, all operations appear to have executed atomically in an order that is consistent with the global real-time ordering of operations."

HERLIHY & WING 1991

LESS FORMALLY:

A read will return the last completed write (made on any replica)

DISSECTING CAP

DISSECTING CAP

1. Very influential—but very NARROW scope

DISSECTING CAP

1. Very influential—but very NARROW scope
2. "[CAP] has lead to confusion and misunderstandings regarding replica consistency, transactional isolation and high availability" - BAILIS ET.AL IN HAT PAPER

DISSECTING CAP

1. Very influential—but very NARROW scope
2. "[CAP] has lead to confusion and misunderstandings regarding replica consistency, transactional isolation and high availability" - BAILIS ET.AL IN HAT PAPER
3. Linearizability is very often NOT required

DISSECTING CAP

1. Very influential—but very NARROW scope
2. "[CAP] has lead to confusion and misunderstandings regarding replica consistency, transactional isolation and high availability" - BAILIS ET.AL IN HAT PAPER
3. Linearizability is very often NOT required
4. Ignores LATENCY—but in practice latency & partitions are deeply related

DISSECTING CAP

1. Very influential—but very NARROW scope
2. "[CAP] has lead to confusion and misunderstandings regarding replica consistency, transactional isolation and high availability" - BAILIS ET.AL IN HAT PAPER
3. Linearizability is very often NOT required
4. Ignores LATENCY—but in practice latency & partitions are deeply related
5. Partitions are RARE—so why sacrifice C or A ALL the time?

DISSECTING CAP

1. Very influential—but very NARROW scope
2. "[CAP] has lead to confusion and misunderstandings regarding replica consistency, transactional isolation and high availability" - BAILIS ET.AL IN HAT PAPER
3. Linearizability is very often NOT required
4. Ignores LATENCY—but in practice latency & partitions are deeply related
5. Partitions are RARE—so why sacrifice C or A ALL the time?
6. NOT black and white—can be fine-grained and dynamic

DISSECTING CAP

1. Very influential—but very NARROW scope
2. "[CAP] has lead to confusion and misunderstandings regarding replica consistency, transactional isolation and high availability" - BAILIS ET.AL IN HAT PAPER
3. Linearizability is very often NOT required
4. Ignores LATENCY—but in practice latency & partitions are deeply related
5. Partitions are RARE—so why sacrifice C or A ALL the time?
6. NOT black and white—can be fine-grained and dynamic
7. Read 'CAP Twelve Years Later' - ERIC BREWER

CONSENSUS

CONSENSUS

"The problem of reaching agreement among remote processes is one of the most fundamental problems in distributed computing and is at the core of many algorithms for distributed data processing, distributed file management, and fault-tolerant distributed applications."

FISCHER, LYNCH & PATERSON 1985

CONSISTENCY MODELS

CONSISTENCY MODELS

Strong

CONSISTENCY MODELS

Strong

Weak

CONSISTENCY MODELS

Strong

Weak

Eventual

TIME & Order

LAST WRITE WINS

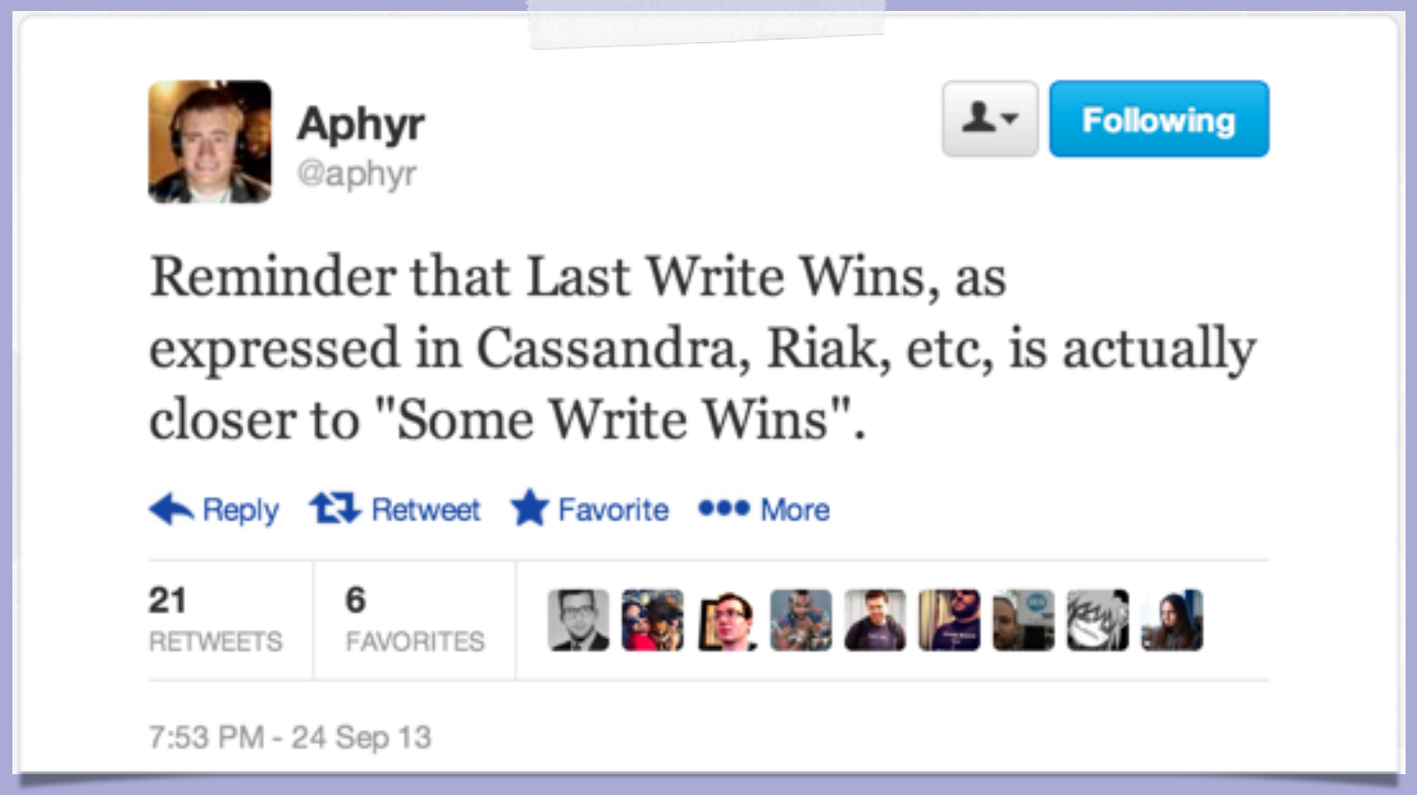
GLOBAL CLOCK

TIMESTAMP

LAST WRITE WINS

GLOBAL CLOCK

TIMESTAMP



LAMPORT CLOCKS

LOGICAL CLOCK

CAUSAL CONSISTENCY

LESLIE LAMPORT 1978

LAMPORT CLOCKS

LOGICAL CLOCK

CAUSAL CONSISTENCY

LESLIE LAMPORT 1978

1. When a process does work, increment the counter

LAMPORT CLOCKS

LOGICAL CLOCK

CAUSAL CONSISTENCY

LESLIE LAMPORT 1978

1. When a process does work, increment the counter
2. When a process sends a message, include the counter

LAMPORT CLOCKS

LOGICAL CLOCK

CAUSAL CONSISTENCY

LESLIE LAMPORT 1978

1. When a process does work, increment the counter
2. When a process sends a message, include the counter
3. When a message is received, merge the counter
   (set the counter to max(local, received) + 1)

VECTOR CLOCKS

EXTENDS

LAMPORT CLOCKS

COLIN FIDGE 1988

VECTOR CLOCKS

EXTENDS

LAMPORT CLOCKS

COLIN FIDGE 1988

1. Each node owns and increments its own Lamport Clock

VECTOR CLOCKS

EXTENDS

LAMPORT CLOCKS

COLIN FIDGE 1988

1. Each node owns and increments its own Lamport Clock

```
[node -> lamport clock]
```

VECTOR CLOCKS

EXTENDS

LAMPORT CLOCKS

COLIN FIDGE 1988

1. Each node owns and increments its own Lamport Clock

```
[node -> lamport clock]
```

VECTOR CLOCKS

EXTENDS

LAMPORT CLOCKS

COLIN FIDGE 1988

1. Each node owns and increments its own Lamport Clock

   `[node -> lamport clock]`

2. Alway keep the full history of all increments

VECTOR CLOCKS

EXTENDS

LAMPORT CLOCKS

COLIN FIDGE 1988

1. Each node owns and increments its own Lamport Clock

   ```
   [node -> lamport clock]
   ```

2. Alway keep the full history of all increments
3. Merges by calculating the max—monotonic merge

QUORUM

QUORUM

Strict MAJORITY VOTE

QUORUM

Strict MAJORITY VOTE

Sloppy PARTIAL VOTE

QUORUM

Strict MAJORITY VOTE

Sloppy PARTIAL VOTE

- Most use $R + W > N \Rightarrow$ R & W overlap

QUORUM

Strict MAJORITY VOTE

Sloppy PARTIAL VOTE

- Most use $R + W > N \Rightarrow$ R & W overlap
- If $N / 2 + 1$ is still alive $\Rightarrow$ all good

QUORUM

Strict MAJORITY VOTE

Sloppy PARTIAL VOTE

- Most use $R + W > N \Rightarrow$ R & W overlap
- If $N / 2 + 1$ is still alive $\Rightarrow$ all good
- Most use $N == 3$

FAILURE

Detection

FAILURE DETECTION

FORMAL MODEL

FAILURE DETECTION

FORMAL MODEL

STRONG COMPLETENESS

FAILURE DETECTION

FORMAL MODEL

STRONG COMPLETENESS

Every crashed process is eventually suspected by EVERY correct process

FAILURE DETECTION

FORMAL MODEL

STRONG COMPLETENESS

Every crashed process is eventually suspected by EVERY correct process

FAILURE DETECTION

FORMAL MODEL

STRONG COMPLETENESS

Every crashed process is eventually suspected by EVERY correct process

WEAK COMPLETENESS

FAILURE DETECTION

FORMAL MODEL

STRONG COMPLETENESS

Every crashed process is eventually suspected by EVERY correct process

WEAK COMPLETENESS

Every crashed process is eventually suspected by SOME correct process

FAILURE DETECTION

FORMAL MODEL

STRONG COMPLETENESS

Every crashed process is eventually suspected by EVERY correct process

WEAK COMPLETENESS

Every crashed process is eventually suspected by SOME correct process

FAILURE DETECTION

FORMAL MODEL

STRONG COMPLETENESS

Every crashed process is eventually suspected by EVERY correct process

WEAK COMPLETENESS

Every crashed process is eventually suspected by SOME correct process

STRONG ACCURACY

FAILURE DETECTION

FORMAL MODEL

STRONG COMPLETENESS

Every crashed process is eventually suspected by EVERY correct process

WEAK COMPLETENESS

Every crashed process is eventually suspected by SOME correct process

STRONG ACCURACY

No correct process is suspected ever

FAILURE DETECTION

FORMAL MODEL

STRONG COMPLETENESS

Every crashed process is eventually suspected by EVERY correct process

WEAK COMPLETENESS

Every crashed process is eventually suspected by SOME correct process

STRONG ACCURACY

No correct process is suspected ever

FAILURE DETECTION

FORMAL MODEL

STRONG COMPLETENESS

Every crashed process is eventually suspected by EVERY correct process

WEAK COMPLETENESS

Every crashed process is eventually suspected by SOME correct process

STRONG ACCURACY

No correct process is suspected ever

WEAK ACCURACY

FAILURE DETECTION

FORMAL MODEL

STRONG COMPLETENESS

Every crashed process is eventually suspected by EVERY correct process

WEAK COMPLETENESS

Every crashed process is eventually suspected by SOME correct process

STRONG ACCURACY

No correct process is suspected ever

WEAK ACCURACY

Some correct process is never suspected

FAILURE DETECTION

FORMAL MODEL

STRONG COMPLETENESS

Every crashed process is eventually suspected by EVERY correct process

WEAK COMPLETENESS

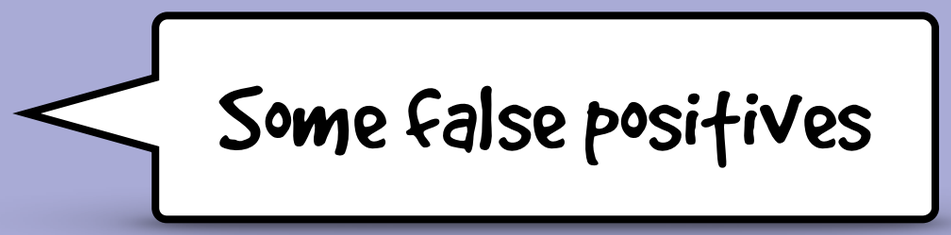
Every crashed process is eventually suspected by SOME correct process

STRONG ACCURACY

No correct process is suspected ever

WEAK ACCURACY

Some correct process is never suspected



ACCRUAL FAILURE DETECTOR

HAYASHIBARA ET. AL. 2004

ACCRUAL FAILURE DETECTOR

HAYASHIBARA ET. AL. 2004

Keeps history of heartbeat statistics

ACCRUAL FAILURE DETECTOR

HAYASHIBARA ET. AL. 2004

Keeps history of heartbeat statistics

Decouples monitoring from interpretation

ACCRUAL FAILURE DETECTOR

HAYASHIBARA ET. AL. 2004

Keeps history of heartbeat statistics

Decouples monitoring from interpretation

Calculates a likelihood

(PHI VALUE)

that the process is down

ACCRUAL FAILURE DETECTOR

HAYASHIBARA ET. AL. 2004

Keeps history of heartbeat statistics

Decouples monitoring from interpretation

Calculates a likelihood

(PHI VALUE)

that the process is down

ACCRUAL FAILURE DETECTOR

HAYASHIBARA ET. AL. 2004

Keeps history of heartbeat statistics

Decouples monitoring from interpretation

Calculates a likelihood

(PHI VALUE)

that the process is down

Takes network hiccups into account

ACCRUAL FAILURE DETECTOR

HAYASHIBARA ET. AL. 2004

Keeps history of heartbeat statistics

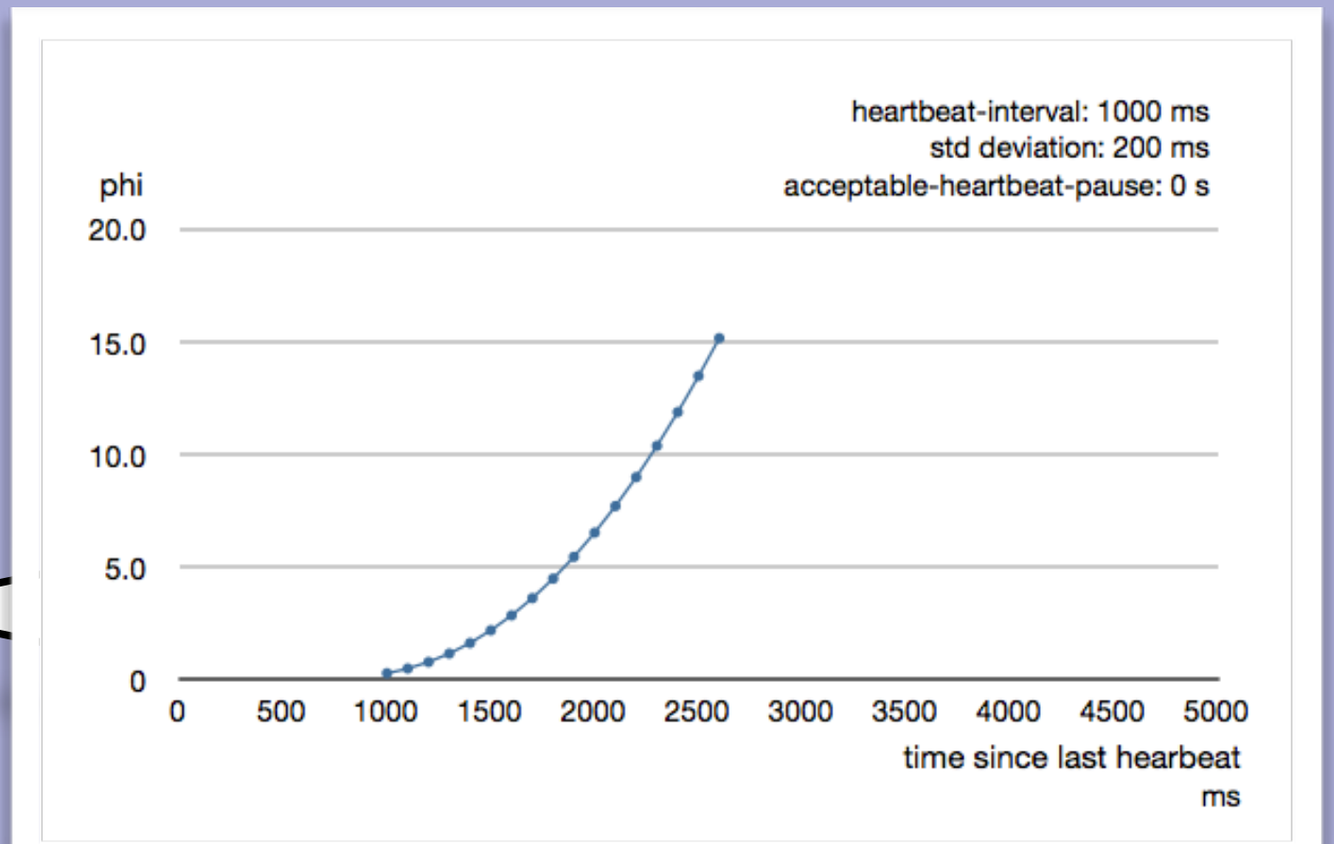
Decouples monitoring from interpretation

