Ingineria Calculatoarelor

Toleranța la defecte pentru sisteme software

Toleranță la defecte

Defecte

- Software
- Hardware
- Design
- Operaționale

Latente → Active

↓

Eroare

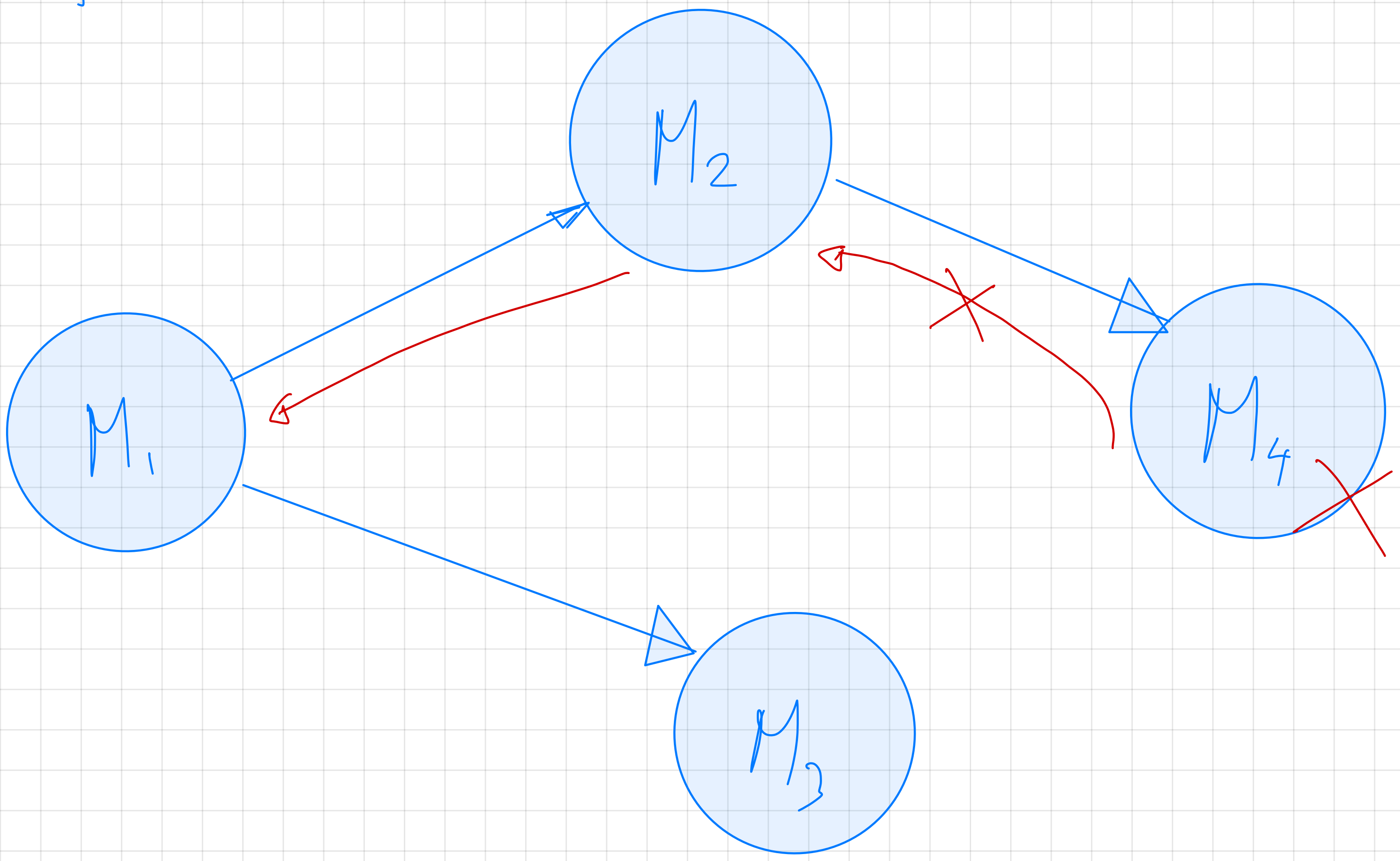
↓

Defecțiune

1. Componente nefiabile

2. ??

3. Sistem fiabil



Example

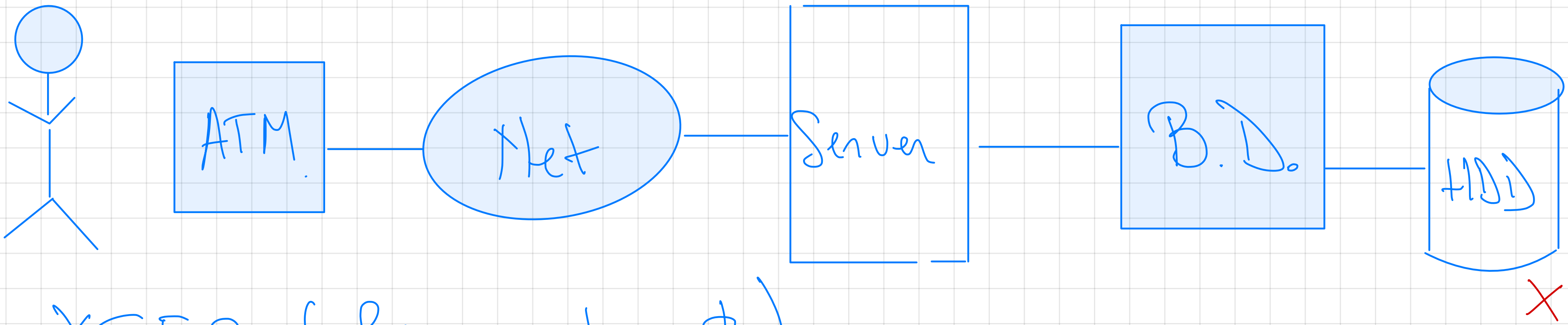
- Pool locking
- Routing
- Packet loss
- Congestion collapse
- DNS

Abordare sistematică

1. Modularizare
2. Detectare erori
3. Mascare erori

] → Conform cu specificațiile

Redundanță



XFER (from, to, $)

