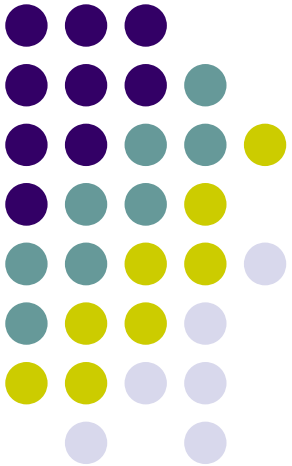
Κατανεμημένα Συστήματα

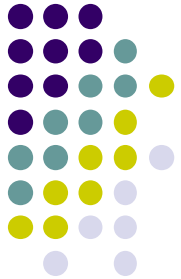
Μάθημα #10



Περιεχόμενα

1. Ανοχή σε βλάβες – Βασικές Έννοιες
2. Μοντέλα Αστοχιών
3. Συγκάλυψη αστοχιών μέσω υπερεπάρκειας
4. Συγκάλυψη Αστοχιών και Αναπαραγωγή
5. Κατανεμημένη Δέσμευση
6. Πρωτόκολλο Δέσμευσης δύο φάσεων (2PC)
7. Πρωτόκολλο Δέσμευσης Τριών Φάσεων (3PC)
8. Bitcoin & Blockchain

Ανοχή σε βλάβες – Βασικές Έννοιες

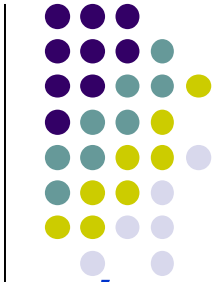


Η ανοχή σε βλάβες σχετίζεται στενά με την έννοια των φερέγγυων συστημάτων (dependable systems)

Η ποιότητα της λειτουργίας **ή η φερεγγυότητα (dependability)** ενός συστήματος εξαρτάται από:

- Τη Διαθεσιμότητά του (Availability)
- Την Αξιοπιστία του (Reliability)
- Την Ασφάλεια του (Safety)
- Τη Δυνατότητα Συντήρησής του (Maintainability)

Λάθη (faults) και Αποτυχίες (failures)



Ένα Λάθος είναι μια δυσλειτουργία (malfunction) η οποία μπορεί να προκύψει από:

- **Σφάλμα στον σχεδιασμό**
- **Σφάλμα στην κατασκευή**
- **Προγραμματιστικό σφάλμα**
- **Μη αναμενόμενη είσοδο**
- **Λειτουργικό σφάλμα**
- **Φυσική καταστροφή**
- **Φθορά λόγω της παρόδου του χρόνου**
- **Άλλους λόγους**

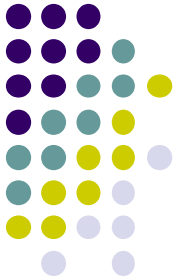
Δεν είναι απαραίτητο ένα λάθος να οδηγήσει αμέσως σε Αποτυχία ή Αστοχία (failure)

Μοντέλα Αστοχιών (failure models)

- **Αστοχία κατάρρευσης (crash failure)**
- **Αστοχία παράλειψης (omission failure)**
- **Χρονική αστοχία (timing failure)**
- **Αστοχία απόκρισης (response failure)**
- **Τυχαία αστοχία γνωστή ως Βυζαντινή αστοχία**

Type of failure	Description
Crash failure	A server halts, but is working correctly until it halts
Omission failure *Receive omission* *Send omission*	A server fails to respond to incoming requests A server fails to receive incoming messages A server fails to send messages
Timing failure	A server's response lies outside the specified time interval
Response failure *Value failure* *State transition failure*	A server's response is incorrect The value of the response is wrong The server deviates from the correct flow of control
Arbitrary failure	A server may produce arbitrary responses at arbitrary times

Τύποι Σφαλμάτων σε Επεξεργαστές



Αθόρυβα λάθη (Fail-silent faults)

- **Ο επεξεργαστής ο οποίος έχει το σφάλμα δεν ανταποκρίνεται. Σταματά να λειτουργεί. Ονομάζονται και fail-stop faults.**

Βυζαντινά λάθη (Byzantine faults)

- **Ο επεξεργαστής στον οποίο υπάρχει σφάλμα συνεχίζει τη λειτουργία του και είναι δυνατό να δίνει λάθος απαντήσεις σε ερωτήματα. Πολλές φορές δίνεται η εντύπωση ότι το σύστημα λειτουργεί σωστά.**

Συγκάλυψη Αστοχιών μέσω Υπερεπάρκειας

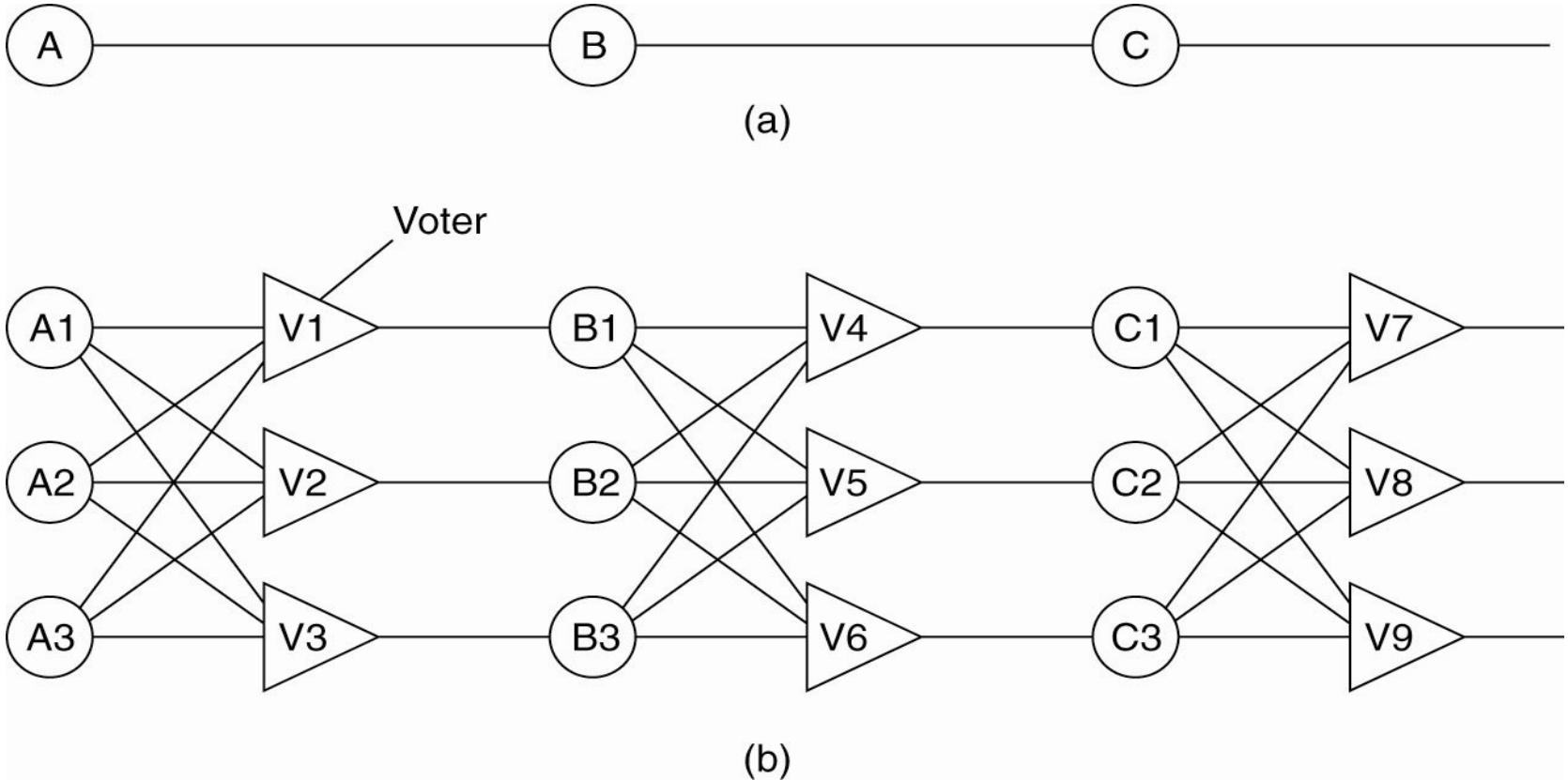
Μια γενική τεχνική αντιμετώπισης σφαλμάτων είναι η χρήση περίσσειας ή υπερεπάρκειας (redundancy) πόρων. Υπάρχουν τριών ειδών τέτοιοι πόροι: Πληροφορία, Χρόνος, Φυσικά μέρη (active or primary backup schemes)

1. **Υπερεπάρκεια πληροφοριών (προστίθενται επιπλέον bits)**
2. **Υπερεπάρκεια χρόνου (κάποιες ενέργειες εκτελούνται περισσότερες από μία φορά)**
3. **Φυσική υπερεπάρκεια (προστίθεται επιπλέον εξοπλισμός ή διεργασίες)**

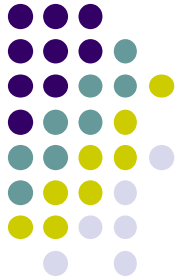
Με τον τρόπο αυτό το σύστημα είναι ανεκτικό σε βλάβες και είναι σε θέση να ανεχθεί την απώλεια ή τη δυσλειτουργία κάποιων συστατικών μερών του.

Παράδειγμα Υπερεπάρκειας στο Σχεδιασμό Κυκλωμάτων: *Τριπλή αρθρωτή υπερεπάρκεια (Triple Modular Redundancy - TMR)*

Κάθε voter είναι ένα κύκλωμα με 3 εισόδους και μία έξοδο. Αν δύο ή τρεις είσοδοι είναι ίδιες, η έξοδος είναι ίδια με τις εισόδους. Αν και οι τρεις είσοδοι είναι διαφορετικές, η έξοδος είναι απροσδιόριστη.



Ομάδες Διεργασιών



Βασική προσέγγιση για την ανοχή σε βλάβες: Η οργάνωση πολλών πανομοιότυπων διεργασιών σε ομάδα.

