In-ConcReTeS

Interactive Consistency meets Distributed Real-Time Systems, Again!

Arpan Gujarati and Ningfeng Yang (UBC)
Björn Brandenburg (MPI-SWS)

This paper in a nutshell …

In-ConcReTeS

Distributed key-value store

Time-sensitive, fault-tolerant replica coordination

Random environmentally induced faults

Motivation

Embedded Systems are Susceptible to Transient Faults

Harsh environments

- **Electric motors, spark plugs** inside automobiles
- Industrial systems and robots operating under **hard radiation** and **near high power** machinery

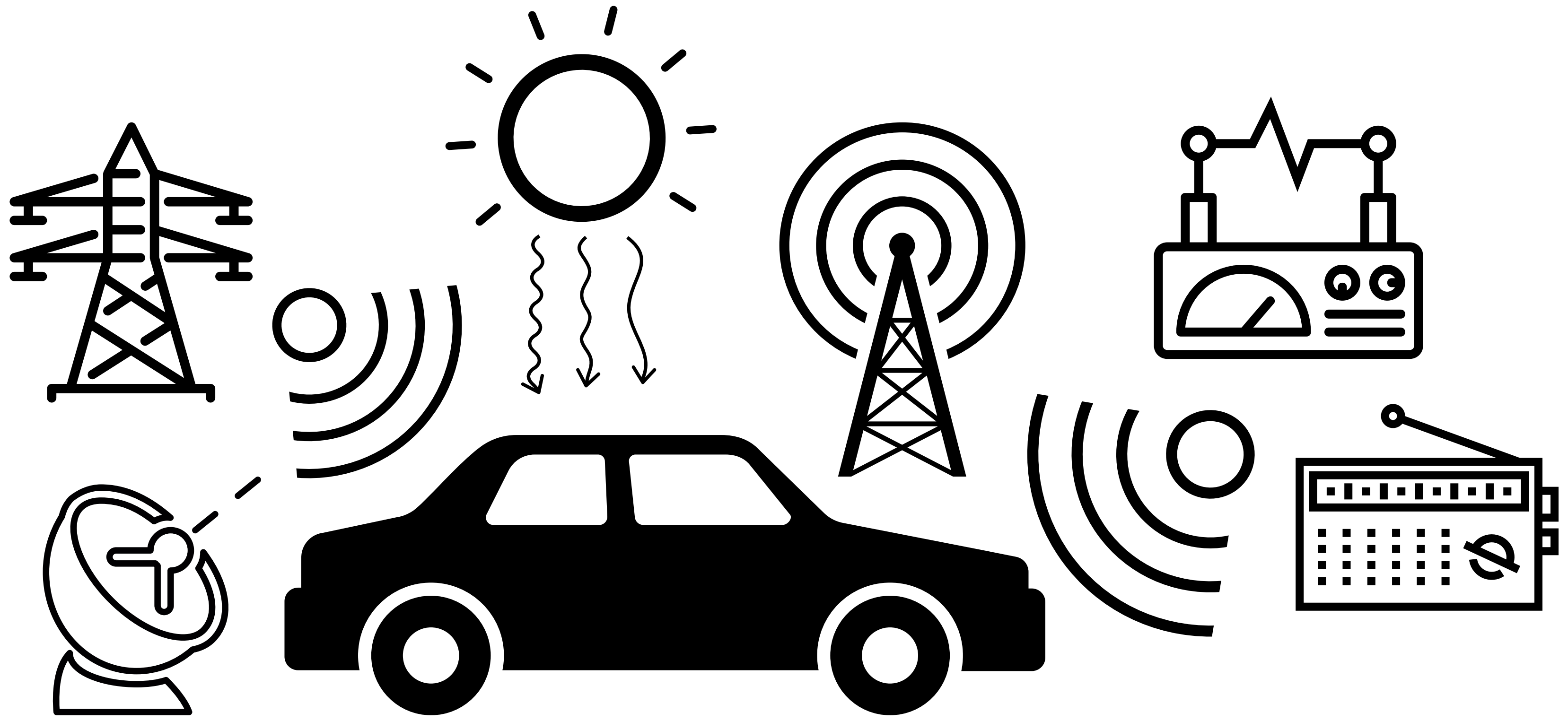
Embedded Systems are Susceptible to Transient Faults

Harsh environments

- **Electric motors, spark plugs** inside automobiles
- Industrial systems and robots operating under **hard radiation** and **near high power** machinery

Transient faults or bit flips in registers, buffers, and networks

- For example, with 1 bit flip in a 1 MB SRAM every 10^{12} hours of operation
- and 0.5 billion cars with an average daily operation time of 5%
- **about 5000 cars may be affected by a bit flip every day!***



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

Transient Faults can lead to Errors and Failures

Transmission errors
- Faults in the network

Omission errors
- Fault-induced kernel panics, hangs

Incorrect computation errors
- Faults in memory buffers

Byzantine / inconsistent broadcast errors*
- Faults in distributed systems

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

Transient Faults can lead to Errors and Failures

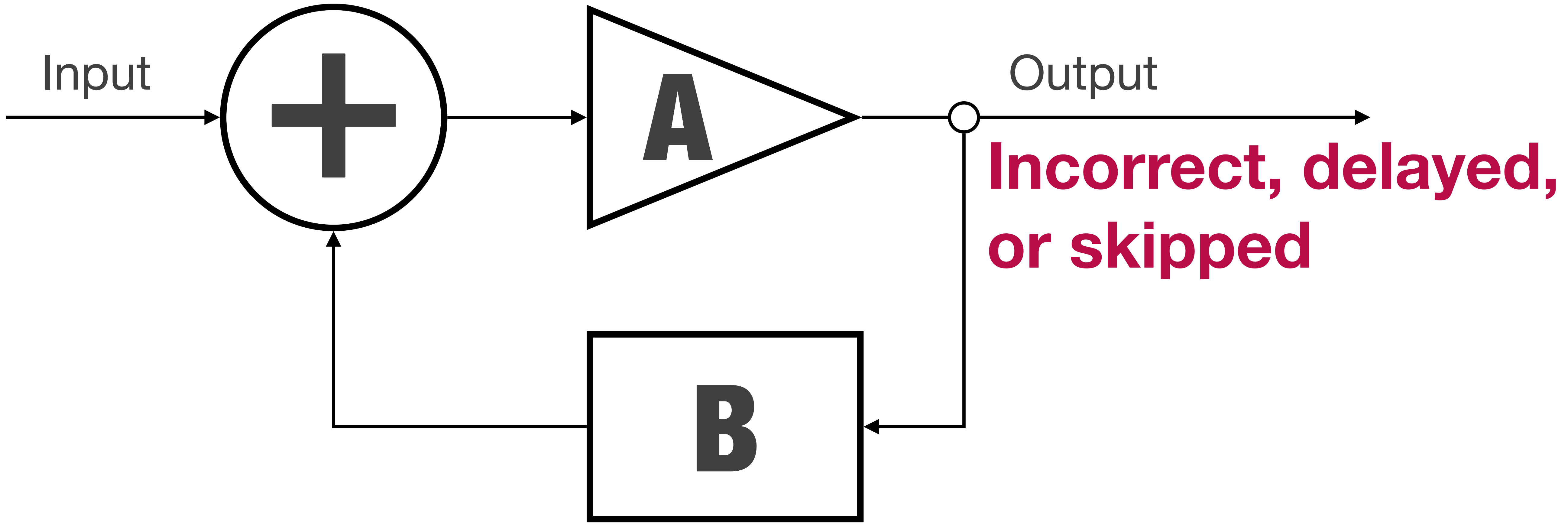
Transmission errors
- Faults in the network

Omission errors
- Fault-induced kernel panics, hangs

Incorrect computation errors
- Faults in memory buffers

Byzantine / inconsistent broadcast errors*
- Faults in distributed systems

Example: Safety-critical control systems can fail in both time and value domains



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

Transient Faults can lead to Errors and Failures

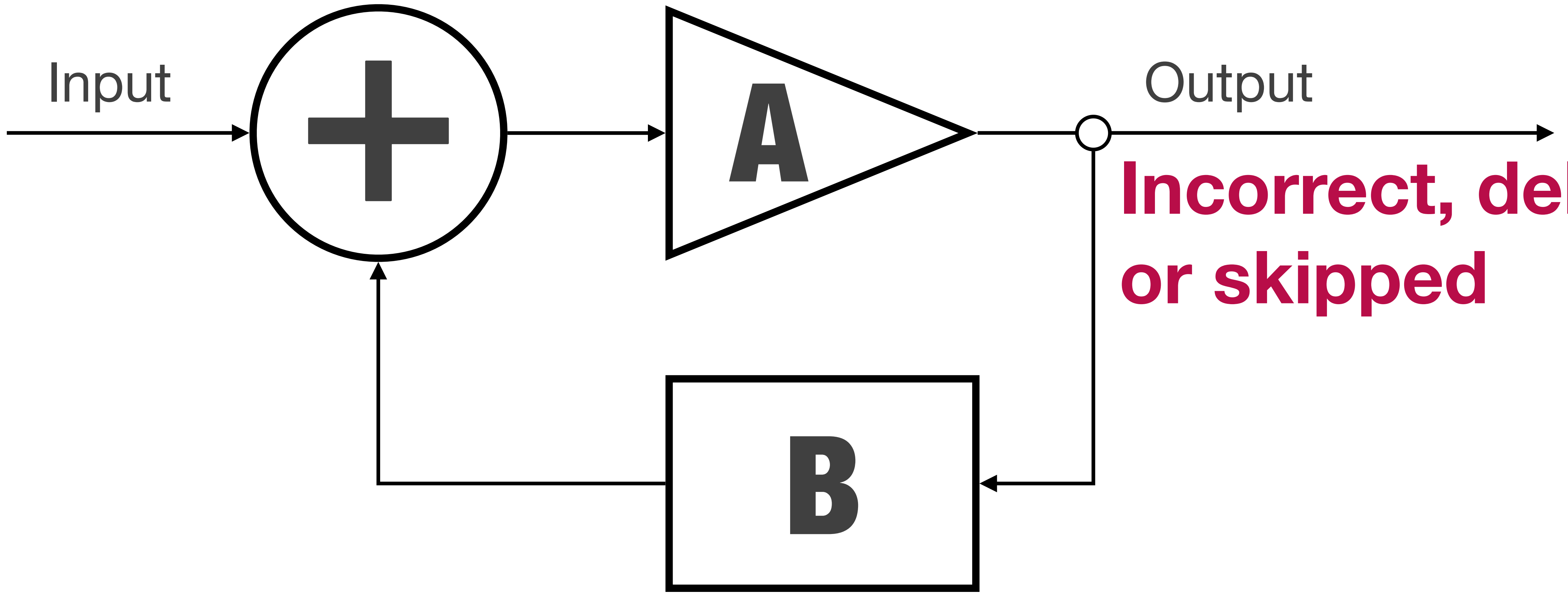
Transmission errors
- Faults in the network

