You Only Live Multiple Times

Black box re-use of Crash-Stop Algorithms In Realistic Crash-Recovery Settings

DAVID KOZHAYA[1], OGNJEN MARIC[2], AND YVONNE-ANNE PIGNOLET[1]

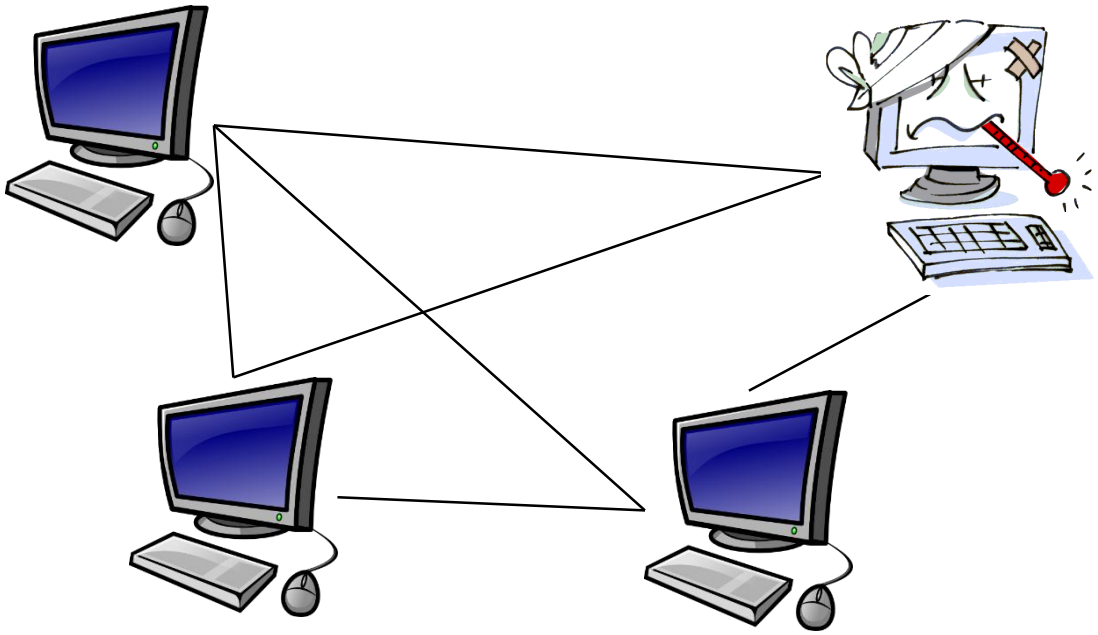
[1] ABB CORPORATE RESEARCH SWITZERLAND , [2] DIGITAL ASSET SWITZERLAND

A big thank you to Klaus Tycho-Foerster for presenting on our behalf!

Mitigating The Effect of Failures

Fault Tolerance in Distributed Systems

Failures happen in real systems



Consensus: typical way to mitigate the effect of failures

To minimize the impact of failures on service interruption, implement consensus protocols that tolerate failures

