Defending Distributed Cyber-Physical Systems with Bounded Time Recovery

Brian Sandler, Neeraj Gandhi, Linh Thi Xuan Phan, Andreas Haeberlen

NSF/Intel CPS PI Meeting

July 2018

Machines in Control

Vulnerable CPS can cause disaster.

- Explosion
- Equipment damage
- Power outages
- …

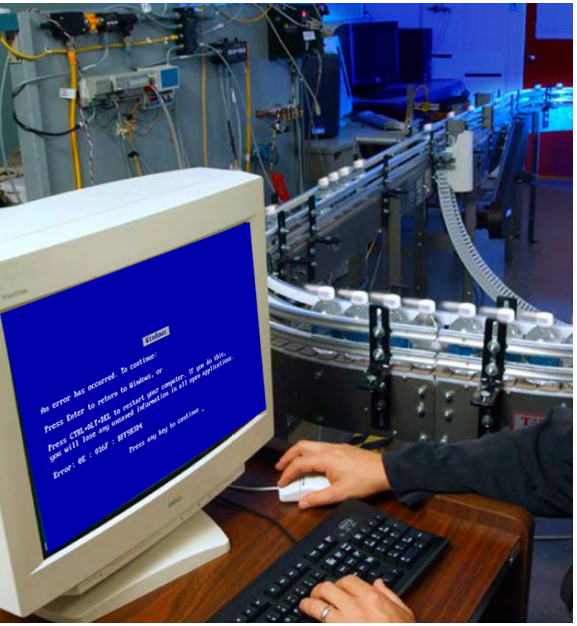
We want to prevent disaster.

Bellingham, WA
Oil pipeline explosion after the two controlling computers failed.

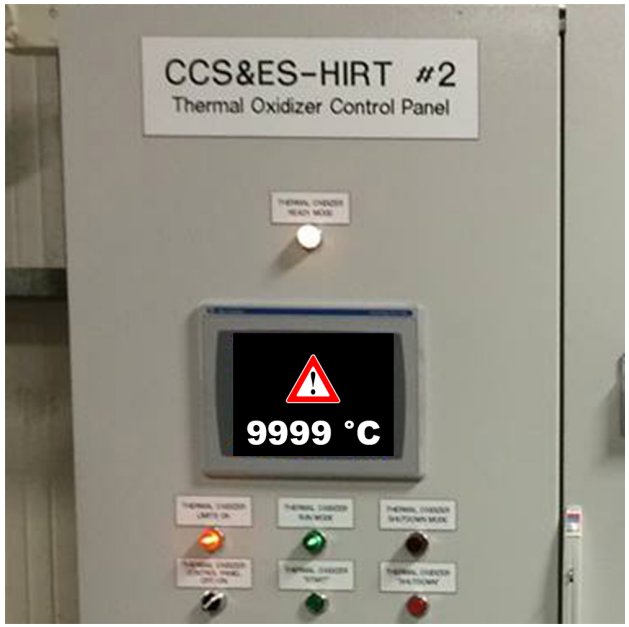
Iran
Stuxnet vulnerability destroyed centrifuges used for nuclear enrichment.

Ivano-Frankivsk, Ukraine
Controlling power grid systems were compromised leaving residents in the dark.

