Building Fault-Tolerant Distributed Real-Time Systems

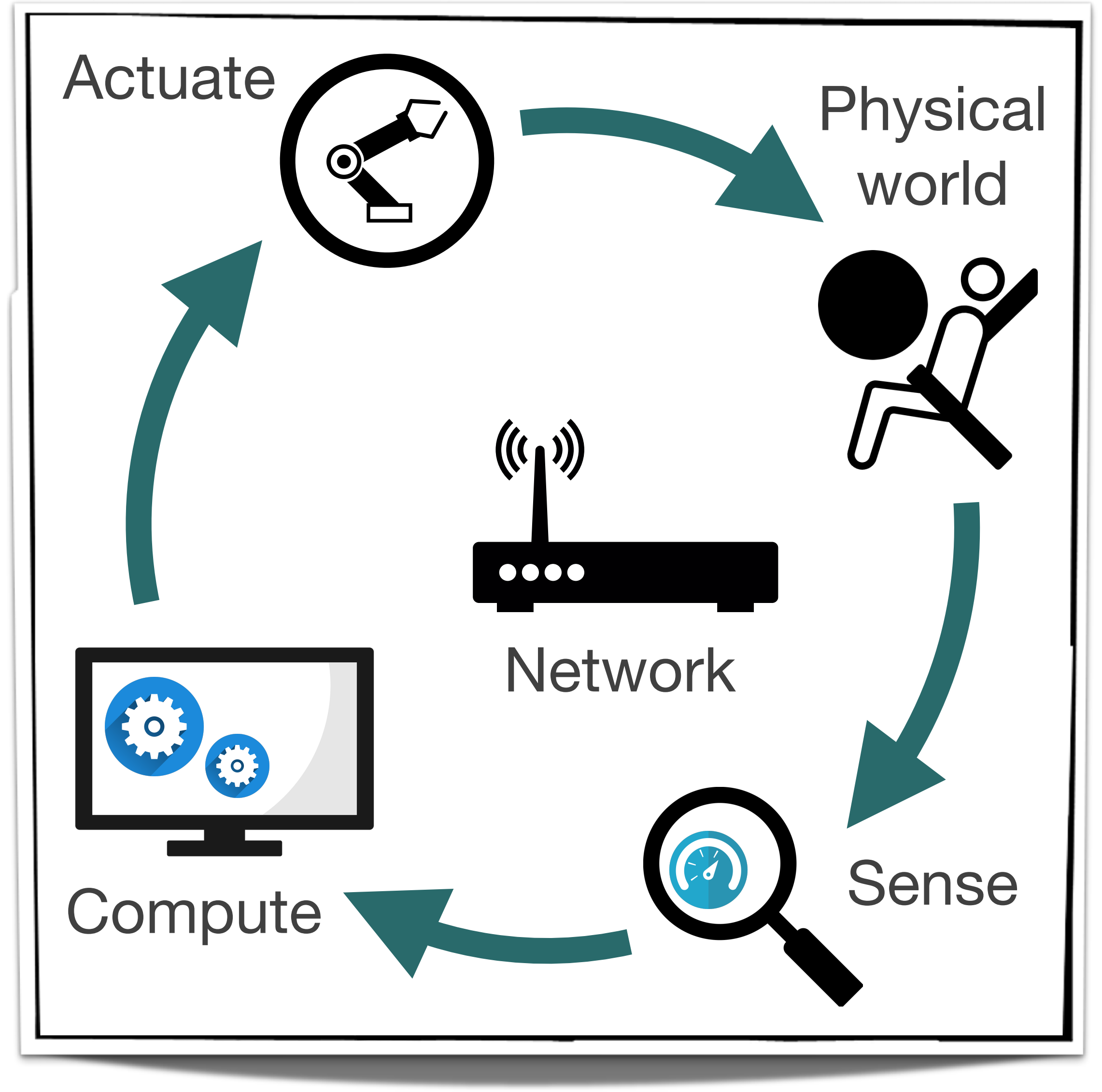
Arpan Gujarati | University of British Columbia, Vancouver (Canada)

Cyber-Physical Systems (CPS)

Tight and seamless integration

- Computation
- Networking
- Actuation and control
- Sensing of the physical world

Feedback control loops

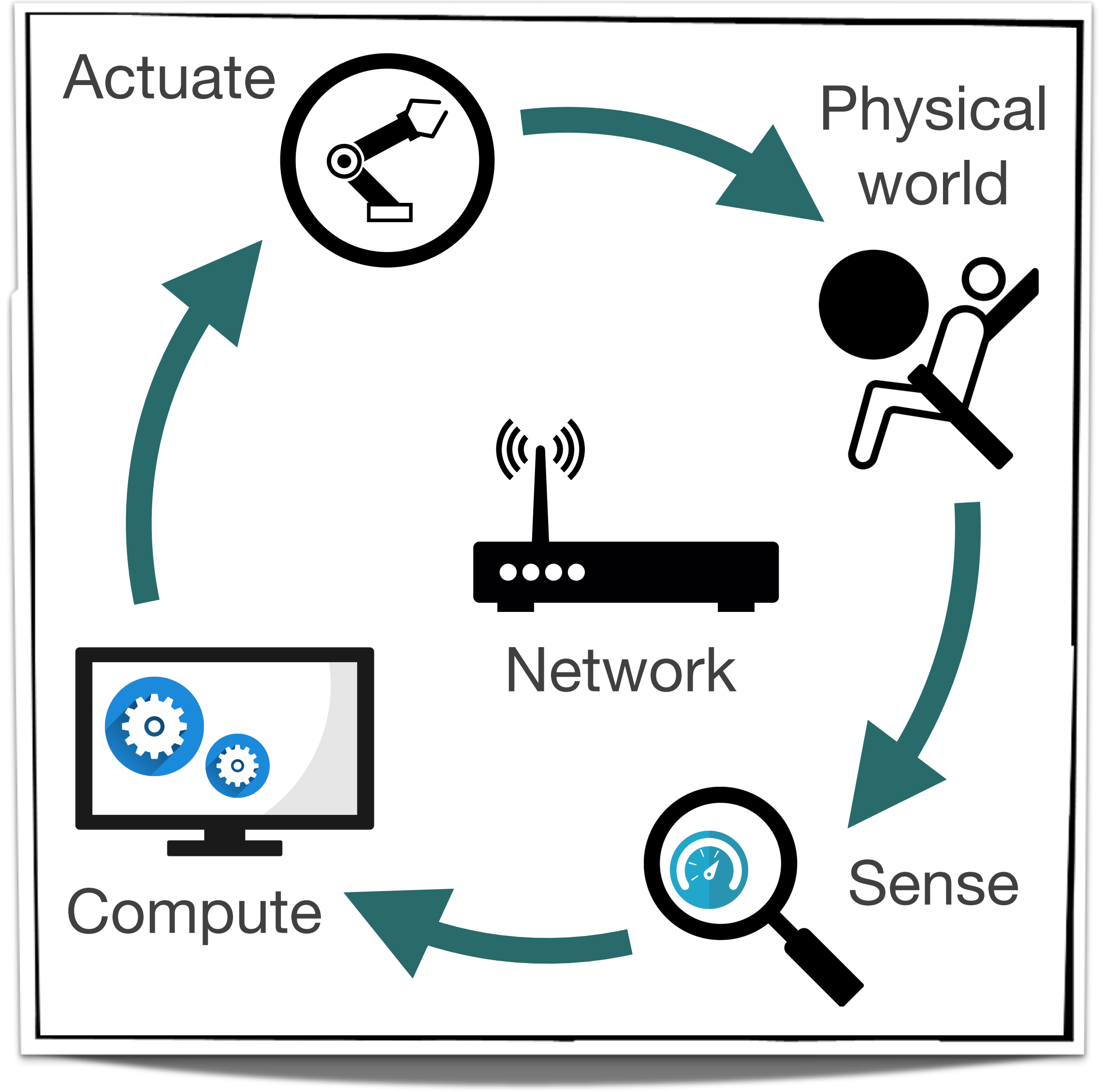


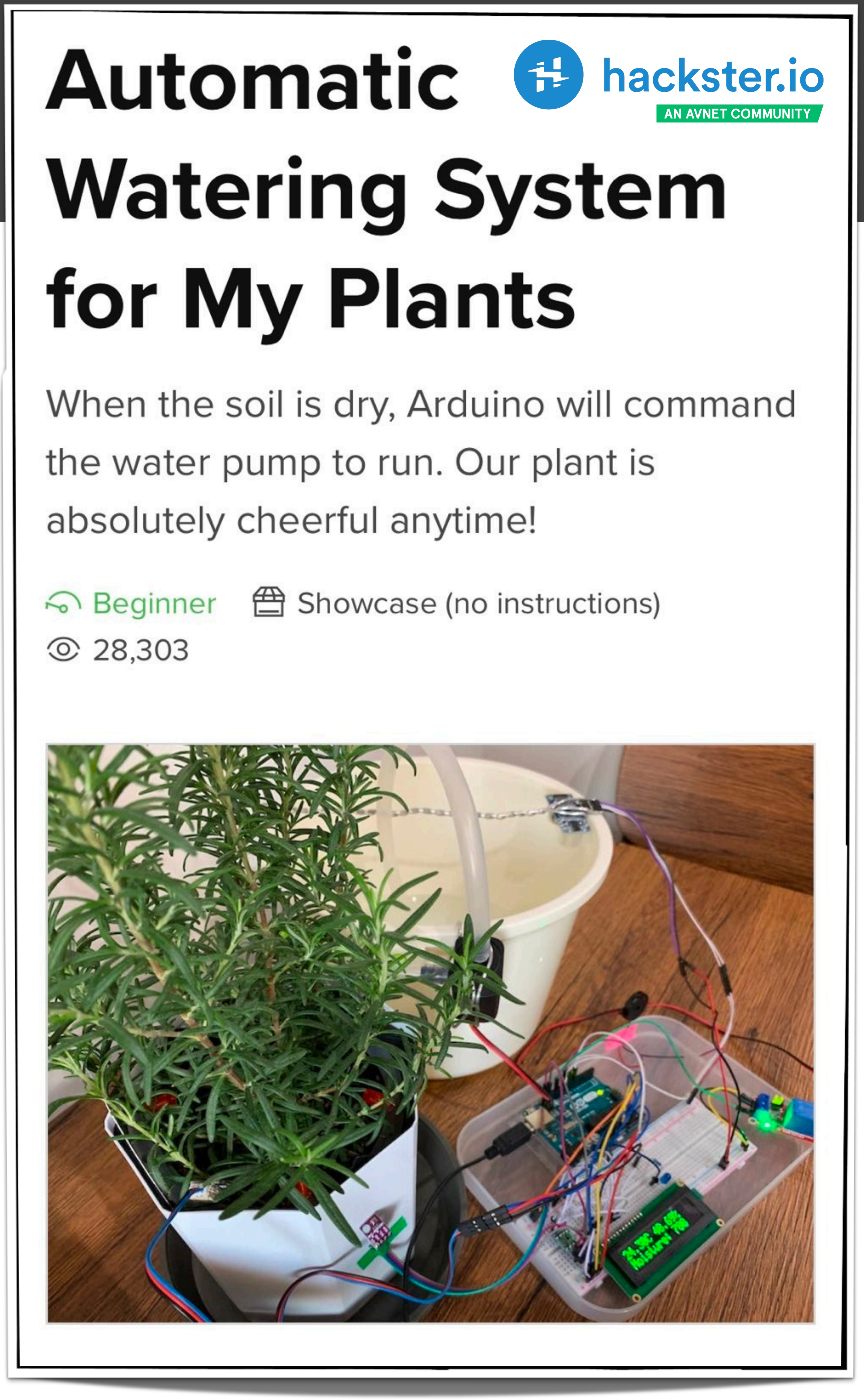
Cyber-Physical Systems (CPS)

Tight and seamless integration

- Computation
- Networking
- Actuation and control
- Sensing of the physical world

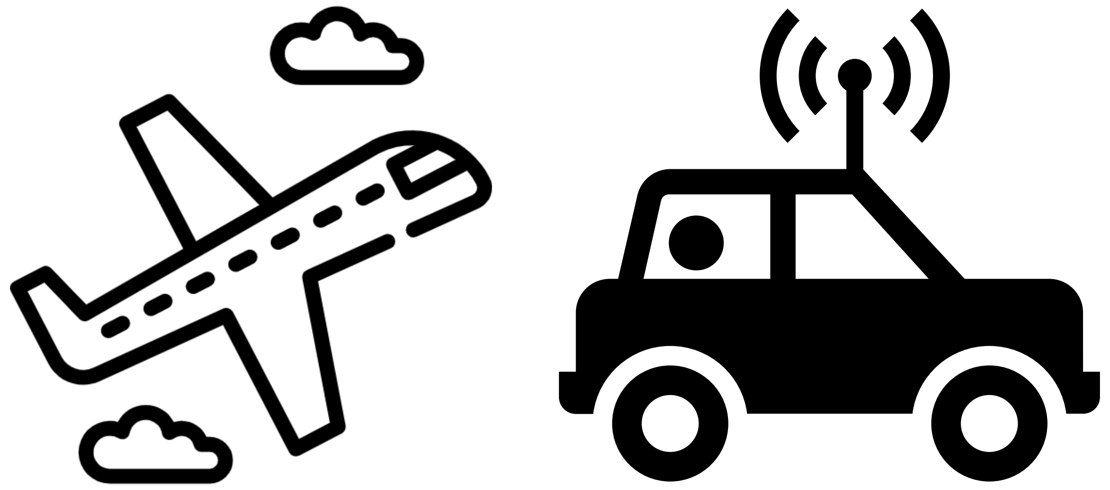
Feedback control loops





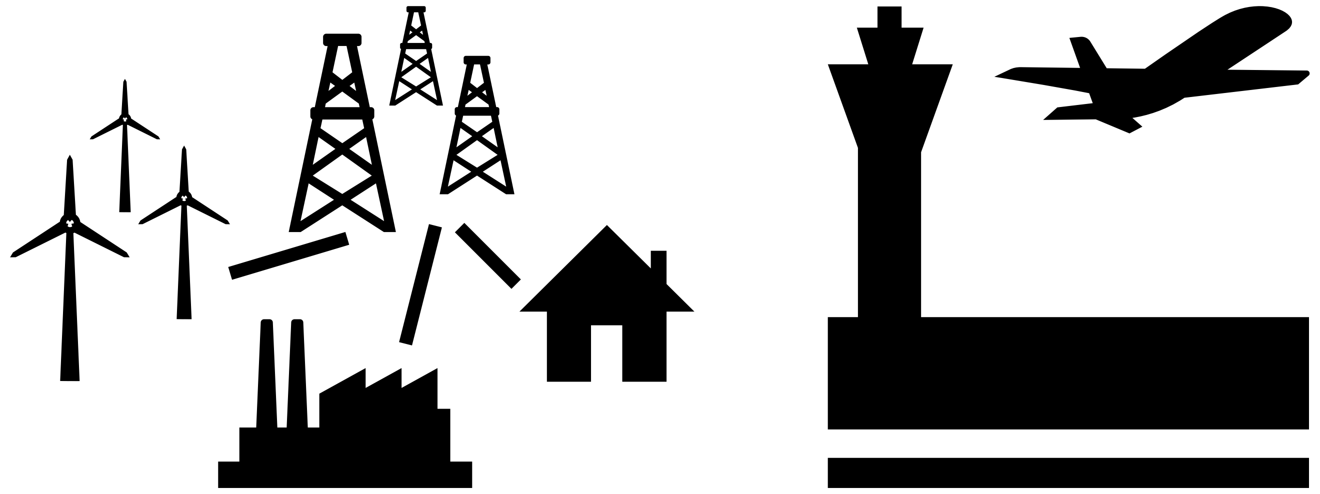
CPS are Ubiquitous, Diverse, and Safety-Critical

