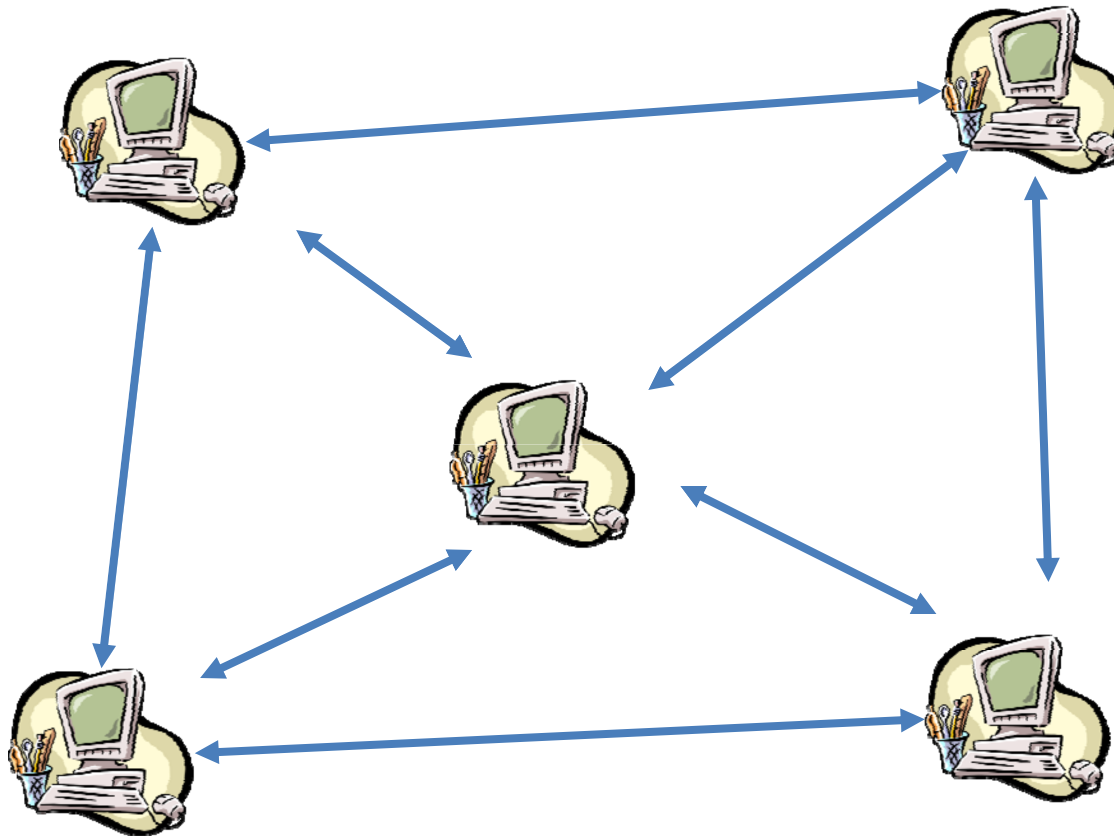
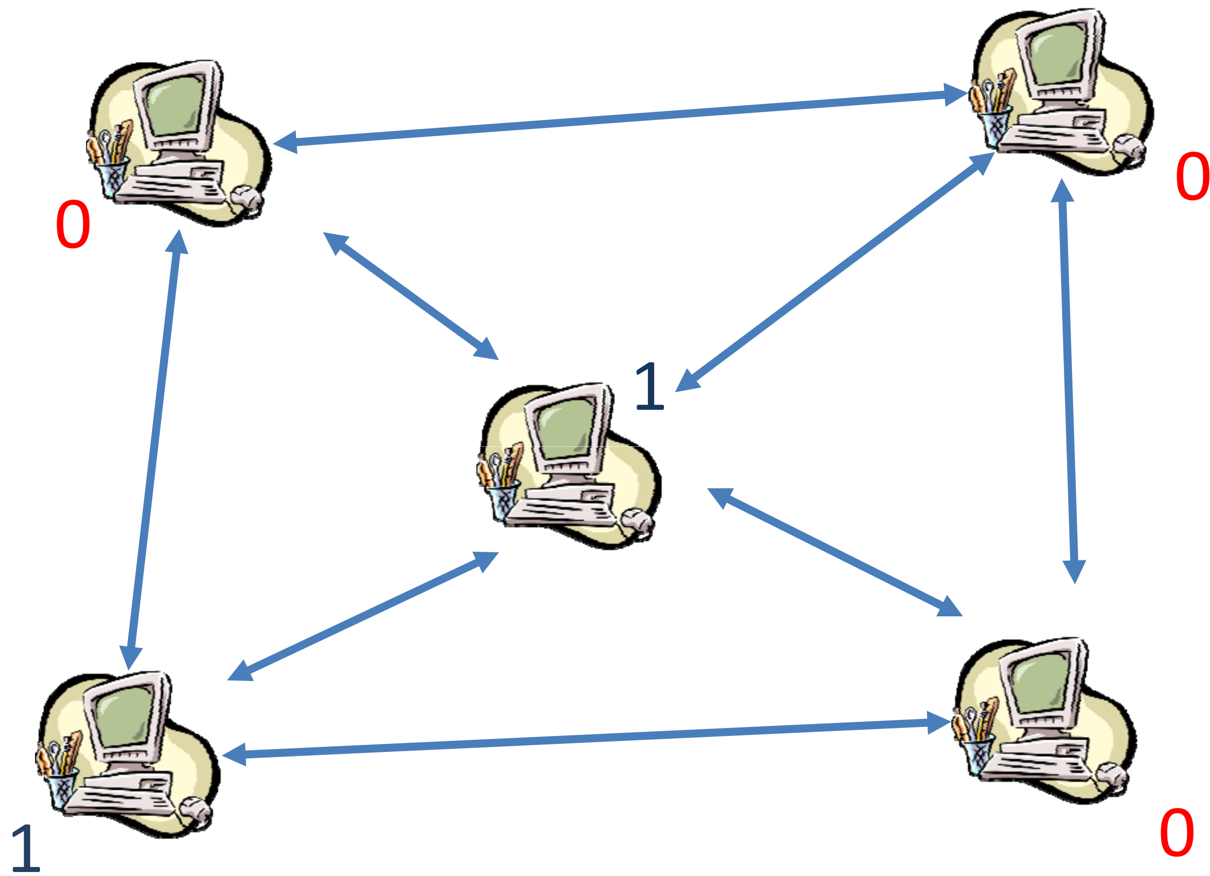
Generating Fast Indulgent Algorithms