- Distributed parties agree on the actions to perform despite failures

Consensus protocols studied under different synchrony and failure assumptions

Solving Consensus in Presence of Failures

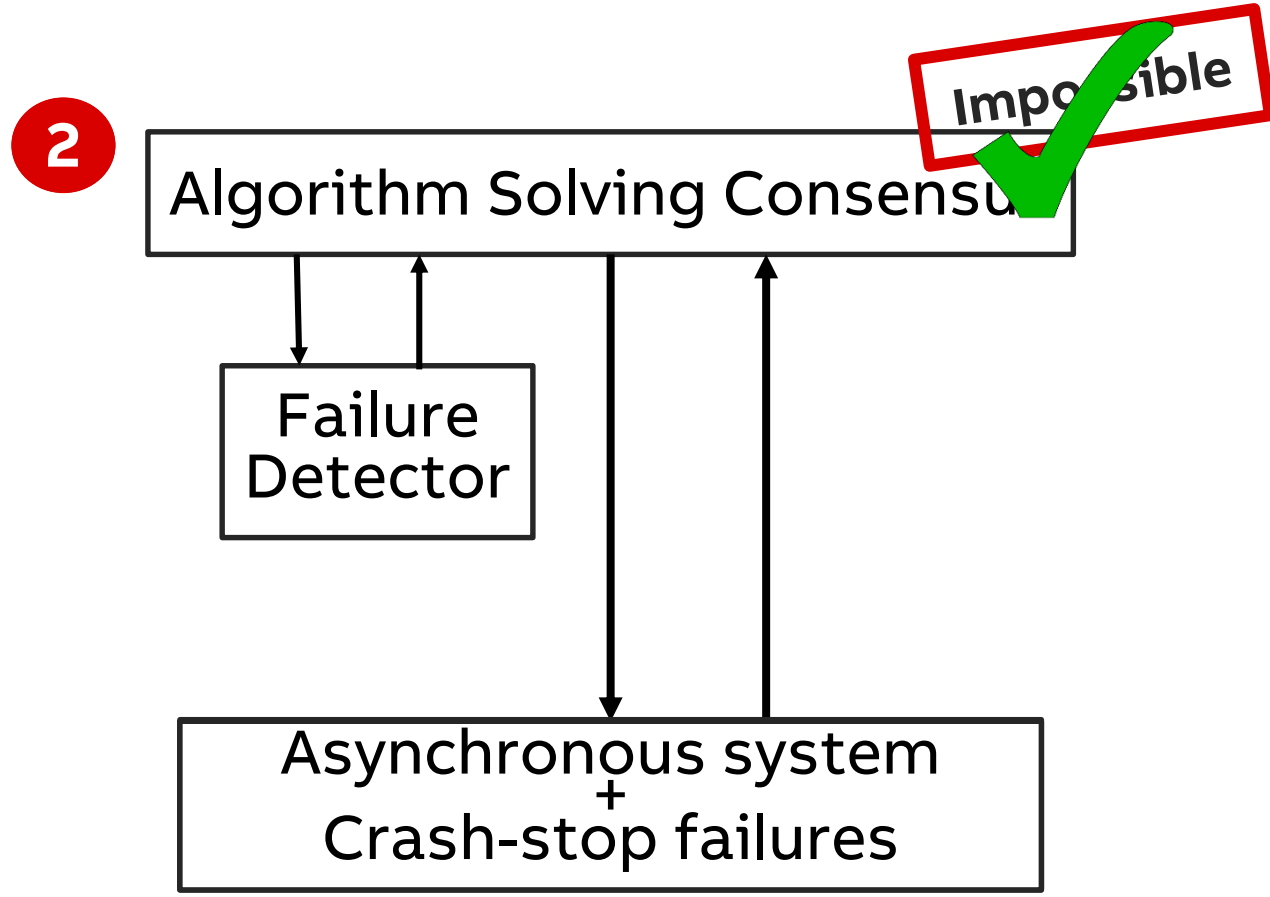
Asynchronous system model

- Unbounded processing delays
- Unbounded communication delays

Crash-stop failure model (CS model)

- The simplest failure model
- A process crashes by stopping to execute the algorithm forever

Partial synchrony and failure detectors rely on system conditions that eventually hold forever



In reality, failure and recovery modes of processes and links are probabilistic and temporary

