
Distributed Systems
Databases

WEAKER FLAVORS OF ISOLATION

It-That-Shall-Not-Be-Named

dirty writes - transaction modifies item previously modified by undecided transaction

Read-Uncommitted

dirty reads: one transaction may see uncommitted state of another transaction

Read-Committed

no dirty reads or writes, but allows for non-repeatable reads

Repeatable Reads

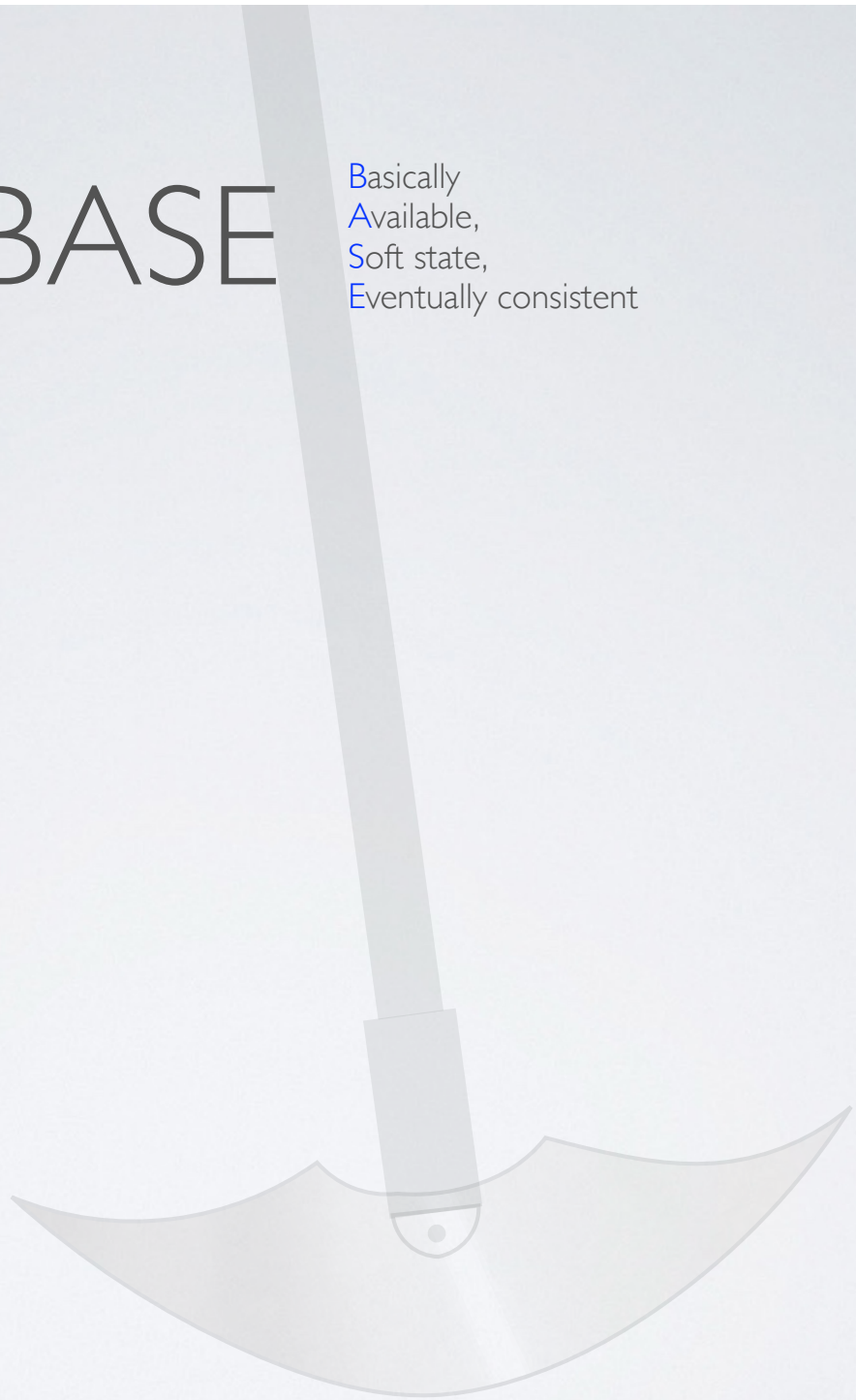
non-repeatable range reads

Snapshot Isolation

none of the above, but write skew

BASE

Basically
Available,
Soft state,
Eventually consistent

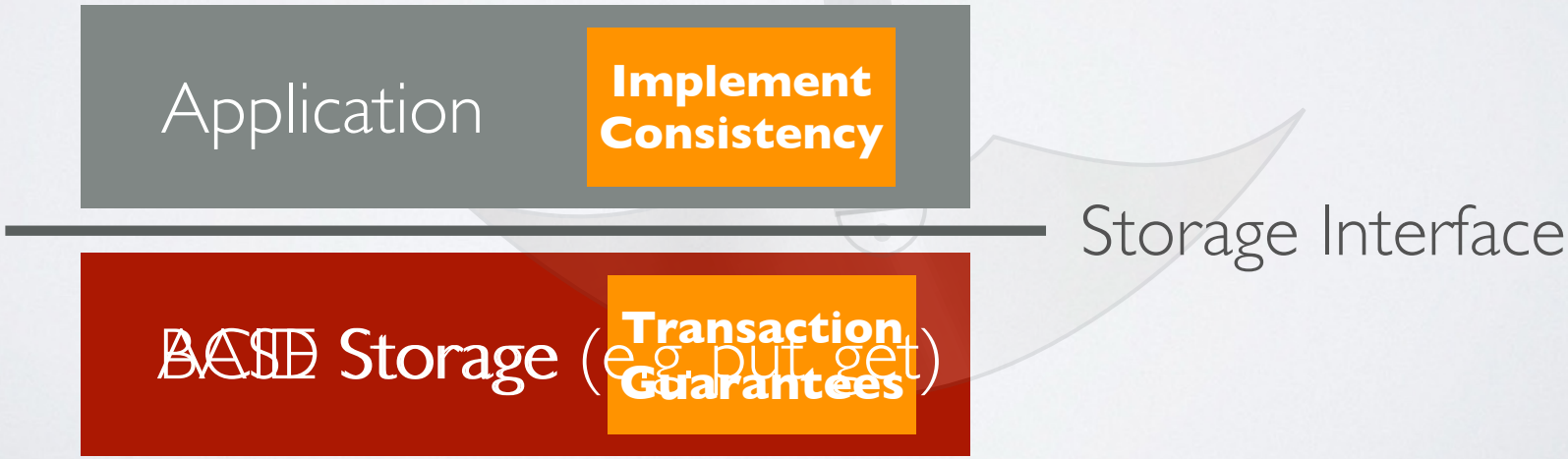


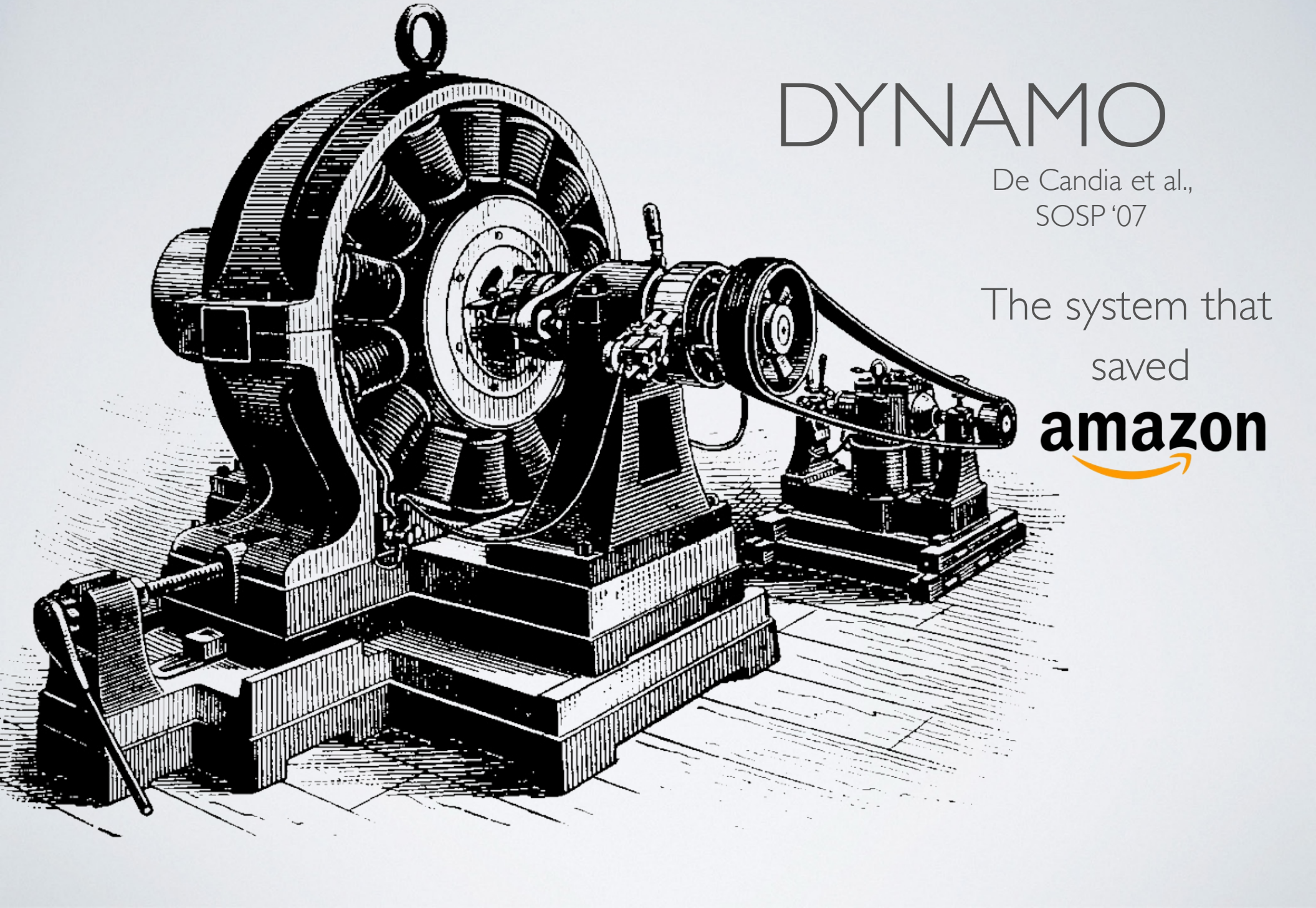
BASE

Basically
Available,
Soft state,
Eventually consistent

Custom code for better performance

Complexity gets quickly out of control





DYNAMO

De Candia et al.,
SOSP '07

The system that saved amazon

ON-PREM

Amazon unavailable for holiday shopping madness

Things are really complicated