- Fail-stop
- Fail-fast
- Fail-soft
- Fail-safe

Modelare

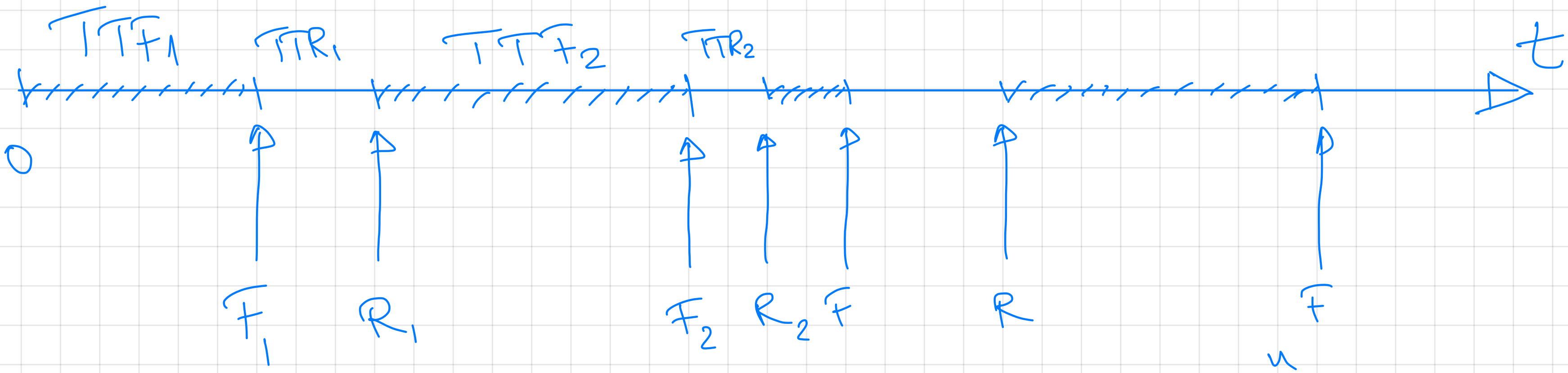
1. Nr. erori tolerate

2. Media timpului de bună funcționare MTBF

Disponibilitate

$$A = \frac{\sum TTF_i}{\sum TTF_i + \sum TTR_i}$$

$$A = \frac{MTBF}{MTBF + MTR}$$

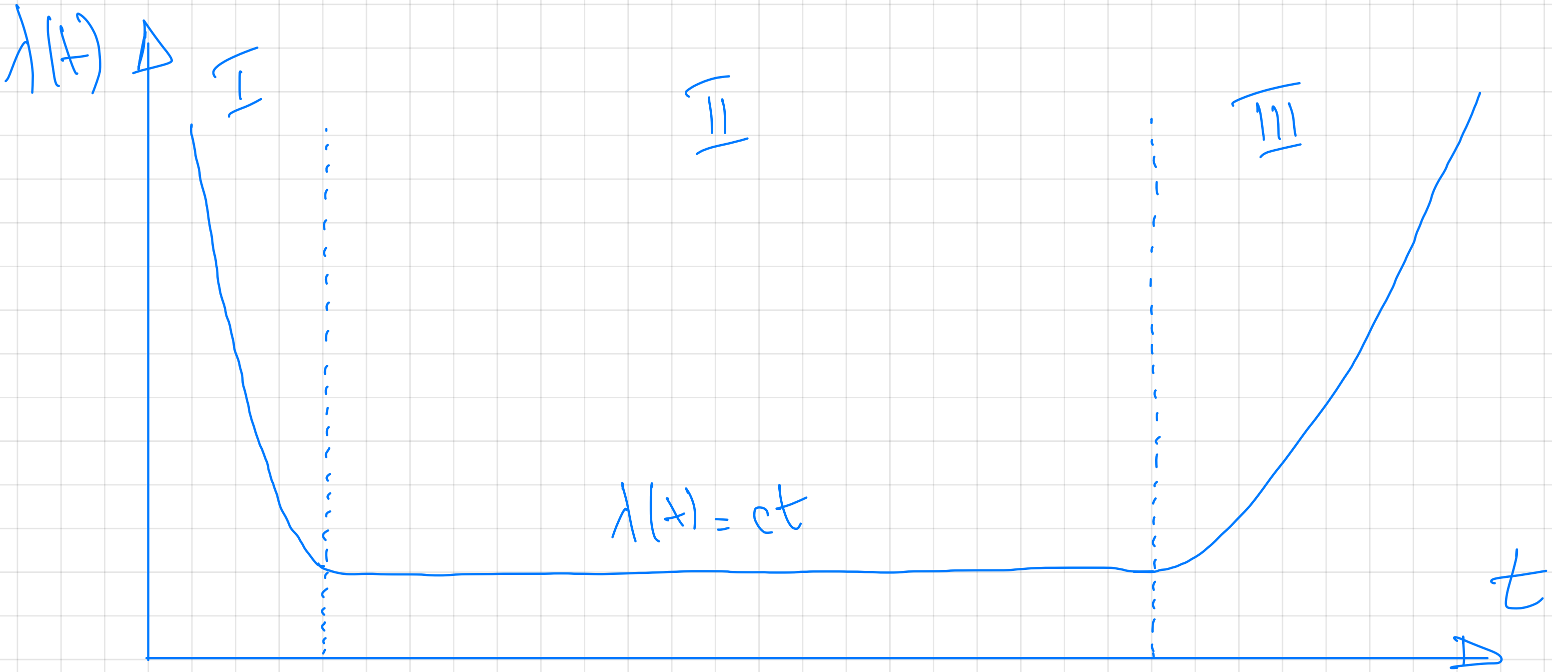


$$MTBF = \frac{\sum_{i=1}^{n} TTF_i}{n}$$

$$MTR = \frac{\sum_{i=1}^{n} TTR_i}{n}$$

intensitatea defectărilor

$$\lambda(t) = P(\text{fail in } [t, t+\Delta t] \mid OK@t)$$



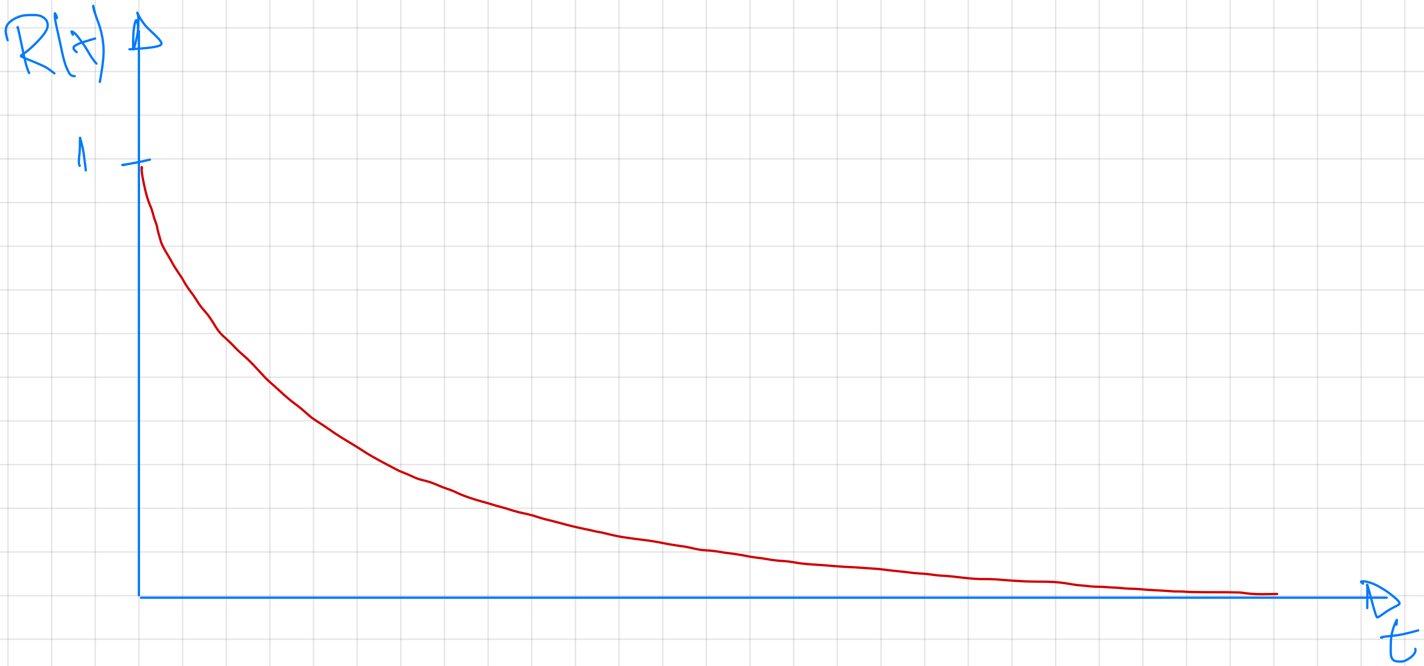
Bath-tub curve

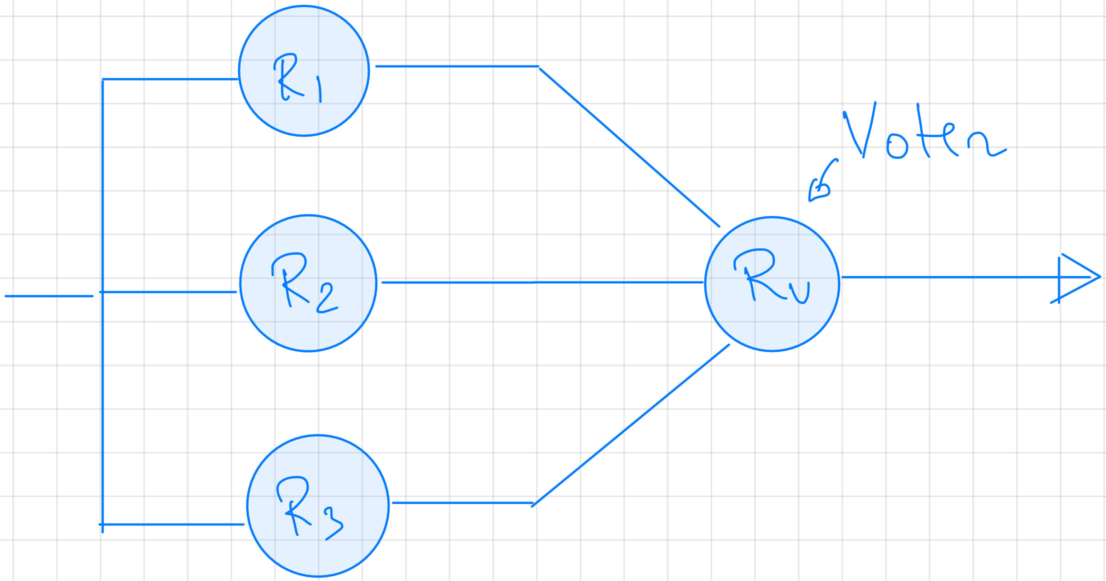
$$\lambda = \frac{1}{MTBF}$$

Fiabilità

$$R(t) = P(\text{OK @ time } t)$$

$$R(t) = e^{-\lambda t}$$





$R = 10\% = 0{,}1$

$R_T = 3(0{,}1)^2 - 2(0{,}1)^3 = 0{,}03 - 0{,}002 =$

$= 0{,}028 = 2{,}8\%$

$R = 90\% = 0{,}9$

$R_T = 3\cdot(0{,}9)^2 - 2\cdot(0{,}9)^3 = 3\cdot 0{,}81 - 2\cdot 0{,}729$

$= 0{,}972 = 97{,}2\%$

$$R_{Total} = R_V \cdot \left(R_1 \cdot R_2 \cdot R_3 + (1-R_1) R_2 R_3 + R_1 (1-R_2) R_3 + R_1 R_2 (1-R_3) \right)$$