Calculates a likelihood

(PHI VALUE)

that the process is down

Takes network hiccups into account



phi = -log10(1 - F(timeSinceLastHeartbeat))

F is the cumulative distribution function of a normal distribution with mean and standard deviation estimated from historical heartbeat inter-arrival times

SWIM FAILURE DETECTOR

DAS ET. AL. 2002

SWIM FAILURE DETECTOR

DAS ET. AL. 2002

SEPARATES HEARTBEATS from cluster dissemination

SWIM FAILURE DETECTOR

DAS ET. AL. 2002

SEPARATES HEARTBEATS from cluster dissemination

QUARANTINE: suspected ⇒ time window ⇒ faulty

SWIM FAILURE DETECTOR

DAS ET. AL. 2002

SEPARATES HEARTBEATS from cluster dissemination

QUARANTINE: suspected ⇒ time window ⇒ faulty

DELEGATED HEARTBEAT to bridge network splits

BYZANTINE FAILURE DETECTOR

LISKOV ET. AL. 1999

BYZANTINE FAILURE DETECTOR

LISKOV ET. AL. 1999

SUPPORTS MISBEHAVING PROCESSES

BYZANTINE FAILURE DETECTOR

LISKOV ET. AL. 1999

SUPPORTS OMISSION FAILURES MISBEHAVING PROCESSES

BYZANTINE FAILURE DETECTOR

LISKOV ET. AL. 1999

SUPPORTS MISBEHAVING PROCESSES

OMISSION FAILURES

Crash failures, failing to receive a request, or failing to send a response

BYZANTINE FAILURE DETECTOR

LISKOV ET. AL. 1999

SUPPORTS

MISBEHAVING PROCESSES

OMISSION FAILURES

Crash failures, failing to receive a request, or failing to send a response

COMMISSION FAILURES

BYZANTINE FAILURE DETECTOR

LISKOV ET. AL. 1999

SUPPORTS MISBEHAVING PROCESSES

OMISSION FAILURES

Crash failures, failing to receive a request, or failing to send a response

COMMISSION FAILURES

Processing a request incorrectly, corrupting local state, and/or sending an incorrect or inconsistent response to a request

BYZANTINE FAILURE DETECTOR

LISKOV ET. AL. 1999

SUPPORTS MISBEHAVING PROCESSES

OMISSION FAILURES

Crash failures, failing to receive a request, or failing to send a response

COMMISSION FAILURES

Processing a request incorrectly, corrupting local state, and/or sending an incorrect or inconsistent response to a request

Very expensive, not practical

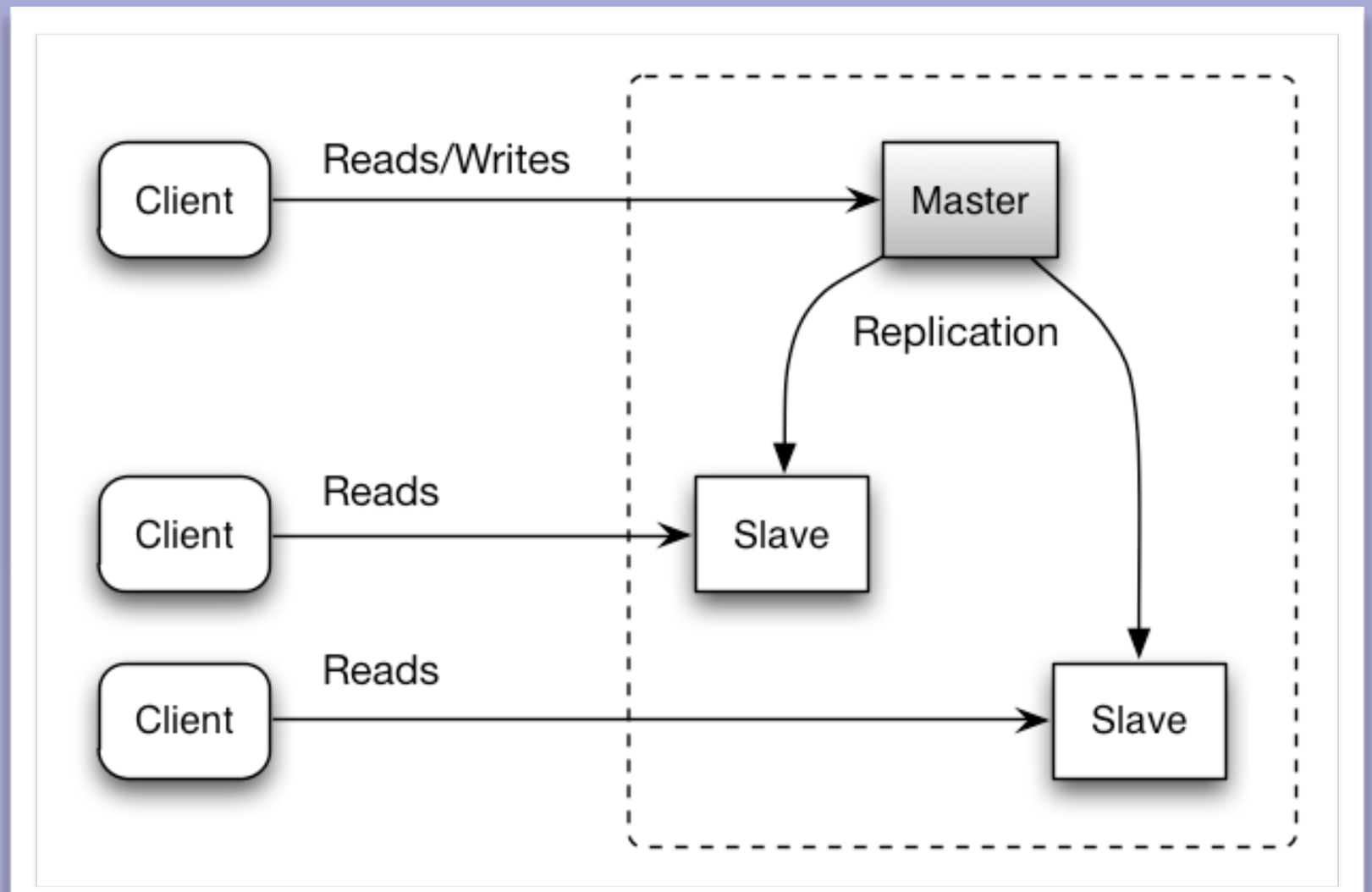
REPLICATION

TYPES OF REPLICATION

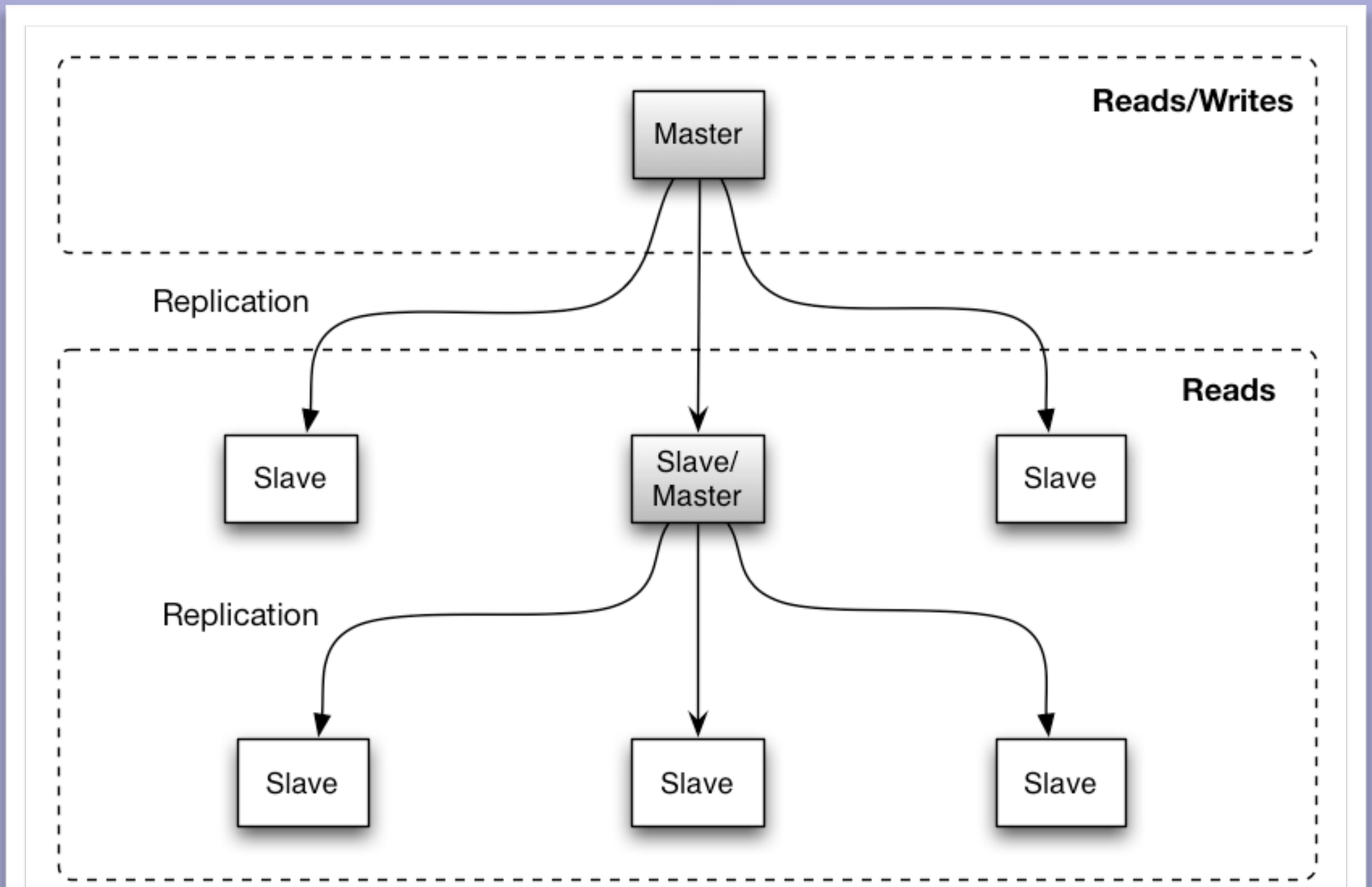
Active (PUSH) vs Passive (PULL)

