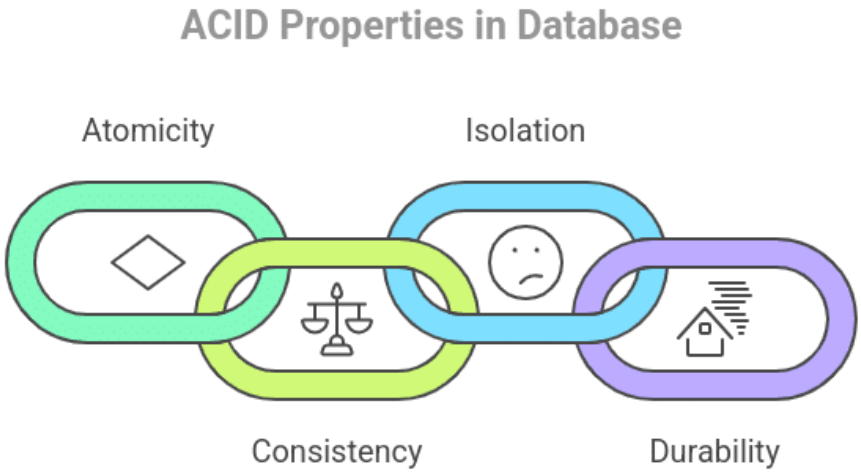
Curs 6
Tranzacții Distribuite: ACID, Commit Distribuit și Recuperare

6.1 Proprietățile ACID în sisteme distribuite

ACID este un acronim care descrie patru proprietăți esențiale ale tranzacțiilor de bază de date: *Atomicitate*, *Consistență*, *Izolare* și *Durabilitate*.



https://www.mygreatlearning.com/blog/wp-content/uploads/2025/01/acid-properties-in-databse.png

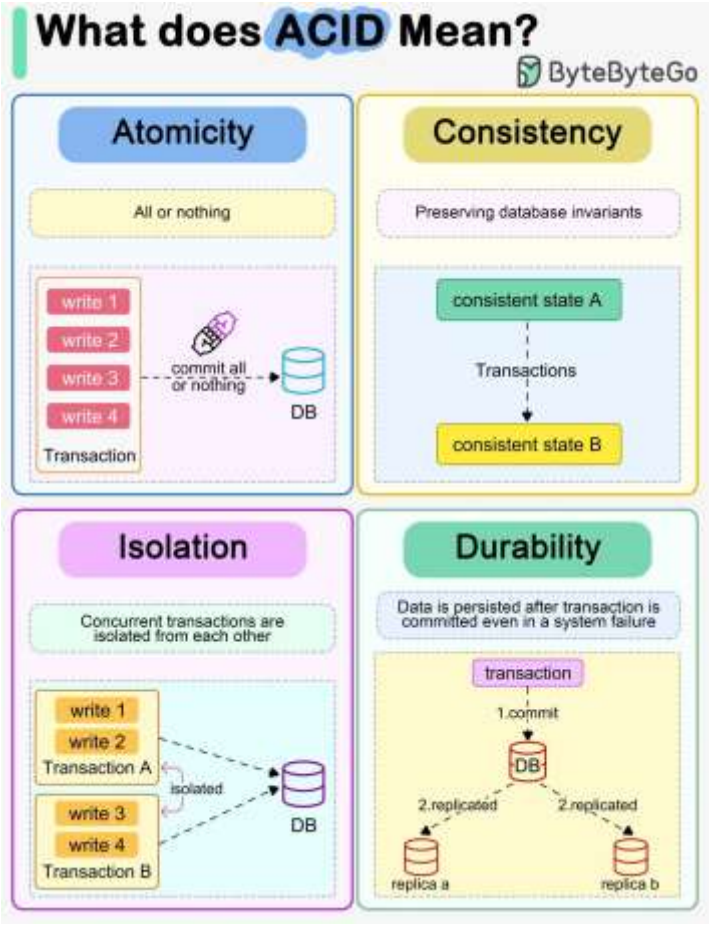
Împreună, aceste proprietăți asigură că o serie de operații executate ca o tranzacție lasă baza de date într-o stare validă chiar și în cazul apariției unor erori[1].

