Handling Failures in Cyber-Physical Systems: Potential Directions

Taylor Johnson and Sayan Mitra

Coordinated Science Laboratory
University of Illinois at Urbana-Champaign
Real-Time Systems Symposium (RTSS) 2009

December 1, 2009

Motivational example from distributed computing

Consensus (synchronous)

- Every process has an input and all non-faulty ones must decide on a common value in finite time

Motivational example from distributed computing

Consensus (synchronous)

- Every process has an input and all non-faulty ones must decide on a common value in finite time

in spite of failures	**processes (at least)**	**rounds**
f **crash failures**	$f+1$	$f+1$
t **Byzantine failures**	$3t+1$	$t+1$

Motivational example from distributed computing

Consensus (synchronous)

- Every process has an input and all non-faulty ones must decide on a common value in finite time

in spite of failures	**processes (at least)**	**rounds**
f **crash failures**	$f+1$	$f+1$
t **Byzantine failures**	$3t+1$	$t+1$

- Natural question: how many processes are required to tolerate both f crash failures and t Byzantine failures?

Motivational example from distributed computing

Consensus (synchronous)

- Every process has an input and all non-faulty ones must decide on a common value in finite time

in spite of failures	processes (at least)	rounds
f **crash failures**	$f+1$	$f+1$
t **Byzantine failures**	$3t+1$	$t+1$

- Natural question: how many processes are required to tolerate both f crash failures and t Byzantine failures?
- CPS can suffer the previous failures and *many more*!

Motivational example from distributed computing

Consensus (synchronous)

- Every process has an input and all non-faulty ones must decide on a common value in finite time

in spite of failures	processes (at least)	rounds
f **crash failures**	$f+1$	$f+1$
t **Byzantine failures**	$3t+1$	$t+1$

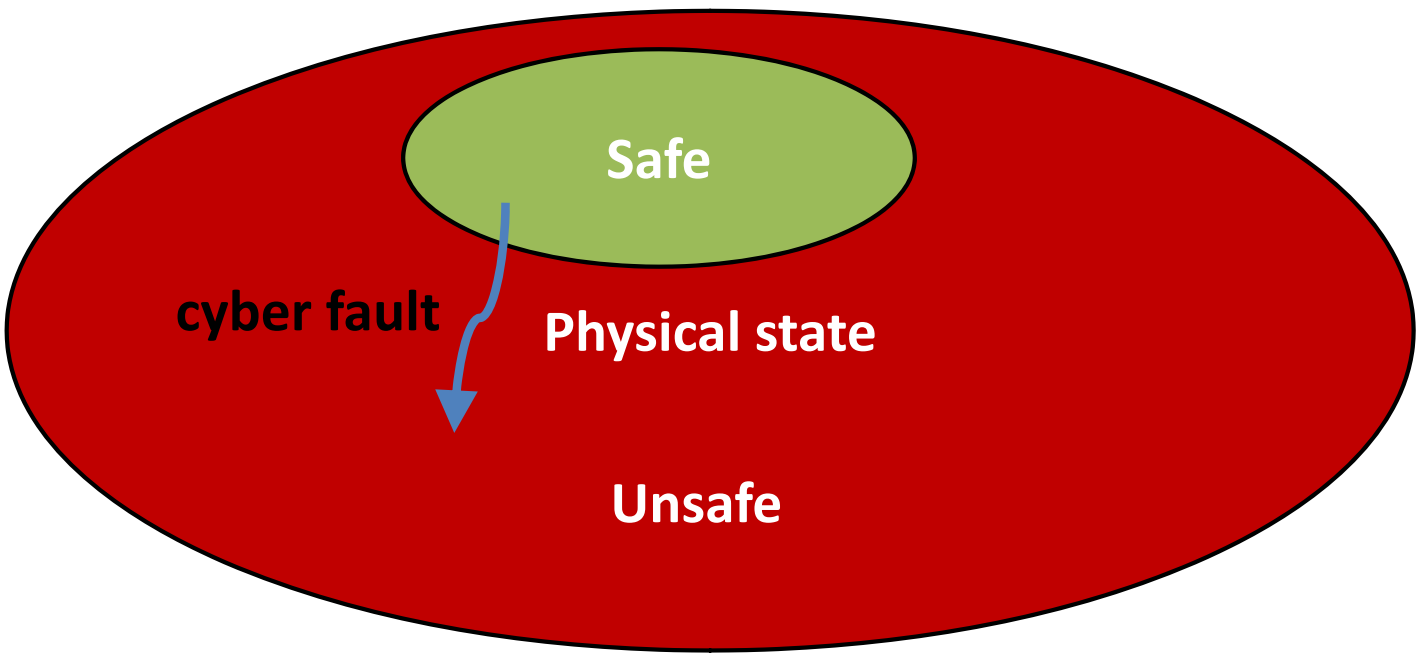
- Natural question: how many processes are required to tolerate both f crash failures and t Byzantine failures?
- CPS can suffer the previous failures and *many more*!