Goal: General Defense



Crashes



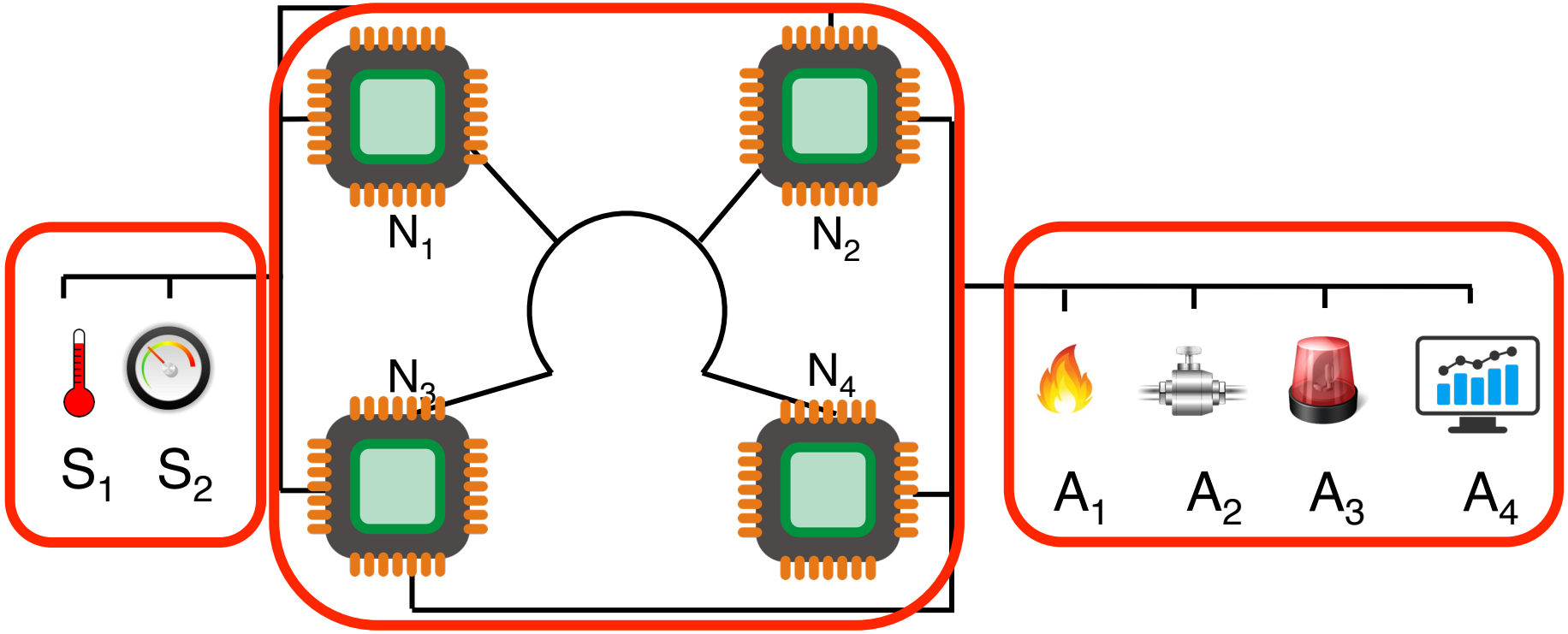
Non-Crash Bugs

Hacking

Byzantine Faults

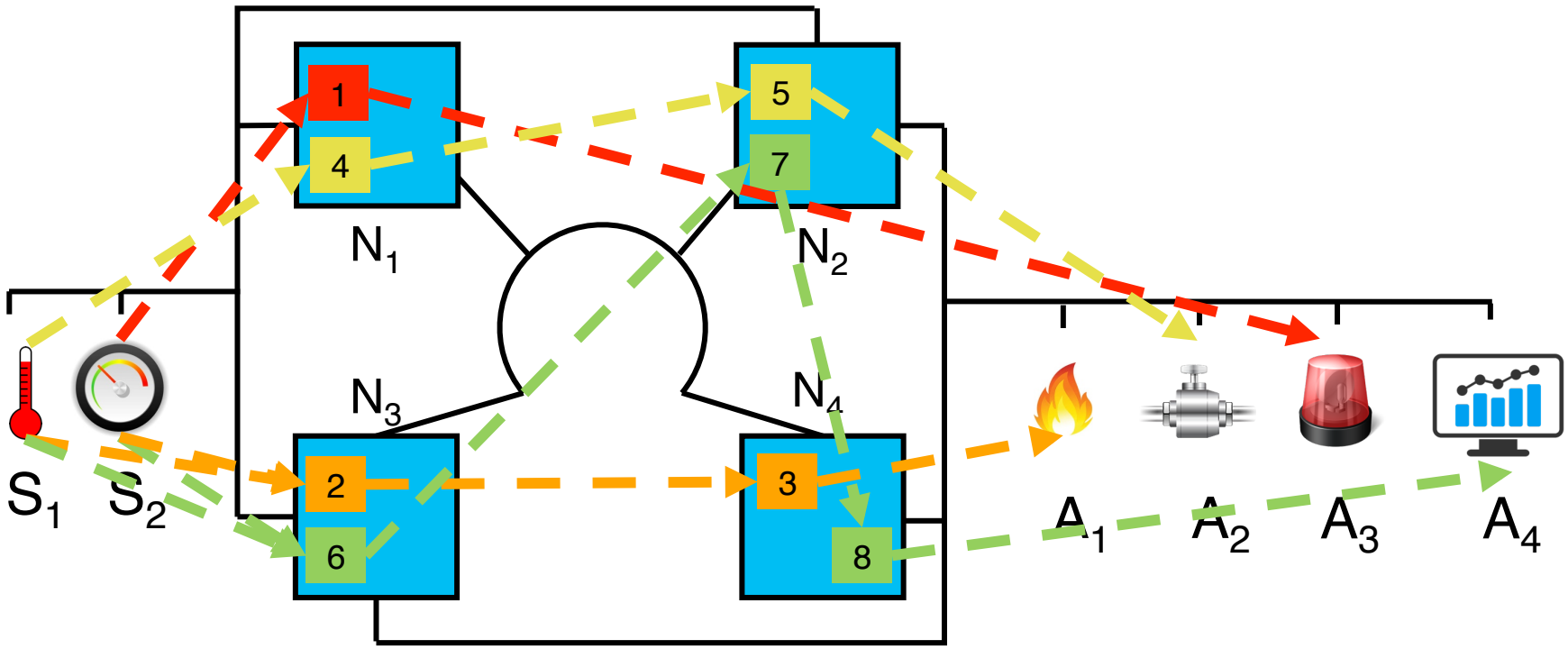
Example: Industrial Automation

Let's take a simple example system…



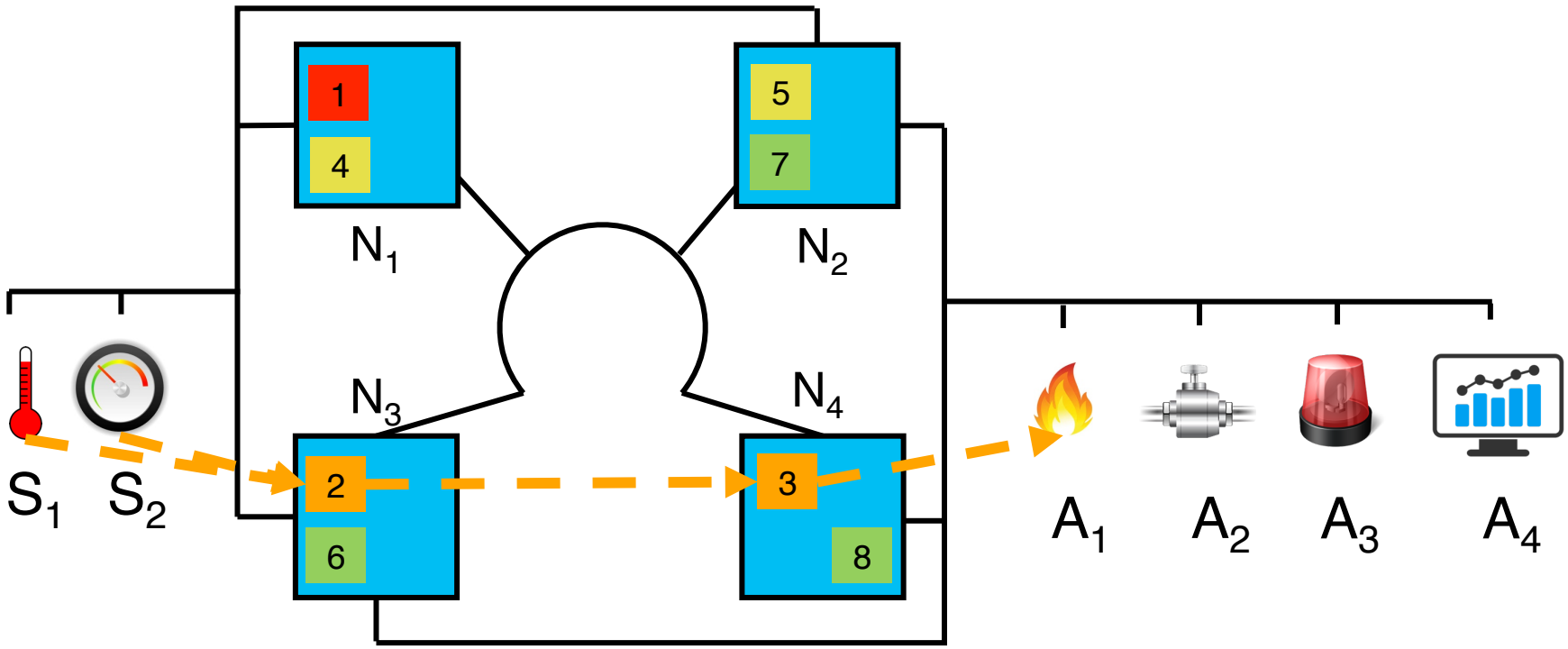
Example: Industrial Automation

This system will run four applications.



