SYNCHRONISERS: ATTEMPT TO SYNCHRONISE ASYNC NETWORKS (SEE LYNCH CH 16)

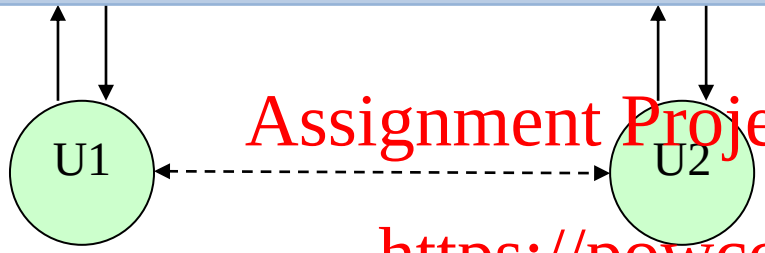
DO NOT COPE WITH STOPPING (OR, EVEN LESS, WITH BYZ) FAILURES

GlobSync (Global Synchronizer)

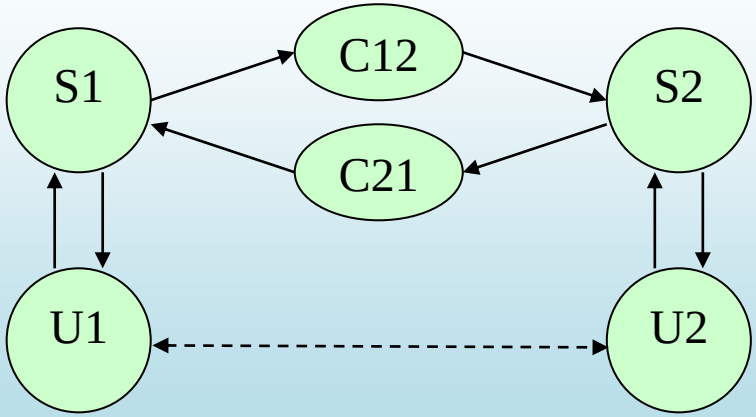
- ensures that *all* messages are sent before receiving
- collects *all* sent messages and then delivers them to their targets

LocSync (Local Synchronizer)

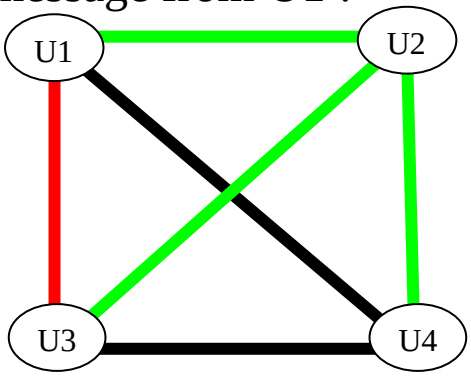
- similar to GlobSync
- but only ensures that *all neighborhood* messages are sent before receiving



SimpleSync (Simple Synchronizer)
an implementation of the LocSync



- Synchronizers need to keep the round number, explicitly or implicitly.
- Even if no faulty nodes are present, messages from different rounds may arrive interleaved (**fast**, **slow**).
- Here, process U3 may receive a round 2 message from U2, before receiving its round 1 message from U1 !



SYNCHRONISERS: CAN THEY HELP?

- Using a simpler sync algorithm instead of a much more complex async !
  - E.g. Sync Distributed MST instead of the Async Distributed MST (GHS)

- Obtaining a faster runtime ?

  - E.g. could this avoid greedy choices, which may be very costly in the end ?

  - What if we apply a synchroniser to Distributed Bellman-Ford ?

COMMUNICATION FAILURES

Is it possible to reach a distributed agreement with (unbounded) communication failures?

Formal conditions:

- **Termination**:
  - All processes eventually decide.
- **Agreement**:
  - No two processes ever decide on different values.
- **Validity**:
  - If all processes start with the same initial value 1 and all messages are properly delivered, then 1 is the only one possible decision value. [WEAK]
    - relaxation: allow fallback on 0 if communications fail (approx. v0 could be 0)

  and
  - If all processes start with the same initial value 0, then 0 is the only one possible decision value. [STRONG]

We also assume that a decision once taken can be immediately acted upon (by *all processes in sync*).