Dan Alistarh, Seth Gilbert,
Rachid Guerraoui, Corentin Travers

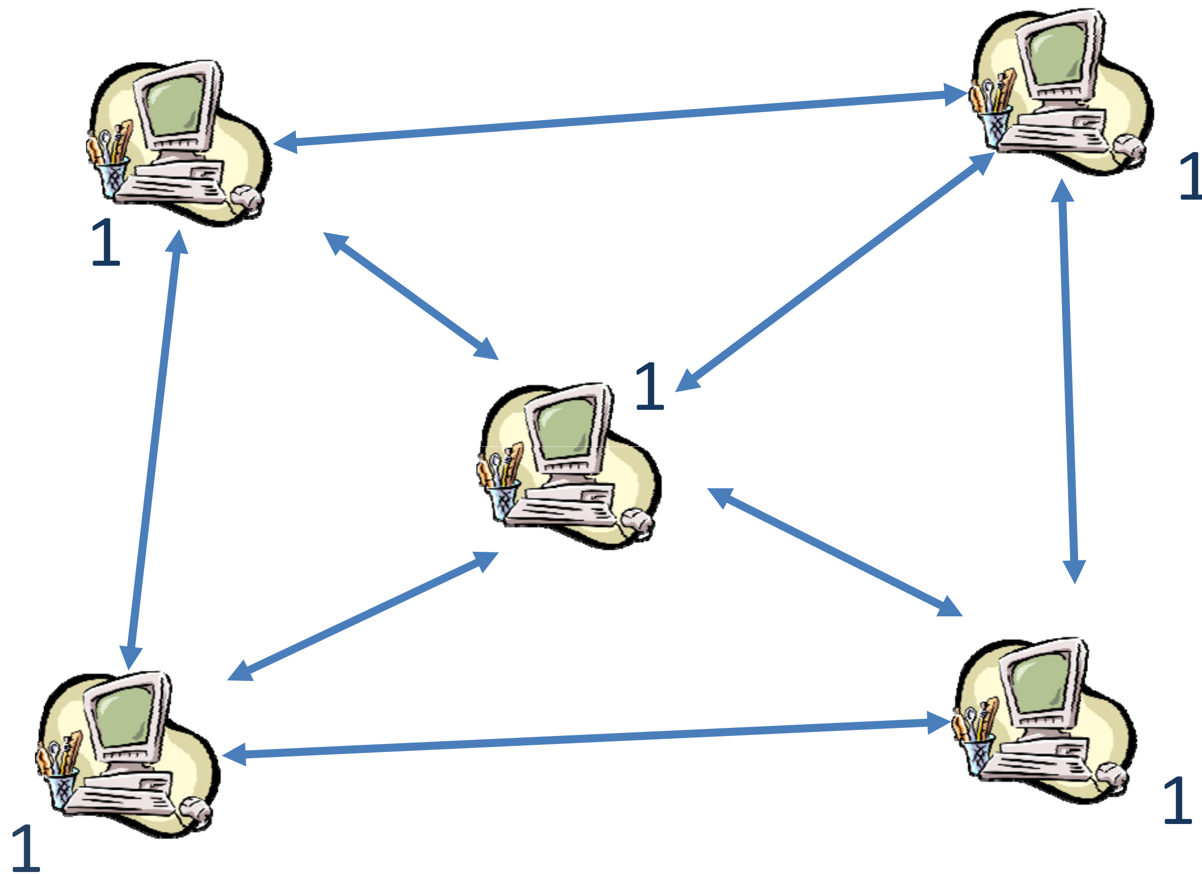
Distributed Tasks



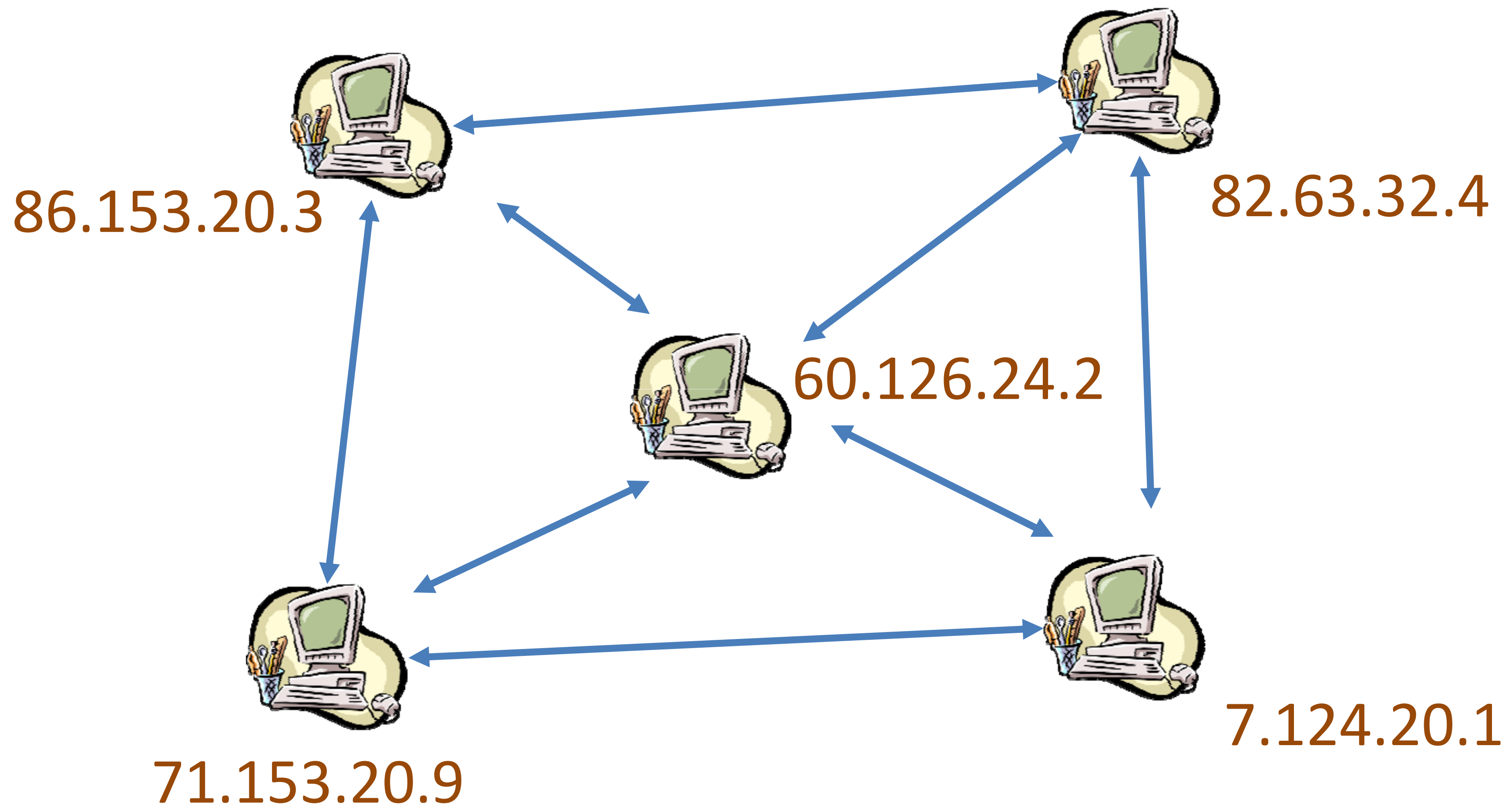
Distributed Tasks: Consensus