Asynchronous vs Synchronous

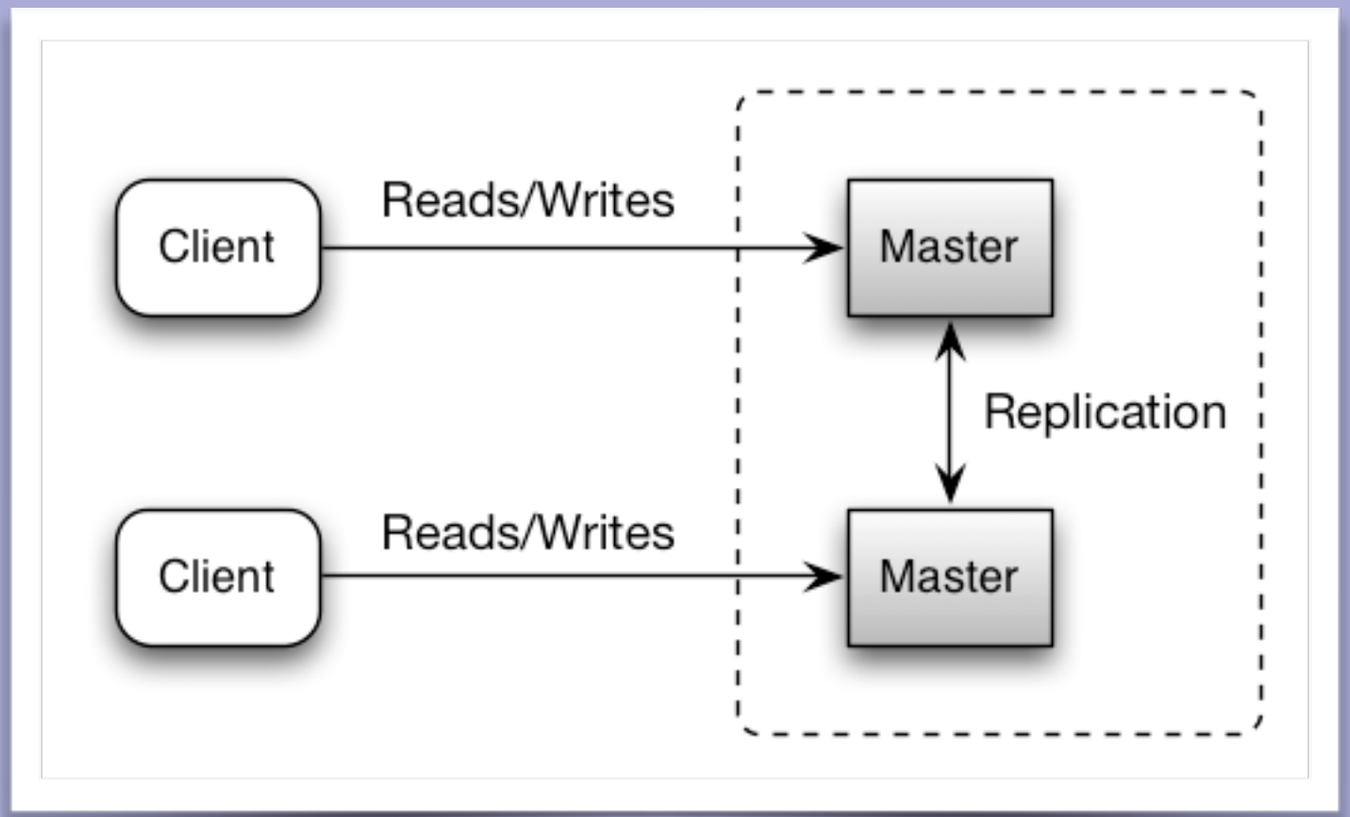
MASTER/SLAVE REPLICATION



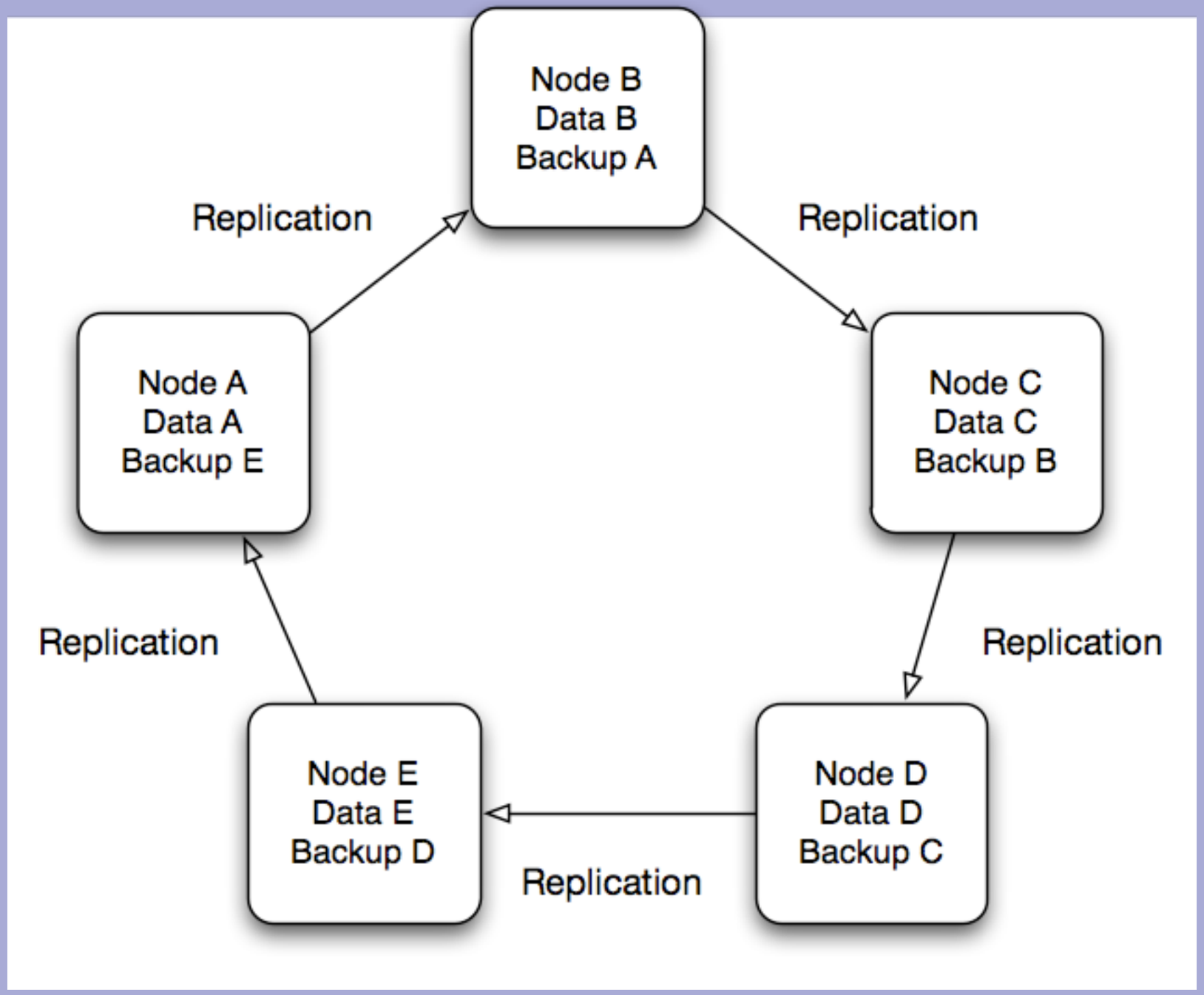
TREE REPLICATION



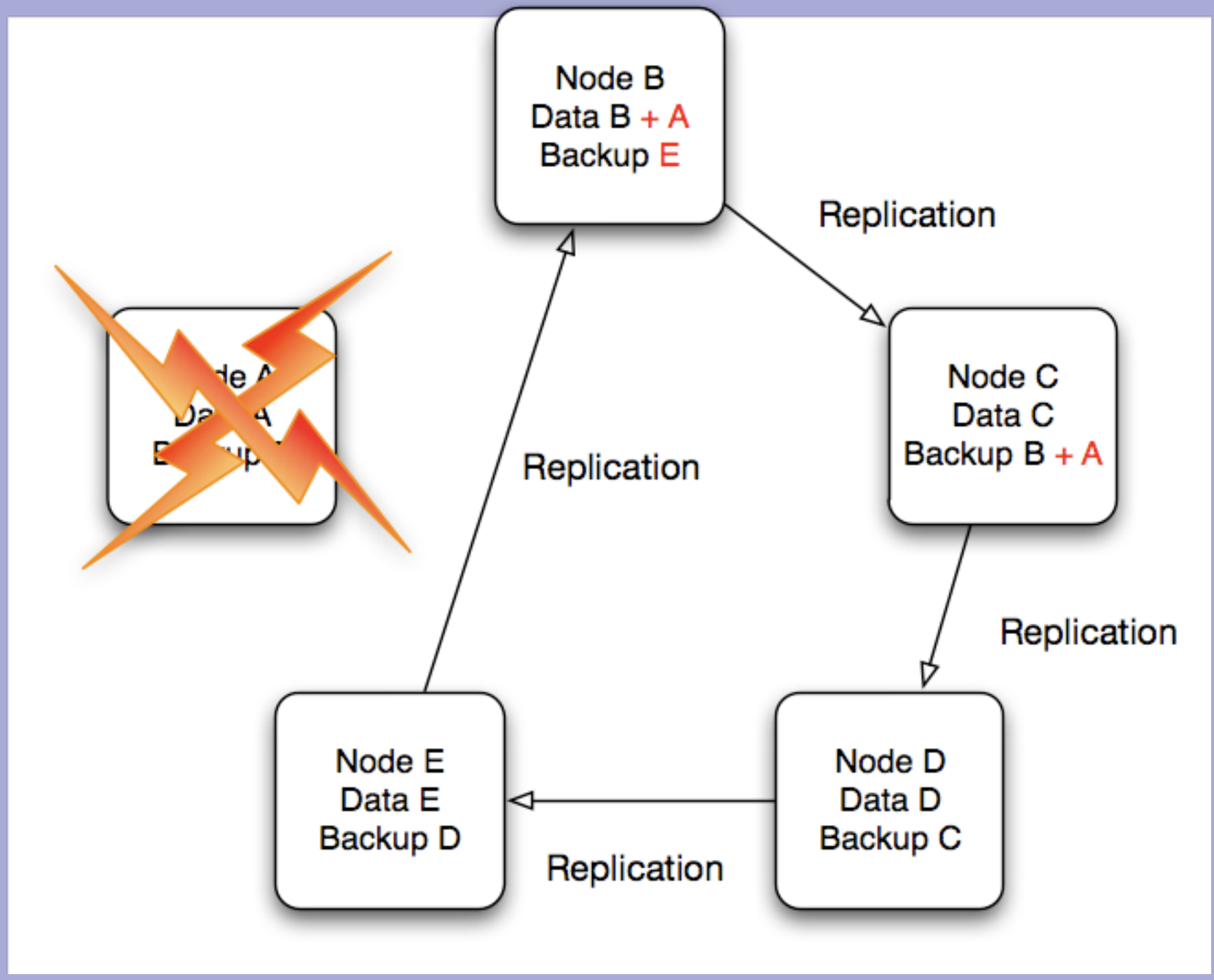
MASTER/MASTER REPLICATION



BUDDY REPLICATION



BUDDY REPLICATION



ANALYSIS OF REPLICATION CONSENSUS STRATEGIES

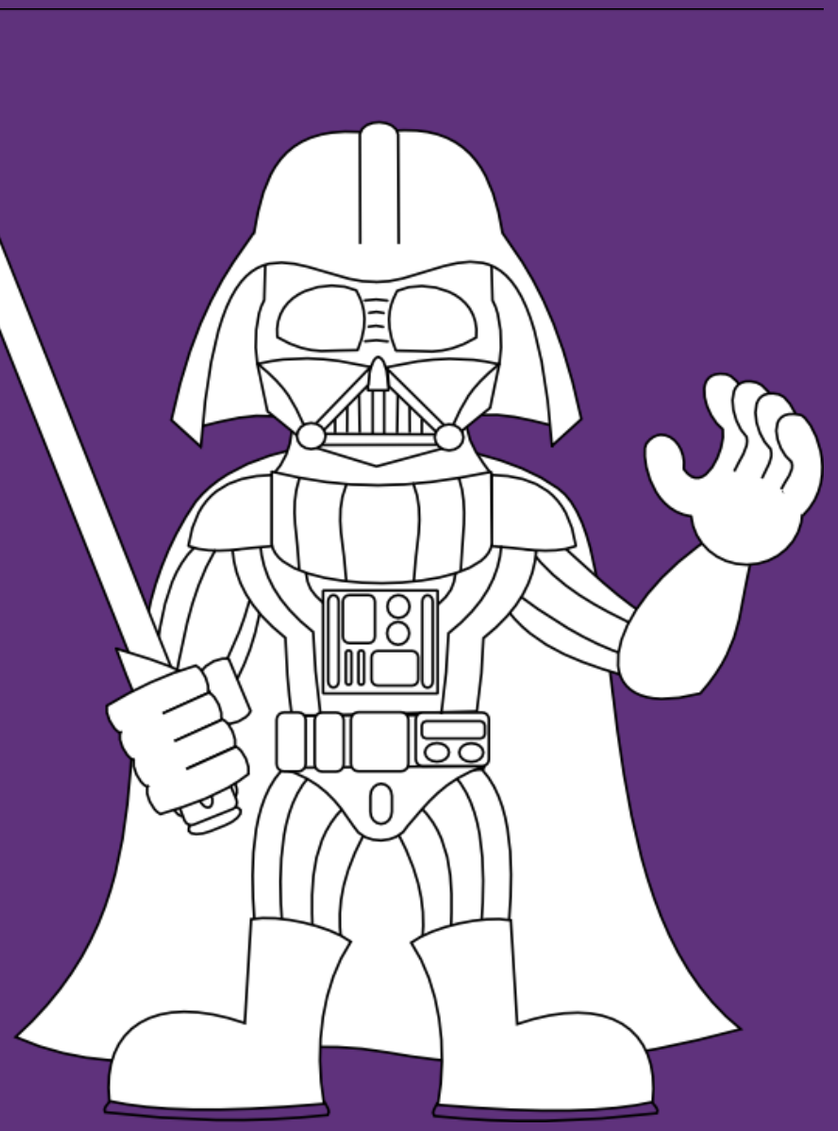
	Backups	M/S	MM	2PC	Paxos
Consistency	Weak	Eventual		Strong	
Transactions	No	Full	Local	Full	
Latency	Low			High	
Throughput	High			Low	Medium
Data loss	Lots	Some		None	
Failover	Down	Read only	Read/write		

RYAN BARRETT 2009

STRONG

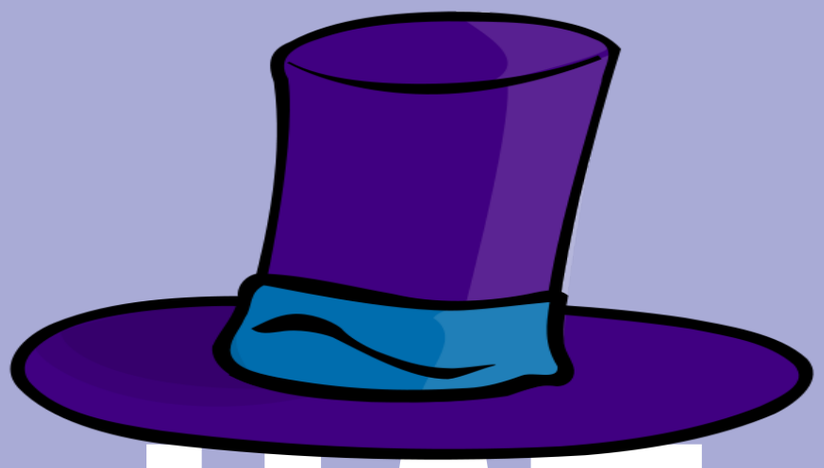
Consistency

DISTRIBUTED TRANSACTIONS Strikes Back



HIGHLY AVAILABLE TRANSACTIONS

PETER BAILIS ET. AL. 2013



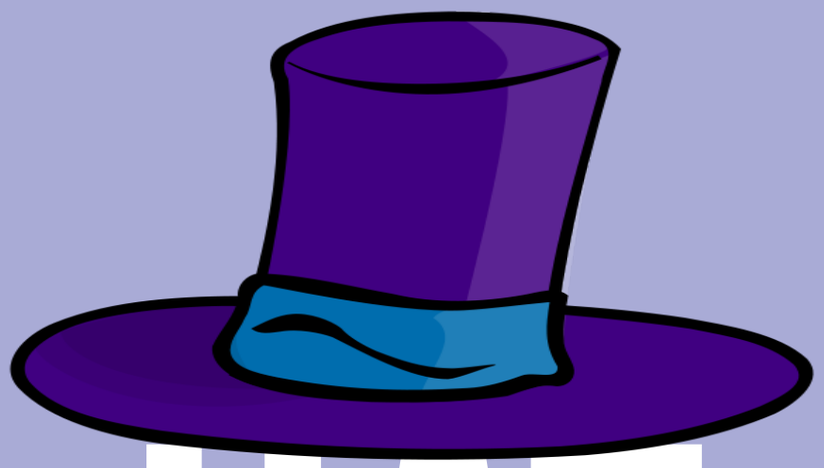
HAT

NOT

CAP

HIGHLY AVAILABLE TRANSACTIONS

PETER BAILIS ET. AL. 2013



HAT

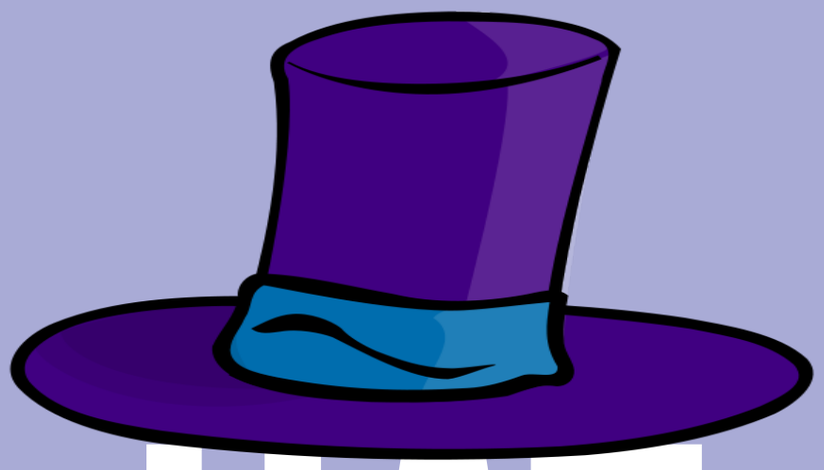
NOT

CAP

EXECUTIVE SUMMARY

HIGHLY AVAILABLE TRANSACTIONS

PETER BAILIS ET. AL. 2013



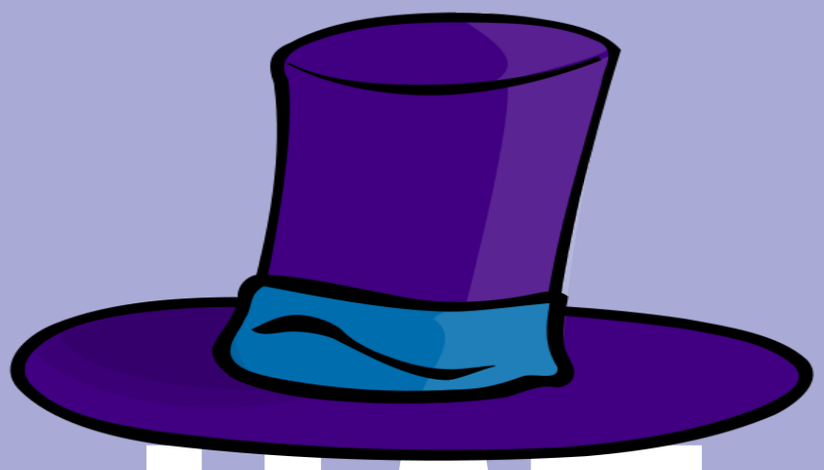
HAT NOT CAP

EXECUTIVE SUMMARY

- Most SQL DBs do not provide Serializability, but weaker guarantees—for performance reasons

HIGHLY AVAILABLE TRANSACTIONS

PETER BAILIS ET. AL. 2013



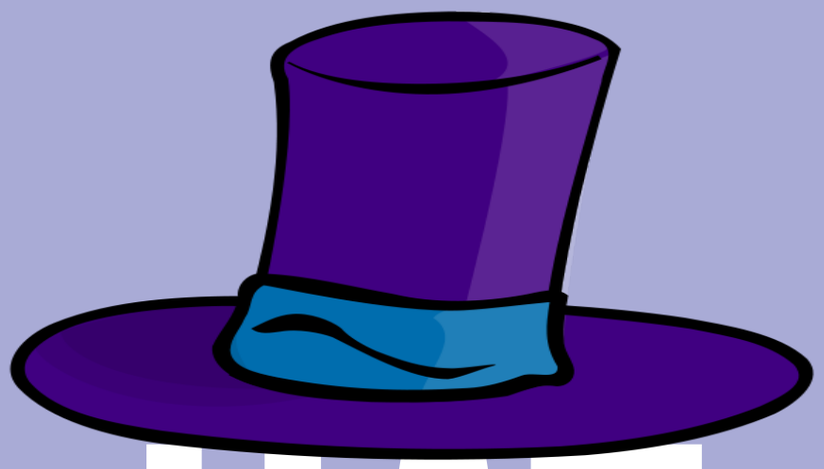
HAT NOT CAP

EXECUTIVE SUMMARY

- Most SQL DBs do not provide Serializability, but weaker guarantees—for performance reasons
- Some weaker transaction guarantees are possible to implement in a HA manner

HIGHLY AVAILABLE TRANSACTIONS

PETER BAILIS ET. AL. 2013



HAT NOT CAP

EXECUTIVE SUMMARY

- Most SQL DBs do not provide Serializability, but weaker guarantees—for performance reasons
- Some weaker transaction guarantees are possible to implement in a HA manner
- What transaction semantics can be provided with HA?