În contextul bazelor de date distribuite, menținerea proprietăților ACID necesită mecanisme suplimentare (precum protocoale de commit distribuit) deoarece tranzacțiile pot implica mai multe noduri.

1) **Atomicitate** – *Totul sau nimic*: toate operațiile unei tranzacții se execută cu succes în întregime sau se anulează complet. În caz de eșec, tranzacția este rollback (anulată) integral, astfel încât niciun site nu rămâne cu modificări parțiale. Atomicitatea în medii distribuite este asigurată de protocoale de tip commit atomic, care garantează un rezultat unitar pe toate nodurile (detaliat la secțiunea 5.2)[2].
2) **Consistență** – *Stare validă a datelor*: tranzacția trebuie să ducă baza de date dintr-o stare consistentă în altă stare consistentă, respectând toate constrângerile definite (integritate, reguli de validare etc.). În sisteme distribuite, consistența implică propagarea corectă a modificărilor pe toate nodurile și respectarea constrângerilor globale. Dacă o tranzacție ar viola consistența (de exemplu prin încălcarea unei constrângeri), atunci modificările sunt anulate (tranzacția se abortă). Consistența este strâns legată de atomicitate și izolare – dacă tranzacția este atomică și izolată, ea nu va vedea sau produce stări intermediare inconsistente.
3) **Izolare** – *Concurență fără interferență*: tranzacțiile par să ruleze secvențial (una câte una), chiar dacă în realitate pot rula concurent. Modificările făcute de o tranzacție nu devin vizibile altor tranzacții înainte de commit-ul final, iar execuțiile intercalate ale tranzacțiilor

concurente nu trebuie să ducă la stări invalide. Sisteme de baze de date folosesc de obicei mecanisme de **control al concurenței** (blocări – *locking*, marca timp – *timestamp ordering* etc.) pentru a asigura izolare. În contexte distribuite, izolarea trebuie asigurată atât local (la nivelul fiecărui nod, ex. prin *snapshot isolation* sau *two-phase locking* local) cât și global, uneori implicând coordonare între noduri (ex: un manager de blocări distribuit sau protocoale de sincronizare a snapshot-urilor)[3].

4) **Durabilitate** – *Persistența datelor*: odată ce o tranzacție a fost confirmată (commit), rezultatele sale persistă permanent în baza de date, chiar dacă sistemul se oprește brusc sau apar defecțiuni hardware. Fiecare nod participant la tranzacție își salvează modificările pe stocare non-volatilă (de exemplu în jurnalul de tranzacții pe disc) înainte de a confirma commit-ul, astfel încât după o cădere, la repornire, sistemul poate recupera starea consistentă. În sisteme distribuite, durabilitatea este asigurată de obicei prin jurnalizare pe fiecare nod (fiecare participant face *write-ahead logging* local) și adesea prin replicarea datelor – dacă un nod se defectează, copiile sale replică mențin datele commit-ului. Un exemplu: MongoDB utilizează un jurnal de operațiuni (OpLog) pentru a relua scrierile nefinalizate în caz de întrerupere de curent[1].



https://i.pinimg.com/originals/ec/12/9e/ec129ecf1ae366016a2b4d567bfa53d1.jpg

ACID vs. BASE în lumea NoSQL: Tranzacțiile ACID oferă consistență puternică, însă în sisteme distribuite la scară foarte mare, menținerea ACID poate afecta disponibilitatea și performanța.

Multe baze de date NoSQL tradiționale au urmat principiul *BASE* (*Basically Available, Soft state, Eventually consistent*) sacrificând consistența strictă pentru a obține disponibilitate și scalabilitate sporită.

Inițial, multe sisteme NoSQL nu au oferit tranzacții multi-obiect, limitându-se la consistență eventuală sau la tranzacții restrânse pe un singur element (de ex. single-document)[4].