Omission errors
- Fault-induced kernel panics, hangs

Incorrect computation errors
- Faults in memory buffers

Byzantine / inconsistent broadcast errors*
- Faults in distributed systems

Example: Safety-critical control systems can fail in both time and value domains

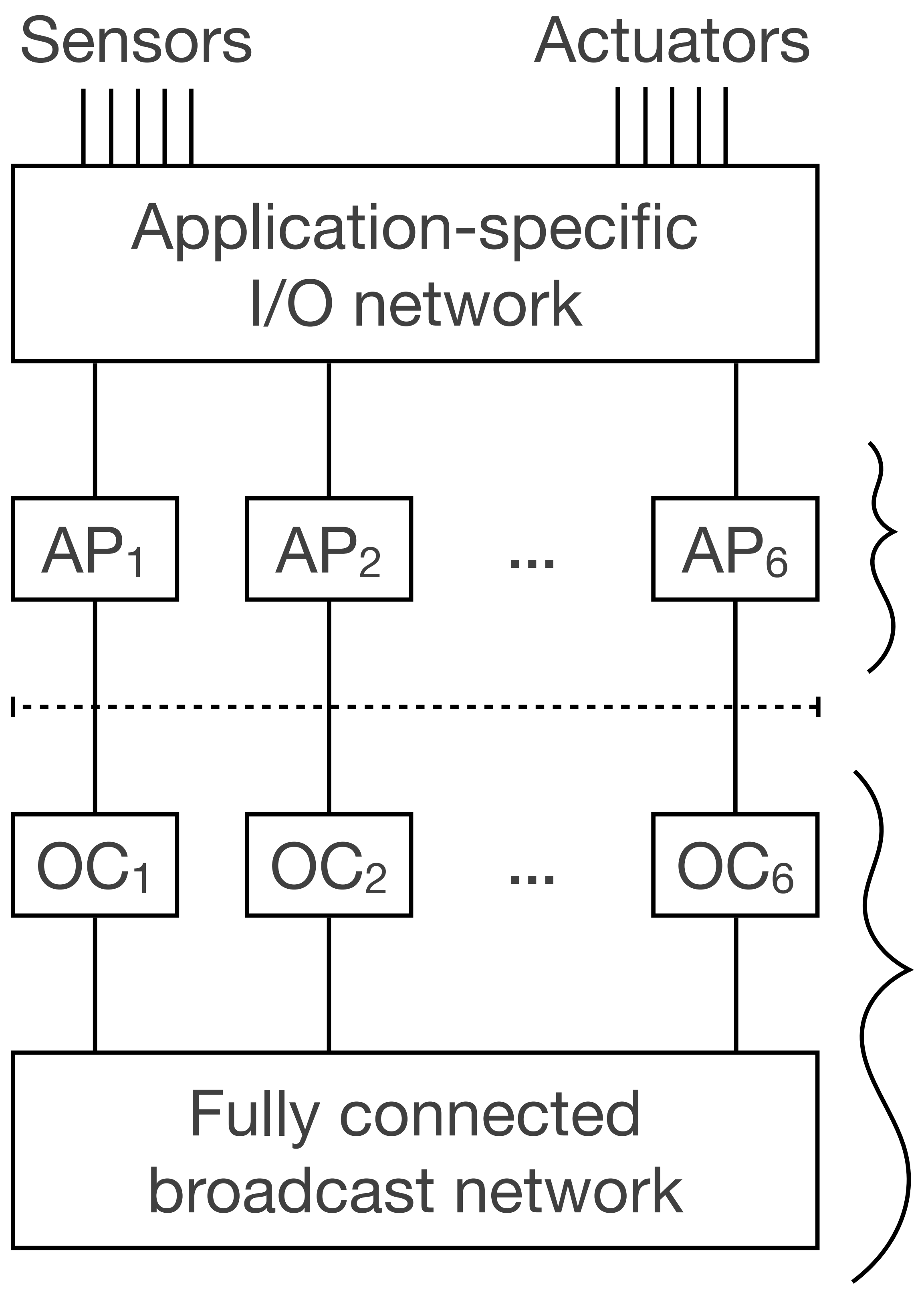


Incorrect, delayed, or skipped

Many influential avionics domain architectures in the 80s and 90s designed for safety-critical real-time systems addressed this problem in depth!

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

Multiprocessor Architecture for Fault Tolerance (MAFT)*

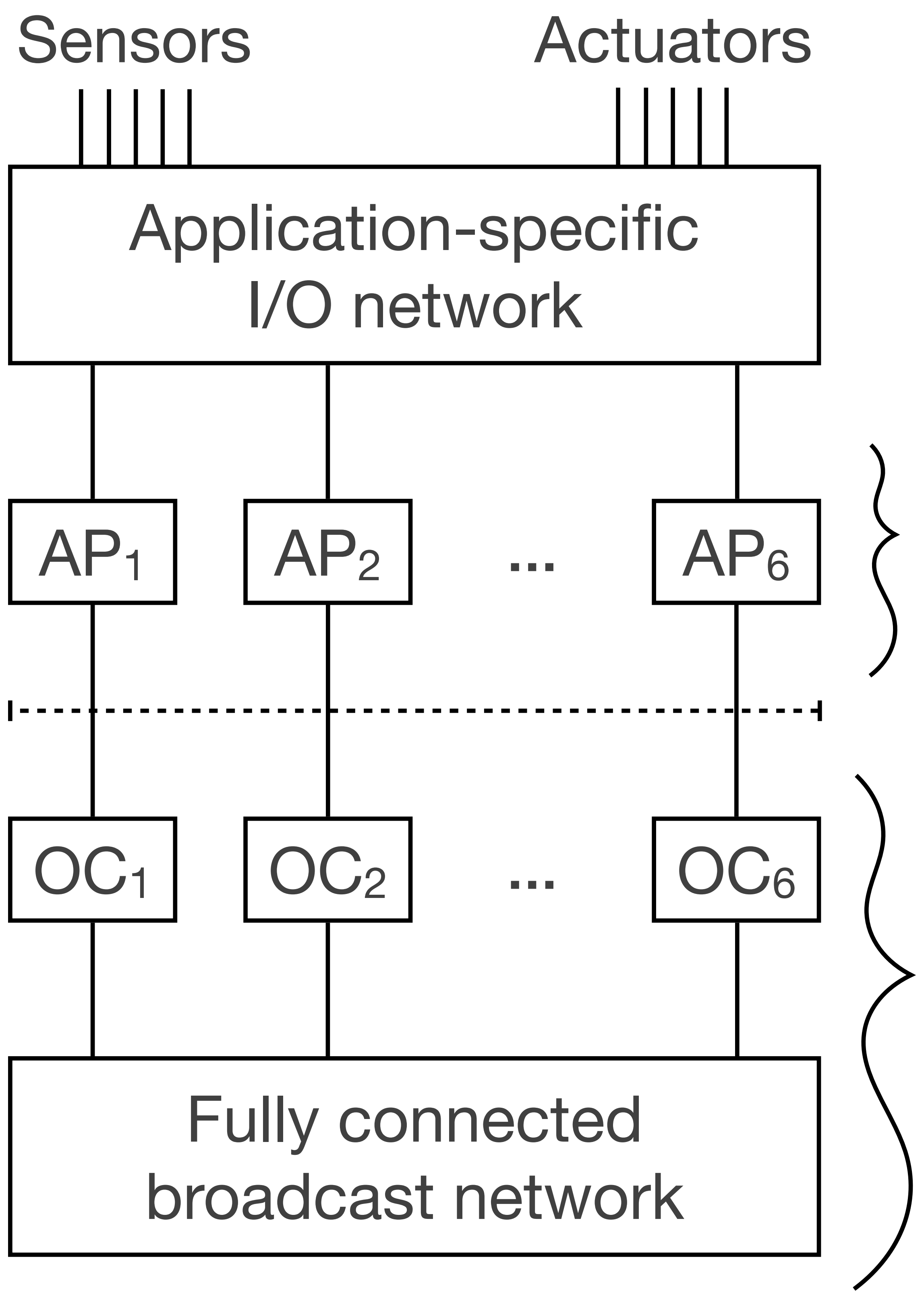


General-purpose application processors (APs)

Custom operation controllers (OCs) for voting, scheduling, synchronization, etc.

* Kieckhafer *et al.* “The MAFT Architecture for Distributed Fault Tolerance.” IEEE Transactions on Computers (1988)

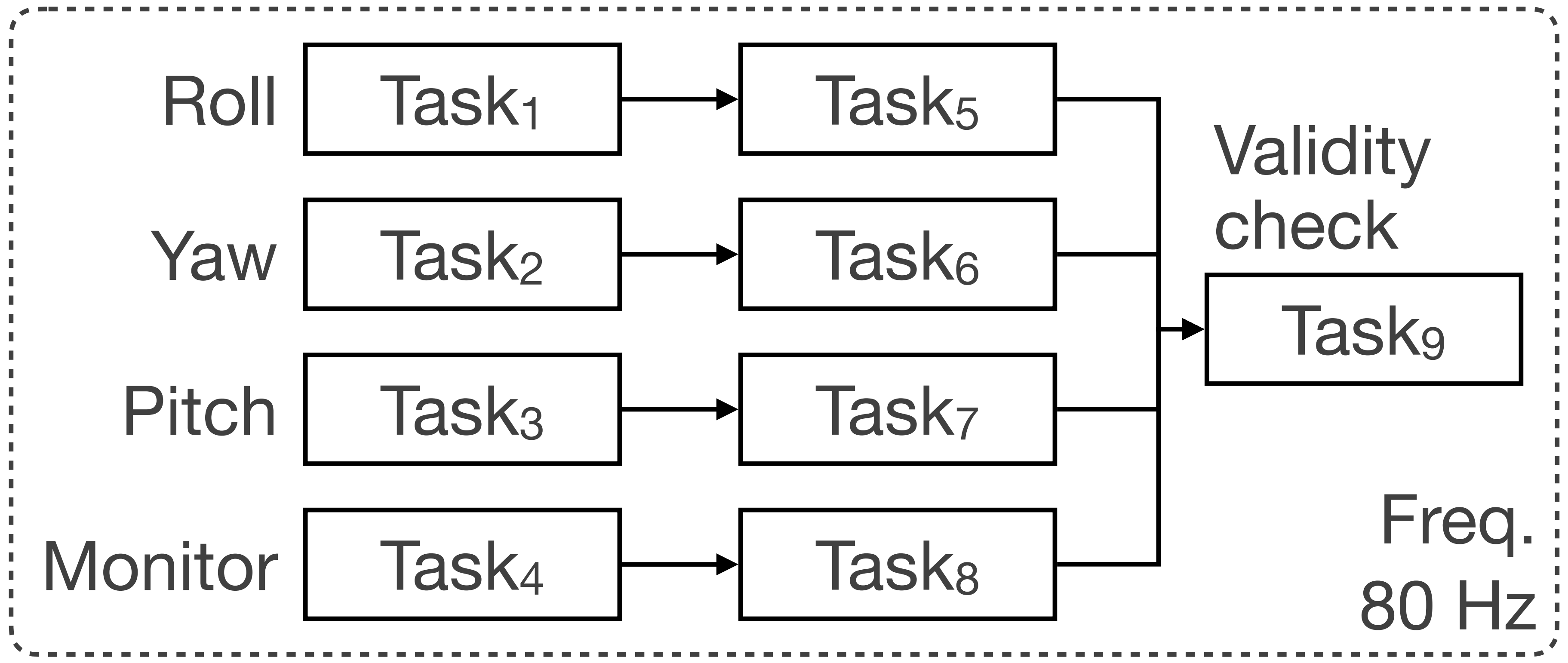
Multiprocessor Architecture for Fault Tolerance (MAFT)*



General-purpose application processors (APs)

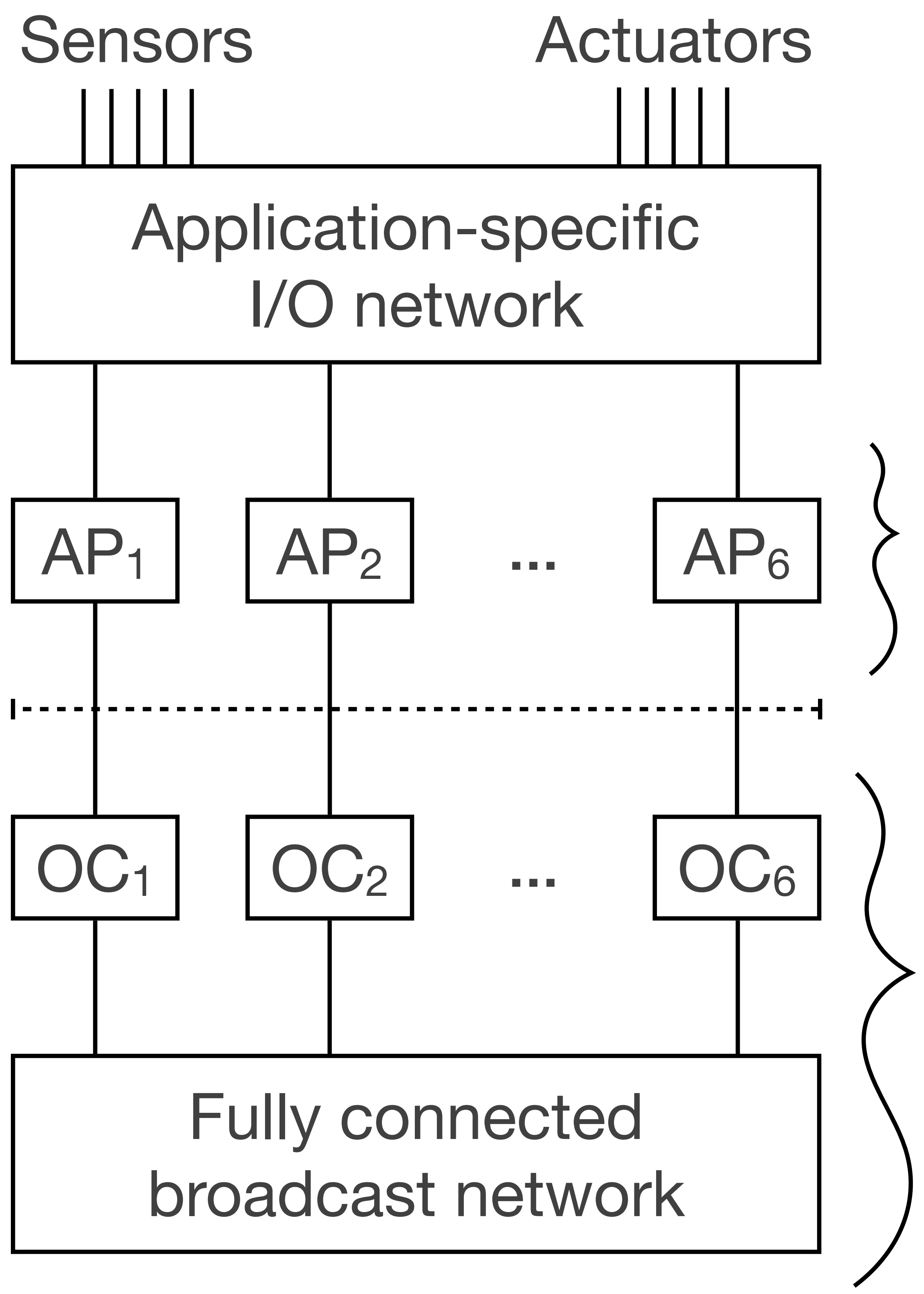
Custom operation controllers (OCs) for voting, scheduling, synchronization, etc.