Ashlee Vance

Published Mon 06 Dec 2004 // 23:05 UTC

Make us pref

Amazon.com went down just in time for the holidays.

The shopping site was tossing up "Http/1.1 Service Unavailable" messages to users in the US throughout Monday. The problem started in the morning and continued on throughout the afternoon. According to *The Register's* clock, Amazon was mostly functional at 4:30 p.m. CST. Several readers, however, note that while they can access the site now, they are still unable to use the "My Account" tools.

"We just had some intermittent problems this morning," said Amazon spokesman Craig Berman. Apparently "morning" defines a much broader stretch of time in Amazon country than it does elsewhere.

DYNAMO

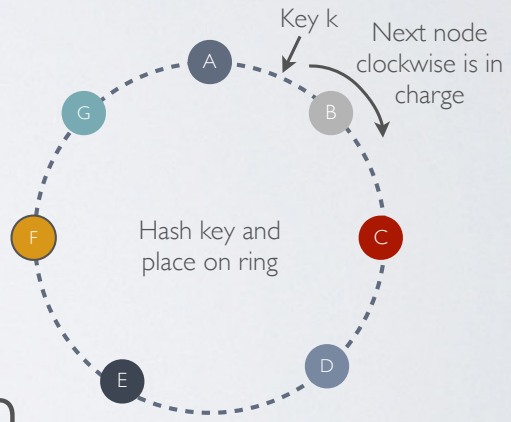
- A highly available, distributed key-value store
  - put(key, context, object); get(key)
- Sacrifices consistency for "always on" availability