Large-scale systems



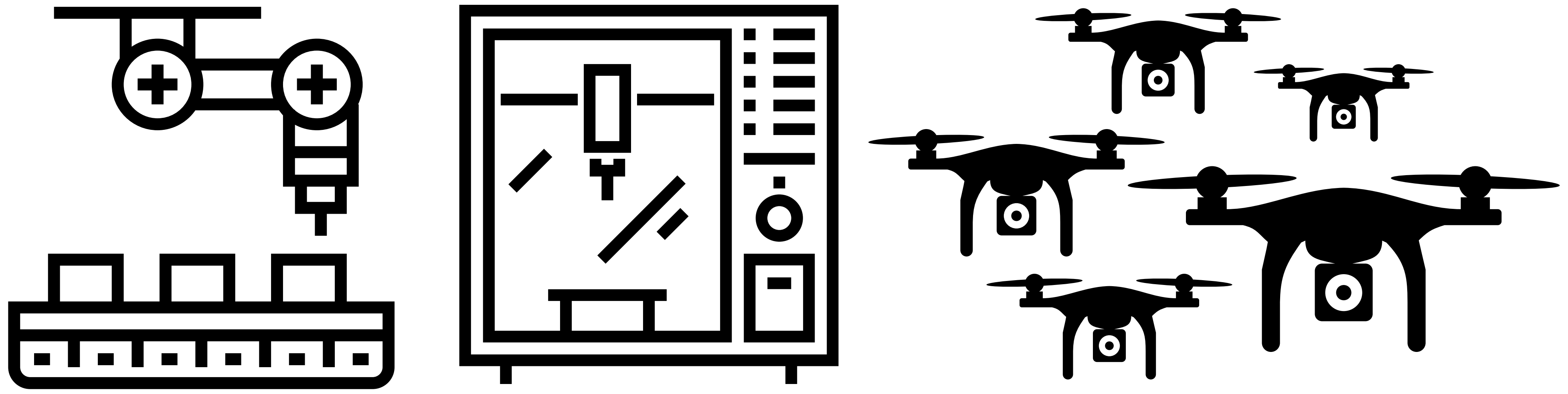
Airplanes Automobiles

Integration of diverse systems



Smart power grids Air traffic control

Small-scale systems



Robotic arms 3D printer Drone fleets

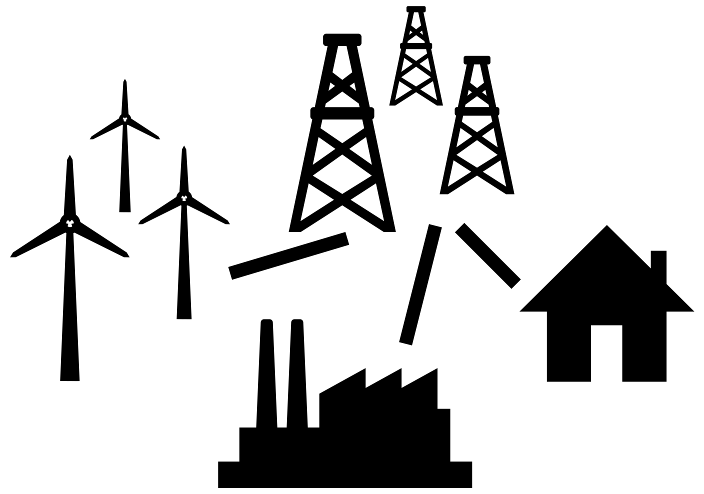
CPS are Ubiquitous, Diverse, and Safety-Critical

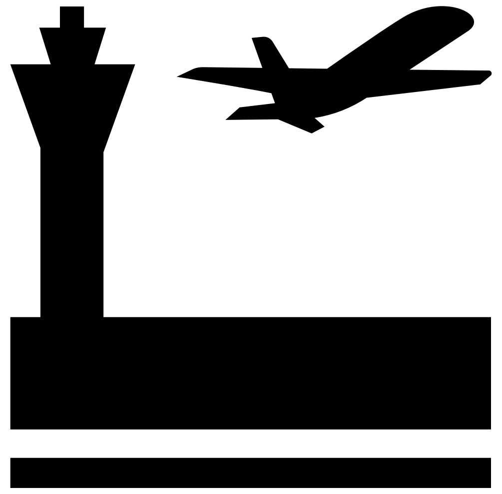
Large-scale systems


Airplanes

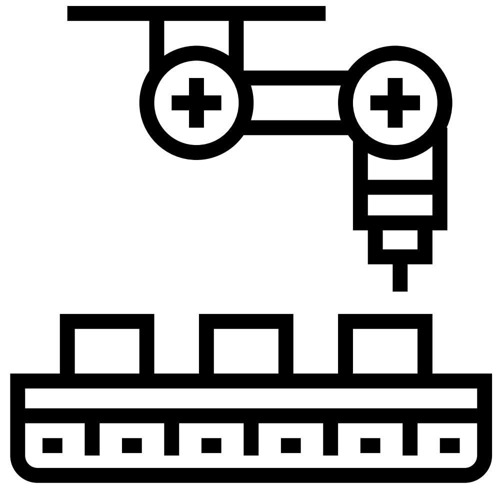

Automobiles

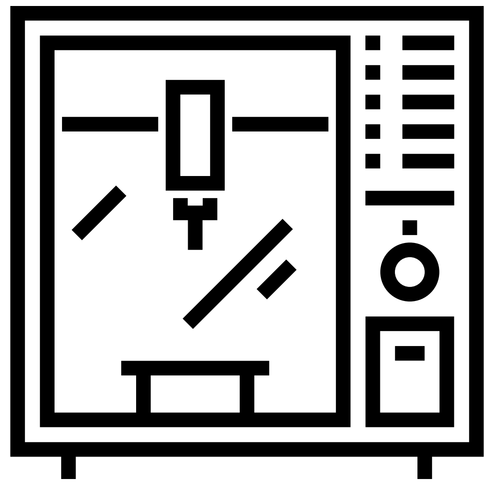
Integration of diverse systems


Smart power grids


Air traffic control

Small-scale systems


Robotic arms


3D printer


Drone fleets

Failures can be catastrophic!

- Severe **damage to property**
- **Death or serious injury** to humans

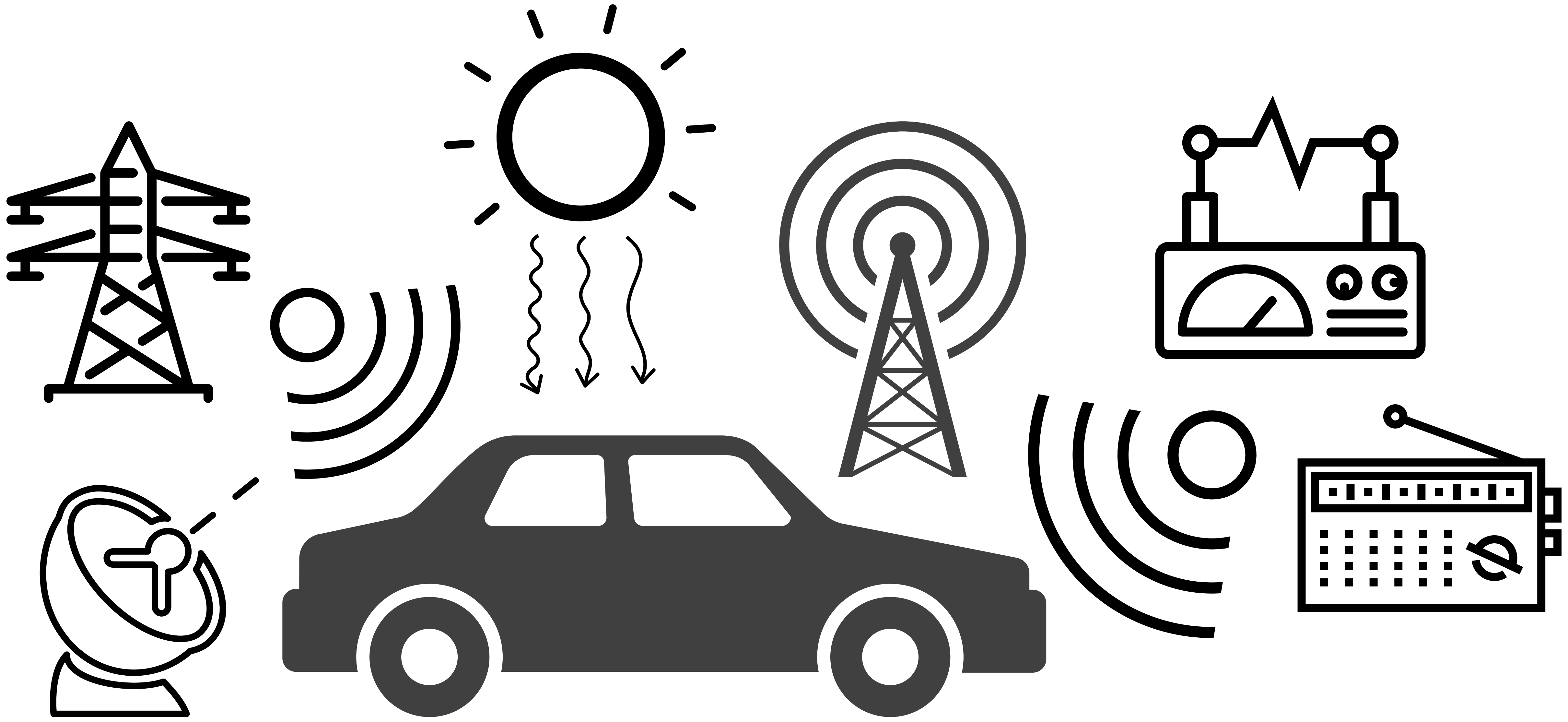
CPS are Susceptible to Transient Faults

Harsh environments

- Motors, spark plugs
- High power machinery, hard radiation
- Electromagnetic interference

Transient faults or soft errors

- Bit flips in registers, buffers, networks



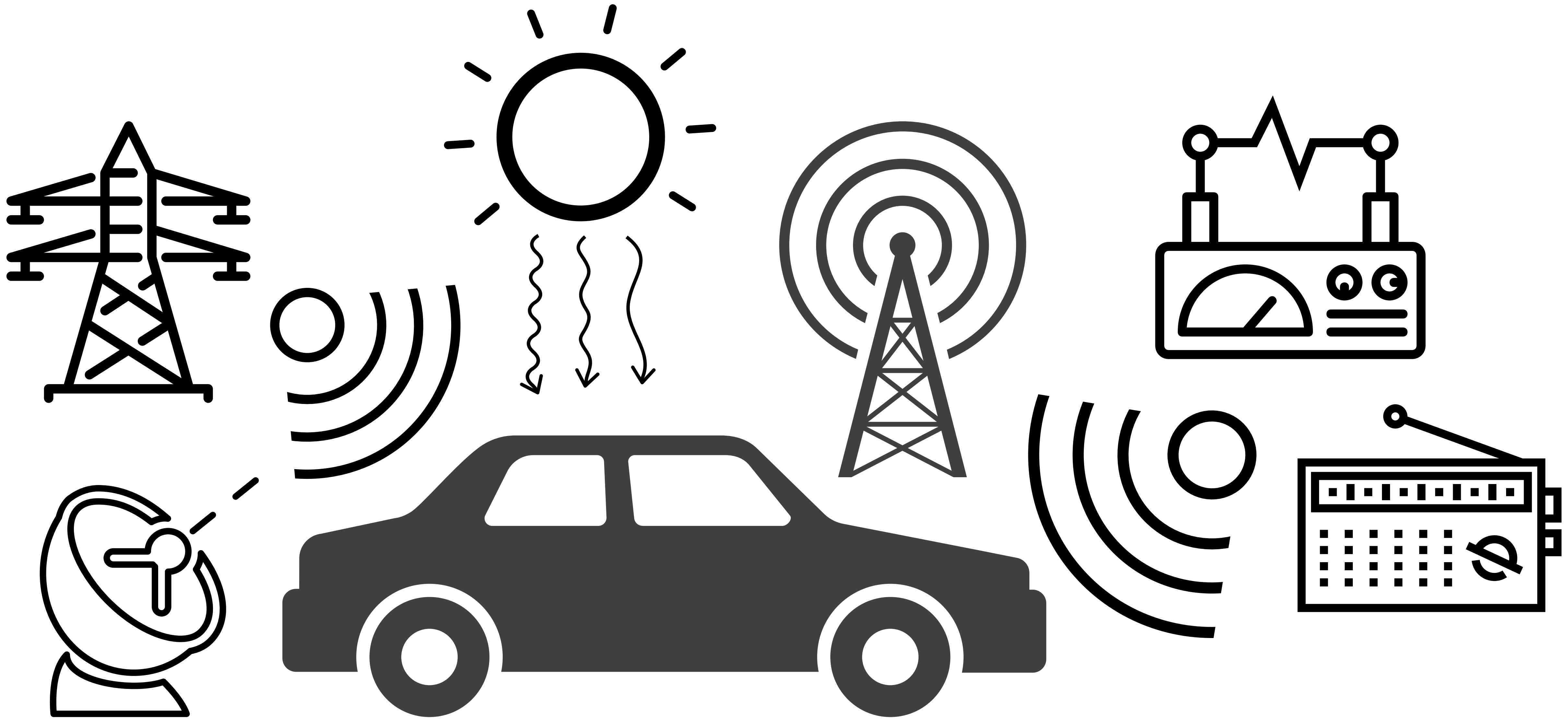
CPS are Susceptible to Transient Faults

Harsh environments

- Motors, spark plugs
- High power machinery, hard radiation
- Electromagnetic interference