Simple periodic real-time workload



* Kieckhafer *et al.* “The MAFT Architecture for Distributed Fault Tolerance.” IEEE Transactions on Computers (1988)

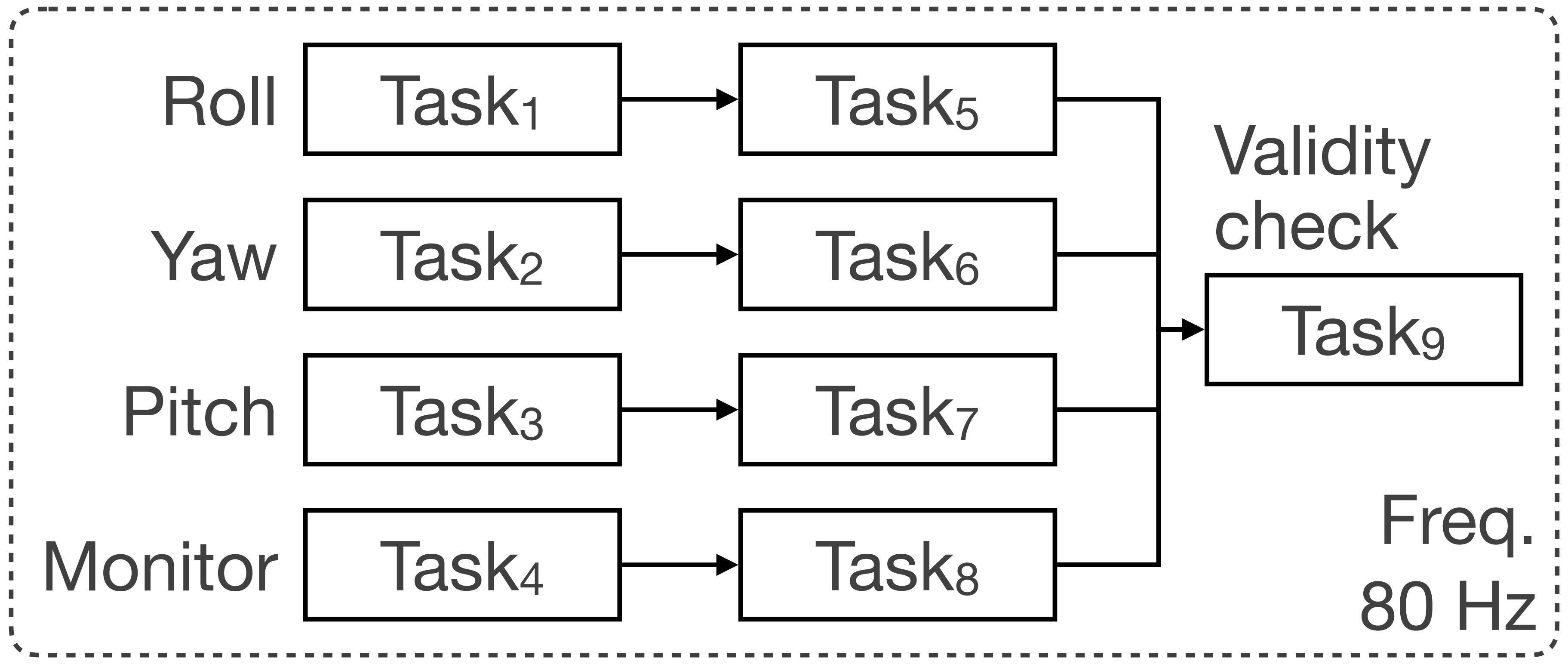
Multiprocessor Architecture for Fault Tolerance (MAFT)*



General-purpose application processors (APs)

Custom operation controllers (OCs) for voting, scheduling, synchronization, etc.

Simple periodic real-time workload



Reliability goals

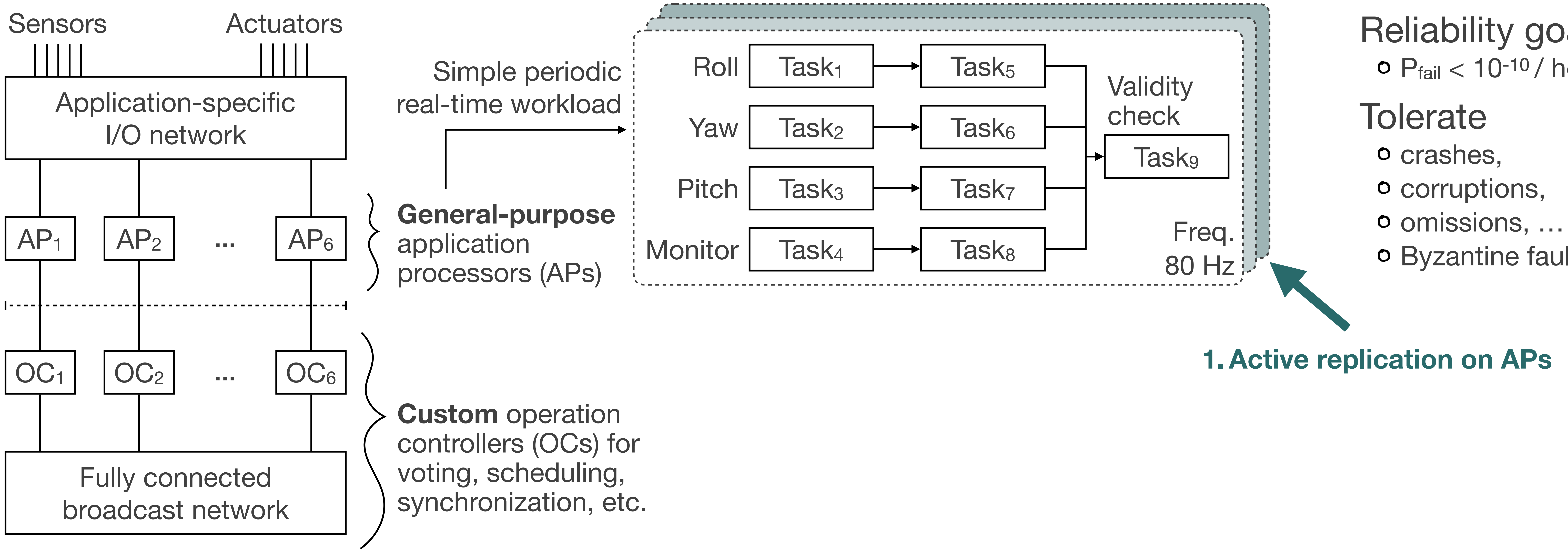
- $P_{fail} < 10^{-10}$ / hour

Tolerate

- crashes,
- corruptions,
- omissions, …
- Byzantine faults

* Kieckhafer *et al.* “The MAFT Architecture for Distributed Fault Tolerance.” IEEE Transactions on Computers (1988)

Multiprocessor Architecture for Fault Tolerance (MAFT)*

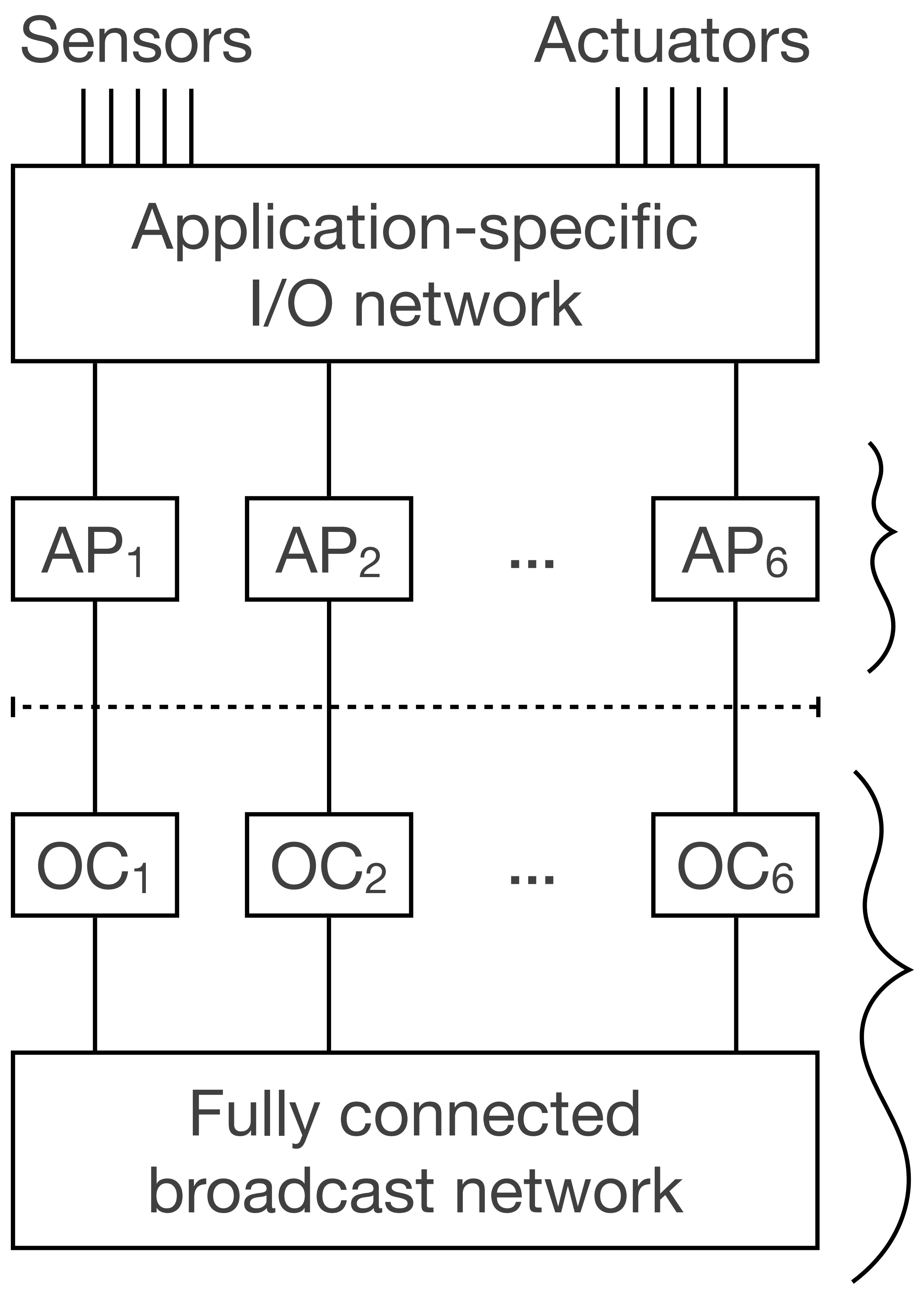


Reliability goals
- $P_{fail} < 10^{-10}$ / hour

Tolerate
- crashes,
- corruptions,
- omissions, …
- Byzantine faults

* Kieckhafer *et al.* “The MAFT Architecture for Distributed Fault Tolerance.” IEEE Transactions on Computers (1988)