Totuși, pe măsură ce cerințele aplicațiilor au evoluat, unele SGBD NoSQL moderne au introdus suport pentru tranzacții ACID la nivel distribuit:

- **Tranzacții NoSQL native:** MongoDB, de exemplu, suportă tranzacții ACID multi-document (în replica set de la v4.0, extins la clustere shard-uite în v4.2) – permițând operații atomice care afectează mai multe documente/colecții[5]. DynamoDB oferă operații tranzacționale (TransactWriteItems, TransactGetItems) ce garantează ACID la nivel de mai multe itemi și tabele distribuite[5]. Couchbase și RavenDB sunt alte exemple de baze NoSQL care au implementat tranzacții distribuite ACID, extinzând modelul inițial BASE.
- **Tranzacții restrânse (single item/shard):** Multe baze NoSQL asigură atomicitate doar la nivel de un singur document sau cheie (ex: operațiile asupra unui singur document JSON în Couchbase sunt atomice). Aceasta simplifică menținerea consistenței (nu există tranzacții multi-document de coordonat), dar nu acoperă cazurile de utilizare ce cer actualizări atomice în mai multe entități.
- **Soluții la nivel de aplicație:** În absența suportului tranzacțional nativ, dezvoltatorii au recurs la implementarea logicii de tranzacție în aplicație, fie folosind protocoale de commit în două faze la nivel de aplicație (coordonând mai multe operații pe diferite noduri), fie modele de compensare (*Saga pattern* pentru consistență finală). Aceste abordări însă adaugă complexitate și nu oferă garanțe la fel de solide ca tranzacțiile ACID native[5].
- **NewSQL – ACID la scară:** O clasă nouă de sisteme numite *distributed SQL/NewSQL* (ex: Google Spanner, CockroachDB, YugabyteDB) încearcă să ofere atât scalabilitatea NoSQL, cât și tranzacții ACID distribuite complet. Aceste sisteme implementează mecanisme de coordonare globală a tranzacțiilor – de exemplu, Spanner folosește un protocol de consens (Paxos) și ceasuri sincronizate (TrueTime) pentru a asigura tranzacții distribuite cu consistență puternică la nivel global[3]. Astfel de sisteme demonstrează că este posibilă menținerea proprietăților ACID și a performanței prin algoritmi avansați: Spanner efectuează practic un commit în două faze peste Paxos (replicând decizia de commit în mod distribuit), evitând blocarea sistemului în caz de cădere a unui nod coordonator[6]. Evident, aceste soluții vin cu costuri de implementare și infrastructură (ex: ceasuri atomice, rețele fiabile), dar sunt folosite în aplicații critice ce necesită **consistență puternică și disponibilitate ridicată** (bănci, sisteme financiare globale, etc.).

Proprietățile ACID rămân fundamentale pentru integritatea datelor într-un sistem de baze de date. În medii distribuite, asigurarea ACID necesită mecanisme dedicate (protocol de commit atomic, sincronizare între noduri, replicare și jurnalizare distribuited) pentru a garanta că tranzacțiile care implică multiple noduri sunt **atomice** (toate nodurile reflectă aceeași decizie), mențin **consistența** (regulile și constrângerile nu sunt încălcate global), oferă **izolare** (concurența nu duce la anomalii), și sunt **durabile** (niciun eșec nu va anula efectele unui commit confirmat).

6.2 Protocoale de commit distribuite (2PC și 3PC)

Pentru a realiza *atomicitatea* tranzacțiilor într-o bază de date distribuită, unde o singură tranzacție poate afecta date pe mai multe noduri, este necesar un **protocol de commit distribuit**. Acesta coordonează nodurile participante astfel încât toate să ia aceeași decizie: **toate fac commit** sau **toate fac rollback (abort)**, garantând un rezultat atomic global.

Cel mai utilizat protocol de acest tip este **Two-Phase Commit (2PC)**, iar o variantă avansată este **Three-Phase Commit (3PC)**. Vom descrie pe rând aceste protocoale, modul lor de funcționare și implicațiile în practică.

6.2.1. Protocolul în două faze – Two-Phase Commit (2PC)

Two-Phase Commit (2PC) este un protocol atomic de commit utilizat pe scară largă în sisteme de baze de date distribuite pentru a asigura că o tranzacție distribuită este fie comisă pe toate nodurile, fie anulată pe toate, evitând stările parțiale inconsistene[2].

În 2PC există două roluri principale: **coordonatorul** (unul dintre noduri sau un manager dedicat care coordonează tranzacția globală) și **participanții** (toate nodurile care dețin o parte din datele tranzacției).