Transient faults or soft errors

- Bit flips in registers, buffers, networks



*"About **5000 vehicles per day** will be affected by a soft error, with potentially catastrophic consequences." **

* Mancuso. "Next-Generation Safety-Critical Systems on Multi-Core Platforms." PhD Thesis, UIUC (2017)

Transient Faults can Lead to Complex Errors

Transmission: Faults in the network

Omission: Fault-induced kernel panics, hangs

Incorrect computation: Faults in memory buffers

Byzantine: Inconsistent broadcasts in distributed systems

- Environmentally-induced non-malicious Byzantine errors

Transient Faults can Lead to Complex Errors

Transmission: Faults in the network

Omission: Fault-induced kernel panics, hangs

Incorrect computation: Faults in memory buffers

Byzantine: Inconsistent broadcasts in distributed systems

- Environmentally-induced non-malicious Byzantine errors

Honeywell

Driscoll *et al.* Byzantine Fault Tolerance, from Theory to Reality. SAFECOMP (2003)

7 Conclusions

Byzantine Problems are real. The probability of their occurrence is much higher than most practitioners believe. The myth that Byzantine faults are only isolated transients is contradicted by real experience. Their propensity for escaping normal fault containment zones can make each Byzantine fault a threat to whole system dependability.

Transient Faults can Lead to Complex Errors

Transmission: Faults in the network

Omission: Fault-induced kernel panics, hangs

Incorrect computation: Faults in memory buffers

Byzantine: Inconsistent broadcasts in distributed systems

- Environmentally-induced non-malicious Byzantine errors

For high reliability targets

- E.g., $P_{fail} < 10^{-10}$/hr
- Every type of error must be handled

Honeywell

Driscoll *et al.* Byzantine Fault Tolerance, from Theory to Reality. SAFECOMP (2003)

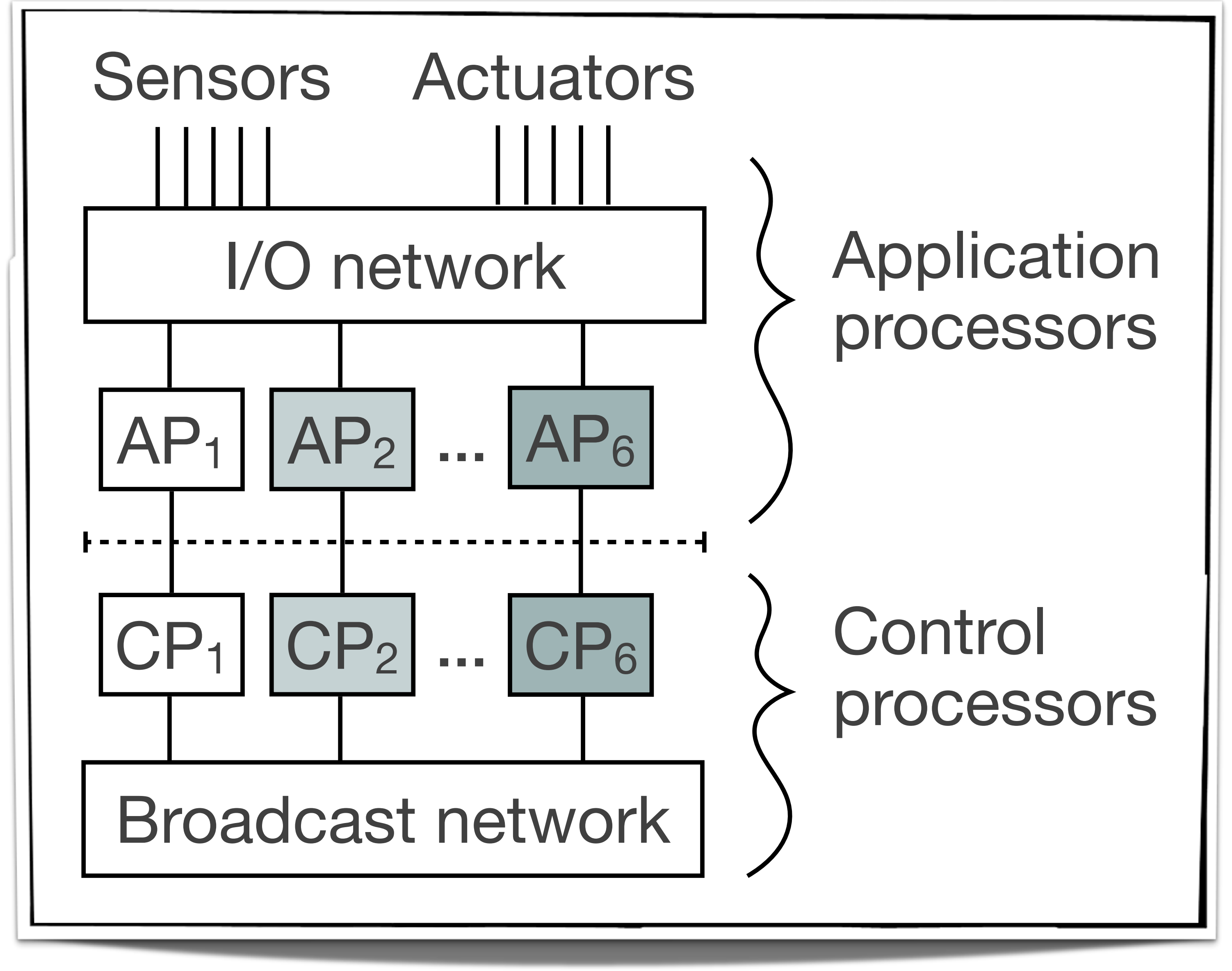
7 Conclusions

Byzantine Problems are real. The probability of their occurrence is much higher than most practitioners believe. The myth that Byzantine faults are only isolated transients is contradicted by real experience. Their propensity for escaping normal fault containment zones can make each Byzantine fault a threat to whole system dependability.

Example: Dependable CPS for Airplanes

Expensive custom-made fault-tolerant architectures

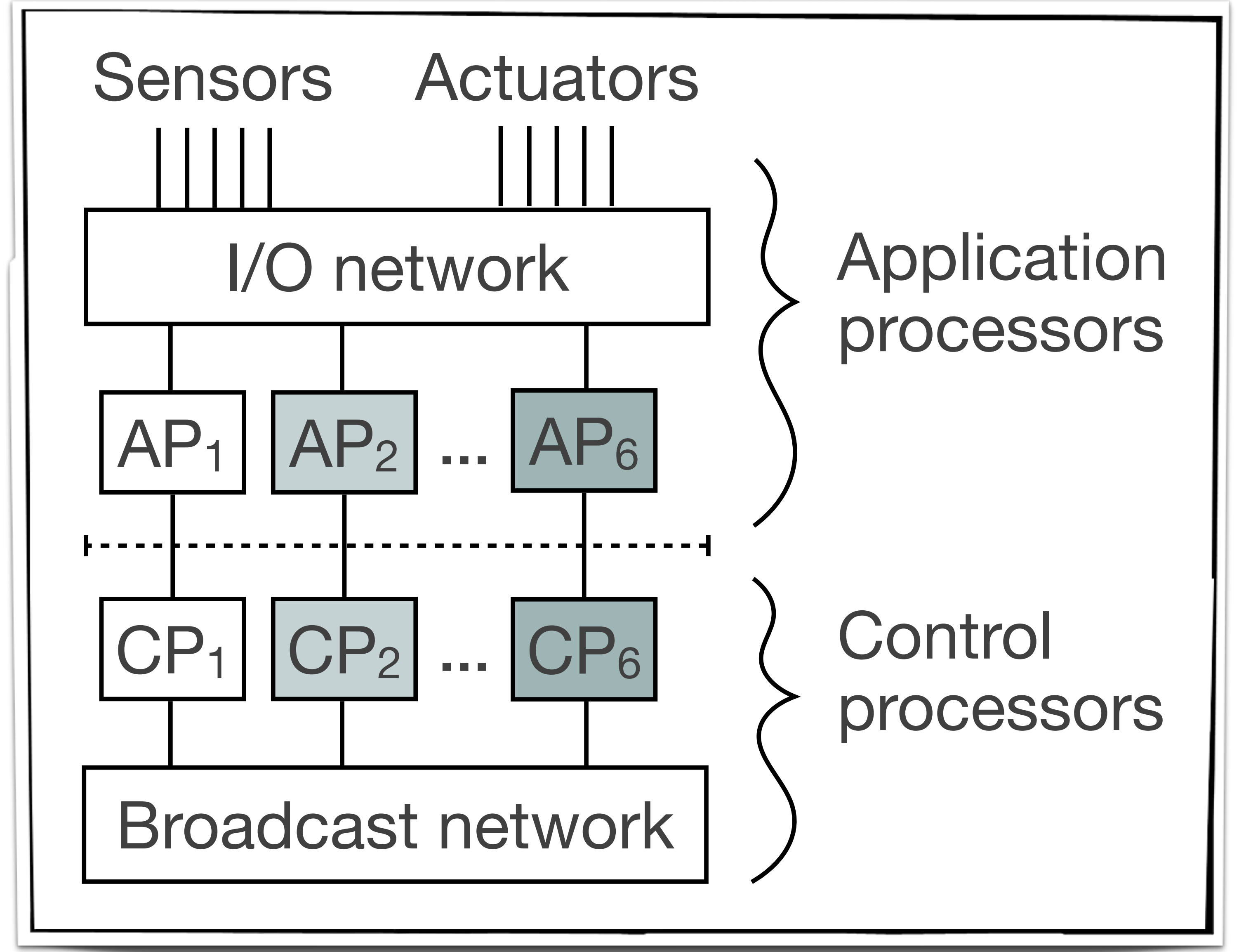
- Classical example: “The MAFT Architecture for Distributed Fault Tolerance” by Kieckhafer *et al.* (1988)



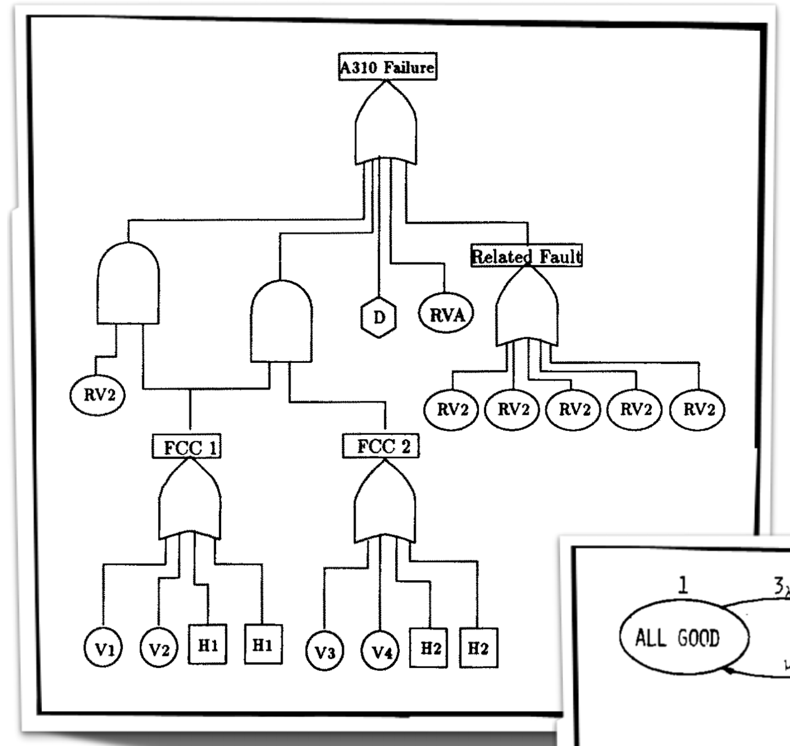
Example: Dependable CPS for Airplanes

Expensive custom-made fault-tolerant architectures

- Classical example: "The MAFT Architecture for Distributed Fault Tolerance" by Kieckhafer *et al.* (1988)