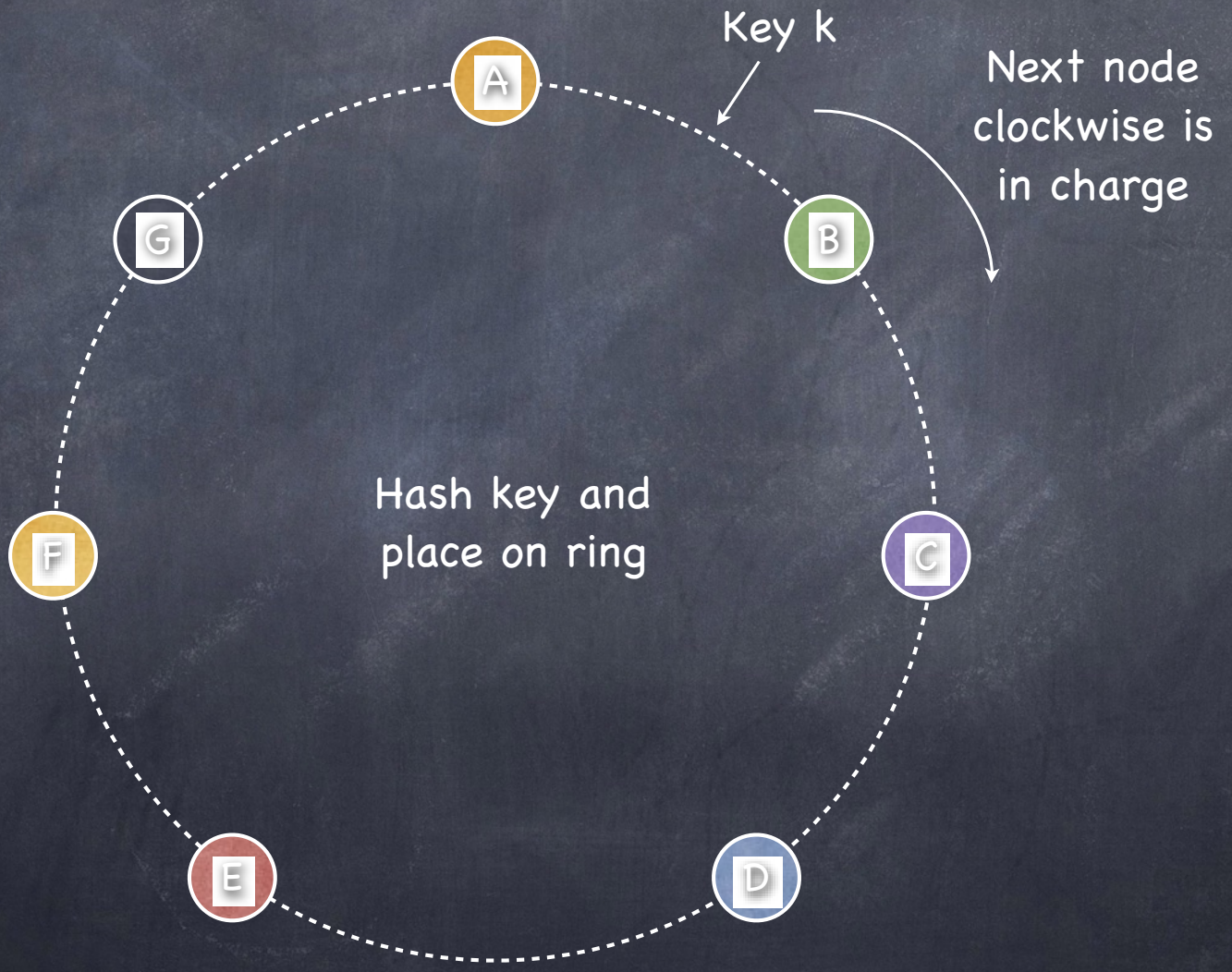
- Supports some of Amazon key features (shopping cart)
- "Synthesis of well-known techniques to achieve scalability and availability"

CONSISTENCY?

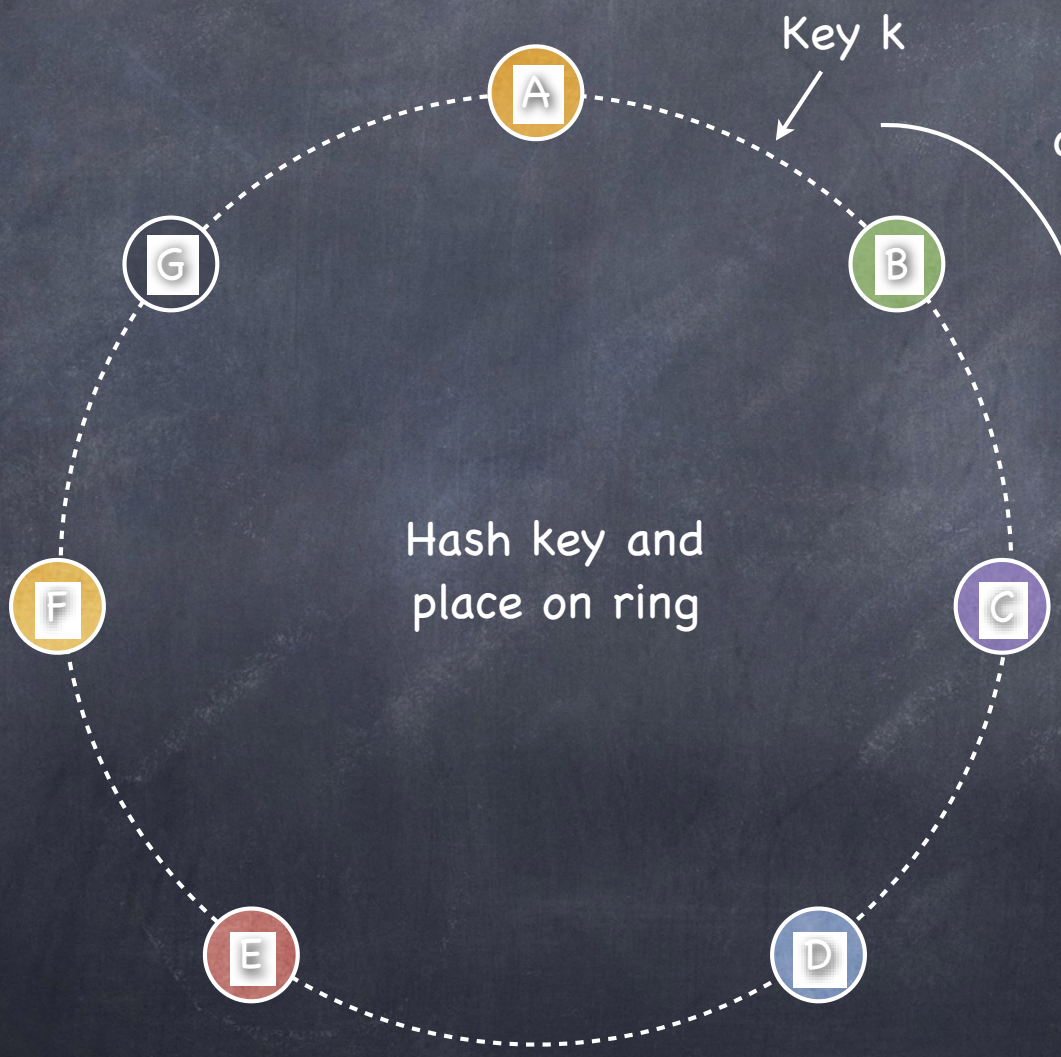
- BASE
  - Basically Available, Soft State, Eventually Consistent
- Always writable
  - Can always write to shopping cart
  - Conflict resolution on reads
- Replication via Consistent Hashing
- Application-driven conflict resolution
  - conflict resolution on reads, not writes
  - merge conflicting shopping carts to never lose "Add Cart"