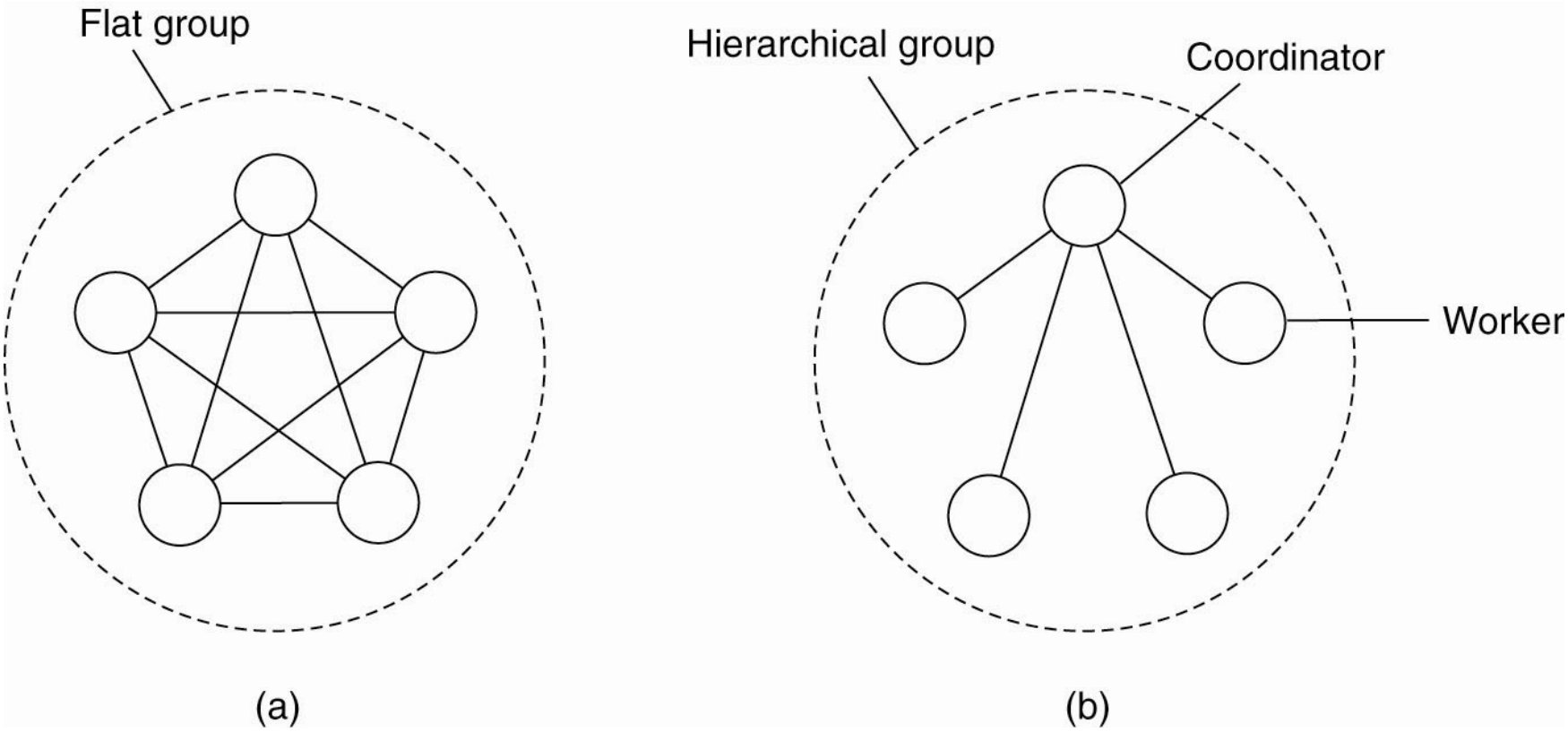
- **Αναπαραγωγή** μιας διεργασίας και οργάνωση των αντιγράφων σε μια ομάδα **ώστε να αντικατασταθεί μια μοναδική ευάλωτη διεργασία με μια ομάδα ανεκτική σε βλάβες**.
- Οι διεργασίες αντιμετωπίζουν μία ομάδα από άλλες διεργασίες ως μια οντότητα.
- Αν μια διεργασία της ομάδας αποτύχει με μεγάλη πιθανότητα μια άλλη θα μπορεί να αναλάβει στη θέση της.

Δομή ομάδων

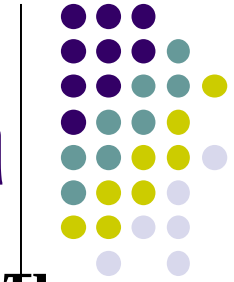
Επίπεδες ομάδες *(flat groups):* όλες οι αποφάσεις λαμβάνονται συλλογικά (διεξαγωγή ψηφοφορίας)

Ιεραρχικές ομάδες *(hierarchical groups)*: ύπαρξη διεργασίας-συντονιστή (coordinator) η οποία αποφασίζει ποια διεργασία-εργάτης (worker) θα διεκπεραιώσει κάθε αίτηση

Επικοινωνία στις Επίπεδες και στις Ιεραρχικές ομάδες



Συγκάλυψη Αστοχιών και Αναπαραγωγή

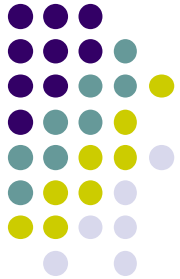


Χρήση ομάδων διεργασιών για την ανοχή σε βλάβες: Σε τι βαθμό θα πρέπει να εφαρμοστεί η αναπαραγωγή;

Ορισμός: Ένα σύστημα είναι *k*-ανεκτικό σε βλάβες αν μπορεί να ανταπεξέλθει σε βλάβες που παρουσιάζονται σε *k* συστατικά στοιχεία του.

- **Αν τα συστατικά στοιχεία (π.χ., διεργασίες) αποτύχουν αθόρυβα (σταματήσουν), τότε *k +1* διεργασίες αρκούν για την παροχή *k*-ανοχής σε βλάβες.**
- **Αν οι διεργασίες παρουσιάζουν Βυζαντινές αστοχίες (στέλνουν λανθασμένες ή τυχαίες απαντήσεις), τότε απαιτούνται *2k +1* διεργασίες για να επιτευχθεί *k*-ανοχή σε βλάβες.**

Συμφωνία στα κατανεμημένα συστήματα

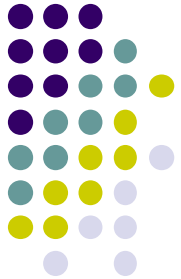


Σε πολλές περιπτώσεις κατανεμημένων συστημάτων χρειάζεται οι διεργασίες να φθάσουν σε συμφωνία.
Παραδείγματα: επιλογή συντονιστή, απόφαση για τη δέσμευση ή όχι μιας συναλλαγής, καταμερισμός καθηκόντων, συγχρονισμός

Στην περίπτωση που δεν υπάρχουν σφάλματα στην επικοινωνία και στους επεξεργαστές τότε η επίτευξη συναίνεσης είναι απλή υπόθεση.

Τι συμβαίνει όμως στην περίπτωση που στο σύστημα υπάρχουν σφάλματα;

Συμφωνία σε κατανεμημένα συστήματα με σφάλματα

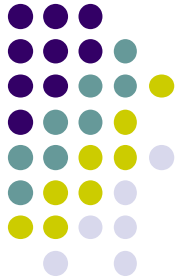


Ο γενικός στόχος των κατανεμημένων αλγορίθμων συμφωνίας είναι:

Όλες οι μη εσφαλμένες διεργασίες να καταλήξουν σε συμφωνία σε πεπερασμένο αριθμό βημάτων. Υπάρχουν διάφορες περιπτώσεις οι οποίες σχετίζονται με διάφορες παραμέτρους του συστήματος, όπως:

- **Τα μηνύματα παραδίνονται αξιόπιστα πάντα;**
- **Υπάρχει περίπτωση να προκύψουν σφάλματα σε διεργασίες; Αν ναι τί τύπου σφάλματα (Αθόρυβα ή Βυζαντινά);**
- **Τι είναι το σύστημα, σύγχρονο ή ασύγχρονο;**

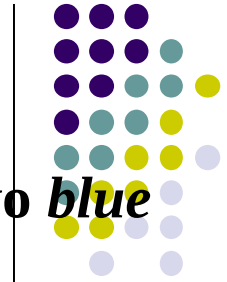
Συμφωνία στην περίπτωση αξιόπιστων διεργασιών και μη αξιόπιστης επικοινωνίας



Πρόβλημα των δύο στρατών (two-army problem):

δείχνει τη δυσκολία επίτευξης συμφωνίας ακόμα και μεταξύ δύο τέλειων διεργασιών σχετικά με 1 bit πληροφορίας, στην περίπτωση μη αξιόπιστης επικοινωνίας

The two-army problem

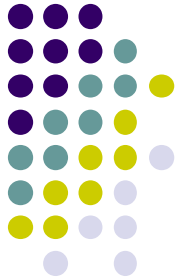


- **There is a *red army* of 5k troops encamped in a valley, and two *blue armies* of 3k troops each, on surrounding hills.**

- **The red army wins blue armies if they do not cooperate but loses to them if they cooperate.**

- **General Alexander, commander of blue army 1, sends General Bonaparte, commander of blue army 2, a message: “Hi Bona, let’s attack at dawn tomorrow”**
- **General Bonaparte answers: “Great idea Al, see you at dawn tomorrow”**
- **General Alexander receives the message, but suddenly he realizes: Bonaparte does not know whether I got his acknowledgement and may not dare to attack. Therefore he sends a messenger to tell Bonaparte that he received the acknowledgement.**
- **Bonaparte receives the new message and thinks. Alexander does not know that received the message and may not dare to attack. Therefore he sends a messenger ...**

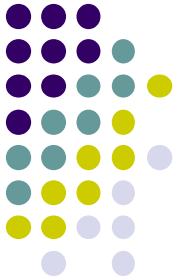
The two-army problem



If the arrival of the messenger is not acknowledged, then

the other general may think that the messenger was captured by the enemy and the ally did not get the message, and hence it is not safe to attack. Hence, the generals cannot rely communication, messengers move back and forth until eternity, even though nothing is wrong.

Συμφωνία στην περίπτωση μη αξιόπιστων διεργασιών και αξιόπιστης επικοινωνίας



Πρόβλημα των Βυζαντινών στρατηγών οι οποίοι πρέπει να συντονίσουν την επίθεση τους έναντι ενός κοινού εχθρού. Κάποιοι από τους στρατηγούς μπορεί να είναι προδότες.

Έστω *n* στρατηγοί εκ των οποίων *m* είναι προδότες. Κάθε στρατηγός γνωρίζει τη δύναμη του στρατού του οποίου ηγείται.