$R_1 = R_2 = R_3 = R$

$R_V >> R \qquad R_V \simeq 1$

$$R_{Total} = R^3 + 3(1-R)R^2 = R^3 + 3R^2 - 3R^3 = 3R^2 - 2R^3$$

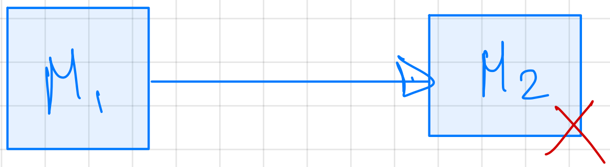
$R_{Total} > R$? $\qquad 3R^2 - 2R^3 > R \Rightarrow 2R^3 - 3R^2 + R < 0 \Rightarrow R(2R^2 - 3R + 1) < 0$

$R \in [0, 1] \quad \Rightarrow \quad 2R^2 - 3R + 1 < 0 \Rightarrow (R-1)\left(R - \frac{1}{2}\right) < 0 \quad \Rightarrow \quad R > \frac{1}{2}$

Defecțiuni

1. Replicare + Votare

2. Recuperabilitate



← Nu expune o stare parțială

XFER (from, to, $) → Se execută

→ Face crash recovery

RECUPERABILITATE → "totul sau nimic"

```
transfer(from_acct, to_acct, amount) {

    x ← read_disk(from_acct);
    x ← x - amount;
    write_disk(from_acct, x);

    y ← read_disk(to_acct);
    y ← y + amount;
    write_disk(to_acct, y);
}
```

← COMMIT POINT

Concurență

$S = 1000 \qquad C = 0$

$xfer(S, C, 100) : A_1$

$xfer(S, C, 200) : A_2$

$S = 700 \qquad C = 300$

A_1 înainte de A_2

SAU

A_2 înainte de A_1

IZOLABILITATE

Atomicitate

- Asumăm faptul că acțiunile sunt compuse
- Recuperabilitate + Izolabilitate

- Consistență → data invariant
- Durabilitate

TRANZACȚII ← REC + IZO + CONS + DURA

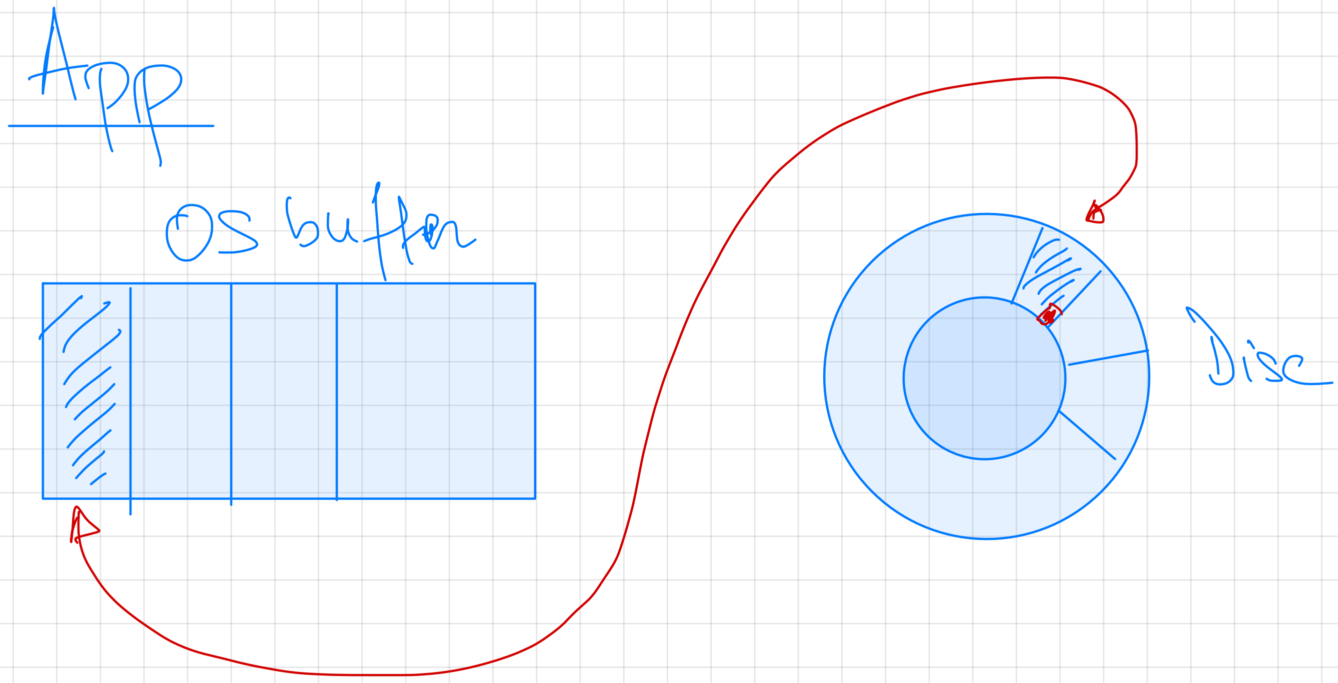
Recuperabilidade

1. Fail-fast
2. Crash recovery
3. Restart

- Recoverable sector
  - Version history
  - Logging

Modele

- Forcë concurrente
- Forcë errori hardware
- Errori software

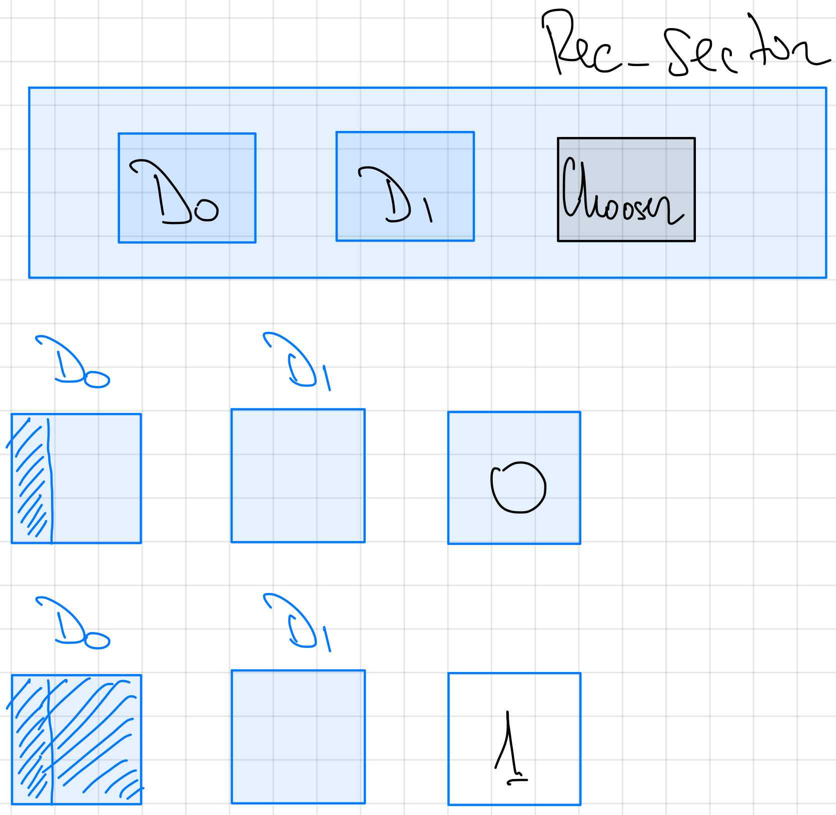


Model

confue_put (sec, data)
confue_get (sec)

] – elimină toate erorile HW

recoverable_put (sec, data)
recoverable_get (sec)



Soluție: facem copii

chosen = 0 → scriem în D_0
citim din D_1

chosen = 1 → citim din D_0
scriem în D_1

```
recoverable_put(data, rec_sector) {
  status <- careful_get(which, rec_sector.chosen)
  if (status == NOT_OK) which = 0; // arbitrary!
  if (which == 0)
      careful_put(data, rec_sector.D0);
                                        <- COMMIT POINT
  else
      careful_put(data, rec_sector.D1);
                                        <- COMMIT POINT
  which = which XOR 1;
  careful_put(which, rec_sector.chosen);
}
```