Sharding via Consistent Hashing



Consistent Hashing



Issues

- Uneven load on nodes
- Uneven resources per node

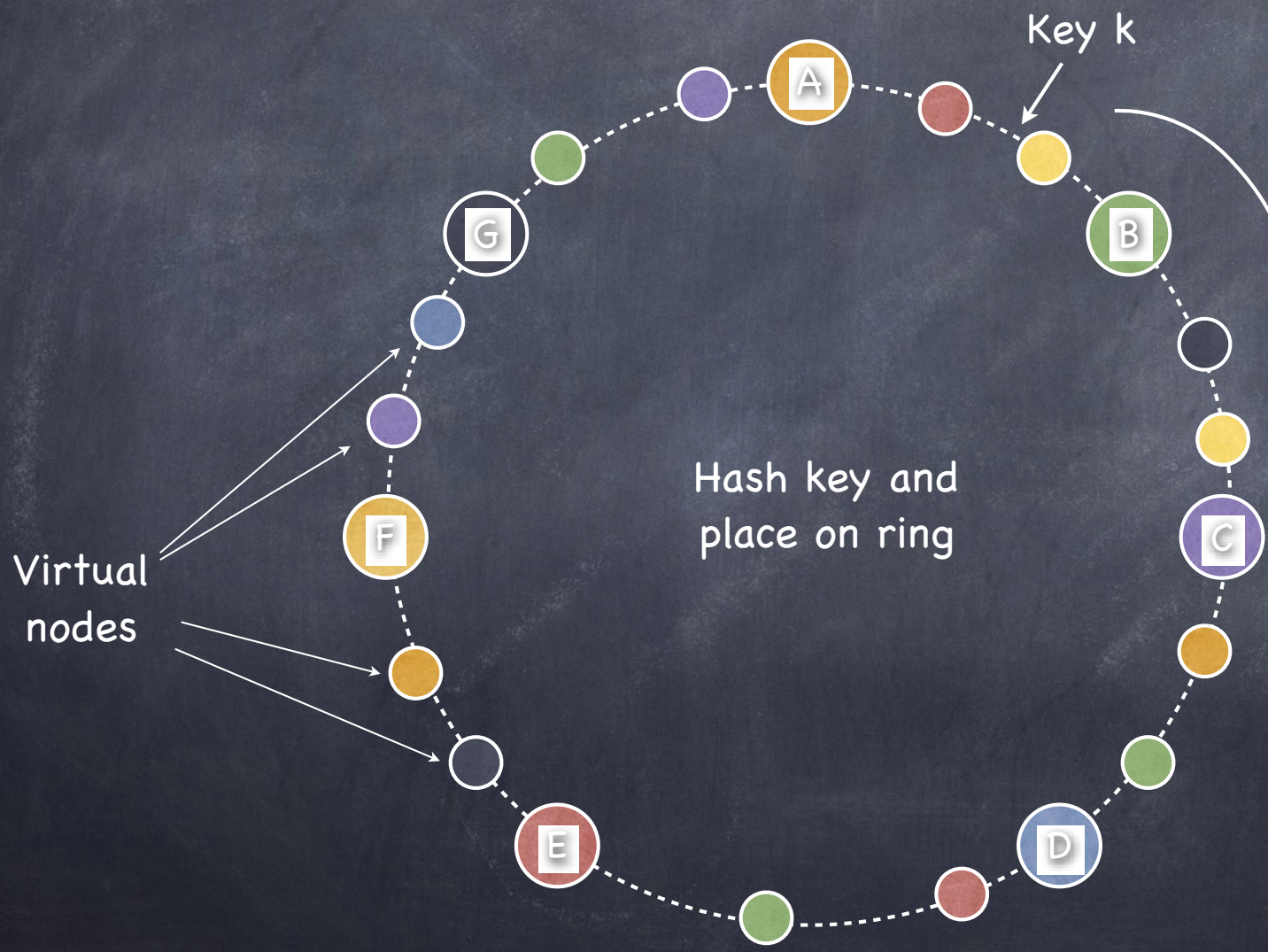
Consistent Hashing



Issues

- Uneven load on nodes
- Uneven resources per node

Consistent Hashing

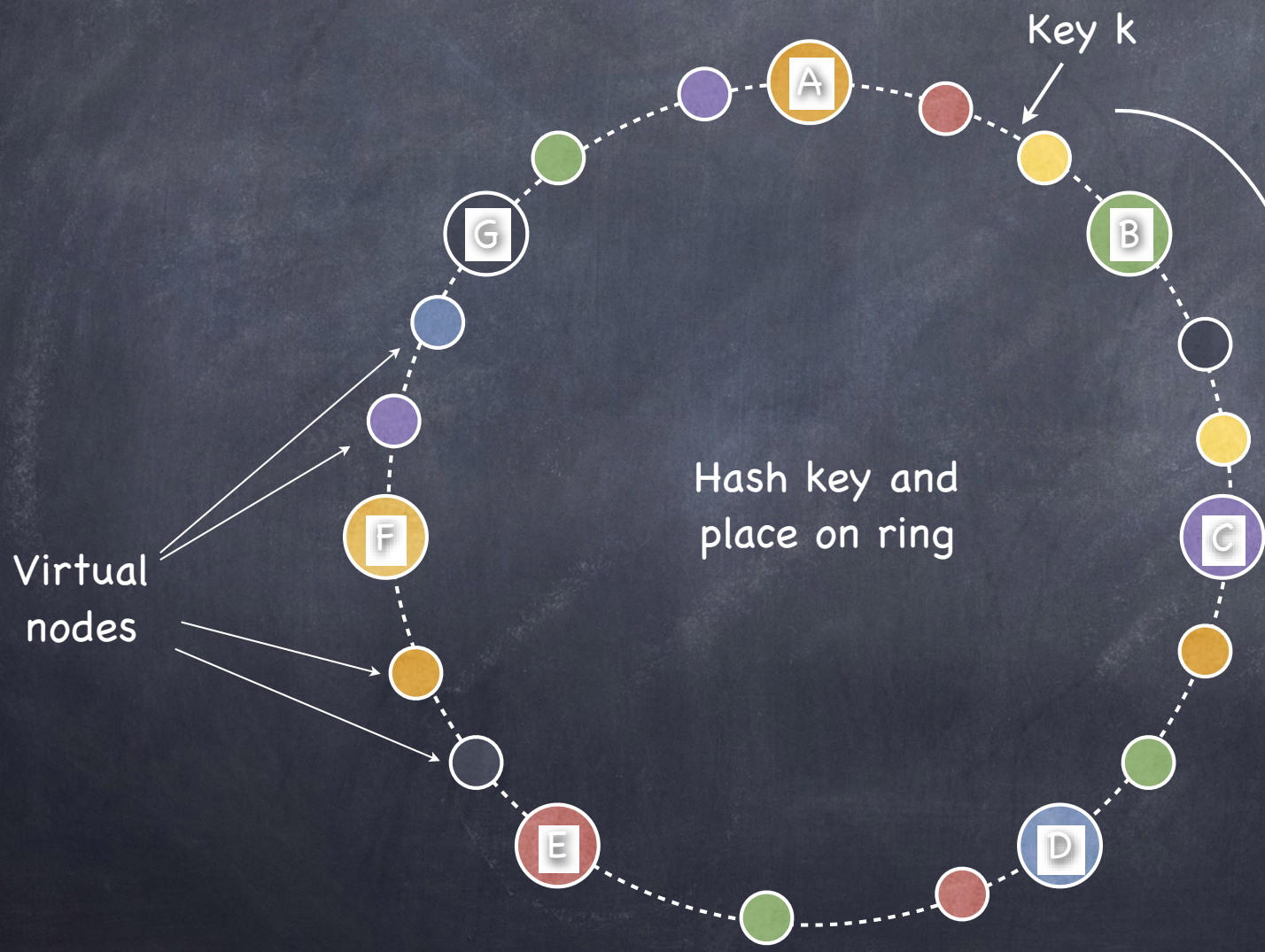


Issues

- Uneven load on nodes

Node has multiple points in ring: load more likely to even out

Consistent Hashing

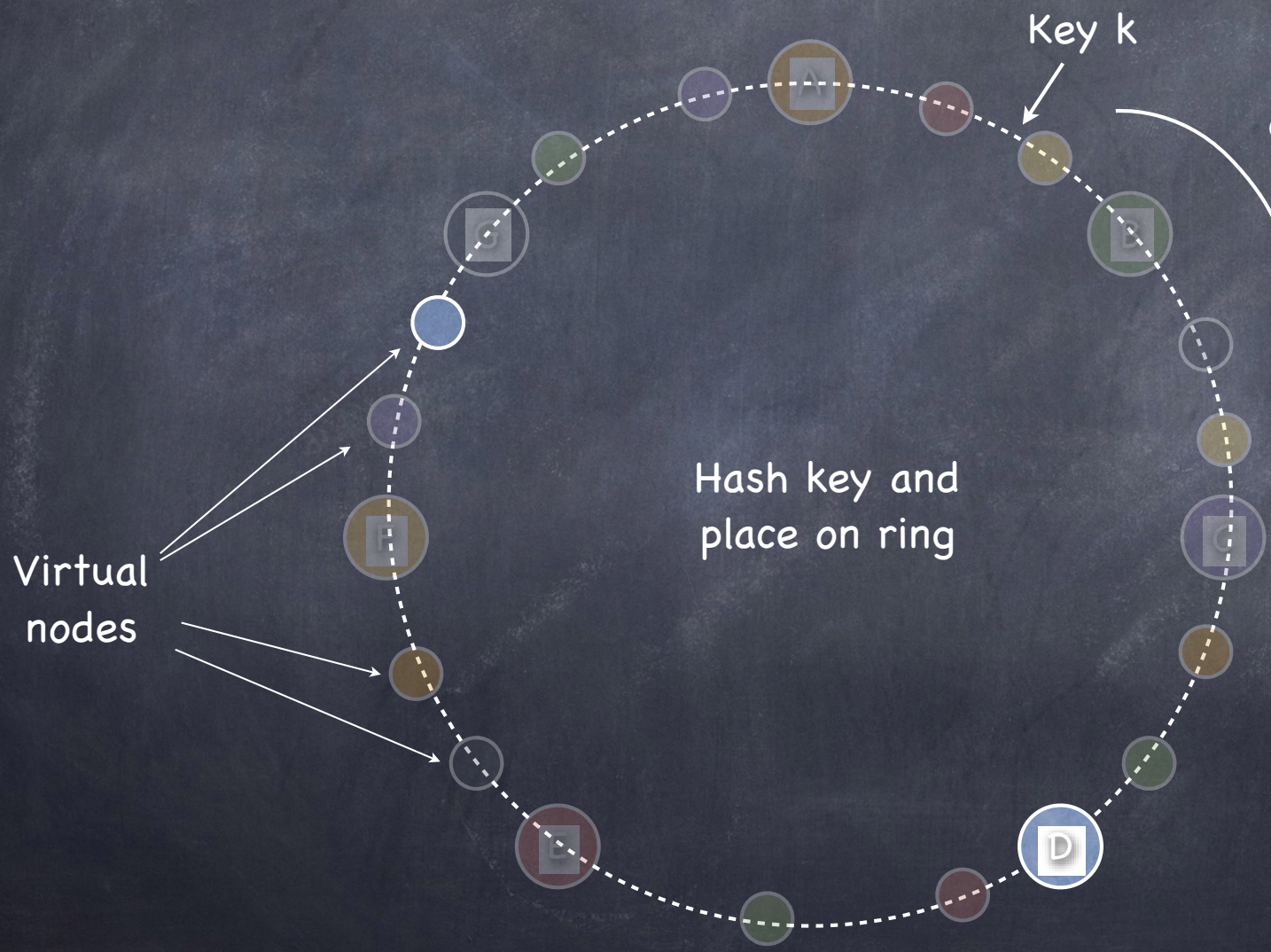


Issues

- Uneven resources per node

Different no. of virtual nodes/ physical node

