BUILDING DISTRIBUTED SYSTEMS WITH RIAK CORE

Steve Vinoski

Basho Technologies

http://basho.com/

Cambridge, MA USA

@stevevinoski

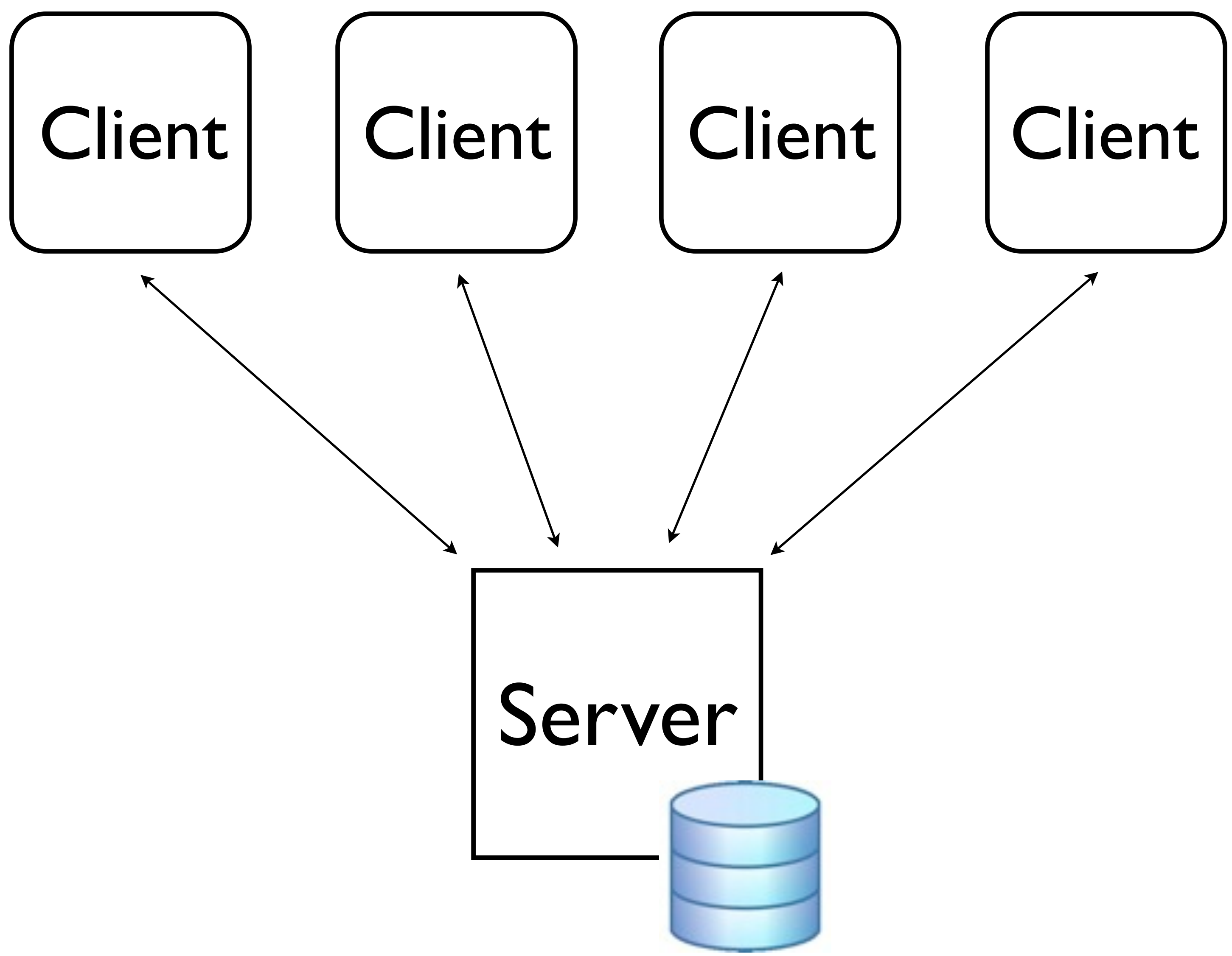
vinoski@ieee.org

http://steve.vinoski.net/

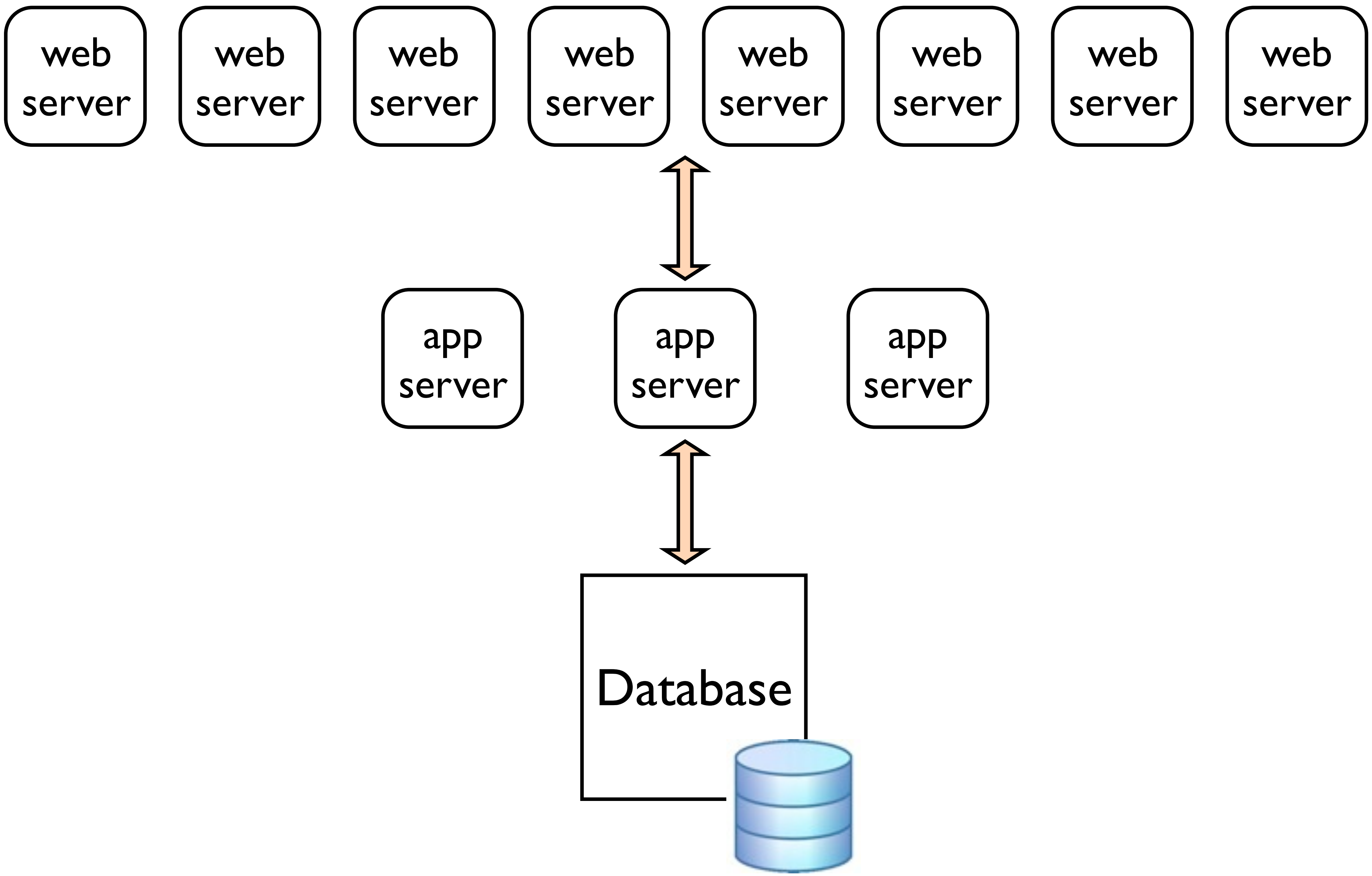
INTERNATIONAL SOFTWARE DEVELOPMENT CONFERENCE

gotocon.com

20 Years Ago: Client-Server



Early-ish Web Apps



Scaling Up

- Scaling up meant getting bigger boxes
- Worked for client/server and early web apps
- But couldn't keep up with web growth