HAT

Database	Default	Maximum
Actian Ingres 10.0/10S [1]	S	S
Aerospike [2]	RC	RC
Akiban Persistit [3]	SI	SI
Clustrix CLX 4100 [4]	RR	RR
Greenplum 4.1 [8]	RC	S
IBM DB2 10 for z/OS [5]	CS	S
IBM Informix 11.50 [9]	Depends	S
MySQL 5.6 [12]	RR	S
MemSQL 1b [10]	RC	RC
MS SQL Server 2012 [11]	RC	S
NuoDB [13]	CR	CR
Oracle 11g [14]	RC	SI
Oracle Berkeley DB [7]	S	S
Oracle Berkeley DB JE [6]	RR	S
Postgres 9.2.2 [15]	RC	S
SAP HANA [16]	RC	SI
ScaleDB 1.02 [17]	RC	RC
VoltDB [18]	S	S

RC: read committed, RR: repeatable read, SI: snapshot isolation, S: serializability, CS: cursor stability, CR: consistent read

HAT

Database	Default	Maximum
Actian Ingres 10.0/10S [1]	S	S
Aerospike [2]	RC	RC
Akiban Persistit [3]	SI	SI
Clustrix CLX 4100 [4]	RR	RR
Greenplum 4.1 [8]	RC	S
IBM DB2 10 for z/OS [5]	CS	S
IBM Informix 11.50 [9]	Depends	S
MySQL 5.6 [12]	RR	S
MemSQL 1b [10]	RC	RC
MS SQL Server 2012 [11]	RC	S
NuoDB [13]	CR	CR
Oracle 11g [14]	RC	SI
Oracle Berkeley DB [7]	S	S
Oracle Berkeley DB JE [6]	RR	S
Postgres 9.2.2 [15]	RC	S
SAP HANA [16]	RC	SI
ScaleDB 1.02 [17]	RC	RC
VoltDB [18]	S	S

RC: read committed, RR: repeatable read, SI: snapshot isolation, S: serializability, CS: cursor stability, CR: consistent read

UNAVAILABLE

- Serializable
- Snapshot Isolation
- Repeatable Read
- Cursor Stability
- etc.

HIGHLY AVAILABLE

- Read Committed
- Read Uncommited
- Read Your Writes
- Monotonic Atomic View
- Monotonic Read/Write
- etc.

OTHER SCALABLE OR HIGHLY AVAILABLE TRANSACTIONAL RESEARCH

OTHER SCALABLE OR HIGHLY AVAILABLE TRANSACTIONAL RESEARCH

Bolt-On Consistency

BAILIS ET. AL. 2013

OTHER SCALABLE OR HIGHLY AVAILABLE TRANSACTIONAL RESEARCH

Bolt-On Consistency

Calvin

BAILIS ET. AL. 2013

THOMPSON ET. AL. 2012

OTHER SCALABLE OR HIGHLY AVAILABLE TRANSACTIONAL RESEARCH

Bolt-On Consistency	BAILIS ET. AL. 2013
Calvin	THOMPSON ET. AL. 2012
Spanner (Google)	CORBETT ET. AL. 2012

CONSENSUS

Protocols

SPECIFICATION

SPECIFICATION

PROPERTIES

SPECIFICATION

PROPERTIES

EVENTS

1. Request(v)
2. Decide(v)

SPECIFICATION

EVENTS

1. Request(v)
2. Decide(v)

PROPERTIES

1. **TERMINATION:** every process eventually decides on a value v

SPECIFICATION

EVENTS

1. Request(v)
2. Decide(v)

PROPERTIES

1. TERMINATION: every process eventually decides on a value v
2. VALIDITY: if a process decides v, then v was proposed by some process

SPECIFICATION

EVENTS

1. Request(v)
2. Decide(v)

PROPERTIES

1. TERMINATION: every process eventually decides on a value v
2. VALIDITY: if a process decides v, then v was proposed by some process
3. INTEGRITY: no process decides twice

SPECIFICATION

EVENTS

1. Request(v)
2. Decide(v)

PROPERTIES

1. TERMINATION: every process eventually decides on a value v
2. VALIDITY: if a process decides v, then v was proposed by some process
3. INTEGRITY: no process decides twice
4. AGREEMENT: no two correct processes decide differently