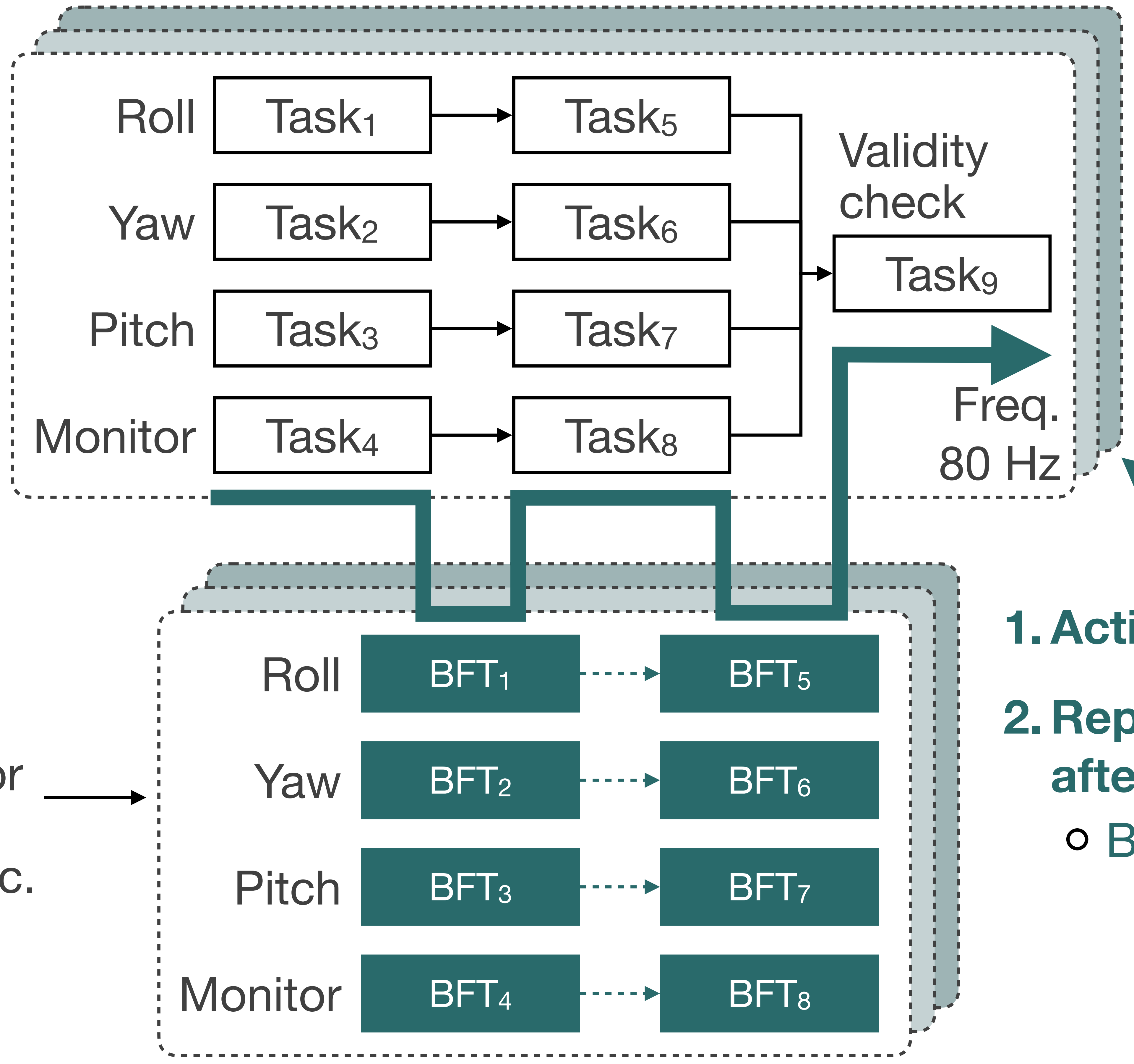
Multiprocessor Architecture for Fault Tolerance (MAFT)*



Simple periodic real-time workload

General-purpose application processors (APs)

Custom operation controllers (OCs) for voting, scheduling, synchronization, etc.



Reliability goals

- $P_{fail} < 10^{-10}$ / hour

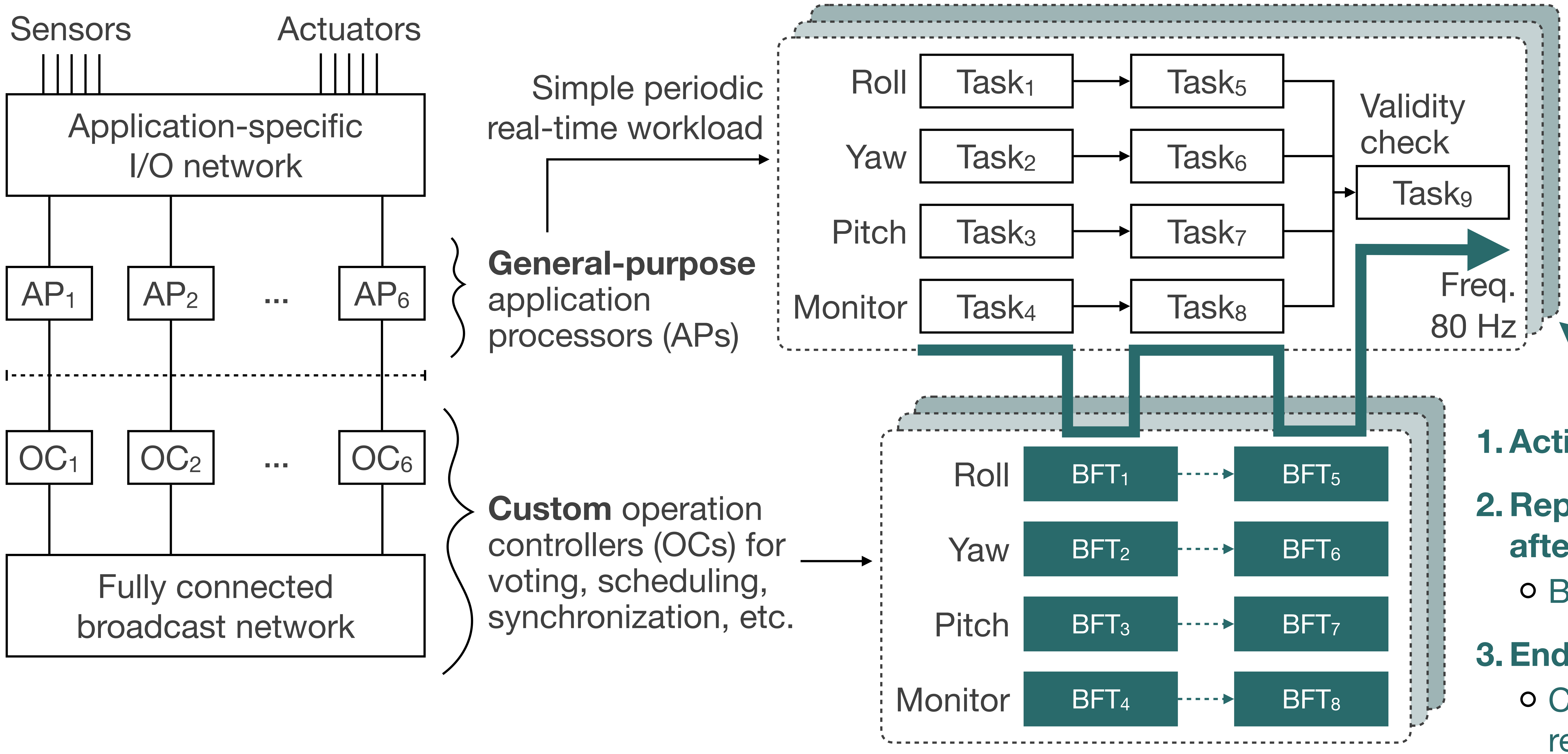
Tolerate

- crashes,
- corruptions,
- omissions, …
- Byzantine faults

1. **Active replication on APs**
2. **Replica coordination on OCs after every task**
   - Byzantine fault tolerance (BFT)

* Kieckhafer *et al.* “The MAFT Architecture for Distributed Fault Tolerance.” IEEE Transactions on Computers (1988)

Multiprocessor Architecture for Fault Tolerance (MAFT)*



Reliability goals

- $P_{fail} < 10^{-10}$ / hour

Tolerate

- crashes,
- corruptions,
- omissions, …
- Byzantine faults

1. **Active replication on APs**
2. **Replica coordination on OCs after every task**
   - Byzantine fault tolerance (BFT)
3. **End-to-end timing analysis**
   - Considering application tasks & replica coordination services

* Kieckhafer *et al.* “The MAFT Architecture for Distributed Fault Tolerance.” IEEE Transactions on Computers (1988)

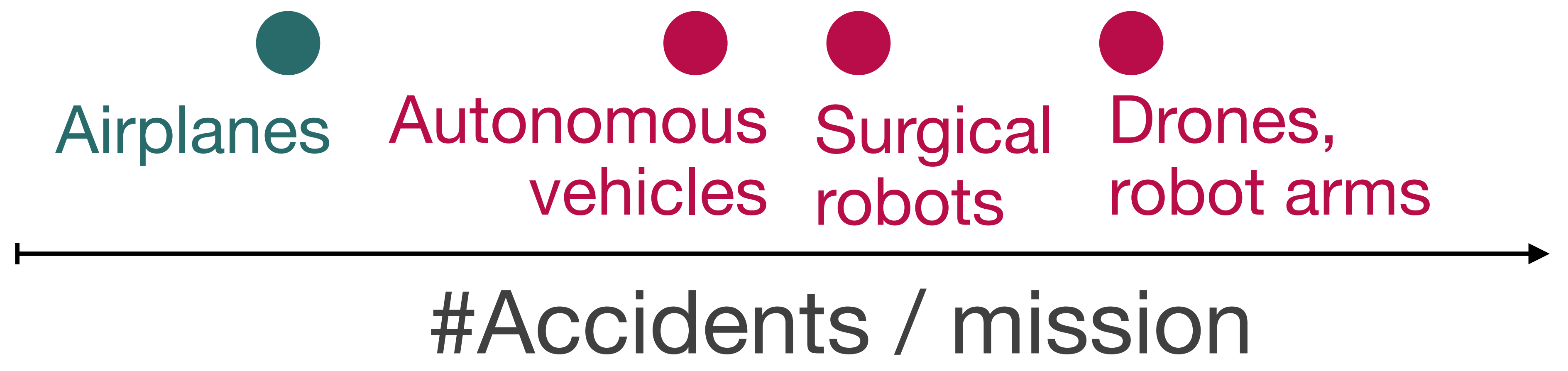
Not all CPS are engineered as reliably as airplanes[1]

[1] Banerjee et al. “Hands Off the Wheel in Autonomous Vehicles?: A Systems Perspective on over a Million Miles of Field Data.” DSN (2018)

Not all CPS are engineered as reliably as airplanes[1]

Contemporary CPS, e.g., autonomous vehicles, drones, robot arms, surgical robots, etc.

- Inexpensive but unreliable commercial off-the-shelf (COTS) hardware
- Inadequate resources (developer hours, computing power, component costs)
- Time to market pressures!

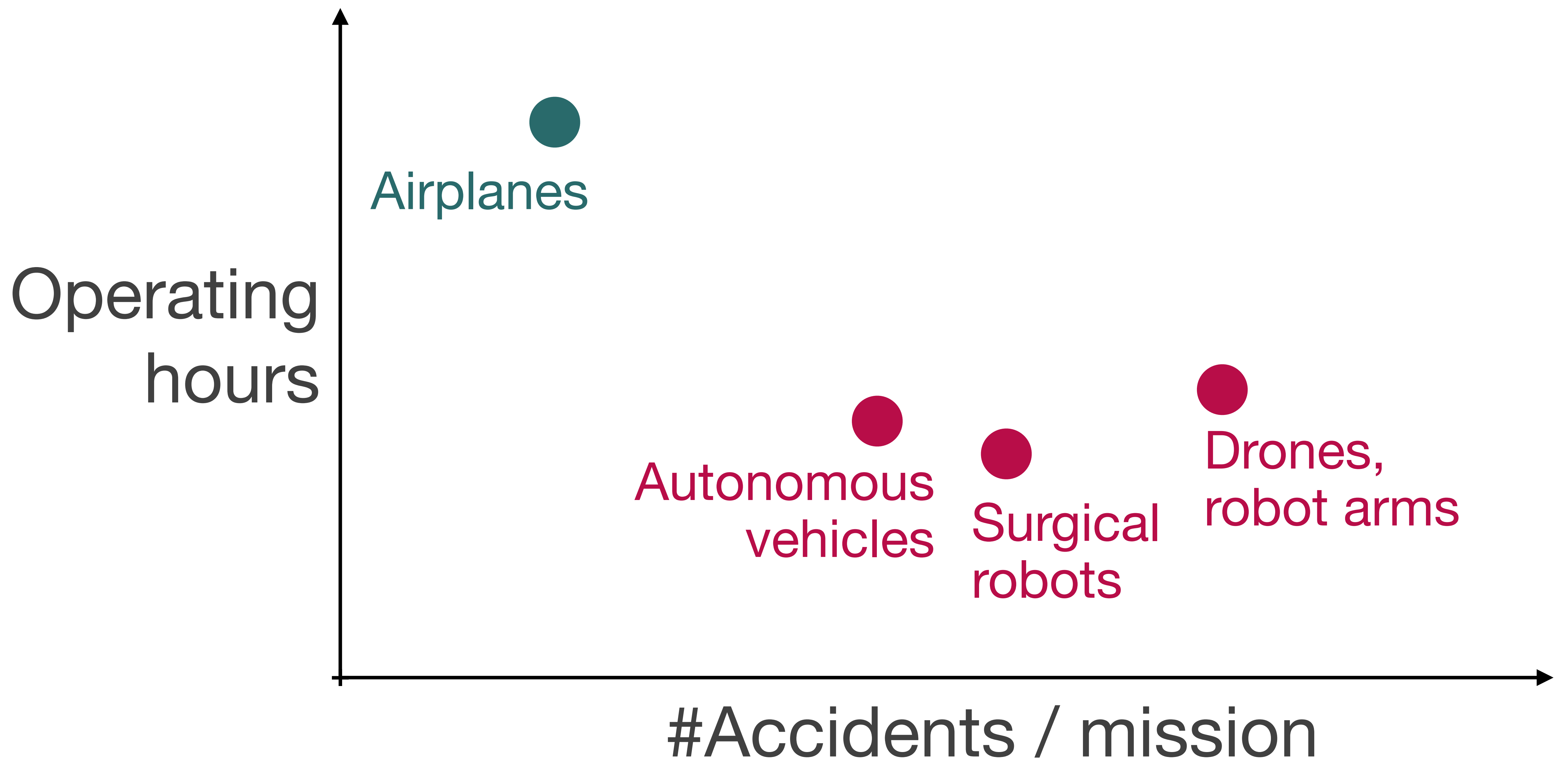


[1] Banerjee et al. “Hands Off the Wheel in Autonomous Vehicles?: A Systems Perspective on over a Million Miles of Field Data.” DSN (2018)