Scaling Out

- As businesses went from “having” websites to “being” websites:
  - increasing number of commodity boxes
  - eventually across multiple data centers

Scaling Out Changed Everything

- More concurrency, more distribution, more replication, more latency, more consistency issues
- And more operational issues
- As well as more system failures
- While also needing higher reliability and uptime

CAP Theorem

- A conjecture put forth in 2000 by Dr. Eric Brewer
- Formally proven in 2002
- A distributed system can never completely guarantee these three properties:
  - Consistency
  - Availability
  - Partition tolerance

Partition Tolerance

- Guarantees continued system operation even when the network breaks and messages are lost
- When—**not if**—a partition occurs, choose between C and A

Consistency

- Distributed nodes see the same updates at the same logical time
- Hard to guarantee across a distributed system
- **Any** replication introduces consistency vs. latency issues

Availability

- Guarantees the system will service every read and write sent to it
- Even when things are breaking

Choosing AP

- Provides read/write availability even when network breaks or nodes die
- Provides <u>eventual consistency</u>
- Example: Domain Name System (DNS) is an AP system

Example AP Systems

- Amazon Dynamo
- Cassandra
- CouchDB
- Voldemort
- Basho Riak

PACELC

PACELC

- Work by Daniel Abadi of Yale University to augment CAP

PACELC

- Work by Daniel Abadi of Yale University to augment CAP
- When **P**artitioned, trade off **A**vailability and **C**onsistency

PACELC

- Work by Daniel Abadi of Yale University to augment CAP
- When **P**artitioned, trade off **A**vailability and **C**onsistency
- **E**lse

PACELC

- Work by Daniel Abadi of Yale University to augment CAP
- When **P**artitioned, trade off **A**vailability and **C**onsistency
- **E**lse
- Trade off **L**atency and **C**onsistency

Handling Tradeoffs for AP Systems

Assumptions

- We want to scale out
- We have a networked cluster of nodes, each with local storage
- We're choosing availability over consistency when partitions occur

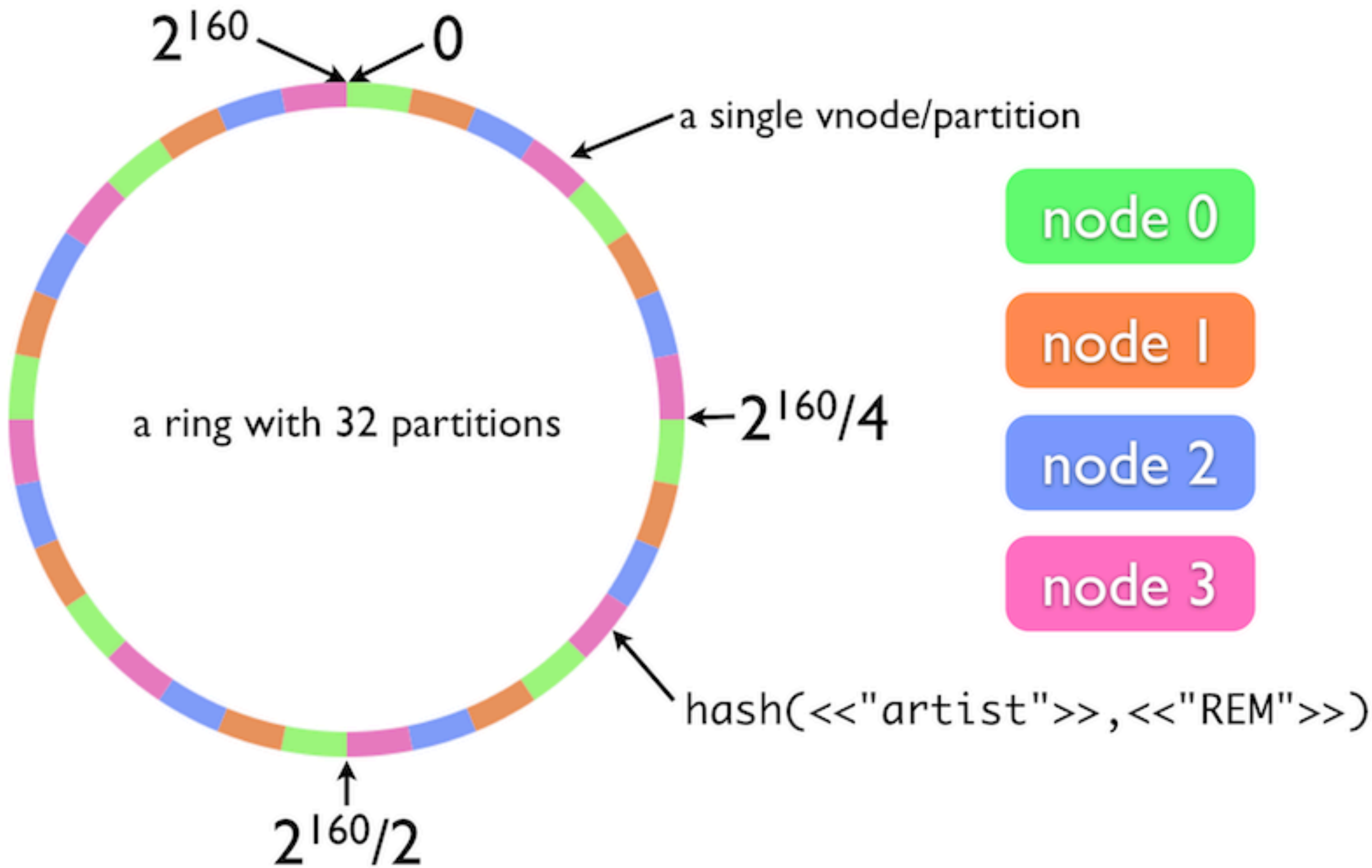
- Problem: how to make the system available even if nodes die or the network breaks?
- Solution:
  - allow reading and writing from multiple nodes in the system
  - avoid master nodes, instead make all nodes peers

- Problem: if multiple nodes are involved, how do you reliably know where to read or write?
- Solution:
  - assign virtual nodes (vnodes) to physical nodes
  - use consistent hashing to find vnodes for reads/writes

Node vs. Vnode

- Vnode: Erlang process managing a ring partition
- Node: physical machine that hosts vnodes
- Vnodes / node = (ring size) / (node count)