Example: Industrial Automation

We'll focus on the burner control application…

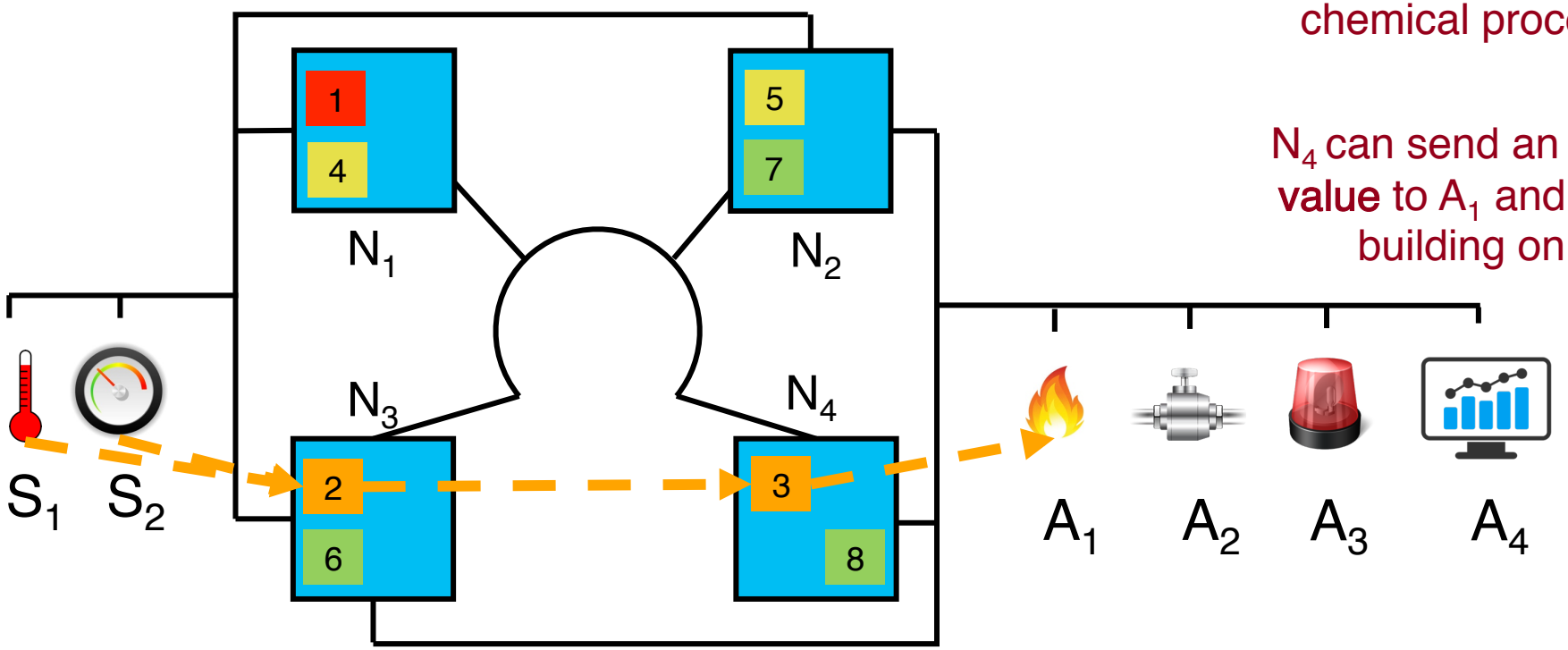


Example: Impact of Failures

What can go wrong?

N_4 can **drop** or **delay** messages and ruin th chemical processing

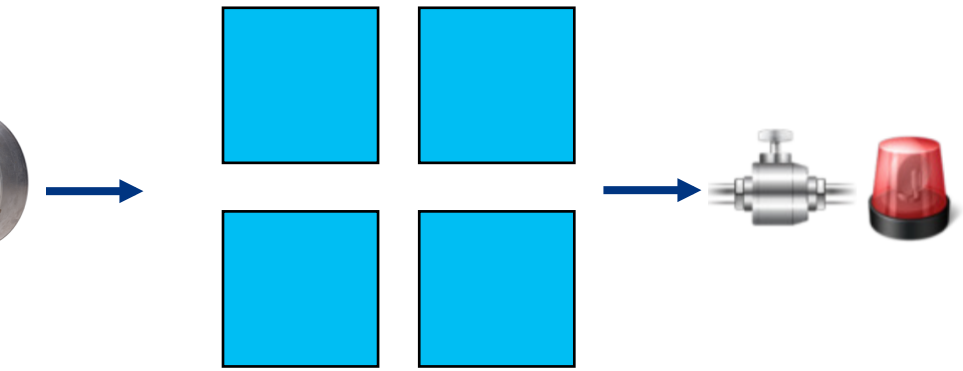
N_4 can send an **incorr value** to A_1 and light t building on fire.



State of the Art: Byzantine Fault Tolerance

Benefits

- Adversarial Scenarios
- Strong Guarantees
- Nice Programming Model



Problem: Expensive

- Lots of redundancy

Problem: Not Sufficient

- Cannot meet deadlines
- Guarantees drop after f faulty nodes

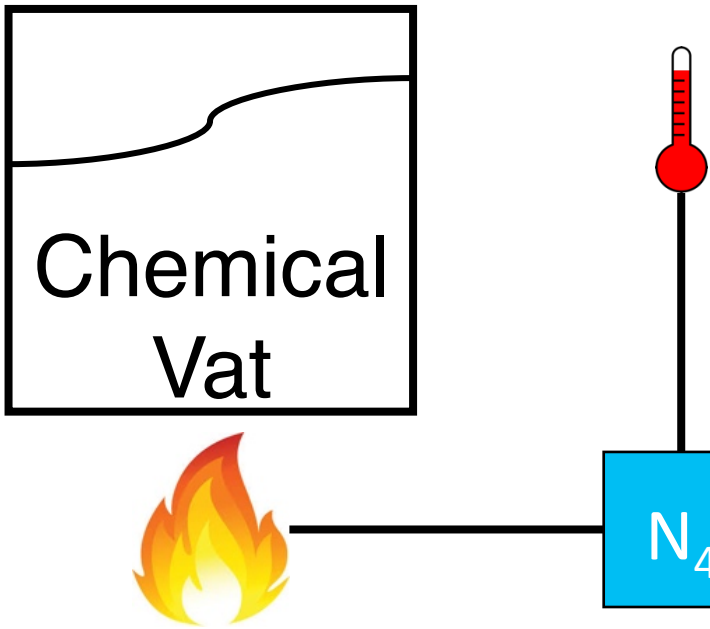
s continuous perfection required?

How bad is it if the adversary gains control?

Many CPS have properties that resist quick changes

- inertia
- thermal capacity

We don't have to always be perfect



We can leverage this!

For how long is faulty behavior okay?

Different applications have different tolerances.

DC/DC converters (STM)	20µs
Direct torque control (ABB)	25µs
AC/DC converters	50µs
Electronic throttle control (Ford)	5ms
Traction control (Ford)	20ms
Micro-scale race cars	40ms
Autonomous vehicle steering	50ms
Energy-efficient building control	500ms

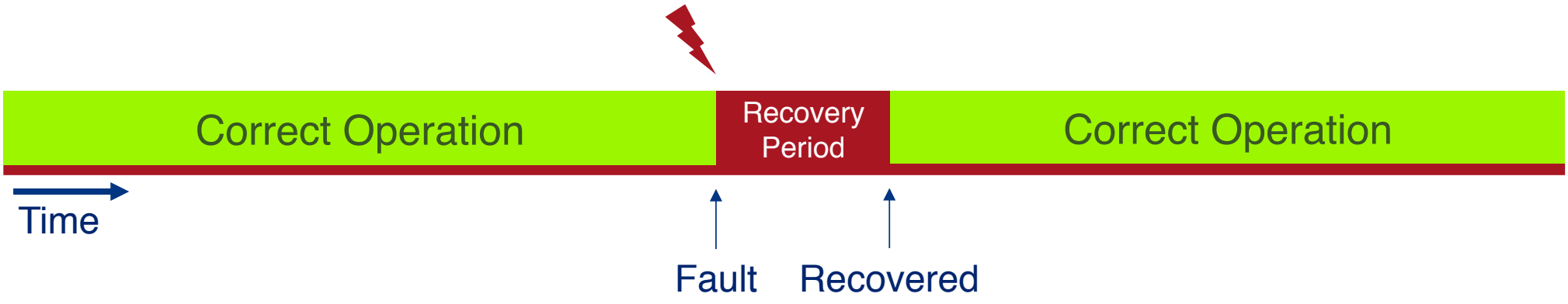
Source: M. Morari. Fast model predictive control (mpc).

A time period usually exists where faulty behavior is ok so long as the system returns to its correct behavior within that period.

Approach: Bounded Time Recovery

BTR guarantees that system recovers from any fault within a short period of time, so that the end goal will be met

Weaker guarantee is often sufficient



So, how do we make this happen?