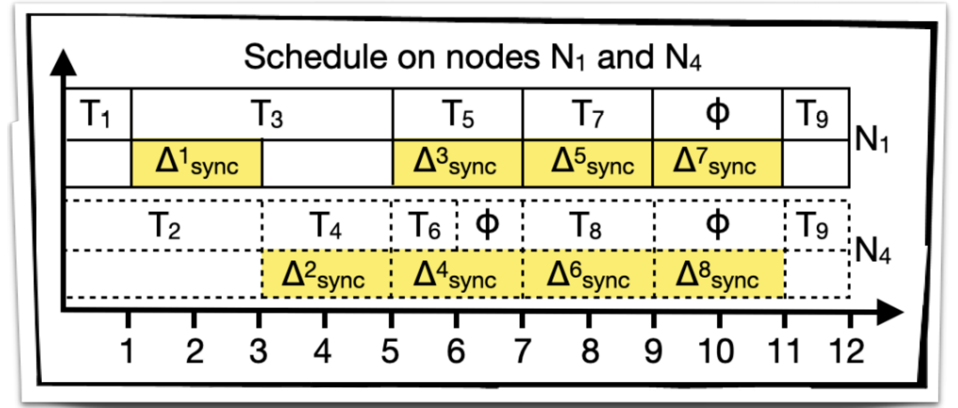


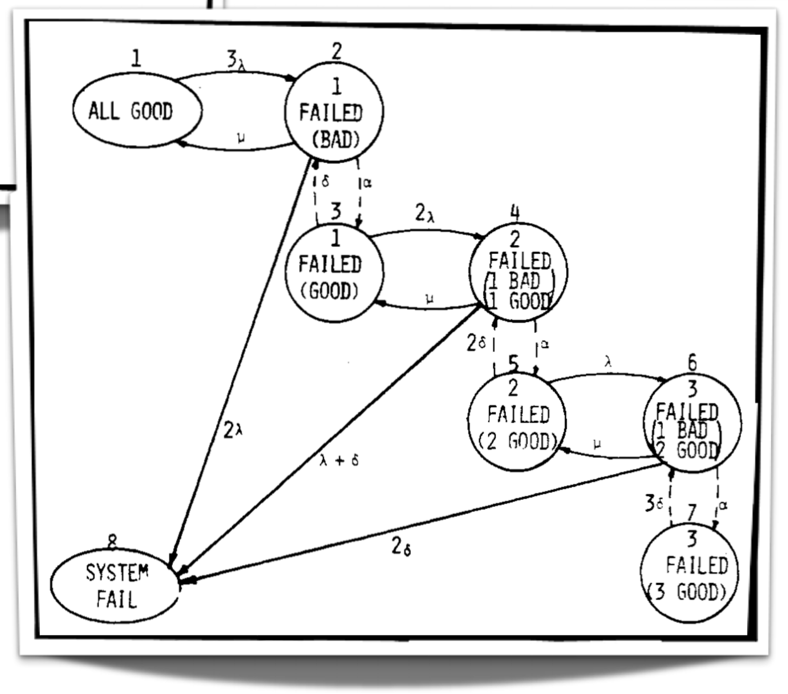
Rigorous testing and mathematical analyses



Fault trees

Timing analyses



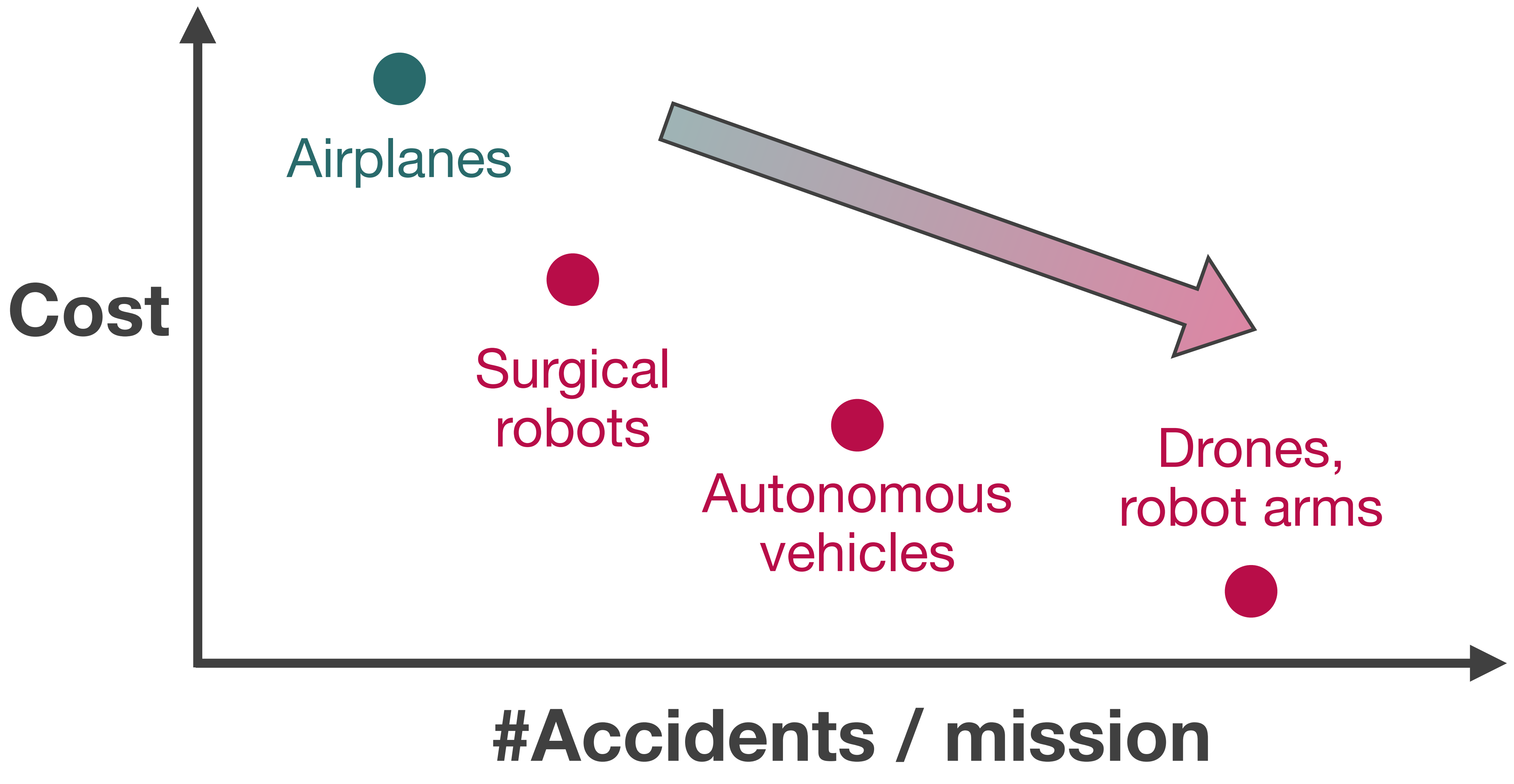


Markov processes

"Ultra-reliability"

- Quantifiably negligible failure rates
- $P_{fail} < 10^{-10}$ / hour

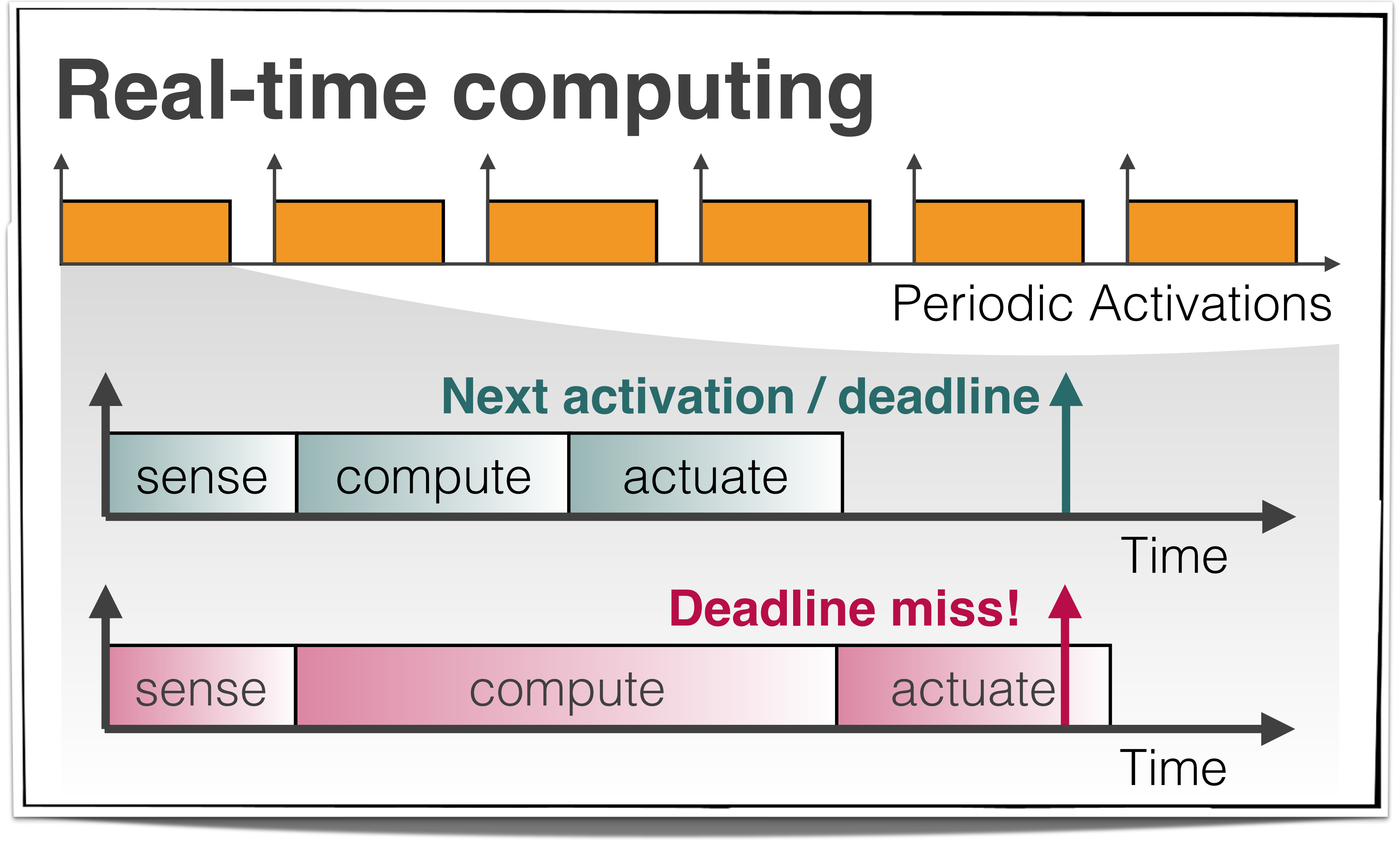
Not all CPS are Engineered like Airplanes

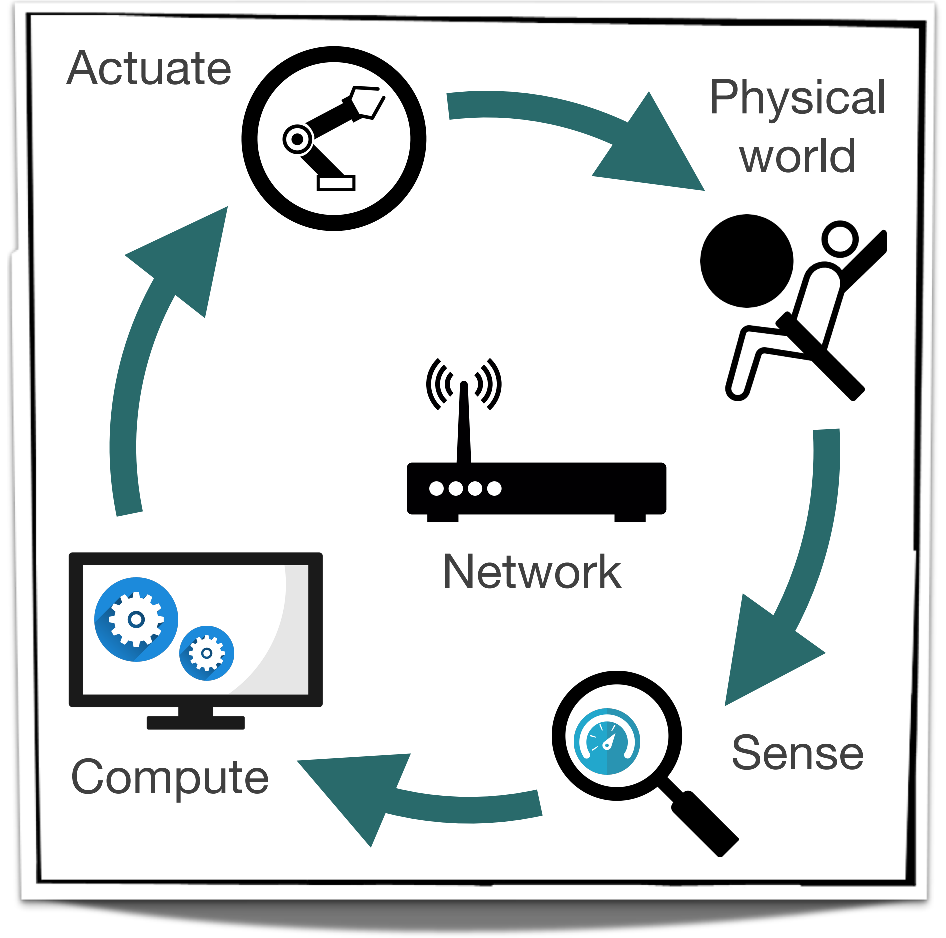


- × Inexpensive but **unreliable** off-the-shelf hardware
- × Open-source **unpredictable** software
- × **Inadequate resources**
- × Safety concerns regarding **ML** and **security**