Consistent Hashing



Consistent Hashing and Multiple Vnode Benefits

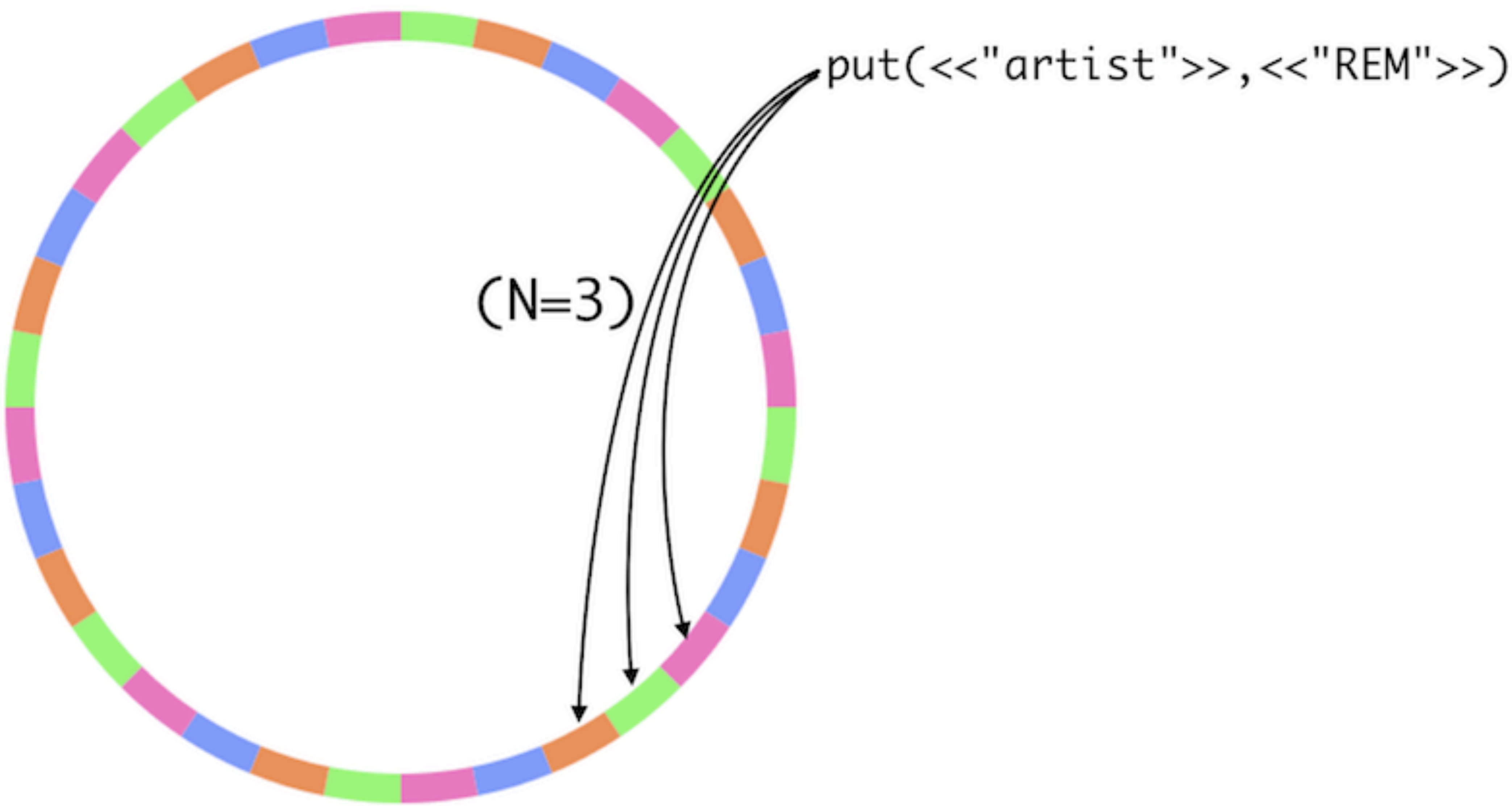
- Data is stored in multiple locations
- Loss of a node means only a single replica is lost
- No master to lose
- Adding nodes is trivial, data gets rebalanced minimally and automatically

- Problem: what about availability? What if the node you write to dies or becomes inaccessible?
- Solution: sloppy quorums (as opposed to strict quorums)
  - write to multiple vnodes
  - attempt reads from multiple vnodes

N/R/W Values

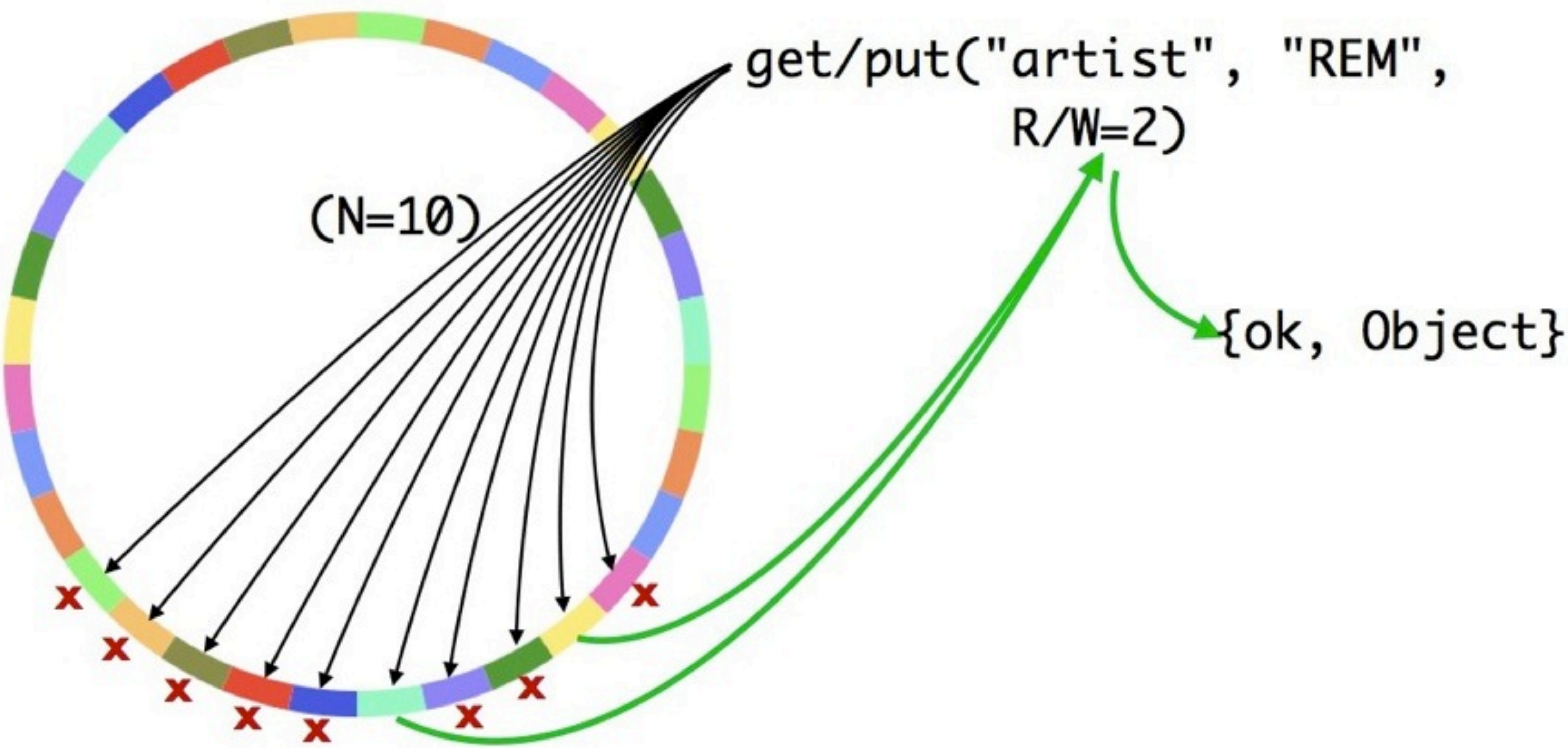
- N = number of replicas to store (on distinct nodes)
- R = read quorum = number of replica responses needed for a successful read (specified per-request)
- W = write quorum = number of replica responses needed for a successful write (specified per-request)

N/R/W Values



- Problem: what happens if a key hashes to vnodes that aren't available?
- Solution:
  - read from or write to the next available vnode (hence "sloppy" not "strict" quorums)
  - eventually repair via hinted handoff

N/R/W Values



Hinted Handoff

- Fallback vnode holds data for unavailable actual vnode
- Fallback vnode keeps checking for availability of actual vnode
- Once actual vnode becomes available, fallback hands off data to it

Quorum Benefits

- Allows applications to tune consistency, availability, reliability per read or write

- Problem: how do the nodes in the ring keep track of ring state?