Crash Recovery Settings

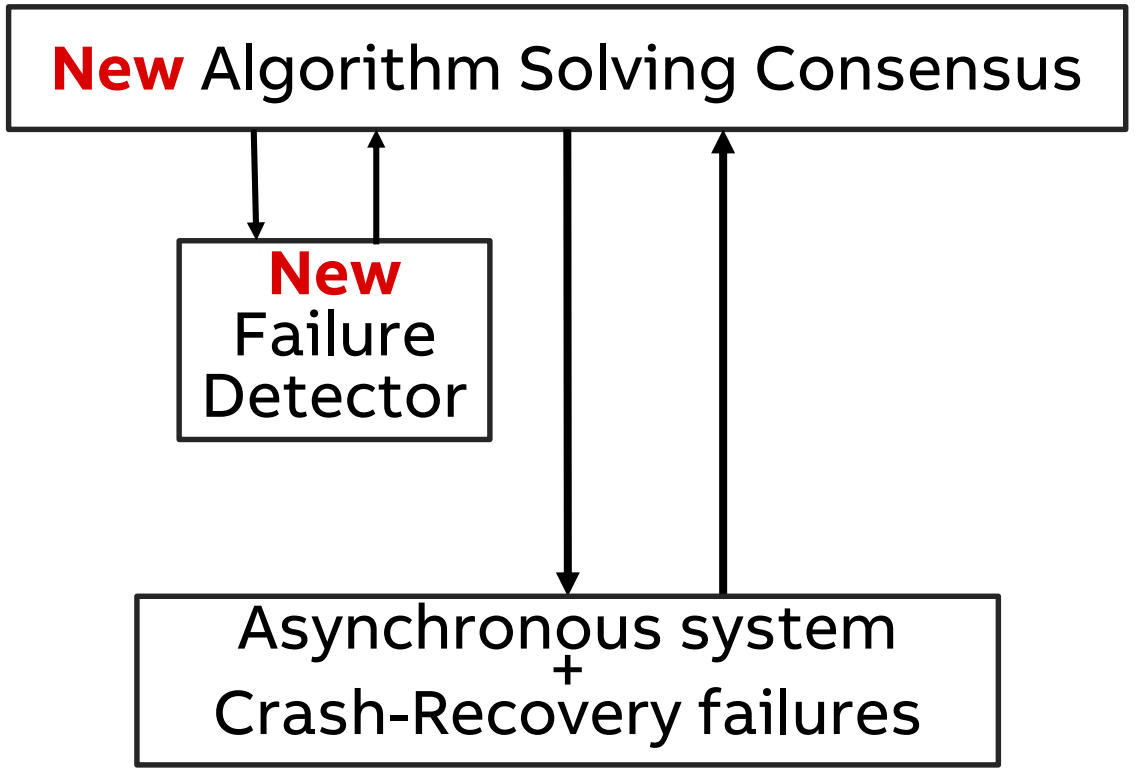
A Way To Capture a System's Dynamicity

Crash-recovery failure model (CR model)

- Process can join and leave unannounced
- Does not address communication

Subsequently:

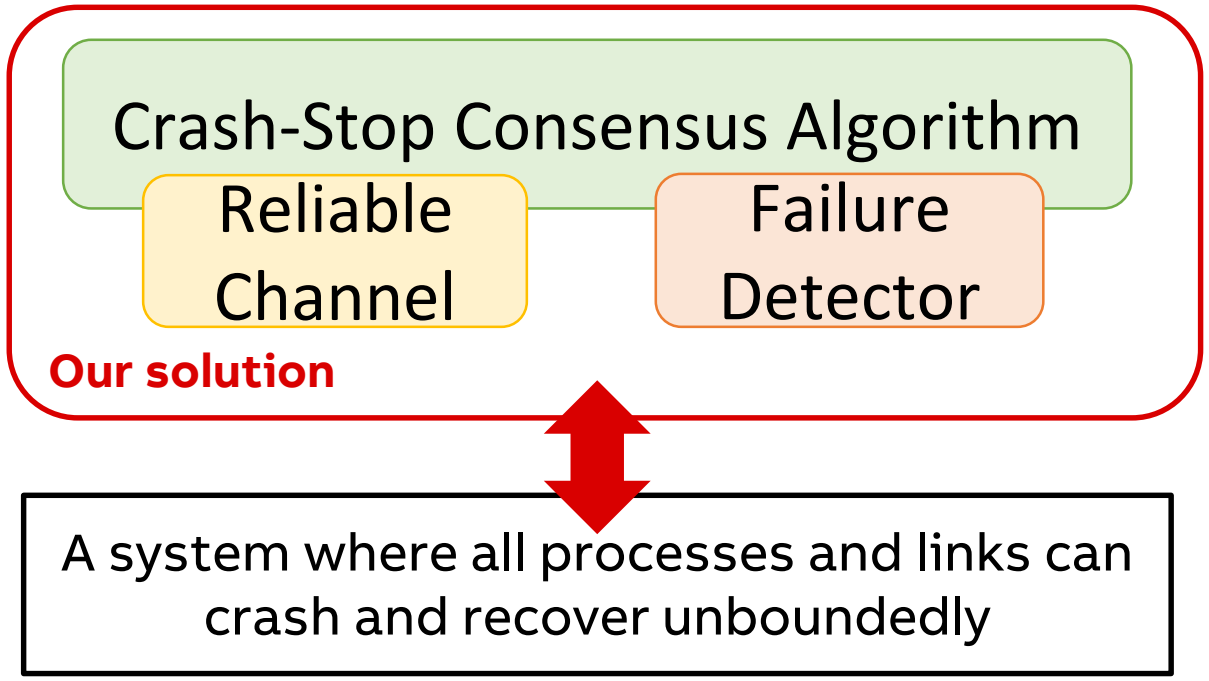
- New failure detectors and new consensus algorithms on top
- Processes that crash and recover infinitely often are excluded – not required to satisfy consensus properties



What remains unanswered:
Can the plethora of existing crash-stop algorithms be reused unchanged in crash-recovery settings?