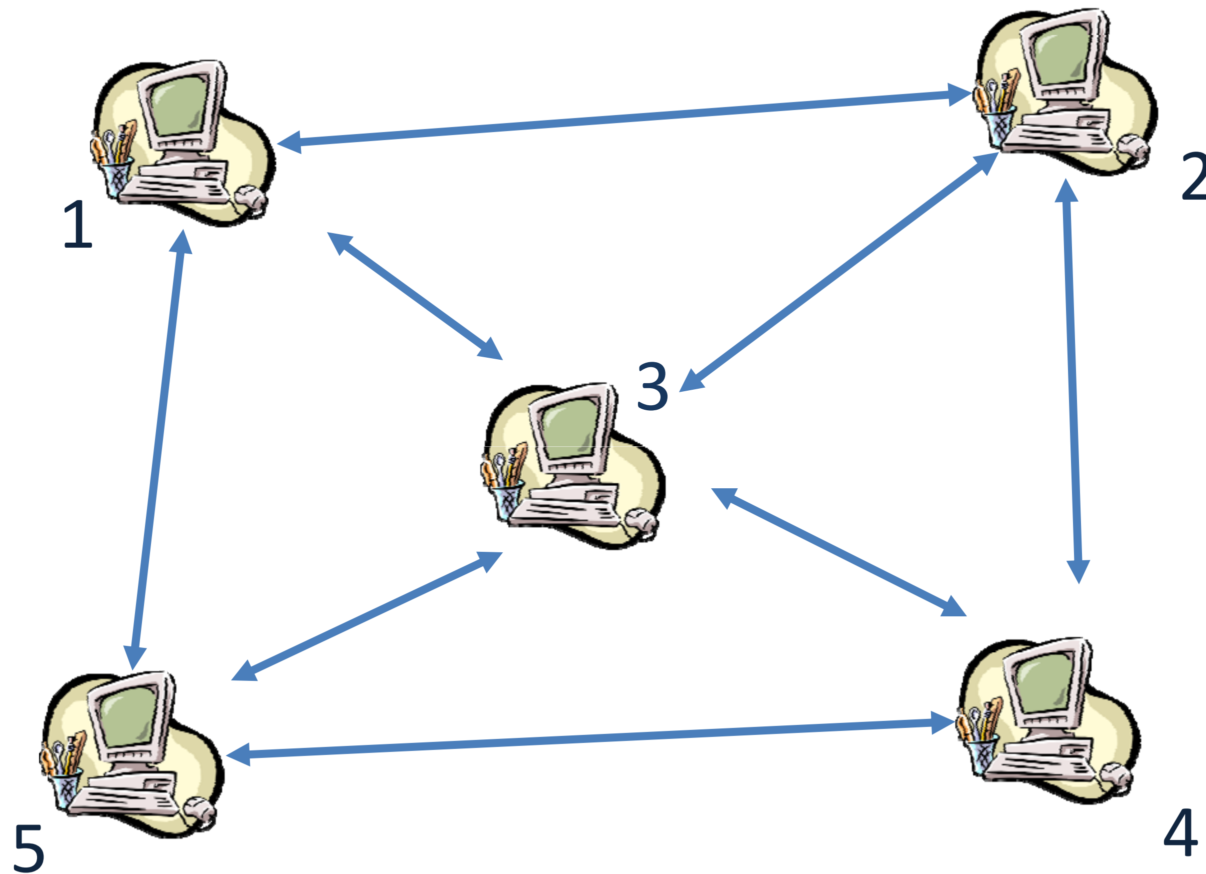
Distributed Tasks: Consensus



Distributed Tasks: Renaming



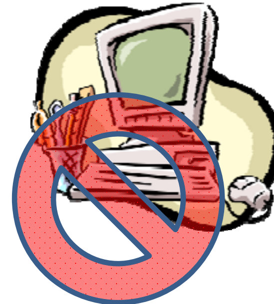
Distributed Tasks: Renaming



The enemy

- Asynchrony

- Process Failures



Hundreds of impossibility results

Theorem [FLP]. Consensus is *impossible* in an *asynchronous system* where a process may crash.