- Solution: gossip protocol

Gossip Protocol

- Nodes "gossip" their view of the state of the ring to other nodes
- If a node changes its claim on the ring, it lets others know
- The overall state of the ring is kept consistent among all nodes in the ring without needing a master

- Problem: what happens if vnode replicas get out of sync?
- Solution:
    - vector clocks
    - read repair

- Problem: what happens if vnode replicas get out of sync?
- Solution:
  - vector clocks
  - read repair

Vector Clocks

- Reasoning about time and causality in distributed systems is hard
- Integer timestamps don't necessarily capture causality
- Vector clocks provide a happens-before relationship between two events

Vector Clocks

- Simple data structure: [{ActorID,Counter}]
- All data has an associated vector clock, actors update their entry when making changes
- ClockA happened-before ClockB if all actor-counters in A are less than or equal to those in B

Vector Clocks are Easy

- Bryan Fink's blog post: http://basho.com/blog/technical/2010/01/29/why-vector-clocks-are-easy/
- Explains vector clocks using a dinner invitation example

Dinner Example

- Alice, Ben, Cathy, Dave exchange some email to decide when to meet for dinner
- Alice emails everyone to suggest Wednesday

Dinner Example

- Ben and Dave email each other and decide Tuesday
- Cathy and Dave email each other and Cathy prefers Thursday, and Dave changes his mind and agrees

Dinner Example

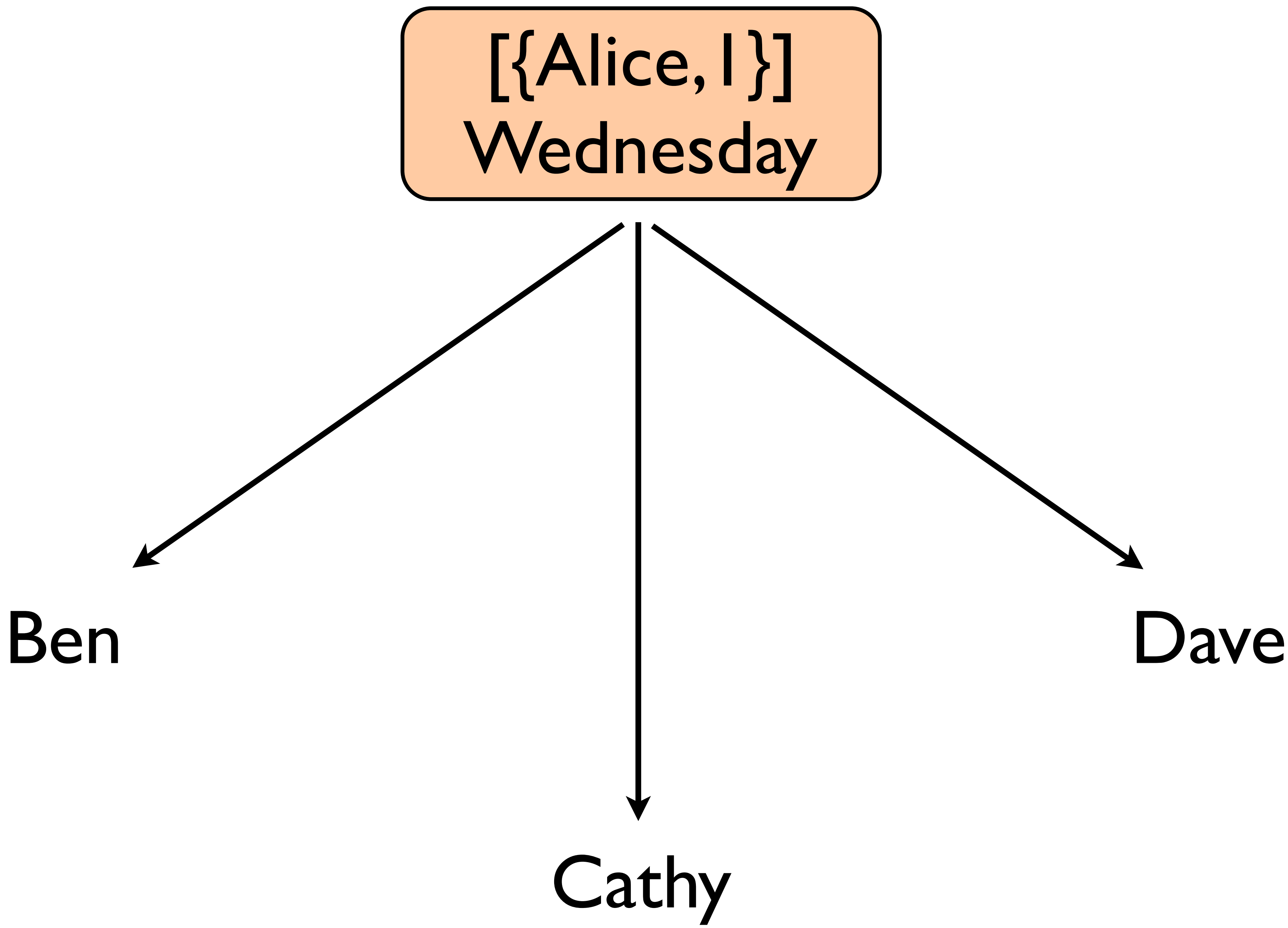
- Alice then pings everyone to check that Wednesday is still OK
- Ben says he and Dave prefer Tuesday
- Cathy says she and Dave prefer Thursday
- Dave doesn't answer

Dinner Example

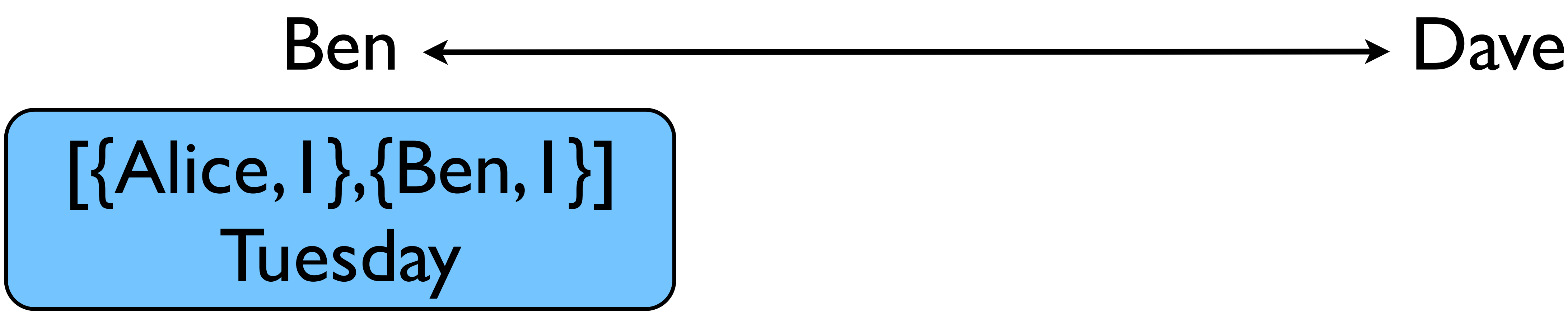
- Alice then pings everyone to check that Wednesday is still OK
- Ben says he and Dave prefer Tuesday
- Cathy says she and Dave prefer Thursday
- Dave doesn't answer

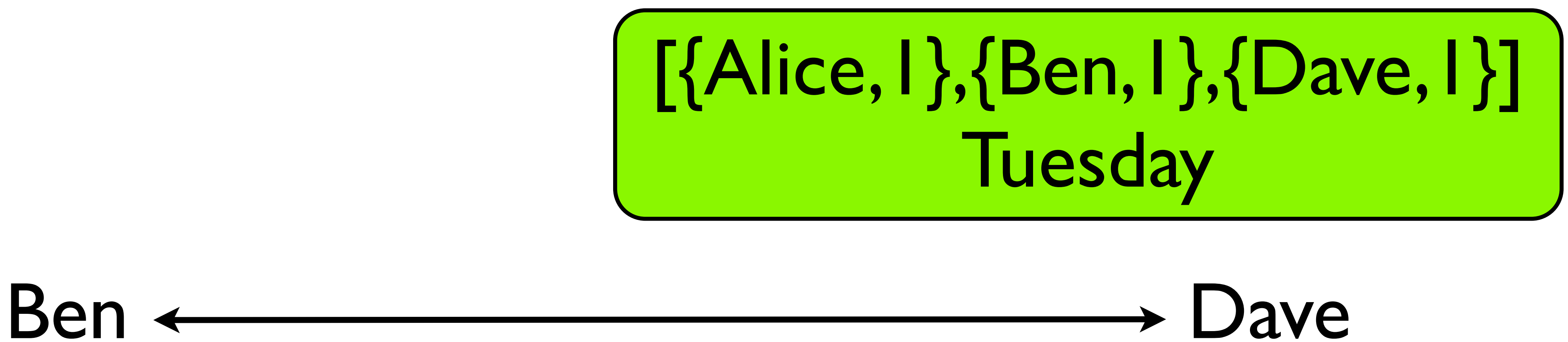
Conflict!