Protocolul se desfășoară în **două faze** succesive de comunicare între coordonator și participanți:

Faza 1 – Pregătire (Colectare a voturilor): coordonatorul inițiază commit-ul global trimițând un mesaj de **pregătire** (*prepare* sau *vote request*) către toți participanții implicați. În această etapă, fiecare participant își analizează propria execuție locală a tranzacției:

a) Dacă participantul **poate confirma** tranzacția (toate operațiile locale au reușit, nu există motive de abort local), atunci *votează „Yes”* (gata de commit). În practică, participantul va **scrie în jurnal un record <ready T>** (ce indică faptul că este pregătit să comită T) și trimite coordonatorului mesajul de confirmare (eg. **“Ready T”**)[7].

b) Dacă participantul întâmpină o problemă locală (eroare, constrângere încălcată) sau nu poate garanta succesul, *votează „No”* (abort). În acel caz, participantul **notează în jurnal <abort T>** și trimite coordonatorului un mesaj de **Abort (refuz)**[7]. Orice participant care votează *No* determină tranzacția globală să fie abandonată.

c) Participantul poate întârzia răspunsul dacă încă procesează tranzacția, dar **trebuie să răspundă într-un interval de timp rezonabil**. Dacă coordonatorul nu primește răspuns de la un participant într-un timp-limită, va **presupune un vot negativ** (abort) din partea acelui nod[7].

d) În această fază, tranzacția rămâne **nefinalizată**: participanții care au votat "Yes" se consideră în stare de *pregătit* și **țin blocate resursele modificate** (ex. păstrează lacătele pe date) până la decizia finală. Ei nu fac commit efectiv încă, așteptând indicația coordonatorului.

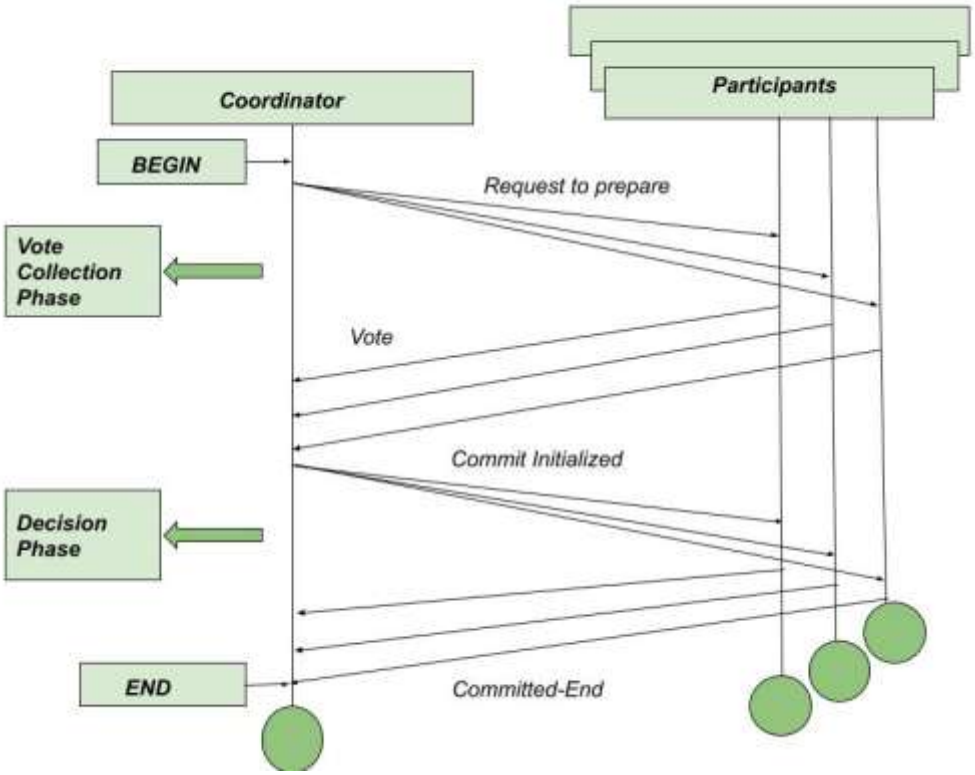
Faza 2 – Decizie (Commit/Abort global): coordonatorul centralizează toate voturile participanților și decide acțiunea finală:

a) Dacă **toți participanții au răspuns “Yes/Ready”**, coordonatorul decide **commit**. El va **scrie în jurnalul său <commit T>** (marcând decizia de a comite tranzacția T) și trimite un

mesaj de **Commit** către toate nodurile participante[7]. Fiecare participant, la primirea mesajului de commit, execută commit local (persistă schimbările) și scrie la rândul său <commit T> în propriul jurnal, apoi eliberează resursele (ex. deblochează datele)[7].

b) Dacă **unul sau mai mulți participanți au răspuns "No/Abort"** (sau au fost presupuși abort din timeout), coordonatorul decide **abort** pentru întreaga tranzacție. El va nota <abort T> în jurnalul propriu și va trimite mesaje de **Abort** către toate nodurile[7]. Toți participanții, la primirea mesajului de abort, vor anula tranzacția locală (rollback, undo modificările locale neconfirmate) și vor scrie <abort T> în jurnalul lor[7].

c) Astfel, tranzacția este fie comisă la *toate* site-urile, fie anulată la *toate*, menținând atomicitatea globală. Indiferent de decizie, coordonatorul poate trimite și un mesaj final de **confirmare a încheierii** (echivalent "END") dacă protocolul o cere, însă de obicei simpla primire a commit/abort de către participanți finalizează protocolul.



Schema comunicării în protocolul 2PC.

Coordonatorul inițiază faza de vot (*prepare request*), participanții trimit voturile ("Yes"/"No"), apoi coordonatorul transmite decizia finală de commit sau abort (faza de decizie). Toate nodurile se sincronizează astfel la aceeași hotărâre.

2PC garantează *atomicitatea distribuției* – dacă oricare participant votează abort, întreaga tranzacție se va anula, iar dacă toți votează commit, tranzacția se confirmă peste tot. **Avantajul** major este simplitatea conceptului și faptul că asigură consistență globală în mod determinist.

Acest protocol stă la baza managerilor de tranzacții distribuite din numeroase SGBD-uri și medii enterprise (ex: standardul X/Open XA pentru tranzacții distribuite între baze de date folosește 2PC ca mecanism de commit).

Totuși, 2PC prezintă și **dezavantaje importante**, în special în scenarii de defecțiuni:

i. **Blocarea în caz de eșec al coordonatorului:** Dacă coordonatorul cade după ce participanții au votat *"Ready"* (deci tranzacția este în faza de incertitudine), participanții rămân blocați așteptând decizia finală. Ei au resursele blocate (de ex. lacăte pe rânduri) și nu știu dacă să facă commit sau rollback. Această situație se numește **blocking problem** – tranzacția rămâne *în-doubt* (incertă) până revine coordonatorul sau se intervine manual[7]. În tot acest timp, datele implicate nu pot fi modificate de alte tranzacții, afectând concurența. Problema apare deoarece 2PC **nu oferă toleranță la eșec pentru coordonator** – coordonatorul este un punct unic de decizie; dacă el nu comunică decizia, nimeni altcineva nu o poate deduce cu certitudine (vom vedea la secțiunea 5.3 cum se tratează această situație și cum 3PC încearcă să o evite).

ii. **Timp de commit mai mare și impact pe performanță:** 2PC necesită două runde de mesaje (două *round-trip*-uri) către fiecare participant, ceea ce adaugă latență. De asemenea, participanții trebuie să **aștepte** decizia coordonatorului, menținând eventual blocări de resurse. Acest lucru poate reduce performanța în medii cu tranzacții intense. În mod special, dacă un participant este mai lent sau rețeaua are latență, toată tranzacția este ținută în wait. În practică, problema mai severă nu este neapărat cele două faze, ci **retenția blocărilor**: tranzacția ține lacăte la participanți pe toată durata protocolului, ceea ce poate crește șansele de blocaje și timeout-uri atunci când multe tranzacții concurente apar pe aceleași date[16].