REBOUND

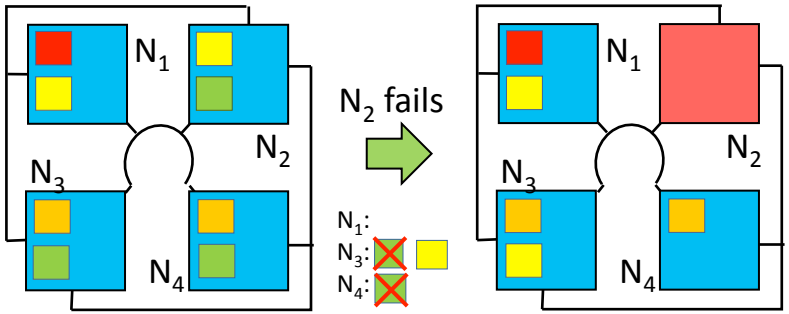
REBOUND

. Planning

Before system is compromised, think about what it should do.

System operates in different modes for any given set of faults.

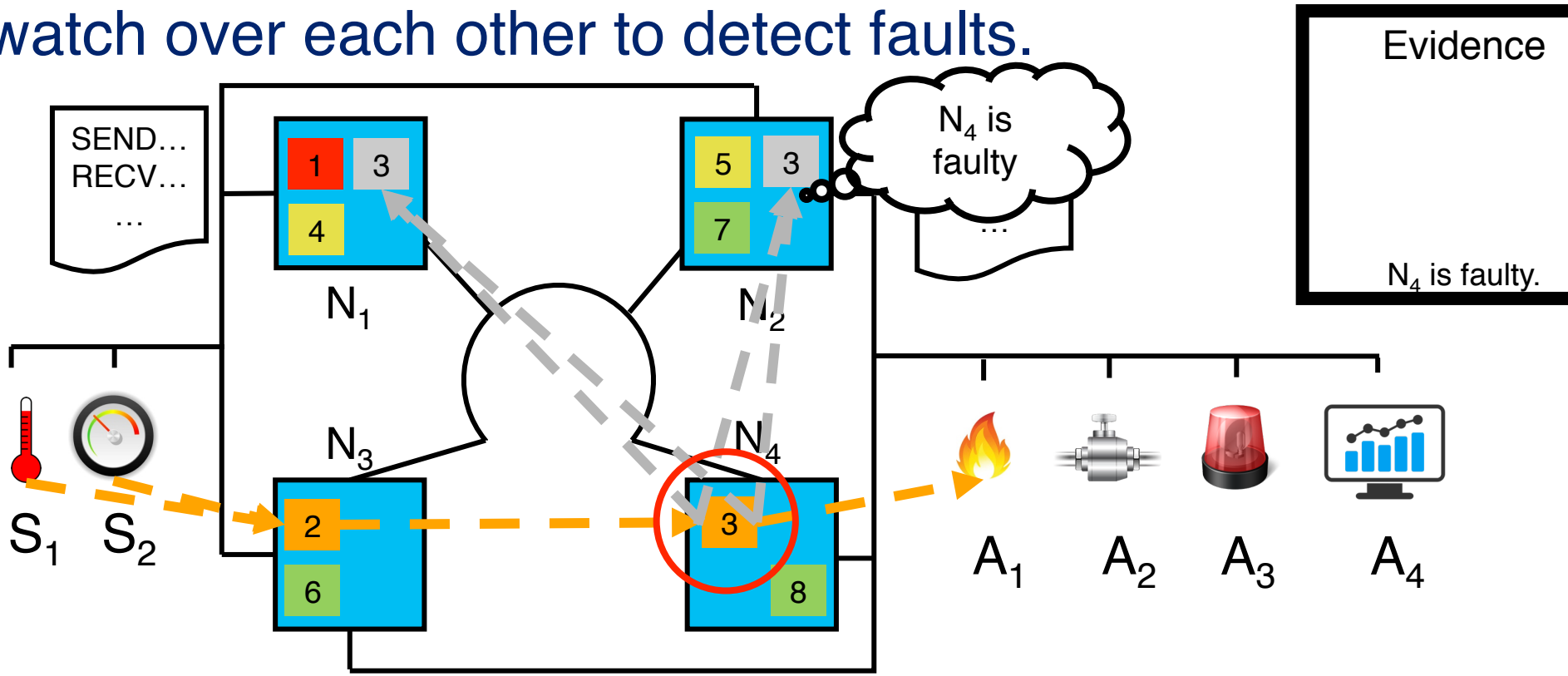
Can drop less critical tasks as necessary.



REBOUND

2. Detection

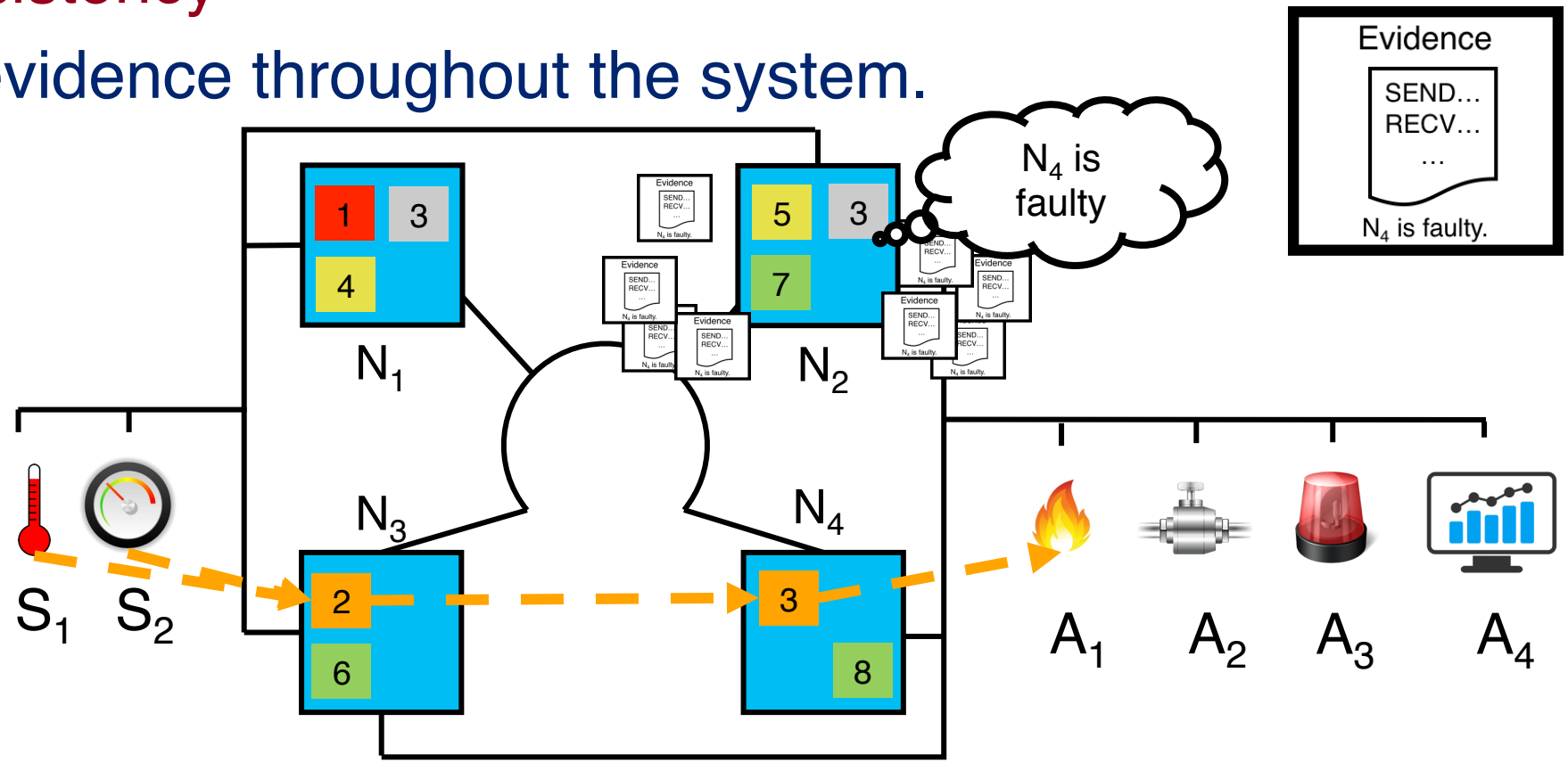
Nodes watch over each other to detect faults.