Interdisciplinary research problem

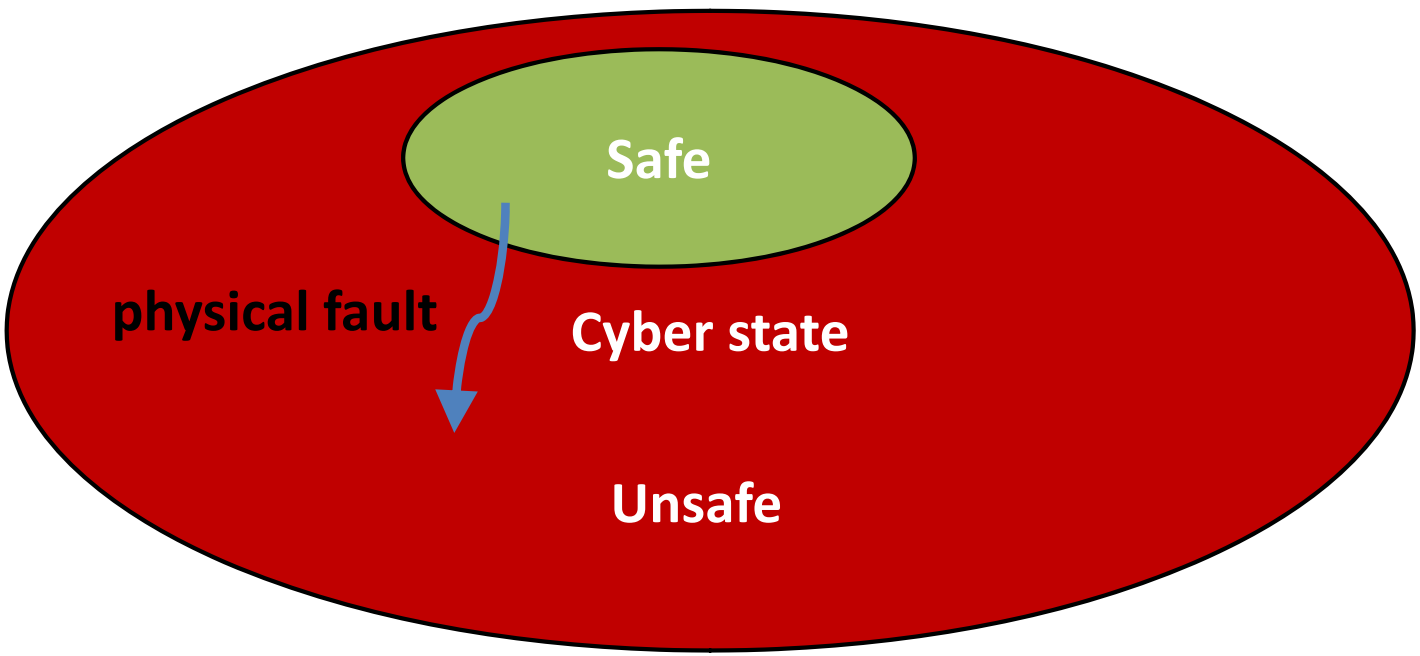
Develop failure detection and mitigation methods for cyber-physical systems