iii. **Lipsa de suport pentru partitionări de rețea/extensii la medii nefiabile:** Protocolul 2PC, în forma de bază, presupune că mesajele coordonator-participanți vor ajunge (eventual, cu retry) și că eșecurile se manifestă ca căderi de nod detectabile. În caz de **partiționare de rețea** (clusterul se împarte în două sub-rețele izolate), 2PC nu poate preveni complet incoerențele sau blocajele. De exemplu, dacă coordonatorul se află într-o partiție cu un subset de participanți, celelalte noduri îl vor percepe ca *dispărut* și vor rămâne blocate (simulând cazul de coordonator căzut)[8]. Vom discuta la 3PC cum se încearcă ameliorarea acestei situații, deși problemele cauzate de partitionare nu pot fi eliminate 100% fără un mecanism de consens.

Chiar dacă 2PC suferă de potențial de blocare, multe sisteme practice implementează mecanisme de atenuare.

De exemplu, pot exista **coordonatori fail-over** (replicați) sau proceduri de **time-out și abort unilateral** dacă un coordonator nu revine într-un timp foarte îndelungat (deși aceasta riscă inconsistențe și se folosește doar în ultimă instanță). Unele SGBD (ex: Oracle) desemnează un *commit point site* mai robust, care să acționeze preferențial ca coordonator pentru a minimiza șansa de blocaj[9].

Soluția robustă în industrie pentru a elimina blocajele implică utilizarea unui **algoritm de consens** distribuit (ex: Paxos, Raft) care să decidă atomic commit-ul – practic, *2PC combinat cu replicare prin consens*.

Spre exemplu, Google Spanner execută 2PC peste Paxos, astfel încât dacă un coordonator cade, alt nod poate prelua decizia pe baza jurnalelor replicate; acest design "mitighează problemele de disponibilitate" ale 2PC[6]. Totuși, astfel de implementări sunt complexe și depășesc sfera 2PC standard.

6.2.2. Protocolul în trei faze – Three-Phase Commit (3PC)

Three-Phase Commit (3PC) este o extensie a protocolului 2PC, proiectată pentru a elimina blocarea sistemului în anumite scenarii de eșec. Ideea principală în 3PC este introducerea unei a treia faze intermediare (fază de **diseminare/pre-commit**) înainte de decizia finală, astfel încât participanții să cunoască intenția de commit a coordonatorului și să poată acționa în absența acestuia. 3PC încearcă să asigure că **niciun nod nu rămâne blocat** permanent dacă coordonatorul cade, *presupunând anumite condiții* (de ex., că nu are loc o partitionare majoră de rețea și că eșecurile de nod sunt limitate)[10].

Protocolul 3PC se desfășoară astfel, în termeni generali:

Faza 1 – Votare (identică cu 2PC): Coordonatorul trimite mesajele de **Pregătire** către participanți, iar aceștia răspund cu "Yes" sau "No" similar ca în 2PC (logând <ready T> la cei care votează *Yes*). Dacă oricare votează *No*, coordonatorul poate aborta tranzacția imediat (protocolul se termină similar 2PC cu abort).

Faza 2 – Pre-Commit (Diseminare intenție): Dacă toți participanții au răspuns **gata (Yes)**, spre deosebire de 2PC unde s-ar trimite direct commit, în 3PC coordonatorul trece printr-o fază intermediară. El **trimite un mesaj de Pre-Commit (sau "prepare-to-commit")** către participanți, informându-i că intenționează să comită tranzacția. Coordonatorul **nu scrie încă <commit> în jurnal**, ci eventual un <pre-commit T> (sau notează intenția). Când participanții primesc Pre-Commit, **își înregistrează starea de pre-commit (ex. log <prepared T>) și răspund cu un ACK** coordonatorului[10]. În acest moment, toți participanții știu că *toți ceilalți au votat da* și că urmează commit, dar **încă nu au comis efectiv**.

Faza 3 – Commit final: După ce coordonatorul primește **confirmările (ACK) de la participanți** pentru faza de pre-commit, poate trimite mesajul final de **Commit** (similar fazei 2 din 2PC). În acel moment, participanții fac commit local și tranzacția se încheie. Dacă însă coordonatorul se prăbușește înainte de a ajunge la decizia finală, *situația poate fi rezolvată fără blocare*: deoarece participanții au primit *intenția de commit (Pre-Commit)*, ei pot conveni între ei asupra deciziei: - Se alege un **nou coordonator** (din participanții care știu de tranzacție).