Σκοπός όλοι οι στρατηγοί να έχουν ένα διάνυσμα μήκους n τέτοιο ώστε κάθε στοιχείο j να περιέχει τη δύναμη του στρατεύματος του στρατηγού j, αν ο j είναι πιστός (αν ο j είναι προδότης τότε το στοιχείο j είναι μη ορισμένο δηλ. μπορεί να έχει οποιαδήποτε τιμή).

Αλγόριθμος του Lamport κ.ά (1982)

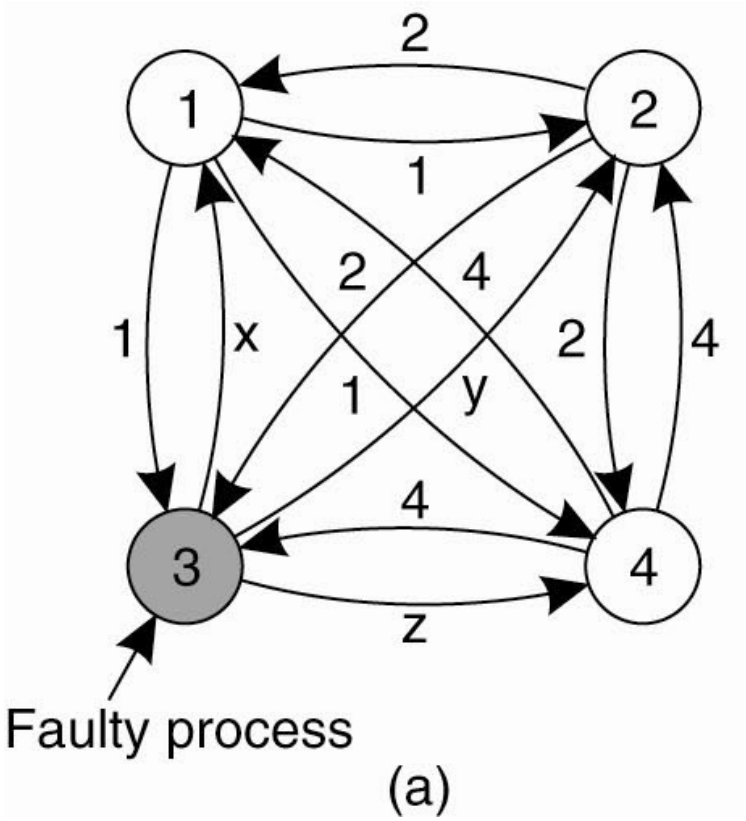
(a) Κάθε στρατηγός ανακοινώνει σε όλους τους άλλους στρατηγούς τη δύναμη του στρατεύματός του. ***Οι πιστοί στρατηγοί λένε την αλήθεια. Οι προδότες μπορούν να πουν σε κάθε άλλο στρατηγό ένα διαφορετικό ψέμα.***

(b) Κάθε στρατηγός συλλέγει σε ένα διάνυσμα τις πληροφορίες που του έστειλαν οι άλλοι στο βήμα (a).

(c) Κάθε στρατηγός στέλνει σε όλους τους άλλους το διάνυσμά του.

(d) Κάθε στρατηγός ελέγχει για κάθε i τις i-οστές θέσεις των διανυσμάτων, που έλαβε, και αν μια τιμή εμφανίζεται περισσότερες φορές από τις άλλες, την κρατά στο τελικό του διάνυσμα. Αν όχι τότε θέτει UnKnown σε αυτήν τη θέση του τελικού διανύσματος.

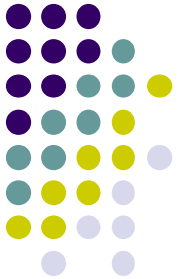
Παράδειγμα για n=4 και m=1



Το πρόβλημα των (Βυζαντινών) στρατηγών τρείς από τους οποίους είναι πιστοί και ένας ψεύτης. Οι αριθμοί στα τόξα του γραφήματος είναι οι χιλιάδες των στρατιωτών του κάθε στρατού.

(a) Κάθε στρατηγός ανακοινώνει σε όλους τους άλλους στρατηγούς τις χιλιάδες των στρατιωτών του.

Παράδειγμα για n=4 και m=1 (3 πιστοί στρατηγοί και 1 προδότης)



1	Got(1, 2, x, 4)
2	Got(1, 2, y, 4)
3	Got(1, 2, 3, 4)
4	Got(1, 2, z, 4)

(b)

1 Got	2 Got	4 Got
(1, 2, y, 4)	(1, 2, x, 4)	(1, 2, x, 4)
(a, b, c, d)	(e, f, g, h)	(1, 2, y, 4)
(1, 2, z, 4)	(1, 2, z, 4)	(i, j, k, l)

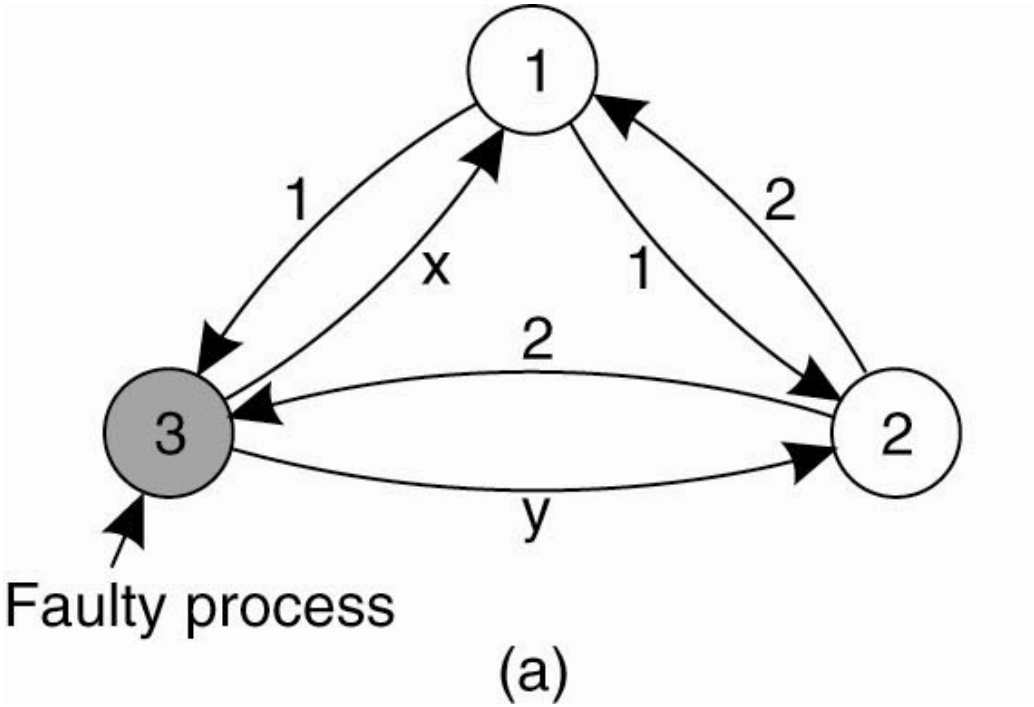
(c)

(b) Τα διανύσματα που έχει κάθε στρατηγός με βάση το (a).
(c) Τα διανύσματα που λαμβάνει κάθε στρατηγός στο τρίτο βήμα.

Οι στρατηγοί 1,2 και 4 φθάνουν όλοι σε συμφωνία για το διάνυσμα:

(1,2,Unknown,4)

Παράδειγμα για n=3 και m=1 (2 πιστοί στρατηγοί και 1 προδότης)



1 Got(1, 2, x)
2 Got(1, 2, y)
3 Got(1, 2, 3)

(b)

1 Got	2 Got
(1, 2, y)	(1, 2, x)
(a, b, c)	(d, e, f)

(c)

Καμία τιμή δεν έχει πλειοψηφία και επομένως, δεν επιτυγχάνεται συμφωνία!

Πρόβλημα των Βυζαντινών στρατηγών (η τυπική εκδοχή)

Byzantine Generals Problem. A commanding general must send an order to his $n - 1$ lieutenant generals such that

IC1. All loyal lieutenants obey the same order.

IC2. If the commanding general is loyal, then every loyal lieutenant obeys the order he sends.

Μόνο ένας επεξεργαστής έχει είσοδο, ο Στρατηγός, και πρέπει να τη μεταδώσει στους υπόλοιπους. Όλοι οι επεξεργαστές (λοχαγοί) γνωρίζουν τον στρατηγό. Υπάρχουν το πολύ m επεξεργαστές που μπορεί να είναι Βυζαντινοί. Ανάμεσά τους μπορεί να είναι και ο Στρατηγός. Οι επεξεργαστές δεν γνωρίζουν αν ο στρατηγός είναι βυζαντινός ή όχι.

- Παράδειγμα για n=3 και m=1 (1 προδότης – είτε ο στρατηγός είτε ένας από τους λοχαγούς) (βλ. επόμενες διαφάνειες)
- Καμία τιμή δεν έχει πλειοψηφία και επομένως, δεν επιτυγχάνεται συμφωνία! Τι γίνεται αν n=4 και m=1 ?

Η παραπάνω λογική γενικεύεται και για n>3 και αποδεικνύεται ότι σε ένα σύστημα με m προβληματικές διεργασίες μπορεί να επιτευχθεί συμφωνία αν υπάρχουν 2m+1 αξιόπιστες διεργασίες (συνολικός αριθμός διεργασιών 3m+1) (Lamport et al, 1982)