Outline

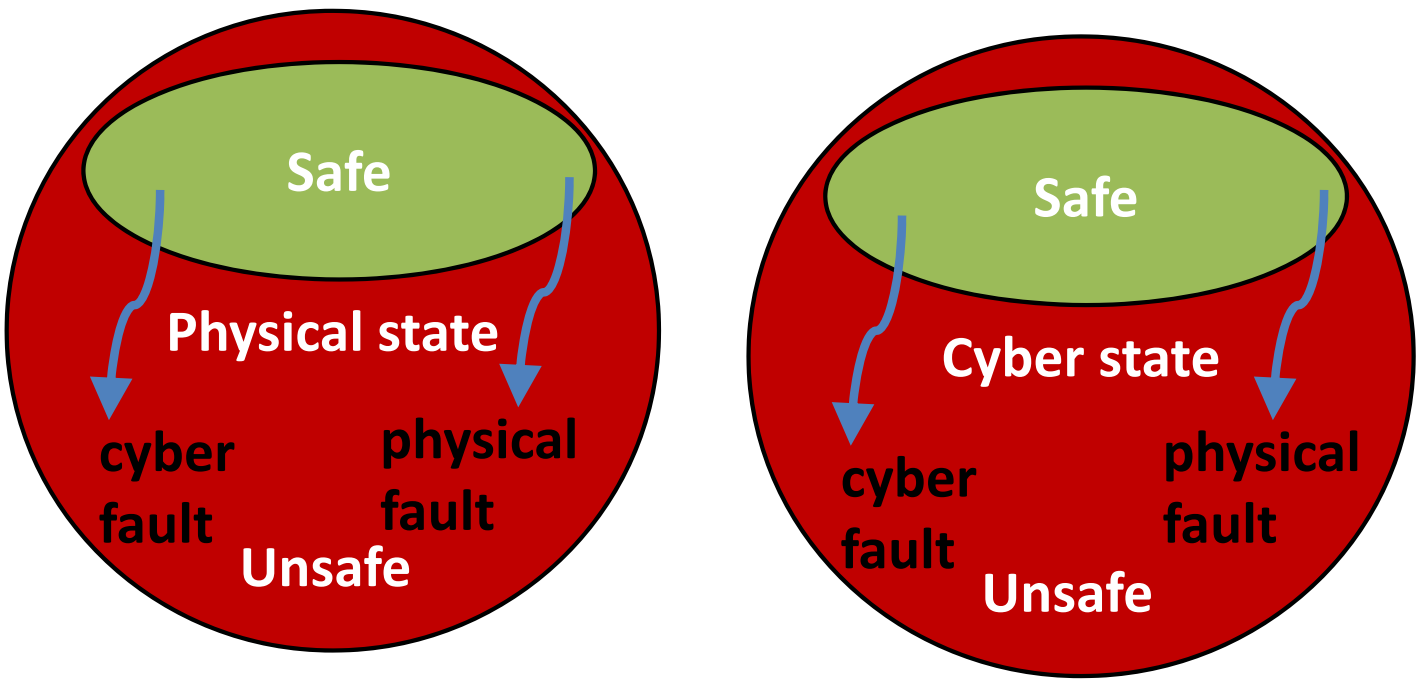
Cyber-physical fault interaction



Cyber-physical fault interaction



Cyber-physical fault interaction



Classes of failures

Cyber (software) failures

- Distributed computing: crash; Byzantine
- General: bugs
- Real-time systems: timing (missing deadlines)

Physical failures

- Sensor; actuator and control surface
- Robustness

Failures between cyber and physical

Communications

Occurrence

Single, permanent, transient, intermittent, or incessant

Prior work

Example solutions

- Simplex architecture
- Giotto
- Etherware

Prior work

Example solutions

- Simplex architecture
- Giotto
- Etherware

Common theme: solutions through abstraction!