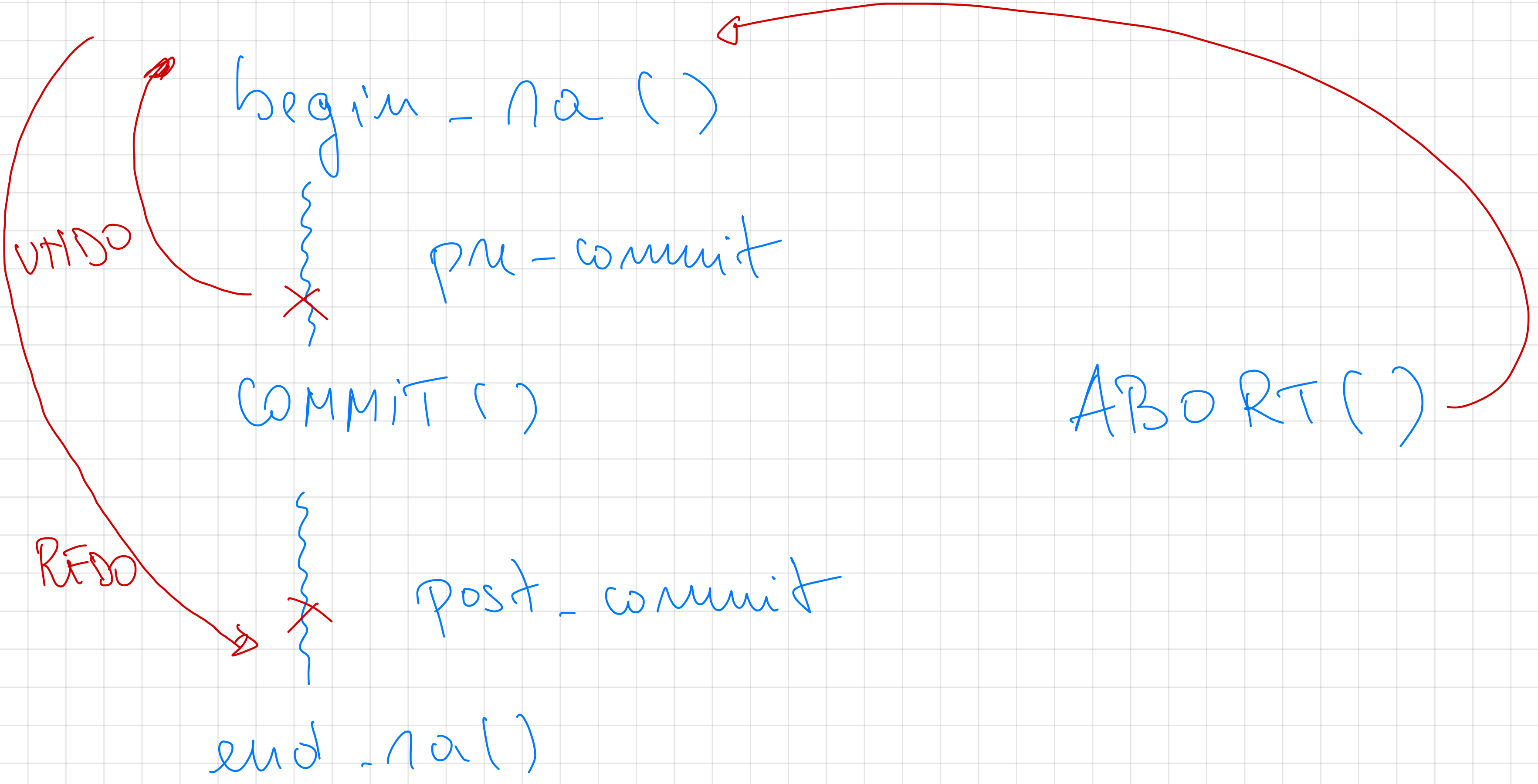
```
recoverable_get(data, rec_sector) {
    status = careful_get(which, rec_sector.chooser);
    if (status == NOT_OK) which = 1; // arbitrary!
    if (which == 0)
        ds = careful_get(data, rec_sector.D1);
    else
        ds = careful_get(data, rec_sector.D0);

    return ds;
}
```

Commit point

Before ← schimbările nu sunt vizibile

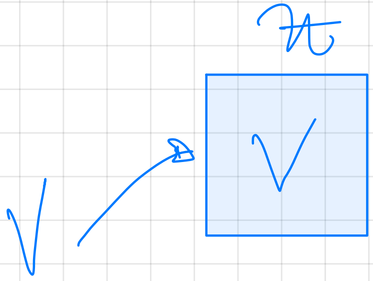
After ← toate modificările sunt vizibile

Generalizzazione



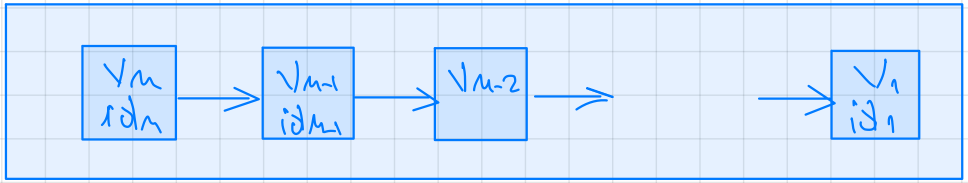
Version history

Cell Storage



Journal Storage

x :



WriteJournal (item x , v , id)

ReadJournal (item x , id)

Commit record table

id1	~~P~~	C
id2	P	
⋮		
id7	~~P~~	A
⋮		

LENT !

Logging

Cell storage : read / write

Log : stocare **nevolatilă** și **secvențială**

↓ CRASH free.

↓ RAPID !

Plan

1. Crash → recovery din log
   - → uncommitted → UNDO
   - → committed → REDO (install in cell st.)

2. ABORT( ) → UNDO implementation (REDO)

Append-only

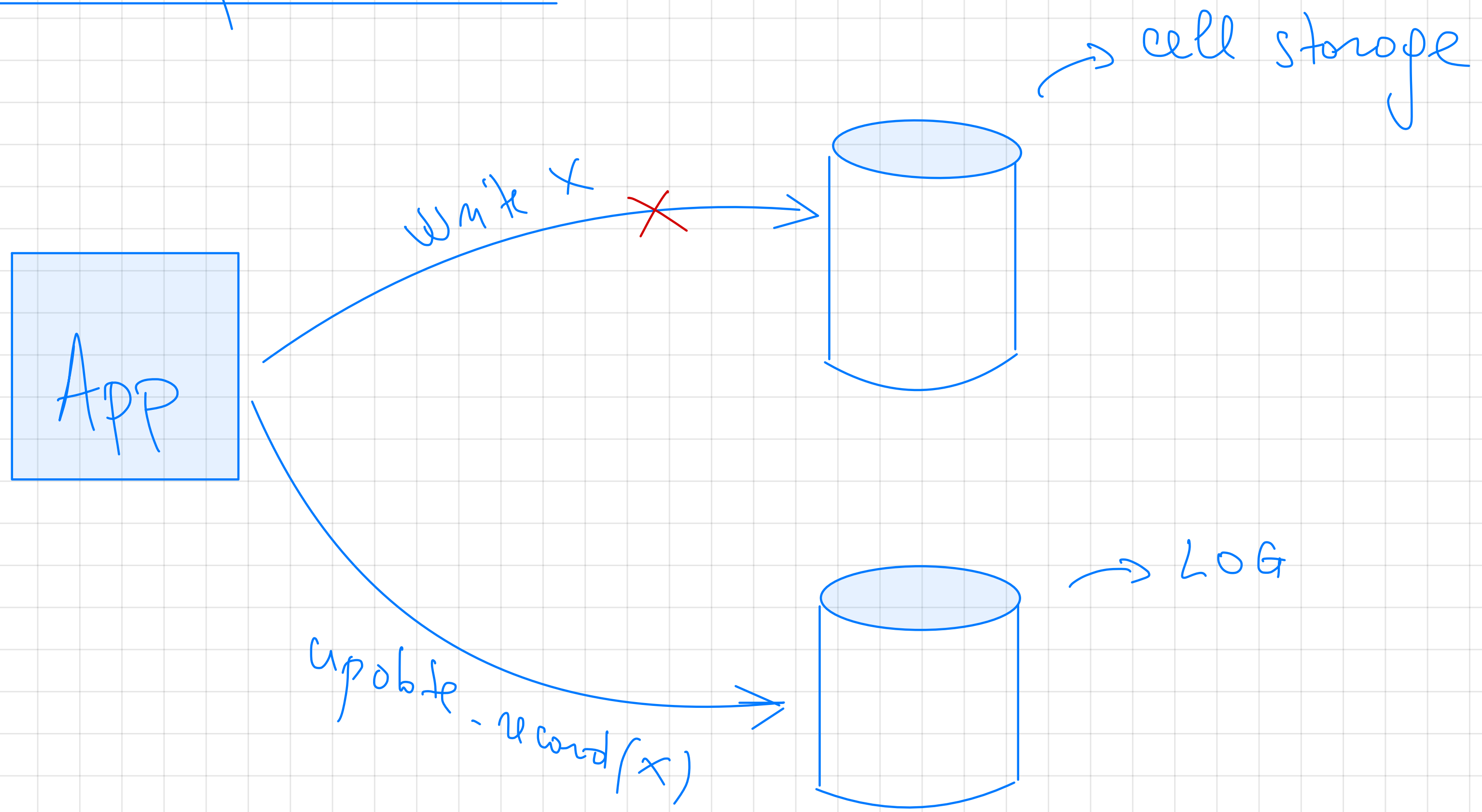
type : UPDATE
id ← 123
undo : x ← old
redo : x ← new