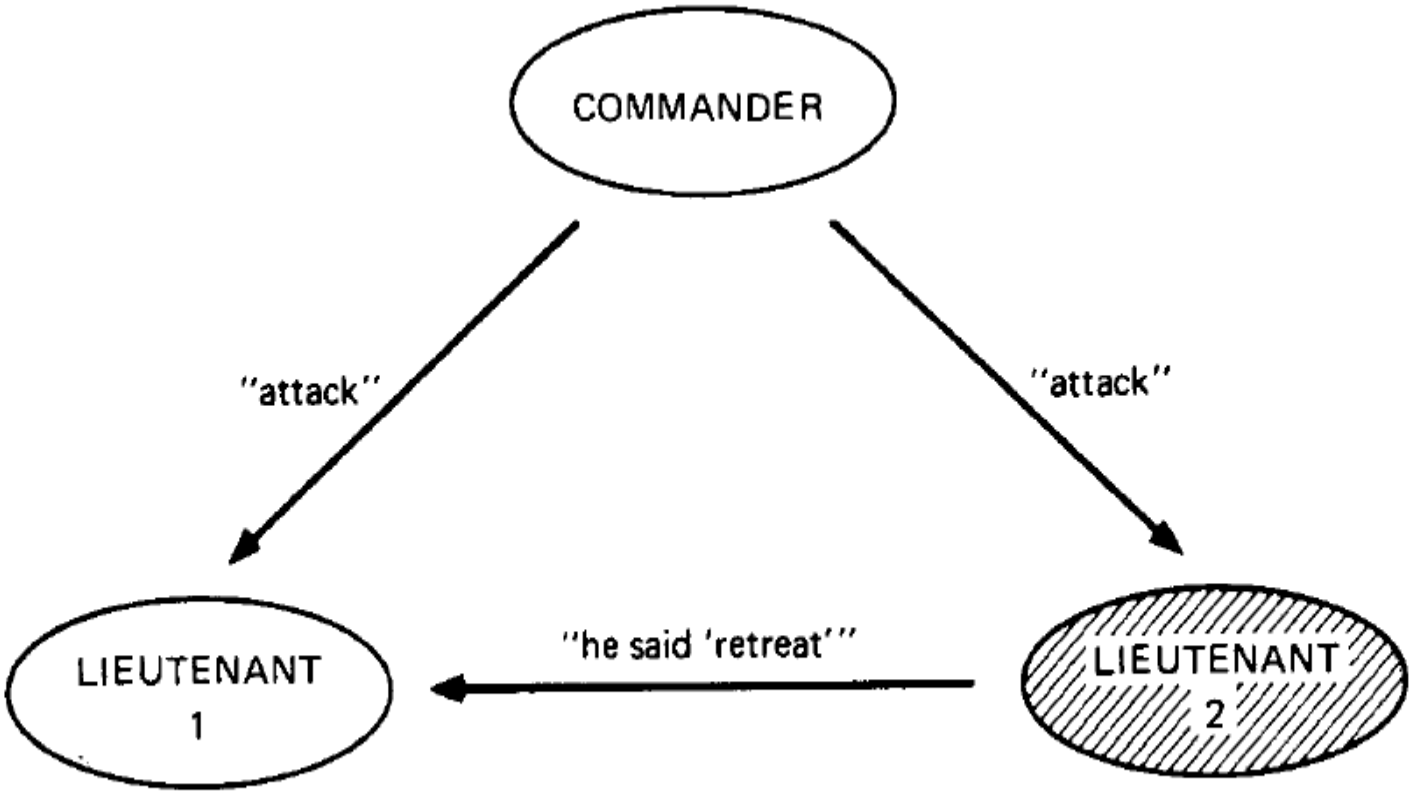


Fig. 1. Lieutenant 2 a traitor.

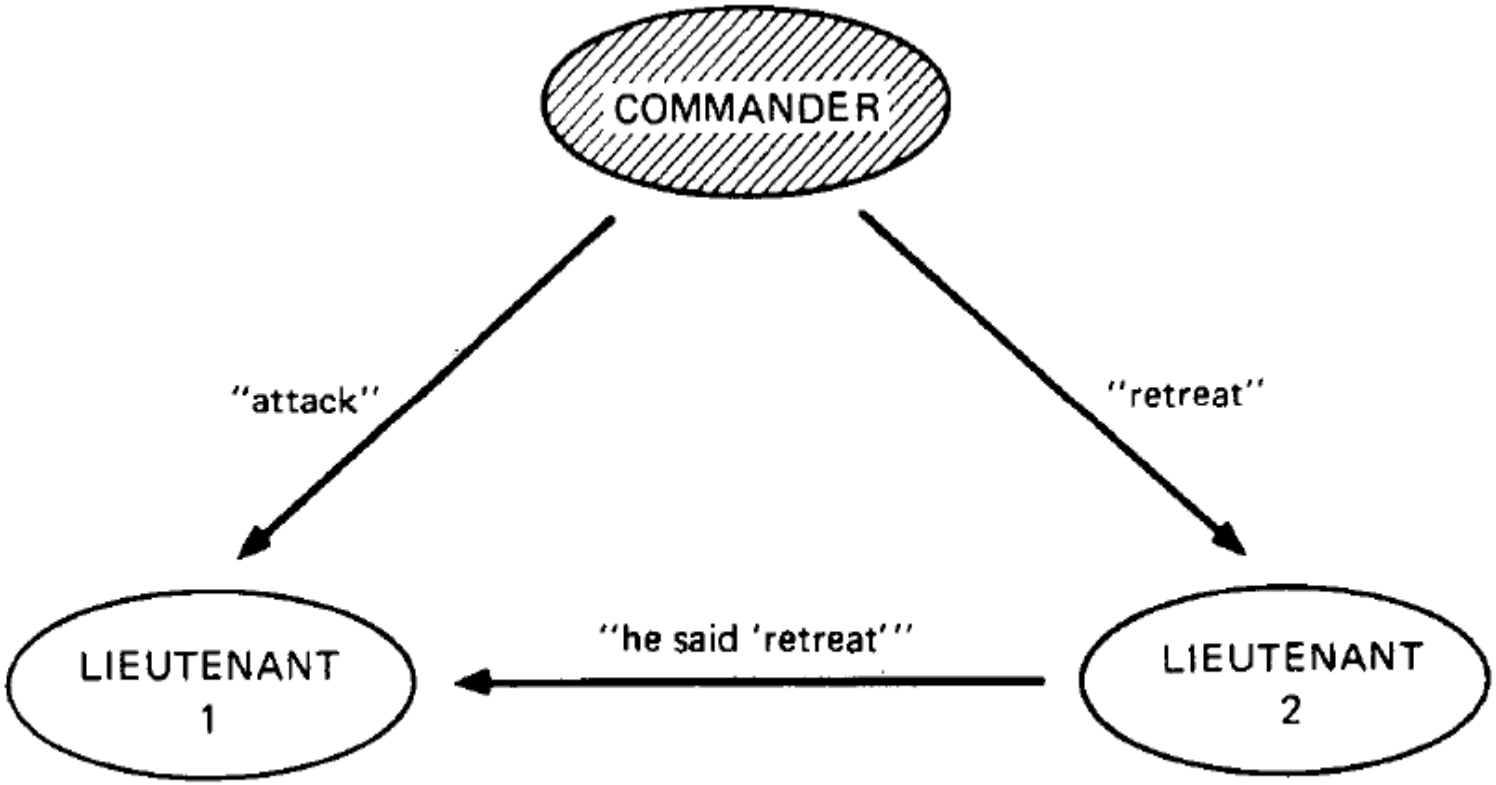
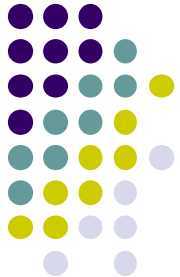


Fig. 2. The commander a traitor.

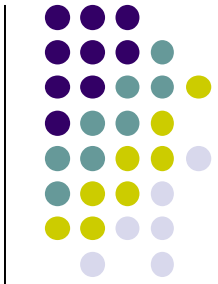
Σκοπός ενός αλγορίθμου consensus



- Ανοχή σε σφάλματα
  - Δεν μπλοκάρει όταν η πλειονότητα των διεργασιών δουλεύουν
- Συμφωνία σε ένα αποτέλεσμα ανάμεσα σε ομάδα διεργασιών ακόμα κι αν:
  - Κάποιες διεργασίες πεθάνουν
  - Κάποια μηνύματα χαθούν ή έρθουν εκτός σειράς
  - Αν παραδοθούν, τα μηνύματα δεν αλλοιώνονται

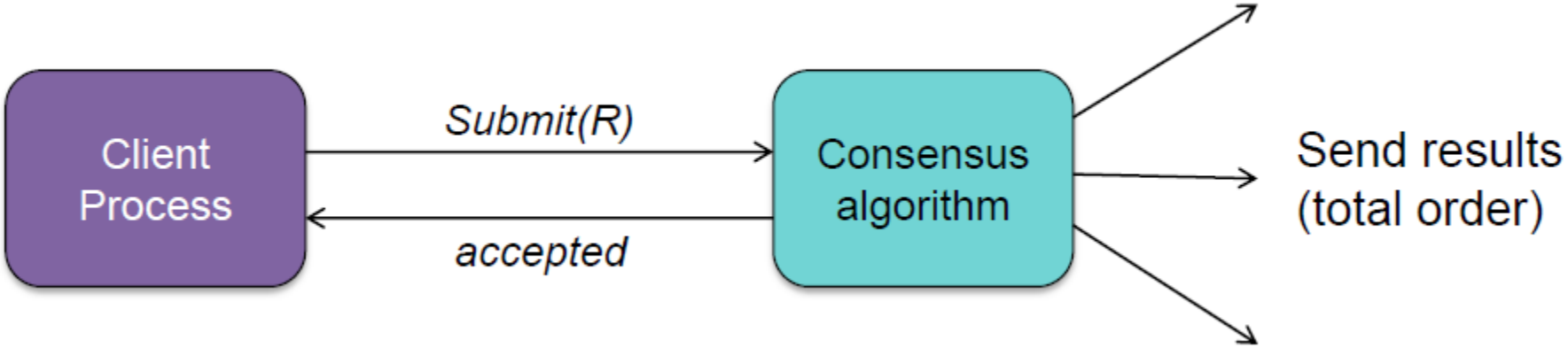
- PAXOS: Ένας αλγόριθμος consensus
  - Δημιουργήθηκε από τον Leslie Lamport
  - Ένας από τους πιο αποδοτικούς και κομψούς αλγορίθμους
  - Πασίγνωστος - πολλά πραγματικά συστήματα υλοποιούν το Paxos
    - Google Chubby
    - MS Bing cluster management
    - AWS

Υποθέσεις – Επιθυμητές Ιδιότητες



- Το δίκτυο είναι ασύγχρονο με καθυστερήσεις στα μηνύματα
- Το δίκτυο μπορεί να χάσει μηνύματα ή να τα στείλει περισσότερες από μια φορές αλλά δεν μπορεί να τα αλλοιώσει
- Οι διεργασίες μπορεί να κρασάρουν (και να σταματήσουν)
- Οι διεργασίες έχουν *permanent storage* (δίσκο)
- Οι διεργασίες μπορούν να προτείνουν τιμές
- Ο στόχος είναι όλες οι διεργασίες να συμφωνήσουν σε μία από τις προτεινόμενες τιμές
- Safety
  - Μπορεί να επιλεγεί μόνο τιμή που έχει προταθεί
  - Επιλέγεται μια μόνο τιμή
  - Μια διεργασία μαθαίνει μια τιμή μόνο αν έχει

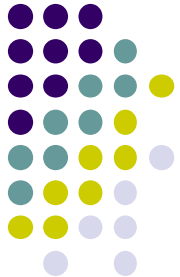
Από τη μεριά του Χρήστη



while (submit_request(R) != ACCEPTED) ;

Το R θα μπορούσε να είναι ένα key:value ζεύγος μιας βάσης

Βασικές Συνιστώσες – Διεργασίες στο PAXOS



- **Client:**
  - Στέλνει ένα αίτημα
- **Proposers:**
  - Λαμβάνουν αίτημα από τον client και τρέχουν το πρωτόκολλο
  - Leader: Εκλεγμένος coordinator ανάμεσα στους proposers (δεν είναι απαραίτητος, απλώς απλοποιεί τη διάταξη μηνυμάτων και διασφαλίζει ότι δεν υπάρχει διαφωνία)
- **Acceptors:**
  - Πολλαπλές διεργασίες που θυμούνται το state του πρωτοκόλλου
  - Quorum = πλειονότητα από acceptors
- **Learners:**
  - Όταν οι acceptors συμφωνήσουν, ο learner εκτελεί το αίτημα ή/και στέλνει απάντηση στον client

Έχει επεκταθεί για την κάλυψη και Βυζαντινών αστοχιών

Βλ. https://www.cs.rutgers.edu/~pxk/417/notes/paxos.