Handling failures: active versus passive

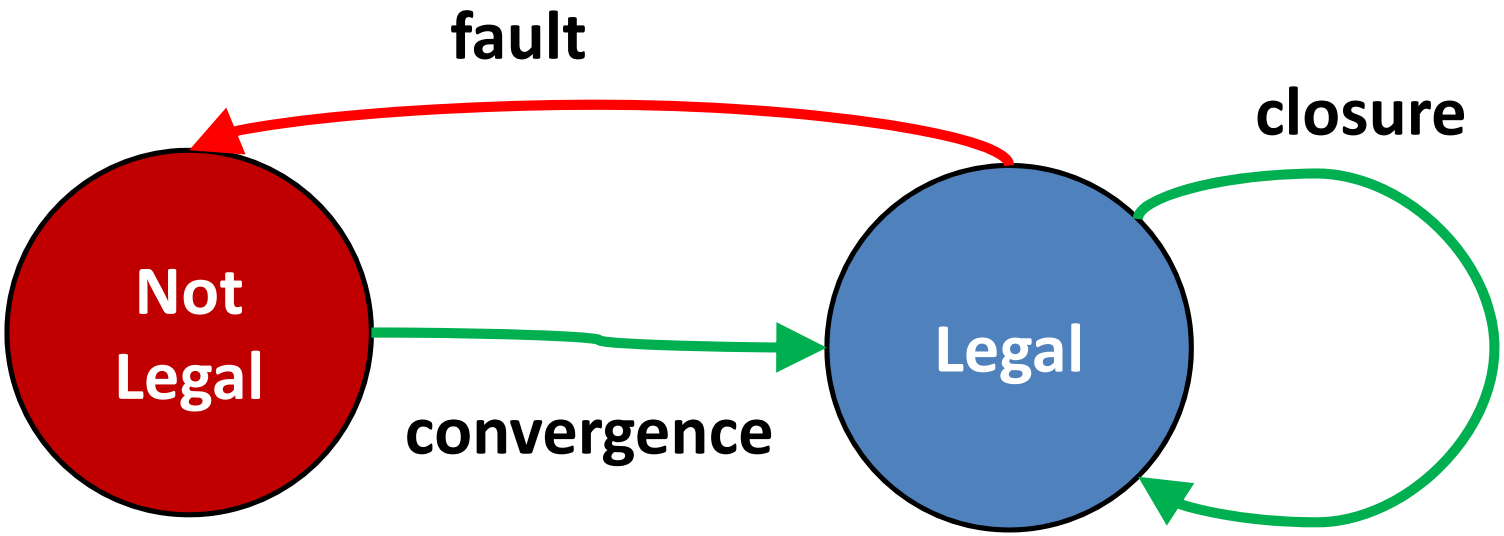
Active (non-masking)

- Failure detectors
- Reliable failure detectors from unreliable processes $\Rightarrow$ reliable systems from unreliable components (e.g., COTS, processes, stochastic processors, robustness, etc.)?
- Fault detection and isolation (FDI)