Consistent Hashing

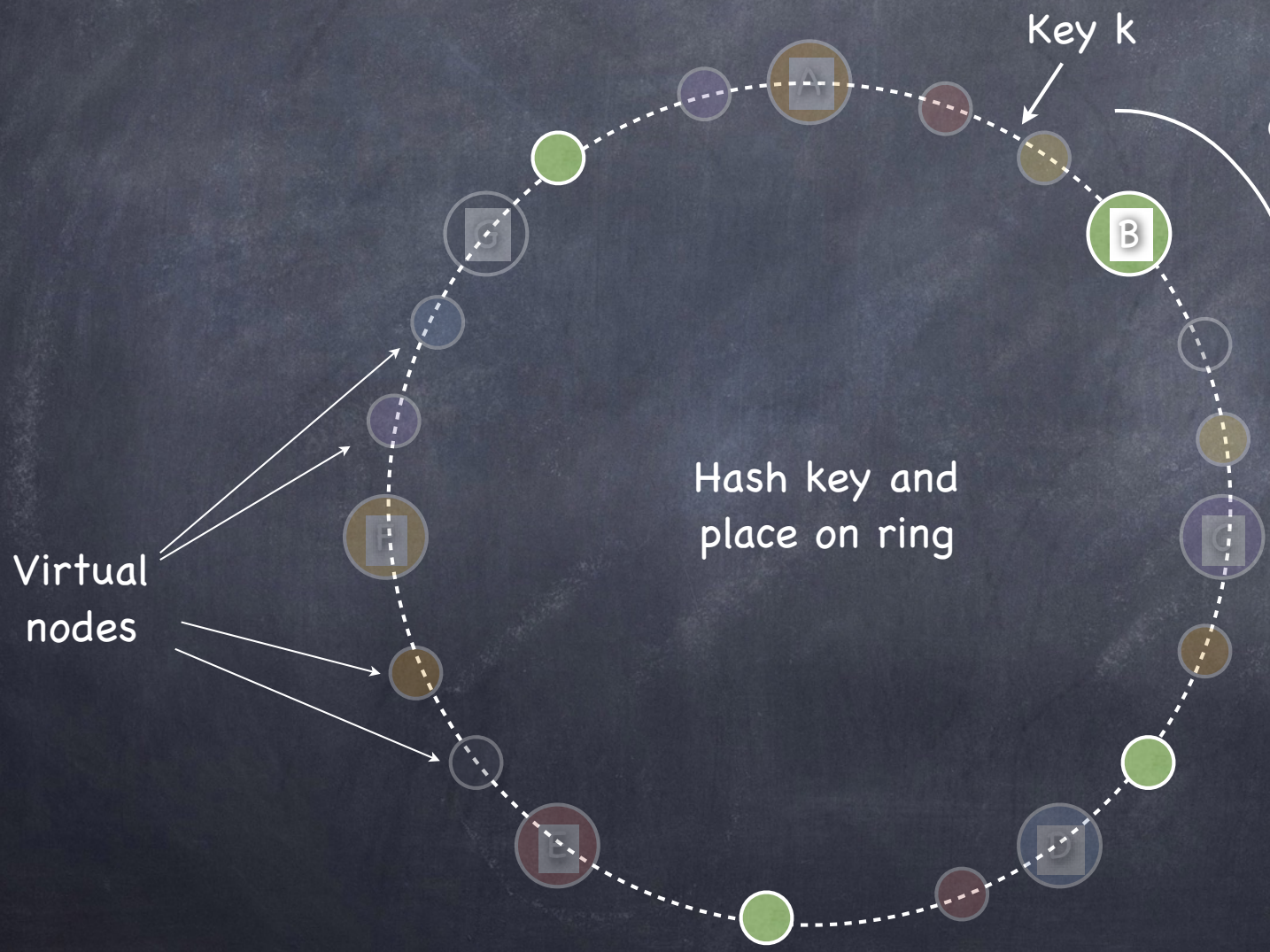


Issues

- Uneven resources per node

  Different no. of virtual nodes/ physical node

Consistent Hashing

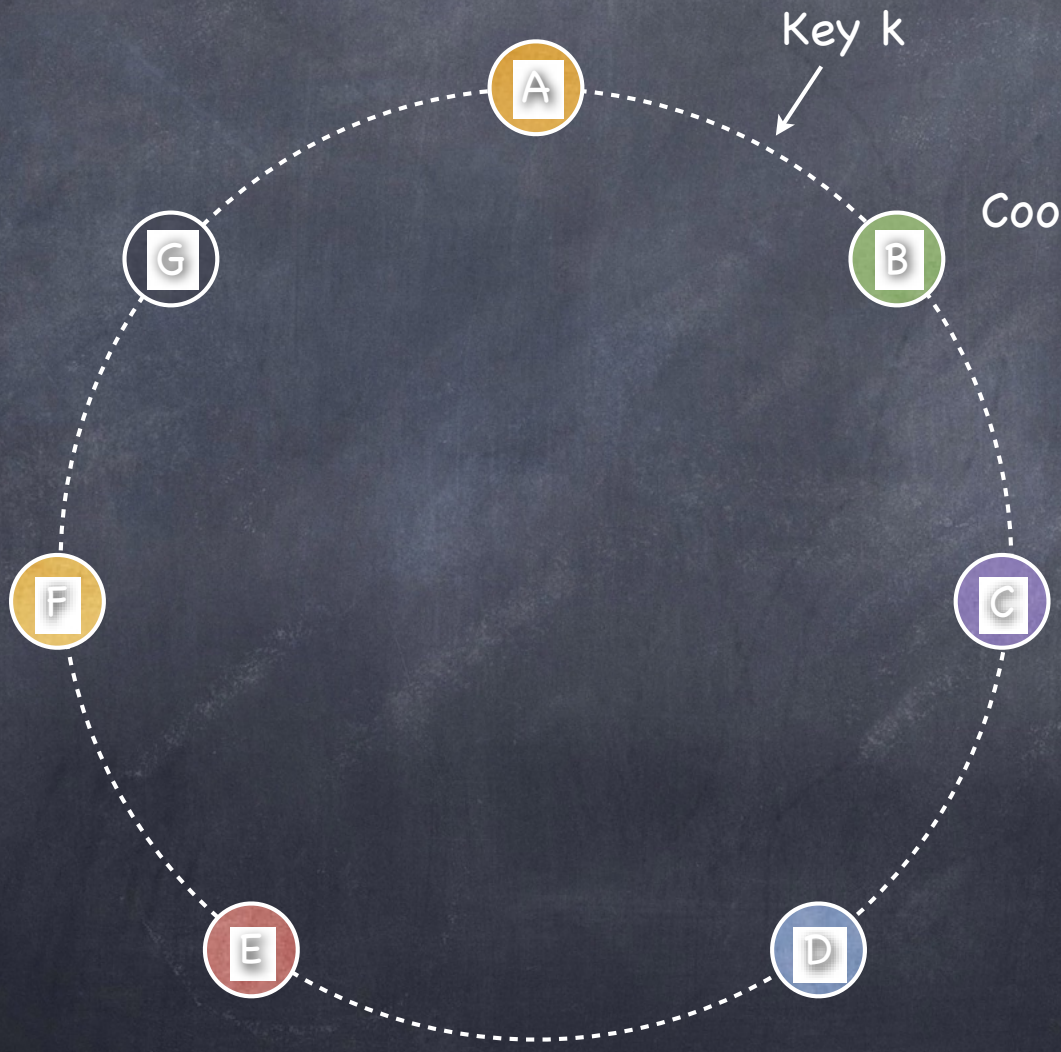


Issues

- Uneven resources per node

Different no. of virtual nodes/ physical node

Replication



Coordinator

Replicates data in preference list of more than N nodes (replication factor)

- N is per key
- N nodes in clockwise order
- mindful of virtual nodes

DATA VERSIONING