CONSENSUS ALGORITHMS

CAP

CONSENSUS ALGORITHMS

CAP

CONSENSUS ALGORITHMS

VR

OKI & LISKOV 1988

CAP

CONSENSUS ALGORITHMS

CAP

VR

OKI & LISKOV 1988

Paxos

LAMPORT 1989

CONSENSUS ALGORITHMS

CAP

VR

OKI & LISKOV 1988

Paxos

LAMPORT 1989

ZAB

REED & JUNQUIERA 2008

CONSENSUS ALGORITHMS

CAP

VR — OKI & LISKOV 1988

Paxos — LAMPORT 1989

ZAB — REED & JUNQUIERA 2008

Raft — ONGARO & OUSTERHOUT 2013

EVENT Log

IMMUTABILITY

"Immutability Changes Everything" - PAT HELLAND

IMMUTABLE Data

SHARE NOTHING Architecture

IMMUTABILITY

"Immutability Changes Everything" - PAT HELLAND

IMMUTABLE Data

SHARE NOTHING Architecture

Is the path towards

TRUE SCALABILITY

THINK IN FACTS

"The database is a cache of a subset of the log" - PAT HELLAND

THINK IN FACTS

"The database is a cache of a subset of the log" - PAT HELLAND

NEVER DELETE data

Knowledge only GROWS

APPEND-ONLY Event Log

Use Event Sourcing and/or CQRS

AGGREGATE ROOTS

Can wrap multiple Entities

Aggregate Root is the TRANSACTIONAL BOUNDARY

AGGREGATE ROOTS

Can wrap multiple Entities

Aggregate Root is the TRANSACTIONAL BOUNDARY

STRONG Consistency WITHIN Aggregate

EVENTUAL Consistency BETWEEN Aggregates

AGGREGATE ROOTS

Can wrap multiple Entities

Aggregate Root is the TRANSACTIONAL BOUNDARY

STRONG Consistency WITHIN Aggregate

EVENTUAL Consistency BETWEEN Aggregates

NO LIMIT TO SCALABILITY

EVENTUAL

Consistency

DYNAMO

VOGELS ET. AL. 2007

VERY

INFLUENTIAL

CAP

DYNAMO

VOGELS ET. AL. 2007

VERY

INFLUENTIAL

CAP

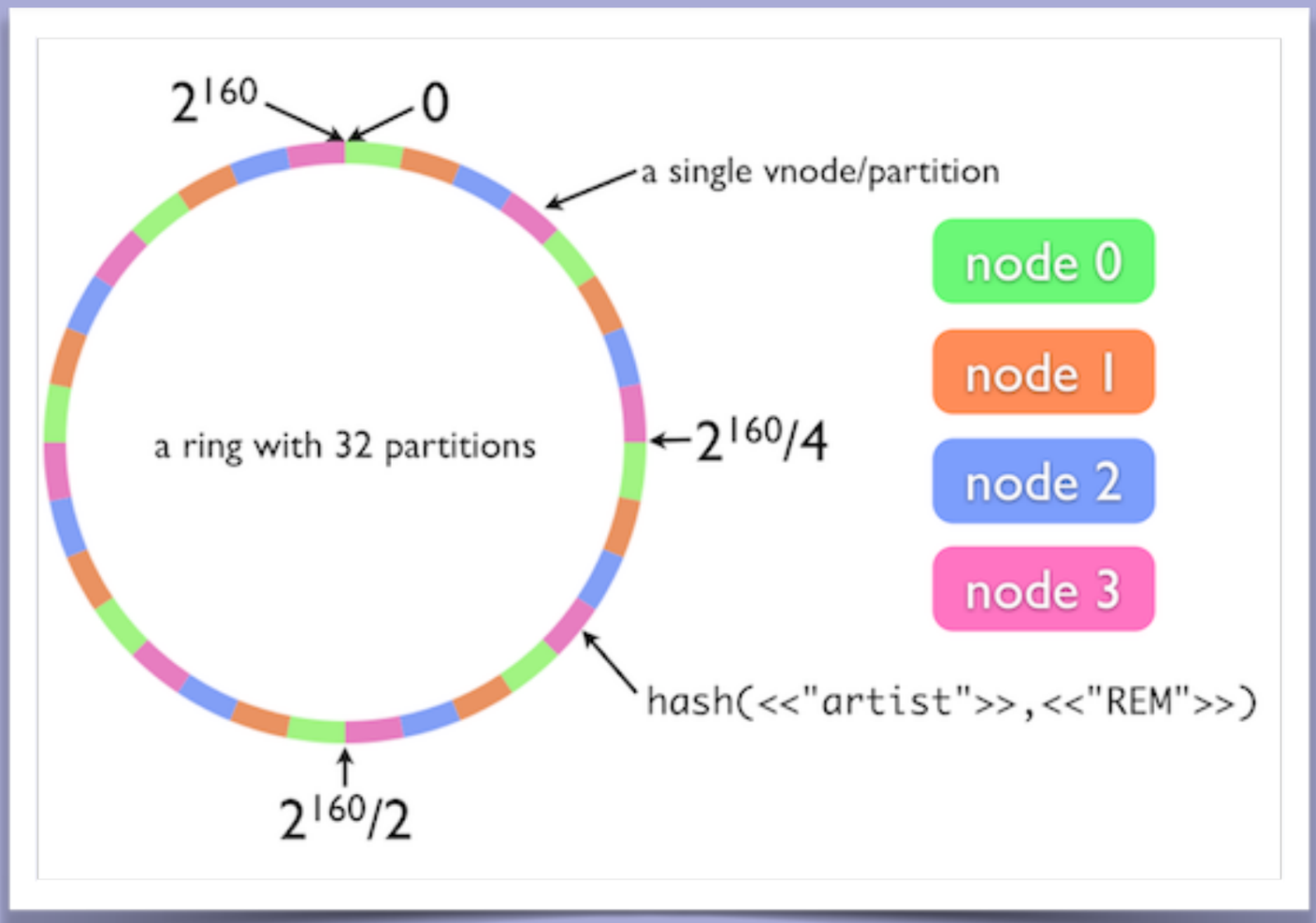
Popularized

- EVENTUAL CONSISTENCY
- EPIDEMIC GOSSIP
- CONSISTENT HASHING

- HINTED HANDOFF
- READ REPAIR
- ANTI-ENTROPY W/ MERKLE TREES

CONSISTENT HASHING

KARGER ET. AL. 1997



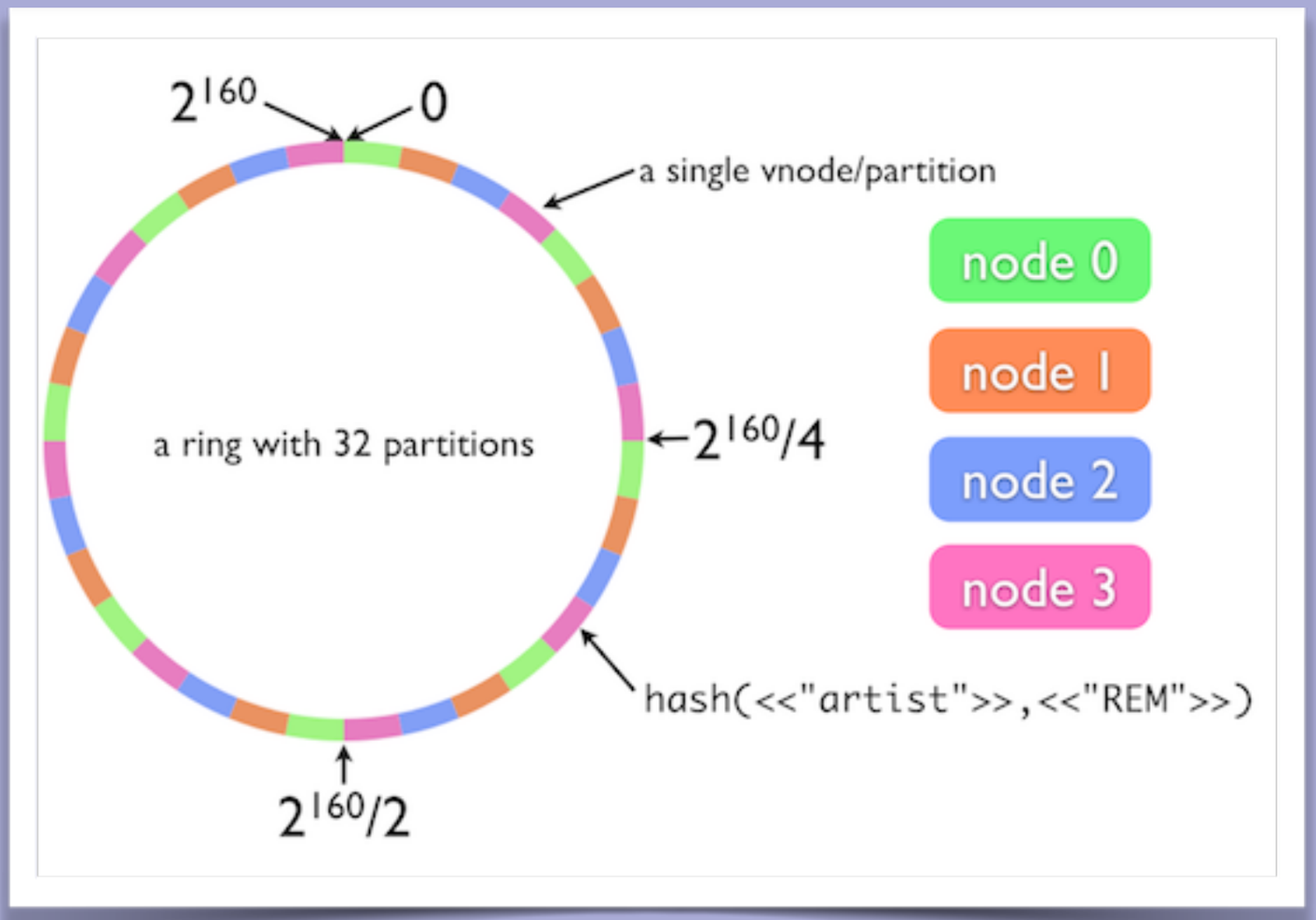
CONSISTENT HASHING

KARGER ET. AL. 1997

Support **ELASTICITY**—
easier to scale up and down

Avoids **HOTSPOTS**

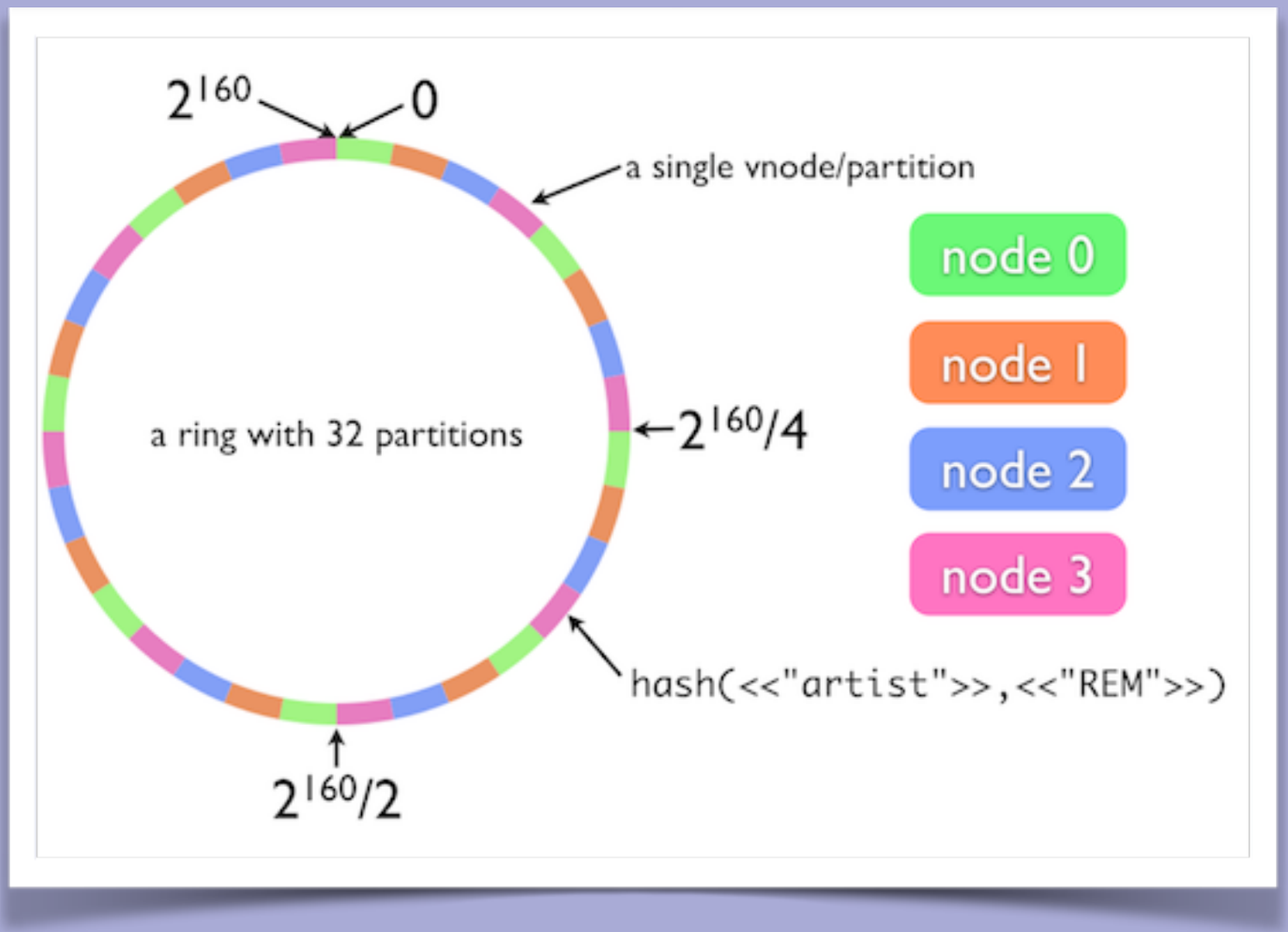
Enables **PARTITIONING** and **REPLICATION**



CONSISTENT HASHING

KARGER ET. AL. 1997

Support **ELASTICITY**—
easier to scale up and down
Avoids **HOTSPOTS**
Enables **PARTITIONING** and **REPLICATION**



Only **K/N NODES** needs to be remapped when adding or removing a node (**K=#KEYS, N=#NODES**)

HOW EVENTUAL IS

HOW EVENTUAL IS EVENTUAL CONSISTENCY?

HOW EVENTUAL IS

HOW CONSISTENT IS

EVENTUAL CONSISTENCY?

HOW EVENTUAL IS

HOW CONSISTENT IS

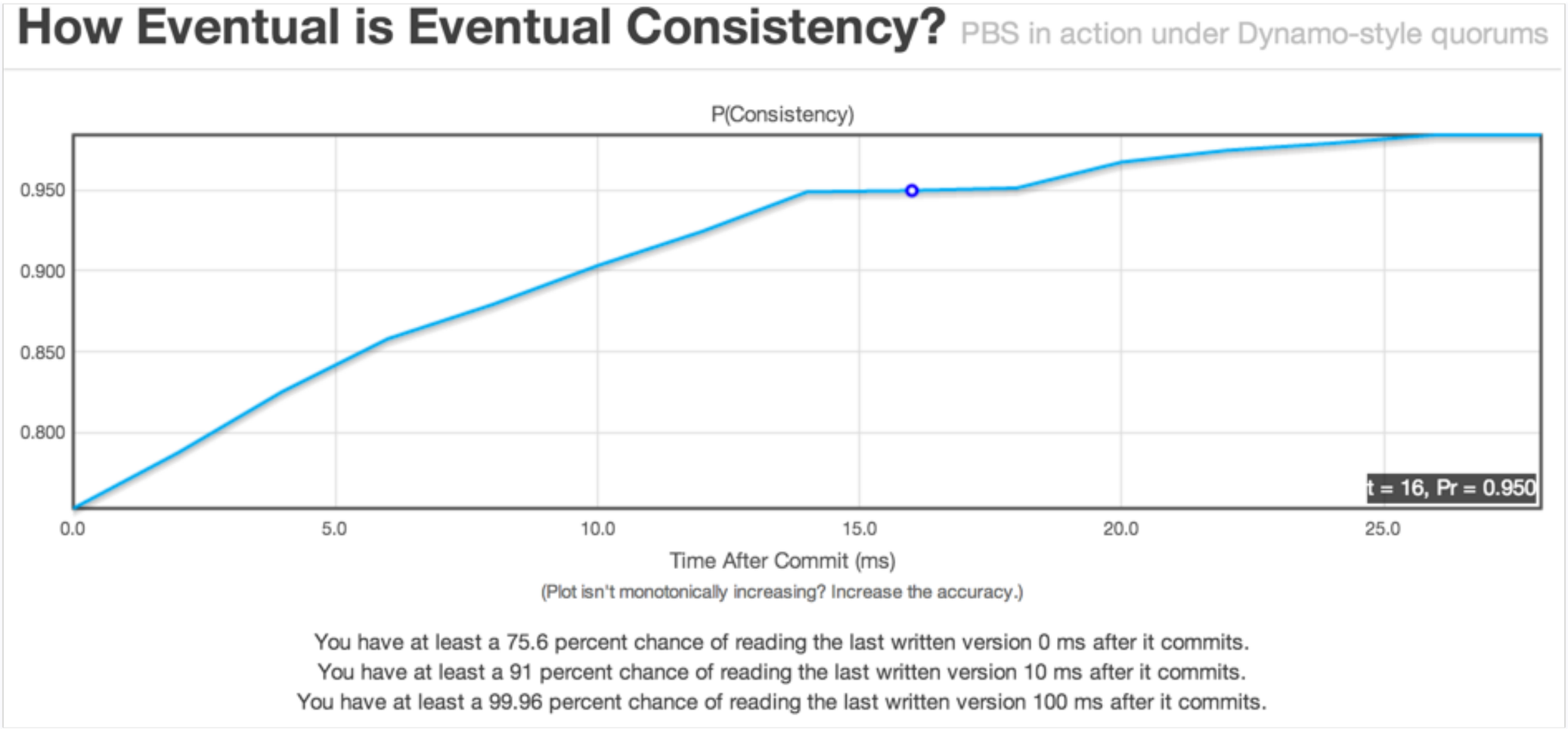
EVENTUAL CONSISTENCY?