Diagrams

	Initial	Decision
No comm fault	∀0	∀0
Comm faults	∀0	∀0

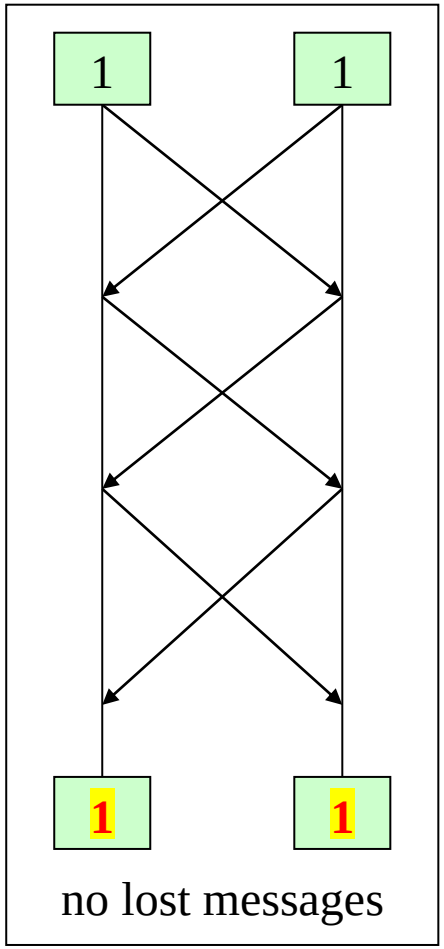
	Initial	Decision
No comm fault	∀1	∀1
Comm faults	∀1	∀1 *or* ∀0

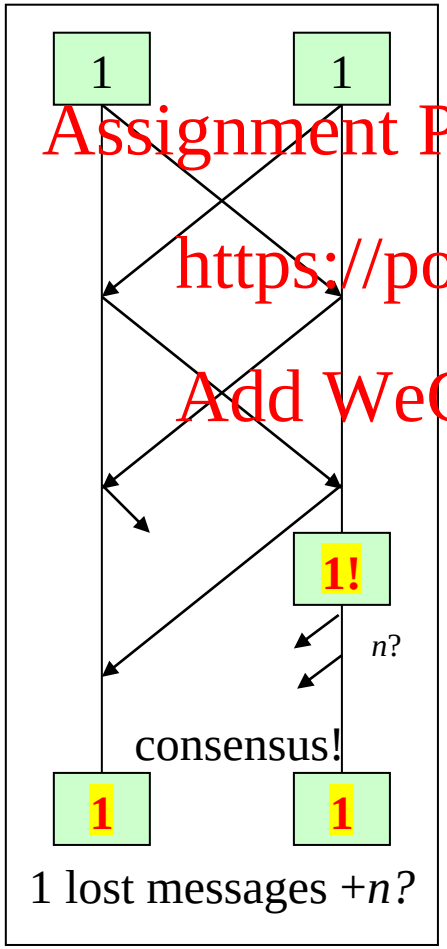
	Initial	Decision
No comm fault	∃0,∃1	∀1 *or* ∀0
Comm faults	∃0,∃1	∀1 *or* ∀0