Κατανεμημένη Δέσμευση

Κατανεμημένη δέσμευση: αφορά την εκτέλεση μιας λειτουργίας είτε από όλες τις εργασίες μιας ομάδας είτε από καμία.

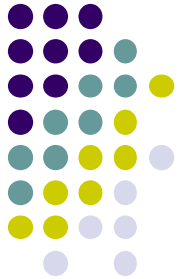
Μία απλή μέθοδος στην κατανεμημένη δέσμευση: η χρήση συντονιστή ο οποίος ειδοποιεί όλες τις διεργασίες να εκτελέσουν (τοπικά) ή όχι την εν λόγω λειτουργία (Πρωτόκολλο δέσμευσης μίας φάσης – one phase commit)

Μειονέκτημα πρωτοκόλου μίας φάσης: κάποια διεργασία μπορεί να μην είναι σε θέση να εκτελέσει τη λειτουργία.

Περισσότερο εξελιγμένες μέθοδοι:

- **Πρωτόκολλο δέσμευσης δύο φάσεων (two phase commit protocol, 2PC)**
- **Πρωτόκολλο δέσμευσης τριών φάσεων (three phase commit protocol, 3PC)**

Πρωτόκολλο Δέσμευσης δύο φάσεων (2PC)



Φάση 1

1α. Ο συντονιστής στέλνει αίτηση ψήφου (VOTE_REQUEST) σε όλους τους συμμετέχοντες.

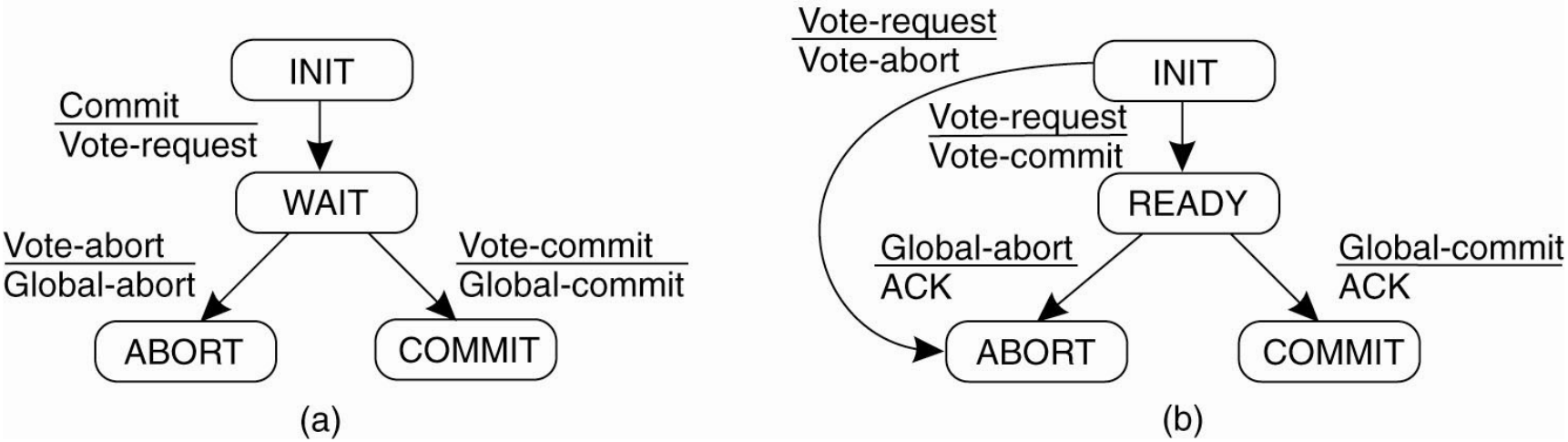
1β. Κάθε συμμετέχων που λάβει το μήνυμα επιστρέφει στο συντονιστή είτε ψήφο για δέσμευση (VOTE_COMMIT) είτε ψήφο για ακύρωση (VOTE_ABORT)

Φάση 2

2α. Αν όλοι οι συμμετέχοντες έχουν στείλει VOTE_COMMIT, ο συντονιστής στέλνει σε όλους καθολική δέσμευση (GLOBAL_COMMIT). Διαφορετικά, στέλνει καθολική ακύρωση (GLOBAL_ABORT)

2β. Κάθε συμμετέχων που έστειλε VOTE_COMMIT περιμένει μήνυμα από το συντονιστή. Αν λάβει GLOBAL_COMMIT δεσμεύει τοπικά τη συναλλαγή. Αν λάβει GLOBAL_ABORT η συναλλαγή ματαιώνεται τοπικά.

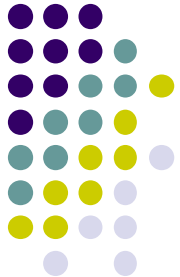
Πρωτόκολλο Δέσμευσης δύο φάσεων (2PC)



(a) Μηχανή πεπερασμένων καταστάσεων (finite state machine) για το συντονιστή.

(b) Μηχανή πεπερασμένων καταστάσεων για έναν συμμετέχοντα

2PC - Αστοχίες στις διεργασίες



Το πρωτόκολλο αποτυγχάνει αν μια διεργασία καταρρεύσει ενώ οι άλλες περιμένουν μήνυμα απ' αυτήν.

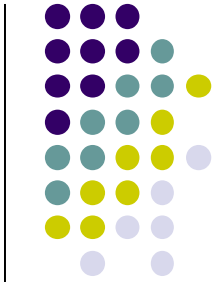
Λύση: Χρησιμοποιούνται μηχανισμοί χρονικών προθεσμιών

Παραδείγματα

- **Ο συντονιστής μπορεί να μπλοκαριστεί στην κατάσταση WAIT. Αν δεν λάβει όλες τις ψήφους έπειτα από κάποιο διάστημα στέλνει σε όλους GLOBAL_ABORT.**
- **Ένας συμμετέχων P μπορεί να μπλοκαριστεί στην κατάσταση READY. Δεν μπορεί να ματαιώσει αμέσως τοπικά τη συναλλαγή. Πρέπει να διαπιστώσει ποιο μήνυμα έστειλε ο συντονιστής.**

Λύση: Ο P επικοινωνεί με άλλον συμμετέχοντα Q για να εξετάσει αν μπορεί να αποφασίσει με βάση την τρέχουσα κατάσταση του Q.

2PC - Αστοχίες στις διεργασίες



State of Q	Action by P
COMMIT	Make transition to COMMIT
ABORT	Make transition to ABORT
INIT	Make transition to ABORT
READY	Contact another participant

Ενέργειες του συμμετέχοντα P όταν βρίσκεται στην κατάσταση READY και επικοινωνήσει με τον Q.

- **Αν διαπιστωθεί ότι όλοι οι συμμετέχοντες βρίσκονται στην κατάσταση READY δεν μπορεί να ληφθεί καμία απόφαση έως ότου ανακάμψει ο συντονιστής.**
- **Προκειμένου οι διεργασίες να μπορούν πραγματικά να ανακάμπτουν θα πρέπει να καταγράφουν την κατάστασή τους.**

«Μπλοκάρισμα» στο 2PC

- **Αν όλοι οι συμμετέχοντες βρίσκονται στην κατάσταση READY δεν μπορεί να ληφθεί καμία απόφαση έως ότου ανακάμψει ο συντονιστής.**
- **Στην περίπτωση αυτή οι συμμετέχοντες «μπλοκάρονται» και γι' αυτό το 2PC καλείται και Πρωτόκολλο Δέσμευσης με Μπλοκάρισμα (blocking commit protocol)**

Λύση #1: Κάθε αποδέκτης ενός μηνύματος στέλνει αμέσως με πολυεκπομπή σε όλες τις άλλες διεργασίες κάθε μήνυμα που λαμβάνει. Η προσέγγιση αυτή επιτρέπει στους συμμετέχοντες να καταλήξουν σε απόφαση ακόμη και αν ο συντονιστής δεν έχει ανακάμψει.

Λύση #2: Πρωτόκολλο δέσμευσης τριών φάσεων (3PC)

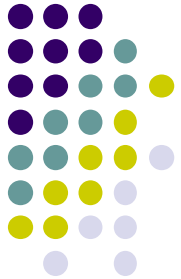
Πρωτόκολλο Δέσμευσης Τριών Φάσεων (3PC)

Οι καταστάσεις του συντονιστή και κάθε συμμετέχοντα ικανοποιούν τις εξής δύο συνθήκες:

1. **Δεν υπάρχει καμία κατάσταση από την οποία να είναι δυνατή η απευθείας μετάβαση είτε σε μια κατάσταση COMMIT ή σε μια κατάσταση ABORT.**
2. **Δεν υπάρχει καμία κατάσταση από την οποία να μην είναι δυνατό να ληφθεί μια τελική απόφαση, και από την οποία να μπορεί να πραγματοποιηθεί μετάβαση σε μια κατάσταση COMMIT.**

Οι δύο αυτές συνθήκες είναι αναγκαίες και ικανές για την ύπαρξη ενός πρωτοκόλλου δέσμευσης χωρίς μπλοκάρισμα.

Πρωτόκολλο Δέσμευσης Τριών Φάσεων (3PC)



Φάση 1

1α. Ο συντονιστής στέλνει αίτηση ψήφου (VOTE_REQUEST) σε όλους τους συμμετέχοντες.