PBS

Probabilistically Bounded Staleness

PETER BAILIS ET. AL 2012

HOW EVENTUAL IS EVENTUAL

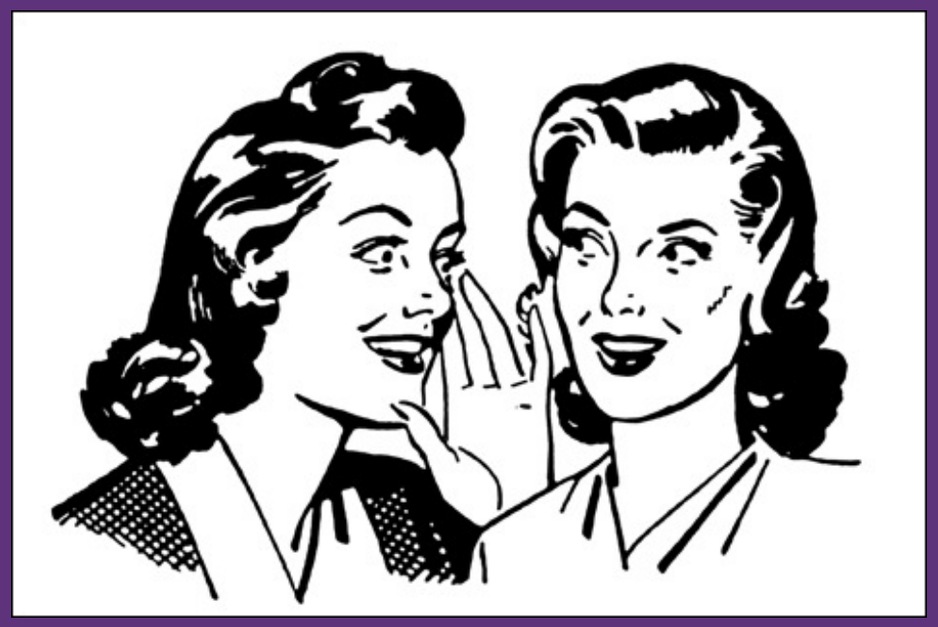


Bounded Staleness

PETER BAILIS ET. AL 2012

EPIDEMIC

Gossip



NODE RING & EPIDEMIC GOSSIP

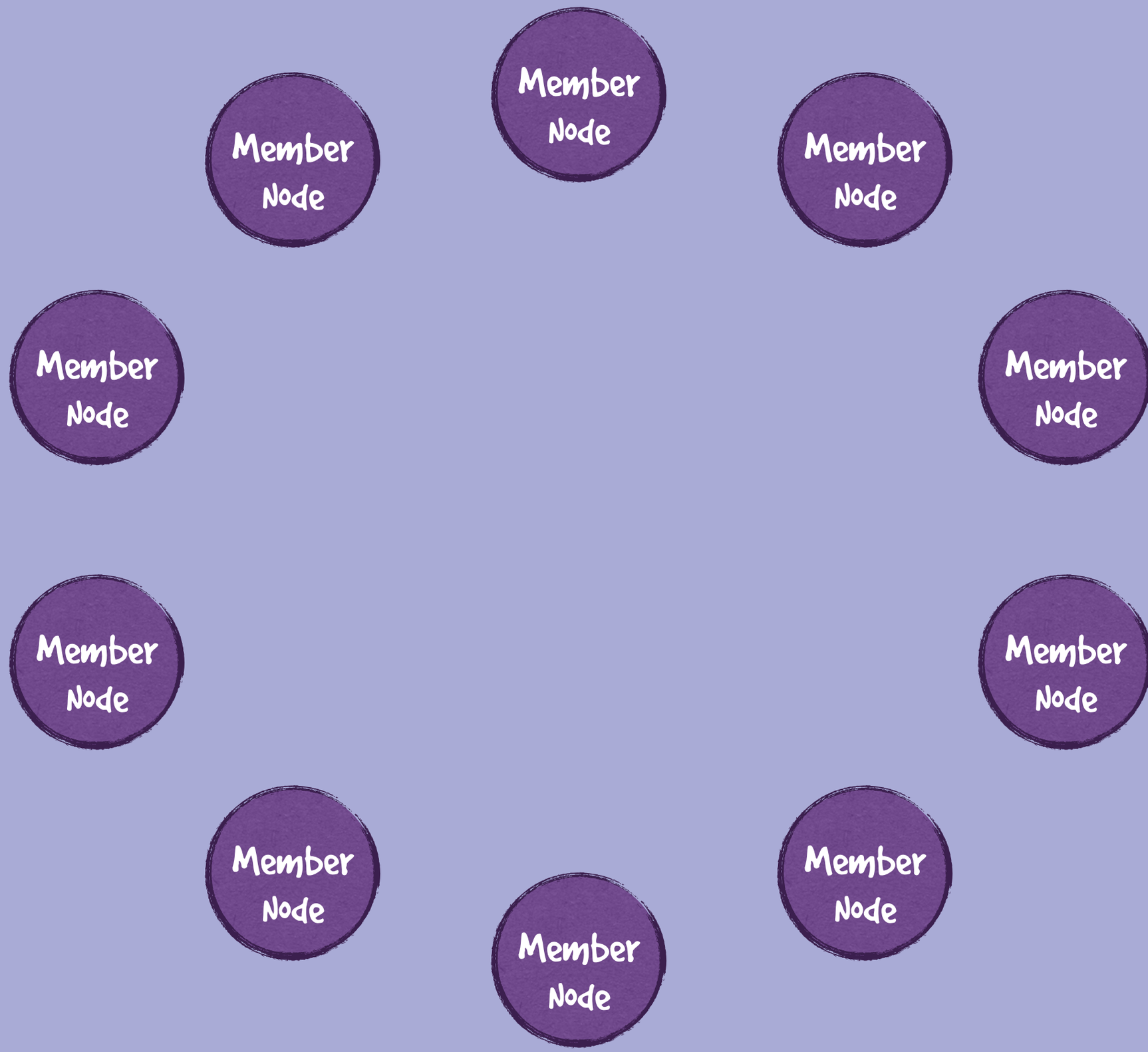
CHORD

STOICA ET AL 2001

NODE RING & EPIDEMIC GOSSIP

CHORD

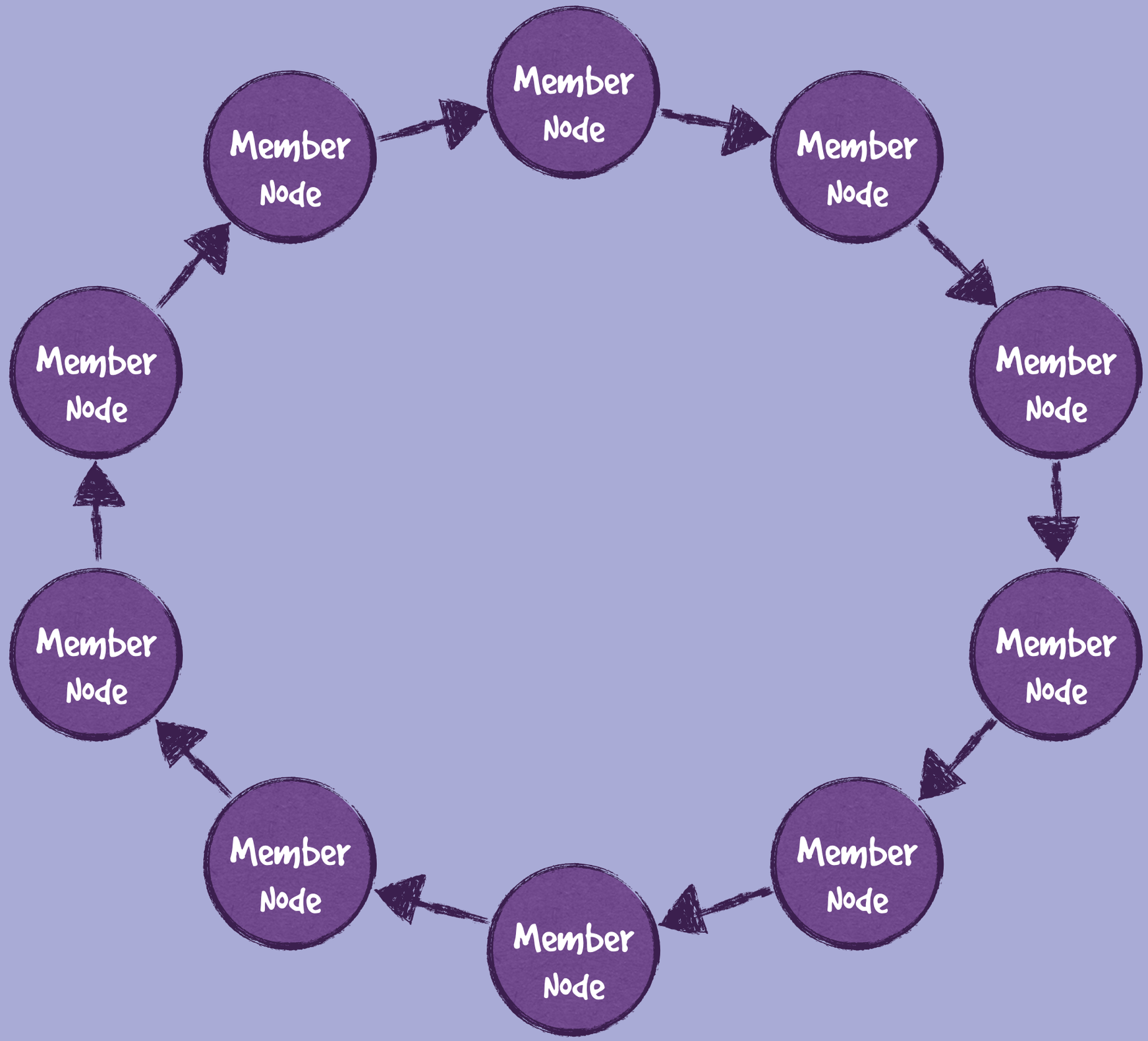
STOICA ET AL 2001



NODE RING & EPIDEMIC GOSSIP

CHORD

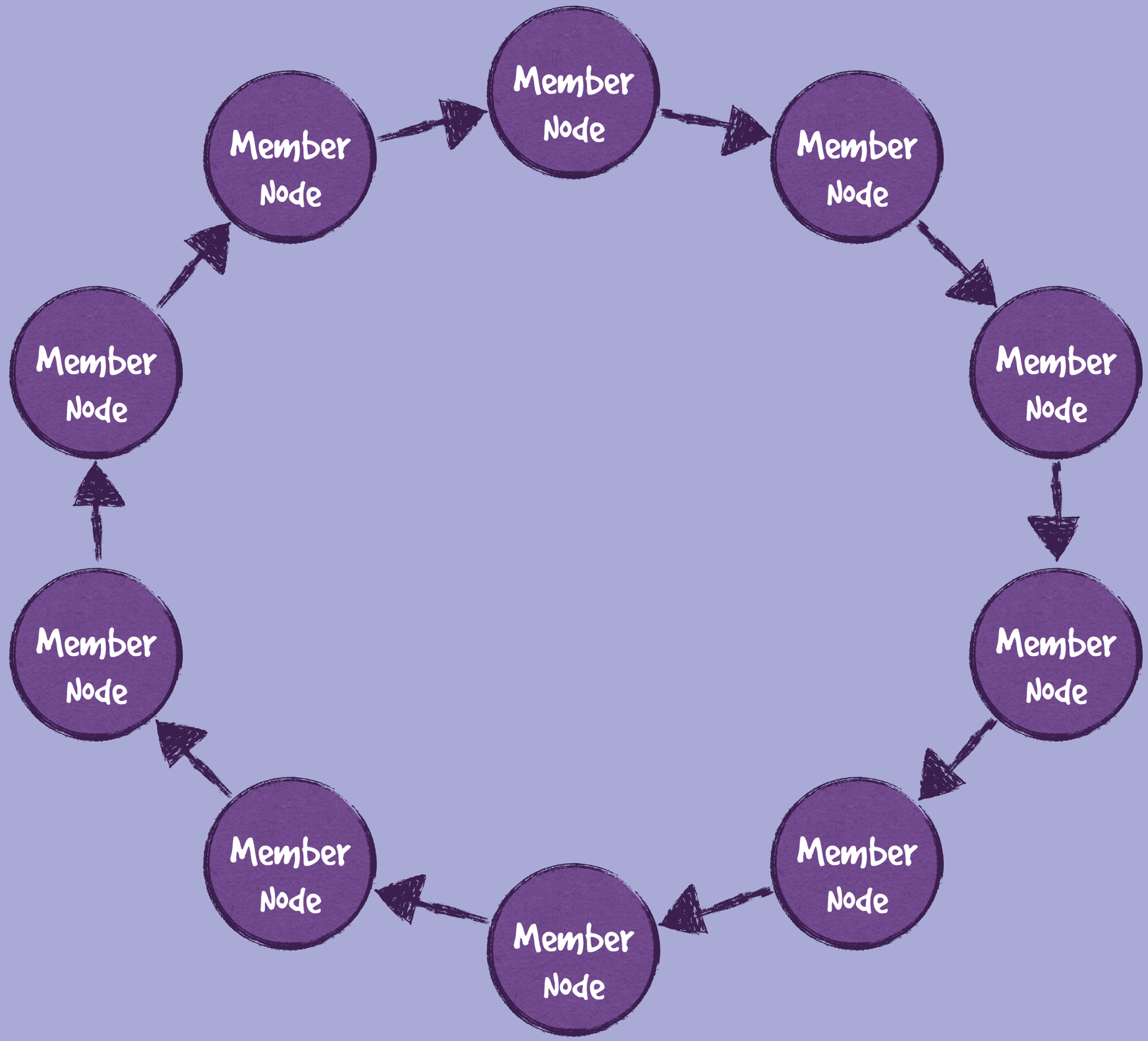
STOICA ET AL 2001



NODE RING & EPIDEMIC GOSSIP

CHORD

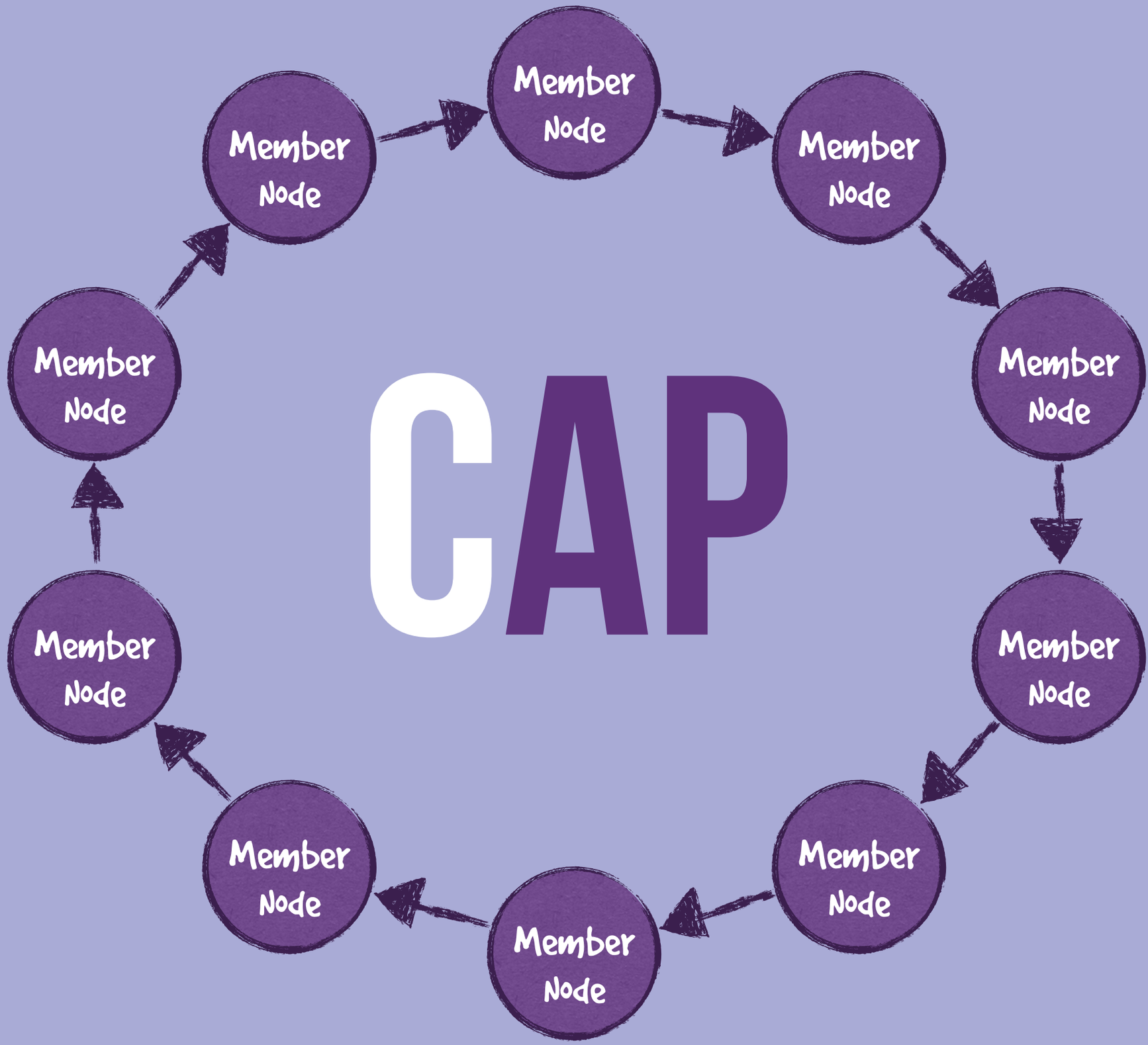
STOICA ET AL 2001



NODE RING & EPIDEMIC GOSSIP

CHORD

STOICA ET AL 2001



CAP

BENEFITS OF EPIDEMIC GOSSIP

Decentralized P2P

No SPOF or SPOB

Very SCALABLE

Fully ELASTIC

Requires minimal ADMINISTRATION

Often used with VECTOR CLOCKS

SOME STANDARD OPTIMIZATIONS TO EPIDEMIC GOSSIP

1. Separation of failure detection heartbeat and dissemination of data – DAS et. al. 2002 (SWIM)
2. Push/Pull gossip - KHAMBATTI ET. AL 2003
    1. Hash and compare data
    2. Use single hash or Merkle Trees

DISORDERLY

Programming

ACID 2.0

ACID 2.0

Associative

Batch-insensitive

(grouping doesn't matter)

$a+(b+c)=(a+b)+c$

ACID 2.0

Associative

Batch-insensitive
(grouping doesn't matter)
$a+(b+c)=(a+b)+c$

Commutative

Order-insensitive
(order doesn't matter)
$a+b=b+a$

ACID 2.0

Associative

Batch-insensitive
(grouping doesn't matter)
a+(b+c)=(a+b)+c

Commutative

Order-insensitive
(order doesn't matter)
a+b=b+a

Idempotent

Retransmission-insensitive
(duplication does not matter)
a+a=a

ACID 2.0

Associative

Batch-insensitive
(grouping doesn't matter)
a+(b+c)=(a+b)+c

Commutative

Order-insensitive
(order doesn't matter)
a+b=b+a

Idempotent

Retransmission-insensitive
(duplication does not matter)
a+a=a

EVENTUALLY CONSISTENT

CONVERGENT & COMMUTATIVE REPLICATED DATA TYPES

SHAPIRO ET. AL. 2011

CONVERGENT & COMMUTATIVE REPLICATED DATA TYPES

SHAPIRO ET. AL. 2011

CRDT

CONVERGENT & COMMUTATIVE REPLICATED DATA TYPES

SHAPIRO ET. AL. 2011

CRDT

Join Semilattice

Monotonic merge function

CONVERGENT & COMMUTATIVE REPLICATED DATA TYPES

SHAPIRO ET. AL. 2011

CRDT

Join Semilattice

Monotonic merge function

DATA TYPES

Counters

Registers

Sets

Maps

Graphs

CONVERGENT & COMMUTATIVE REPLICATED DATA TYPES

SHAPIRO ET. AL. 2011

CRDT

Join Semilattice

Monotonic merge function

DATA TYPES

Counters

Registers

Sets

Maps

Graphs

CAP

2 TYPES OF CRDTS

CvRDT

CONVERGENT

STATE-BASED

CmRDT

COMMUTATIVE

OPS-BASED

2 TYPES OF CRDTS

CvRDT
CONVERGENT
STATE-BASED

Self contained,
holds all history

CmRDT
COMMUTATIVE
OPS-BASED

2 TYPES OF CRDTS

CvRDT

CONVERGENT

STATE-BASED

Self contained,
holds all history

CmRDT

COMMUTATIVE

OPS-BASED

Needs a reliable
broadcast channel

CALM THEOREM

HELLERSTEIN ET. AL. 2011

CONSISTENCY AS LOGICAL MONOTONICITY

CALM THEOREM

HELLERSTEIN ET. AL. 2011

CONSISTENCY AS LOGICAL MONOTONICITY

BLOOM LANGUAGE

Compiler help to detect & encapsulate non-monotonicity

CALM THEOREM

HELLERSTEIN ET. AL. 2011

CONSISTENCY AS LOGICAL MONOTONICITY

DISTRIBUTED LOGIC

Datalog/Dedalus

MONOTONIC FUNCTIONS

Just add facts to the system

MODEL STATE AS LATTICES

Similar to CRDTs (without the scope problem)

BLOOM LANGUAGE

Compiler help to detect & encapsulate non-monotonicity

The Akka Way

Akka ACTORS

Akka IO

Akka ACTORS

Akka REMOTE

Akka IO

Akka ACTORS

Akka CLUSTER

Akka REMOTE

Akka IO

Akka ACTORS

Akka CLUSTER EXTENSIONS

Akka CLUSTER

Akka REMOTE

Akka IO

Akka ACTORS

WHAT IS AKKA CLUSTER ALL ABOUT?

- Cluster Membership
- Leader & Singleton
- Cluster Sharding
- Clustered Routers (adaptive, consistent hashing, . . .)
- Clustered Supervision and Deathwatch
- Clustered Pub/Sub
- and more

CLUSTER MEMBERSHIP IN AKKA

CLUSTER MEMBERSHIP IN AKKA

- Dynamo-style master-less DECENTRALIZED P2P

CLUSTER MEMBERSHIP IN AKKA

- Dynamo-style master-less DECENTRALIZED P2P
- Epidemic GOSSIP—Node Ring

CLUSTER MEMBERSHIP IN AKKA

- Dynamo-style master-less DECENTRALIZED P2P
- Epidemic GOSSIP—Node Ring
- VECTOR CLOCKS for causal consistency

CLUSTER MEMBERSHIP IN AKKA

- Dynamo-style master-less DECENTRALIZED P2P
- Epidemic GOSSIP—Node Ring
- VECTOR CLOCKS for causal consistency
- Fully ELASTIC with NO SPOF OR SPOB

CLUSTER MEMBERSHIP IN AKKA

- Dynamo-style master-less DECENTRALIZED P2P
- Epidemic GOSSIP—Node Ring
- VECTOR CLOCKS for causal consistency
- Fully ELASTIC with NO SPOF OR SPOB
- Very SCALABLE—2400 nodes (on GCE)

CLUSTER MEMBERSHIP IN AKKA

- Dynamo-style master-less DECENTRALIZED P2P
- Epidemic GOSSIP—Node Ring
- VECTOR CLOCKS for causal consistency
- Fully ELASTIC with NO SPOF OR SPOB
- Very SCALABLE—2400 nodes (on GCE)
- High THROUGHPUT—1000 nodes in 4 min (on GCE)

GOSSIPING

Gossip STATE

```
case class Gossip(
  members: SortedSet[Member],
  seen: Set[Member],
  unreachable: Set[Member],
  version: VectorClock)
```

GOSSIPING

Gossip STATE

Is a CRDT

```
case class Gossip(
  members: SortedSet[Member],
  seen: Set[Member],
  unreachable: Set[Member],
  version: VectorClock)
```

GOSSIPING

Gossip STATE

ordered node ring

Is a CRDT

```
case class Gossip(
  members: SortedSet[Member],
  seen: Set[Member],
  unreachable: Set[Member],
  version: VectorClock)
```

GOSSIPING

STATE

ordered node ring

Is a CRDT

Seen set
for convergence

```
case class Gossip(
  members: SortedSet[Member],
  seen: Set[Member],
  unreachable: Set[Member],
  version: VectorClock)
```

GOSSIPING

ordered node ring

Is a CRDT

Seen set
for convergence

Unreachable set

```
case class Gossip(
  members: SortedSet[Member],
  seen: Set[Member],
  unreachable: Set[Member],
  version: VectorClock)
```

GOSSIPING

ordered node ring

Is a CRDT

Seen set
for convergence

Unreachable set

```
case class Gossip(
  members: SortedSet[Member],
  seen: Set[Member],
  unreachable: Set[Member],
  version: VectorClock)
```

Version

GOSSIPING

ordered node ring

Is a CRDT

Seen set
for convergence

Unreachable set

```
case class Gossip(
  members: SortedSet[Member],
  seen: Set[Member],
  unreachable: Set[Member],
  version: VectorClock)
```