Our Contribution

Re-use crash-stop consensus algorithms with reliable links and failure detectors in crash-recovery model



Deterministic CS (crash-stop) consensus algorithms implement consensus with probability 1 in CR systems where processes and links crash and recover unboundedly

The Rest of This Talk

- What is different about our approach compared to exiting works
- What do we assume in our models
- How our wrapper works
- What class of algorithms benefit from our results

What is different compared to exiting works?

Difference with Existing Literature

Existing solutions

– Existing crash-recovery deterministic solutions:
  - Implement consensus deterministically
  - Exclude processes that crash and recover unboundedly
  - Introduces new failure detector definitions and consensus algorithms

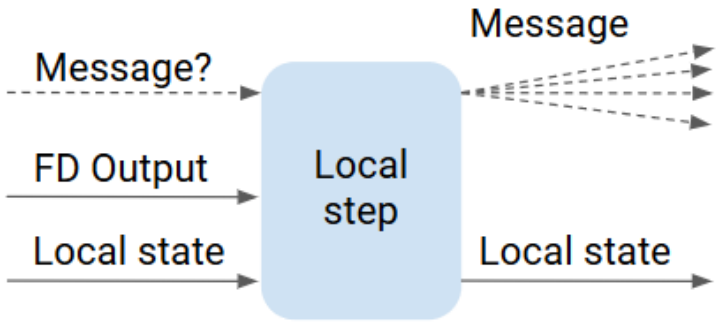
– Existing probabilistic solutions:
  - Implement consensus with probability 1
  - Introduces new consensus algorithms, e.g., based on random coin flips

Our approach

– Our approach:
  - Implement consensus with probability 1
  - Include processes and links that crash and recover infinitely often
  - Is modular – does not introduce new algorithms but rather uses existing crash stop algorithms

Description of Our Models

Reliable asynchronous crash-stop model



Local states	p_1 x: 1 y: 2 p_2 x: 3 y: 0 p_3 x: 3 y: 2
Failed	p_2
In-flight messages	$p_1 \longmapsto p_2$: M_1 $p_3 \longmapsto p_1$: M_2

Reliable asynchronous CS Algorithm

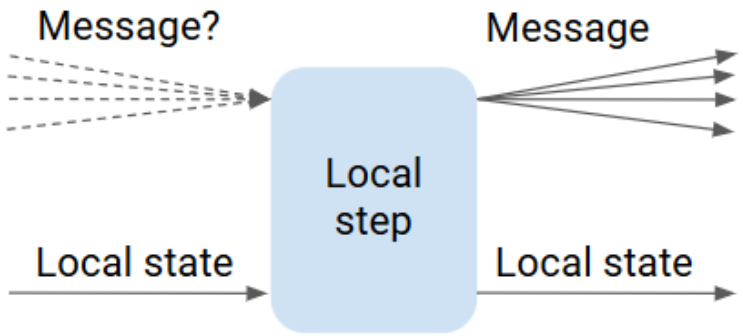
$$next : \Sigma_p \times (\Pi \times \mathcal{M})_\perp \times \mathcal{R} \to \Sigma_p,$$
$$send : \Sigma_p \to (\Pi \rightharpoonup \mathcal{M})$$

Reliable Channel

Failure Detector

Time: asynchronous processes (no clocks), asynchronous links
Failures: processes may crash, all messages get delivered

Lossy synchronous crash-recovery model



Local states	p_1 x: 1 y: 2 ; p_2 x: 3 y: 0 ; p_3 x: 3 y: 2
Failed	p_2
Global round #	3