REBOUND

3. Consistency

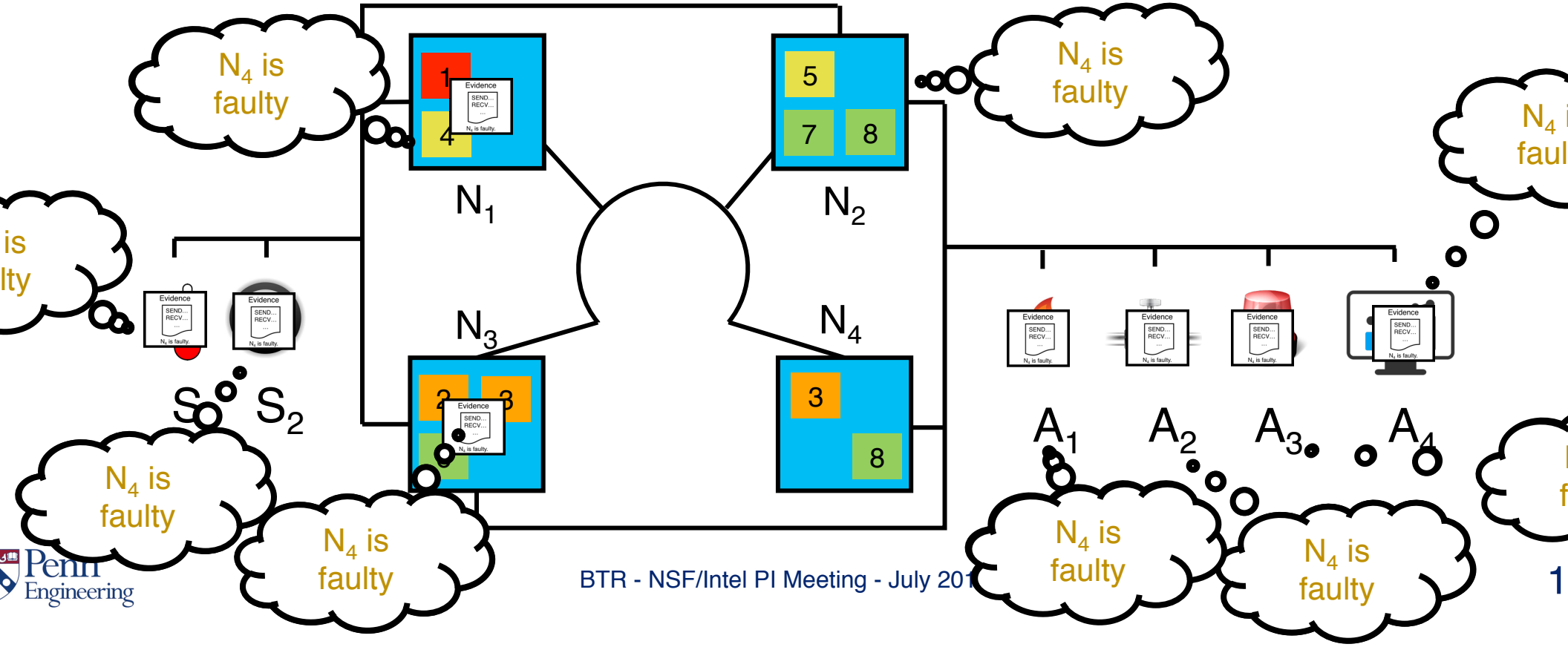
Flood evidence throughout the system.



REBOUND

4. Adaptation

Each node independently transitions to a new mode



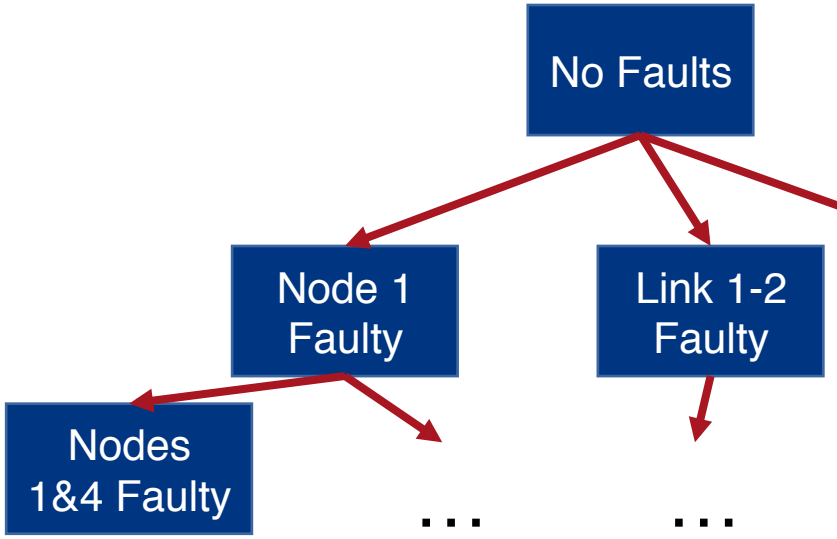
Outline

- Problem Introduction ✓
- Bounded Time Recovery ✓
- REBOUND ✓
- Technical Components
  1. Planning
  2. Detection
  3. Consistency
  4. Adaptation
- Results

1. Planning

For every* mode, we have a precomputed schedule and plan for every node.

- Schedule generated offline
- When tasks should run and where
- Many constraints
- Dependent scheduling problem
- Builds a tree



Can limit the number of faults to improve computation time.