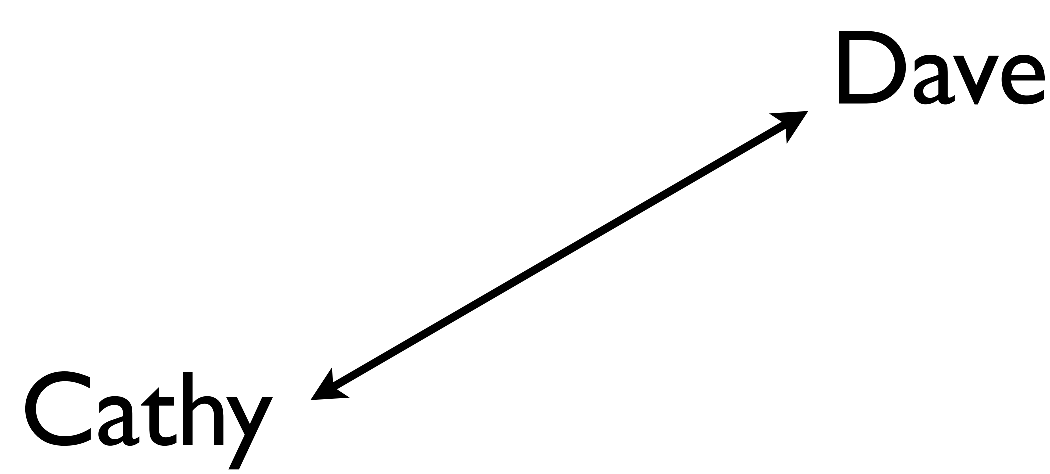
[{Alice,1}]
Wednesday

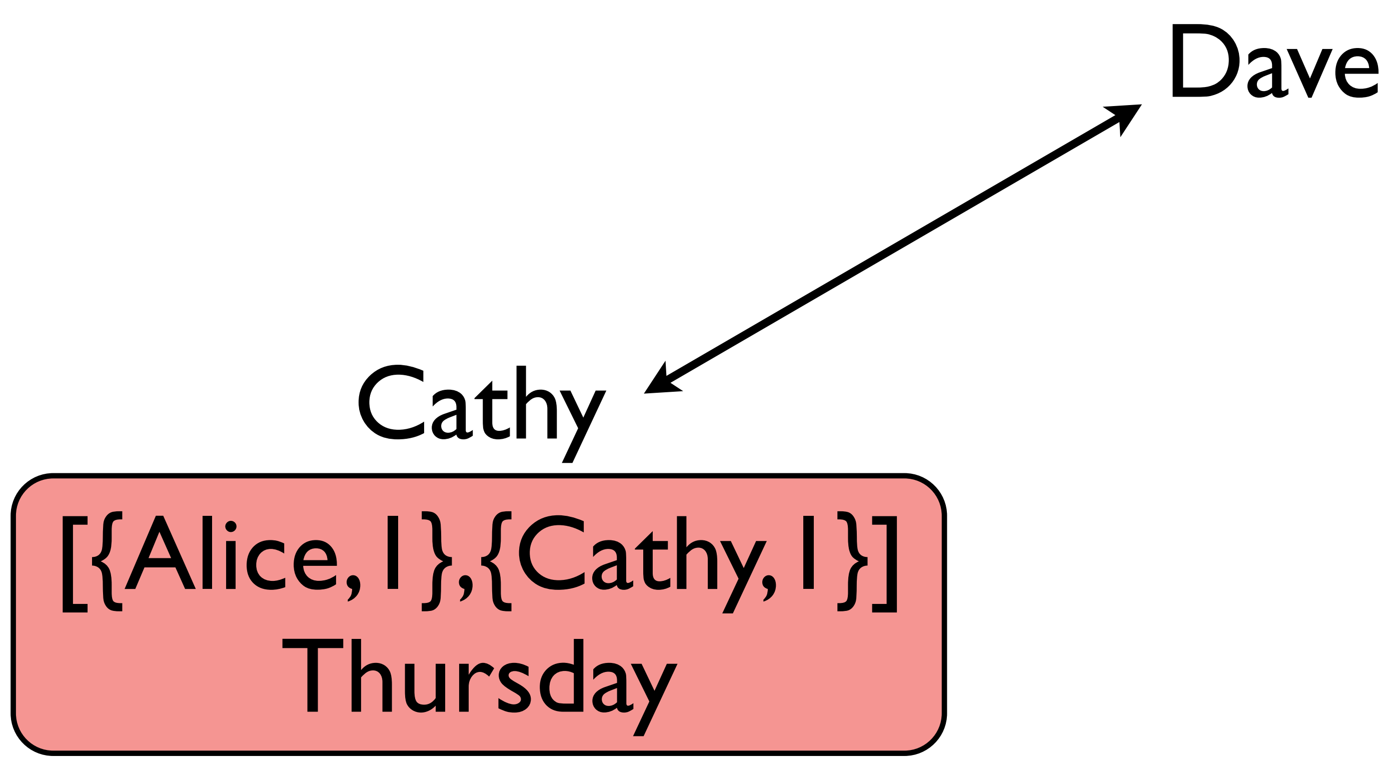

[{Alice,1}]
Wednesday
Ben
Cathy
Dave

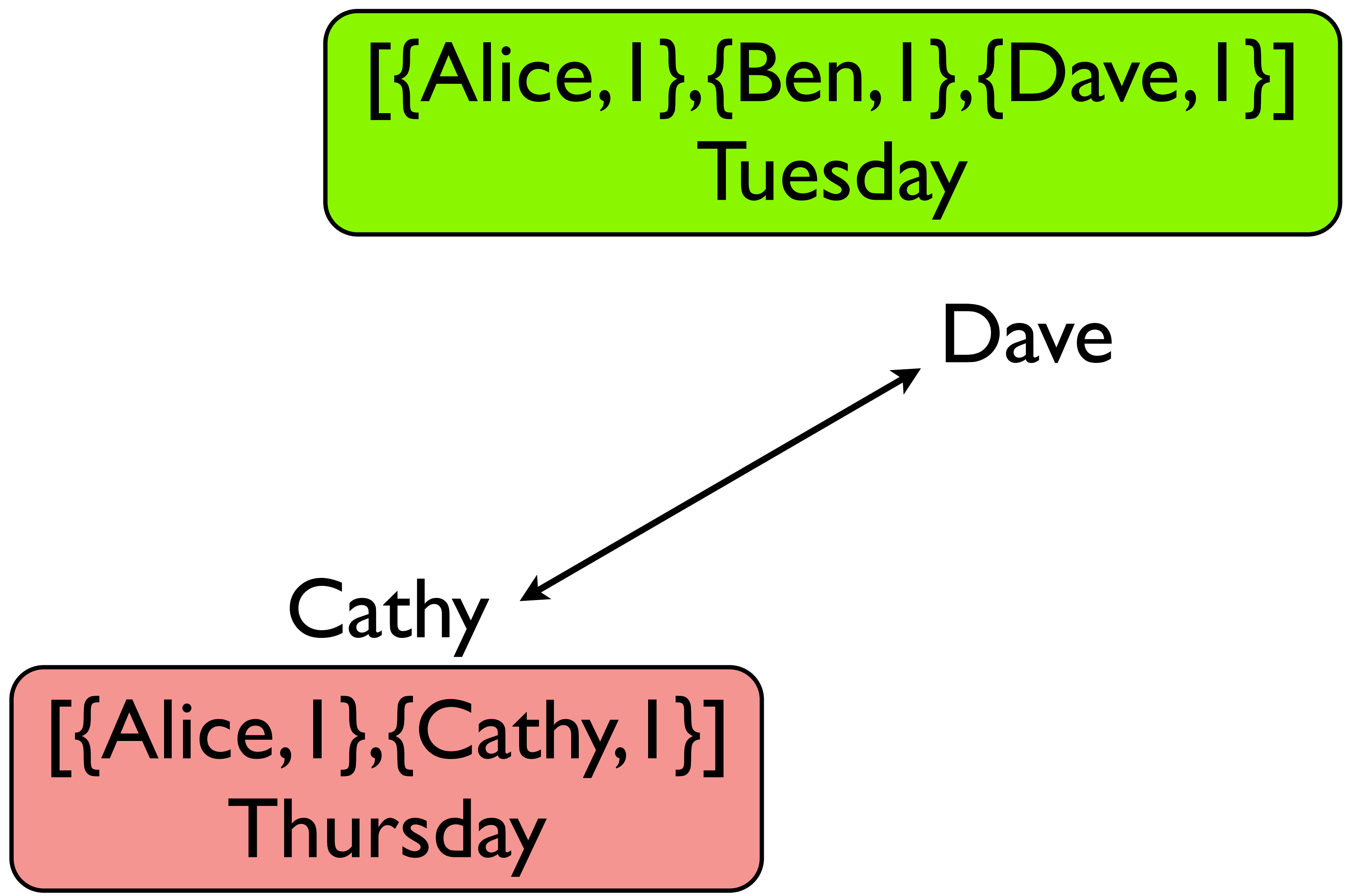
Ben ⟷ Dave

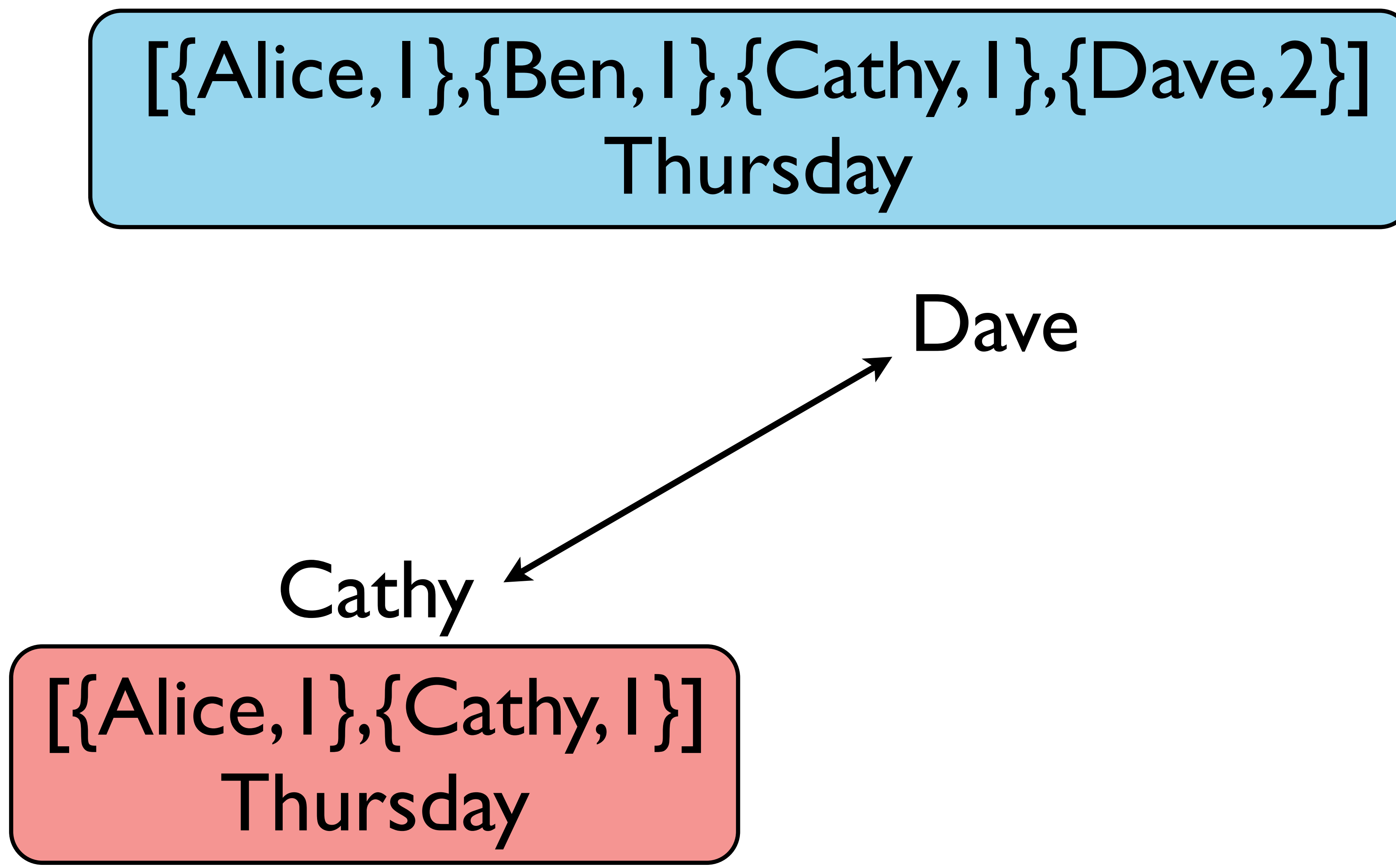

Ben
Dave
[{Alice,1},{Ben,1}]
Tuesday


[{Alice,1},{Ben,1},{Dave,1}]
Tuesday
Ben
Dave

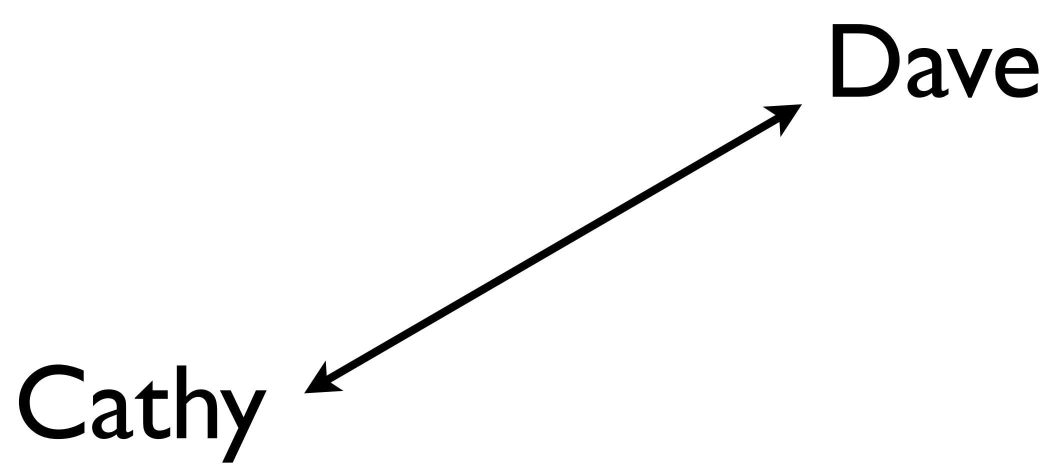

Dave
Cathy

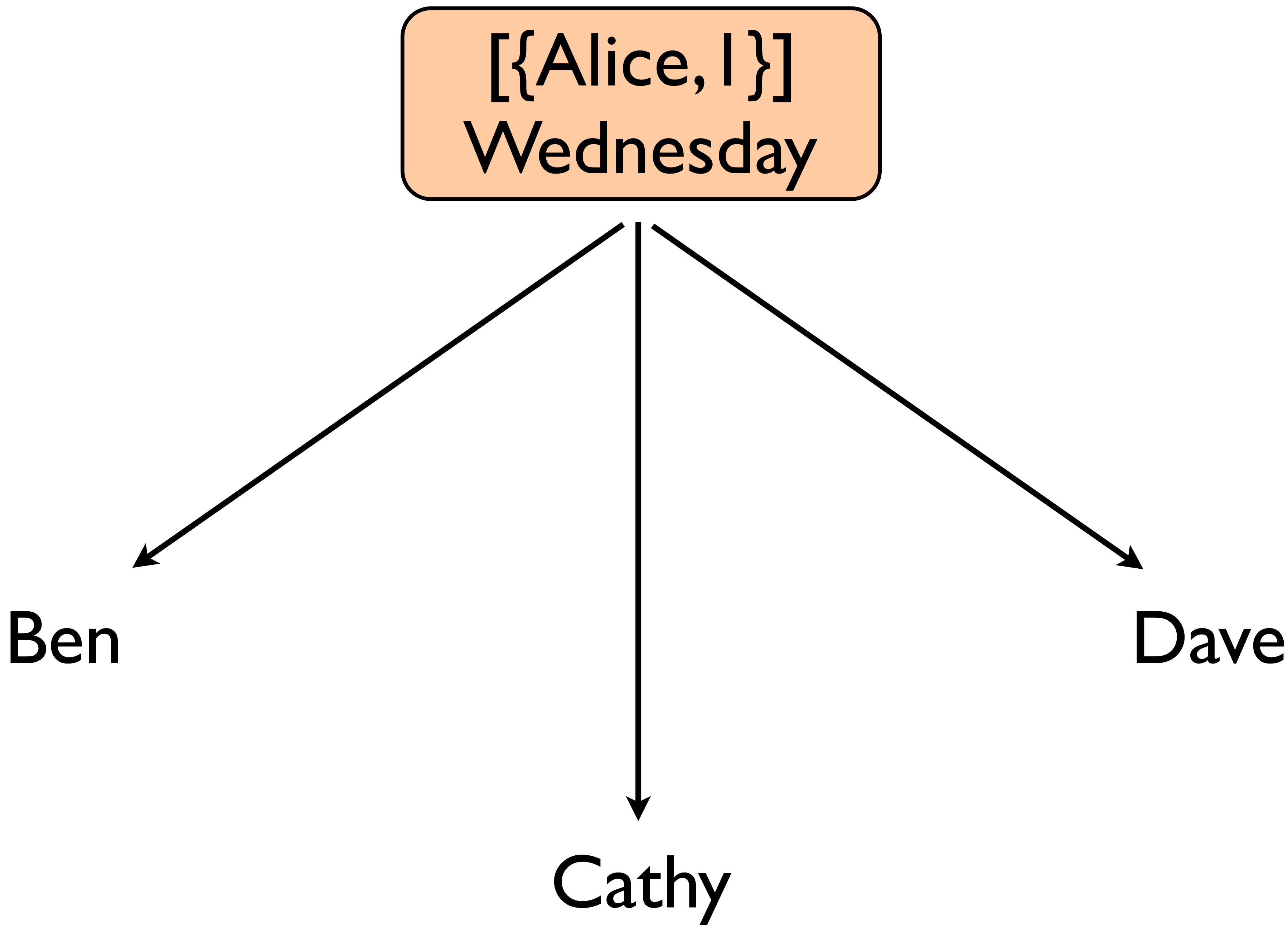

Dave
Cathy
[{Alice,1},{Cathy,1}]
Thursday

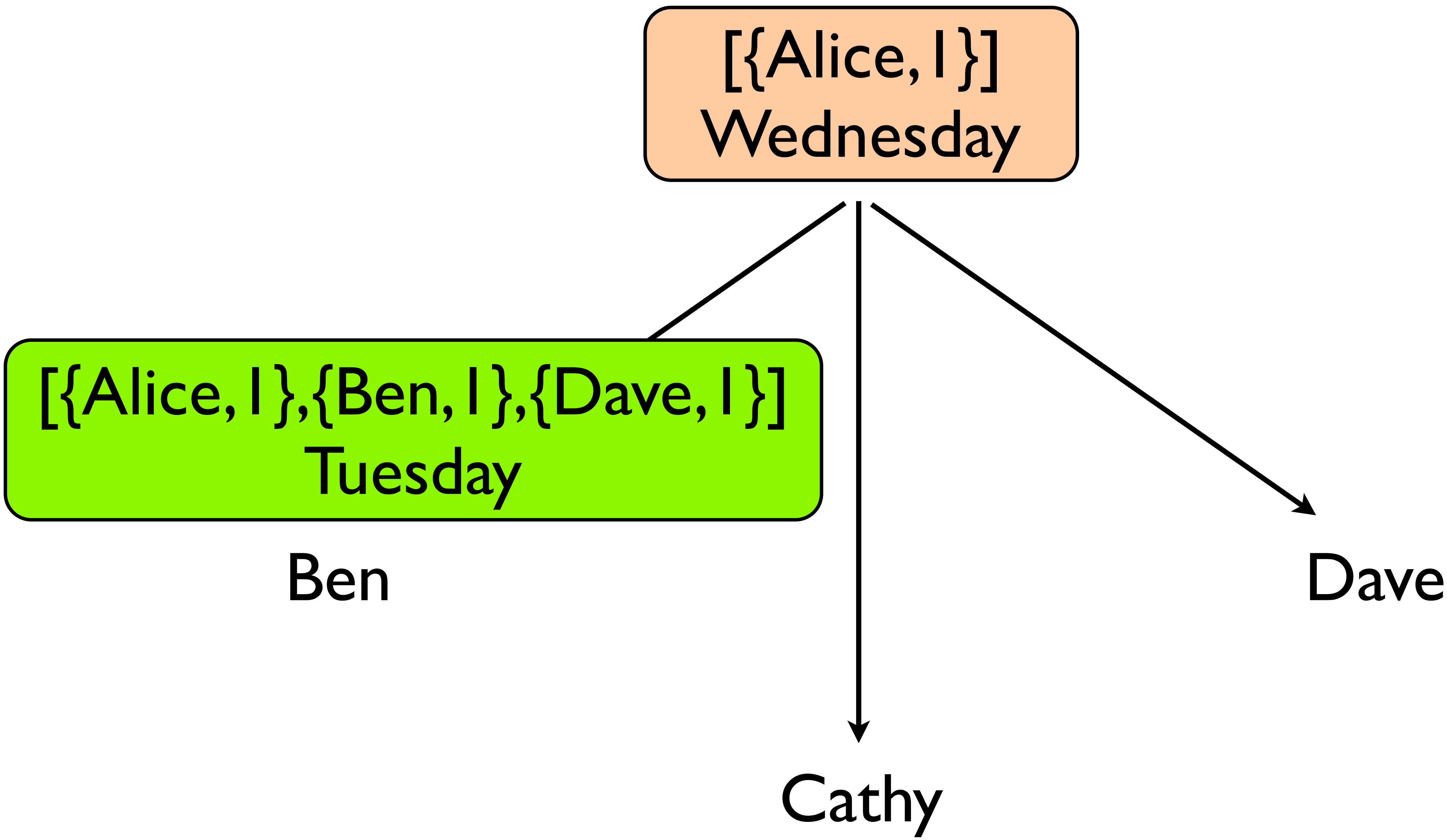

[{Alice,1},{Ben,1},{Dave,1}]