Not all CPS are engineered as reliably as airplanes[1]

Contemporary CPS, e.g., autonomous vehicles, drones, robot arms, surgical robots, etc.

- Inexpensive but unreliable commercial off-the-shelf (COTS) hardware
- Inadequate resources (developer hours, computing power, component costs)
- Time to market pressures!



[1] Banerjee et al. “Hands Off the Wheel in Autonomous Vehicles?: A Systems Perspective on over a Million Miles of Field Data.” DSN (2018)

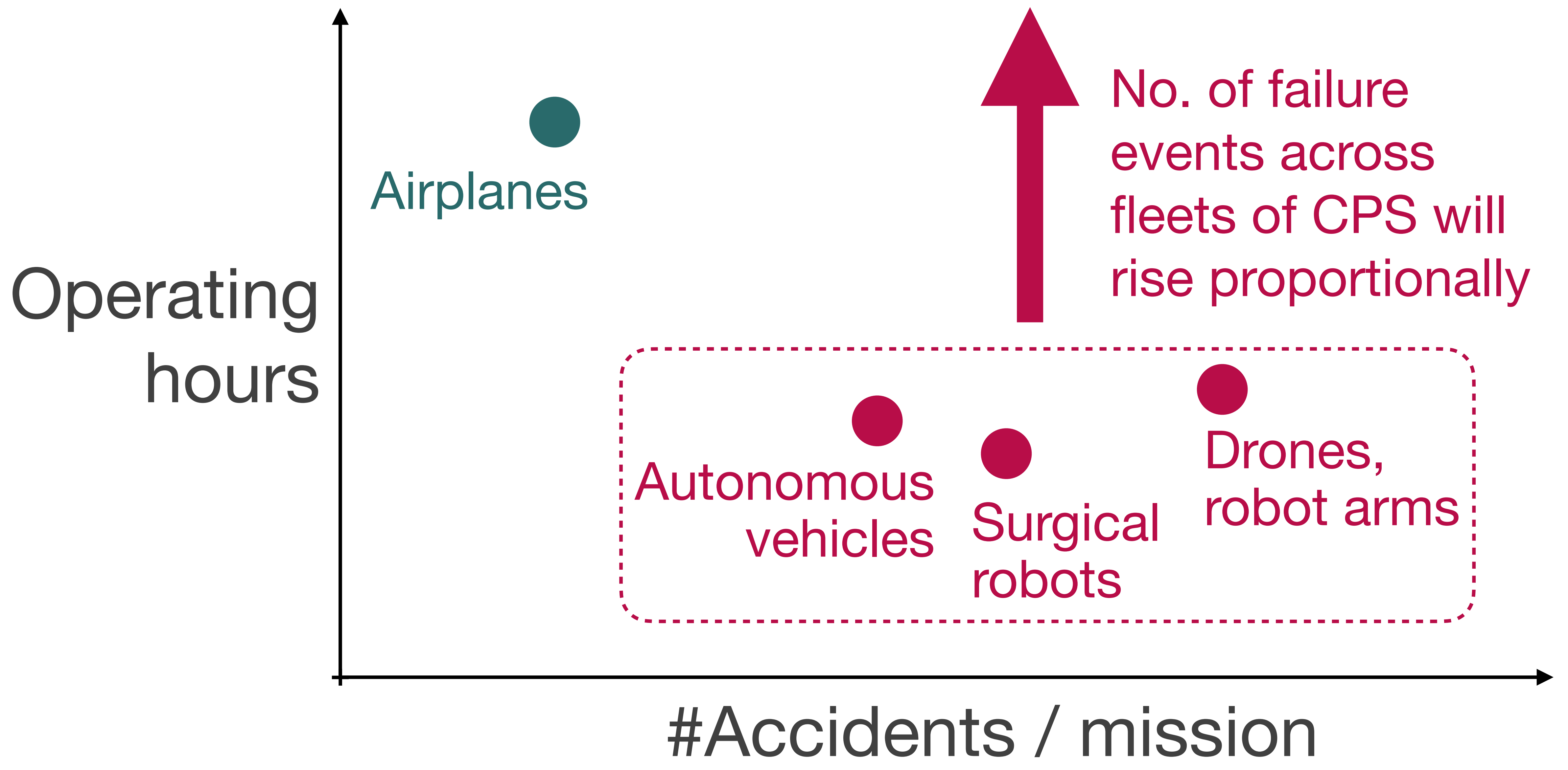
Not all CPS are engineered as reliably as airplanes[1]

Contemporary CPS, e.g., autonomous vehicles, drones, robot arms, surgical robots, etc.

- Inexpensive but unreliable commercial off-the-shelf (COTS) hardware
- Inadequate resources (developer hours, computing power, component costs)
- Time to market pressures!

As these CPS permeate our everyday lives

- … their cumulative operating times are increasing[2]



[1] Banerjee et al. “Hands Off the Wheel in Autonomous Vehicles?: A Systems Perspective on over a Million Miles of Field Data.” DSN (2018)

[2] SESAR Joint Undertaking. “European Drones Outlook Study-Unlocking the value for Europe.” SESAR, Brussels (2016)

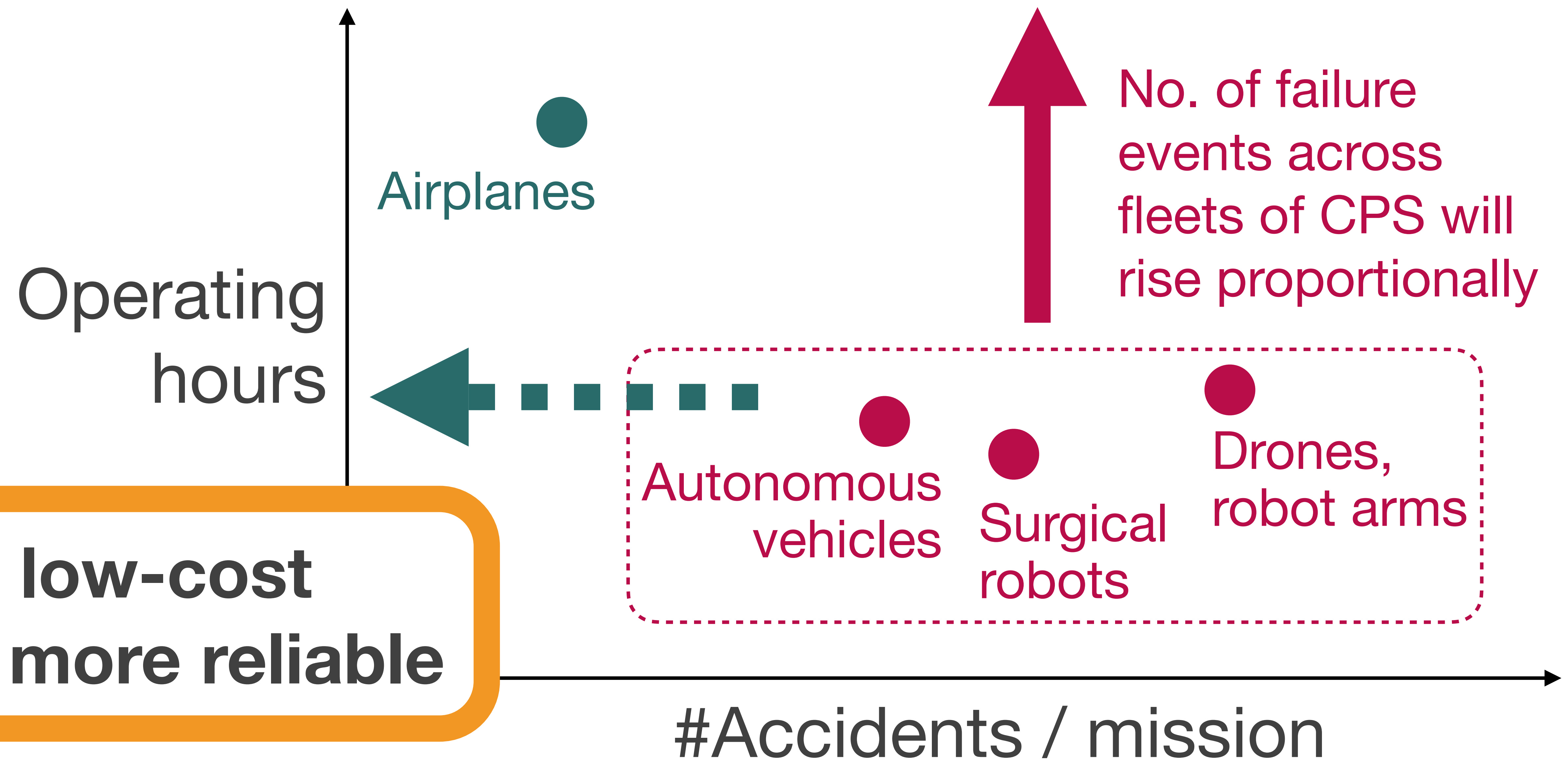
Not all CPS are engineered as reliably as airplanes[1]

Contemporary CPS, e.g., autonomous vehicles, drones, robot arms, surgical robots, etc.

- Inexpensive but unreliable commercial off-the-shelf (COTS) hardware
- Inadequate resources (developer hours, computing power, component costs)
- Time to market pressures!

As these CPS permeate our everyday lives

- … their cumulative operating times are increasing[2]



Goal: Make low-cost consumer CPS more reliable

[1] Banerjee et al. "Hands Off the Wheel in Autonomous Vehicles?: A Systems Perspective on over a Million Miles of Field Data." DSN (2018)

[2] SESAR Joint Undertaking. "European Drones Outlook Study-Unlocking the value for Europe." SESAR, Brussels (2016)

Not all CPS are engineered as reliably as airplanes[1]

Contemporary CPS, e.g., autonomous vehicles, drones, robot arms, surgical robots, etc.

- Inexpensive but unreliable commercial off-the-shelf (COTS) hardware
- Inadequate resources (developer hours, computing power, component costs)
- Time to market pressures!