Lossy Synchronous CR Algorithm

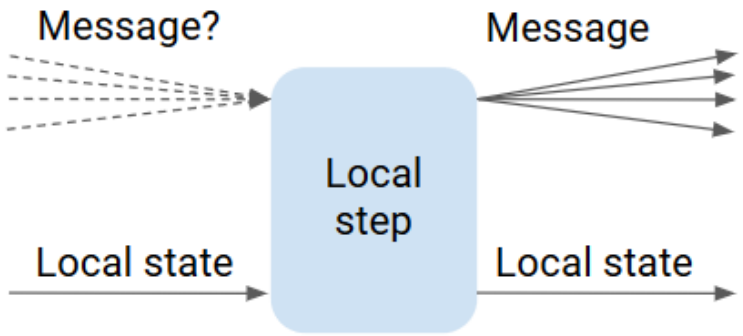
$$next : \Sigma_p \times (\Pi \rightharpoonup \mathcal{M}) \rightarrow \Sigma_p$$

$$send : \Sigma_p \rightarrow (\Pi \rightarrow \mathcal{M})$$

Lossy Channel

Time: synchronous steps, synchronous links (upper bounds for steps and transmission)
Failures: processes may crash and recover infinitely often, messages may get lost

Probabilistic crash-recovery model



Local states	p_1 x: 1 y: 2	p_2 x: 3 y: 0	p_3 x: 3 y: 2
Failed	p_2		
Global round #	3		

Probabilistic CR Algorithm

$$next : \Sigma_p \times (\Pi \rightharpoonup \mathcal{M}) \rightarrow \Sigma_p$$

$$send : \Sigma_p \rightarrow (\Pi \rightarrow \mathcal{M})$$

Prob. lossy Channel

Time: synchronous processes, synchronous links
Failures: processes and link crash and recover with probability in (0,1)

Model overview

Reliable Asynchronous CS

Reliable asynchronous CS Algorithm

$$next : \Sigma_p \times (\Pi \times \mathcal{M})_\perp \times \mathcal{R} \to \Sigma_p,$$
$$send : \Sigma_p \to (\Pi \rightharpoonup \mathcal{M})$$

Reliable Channel

Failure Detector

Async processes and links
Processes may crash,
reliable communication

Lossy Synchronous CR

Lossy Synchronous CR Algorithm

$$next : \Sigma_p \times (\Pi \rightharpoonup \mathcal{M}) \to \Sigma_p$$
$$send : \Sigma_p \to (\Pi \to \mathcal{M})$$

Lossy Channel

Sync processes and links
Processes may crash and recover,
lossy communication

Probabilistic CR

Probabilistic CR Algorithm

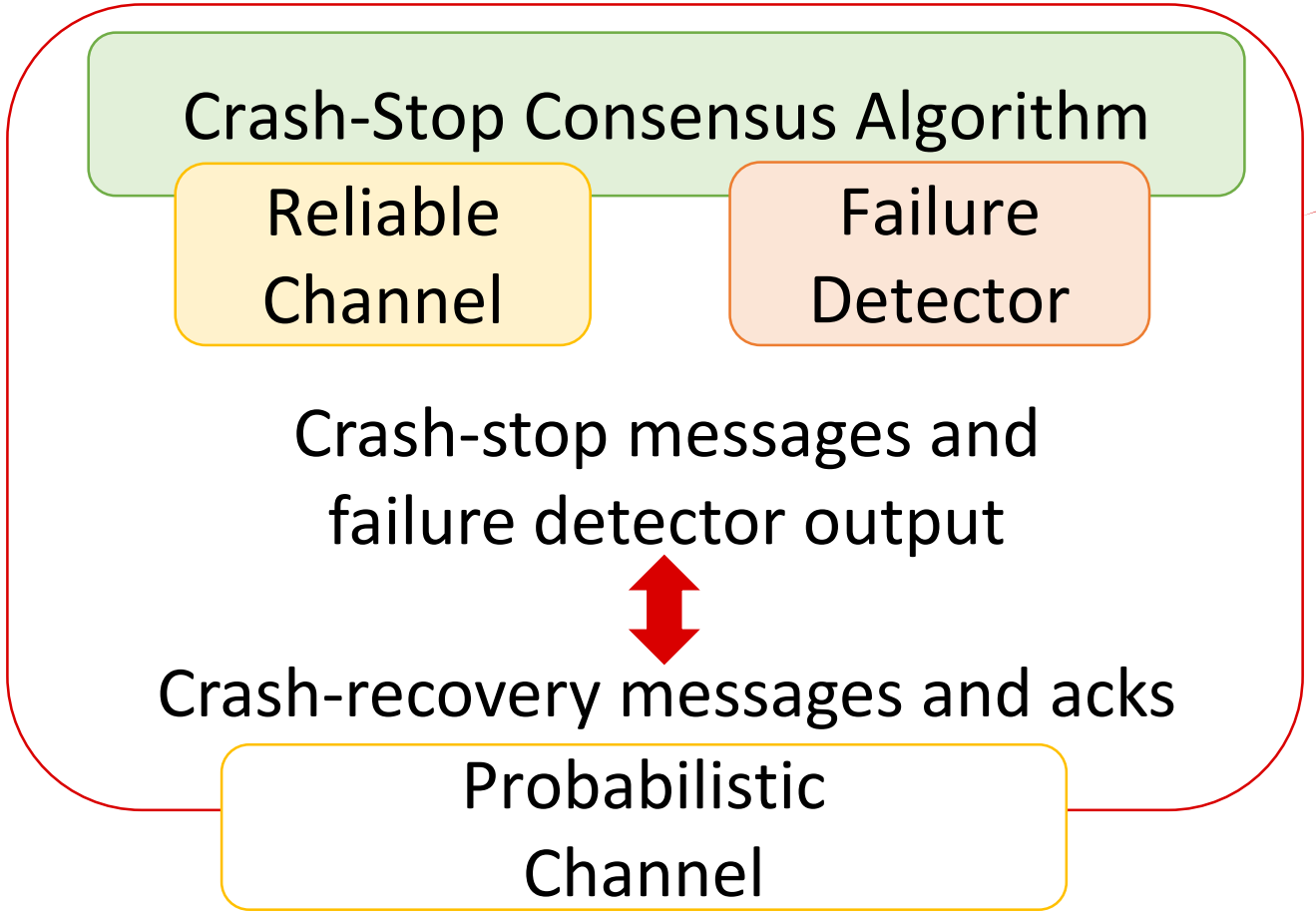
$$next : \Sigma_p \times (\Pi \rightharpoonup \mathcal{M}) \to \Sigma_p$$
$$send : \Sigma_p \to (\Pi \to \mathcal{M})$$