2. Detection

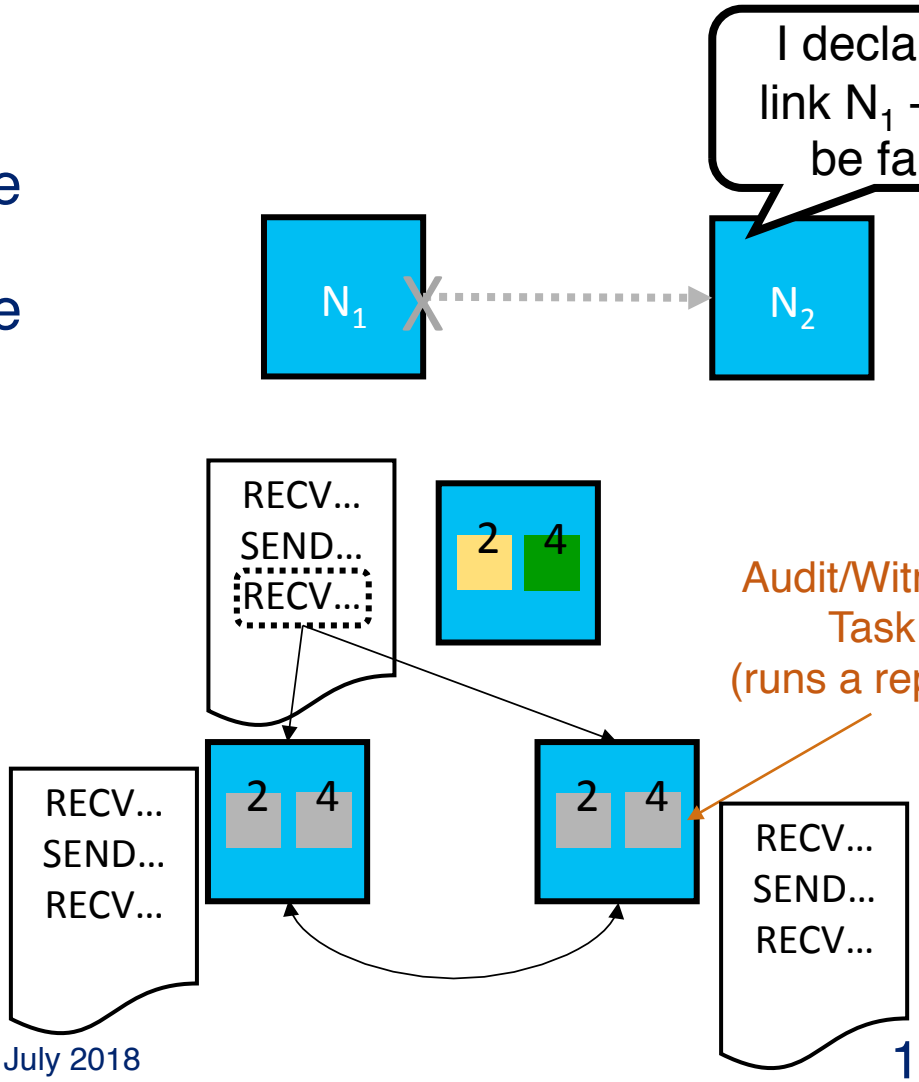
Omission Faults

- Declare link faulty if an expected message from a neighbor is not received
- Declaration causes other nodes to change mode.
- Leverage synchrony.

Commission Faults

- Witness/Audit Nodes and Replicas
  - If fault found, log is used as a proof of misbehavior.
- Large improvement over PeerReview
  - Adding synchrony

Challenge: Bounding Time of Detection



3. Consistency

We need a solution where…

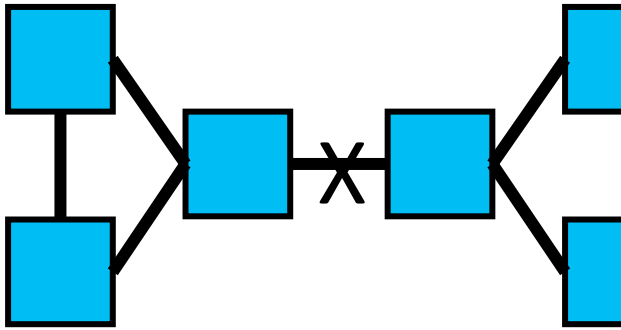
- Any two good nodes agree on the state of the system

or

- The two become aware they cannot communicate

Strawman: flood the system periodically with signed attestations of current mode

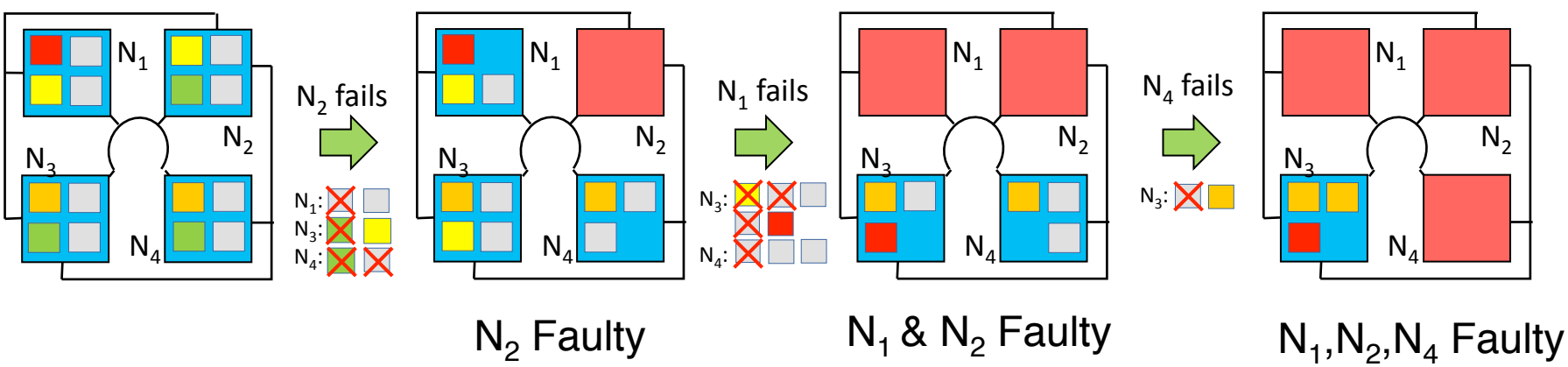
- Actual solution is more efficient



4. Adaptation

Each node individually transitions when its mode changes.

When evidence is received a mode change occurs within a bounded period of time.



Challenges

Bounding every step of the algorithms

Overhead of periodic flood

- Multisignatures → drastically reduce traffic

Handling equivocation

- Different nodes notifying of different faults to their neighbors

Proving everything

- Correctness
- Completeness
- Bounded detection
- Bounded stabilization

Planning

- Unique problem

Theorem 1 (Accuracy (nodes)) *If nodes N_1 and N_2 are co rect in round r, then $N_2 \notin \mathsf{KN}_{N_1,r}$.*

Theorem 2 (Accuracy (links)) *If nodes N_1, N_2, and N correct in round r and there is a direct link between N_2 N_3, then $\mathsf{KL}_r(N_1)$ will not include (N_2, N_3).*

Theorem 3 (Completeness) *Let $(p_1, p_2, \ldots, p_L)$ be a and let $i := p_x$ and $j := p_y$ be two nodes on the path that $x < y$ – that is, i is upstream of j. Suppose th*

...

Theorem 4 (Bounded detection time) *Suppose a fau a node i becomes observable in round r, and let $\max_E \max_{p \in \mathsf{PATH}(E)} |p|$ be the number of hops on the lon*

...

Theorem 5 (Consistency) *Let $D := \max_{i,j} D_{i,j}$ be the h est max-fail distance between any pair of nodes. If N_1 N_2 are correct and there exists a network path that*

...

Outline

- Problem Introduction ✓
- Bounded Time Recovery ✓
- REBOUND ✓
- Technical Components ✓
  1. Planning
  2. Detection
  3. Consistency
  4. Adaption
- Results

Overhead of Schedule Tree

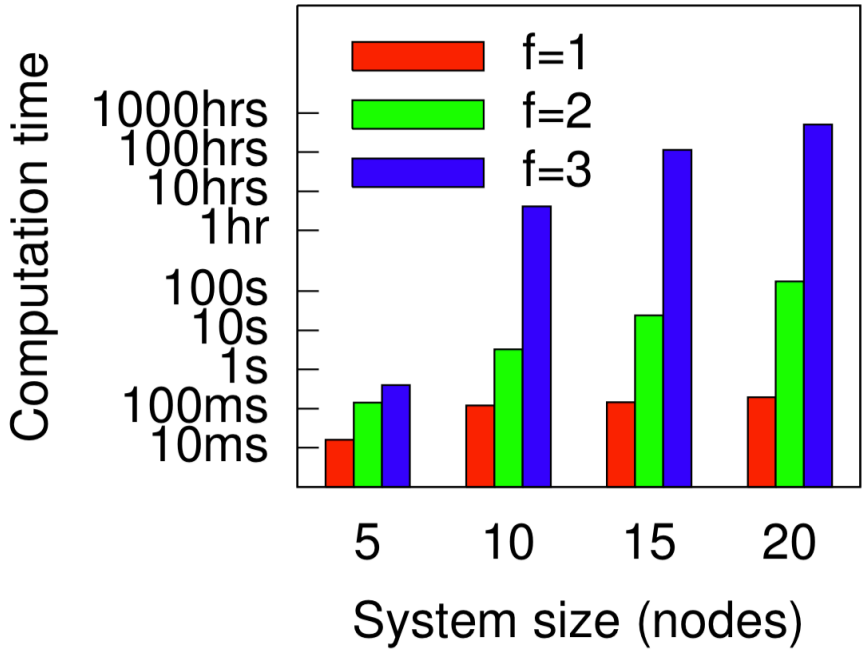
Time depends on:

- The number of nodes.
- Degree of network.
- Number of faulty nodes, f.