As these CPS permeate our everyday lives

- … their cumulative operating times are increasing[2]

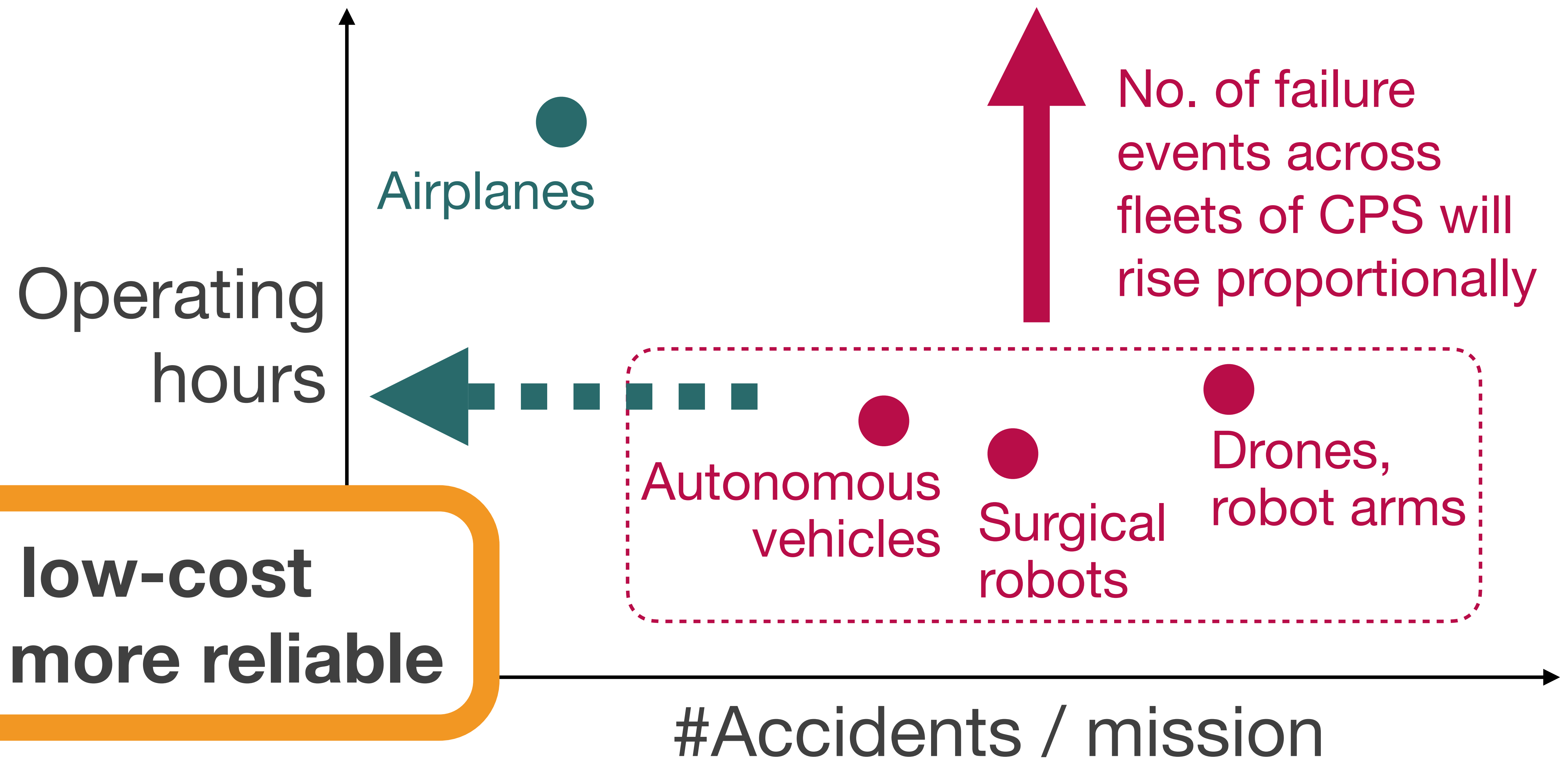
Reliability goals

- $P_{fail} < 10^{-10}$ / hour

Tolerate

- crashes,
- corruptions,
- omissions, …
- Byzantine faults

Goal: Make low-cost consumer CPS more reliable



[1] Banerjee et al. "Hands Off the Wheel in Autonomous Vehicles?: A Systems Perspective on over a Million Miles of Field Data." DSN (2018)

[2] SESAR Joint Undertaking. "European Drones Outlook Study-Unlocking the value for Europe." SESAR, Brussels (2016)

Not all CPS are engineered as reliably as airplanes[1]

Contemporary CPS, e.g., autonomous vehicles, drones, robot arms, surgical robots, etc.

- Inexpensive but unreliable commercial off-the-shelf (COTS) hardware
- Inadequate resources (developer hours, computing power, component costs)

Like MAFT, can we realize
<u>replica coordination with Byzantine fault tolerance</u> (BFT)
for real-time workloads in low-cost consumer CPS?

- corruptions,
- omissions, …
- Byzantine faults

Goal: Make low-cost consumer CPS more reliable

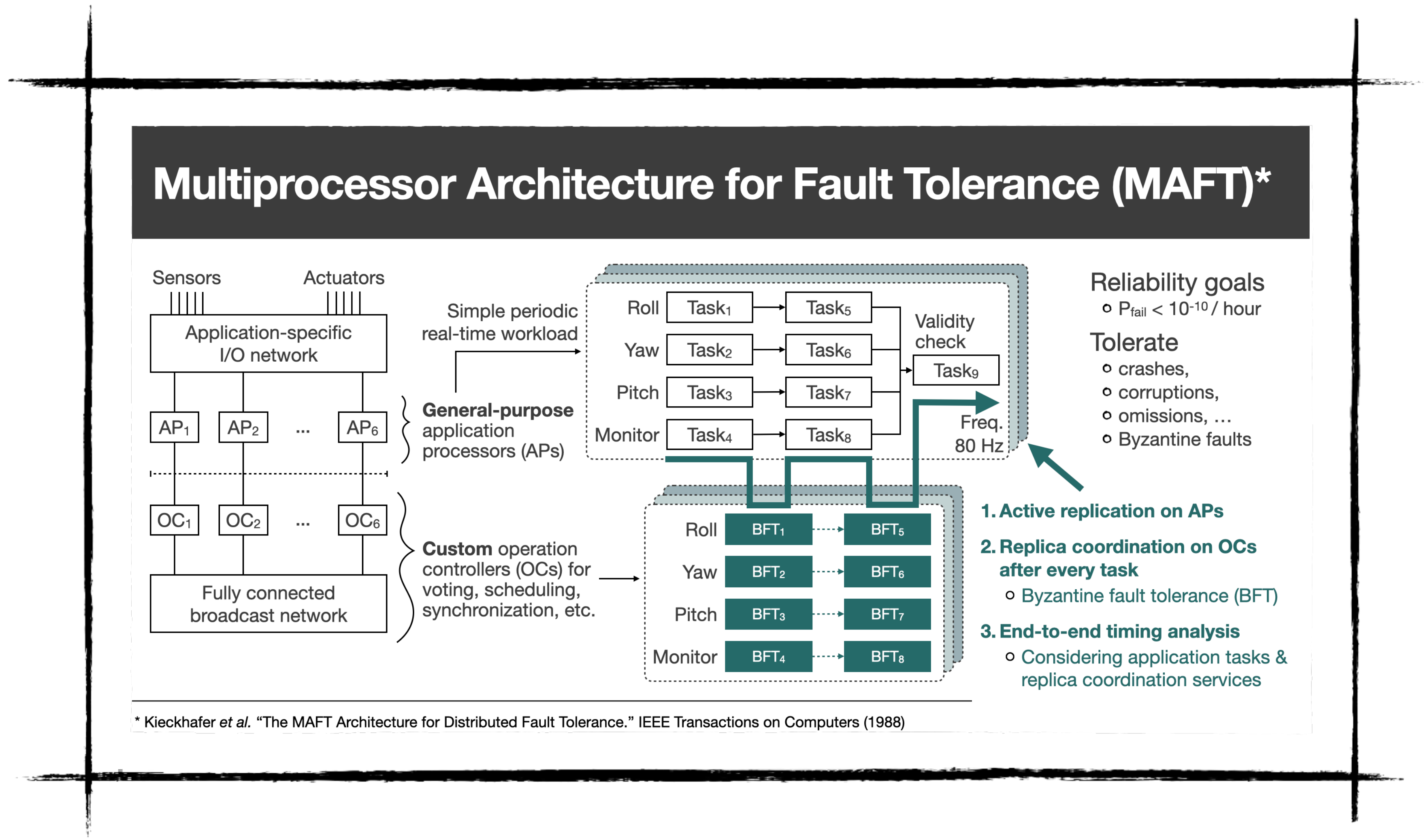
robots

#Accidents / mission

[1] Banerjee et al. "Hands Off the Wheel in Autonomous Vehicles?: A Systems Perspective on over a Million Miles of Field Data." DSN (2018)

[2] SESAR Joint Undertaking. "European Drones Outlook Study-Unlocking the value for Europe." SESAR, Brussels (2016)