1β. Κάθε συμμετέχων που λαμβάνει το μήνυμα επιστρέφει στο συντονιστή είτε ψήφο για δέσμευση (VOTE_COMMIT) είτε ψήφο για ακύρωση (VOTE_ABORT)

Φάση 2

2α. Αν όλοι οι συμμετέχοντες έχουν στείλει VOTE_COMMIT, ο συντονιστής στέλνει σε όλους μήνυμα PREPARE_COMMIT. Διαφορετικά, στέλνει καθολική ακύρωση (GLOBAL_ABORT)

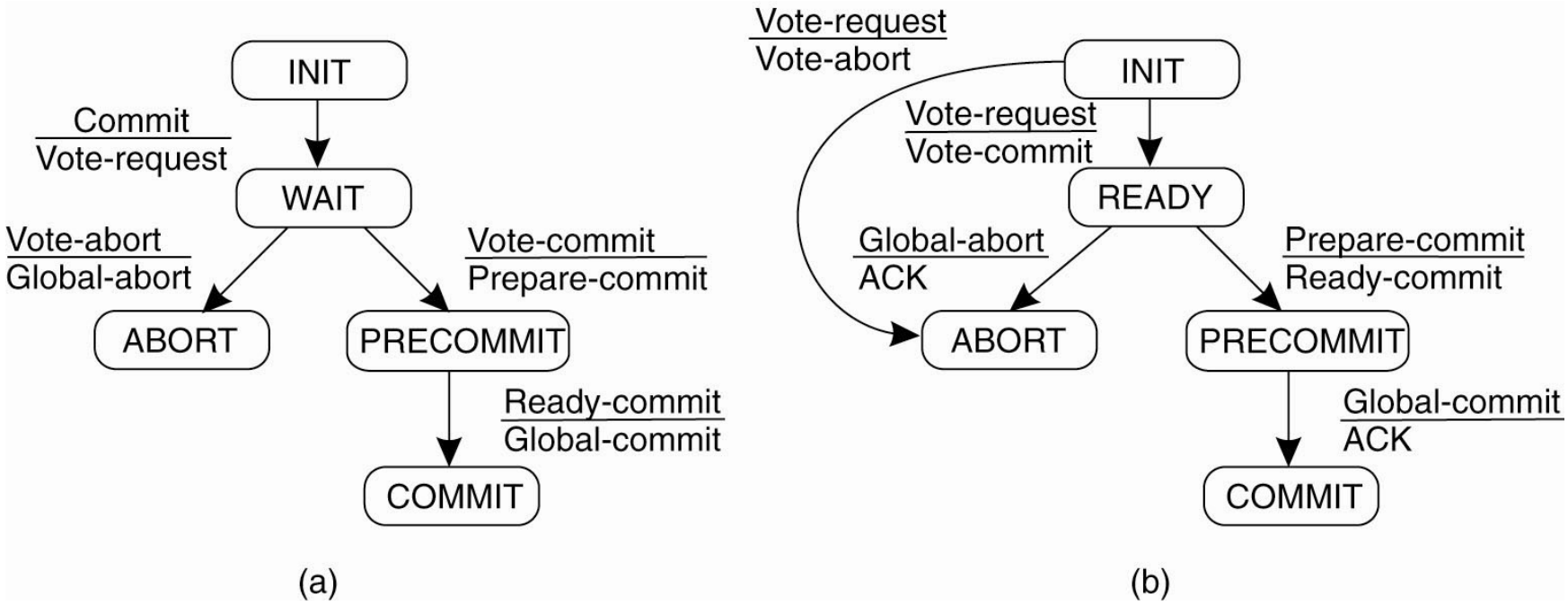
2β. Κάθε συμμετέχων που έστειλε VOTE_COMMIT περιμένει μήνυμα από το συντονιστή. Αν λάβει PREPARE_COMMIT στέλνει μήνυμα επιβεβαίωσης ότι μπορεί να δεσμευθεί (READY_COMMIT). Αν λάβει GLOBAL_ABORT η συναλλαγή ματαιώνεται τοπικά.

Φάση 3

3α. Όταν κάθε συμμετέχων επιβεβαιώσει ότι μπορεί να δεσμευθεί, ο συντονιστής θα στείλει το τελικό μήνυμα καθολική δέσμευσης (GLOBAL_COMMIT).

3β. Κάθε συμμετέχων που έστειλε READY_COMMIT περιμένει μήνυμα από το συντονιστή. Αν λάβει GLOBAL_COMMIT δεσμεύει τοπικά τη συναλλαγή. .

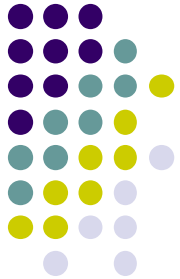
Πρωτόκολλο Δέσμευσης Τριών Φάσεων (3PC)



(a) Μηχανή πεπερασμένων καταστάσεων (finite state machine) για το συντονιστή.

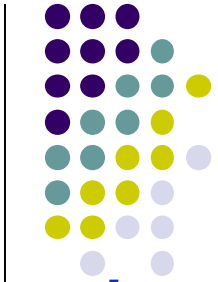
(b) Μηχανή πεπερασμένων καταστάσεων για έναν συμμετέχοντα

Πρωτόκολλο Δέσμευσης Τριών Φάσεων (3PC)



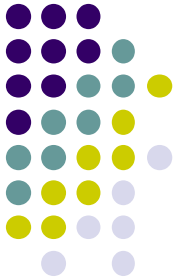
- **Η βασική ιδέα είναι ότι στο διάγραμμα κατάστασης που οδηγεί σε commit, ο συντονιστής και οι συμμετέχοντες να μην διαφέρουν περισσότερο από μία κατάσταση.**

3PC - Αστοχίες στις διεργασίες



- **Ο συντονιστής** μπλοκάρεται **στην κατάσταση WAIT. Αν δεν λάβει όλες τις ψήφους έπειτα από κάποιο διάστημα στέλνει σε όλους GLOBAL_ABORT.**
- **Ο συντονιστής** μπλοκάρεται στην **κατάσταση PRECOMMIT** διότι ένας από τους συμμετέχοντες (έστω ο p) έχει καταρρεύσει: **ο συντονιστής μπορεί να στείλει στους υπόλοιπους GLOBAL_COMMIT διότι ο p έχει ήδη ψηφίσει υπέρ της δέσμευσης και όταν ανακάμψει θα ολοκληρώσει τη δέσμευση.**

3PC - Αστοχίες στις διεργασίες

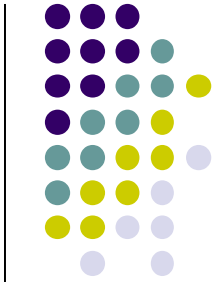


Ένας συμμετέχων P μπορεί να μπλοκαριστεί στην κατάσταση READYή στην PRECOMMIT. Δεν μπορεί να ματαιώσει αμέσως τοπικά τη συναλλαγή. Πρέπει να διαπιστώσει ποιο μήνυμα έστειλε ο συντονιστής.

Λύση: Ο P επικοινωνεί με άλλον συμμετέχοντα Q για να εξετάσει αν μπορεί να αποφασίσει με βάση την κατάσταση του Q.

Κατάσταση του Q	Ενέργεια από τον P
COMMIT	COMMIT
ABORT	ABORT
INIT	ABORT

3PC - Αστοχίες στις διεργασίες

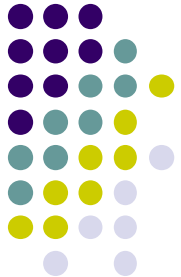


Αν κάθε συμμετέχων με τον οποίο μπορεί να επικοινωνήσει ο P βρίσκεται στην κατάσταση READY τότε η συναλλαγή ματαιώνεται.

Κανένας συμμετέχων που βρίσκεται σε λειτουργία, δεν γνωρίζει ποια θα είναι η κατάσταση του συμμετέχοντα που κατέρρευσε όταν αυτός ανακάμψει:

- **Αν ανακάμψει στην INIT η μόνη σωστή απόφαση είναι η ματαίωση της συναλλαγής.**
- **Αν ανακάμψει στην PRECOMMIT και πάλι η ματαίωση της συναλλαγής δεν δημιουργεί πρόβλημα.**

3PC - Αστοχίες στις διεργασίες



Αν κάθε συμμετέχων με τον οποίο μπορεί να επικοινωνήσει ο P βρίσκεται στην κατάσταση PRECOMMIT τότε είναι ασφαλές να δεσμευθεί η συναλλαγή.

Όλες οι άλλες θα ανακάμψουν στην κατάσταση READY, PRECOMMIT ή COMMIT που βρίσκονταν κατά τη στιγμή της κατάρρευσής τους.

Bitcoin & Blockchain Opportunities and Risks

by Patrick Valduriez

Bitcoin

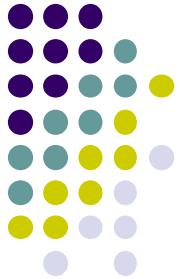
- Bitcoin: A Peer-to-Peer Electronic Cash System
  - Satoshi Nakamoto (pseudo), Oct. 31, 2008 (Halloween)
  - Cryptocurrency and payment system
  - Blockchain is the infrastructure
- Since then
  - Many blockchains: Etherum in 2013, Ripple in 2014, etc.
  - Increasing use for high-risk investment
    - Initial Coin Offerings
  - But also in fraudulent or illegal activities !
    - Scam, purchase on the dark web, money laundering, tax evasion, ...
  - Warnings from market authorities and beginning of regulation (China, South Korea

The Currency of Tomorrow?

- Pros
  - Low transaction fee (set by the sender to speed up processing)
  - Fewer risks for merchants (no fraudulent chargebacks)
  - Security and control (protection from identity theft)
  - Trust through the blockchain
- Cons
  - Unstable: no backing by a state or fed bank
  - Unrelated to real economy
  - High volatility, e.g. between 6K and 7K$ in 3 hours
  - Relatively small user base: 50+ million bitcoin owners
    - Versus billions of users of e-payment systems like AliPay and Paypal
- The Crypto Bubble (2017)*
  - Bitcoin price increased from $1k to 10K, then peaked almost at $20K in December 2017 to collapse 4 months later to below $6k (down 70% from the peak), and close to $6k and $7k in 2018, 19, then up again in 2020 due to covid (today $30-40k!)