type : OUTCOME
id ← 422
status : COMMITED

1. Cand scriem in log?

2. Cum facem crash recovery?

BD stocată pe disc



1. Write-ahead logging (WAL)

Crash recovery

1. Scan a log in seq inverse

2. Winners : COMMITTED & ABORTED

   Losers : anything else

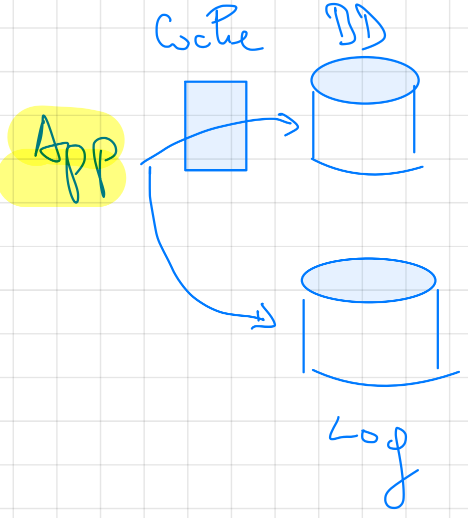
3. REDO COMMITTED winners

   UNDO Losers

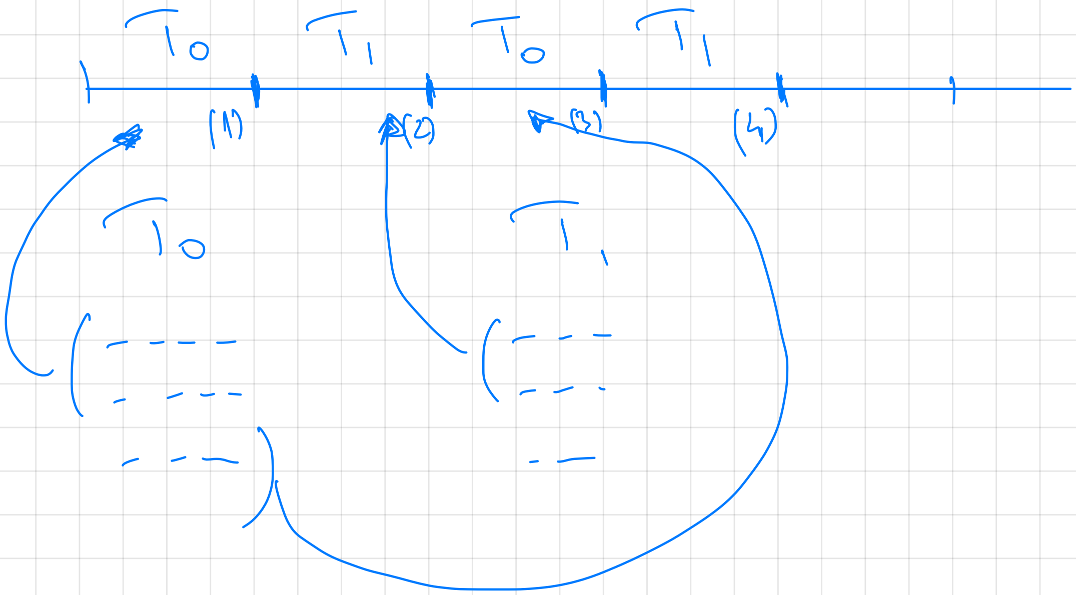
Atomicità

Recuperabilità **Isolabilità**

→ Rec. Sector → Version history
→ Logging (WAL) → ABORT / CRASH →



Isolabilità



$r(z) \to w(z)$
$w(z) \to r(z)$
$w(z) \to w(z)$

T_0	T_1
1) read x	2) write x
4) write y	3) write y

$r_0(x) \quad w_1(x) \quad w_0(y) \quad w_1(y)$

$\equiv$

$T_0 \to T_1$

$r_0(x) \quad w_1(x) \quad w_0(y) \quad w_1(y)$

Action graph

T_1	T_2	T_3	T_4
① $r_1(x)$	② $w_2(x)$	③ $r_3(y)$	④ $r_4(x)$
⑤ $w_1(y)$	⑥ $w_2(y)$	⑦ $w_3(z)$	

DAG

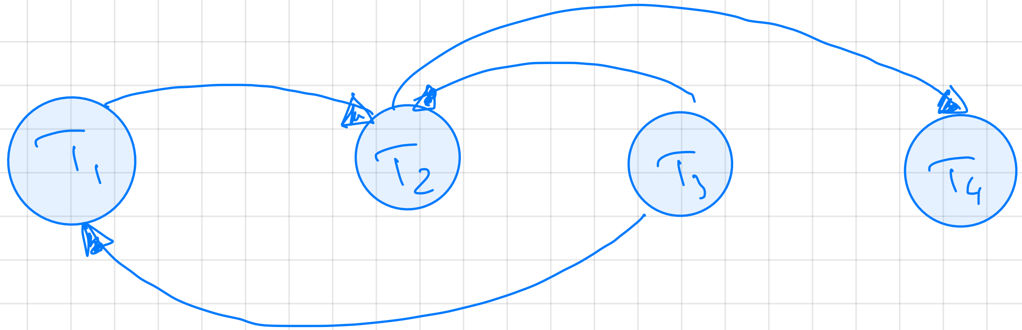
Conflicts?

T_1 T_2

T_1 T_3

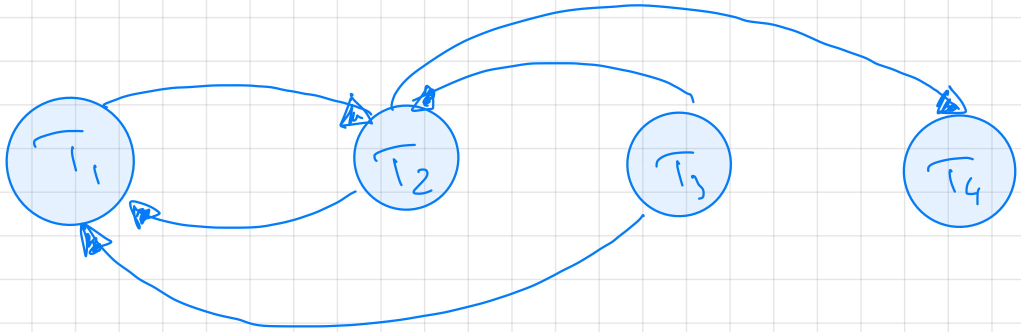
T_2 T_3

T_2 T_4



$T_3 \rightarrow T_1 \rightarrow T_2 \rightarrow T_4$

T_1	T_2	T_3	T_4
① $r_1(x)$	② $w_2(x)$	③ $r_3(y)$	④ $r_4(x)$
⑥ $w_1(y)$	⑤ $w_2(y)$	⑦ $w_3(z)$	



Dacă Action graph-ul este aciclic =>

=> trace-ul este serializabil => acțiunile sunt izolabile

Sortare topologică

Locks

acq(x)

rel(x)

T_0		T_1	
	acq(x)		acq(x)
①	r_0(x)	②	w_1(x)
	rel(x)		
	acq(y)		acq(y)
③	w_0(y)		w_1(y)
	rel(y)	④	rel(x)
			rel(y)