Version

1. Picks random node with older/newer version

GOSSIPING

ordered node ring

Is a CRDT

Seen set
for convergence

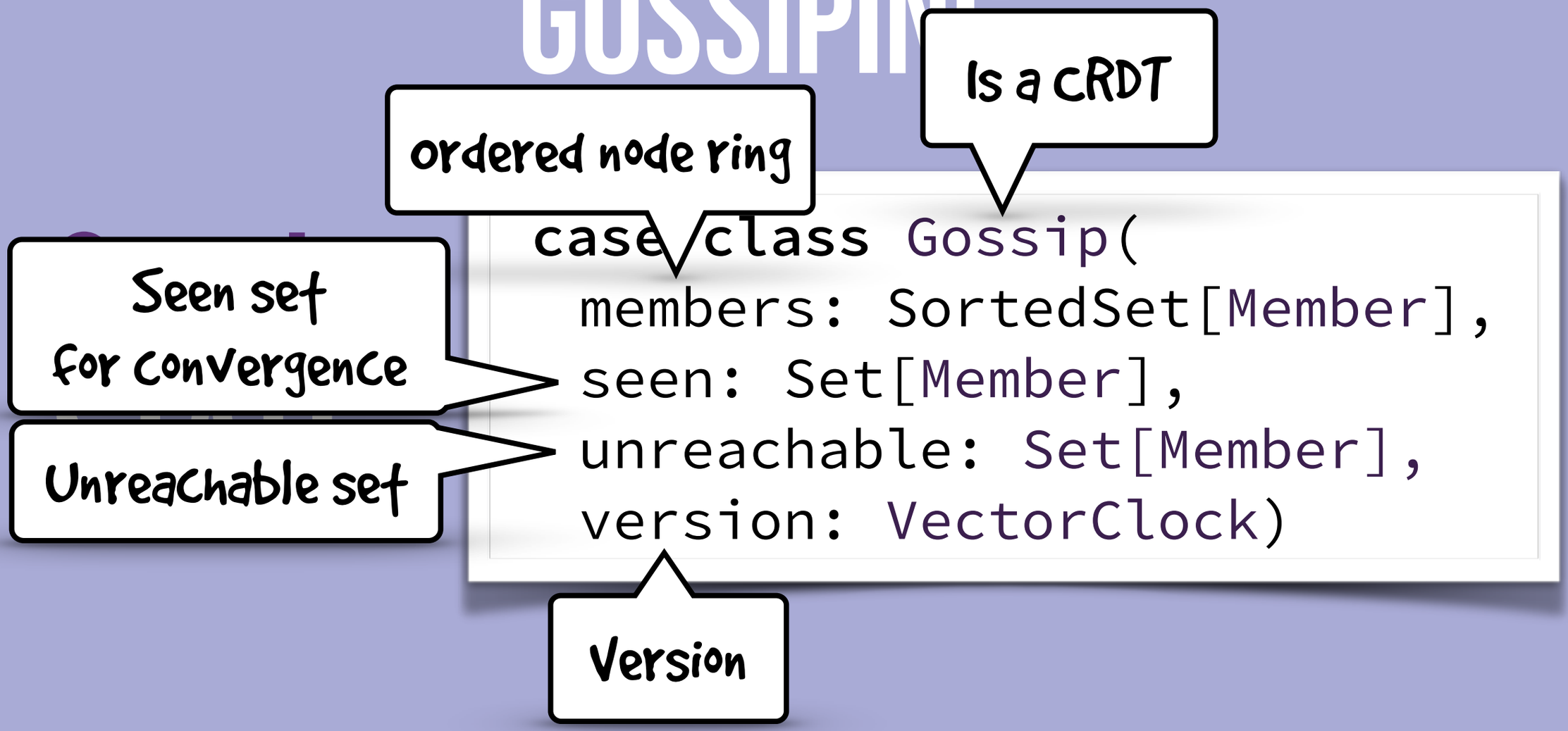
Unreachable set

```
case class Gossip(
  members: SortedSet[Member],
  seen: Set[Member],
  unreachable: Set[Member],
  version: VectorClock)
```

Version

1. Picks random node with older/newer version
2. Gossips in a request/reply fashion

GOSSIPING



1. Picks random node with older/newer version
2. Gossips in a request/reply fashion
3. Updates internal state and adds himself to ‘seen’ set

CLUSTER

CONVERGENCE

CLUSTER

CONVERGENCE

Reached when:

1. All nodes are represented in the SEEN set
2. No members are UNREACHABLE, or
3. All UNREACHABLE members have status DOWN or EXITING

BIASED GOSSIP

BIASED GOSSIP

80% bias to nodes not in seen table

Up to 400 nodes, then reduced

PUSH/PULL
GOSSIP

PUSH/PULL
GOSSIP
VARIATION

PUSH/PULL GOSSIP VARIATION

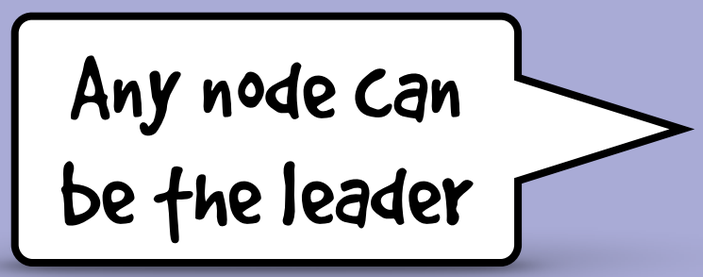
```
case class Status(version: VectorClock)
```

LEADER ROLE

LEADER ROLE

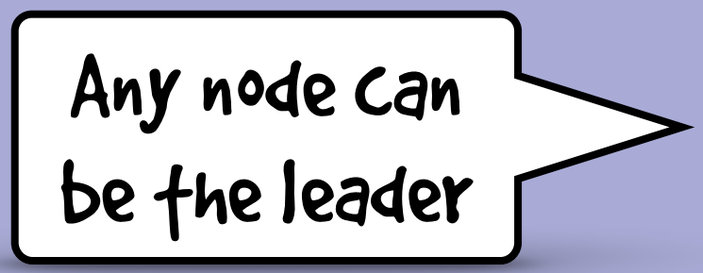
Any node can be the leader

LEADER ROLE



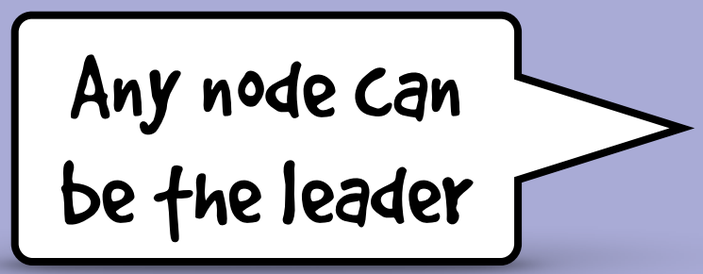
1. No election, but DETERMINISTIC

LEADER ROLE



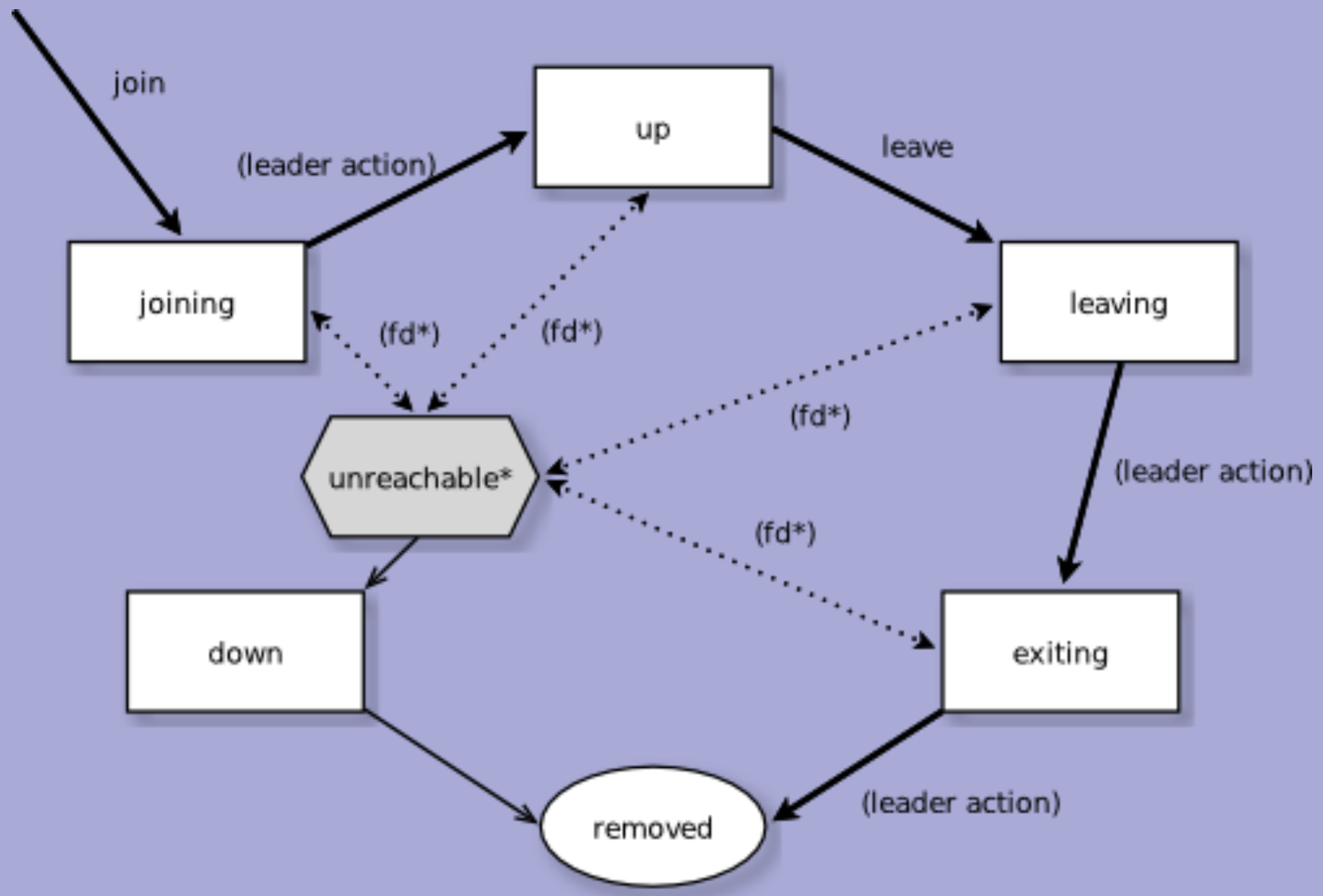
1. No election, but DETERMINISTIC
2. Can change after cluster CONVERGENCE

LEADER ROLE



1. No election, but DETERMINISTIC
2. Can change after cluster CONVERGENCE
3. Leader has special DUTIES

NODE LIFECYCLE IN AKKA



FAILURE DETECTION

FAILURE DETECTION

HASHES the node ring

PICKS 5 nodes

REQUEST/REPLY heartbeat

FAILURE DETECTION

To increase likelihood of bridging racks and data centers

HASHES the node ring

PICKS 5 nodes

REQUEST/REPLY heartbeat

FAILURE DETECTION

Used by

To increase likelihood of bridging racks and data centers

Cluster MEMBERSHIP HASHES the node ring

Remote DEATH WATCH PICKS 5 nodes

Remote SUPERVISION REQUEST/REPLY heartbeat

FAILURE DETECTION

Is an ACCRUAL Failure Detector

FAILURE DETECTION

Is an ACCRUAL Failure Detector

Does not help much in practice

FAILURE DETECTION

Is an ACCRUAL Failure Detector

Does not help much in practice

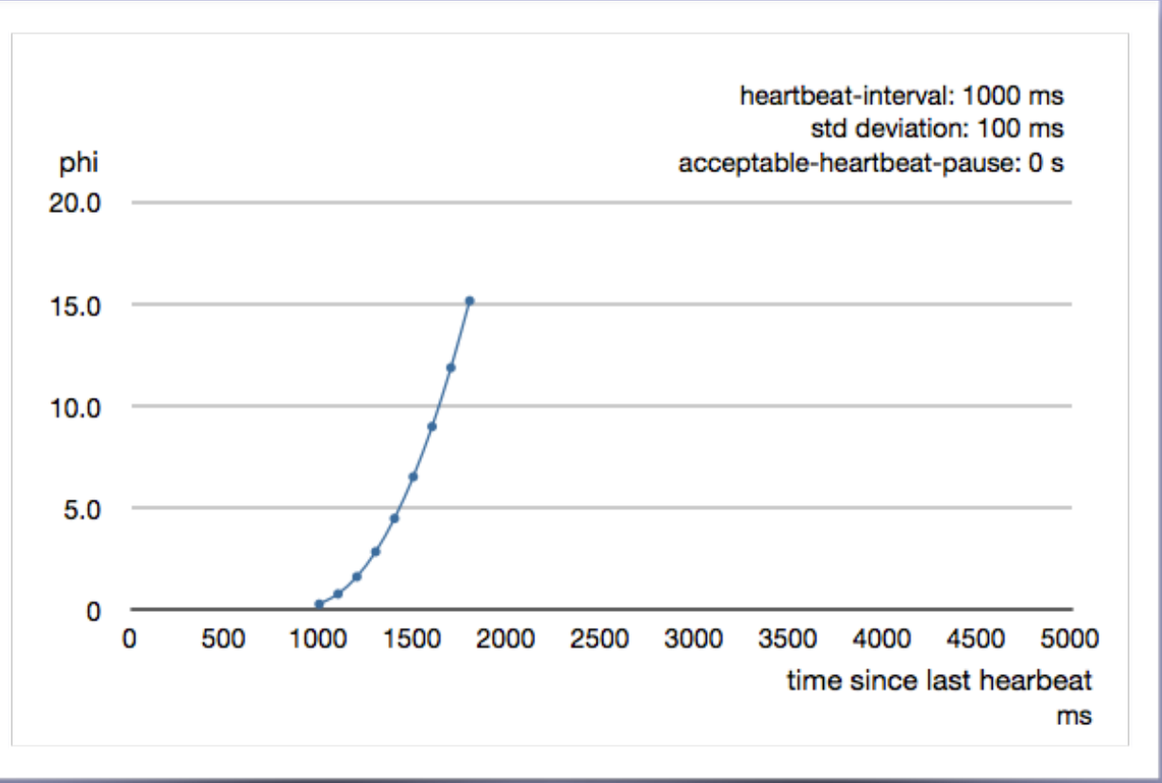
Need to add delay to deal with Garbage Collection

FAILURE DETECTION

Does not help much in practice

Is an ACCRUAL Failure Detector

Instead of this



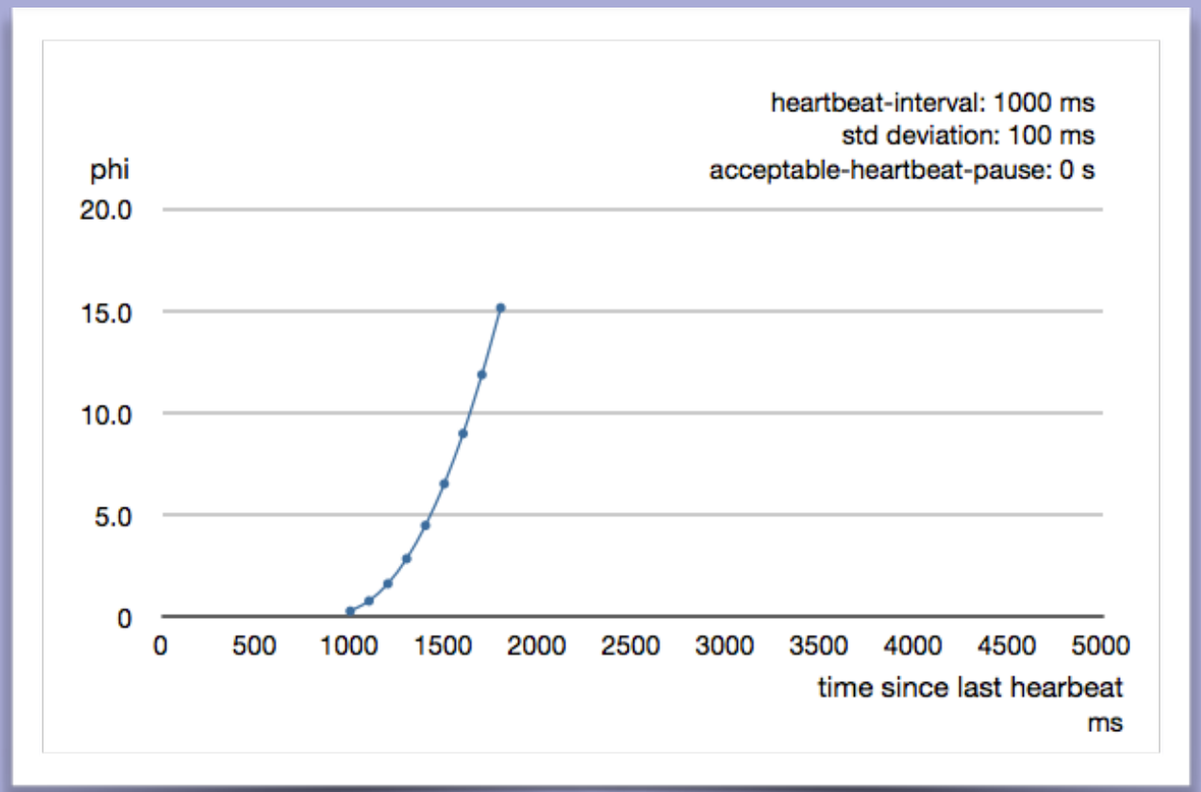
Need to add delay to deal with Garbage Collection