Acesta întreabă pe ceilalți starea tranzacției.

- Dacă cel puțin un participant indică faptul că a primit mesajul de Pre-Commit (adică are în jurnal <prepared T>), înseamnă că vechiul coordonator **decisese commit-ul** înainte de a cădea. Prin urmare, noul coordonator **va continua cu commit** (toți participanții vor putea face commit pe baza informațiilor de la ceilalți)[10].

- Dacă niciun participant nu a ajuns în starea de pre-commit (de exemplu coordonatorul a căzut foarte devreme, înainte de faza 2), atunci noul coordonator **poate safly să dea Abort**, știind că nimeni nu a considerat tranzacția ca fiind aproape de commit.

- Pe baza acestor reguli, 3PC **evită blocarea**: un participant nu rămâne niciodată într-o situație de *incertitudine totală* – ori știe că tranzacția a fost pre-comisă (și deci alt nod poate finaliza commit-ul), ori știe că nu s-a ajuns la decizie și poate aborta. Condiția este ca nu mai mult de un anumit număr k de noduri să cadă simultan și rețeaua să nu se împartă (fără partitionare care să segmenteze participanții)[10].

Diagrama 3PC : Protocolul 3PC are tot o fază de vot inițială, dar introduce o **fază de diseminare** a deciziei de commit intenționate, înainte de commit-ul efectiv. Participanții primesc un mesaj Prepare to commit și confirmă că au *pregătit commit-ul* (au totul gata de finalizare). Abia apoi coordonatorul (sau un înlocuitor) comandă commit-ul final.

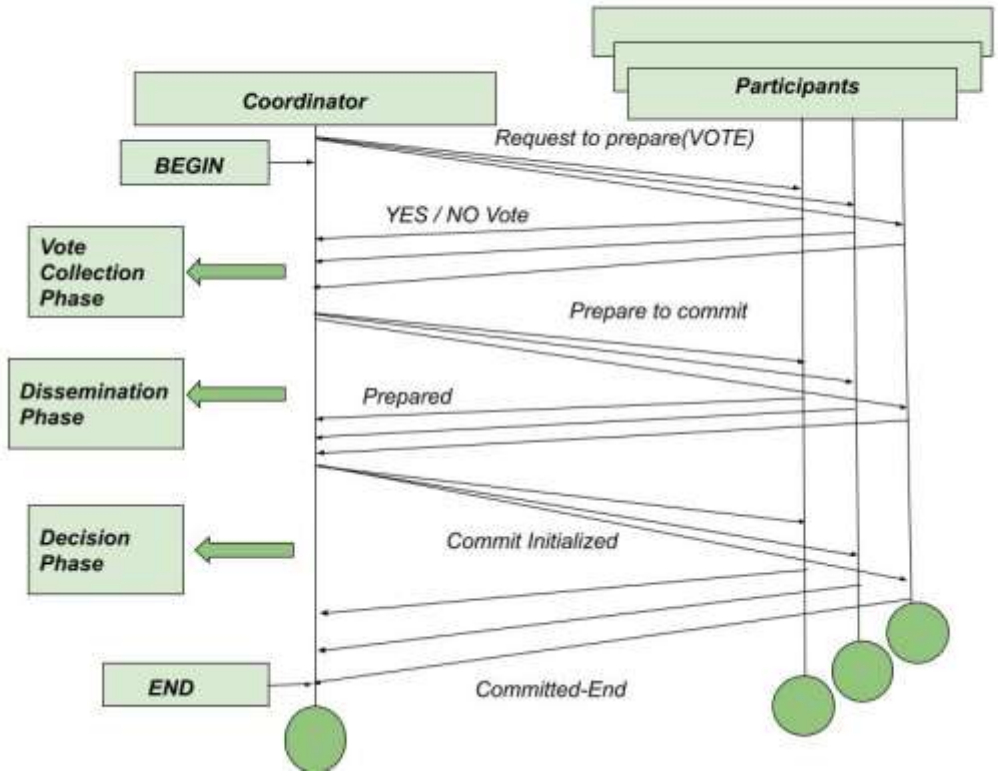


Diagrama mesajelor în protocolul 3PC.