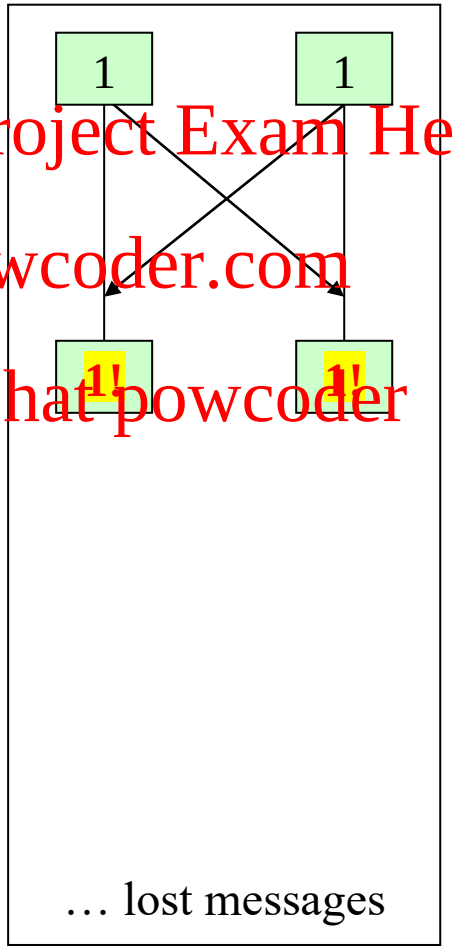
Fundamental result: Even under such relaxed conditions, no deterministic agreement is possible if unlimited communication failures are possible (but then, how do TCP and the internet work?)

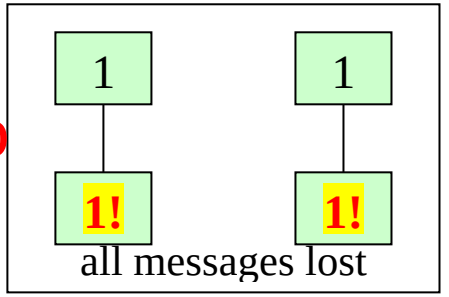
Proof: by induction, using the given validity, agreement and termination conditions

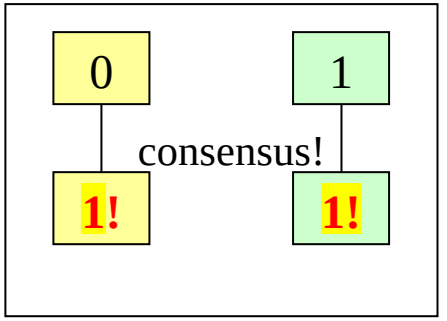
In our diagrams, ! denotes a no-return state, where a certain decision will inevitably follow (even if no more messages are received)

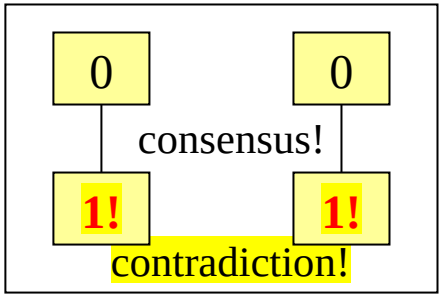












Distributed Commit

- 2 Phase Commit – Weak Termination ("blocking")
- 3 Phase Commit – Strong Termination ("non-blocking")

Termination: All non-faulty processes eventually decide [STRONG]. (3PC)

or

If there are no failures then all processes eventually decide [WEAK]. (2PC)

Agreement: No two processes (even faulty) ever decide on different values [STRONG]. (2&3PC)

Validity: If any process starts with the initial value 0, then 0 is the only one possible decision value.

initial 0 → can start as decided 0!

and If all processes start with the same initial value 1 and there are no failures, then 1 is the only one possible decision value.
[WEAK]. (2&3PC)

Diagrams

Process	Initial	Decision
Decided, not failing	∀1	∀1 *or* ∀0†
Decided, before failing		∀1 *or* ∀0†
Non-decided, failed		—
Non-decided, not failing (2PC♣)		—

Process	Initial	Decision
Decided, not failing	∃0	∀0
Decided, before failing		∀0
Non-decided, failed		—
Non-decided, not failing (2PC♣)		—

† Decision 0 is allowed only if one/more processes fail

♣ The 2PC weak termination condition allows non-faulty process (with initial value 1) that never decide, that remain "blocked" (unless they receive some "magical" help).