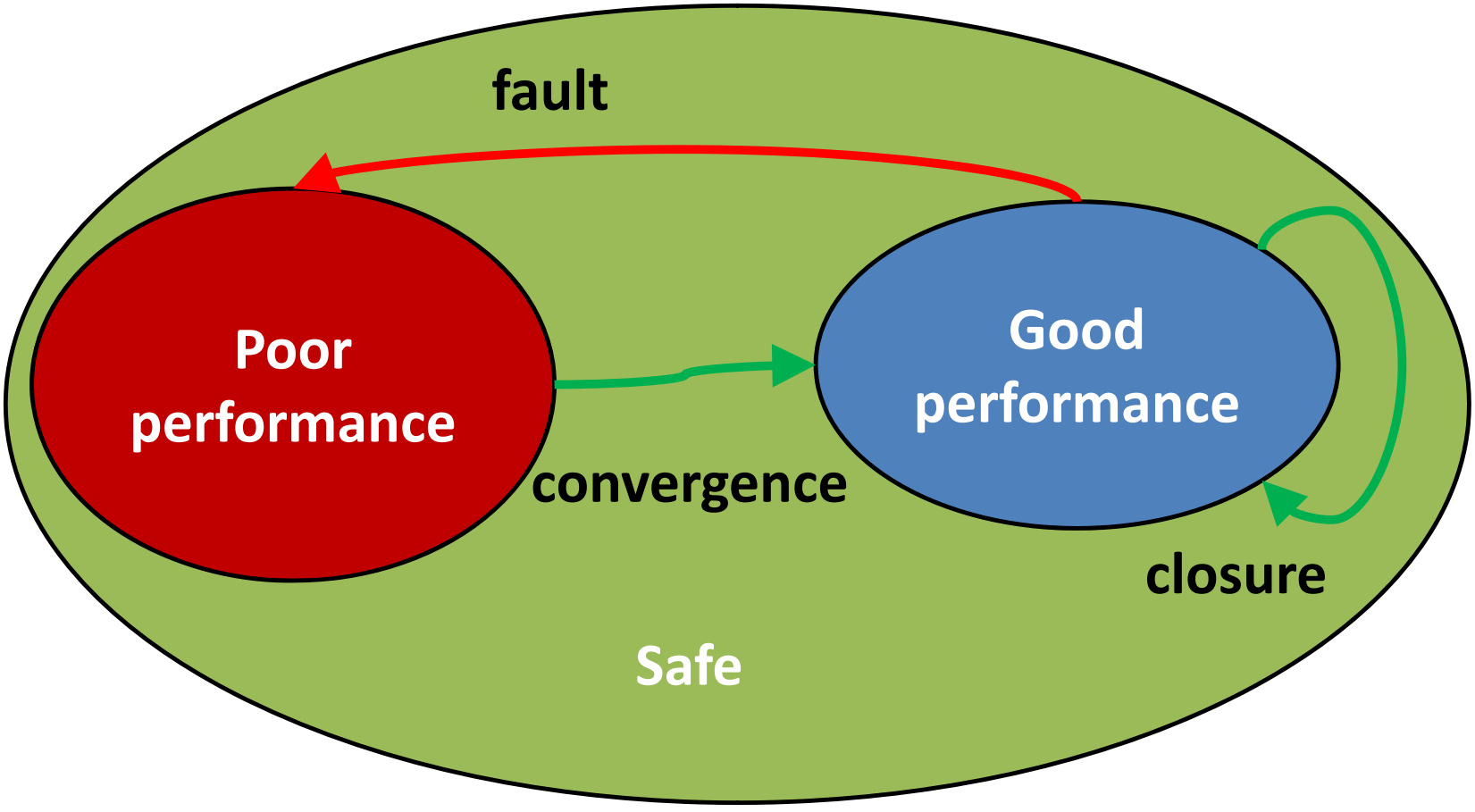
Passive (masking)

- Redundancy from the consensus example
- Self-stabilizing algorithms $\Rightarrow$ self-stabilizing systems?

Self-stabilizing algorithms



Self-stabilizing systems?



Formal methods and verification

Motivation

- Why formal methods?
- Provable guarantees
- Successfully applied in a variety of problems
- Maturing tools and formalisms

Useful concepts

- Abstraction
- Compositional reasoning
- Temporal logic and verification
- Actor model

Challenges and questions

- Model cyber and physical faults in such a way that they can be decoupled from one another, if possible
  - Must make any solutions compositional to avoid explosion of interaction cases
  - Complexity of analyzing all these fault sources simultaneously must be reduced: how does one fault influence another influence another is intractable
- Impossibility results
- Formal methods challenges ([Emerson, Clarke, and Sifakis, “Model checking: algorithmic verification and debugging”, Nov. 2009]): model checking for (a) software, (b) real-time systems, (c) hybrid systems, (d) probabilistic systems, and compositional model checking
- Lots of work to be done, but many interesting directions!

Thank you and questions

Questions

Hopefully there are lots of questions to motivate the discussion!