- A put() may return to its caller before the update has been applied to all replicas
  - A subsequent get() operation may miss some updates
- A get() may return many different versions of the same object
- Objects have distinct version sub-histories that need to be reconciled!

DATA VERSIONING

- A put() may return to its caller before the update has been applied to all replicas
- A get() may return many different versions of the same object
- Objects have distinct version sub-histories that need to be reconciled!

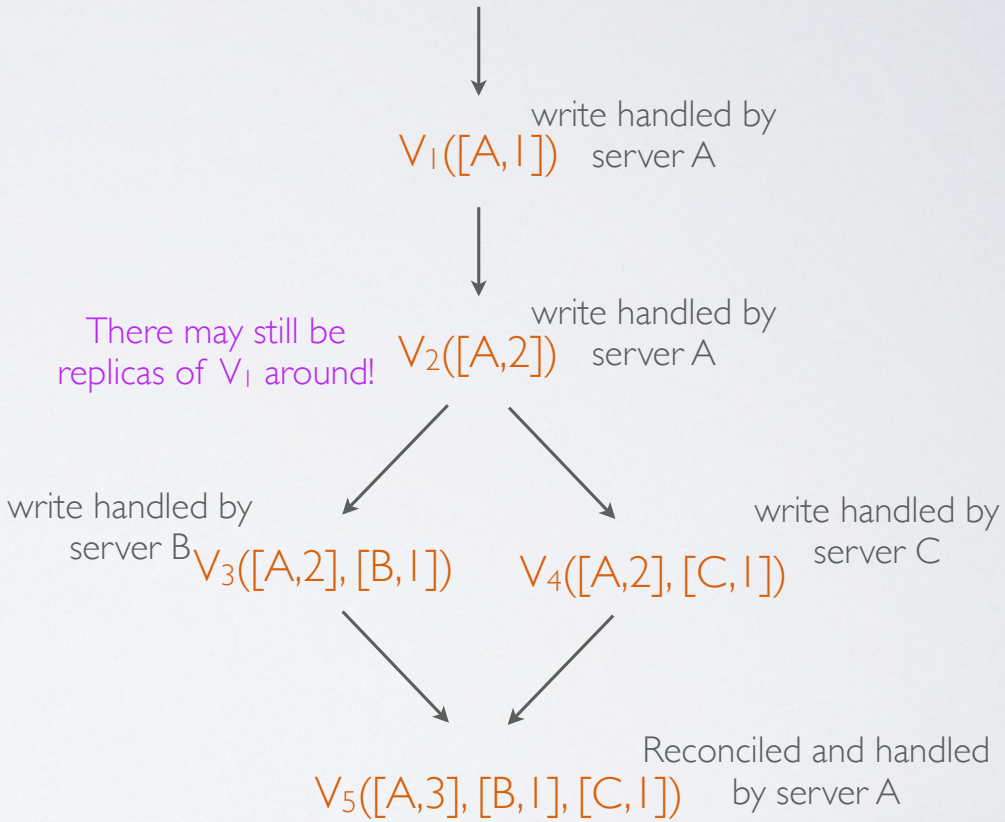
Vector clocks!