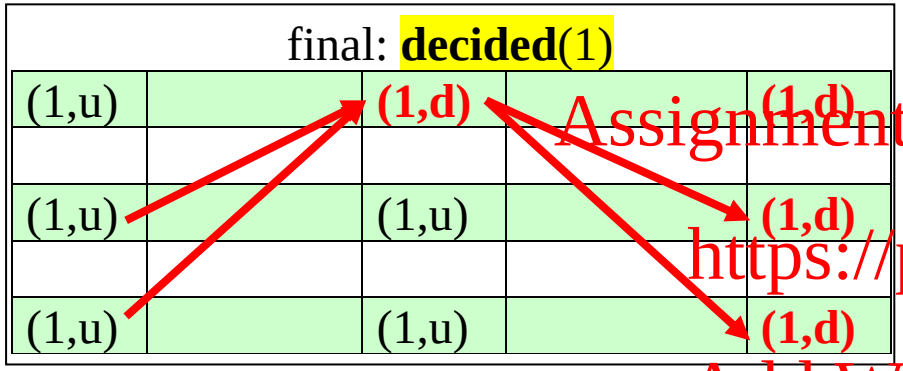
THE 2 PHASE COMMIT – WEAK TERMINATION ("BLOCKING")

We do not discuss here:

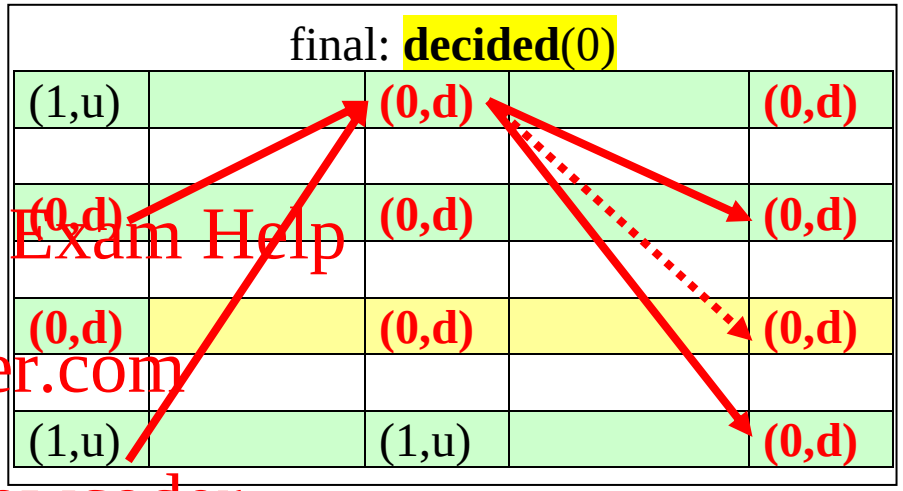
- how the processes start this,
- how do they recover after failures.

- Participants: coordinator (leader), cohorts.
- States: decided(0) ← uncertain(1) → decided(1), or +*failed*

final: **decided**(1)

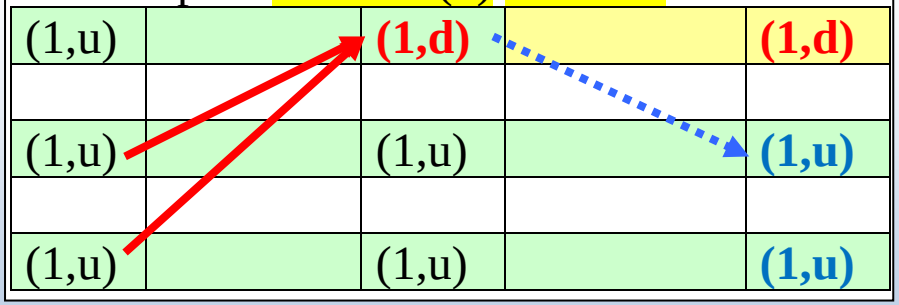


final: **decided**(0)