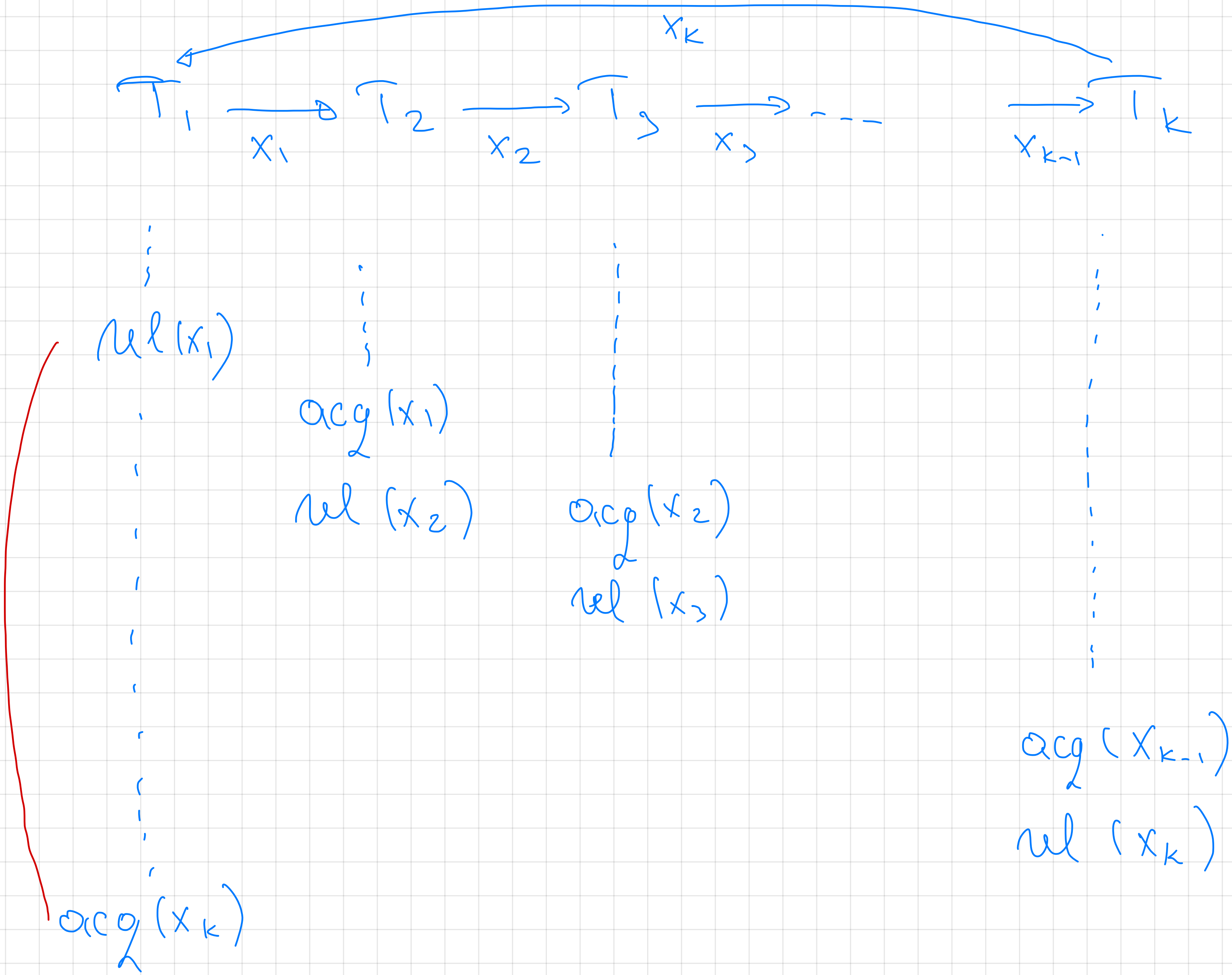
$i > 0$ OK

Rec NOK

Two-phase locking

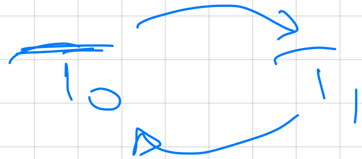
No release before ALL acquires!

acq(x)
r(x)
acq(y)
r(y)
⋮ ← COMMIT POINT
rel(y)
rel(x)



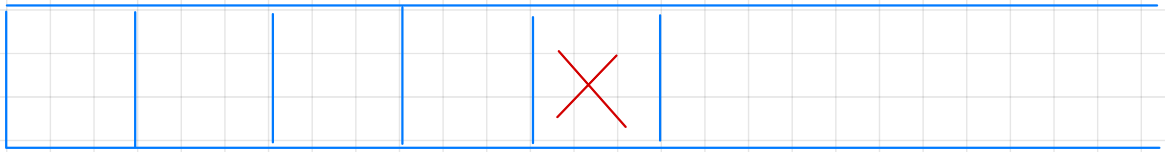
x_k
T_1
x_1
T_2
x_2
T_3
x_3
x_{k-1}
T_k
rel (x_1)
acq (x_1)
rel (x_2)
acq (x_2)
rel (x_3)
acq (x_{k-1})
rel (x_k)
acq (x_k)

T_0	T_1
acq(x)	acq(y)
mod(x)	mod(y)
acq(y)	acq(x)
write(y)	write(x)
rel(y)	rel(x)
rel(x)	rel(y)



1. Timers
2. Waits-for graph

Logs & Locks



Aplicații

1. Tranzacții — Consistency
   — Durabilitate

2. Sisteme distribuite

Tranzacții A.C.I.D.

Consistența datelor

Centralizată → reguli de integritate a datelor

SID	Nume	ID Fac.
35	Herbert	5

F.K. → P.K.

ID Fac.	Nume Fac.
5	A&C
6	ETTI

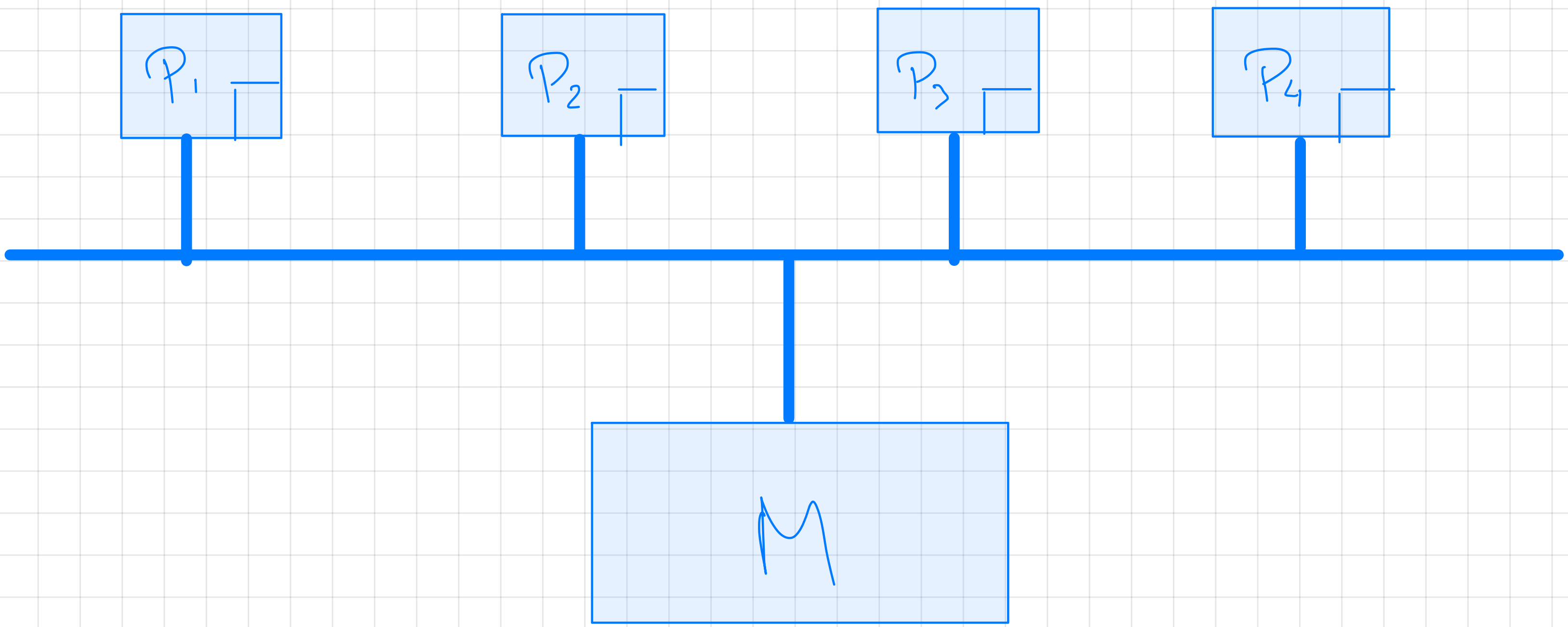
Distribuit

DNS – Expiration time
Web caches – "If modified since"