Tuesday
Dave
Cathy
[{Alice,1},{Cathy,1}]
Thursday

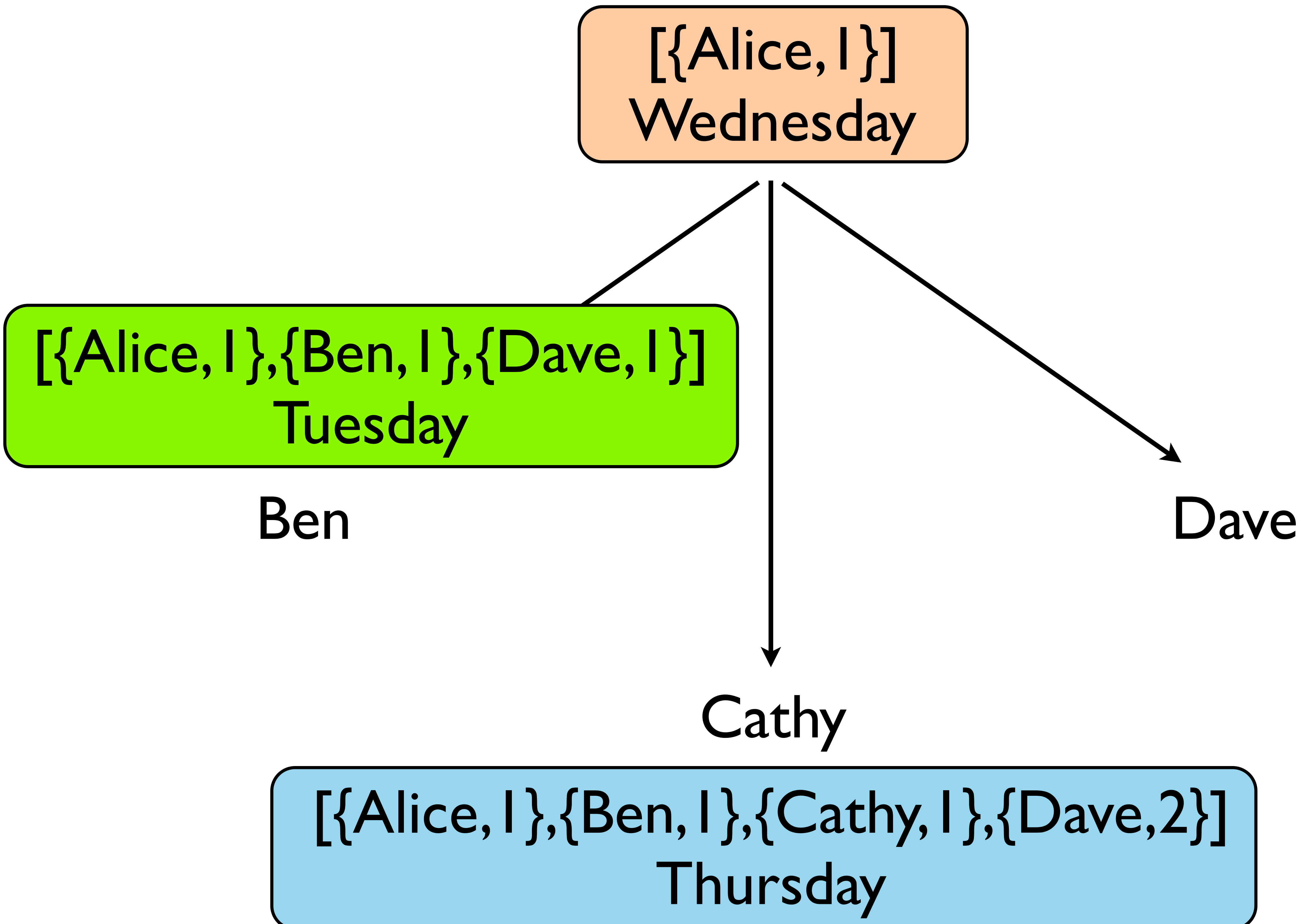

[{Alice,1},{Ben,1},{Cathy,1},{Dave,2}]
Thursday
Dave
Cathy
[{Alice,1},{Cathy,1}]
Thursday

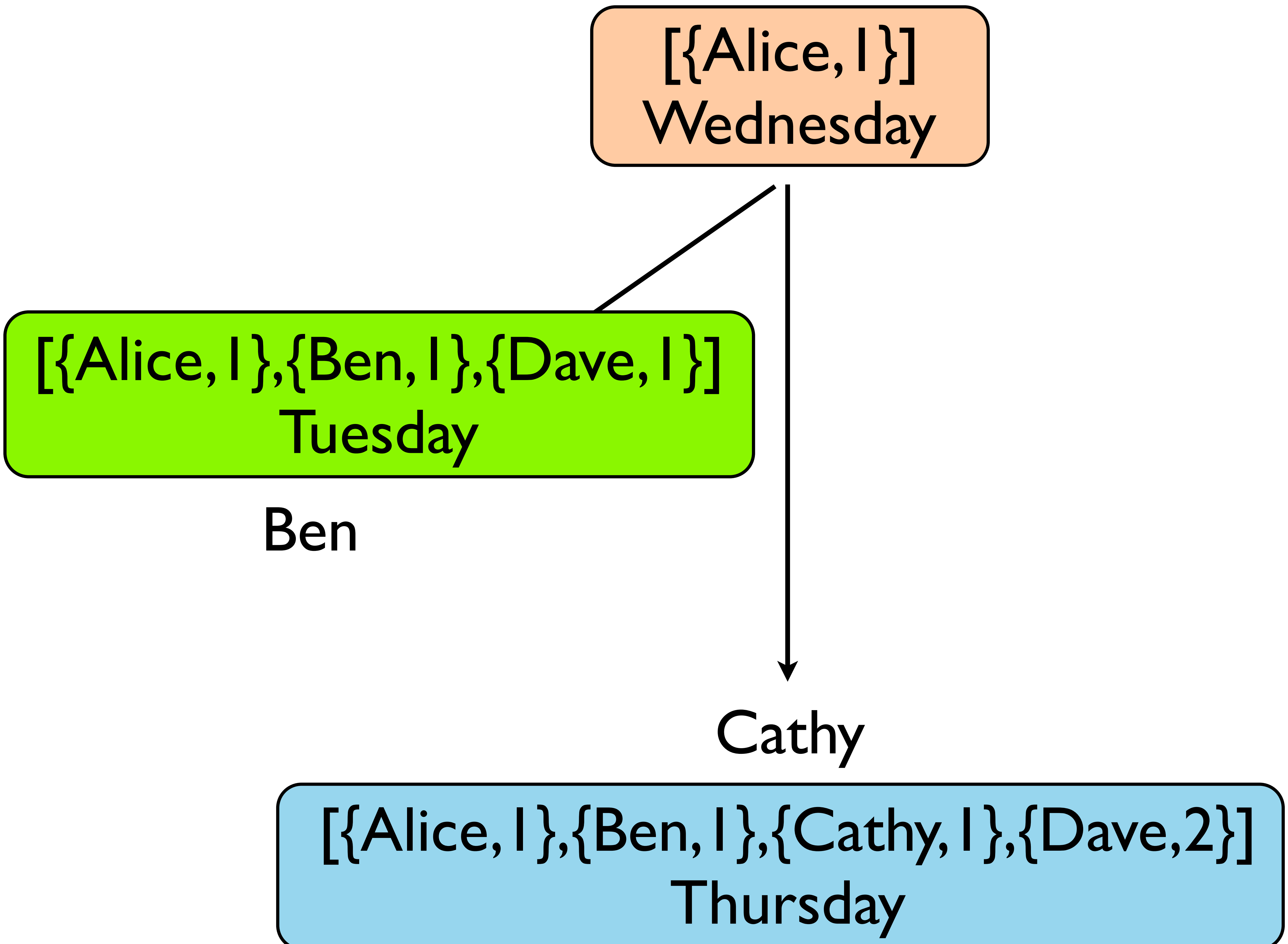
[{Alice,1},{Ben,1},{Cathy,1},{Dave,2}]
Thursday




[{Alice,1}]
Wednesday
Ben
Cathy
Dave


[{Alice,1}]
Wednesday
[{Alice,1},{Ben,1},{Dave,1}]
Tuesday
Ben
Dave
Cathy


[{Alice,1}]
Wednesday
[{Alice,1},{Ben,1},{Dave,1}]
Tuesday
Ben
Dave
Cathy
[{Alice,1},{Ben,1},{Cathy,1},{Dave,2}]
Thursday


[{Alice,1}]
Wednesday