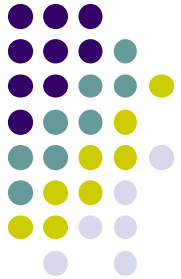
* Testimony for the Hearing of the US Senate Committee on Banking, Housing and Community Affairs On "Exploring the Cryptocurrency and Blockchain Ecosystem", Nouriel Roubini (NYU)

Blockchain Concepts



- Blockchain
  - An *immutable* distributed database, i.e. a log of blocks, which are linked and replicated on *full nodes*
- A block
  - Digital container for transactions, contracts, property titles, etc.
  - Transactions are secured using public key encryption
- The code of each new block is built on that of the preceding block
  - Guarantees that it cannot be changed or tampered
- The blockchain is viewed by all participants
  - Enables validating the entries in the blocks
  - Privacy: users are pseudonomyzed

Blockchain Concepts

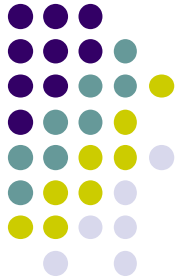


Blockchain is a system of recording information in a way that makes it difficult or impossible to change, hack, or cheat the system.

A blockchain is essentially a digital ledger of transactions that is duplicated and distributed across the entire network of computer systems on the blockchain. **Each block in the chain** contains a number of transactions, and every time a new transaction occurs on the blockchain, a record of that transaction is added to every participant's ledger. The decentralised database managed by multiple participants is known as Distributed Ledger Technology (DLT).

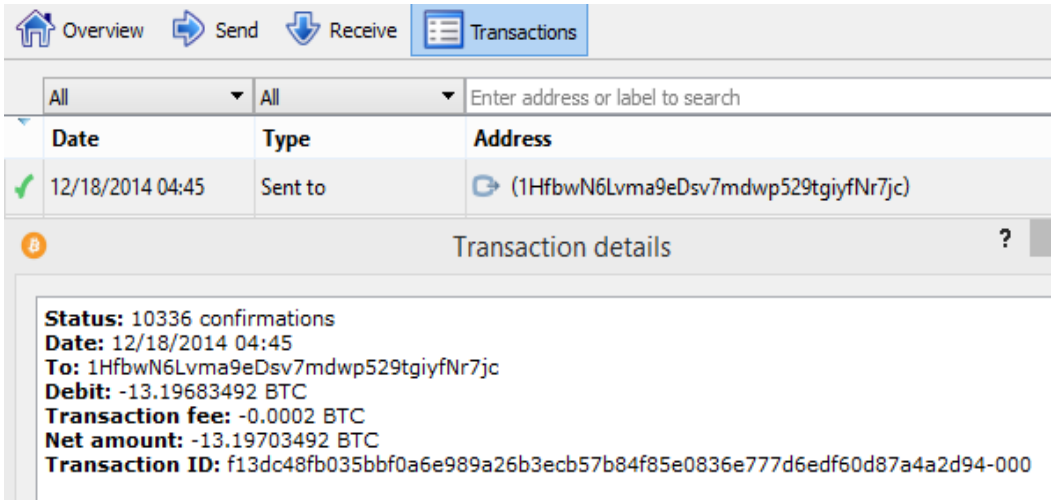
Blockchain is a type of DLT in which transactions are recorded with an immutable cryptographic signature called a ***hash***.

Blockchain Protocol (Nakamoto 2008)

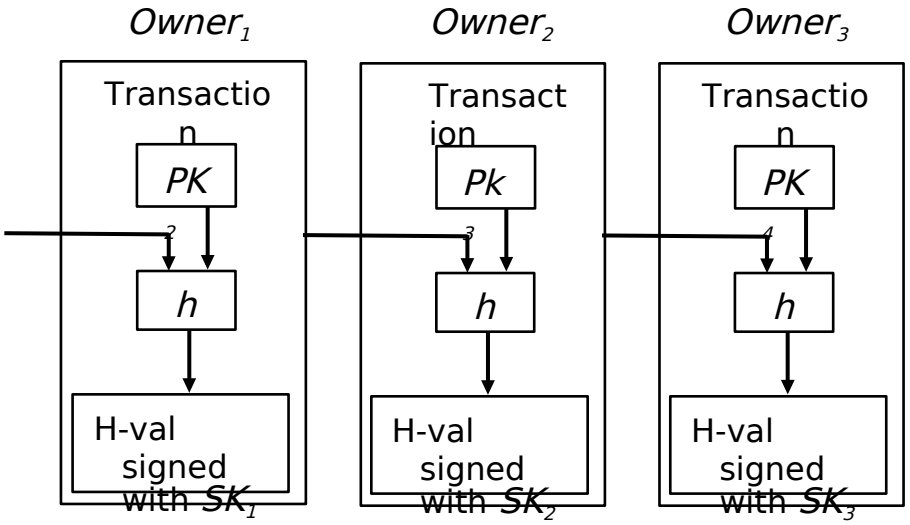


1. Initialization (of a *full node*)
   - Synchronization with the network to obtain the blockchain (185 GB on Q3, 2018)
2. Two users agree on a transaction
   - Information exchange: wallet addresses, public keys, …
3. Grouping with other transactions in a block and validation of the block (and of the transactions)
   - Consensus using "mining"
4. Addition of the validated block in the blockchain and replication in the P2P network
5. Transaction confirmation

Transaction

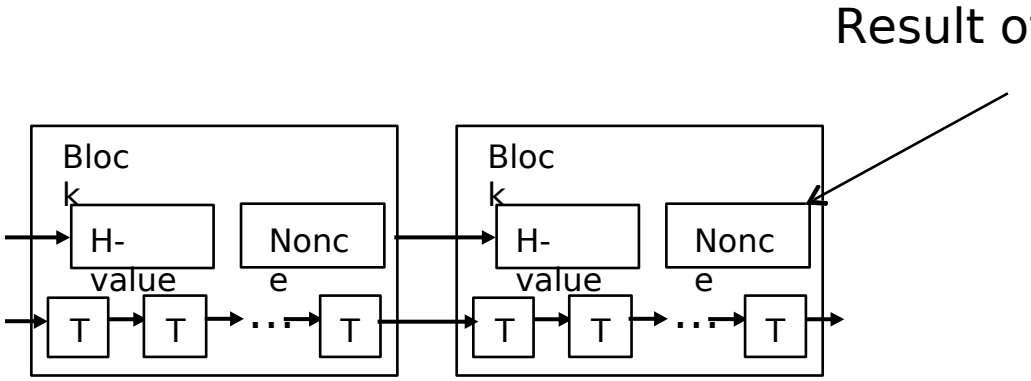


- The coin owner signs the transaction by
  1. Creating a hash value of
     - The previous transaction
     - And the public key (PK) of the next owner
  2. Signing it with its secret key (SK)



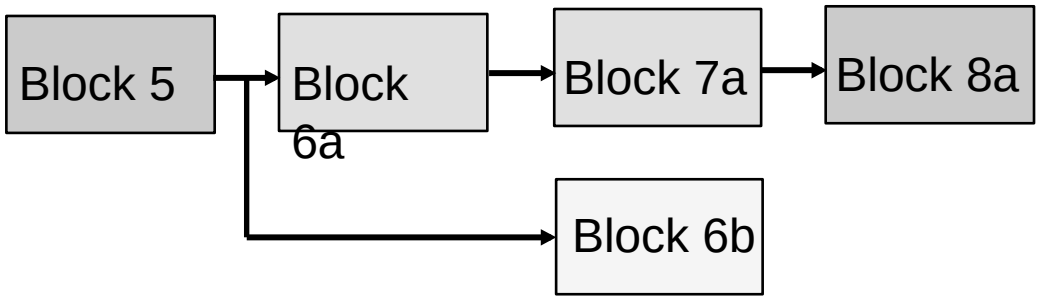
Block Management

- Transactions are placed into blocks, validated (by checking inputs/outputs, etc.) and linked by their addresses
  - Size of a bitcoin block = 1 Megabyte



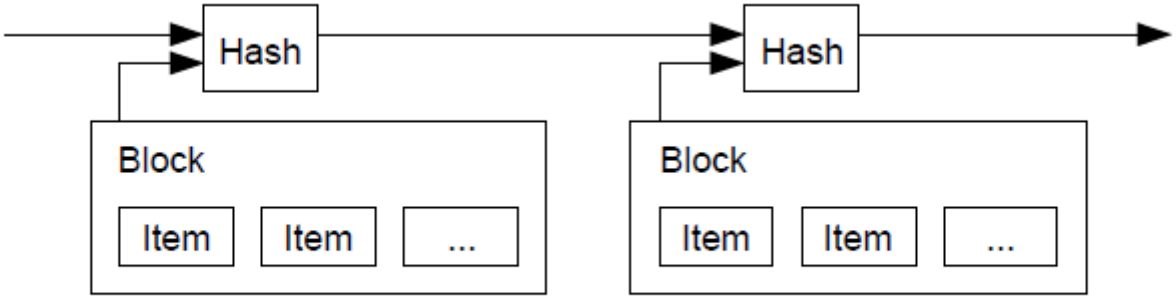
Validation by the Network

- Each block is validated by network nodes, the *miners*, by a consensus protocol (see next)
- Problem: accidental fork
  - As different blocks are validated in parallel, one node can see several candidate chains at any time
  - Solution: longest chain rule



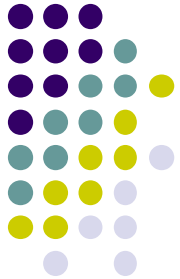
Transactions in a validated block are provisionally validated; confirmation

Block Management and consensus



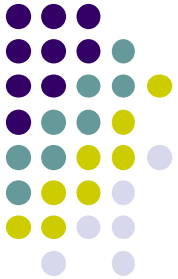
1) New transactions are broadcast to all nodes.
2) Each node collects new transactions into a block.
3) Each node works on finding a difficult proof-of-work for its block.
4) When a node finds a proof-of-work, it broadcasts the block to all nodes.
5) Nodes accept the block only if all transactions in it are valid and not already spent.
6) Nodes express their acceptance of the block by working on creating the next block in the chain, using the hash of the accepted block as the previous hash.