} **Weak Cons.** / **Eventual Cons.**

Strong Consistency

O citire întoarce întotdeauna rezultatul ultimei scrieri



1. Write-through cache
2. Snoopy cache

Multi-site Atomicity

Site Rezervări → Buc. - Ist. (TAROM)

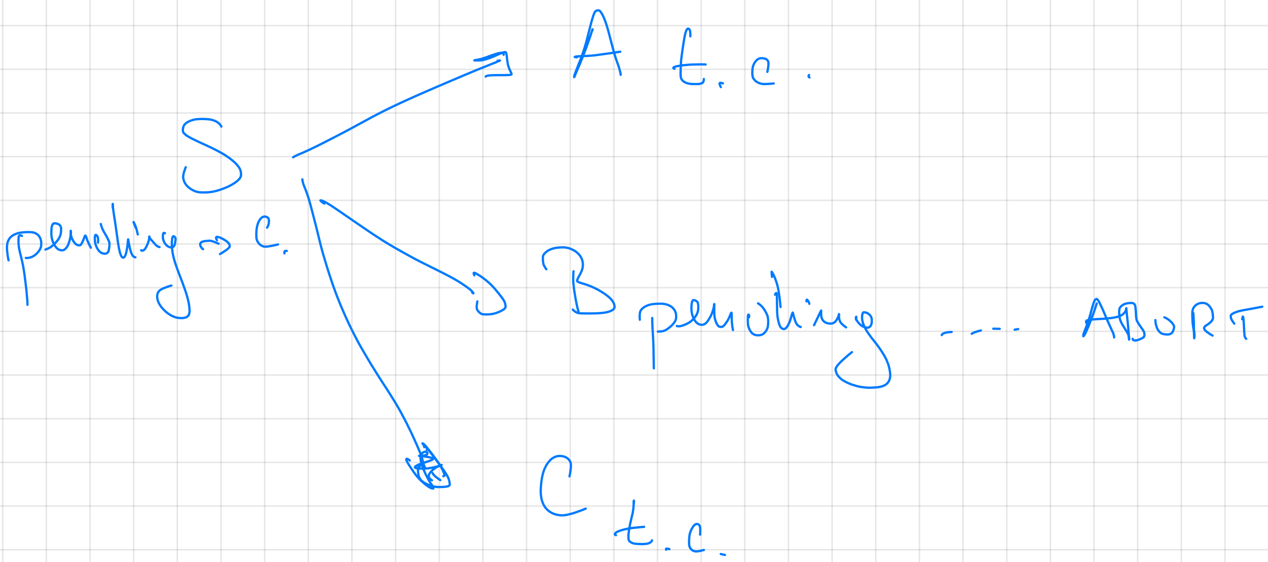
Site Rezervări → Ist. - Tuvolu (Turkish)

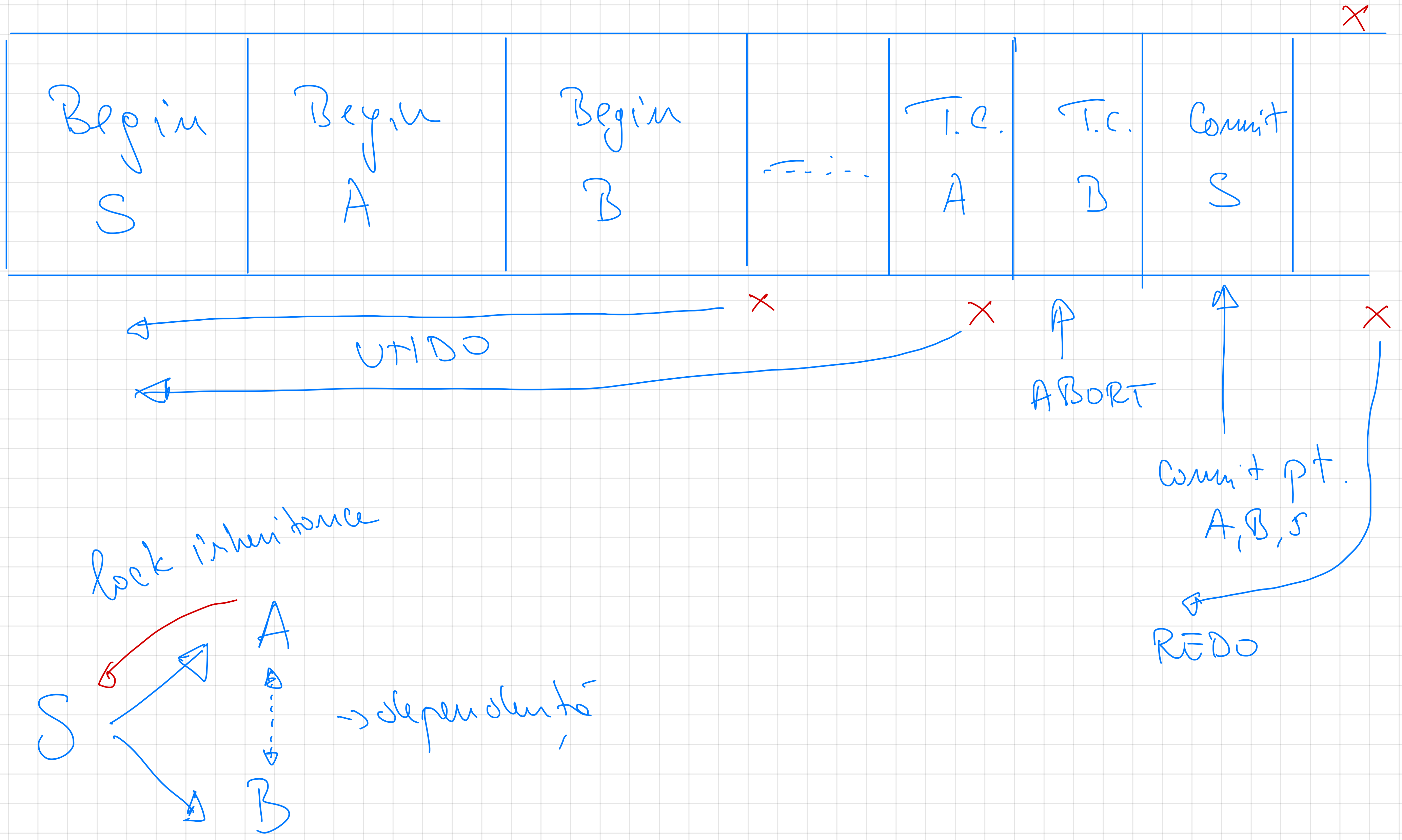
Nested Atomic Action

S [
Begin
    Buy TAROM | A
    Buy Turkish | B
End

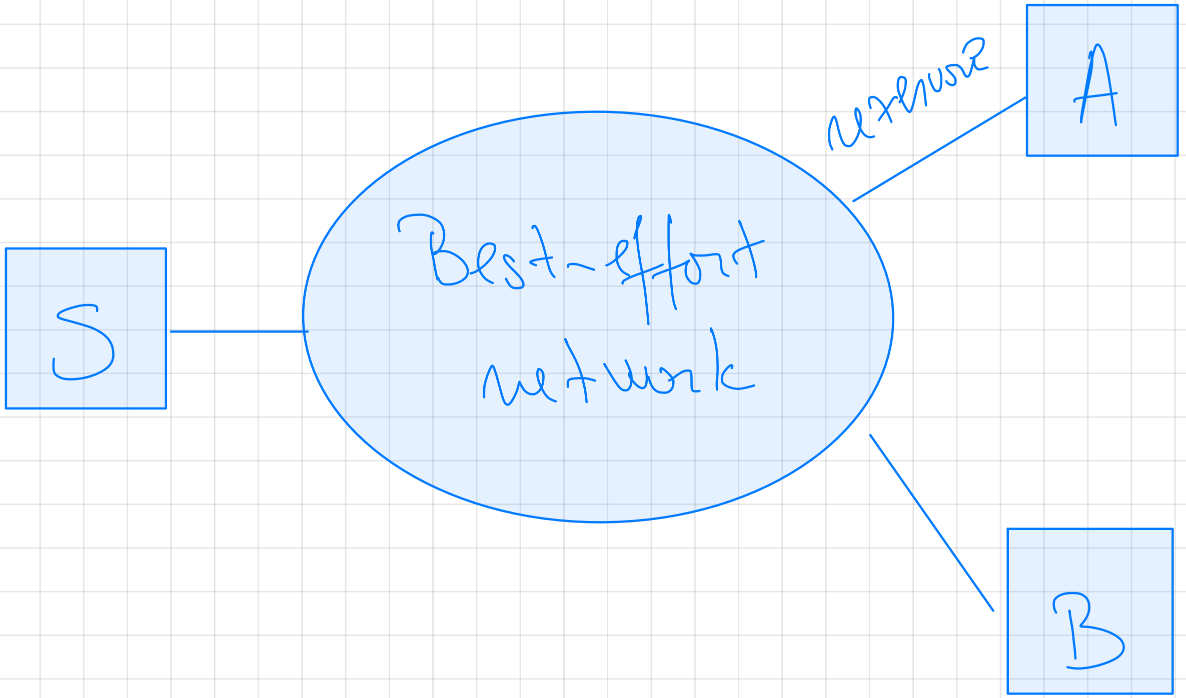
S → (A commits iff B commits)