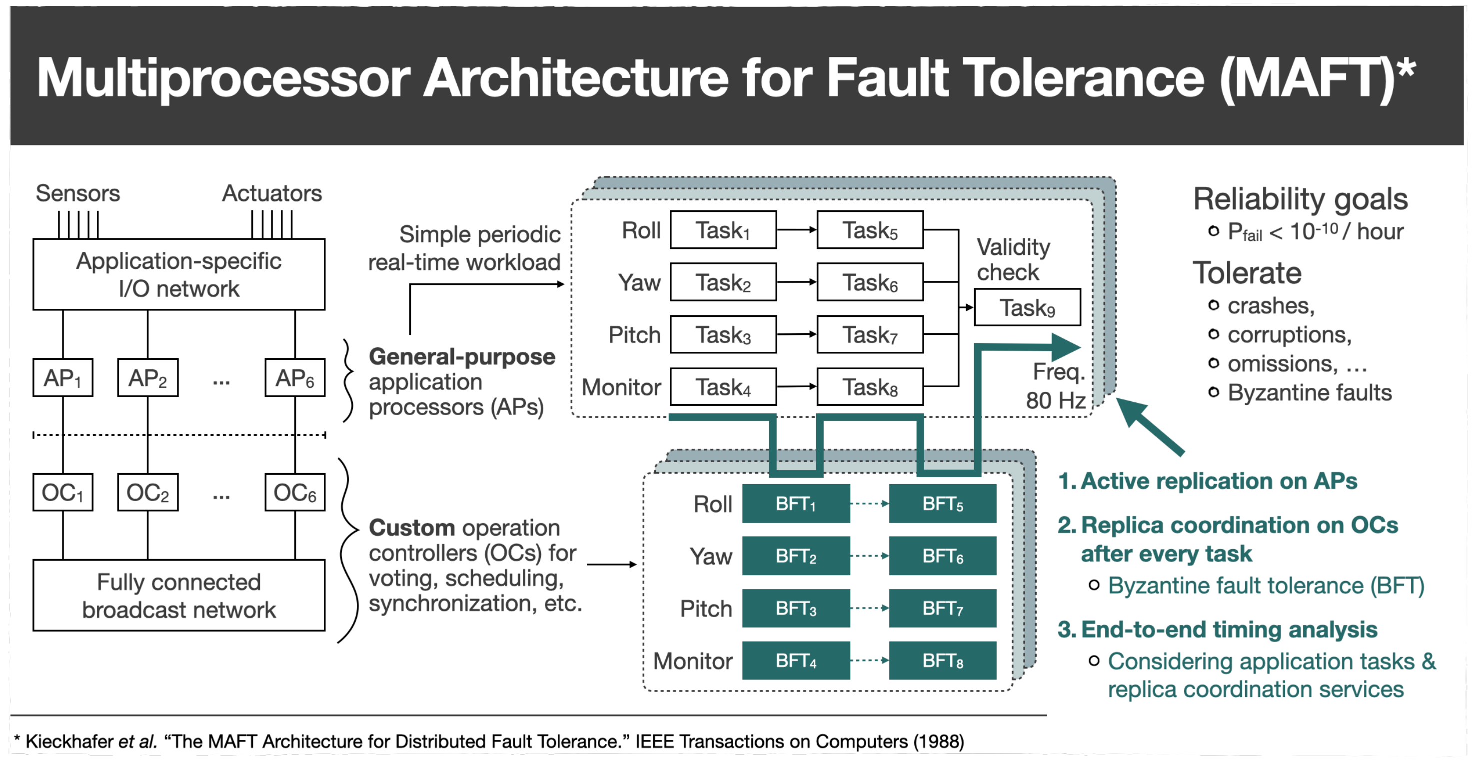
BFT Replica Coordination for CPS



Few domains use custom hardware

Let alone for Byzantine fault tolerance

BFT Replica Coordination for CPS



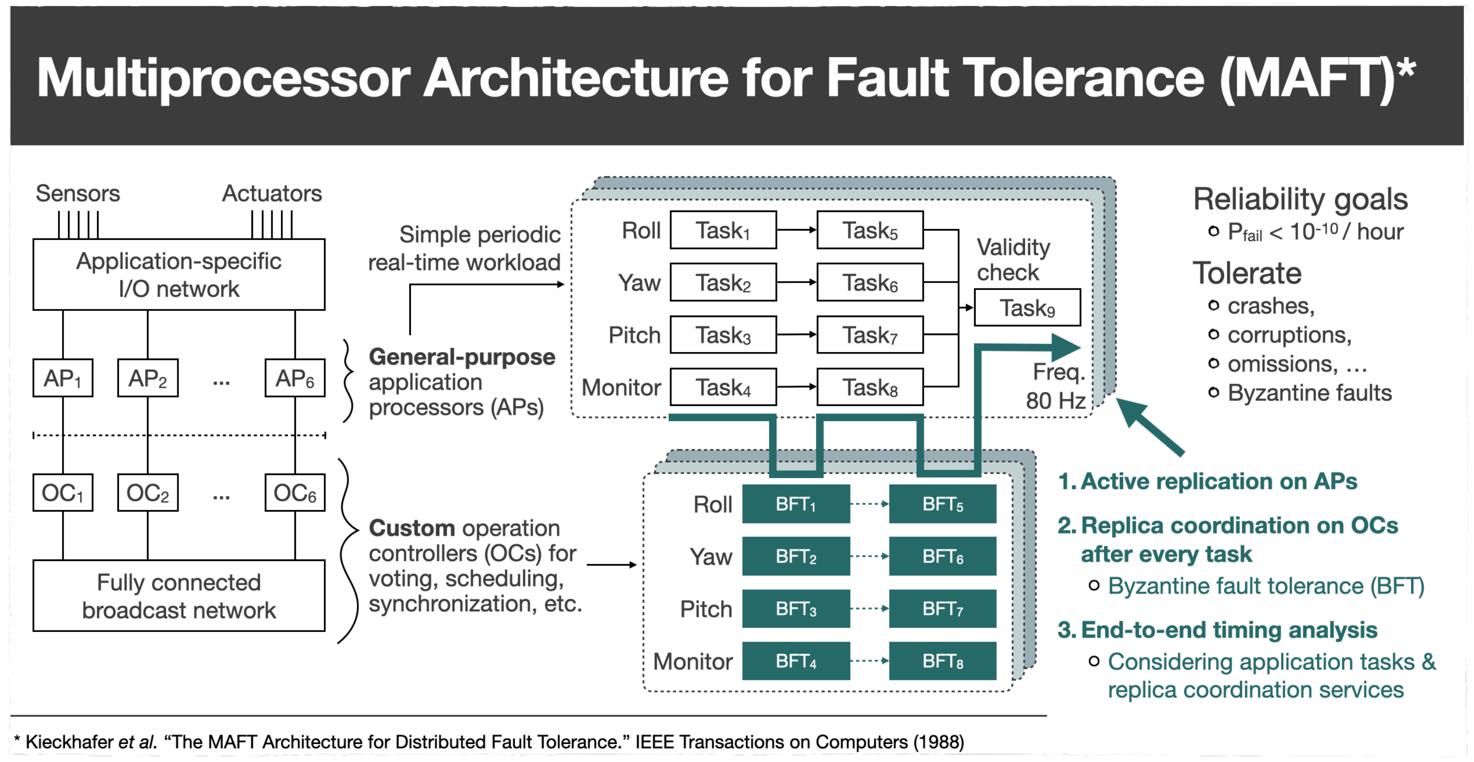
Few domains use custom hardware

Let alone for Byzantine fault tolerance

Easy to build a drone using off-the-shelf hardware and open-source software

But no BFT solutions that can be retrofitted onto these real-time platforms

BFT Replica Coordination for CPS

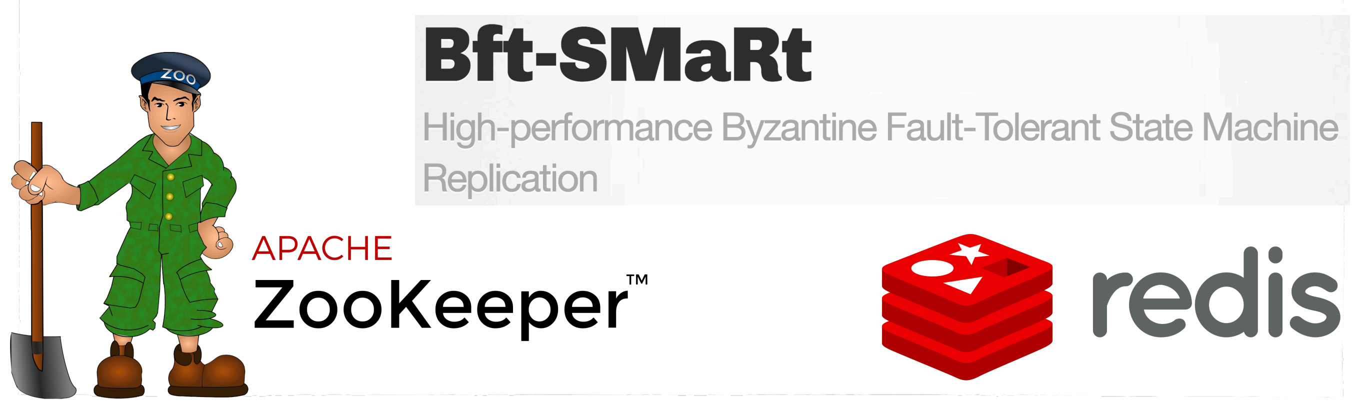


Few domains use custom hardware

Let alone for Byzantine fault tolerance

Easy to build a drone using off-the-shelf hardware and open-source software

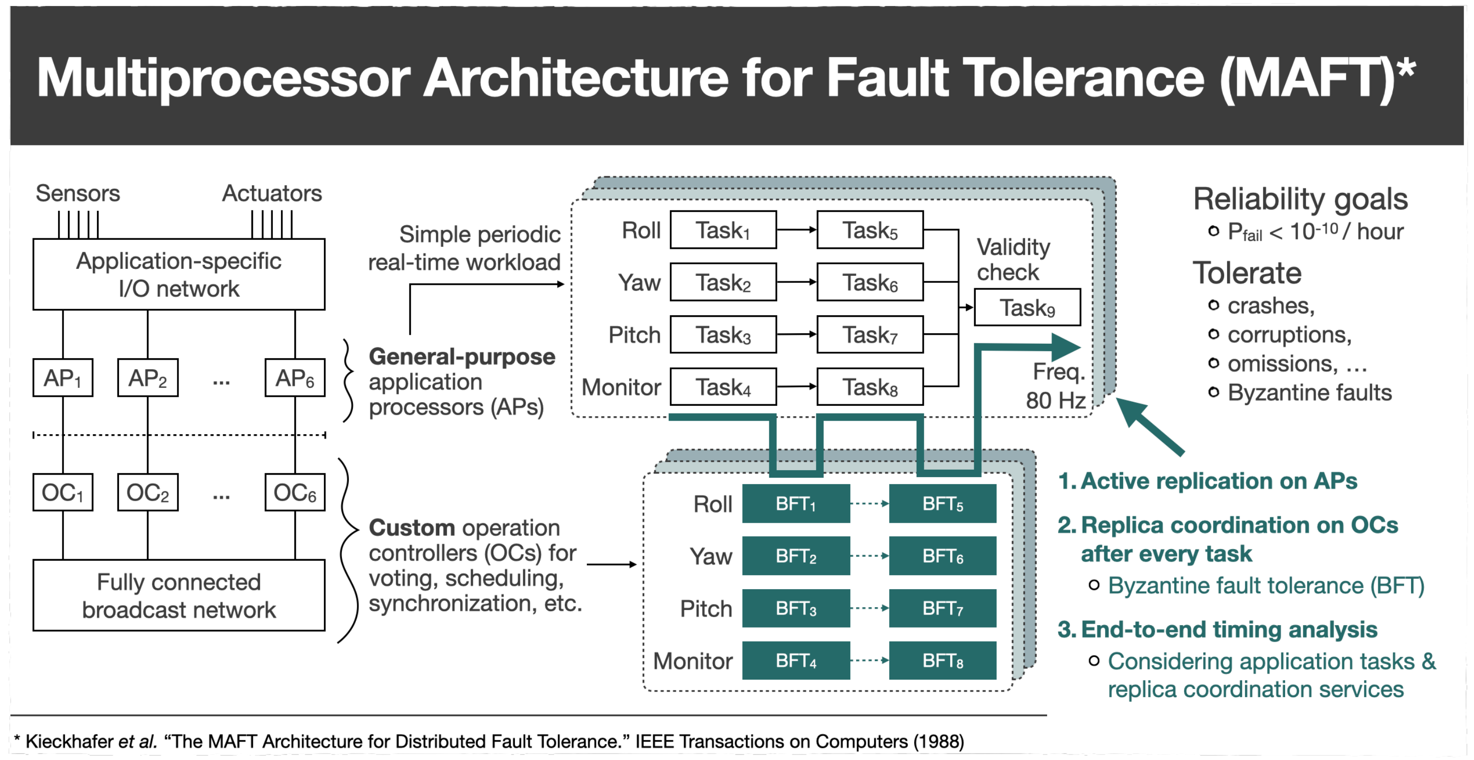
But no BFT solutions that can be retrofitted onto these real-time platforms



Best-effort, throughput-oriented BFT libraries are not suitable

Performance suffers on resource-constrained embedded device or in terms of predictability

BFT Replica Coordination for CPS



Few domains use custom hardware

Let alone for Byzantine fault tolerance

1. Focused on **BFT** with **real-time predictability**

Easy to build a drone using off-the-shelf hardware and open-source software

But no BFT solutions that can be retrofitted onto these real-time platforms

2. Deployable on **COTS platforms** like Raspberry Pis and Ethernet

Bft-SMaRt
High-performance Byzantine Fault-Tolerant State Machine Replication
APACHE ZooKeeper™
redis

In-ConcReTeS

Best-effort, throughput-oriented BFT libraries are not suitable

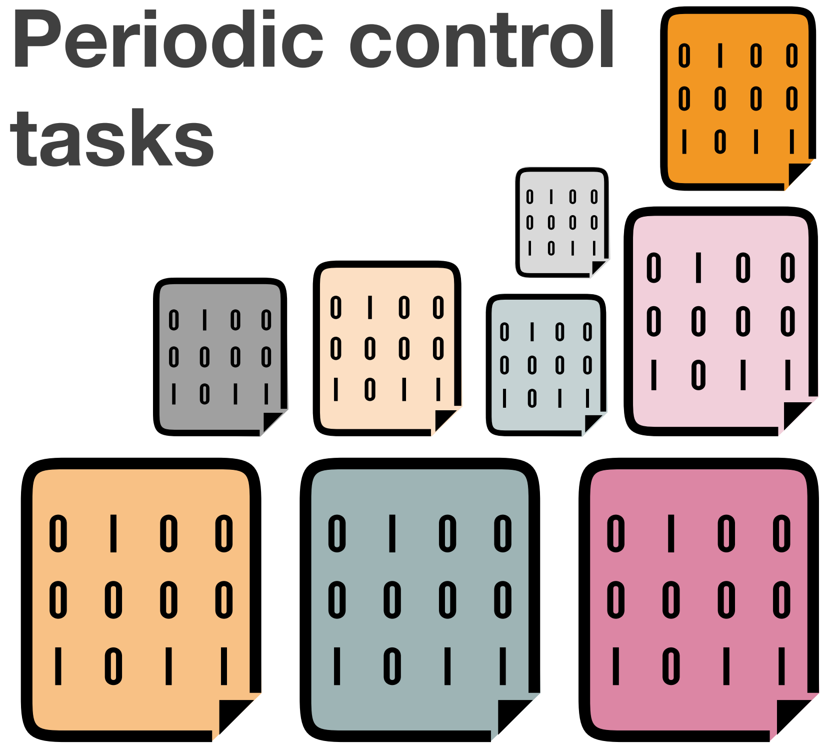
Performance suffers on resource-constrained embedded device or in terms of predictability