QUASI-QUORUMS

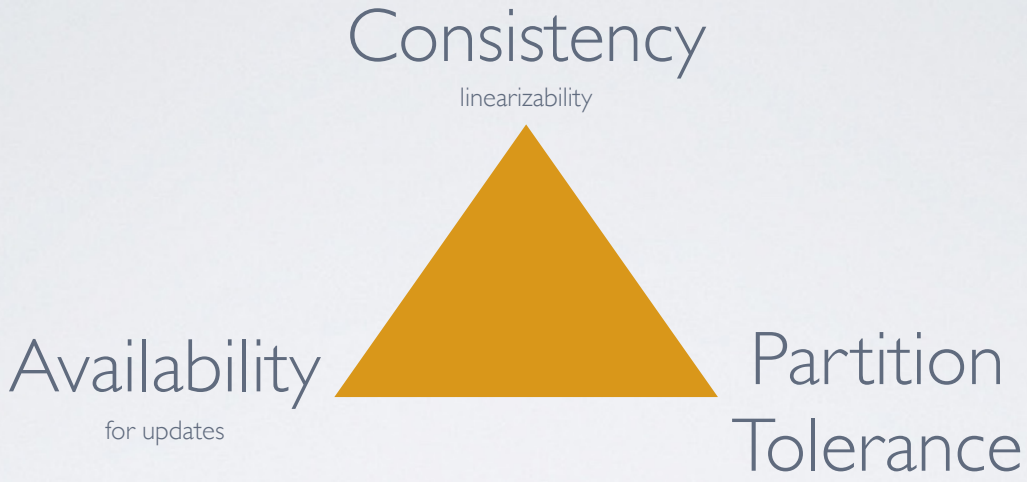
- Each key is stored in a *preference list PL* of nodes
- get() and put() driven by two parameters
  - R: minimum number of replicas to read from
  - W: minimum number of replicas to write to
  - if R+W > N, we have a quorum system!
  - ….but latency dictated by slowest replica — typically R &W chosen so together they are fewer than N
- Sloppy Quorums
  - put() sends new version to top N reachable nodes in PL; declares success as soon as received W-1 responses
  - in case of node failure, a node further in the preference list may receive the data
  - metadata specifies intended originally recipient; it will receive data asynchronously, once it is back up
  - get() retrieves from top N reachable nodes in PL; success as soon as it received R
  - if get() returns multiple versions, they are reconciled and a new corresponding version is written back

VECTOR CLOCKS À LA DYNAMO

- List of (node, counter) pairs
- Each object version has one vector clock, which captures its genesis story
- Used to determine at get time whether a version is subsumed
- If VC becomes too long, it is trimmed



THE RETURN OF THE "C"



- Last decade: a resurgence of strong consistency
  - Application code too complex and buggy without consistency support in DB
  - The choice between C & P is only necessary in the event of a partition failure — when no failure, CAP is no no excuse for weakening C!

The Triumph of Randomization

The Big Picture

- Does randomization make for more powerful algorithms?
  - Does randomization expand the class of problems solvable in polynomial time?
  - Does randomization help compute problems fast in parallel in the PRAM model?

You tell me!

The Triumph of Randomization?

Well, at least for distributed computations!

- no deterministic 1-crash-resilient solution to Consensus
- f-resilient randomized solution to consensus ($f < n/2$) for crash failures
- randomized solution for Consensus exists even for Byzantine failures!

A simple randomized algorithm

M. Ben Or. "Another advantage of free choice: completely asynchronous agreement protocols" (PODC 1983, pp. 27-30)

- exponential number of operations per process
- tolerates. $f < n/2$ crash failures
- BUT more practical protocols exist
  - down to $O(n\,log^2 n)$ expected operations/process
  - $n-1$ resilient

Weakening termination

- Consensus requires correct processes to decide always
- Ben Or's algorithm only requires termination with probability 1
- They are not the same thing!
  - e.g., when sampling a uniform distribution in [0,1], the probability of not sampling 1/2 is 1, and yet you could...

The protocol's structure

An infinite repetition of asynchronous rounds

- In round r, p only handles messages with timestamp r
- Each round has two phases:
- Phase 1: each p broadcasts an a-value
  - a function of the b-values collected in the previous round
  - the first a-value is the input bit
- Phase 2: each p broadcasts a b-value
  - a function of the collected a-values
- decide stutters

Ben Or's Algorithm

1: a_p:= input bit; r:= 1;
2: repeat forever
3: {phase 1}
4: send (a_p, r) to all
5: Let A be the multiset of the first $n-f$ a-values with timestamp r received
6: if $(\exists v \in \{0,1\} : \forall a \in A : a = v)$ then b_p:= v
7: else b_p := $\perp$
8: {phase 2}
9: send (b_p, r) to all
10: Let B be the multiset of the first $n-f$ b-values with timestamp r received
11: if $(\exists v \in \{0,1\} : \forall b \in B : b = v)$ then decide(v); a_p:= v
12: else if $(\exists b \in B : b \neq \perp)$ then a_p := b
13: else a_p:= $ { $ is chosen uniformly at random to be 0 or 1}
14: r := $r+1$

Validity

1: a_p := input bit; r:= 1;
2: repeat forever
3: {phase 1}
4: send (a_p, r) to all
5 Let A be the multiset of the first $n-f$ a-values with timestamp r received
6: if $(\exists v \in \{0,1\} : \forall a \in A : a = v)$ then b_p := v
7: else b_p := $\perp$
8: {phase 2}
9: send (b_p, r) to all
10: Let B be the multiset of the first $n-f$ b-values with timestamp r received
11: if $(\exists v \in \{0,1\} : \forall b \in B : b = v)$ then decide(v); a_p:= v
12: else if $(\exists b \in B : b \neq \perp)$ then a_p:= b
13: else a_p := $ {$ is chosen uniformly at random to be 0 or 1}
14: r := $r+1$

Validity

1: a_p := input bit; r:= 1;
2: repeat forever
3: {phase 1}
4: send (a_p, r) to all
5 Let A be the multiset of the first $n-f$ a-values with timestamp r received
6: if $(\exists v \in \{0,1\} : \forall a \in A : a = v)$ then b_p := v
7: else b_p := $\perp$
8: {phase 2}
9: send (b_p, r) to all
10: Let B be the multiset of the first $n-f$ b-values with timestamp r received
11: if $(\exists v \in \{0,1\} : \forall b \in B : b = v)$ then decide(v); a_p:= v
12: else if $(\exists b \in B : b \neq \perp)$ then a_p:= b
13: else a_p := \$ {\$ is chosen uniformly at random to be 0 or 1}
14: r := $r+1$