Prob. lossy Channel

Sync processes and links
Processes crash and recover probabilistically, messages dropped probabilistically

How our Wrapper works

Crash-Recovery Wrapper



The red box is our wrapper

- Create synchronous crash-recovery step using multiple crash-stop steps (each handling one message)
- Round-by-round failure detector to produce outputs to be fed to CS algo
- Provide reliable links by LIFO buffering and retransmission until ack

What algorithms benefit from our results

Bounded Algorithms

The Class of Algorithms To Which Our Results Apply

A “bounded” crash-stop consensus algorithm satisfies for fixed B, B_s, B_Δ:

(B1) Communication-closed rounds: Processes operate in rounds, only messages from current round are considered.

(B2) Externally triggered state changes: In every round, a process changes state only upon message receipts or failure detector output changes.

(B3) Bounded round messages: In any round, a process sends at most B_s messages to any other process.

(B4) Bounded round gap: The fastest (n-f) processes are always at most B_Δ rounds apart.

(B5) Bounded termination: Given any time t where the fastest (n-f) processes are correct, all other processes are faulty, and the failure detector output is perfect after t, then all (n-f) fastest processes decide before any of them reaches round B_{max} = max_round(t)+B.

Theorem.
If a bounded algorithm solves consensus in the crash-stop setting, then this algorithm using our wrapper solves consensus with probability 1 in our crash-recovery setting

Bounded Algorithms

Examples

Examples of existing algorithms that are bounded are:

- The Chandra-Toueg algorithm [1]
- Algorithms in the generic indulgent framework of [2]

[1] Tushar Deepak Chandra and Sam Toueg. Unreliable failure detectors for reliable distributed systems. J. ACM, 43(2):225–267, 1996.

[2] Rachid Guerraoui and Michel Raynal. A generic framework for indulgent consensus. In ICDCS, 2003.

Conclusion

- Introduced system models that closely capture the messy reality of distributed systems
- Allowed processes and links to fail and recover for an unbounded number of time
- Proposed a wrapper to deploy crash-stop algorithms as a black box in our crash-recovery setting
- Determined the conditions for reusing crash-stop algorithms unchanged in our crash-recovery setting

david.kozhaya@ch.abb.com; ogi.yolmt@mynosefroze.com; yvonneanne@pignolet.ch