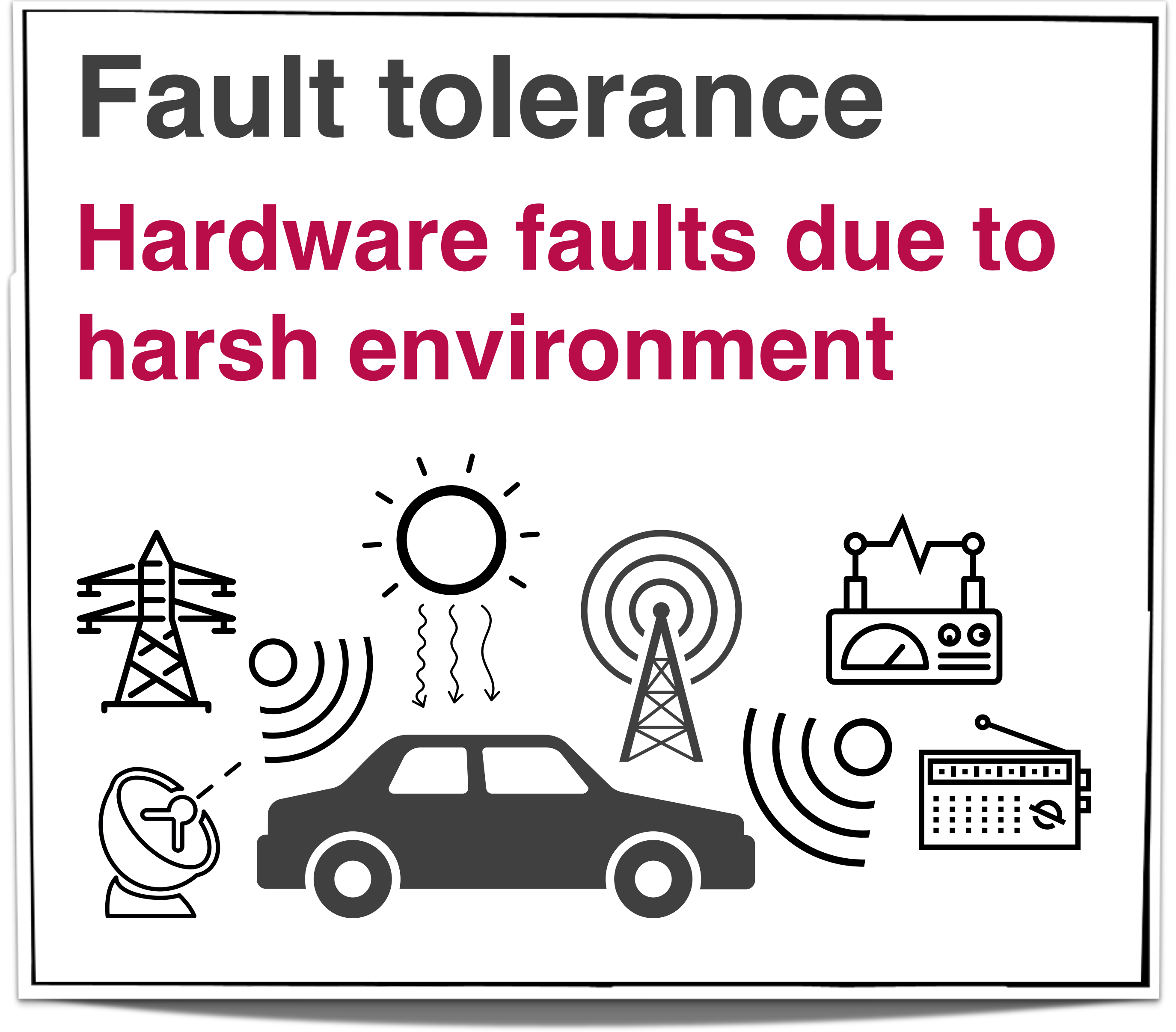
Goal: Make such low-cost consumer CPS more reliable

Focus: Real-Time Computing and Fault-Tolerance



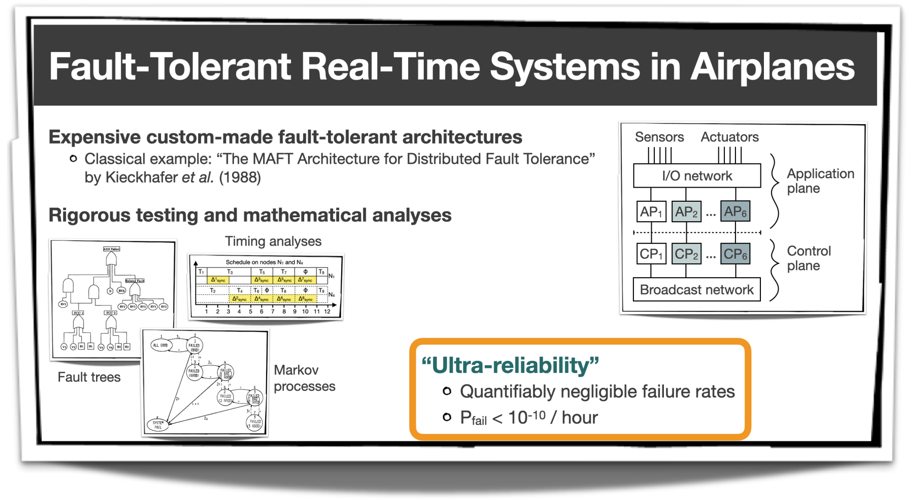


Feedback control loops



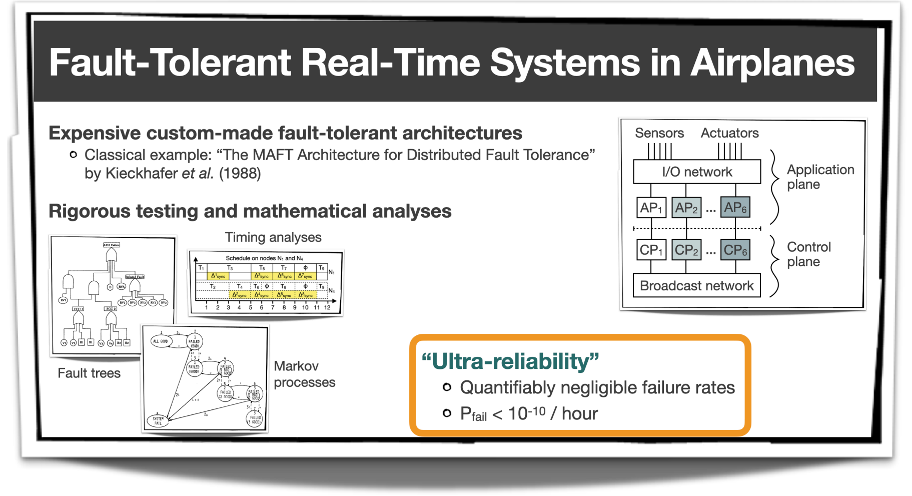
No good solutions for CPS-friendly Byzantine Fault Tolerance

No good solutions for CPS-friendly Byzantine Fault Tolerance



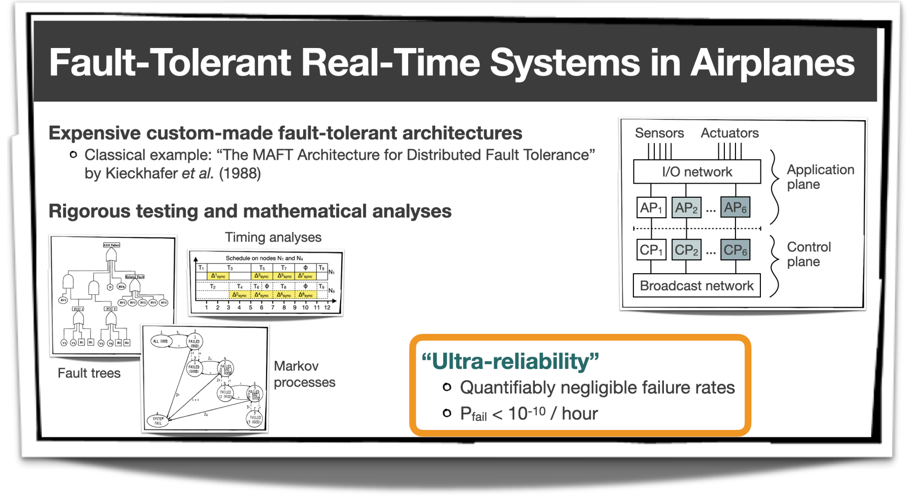
	Byzantine fault tolerance	Real-time predictability	Modern low-cost consumer CPS
Custom hardware	✔	✔	✘

No good solutions for CPS-friendly Byzantine Fault Tolerance



	Byzantine fault tolerance	Real-time predictability	Modern low-cost consumer CPS
Custom hardware	✔	✔	✘
Cloud datstores	✔	✘	✘

No good solutions for CPS-friendly Byzantine Fault Tolerance



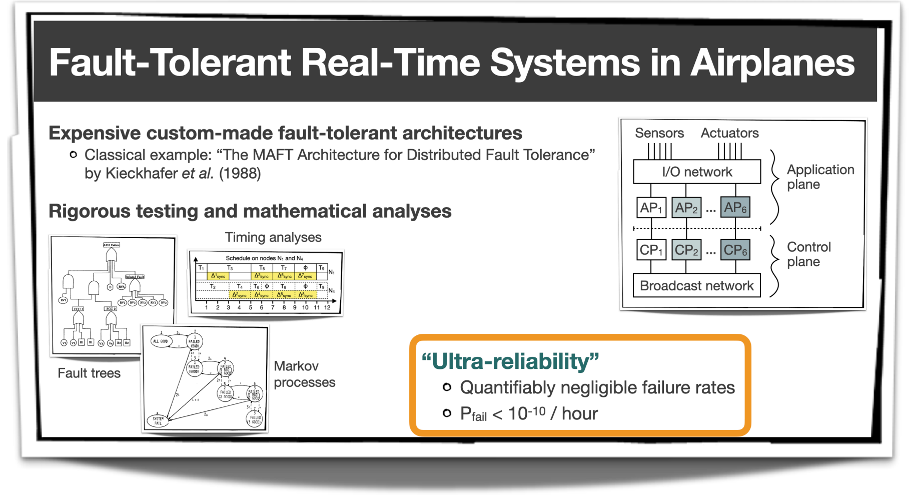
Bft-SMaRt
High-performance Byzantine Fault-Tolerant State Machine Replication

APACHE ZooKeeper™ etcd redis

ARDUPILOT ROS

	Byzantine fault tolerance	Real-time predictability	Modern low-cost consumer CPS
Custom hardware	✔	✔	✘
Cloud datstores	✔	✘	✘
CPS software	✘	✔	✔

No good solutions for CPS-friendly Byzantine Fault Tolerance



	Byzantine fault tolerance	Real-time predictability	Modern low-cost consumer CPS
Custom hardware	✔	✔	✘
Cloud datstores	✔	✘	✘
CPS software	✘	✔	✔
Achal KVS	✔	✔	✔

Distributed timestamped KVS

KVS Semantics

Key-Value Store (KVS)

Key-Value Store (KVS)

API

- `read(key k) -> value v | key error`
- `write(key k, value v) -> success | write error`

Key-Value Store (KVS)

API

- `read(key k) –> value v | key error`
- `write(key k, value v) –> success | write error`

What are the benefits of a KVS API?

- Simplifies programming
- Data sharing
- …

Timestamped KVS

Timestamped KVS

Revised API

- `read(key k, time t) -> value v | key error | time error`
- `write(key k, time t, value v) -> success | write error | time error`

Timestamped KVS

Revised API

- `read(key k, time t) -> value v | key error | time error`
- `write(key k, time t, value v) -> success | write error | time error`

How to interpret the `time` parameter?

- **Freshness constraint** during `read`
  - Return any value `v` that was written at or later than time `t`
- **Publishing time** during `write`
  - Ensure that value `v` cannot be read before time `t`
  - Ensure that value `v` can be read at or later than time `t`
- For simplicity, consider the **unique key** $\mathbf{k_{unique}} = \mathbf{(k,\ t)}$**!**

Timestamped KVS

Revised API

- `read(key k, time t) -> value v | key error | time error`
- `write(key k, time t, value v) -> success | write error | time error`

How to interpret the `time` parameter?

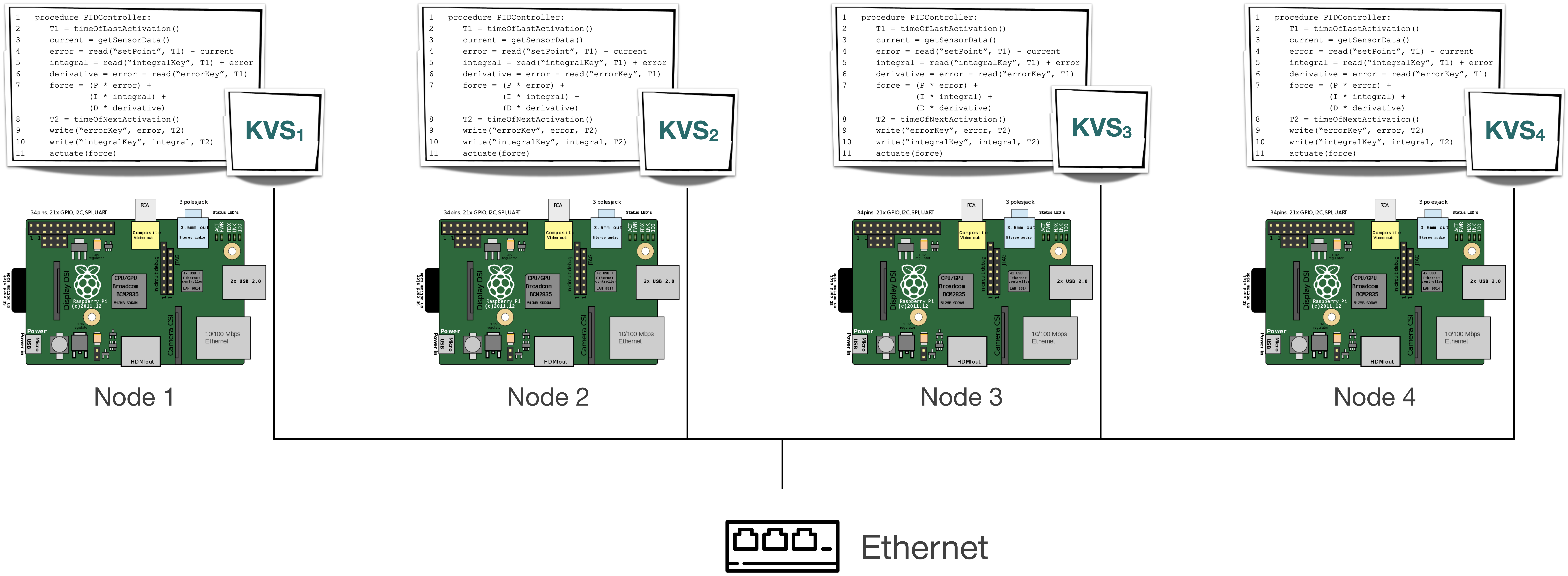
- **Freshness constraint** during `read`
  - Return any value `v` that was written at or later than time `t`
- **Publishing time** during `write`
  - Ensure that value `v` cannot be read before time `t`
  - Ensure that value `v` can be read at or later than time `t`
- For simplicity, consider the **unique key** $k_{unique} = (k, t)$!

What are the benefits of a timestamped KVS?