- All identical inputs (i)
- Each process set a-value := i and broadcasts it to all
- Since at most f faulty, every correct process receives at least $n-f$ identical a-values in round 1
- Every correct process sets b-value := i and broadcasts it to all
- Again, every correct process receives at least $n-f$ identical i b-values in round 1 and decides

A useful observation

1: a_p := input bit; r:= 1;
2: repeat forever
3: {phase 1}
4: send (a_p, r) to all
5 Let A be the multiset of the first $n-f$ a-values with timestamp r received
6: if $(\exists v \in \{0,1\} : \forall a \in A : a = v)$ then b_p := v
7: else b_p := $\perp$
8: {phase 2}
9: send (b_p, r) to all
10: Let B be the multiset of the first $n-f$ b-values with timestamp r received
11: if $(\exists v \in \{0,1\} : \forall b \in B : b = v)$ then decide(v); a_p:= v
12: else if $(\exists b \in B : b \neq \perp)$ then a_p:= b
13: else a_p := \$ {\$ is chosen uniformly at random to be 0 or 1}
14: r := $r+1$

Lemma For all r, either
$b_{p,r} \in \{1, \perp\}$ for all p or
$b_{p,r} \in \{0, \perp\}$ for all p

A useful observation

1: a_p := input bit; r:= 1;
2: repeat forever
3: {phase 1}
4: send (a_p, r) to all
5 Let A be the multiset of the first $n-f$ a-values with timestamp r received
6: if $(\exists v \in \{0,1\} : \forall a \in A : a = v)$ then b_p := v
7: else b_p := $\perp$
8: {phase 2}
9: send (b_p, r) to all
10: Let B be the multiset of the first $n-f$ b-values with timestamp r received
11: if $(\exists v \in \{0,1\} : \forall b \in B : b = v)$ then decide(v); a_p:= v
12: else if $(\exists b \in B : b \neq \perp)$ then a_p:= b
13: else a_p := \$ {\$ is chosen uniformly at random to be 0 or 1}
14: r := $r+1$

Lemma For all r, either
$b_{p,r} \in \{1, \perp\}$ for all p or
$b_{p,r} \in \{0, \perp\}$ for all p

Proof By contradiction.
Suppose p and q at round r such that
$b_{p,r}$ = 0 and $b_{q,r}$ = 1
From lines 6,7 p received $n-f$ distinct 0s, q received $n-f$ distinct 1s.
Then, $2(n-f) \leq n$, implying $n \leq 2f$
Contradiction

Corollary It is impossible that two processes p and q decide on different values at round r

Agreement

1: a_p := input bit; r:= 1;
2: repeat forever
3: {phase 1}
4: send (a_p, r) to all
5 Let A be the multiset of the first $n-f$ a-values with timestamp r received
6: if $(\exists v \in \{0,1\} : \forall a \in A : a = v)$ then b_p := v
7: else b_p := $\perp$
8: {phase 2}
9: send (b_p, r) to all
10: Let B be the multiset of the first $n-f$ b-values with timestamp r received
11: if $(\exists v \in \{0,1\} : \forall b \in B : b = v)$ then decide(v); a_p:= v
12: else if $(\exists b \in B : b \neq \perp)$ then a_p:= b
13: else a_p := \$ {\$ is chosen uniformly at random to be 0 or 1}
14: r := $r+1$

- Let r be the first round in which a decision is made
- Let p be a process that decides in r

Agreement

1: a_p := input bit; r:= 1;
2: repeat forever
3: {phase 1}
4: send (a_p, r) to all
5 Let A be the multiset of the first $n-f$ a-values with timestamp r received
6: if $(\exists v \in \{0,1\} : \forall a \in A : a = v)$ then b_p := v
7: else b_p := $\perp$
8: {phase 2}
9: send (b_p, r) to all
10: Let B be the multiset of the first $n-f$ b-values with timestamp r received
11: if $(\exists v \in \{0,1\} : \forall b \in B : b = v)$ then decide(v); a_p:= v
12: else if $(\exists b \in B : b \neq \perp)$ then a_p:= b
13: else a_p := \$ {\$ is chosen uniformly at random to be 0 or 1}
14: r := $r+1$

- Let r be the first round in which a decision is made
- Let p be a process that decides in r
- By the Corollary, no other process can decide on a different value in r
- To decide, p must have received $n-f$ "i" from distinct processes
- Every other correct process has received "i" from at least $n-2f \geq 1$
- By lines 11 and 12, every correct process sets its new a-value to for round $r+1$ to "i"
- By the same argument used to prove Validity, every correct process that has not decided "i" in round r will do so by the end of round $r+1$

Termination I

1: a_p := input bit; r:= 1;
2: repeat forever
3: {phase 1}
4: send (a_p, r) to all
5 Let A be the multiset of the first $n-f$ a-values with timestamp r received
6: if $(\exists v \in \{0,1\} : \forall a \in A : a = v)$ then b_p := v
7: else b_p := $\perp$
8: {phase 2}
9: send (b_p, r) to all
10: Let B be the multiset of the first $n-f$ b-values with timestamp r received
11: if $(\exists v \in \{0,1\} : \forall b \in B : b = v)$ then decide(v); a_p:= v
12: else if $(\exists b \in B : b \neq \perp)$ then a_p:= b
13: else a_p := \$ {\$ is chosen uniformly at random to be 0 or 1}
14: r := $r+1$

- Remember that by Validity, if all (correct) processes propose the same value "i" in phase 1 of round r, then every correct process decides "i" in round r.
- The probability of all processes proposing the same input value (a landslide) in round 1 is

  Pr[landslide in round 1] = $1/2^n$
- What can we say about the following rounds?

Termination II

1: a_p := input bit; r:= 1;
2: repeat forever
3: {phase 1}
4: send (a_p, r) to all
5 Let A be the multiset of the first $n-f$ a-values with timestamp r received
6: if $(\exists v \in \{0,1\} : \forall a \in A : a = v)$ then b_p := v
7: else b_p := $\perp$
8: {phase 2}
9: send (b_p, r) to all
10: Let B be the multiset of the first $n-f$ b-values with timestamp r received
11: if $(\exists v \in \{0,1\} : \forall b \in B : b = v)$ then decide(v); a_p := v
12: else if $(\exists b \in B : b \neq \perp)$ then a_p := b
13: else a_p := $ {$ is chosen uniformly at random to be 0 or 1}
14: r := $r+1$

- In round r > 1, the a-values are not necessarily chosen at random!
- By line 12, some process may set its a-value to a non-random value v
- By the Lemma, however, all non-random values are identical!
- Therefore, in every r there is a positive probability (at least $1/2^n$) for a landslide
- Hence, for any round r

  Pr[no lanslide at round r] $\leq 1 - 1/2^n$
- Since coin flips are independent:

  Pr[no lanslide for first k rounds] $\leq (1 - 1/2^n)^k$
- When $k = 2^n$, this value is about 1/e; then, if $k = c2^n$

  Pr[landslide within k rounds] ≥

  $1 - (1 - 1/2^n)^k \approx 1 - 1/e^c$

which converges quickly to 1 as c grows