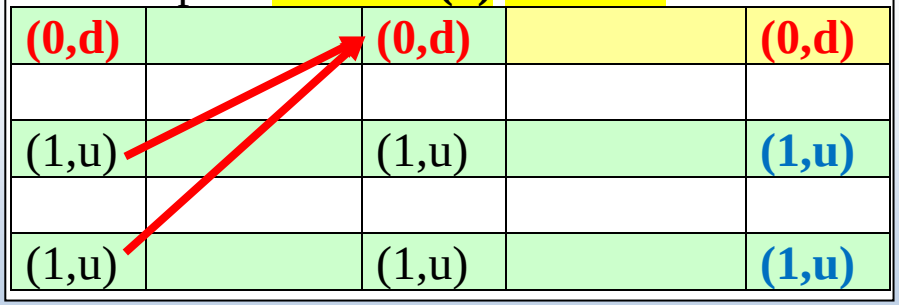


If #1 fails, then #2&3 cannot differentiate between left and right (*even if they talk*), will remain blocked

attempted **decision**(1) **blocked** for ever



attempted **decision(0) blocked** for ever



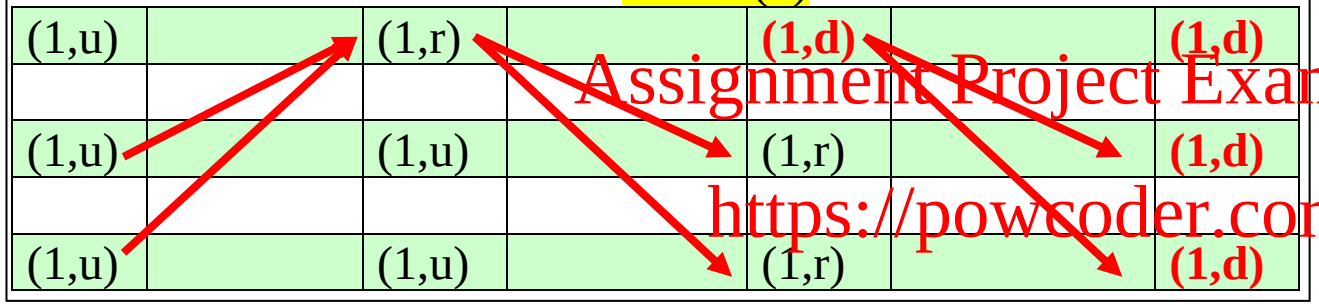
- 2PC blocks because the leader is too greedy to decide on 1 and fails before sending its messages

THE 3 PHASE COMMIT – STRONG TERMINATION ("NON-BLOCKING")

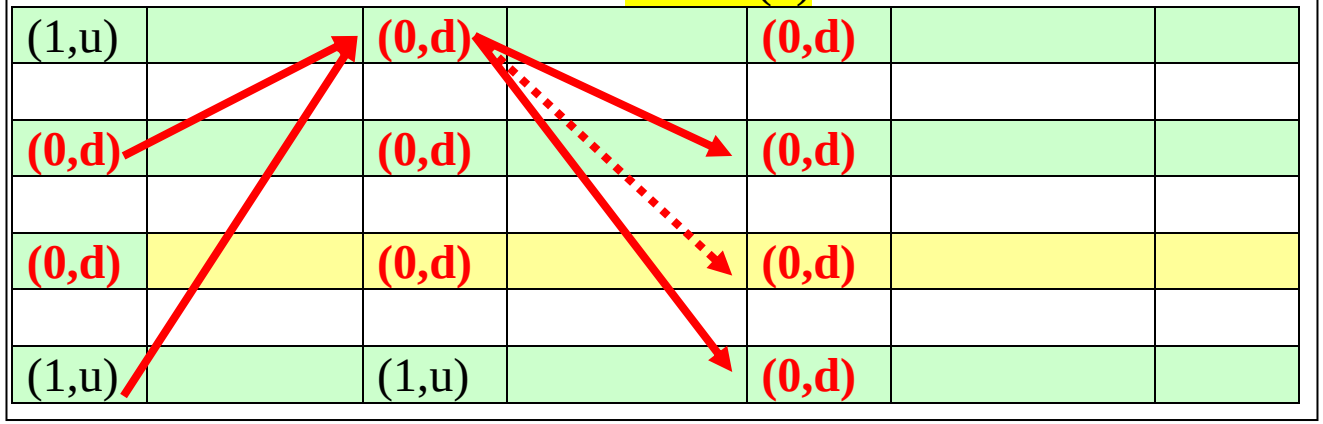
3PC algorithm is less greedy than 2PC and new coordinators arise in place of failed ones