Theorem [HS]. Renaming in less than N + t names is *impossible* in an *asynchronous system* where t processes may crash.

The status

Synchrony with t failures

Synchronous consensus: **t + 1** communication rounds

Synchronous Tight Renaming : **log n + 2** rounds

"Speculative" algorithms

- PAXOS
- PBFT
- *Indulgent algorithms* [Guerraoui, Lynch]

Asynchrony with failures

Consensus is impossible

Tight renaming is impossible

Indulgent algorithms

- "Hope for the best, but provide for the worst"

Fast **Robust** ***Indulgent***

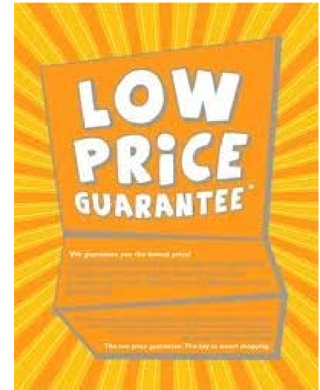
- Are *fast* when the execution is synchronous
- Maintain *safety* when the execution is asynchronous

Our contribution

- A new method transforming synchronous algorithms into indulgent algorithms

- Applies to *colorless* tasks (like consensus) and to *colored* tasks (renaming)

The price?

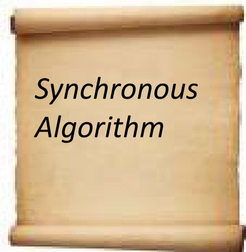


- We pay for indulgence in terms of performance on the "fast path"
- The cost is in terms of *time complexity*
- Additional cost of 2 communication rounds in *synchronous* runs
- For consensus, the price of indulgence is of one communication round [DG]

The idea

- We introduce a new abstraction: *the asynchrony detector*

- The transformation:



Execution

No Asynchrony!

Decide!

The idea

- We introduce a new abstraction: *the asynchrony detector*
- The transformation:

Synchronous Algorithm

Execution

Asynchrony!

Hand-off

Backup algorithm

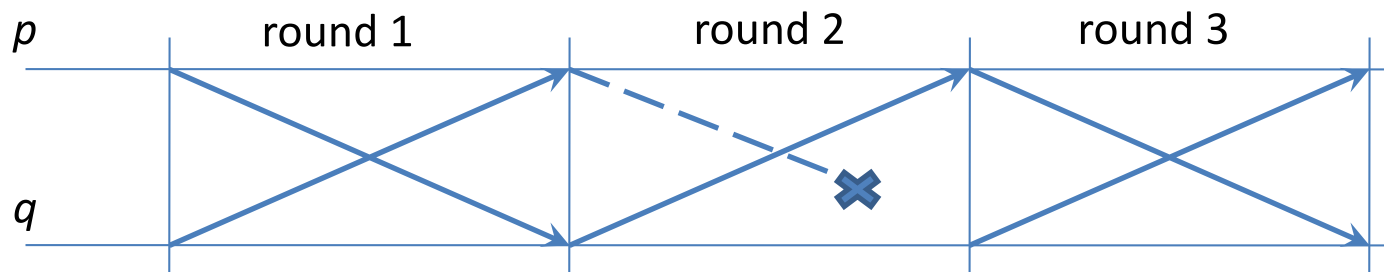
The challenge:
implementing the detector
and the hand-off *asynchronously,*
with failures

The plan

- Indulgent algorithms
- Contribution
- Model
- Detecting asynchrony
- Applications
- Conclusions and Future Work

Model

- N processes, t < N might fail
- Message-passing
- Communication rounds:
  - Synchronized across processes
  - Message delivery is asynchronous
- A process receives at least N – t messages per round