- Data versioning in financial markets
- Sensor data in cyber-physical systems
- …

Distributed Timestamped KVS

Distributed Timestamped KVS



Distributed Timestamped KVS

What are the benefits of a distributed KVS?

- Applications may inherently be distributed
- Fault tolerance
  - Crash
  - Incorrect computation
  - Network issues
- …

Achal KVS

Inverted Pendulum: A Prototypical Control Application

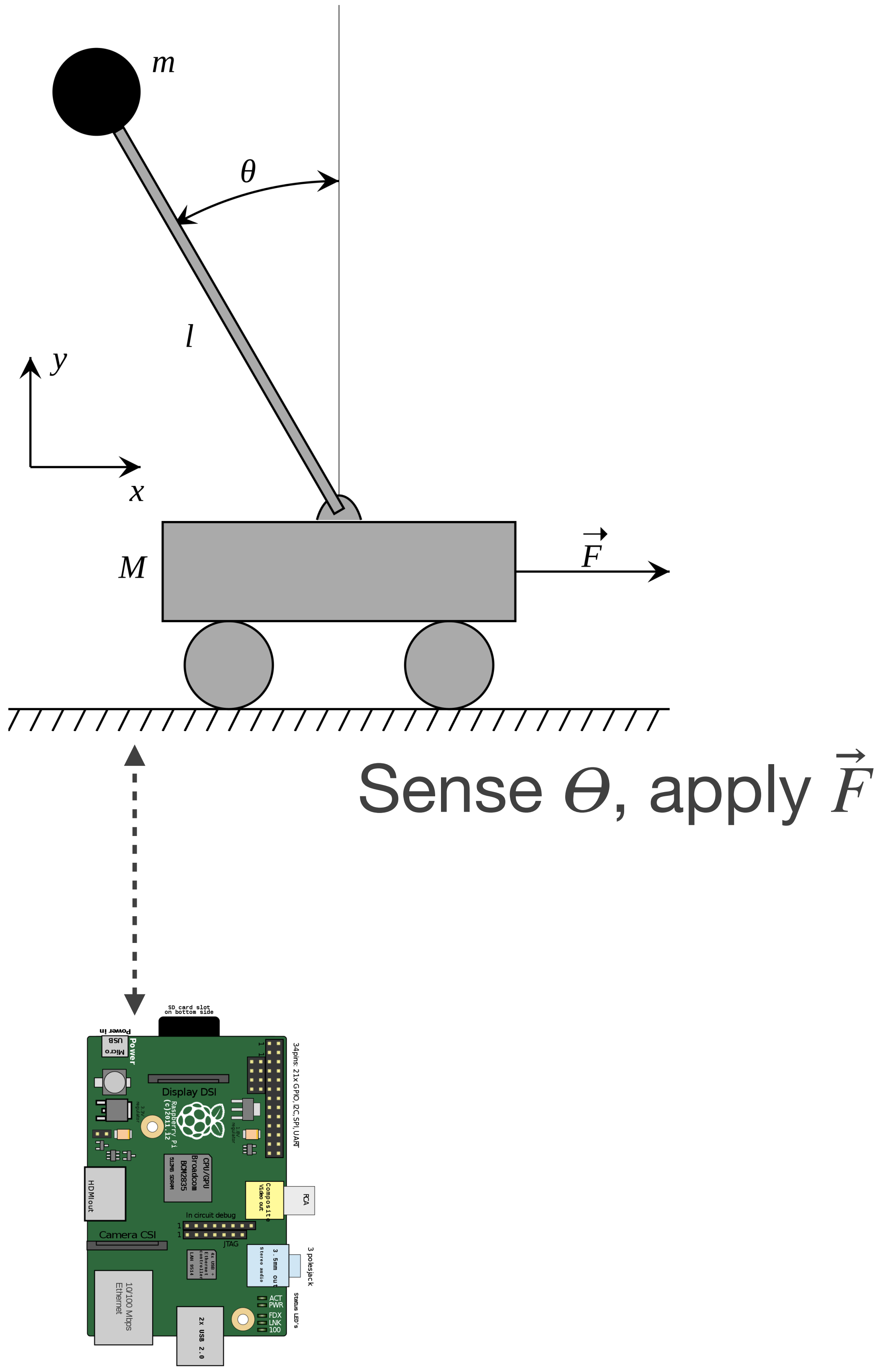
Inverted Pendulum: A Prototypical Control Application

```
procedure PIDController:              // balance an inverted pendulum

    current = getSensorData()         // get angle encoder value
    error = setPoint - current        // compute absolute error
    integral = integral + error       // compute cumulative error
    derivative = error - oldError     // compute change in error
    force = (P * error) +             // compute force using PID
            (I * integral) +
            (D * derivative)

    oldError = error

    actuate(force)                   // apply force on the cart
```



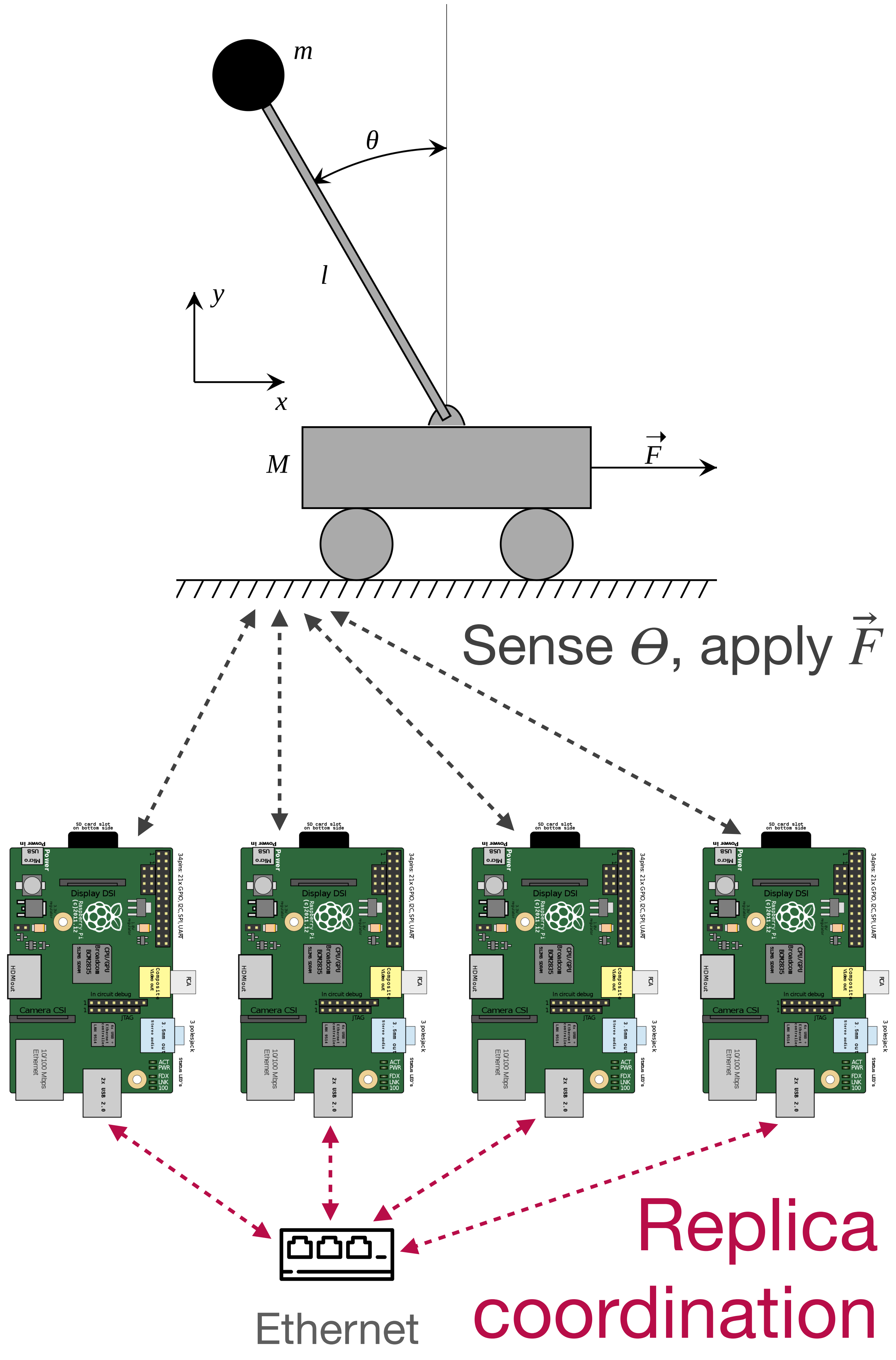
Inverted Pendulum: A Prototypical Control Application

```
procedure PIDController:              // balance an inverted pendulum

    current = getSensorData()         // get angle encoder value
    error = setPoint - current        // compute absolute error
    integral = integral + error       // compute cumulative error
    derivative = error - oldError     // compute change in error
    force = (P * error) +             // compute force using PID
            (I * integral) +
            (D * derivative)

    oldError = error

    actuate(force)                   // apply force on the cart
```



Nontrivial for control application developers!

Time-Aware Key-Value API

```
procedure PIDController:
   T1 = timeOfLastActivation()
   current = getSensorData()
   error = read(“setPoint”, T1) - current
   integral = read(“integralKey”, T1) + error
   derivative = error - read(“errorKey”, T1)
   force = (P * error) +
           (I * integral) +
           (D * derivative)
   T2 = timeOfNextActivation()
   write(“errorKey”, error, T2)
   write(“integralKey”, integral, T2)
   actuate(force)
```

T1 is a **data freshness** constraint

T2 denotes **publishing time**

Time-Aware Key-Value API

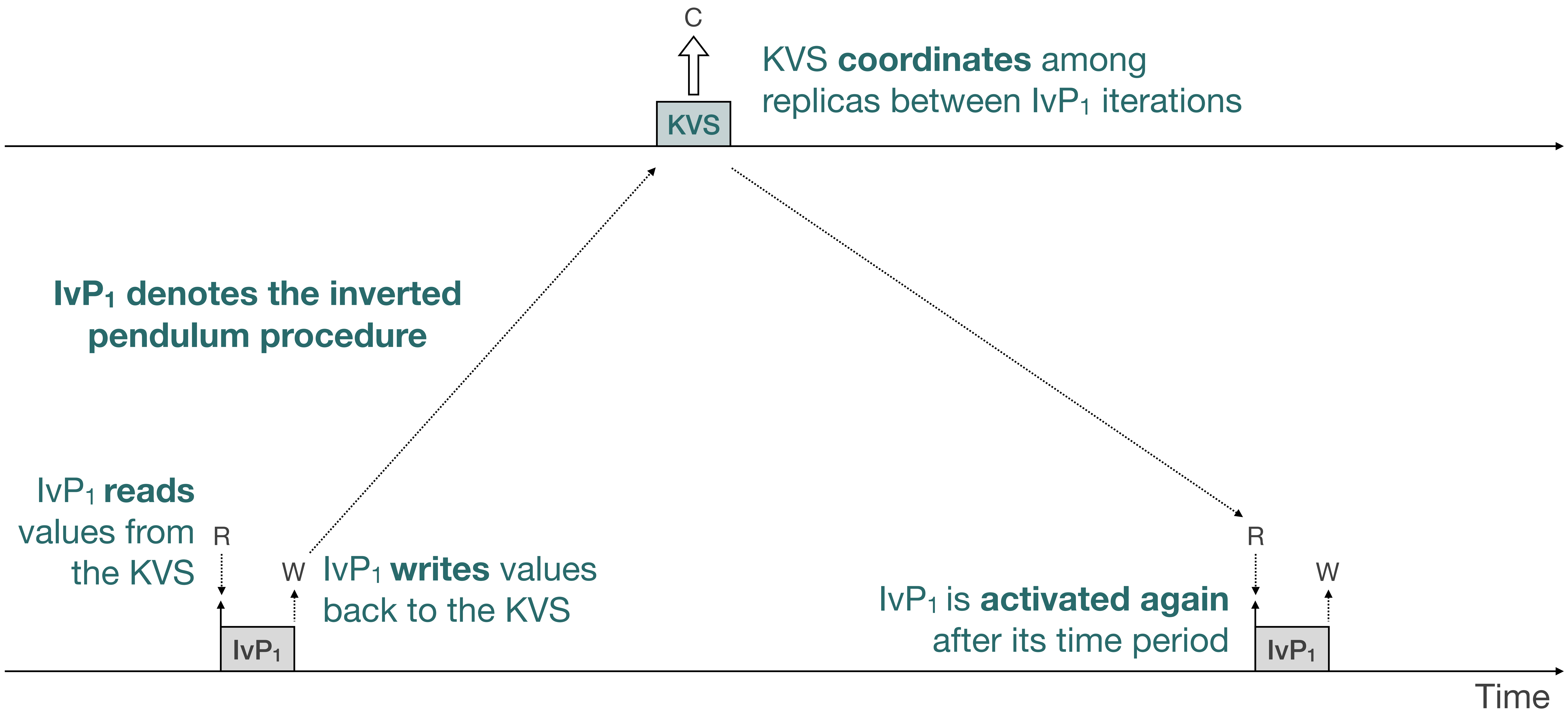
```
procedure PIDController:
   T1 = timeOfLastActivation()
   current = getSensorData()
   error = read("setPoint", T1) - current
   integral = read("integralKey", T1) + error
   derivative = error - read("errorKey", T1)
   force = (P * error) +
           (I * integral) +
           (D * derivative)
   T2 = timeOfNextActivation()
   write("errorKey", error, T2)
   write("integralKey", integral, T2)
   actuate(force)
```