FAILURE DETECTION

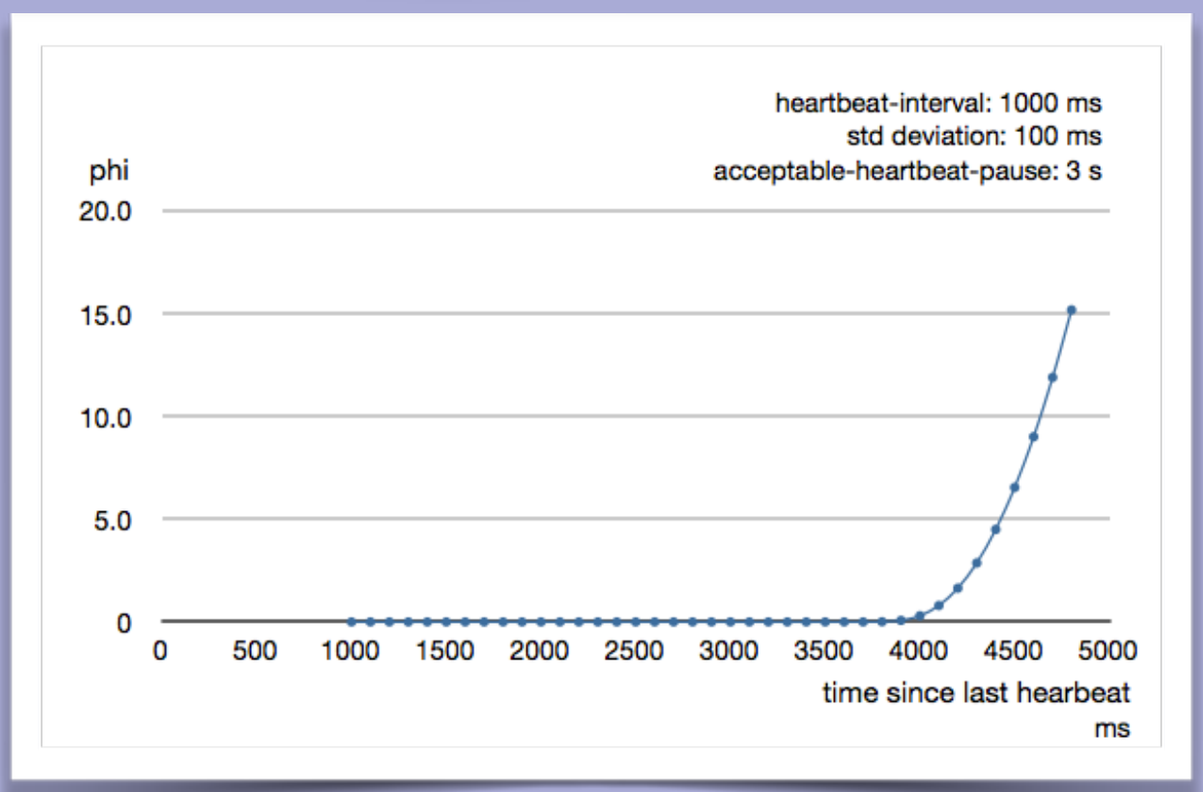
Is an ACCRUAL Failure Detector

Does not help much in practice

Instead of this



It often looks like this



Need to add delay to deal with Garbage Collection

NETWORK PARTITIONS

NETWORK PARTITIONS

- Failure Detector can mark an unavailable member UNREACHABLE

NETWORK PARTITIONS

- Failure Detector can mark an unavailable member UNREACHABLE
- If one node is UNREACHABLE then no cluster CONVERGENCE

NETWORK PARTITIONS

- Failure Detector can mark an unavailable member UNREACHABLE
- If one node is UNREACHABLE then no cluster CONVERGENCE
- This means that the LEADER can no longer perform it's duties

NETWORK PARTITIONS

- Failure Detector can mark an unavailable member UNREACHABLE
- If one node is UNREACHABLE CONVERGENCE
- This means that the LEADER can no longer perform it's duties

Split Brain

NETWORK PARTITIONS

- Failure Detector can mark an unavailable member UNREACHABLE
- If one node
- This means that the LEADER can no longer perform it's duties

Split Brain

- Member can come back from UNREACHABLE—Else:

NETWORK PARTITIONS

- Failure Detector can mark an unavailable member UNREACHABLE
- If one node
- This means that the LEADER can no longer perform it's duties

Split Brain

- Member can come back from UNREACHABLE—Else:
- The node needs to be marked as DOWN—Either through:

NETWORK PARTITIONS

- Failure Detector can mark an unavailable member UNREACHABLE
- If one node
- This means that the LEADER can no longer perform it's duties

Split Brain

- Member can come back from UNREACHABLE—Else:
- The node needs to be marked as DOWN—Either through:
  1. AUTO-DOWN
  2. MANUAL DOWN

POTENTIAL FUTURE OPTIMIZATIONS

POTENTIAL FUTURE OPTIMIZATIONS

- VECTOR CLOCK HISTORY PRUNING

POTENTIAL FUTURE OPTIMIZATIONS

- VECTOR CLOCK HISTORY PRUNING
- DELEGATED HEARTBEAT

POTENTIAL FUTURE OPTIMIZATIONS

- VECTOR CLOCK HISTORY PRUNING
- DELEGATED HEARTBEAT
- “REAL” PUSH/PULL GOSSIP

POTENTIAL FUTURE OPTIMIZATIONS

- VECTOR CLOCK HISTORY PRUNING
- DELEGATED HEARTBEAT
- "REAL" PUSH/PULL GOSSIP
- More OUT-OF-THE-BOX AUTO-DOWN patterns

AKKA MODULES FOR DISTRIBUTION

AKKA MODULES FOR DISTRIBUTION

Akka CLUSTER

Akka REMOTE

Akka HTTP

Akka IO

AKKA MODULES FOR DISTRIBUTION

Akka CLUSTER

Akka REMOTE

Akka HTTP

Akka IO

Clustered SINGLETON

Clustered ROUTERS

Clustered PUB/SUB

Cluster CLIENT

Consistent HASHING

...and BEYOND

AKKA & THE ROAD AHEAD

Akka HTTP

Akka STREAMS

Akka CRDT

Akka RAFT

AKKA & THE ROAD AHEAD

Akka HTTP

Akka STREAMS

Akka CRDT

Akka RAFT

AKKA & THE ROAD AHEAD

Akka HTTP

Akka 2.4

Akka STREAMS

Akka 2.4

Akka CRDT

Akka RAFT

AKKA & THE ROAD AHEAD

Akka HTTP

Akka 2.4

Akka STREAMS

Akka 2.4

Akka CRDT

?

Akka RAFT

AKKA & THE ROAD AHEAD

Akka HTTP

Akka 2.4

Akka STREAMS

Akka 2.4

Akka CRDT

?

Akka RAFT

?

EAGER

for more?

TRY AKKA OUT

akka.io

Join us at

React Conf

San Francisco

Nov 18–21

reactconf.com

Early Registration ends tomorrow

Join us at

React Conf

San Francisco

Nov 18–21

reactconf.com

REFERENCES

- **General Distributed Systems**
  - Summary of network reliability post-mortems—more terrifying than the most horrifying Stephen King novel: http://aphyr.com/posts/288-the-network-is-reliable
  - A Note on Distributed Computing: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.41.7628
  - On the problems with RPC: http://steve.vinoski.net/pdf/IEEE-Convenience_Over_Correctness.pdf
  - 8 Fallacies of Distributed Computing: https://blogs.oracle.com/jag/resource/Fallacies.html
  - 6 Misconceptions of Distributed Computing: www.dsg.cs.tcd.ie/~vjcahill/sigops98/papers/vogels.ps
  - Distributed Computing Systems—A Foundational Approach: http://www.amazon.com/Programming-Distributed-Computing-Systems-Foundational/dp/0262018985
  - Introduction to Reliable and Secure Distributed Programming: http://www.distributedprogramming.net/
  - Nice short overview on Distributed Systems: http://book.mixu.net/distsys/
  - Meta list of distributed systems readings: https://gist.github.com/macintux/6227368

REFERENCES

- **Actor Model**
  - Great discussion between Erik Meijer & Carl Hewitt or the essence of the Actor Model: http://channel9.msdn.com/Shows/Going+Deep/Hewitt-Meijer-and-Szyperski-The-Actor-Model-everything-you-wanted-to-know-but-were-afraid-to-ask
  - Carl Hewitt's 1973 paper defining the Actor Model: http://worrydream.com/refs/Hewitt-ActorModel.pdf
  - Gul Agha's Doctoral Dissertation: https://dspace.mit.edu/handle/1721.1/6952

REFERENCES

- **FLP**
  - Impossibility of Distributed Consensus with One Faulty Process: http://cs-www.cs.yale.edu/homes/arvind/cs425/doc/fischer.pdf
  - A Brief Tour of FLP: http://the-paper-trail.org/blog/a-brief-tour-of-flp-impossibility/
- **CAP**
  - Brewer's Conjecture and the Feasibility of Consistent, Available, Partition-Tolerant Web Services: http://lpd.epfl.ch/sgilbert/pubs/BrewersConjecture-SigAct.pdf
  - You Can't Sacrifice Partition Tolerance: http://codahale.com/you-cant-sacrifice-partition-tolerance/
  - Linearizability: A Correctness Condition for Concurrent Objects: http://courses.cs.vt.edu/~cs5204/fall07-kafura/Papers/TransactionalMemory/Linearizability.pdf
  - CAP Twelve Years Later: How the "Rules" Have Changed: http://www.infoq.com/articles/cap-twelve-years-later-how-the-rules-have-changed
  - Consistency vs. Availability: http://www.infoq.com/news/2008/01/consistency-vs-availability

REFERENCES

- **Time & Order**
  - Post on the problems with Last Write Wins in Riak: http://aphyr.com/posts/285-call-me-maybe-riak
  - Time, Clocks, and the Ordering of Events in a Distributed System: http://research.microsoft.com/en-us/um/people/lamport/pubs/time-clocks.pdf
  - Vector Clocks: http://zoo.cs.yale.edu/classes/cs426/2012/lab/bib/fidge88timestamps.pdf
- **Failure Detection**
  - Unreliable Failure Detectors for Reliable Distributed Systems: http://www.cs.utexas.edu/~lorenzo/corsi/cs380d/papers/p225-chandra.pdf
  - The ϕ Accrual Failure Detector: http://ddg.jaist.ac.jp/pub/HDY+04.pdf
  - SWIM Failure Detector: http://www.cs.cornell.edu/~asdas/research/dsn02-swim.pdf
  - Practical Byzantine Fault Tolerance: http://www.pmg.lcs.mit.edu/papers/osdi99.pdf

REFERENCES

- **Transactions**
  - Jim Gray's classic book: http://www.amazon.com/Transaction-Processing-Concepts-Techniques-Management/dp/1558601902
  - Highly Available Transactions: Virtues and Limitations: http://www.bailis.org/papers/hat-vldb2014.pdf
  - Bolt on Consistency: http://db.cs.berkeley.edu/papers/sigmod13-bolton.pdf
  - Calvin: Fast Distributed Transactions for Partitioned Database Systems: http://cs.yale.edu/homes/thomson/publications/calvin-sigmod12.pdf
  - Spanner: Google's Globally-Distributed Database: http://research.google.com/archive/spanner.html
  - Life beyond Distributed Transactions: an Apostate's Opinion https://cs.brown.edu/courses/cs227/archives/2012/papers/weaker/cidr07p15.pdf
  - Immutability Changes Everything—Pat Hellands talk at Ricon: http://vimeo.com/52831373
  - Unschackle Your Domain (Event Sourcing): http://www.infoq.com/presentations/greg-young-unshackle-qcon08
  - CQRS: http://martinfowler.com/bliki/CQRS.html

REFERENCES

- **Consensus**
  - Paxos Made Simple: http://research.microsoft.com/en-us/um/people/lamport/pubs/paxos-simple.pdf
  - Paxos Made Moderately Complex: http://www.cs.cornell.edu/courses/cs7412/2011sp/paxos.pdf
  - A simple totally ordered broadcast protocol (ZAB): labs.yahoo.com/files/ladis08.pdf
  - In Search of an Understandable Consensus Algorithm (Raft): https://ramcloud.stanford.edu/wiki/download/attachments/11370504/raft.pdf
  - Replication strategy comparison diagram: http://snarfed.org/transactions_across_datacenters_io.html
  - Distributed Snapshots: Determining Global States of Distributed Systems: http://www.cs.swarthmore.edu/~newhall/readings/snapshots.pdf

REFERENCES

- **Eventual Consistency**
  - Dynamo: Amazon's Highly Available Key-value Store: http://www.read.seas.harvard.edu/~kohler/class/cs239-w08/decandia07dynamo.pdf
  - Consistency vs. Availability: http://www.infoq.com/news/2008/01/consistency-vs-availability
  - Consistent Hashing and Random Trees: http://thor.cs.ucsb.edu/~ravenben/papers/coreos/kll+97.pdf
  - PBS: Probabilistically Bounded Staleness: http://pbs.cs.berkeley.edu/

REFERENCES

- **Epidemic Gossip**
  - Chord: A Scalable Peer-to-peer Lookup Service for Internet
  - Applications: http://pdos.csail.mit.edu/papers/chord:sigcomm01/chord_sigcomm.pdf
  - Gossip-style Failure Detector: http://www.cs.cornell.edu/home/rvr/papers/GossipFD.pdf
  - GEMS: http://www.hcs.ufl.edu/pubs/GEMS2005.pdf
  - Efficient Reconciliation and Flow Control for Anti-Entropy Protocols: http://www.cs.cornell.edu/home/rvr/papers/flowgossip.pdf
  - 2400 Akka nodes on GCE: http://typesafe.com/blog/running-a-2400-akka-nodes-cluster-on-google-compute-engine
  - Starting 1000 Akka nodes in 4 min: http://typesafe.com/blog/starting-up-a-1000-node-akka-cluster-in-4-minutes-on-google-compute-engine
  - Push Pull Gossiping: http://khambatti.com/mujtaba/ArticlesAndPapers/pdpta03.pdf
  - SWIM: Scalable Weakly-consistent Infection-style Process Group Membership Protocol: http://www.cs.cornell.edu/~asdas/research/dsn02-swim.pdf

REFERENCES

- **Conflict-Free Replicated Data Types (CRDTs)**
  - A comprehensive study of Convergent and Commutative Replicated Data Types: http://hal.upmc.fr/docs/00/55/55/88/PDF/techreport.pdf
  - Mark Shapiro talks about CRDTs at Microsoft: http://research.microsoft.com/apps/video/dl.aspx?id=153540
  - Akka CRDT project: https://github.com/jboner/akka-crdt
- **CALM**
  - Dedalus: Datalog in Time and Space: http://db.cs.berkeley.edu/papers/datalog2011-dedalus.pdf
  - CALM: http://www.cs.berkeley.edu/~palvaro/cidr11.pdf
  - Logic and Lattices for Distributed Programming: http://db.cs.berkeley.edu/papers/UCB-lattice-tr.pdf
  - Bloom Language website: http://bloom-lang.net
  - Joe Hellerstein talks about CALM: http://vimeo.com/53904989

REFERENCES

- **Akka Cluster**
  - My Akka Cluster Implementation Notes: https://gist.github.com/jboner/7692270
  - Akka Cluster Specification: http://doc.akka.io/docs/akka/snapshot/common/cluster.html
  - Akka Cluster Docs: http://doc.akka.io/docs/akka/snapshot/scala/cluster-usage.html
  - Akka Failure Detector Docs: http://doc.akka.io/docs/akka/snapshot/scala/remoting.html#Failure_Detector
  - Akka Roadmap: https://docs.google.com/a/typesafe.com/document/d/18W9-fKs55wiFNjXL9q50PYOnR7-nnsImzJqHOPPbM4E/mobilebasic?pli=1&hl=en_US
  - Where Akka Came From: http://letitcrash.com/post/40599293211/where-akka-came-from

ANY Questions?

The ROAD to AKKA CLUSTER and BEYOND...

JONAS BONÉR
CTO TYPESAFE
@JBONER