Se observă faza suplimentară ("Dissemination Phase") marcată de mesajele Prepared trimise de coordonator și recunoscute de participanți, înainte de decizia finală. Acest pas asigură că participanții cunosc intenția de commit și pot acționa în absența coordonatorului, evitând blocarea completă.

Avantaje 3PC: Principala realizare este **caracterul ne-blocant** sub asumpțiile date – dacă coordonatorul cade, nodurile pot încă să ajungă la un consens privind commit/abort pe baza informațiilor din faza 2. Practic, 3PC se asigură că **stările intermediare** ale participanților permit distingerea între "coord. a decis commit" și "coord. nu a apucat să decidă" pe baza mesajelor primite, lucru imposibil în 2PC. Astfel, situația de tranzacție în-doubt blocată indefinit nu mai apare decât în cazuri extreme (de exemplu, dacă rețeaua se segmentează sever în același timp cu căderea coordonatorului – ceea ce iese din ipotezele protocolului).

Dezavantaje 3PC: În practică, 3PC este rar implementat integral în sisteme de baze de date comerciale. Motivul principal este că introduce mai multă **complexitate** și are propriile **limitări**:
- 3PC presupune anumite garanții ale sistemului: *latențe finite cunoscute* (pentru timeout-uri) și absența *partiționărilor de rețea* arbitrare. Dacă rețeaua suferă partiționare gravă (echivalent cu >k noduri căzute), protocolul poate totuși eșua similar cu 2PC, ducând la inconsistențe (posibil ca într-o partiție tranzacția să se comită și în alta să se aborteze)[10].
Deci 3PC **nu rezolvă problema partiționării** (conform teoremei CAP, nu se poate avea și consistență și disponibilitate în prezența partiționării rețelei)

- se focusează doar pe evitarea blocajelor în scenarii de eșec detectabil.
- Este mai *lent* (introduce încă o rundă de mesaje) și mai dificil de implementat corect. Coordonatorul trebuie să urmărească stări suplimentare, participanții de asemenea.
- Datorită acestor aspecte, 3PC **nu este utilizat pe scară largă** în platformele industriale obișnuite[10]. Majoritatea sistemelor preferă fie 2PC împreună cu mecanisme de failover/recuperare, fie trec direct la algoritmi de consens (Paxos/Raft) pentru coordonarea tranzacțiilor distribuite. 3PC rămâne totuși un concept important teoretic, ilustrând posibilitatea de a depăși blocajele lui 2PC cu costul unui protocol mai complex.

Bibliografie:

[1] ACID Properties In DBMS Explained | MongoDB | MongoDB
https://www.mongodb.com/resources/basics/databases/acid-transactions
[2] Notes on 2PC · Exactly Once
https://exactly-once.github.io/posts/notes-on-2pc/
[3] Transactions overview | Spanner | Google Cloud
https://cloud.google.com/spanner/docs/transactions
[4] ACID Transactions: The Cornerstone of Database Integrity | Yugabyte
https://www.yugabyte.com/key-concepts/acid-transactions/
[5] Transactions in NoSQL Databases: An Overview | by Avinash Kumar Singh | Medium
https://medium.com/@avinashsingh1152/transactions-in-nosql-databases-an-overview-353114651a76
[6] research.google.com
https://research.google.com/archive/spanner-osdi2012.pdf
[7] Two Phase Commit Protocol (Distributed Transaction Management) - GeeksforGeeks
https://www.geeksforgeeks.org/dbms/two-phase-commit-protocol-distributed-transaction-management/
[8] Recovery from failures in Two Phase Commit Protocol (Distributed Transaction) - GeeksforGeeks
https://www.geeksforgeeks.org/dbms/recovery-from-failures-in-two-phase-commit-protocol-distributed-transaction/
[9] Managing Distributed Transactions
https://docs.oracle.com/en/database/oracle/oracle-database/19/admin/managing-distributed-transactions.html
[10] Three Phase Commit Protocol - GeeksforGeeks
https://www.geeksforgeeks.org/dbms/three-phase-commit-protocol/