Tentative commit



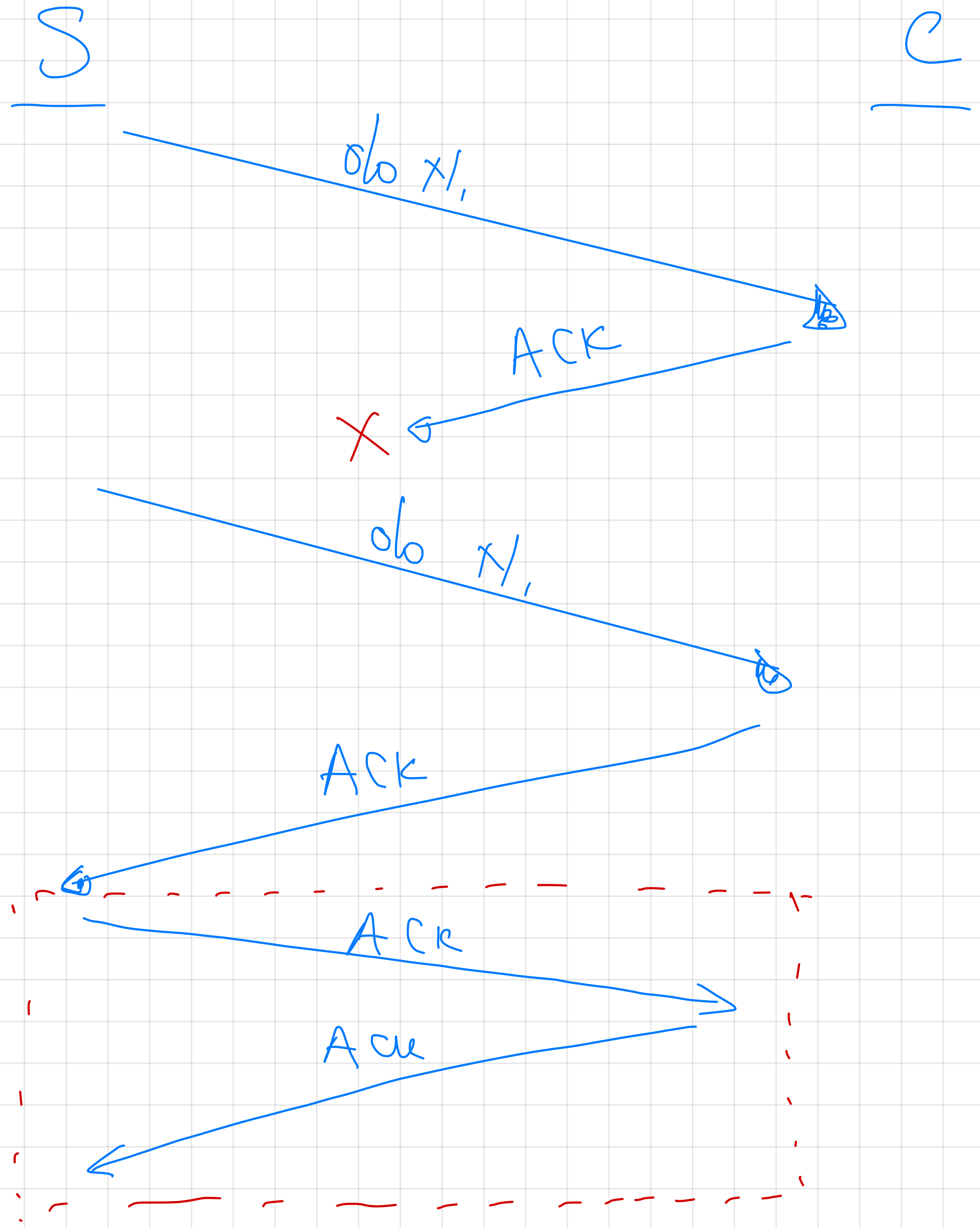

Begin S
Begin A
Begin B
T.C. A
T.C. B
Commit S
UNDO
ABORT
Commit pt. A, B, S
REDO
lock inheritance
S
A
B
→ dependenti

Calcul distribuit

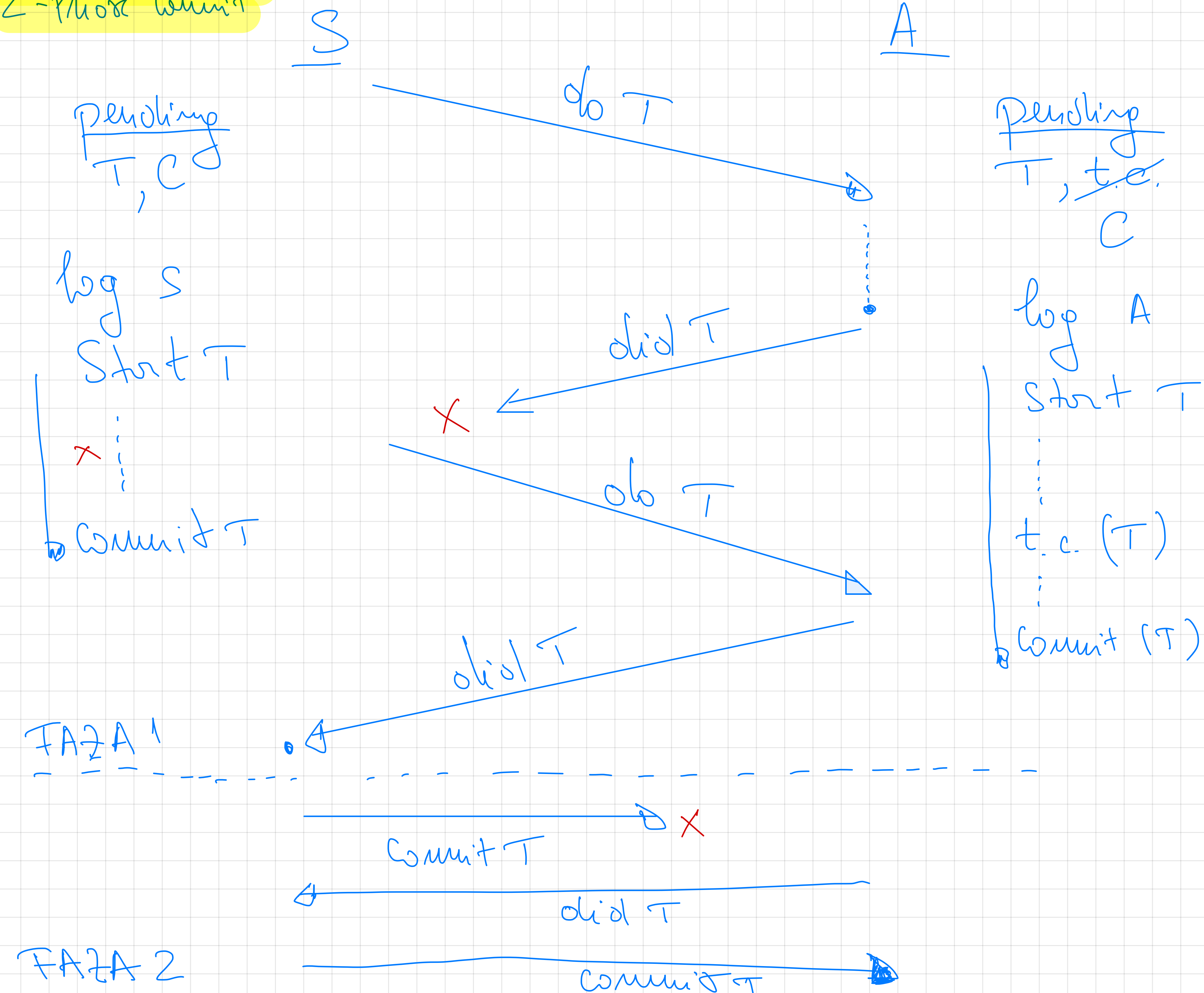


R.P.C.

Exactly-once RPC



2-Phase Commit



S crash

înainte de commit

UNDO T, (notifică A?)

după commit

REDO T, (notifică A?)

A crash

înainte de T.C.

UNDO A, (S face REDO pt. A)

după T.C.

? → depinde de S

Two-phase Commit