[{Alice,1},{Ben,1},{Dave,1}]
Tuesday
Ben
Cathy
[{Alice,1},{Ben,1},{Cathy,1},{Dave,2}]
Thursday

[{Alice,1},{Ben,1},{Cathy,1},{Dave,2}]
Thursday

[{Alice,1},{Ben,1},{Cathy,1},{Dave,2}]
Thursday

See: Easy!

Vector Clocks are Hard

- Justin Sheehy's blog post: http://basho.com/blog/technical/2010/04/05/why-vector-clocks-are-hard/

basho

Vector Clocks are Hard

- Our example shows how vclocks can quickly grow
- Tradeoffs to keep them bounded:
  - mark each entry with a timestamp
  - occasionally drop old entries
  - also trim vclock if too many entries

- Problem: what happens if vnode replicas get out of sync?
- Solution:
  - vector clocks
  - read repair

Read Repair

- If a read detects that a vnode has stale data, it is repaired via asynchronous update
- Helps implement eventual consistency

This is Riak Core

- consistent hashing
- vector clocks
- sloppy quorums
- gossip protocols
- virtual nodes (vnodes)
- hinted handoff

Riak Core Implementation

- Open source
- https://github.com/basho/riak_core
- Implemented in Erlang
- Helps you build AP systems

Questions?