T1 is a **data freshness** constraint

Key-value API **simplifies replica coordination**

T2 denotes **publishing time**

Time parameters help with **temporal determinism**

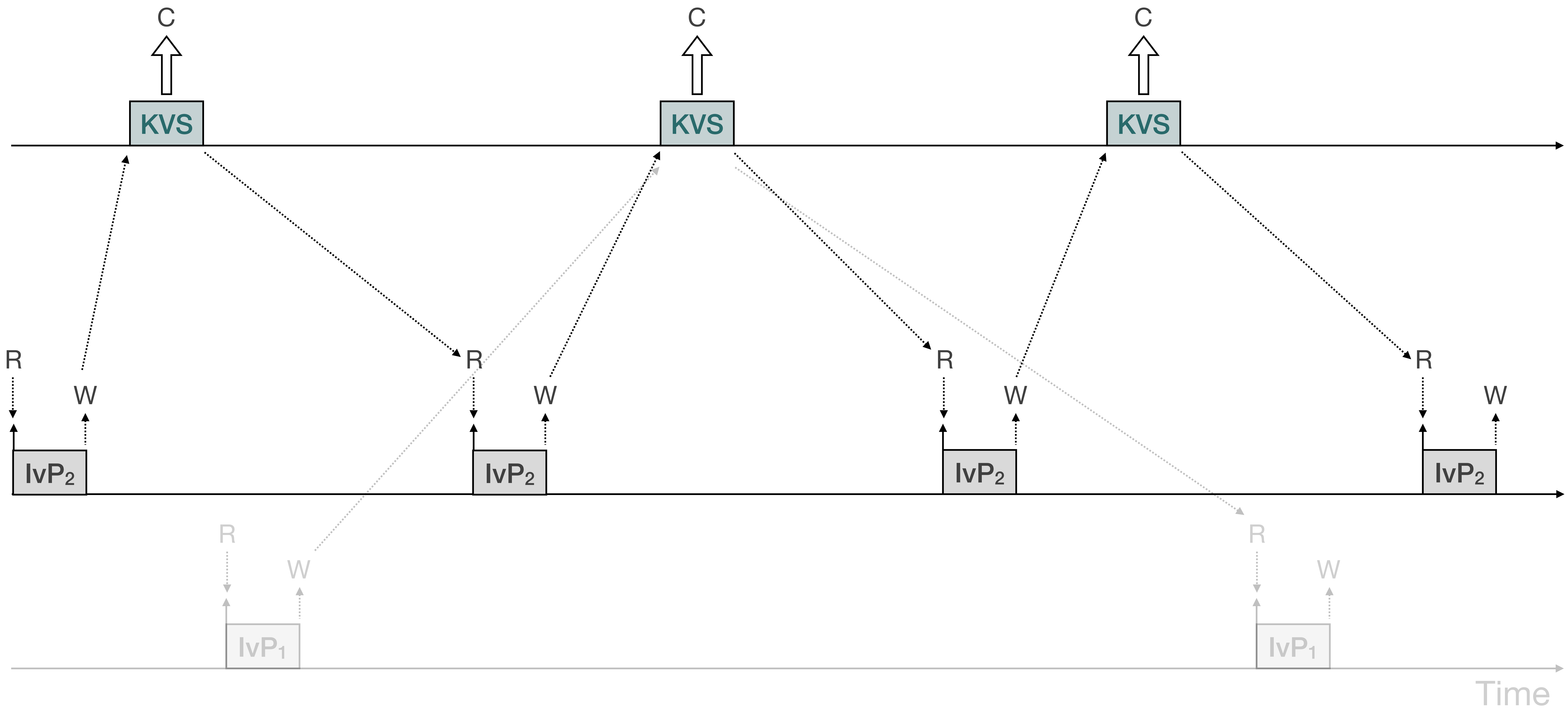


Schedule on node 1

Assume a single core in use

R = read, W = write, C = coordination

IvP = Inverted pendulum control application, KVS = Achal's backend

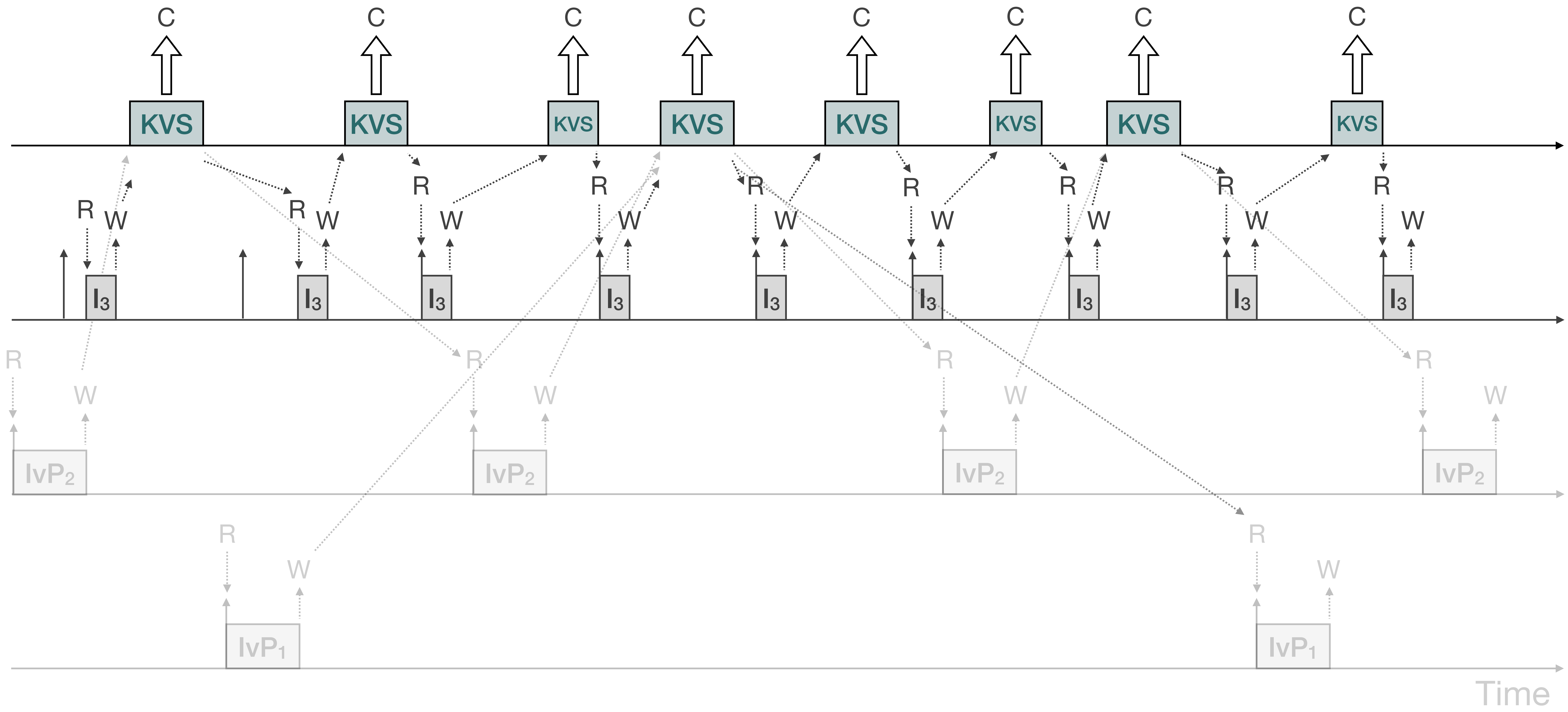


Schedule on node 1

Assume a single core in use

R = read, W = write, C = coordination

IvP = Inverted pendulum control application, KVS = Achal's backend



Schedule on node 1

Assume a single core in use

R = read, W = write, C = coordination

IvP = Inverted pendulum control application, KVS = Achal's backend

Building Blocks

Building Blocks

Clock synchronization

- Make sense of absolute publishing times across distributed nodes

Building Blocks

Clock synchronization

- Make sense of absolute publishing times across distributed nodes

EIGByz*# for Byzantine fault tolerance

- **Synchronous** → Exploits clock synchronization for better performance
- **Leaderless** → Higher reliability!
- **Interactive consistency** → Useful for noisy sensor values
- Simple algorithm → Can be easily parameterized in #nodes, #rounds
- **E**xponential **I**nformation **G**athering trees → Easily flattened for fast reads and writes

---

* Pease, Shostak, and Lamport. "Reaching agreement in the presence of faults." J. ACM (1980)

Borran and Schiper. "A Leader-Free Byzantine Consensus Algorithm." ICDCN (2010)

Design Principle: Make the KVS Strictly Periodic

Design Principle: Make the KVS Strictly Periodic

Worst-case execution time?

- Optimize EIGByz's implementation for predictability + Empirical profiling

Design Principle: Make the KVS Strictly Periodic

Worst-case execution time?

- Optimize EIGByz's implementation for predictability + Empirical profiling

Time period?

- Small enough so that publishing times are satisfiable
- … but not at the cost of poor CPU utilization!
- Partitioned scheduling + uniprocessor response-time analysis

Design Principle: Make the KVS Strictly Periodic

Worst-case execution time?

- Optimize ElGByz's implementation for predictability + Empirical profiling

Time period?

- Small enough so that publishing times are satisfiable
- … but not at the cost of poor CPU utilization!
- Partitioned scheduling + uniprocessor response-time analysis

Achal is tuned as a function of both the workload and the platform!

Evaluation

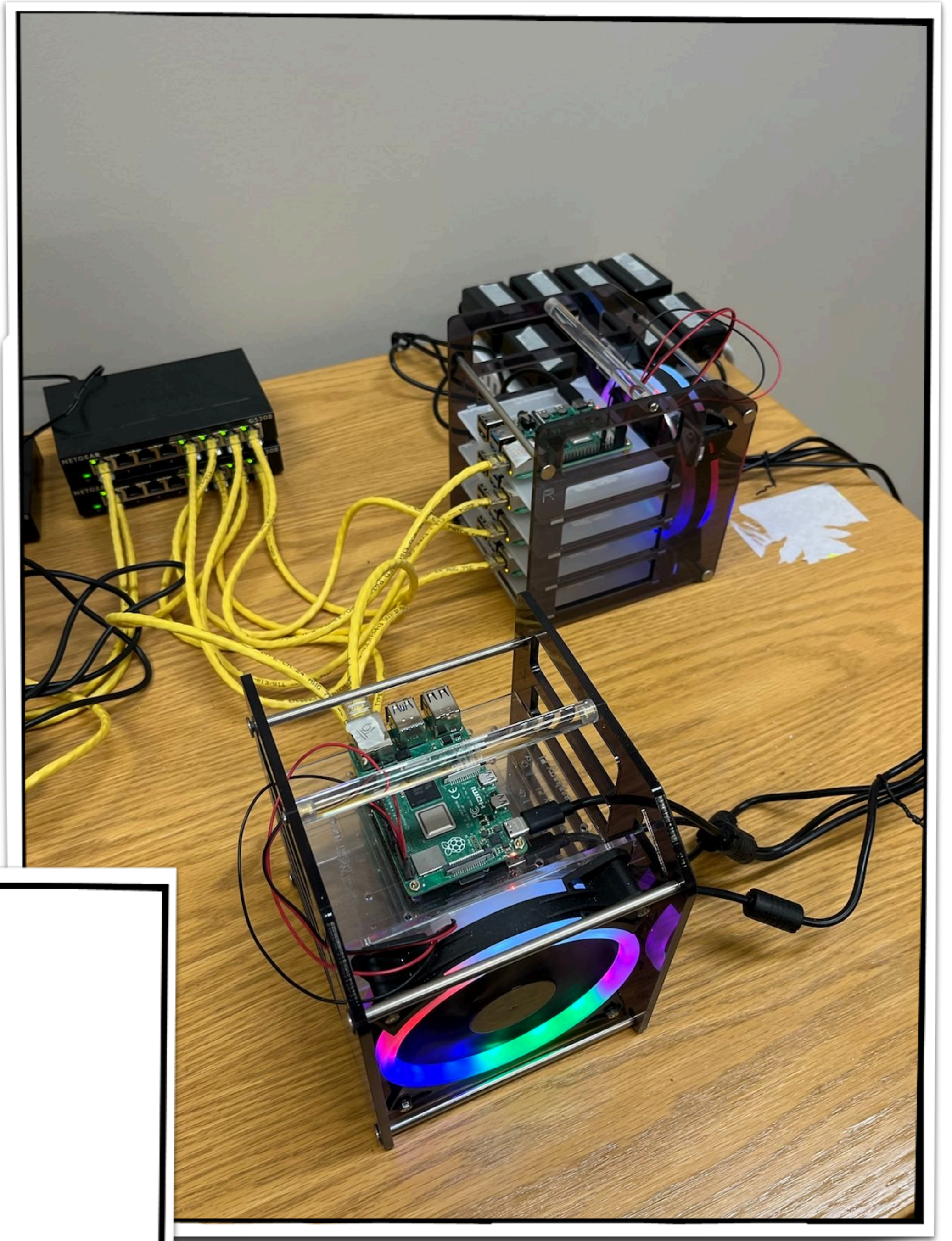
How does Achal compare against well-known datastores?