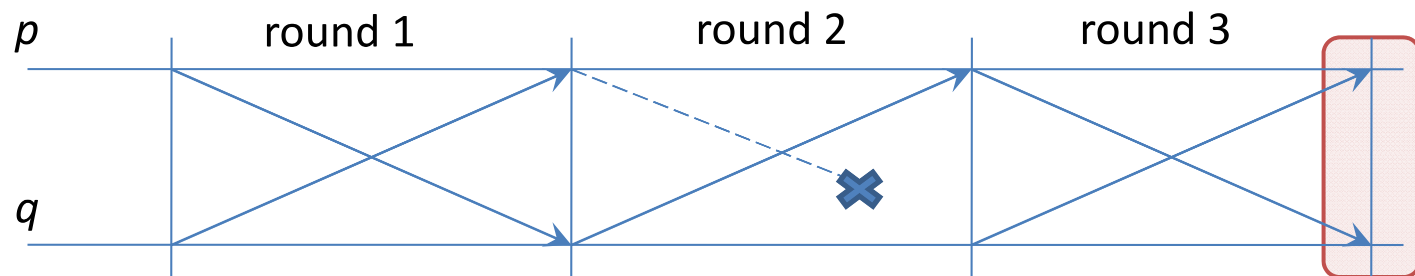


Asynchrony detector

- A distributed service, implemented locally
- Returns a YES/NO indication at each round
- Parameter d (the detection delay)
- Three properties at round R
  1. (**Local detection**) If YES, the process "sees" a synchronous execution
  2. (**Global detection**) All processes getting YES in round R share a synchronous execution prefix in rounds [1, 2, …, R – d]
  3. (**No false positives**) It never returns NO if the execution is synchronous

AD(2) implementation

- A process maintains "logs" of active/failed processes for each round as part of its state
- Process is active if message received, failed otherwise
- Full-information exchange in every round
- Rounds in which processes have been mistakenly believed to have failed are detected as *asynchronous*



Lemma: AD(2) implements an asynchrony detector with delay 2!

Both p and q mark round 2 as *asynchronous*

AD(2) properties

Claim: AD(2) implements an asynchrony detector with delay 2!

- Key property
  - (Global detection) All processes that get YES at round R share a synchronous execution prefix up to round R - d
- AD(2) is implemented asynchronously

Using AD(2) for consensus

procedure propose$_i$(v, A)

1. *in parallel*, for t + 1 rounds
   - *run* algorithm A
   - *run* AD(2)
2. run AD(2) for 2 more rounds
3. if AD(2) returns YES at t + 3
4. return decision from A
5. else run Hand-Off and Backup

Synchronous Algorithm t + 1 rounds

t + 3 rounds

Why is the Hand-Off necessary?

Process 2 detects asynchrony and cannot decide!

The hand-off is needed to ensure that Process 2 does not decide a different value using the backup algorithm!

The Hand-Off Procedure

- Differs for colorless and colored tasks
- For colorless tasks (e.g. consensus):
  - If decision on the "*fast path*", then all processes adopt the decided value as their proposed value for the backup
- For colored tasks (renaming)
  - If decision on the "fast path", no process decide the same value (name) using the backup
- Good news: both conditions can be implemented asynchronously

The back-up

- Can be any speculative algorithm solving the same task
- Can do nothing (abort)
- Can even be another iteration of our scheme!

Wrap-up

- For consensus, we obtain indulgent algorithms that decide in $t + 3$ communication rounds
- For renaming, we obtain an algorithm that uses at most $N + t$ names
  - N names in synchronous executions!
  - Decides in $\log N + 4$ rounds on the "fast path"

Theorem. Given a synchronous algorithm A solving a colorless or colored task with time complexity R, we can obtain an indulgent algorithm with time complexity R + 2.

Future work

- Can we transform synchronous algorithms into ***Byzantine-resilient*** indulgent algorithms?
- Can we generate algorithms that terminate quickly when the system ***stabilizes***? (eventually synchronous)
- Message complexity?

The message

- Theory:
  - Most synchronous algorithms can be made indulgent, modulo a constant increment in time complexity
- Practice:
  - When building a speculative algorithm, it may be a good idea to start from the synchronous algorithm and make it robust
- Extra:
  - We can "detect" asynchrony in executions of a system

Questions?

Consensus Hand-Off

- At round R + 2
  - If AD(2) says YES: decide value from As at R
  - If AD(2) says NO:
    - If no processes had YES in round R + 1, start Backup with initial state
    - Else let Supp = processes with YES in R + 1:
      - Recover a valid state (estimate) from the processes in Supp
      - Use it as initial value for the Backup
- Key property: if a process *decides* at round R + 2, then *all correct processes* have the decision value as their initial value for Backup