Only compute once for the lifetime of the system.

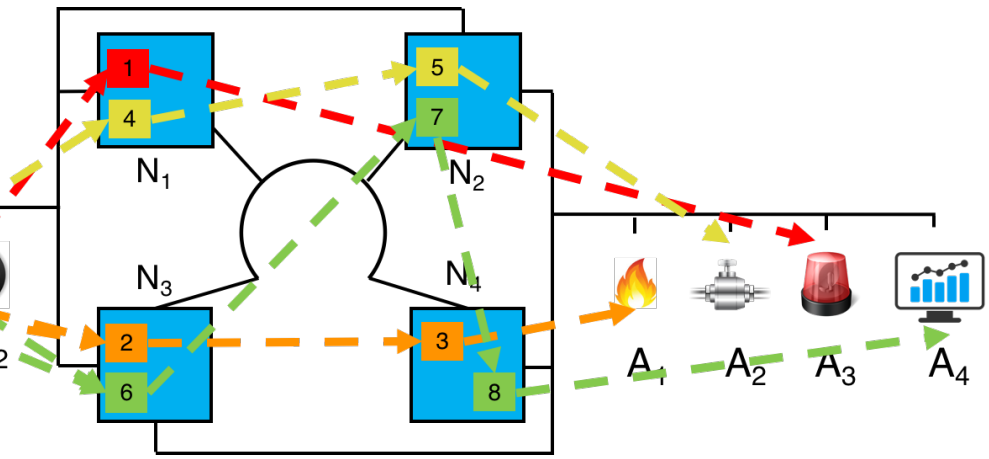
Subtrees easily parallelizable.