States (one more, ready): decided(0) ← uncertain(1) → ready(1) → decided(1), or +*failed*

final: **decide**(1)



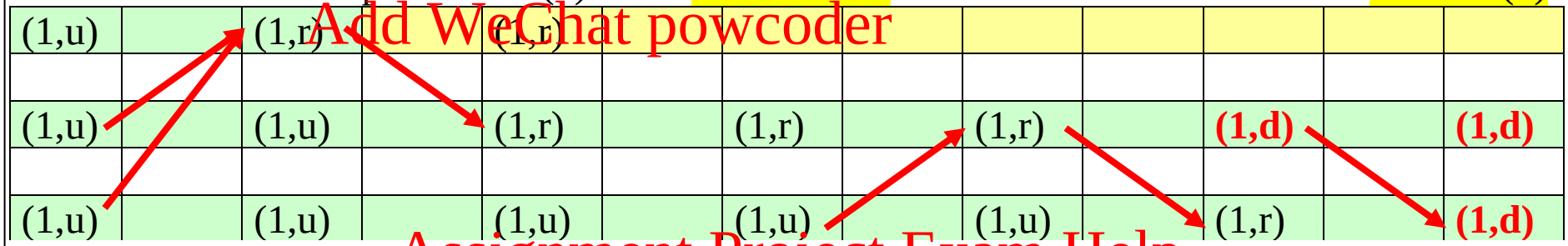
final: **decide**(0)



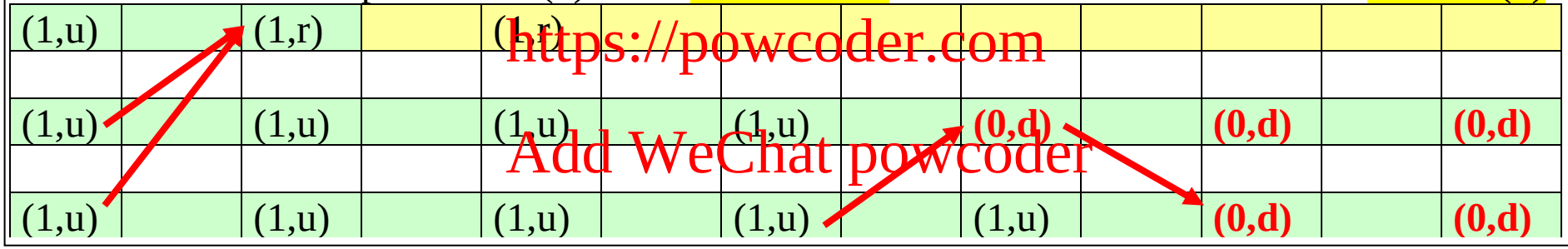
- State transition rules are "natural" and similar for all rounds, except:
- In the first 3 rounds:
  - Any missing message ⇒ decided(0)!
  - If all are uncertain, they will *attempt* ⇒ decided(1)!
- In round 4 and following:
  - Any missing message is ignored!
  - If all the rest are uncertain, they will *certainly* ⇒ decided(0)!

AVOIDING THE BLOCK

2PC would attempt **decision**(1) but it would block – 3PC doesn't block ⇒ final **decided**(1)



2PC would attempt **decision**(1) but it would block – 3PC doesn't block ⇒ final **decided**(0)



2PC would attempt **decision**(0) but it would block – 3PC doesn't block ⇒ final **decided**(0)