Intentional Fork



- Main reasons
  - To add new features to the blockchain (protocol changes) => new software
  - To reverse the effects of hacking or catastrophic bugs
- Soft versus hard fork
  - Soft fork: backward compatible
    - The old software recognizes blocks created with new rules as valid
    - Makes it easy for attackers
  - Hard fork
    - The old software recognizes blocks created with new rules as invalid
    - Example: the battle between (new) Ethereum and Ethereum Classic
      - In 2016, after an attack against the Decentralized Autonomous Organization (DAO), a complex smart contract for venture capital, the blockchain forked but without momentum
      - Battle is more philosophical and ethical than technical

Consensus Protocol: *mining*



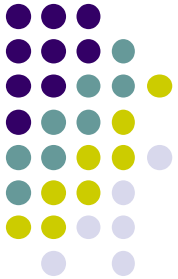
- Why not Paxos?
  - Remember: participants are unknown
- To validate a block, miner nodes compete (as in a lottery) to produce a *nonce* (number used once)
  - One of the first competing solutions is selected, e.g. the one that includes the largest number of transactions
  - The winner miner is paid, e.g. 12.5 bitcoins today (originally 50)
  - This increases the money supply
- Mining is designed to be difficult
  - The more mining power the network has, the harder it is to compute the nonce
  - This allows controlling the injection of new blocks ("inflation") in the system, on avg. 1 block every 10mn
  - Advantages powerful nodes

Mining Difficulty: Proof of Work (PoW)

- PoW
  - A piece of data that is difficult to calculate but easy to verify
  - First proposed to prevent DoS attacks

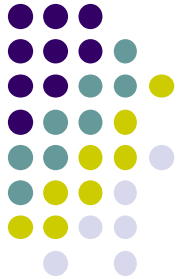
- Hashcash PoW
  - Computed by each miner to produce the nonce
- Goal: produce a value v such that $h(v)<T$ where
  - h is a hash function (SHA-256)
  - T is a target value which is shared by all nodes and reflects the size of the network
  - v is a 256-bit number starting with n zero bits
    - Low probability of success : $1/2^n$

The 51% Attack



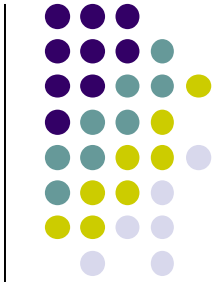
- Also called Goldfinger attack
  - Enables the attacker to invalidate valid transactions and double spend funds
- How
  - By holding more than 50% of the total computing power for mining
    - Miners coalition
  - It then becomes possible to modify a received chain (e.g. by removing a transaction) and produce a longer chain that will be selected by the majority
- Solution: monitoring by the community
  - In January 2014, Ghash.io reached 42%, then dropped to 9% after the Bitcoin community alert

Transaction Confirmation



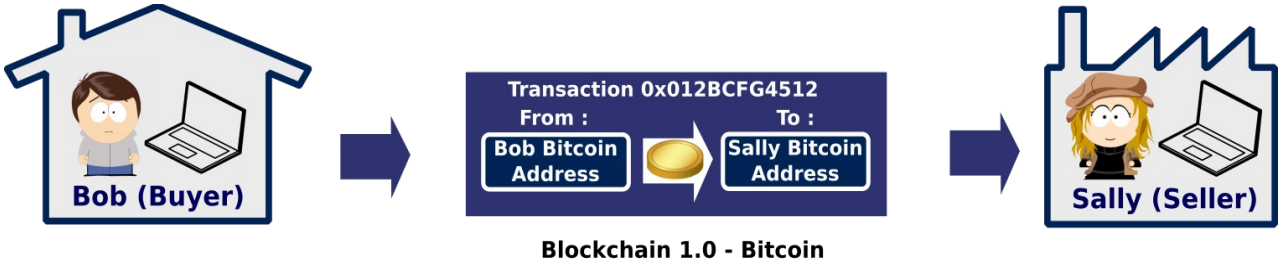
- A provisionally validated transaction in a candidate block ensures that it has been verified and is viable
- Each new block accepted in the chain after the validation of the transaction is considered as a confirmation
  - A transaction is considered mature after 6 confirmations (1 hour on average)
  - New bitcoins (mining products) are only valid after 120 confirmations, to avoid the 51% attack

Public Blockchain Limitations



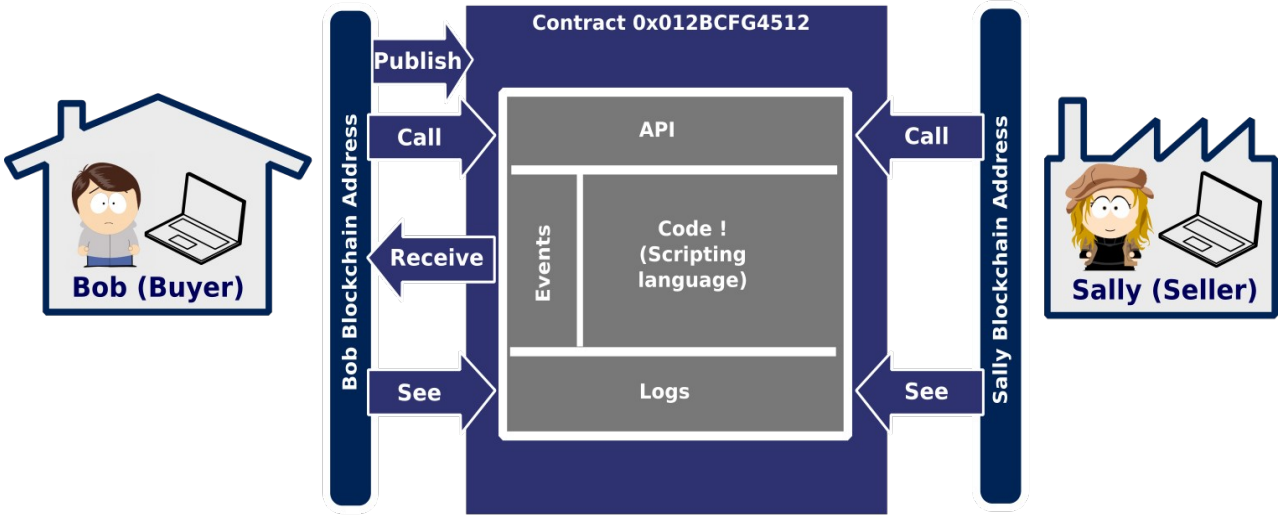
- Complexity and low scalability
  - Difficult evolution of operating rules
  - Increasing chain size
  - Low number of transactions per second (TPS)
    - 5-7 TPS for Bitcoin versus 25K TPS for VISA
  - Unpredictable duration of transactions, from minutes to days
- Cost
  - High energy consumption
  - Favors concentration of miners
- Users are pseudonymized, not anonymized
  - Making a transaction with a user reveals all its other transactions
- Lack of control and regulation
  - Hard for states to watch and tax transactions

Evolution of Paradigm



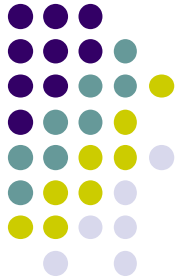
Blockchain 1.0 - Bitcoin

Evolution of Paradigm



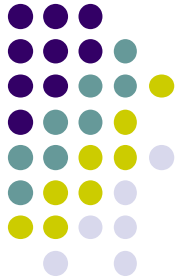
Blockchain 2.0 - e.g. Ethereum, ...

Evolution of Paradigm



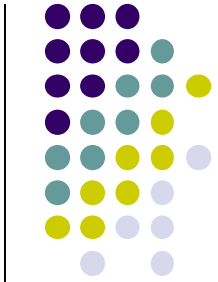
- Beyond Bitcoin and other cryptocurrencies
  - Recording and exchange of assets without powerful intermediaries
  - Example: smart contracts
- Positioning in the internet
  - TCP/IP: the communication protocol
  - Blockchain: the value-exchange protocol?

Blockchain 2.0/3.0 Technology



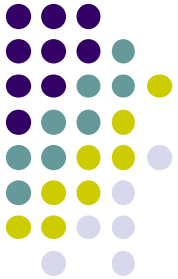
- Programmable blockchain, e.g. Etherum
  - Allows application developpers to build APIs on the Blockchain protocol
    - APIs to allocate digital resources (bandwidth, storage, etc.) to the connected devices, e.g. FileCoin
    - Micropayment APIs tailored to the type of transaction (e.g. tipping a blog versus tipping a car share driver)
- Private blockchain
  - Efficient transaction validation since participants are trusted
    - No need to produce a PoW
  - Efficient management, e.g. in the cloud

Blockchain 2.0/3.0 Development



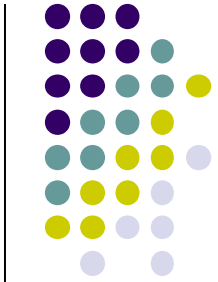
- Support from all major industry players
  - Finance services: Mastercard, VISA, …
  - Audit firms: EY, KPMG, PwC, Deloitte
  - Consulting firms: Accenture, Capgemini,
  - Web giants: Amazon, Google
  - Software suppliers: IBM, Oracle, Microsoft, SAP
  - Technology platform companies: Cisco, Fujitsu, IBM, Intel, NEC, Red Hat, VMware
- New blockchain ISVs
  - Blockchain, ConsenSys, Digital Asset, R3, Onchain

Smart Contracts



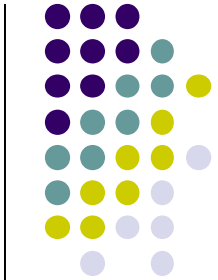
- "Code is law", Lawrence Lessig, Harvard Law School
- Smart contract (Nick Szabo, 1993)
  - Self-executing contract, with code that embeds the terms and conditions of a contract
  - Early application: digital rights management schemes
- Deployment in the blockchain 2.0 (e.g. Etherum)
  - Participants can be unknown to each other
  - Contracts can be with many third parties, e.g. IoT devices, at low cost
- Challenges
  - Bug-free code, which requires code certification
  - Compliance with mandatory regulation, which requires collaboration between programmers and lawyers

Hyperledger Project (Linux Foundation)



- Started in 2015 (IBM, Intel, Cisco, …)
- Open source blockchains and related tools
- Major frameworks
  - Hyperledger Fabric (IBM, digital Asset): a permissioned blockchain infrastructure
    - Smart contracts, configurable consensus (PBFT, …) and membership services
  - Sawtooth (Intel): a new consensus "Proof of Elapsed Time" that builds on trusted execution environments
  - Hyperledger Iroha (Soramitsu): based on Hyperledger Fabric, with a focus on mobile applications

Blockchain 2.0/3.0 Apps



- Critical characteristics of the applications
  - Asset and value are exchanged (transactions)
  - Multiple participants, unknown to each other
  - Trust is critical
- Top use cases
  - Financial services, micropayments
  - Digital rights using smart contracts
  - Digital identity
  - Supply chain management
  - Internet of Things (IoT)
- POCs in many industries
  - Publishing, retail, music, healthcare, rental, real estate, government, energy, agriculture, etc.