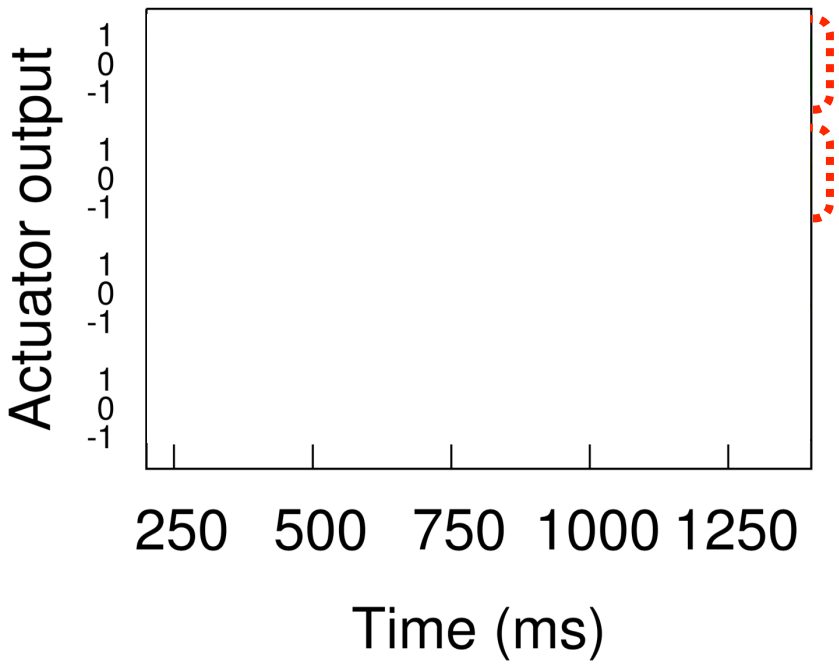
f = # of faulty nodes protected against



Recovery

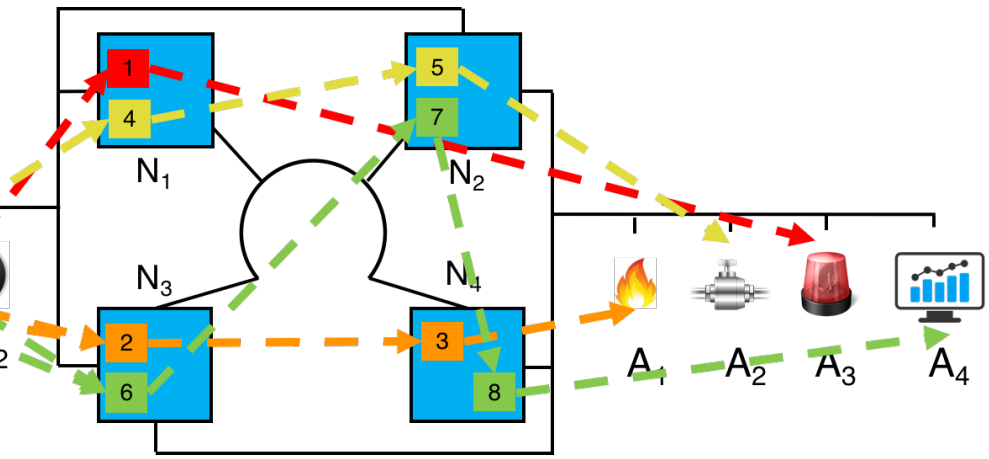
Unprotected System, N_2 Compromised

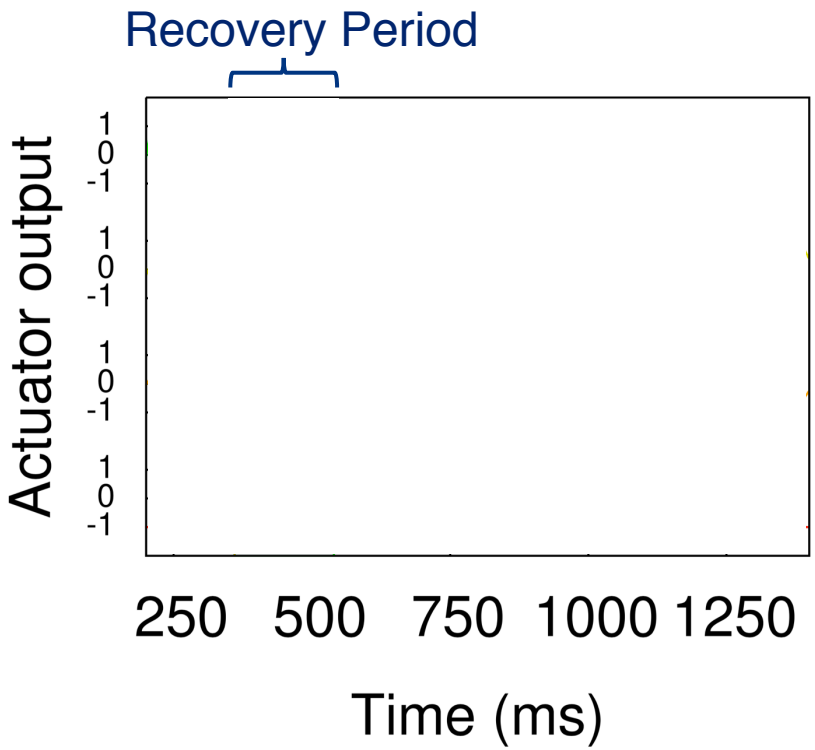




Recovery

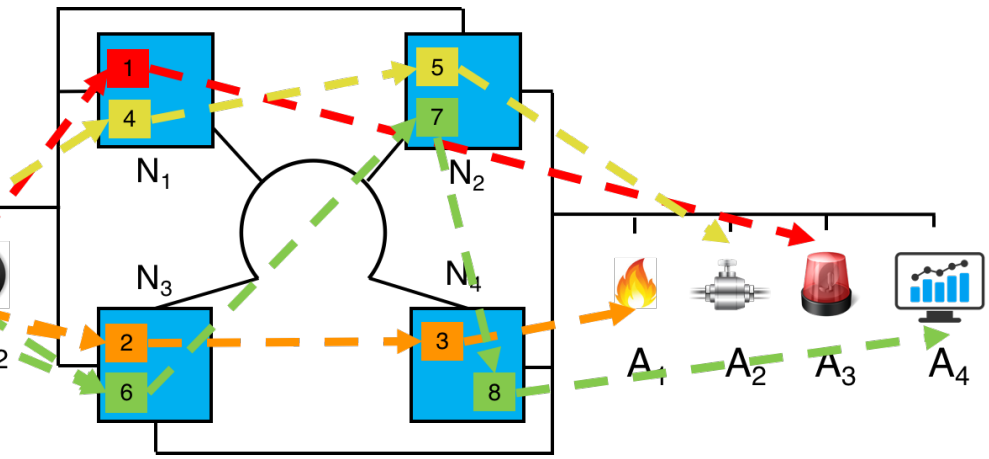
Protected System, N_2 Compromised

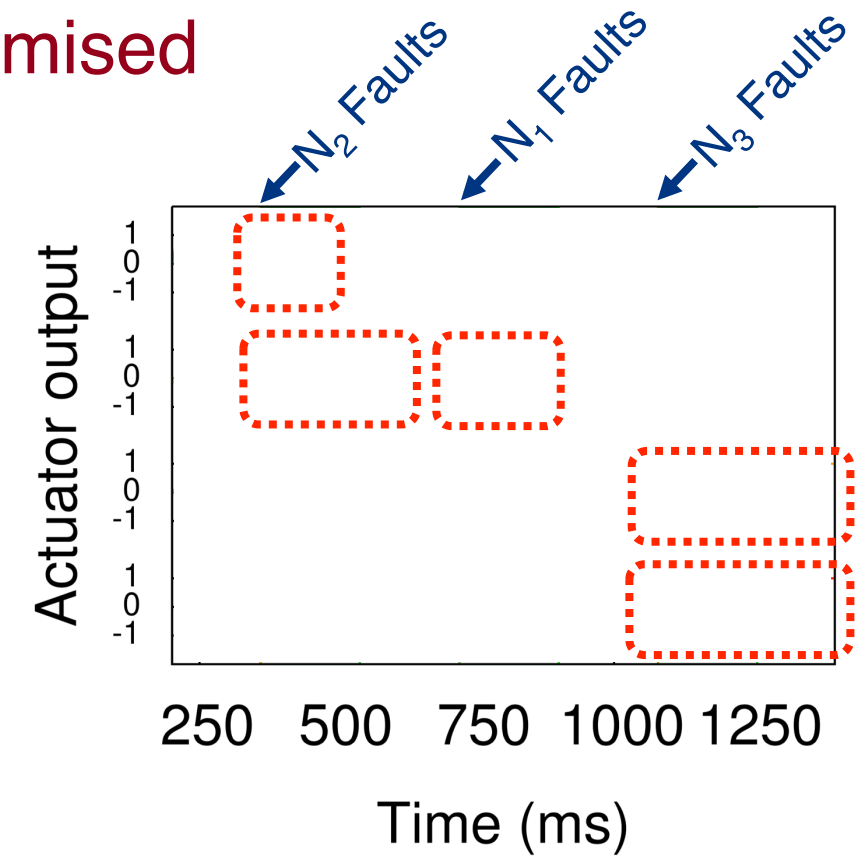




Recovery

Protected System, N_1, N_2, N_3 Compromised





y Idea: Period of Imperfection

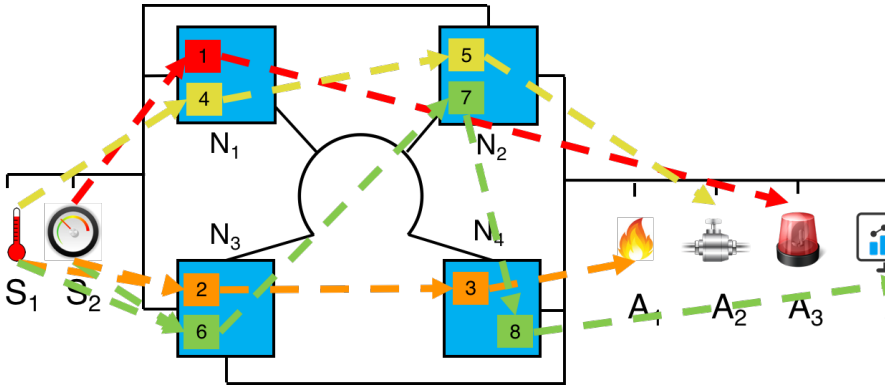
ny CPS can tolerate a short period of
ty behavior.

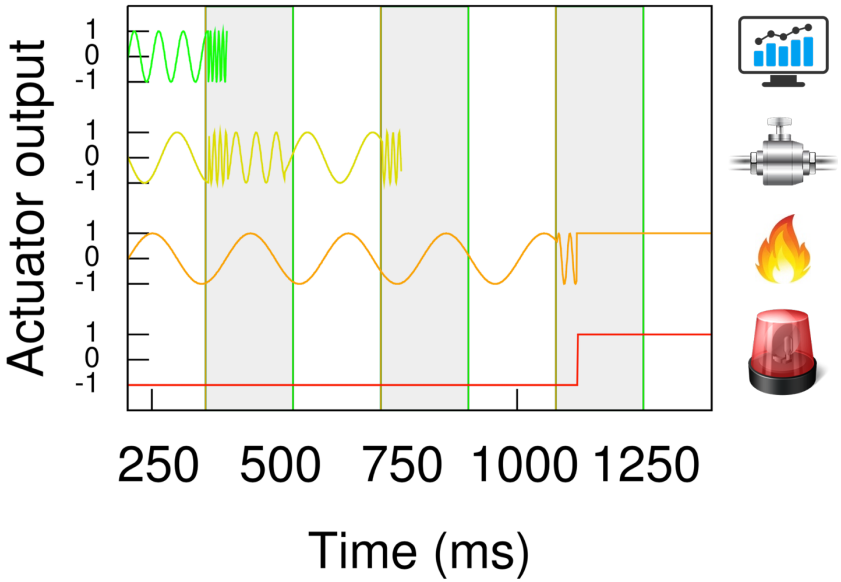
proach: Bounded Time Recovery

unded time recovery guarantees that the
tem quickly returns to correct behavior
r a fault.

ution: REBOUND

orithms and protocols to provide BTR
distributed systems.





Thank you.