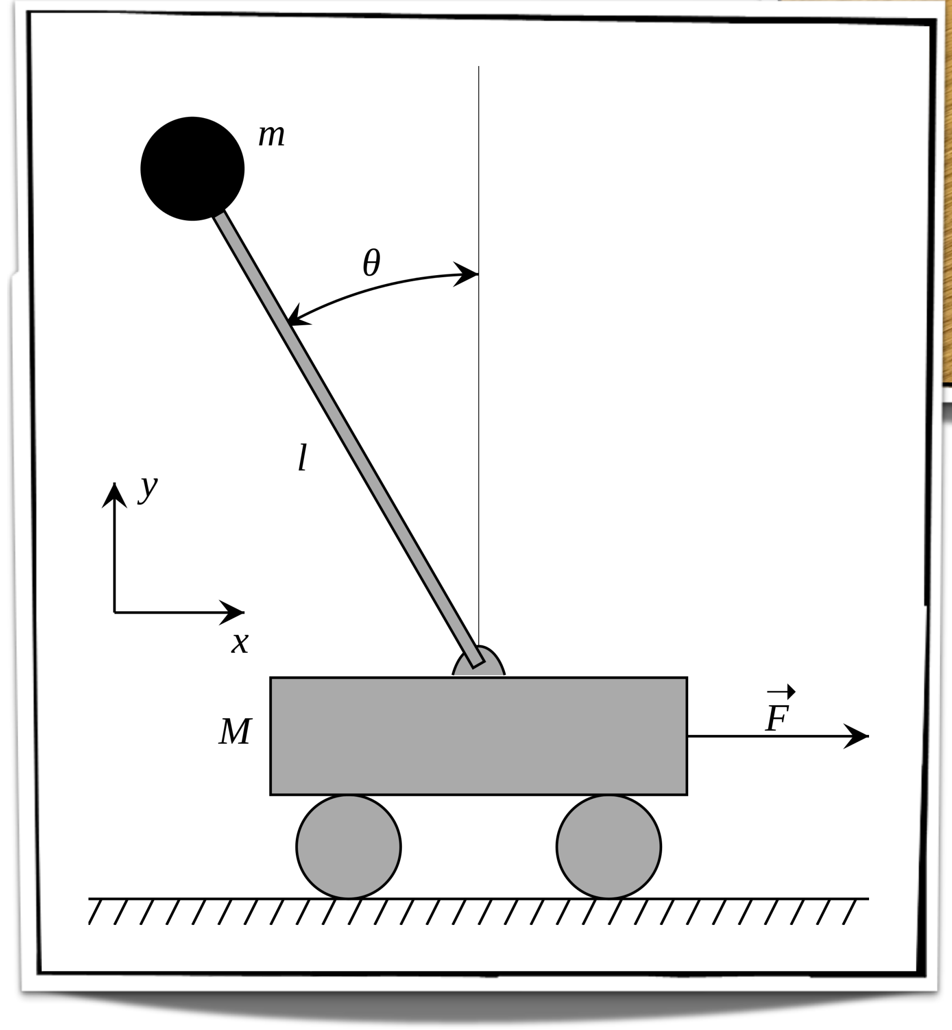
Platform: Four **Raspberry Pi** 4 Model B + Ethernet

Baselines: redis etcd

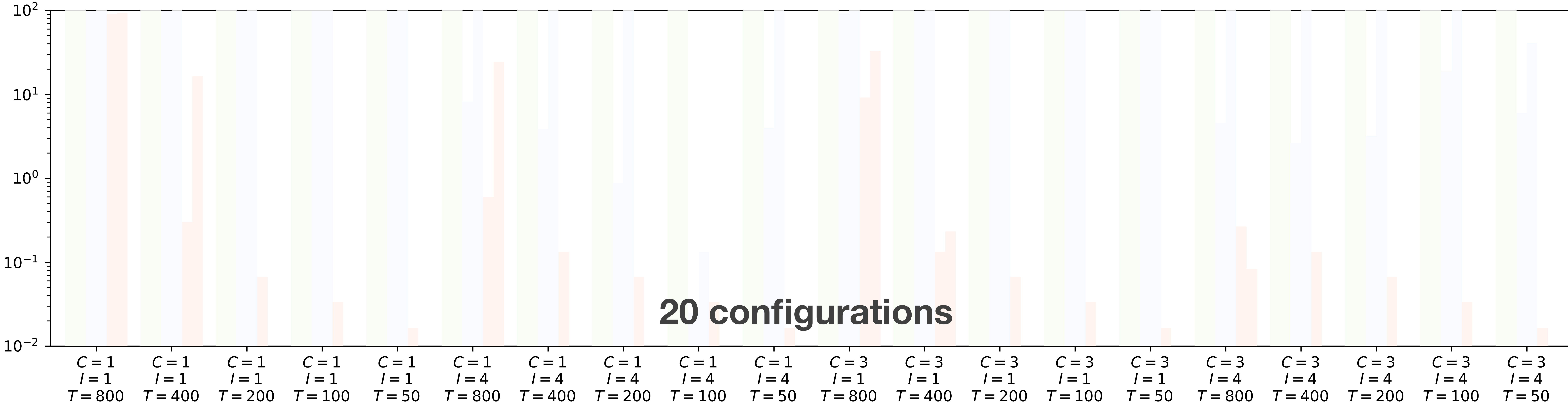
Workload

- **IvPSim:** Periodic task simulating inverted pendulum control
- Each task reads/writes 20 floats
- Coordinate data written by IvPSim replicas every iteration

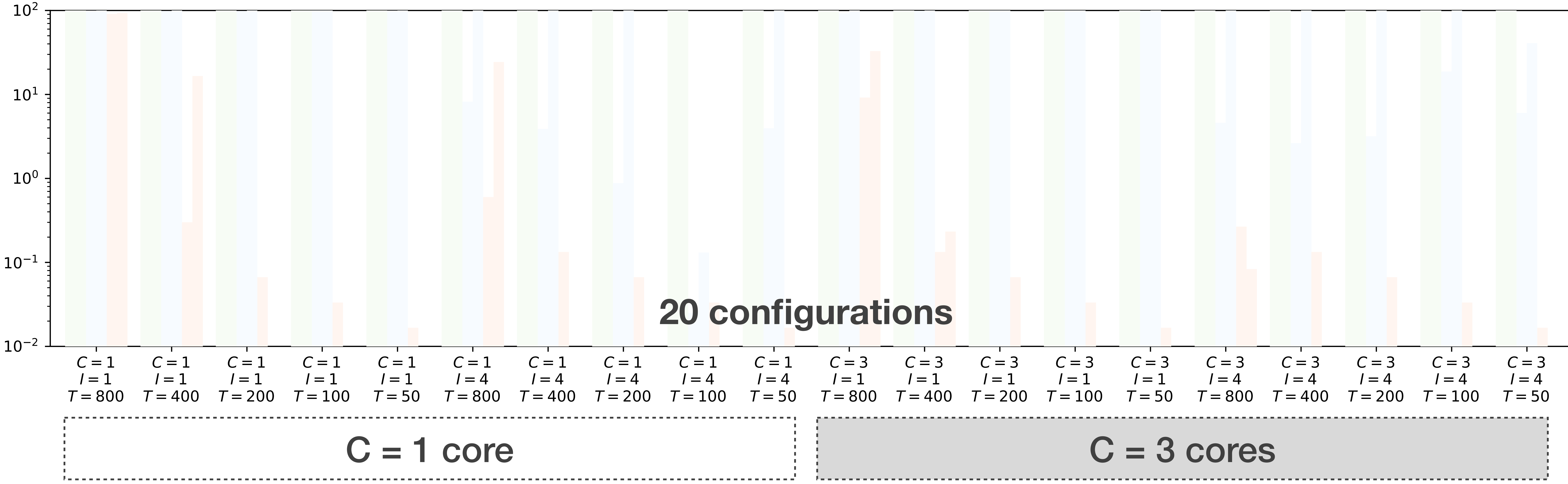




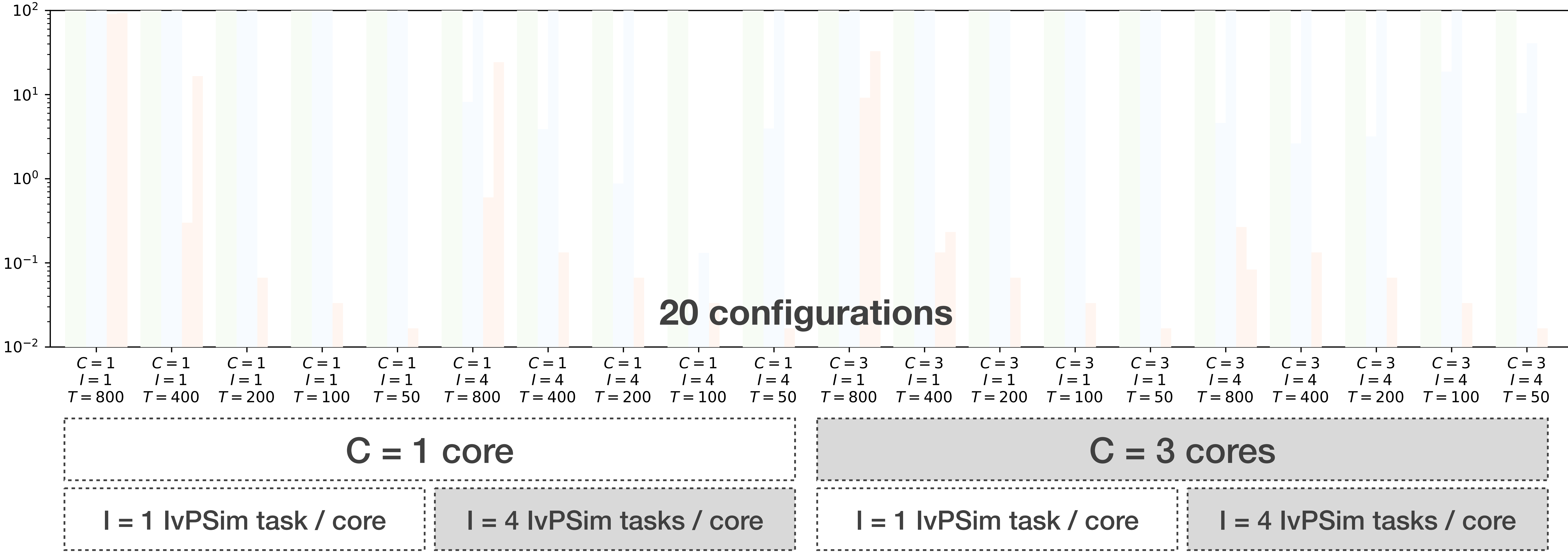
How does Achal compare against well-known KVS?



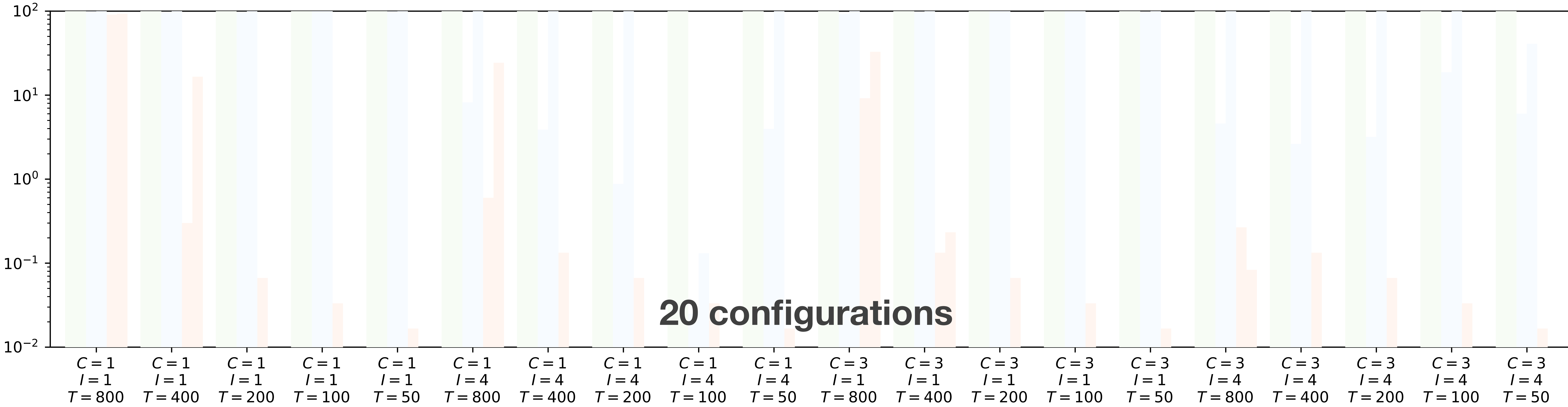
How does Achal compare against well-known KVS?



How does Achal compare against well-known KVS?



How does Achal compare against well-known KVS?



C = 1 core

C = 3 cores

I = 1 IvPSim task / core

I = 4 IvPSim tasks / core

I = 1 IvPSim task / core

I = 4 IvPSim tasks / core

Decreasing Period →

Decreasing Period →

Decreasing Period →

Decreasing Period →

How does Achal compare against well-known KVS?

Metrics

- Read: % of iterations where all reads were successful
- Write: …

Higher is better!

Log scale

C = 1 core		C = 3 cores	
I = 1 IvPSim task / core	I = 4 IvPSim tasks / core	I = 1 IvPSim task / core	I = 4 IvPSim tasks / core
Decreasing Period →	Decreasing Period →	Decreasing Period →	Decreasing Period →

How does Achal compare against well-known KVS?

Higher is better!

Log scale

Achal's success rate is 100%

How does Achal compare against well-known KVS?

Higher is better!

Log scale

Achal's success rate is 100%

etcd almost always underperforms

Redis underperforms when the workload is increased

How does Achal compare against well-known KVS?

Higher is better!

Log scale

Achal's success rate is 100%

etcd almost always underperforms

Achal successfully managed an automotive workload 10x-100x lvPSim BOSCH

Redis underperforms when the workload is increased

How does Achal compare against well-known KVS?

Higher is better!

Log scale

Achal's success rate is 100%

Redis underperforms when the workload is increased

Summary

Key Takeaways

Key Takeaways

Interplay between timing and fault tolerance is nontrivial

- Lots of interesting system design problems to be explored in the space of distributed real-time systems in the CPS domain …

Key Takeaways

Interplay between timing and fault tolerance is nontrivial

- Lots of interesting system design problems to be explored in the space of distributed real-time systems in the CPS domain …

Need for *a priori* timing and reliability analyses makes the system design problem even more challenging!

- How often does Achal fail? Is $P_{fail} < 10^{-10}$/hr?
- Can we add yet another control task without affecting the system timeliness?

Key Takeaways

Interplay between timing and fault tolerance is nontrivial

- Lots of interesting system design problems to be explored in the space of distributed real-time systems in the CPS domain …

Need for *a priori* timing and reliability analyses makes the system design problem even more challenging!

- How often does Achal fail? Is $P_{fail} < 10^{-10}$/hr?
- Can we add yet another control task without affecting the system timeliness?

More details about Achal in the paper …

- "Interactive Consistency meets Distributed Real-Time Systems, Again!" at RTSS 2022

CPS Research

Developing **foundations** needed to engineer complex CPS

… which require **“dependable, high-confidence, or provable behaviors”***

1. **Runtime mechanisms**
2. **Analysis techniques**

* https://beta.nsf.gov/funding/opportunities/cyber-physical-systems-cps

Contact

- Name: Arpan B. Gujarati
- Email: arpanbg@cs.ubc.ca
- Web: https://arpangujarati.github.io/

CPS Research

Developing **foundations** needed to engineer complex CPS

… which require **“dependable, high-confidence, or provable behaviors”***

1. **Runtime mechanisms**
2. **Analysis techniques**

* https://beta.nsf.gov/funding/opportunities/cyber-physical-systems-cps