3. **Easy to integrate** with existing control applications

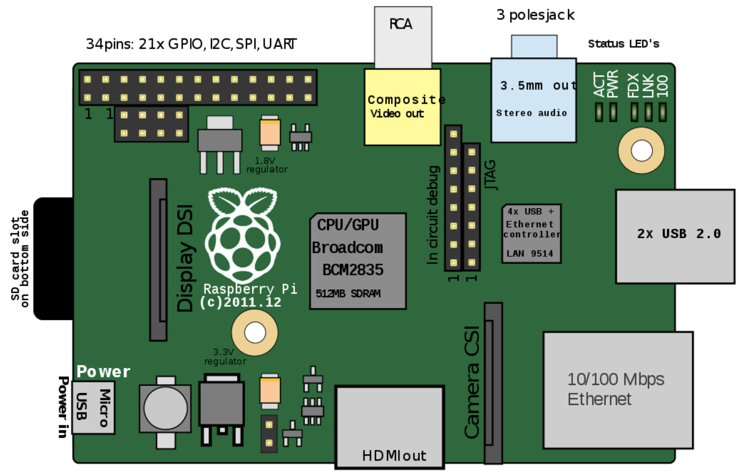
Design and Implementation

Overview

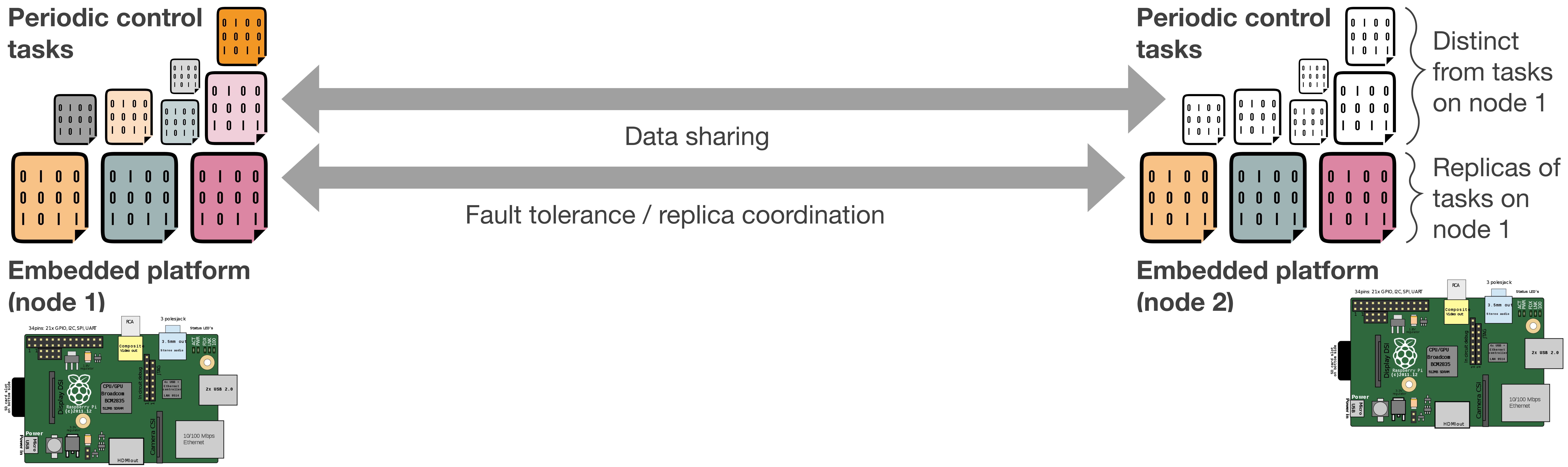
Periodic control tasks



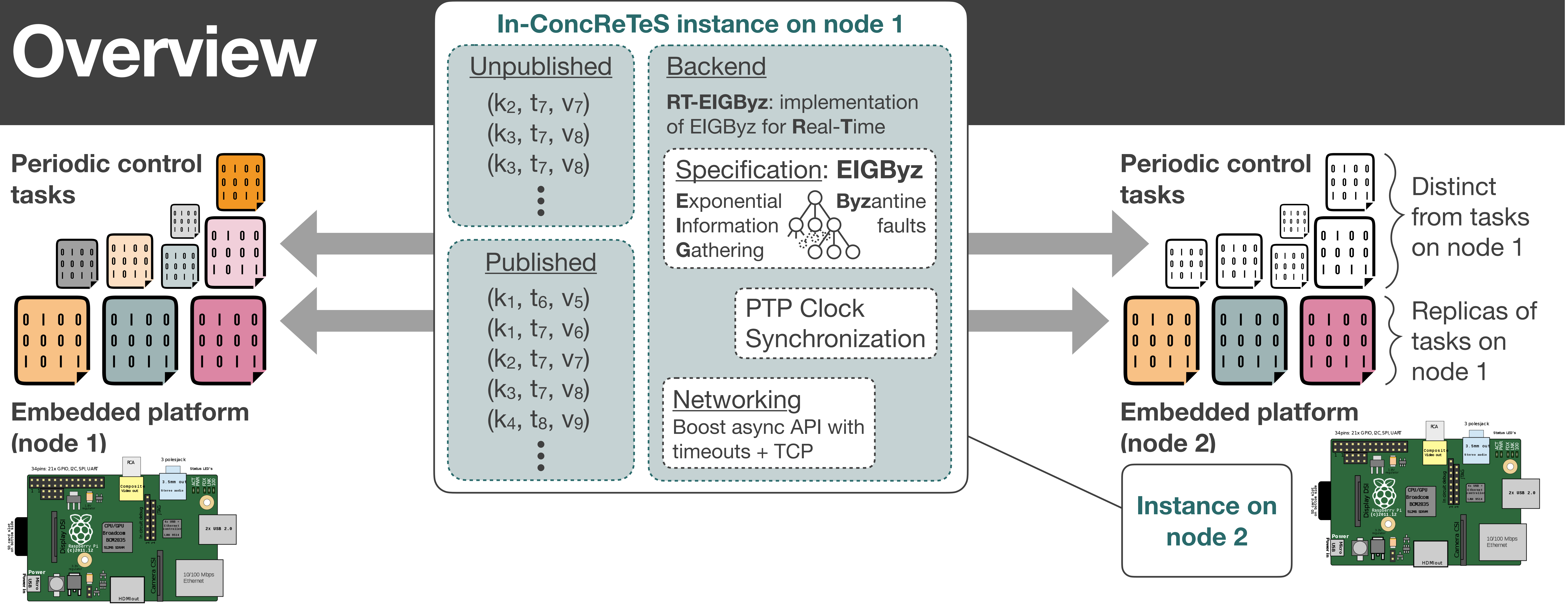
Embedded platform (node 1)



Overview

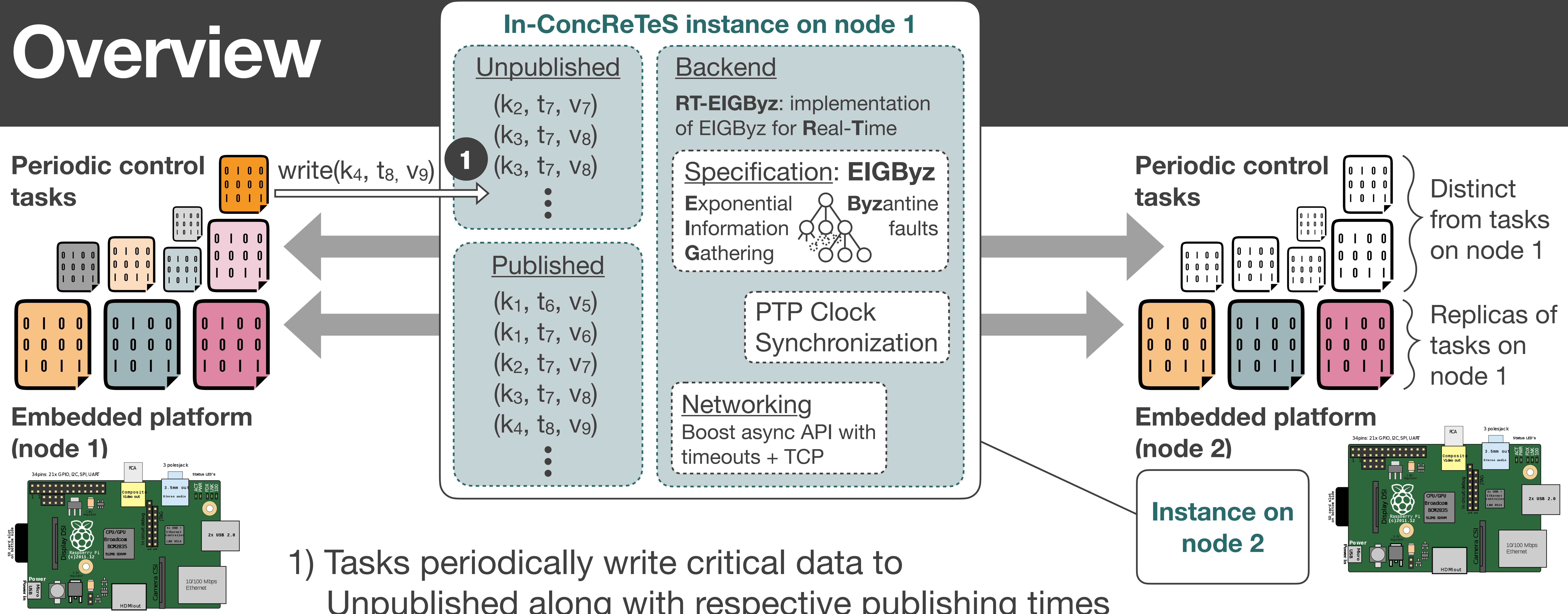

Periodic control tasks
Embedded platform (node 1)
Data sharing
Fault tolerance / replica coordination
Periodic control tasks
Distinct from tasks on node 1
Replicas of tasks on node 1
Embedded platform (node 2)

Overview



In-ConcReTeS instance on node 1
Unpublished
(k_2, t_7, v_7)
(k_3, t_7, v_8)
(k_3, t_7, v_8)
⋮
Published
(k_1, t_6, v_5)
(k_1, t_7, v_6)
(k_2, t_7, v_7)
(k_3, t_7, v_8)
(k_4, t_8, v_9)
⋮
Backend
RT-EIGByz: implementation of EIGByz for **R**eal-**T**ime
Specification: **EIGByz**
Exponential **I**nformation **G**athering
Byzantine faults
PTP Clock Synchronization
Networking
Boost async API with timeouts + TCP
Periodic control tasks
Embedded platform (node 1)
Periodic control tasks
Distinct from tasks on node 1
Replicas of tasks on node 1
Embedded platform (node 2)
Instance on node 2

Overview



1) Tasks periodically write critical data to
Unpublished along with respective publishing times

Overview

In-ConcReTeS instance on node 1

Periodic control tasks

write(k_4, t_8, v_9)

Unpublished

(k_2, t_7, v_7)
(k_3, t_7, v_8)
(k_3, t_7, v_8)
⋮

Published

(k_1, t_6, v_5)
(k_1, t_7, v_6)
(k_2, t_7, v_7)
(k_3, t_7, v_8)
(k_4, t_8, v_9)
⋮

Backend

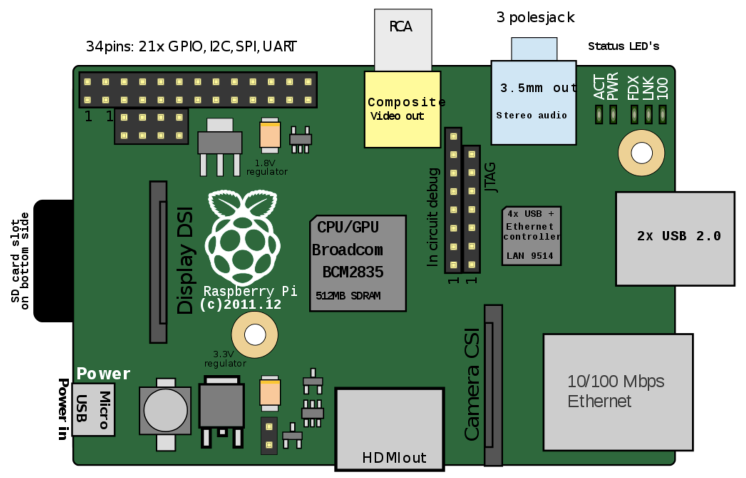
RT-EIGByz: implementation of EIGByz for **R**eal-**T**ime

Specification: **EIGByz**
Exponential **I**nformation **G**athering
Byzantine faults

PTP Clock Synchronization

Networking
Boost async API with timeouts + TCP

Embedded platform (node 1)



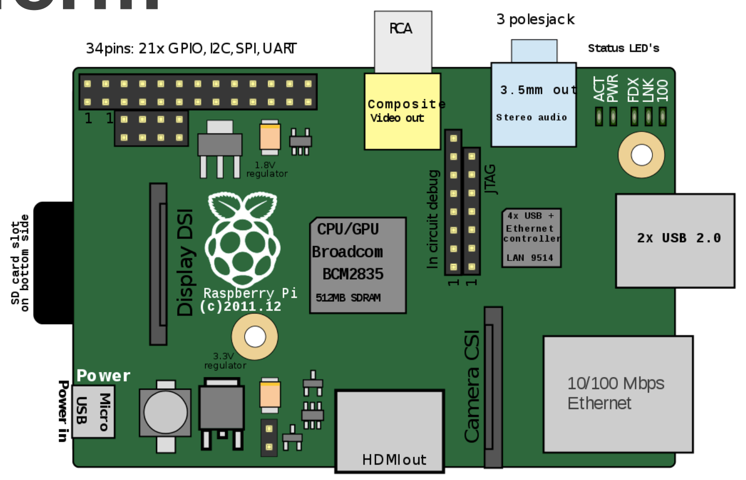
Periodic control tasks

Distinct from tasks on node 1

Replicas of tasks on node 1

Embedded platform (node 2)

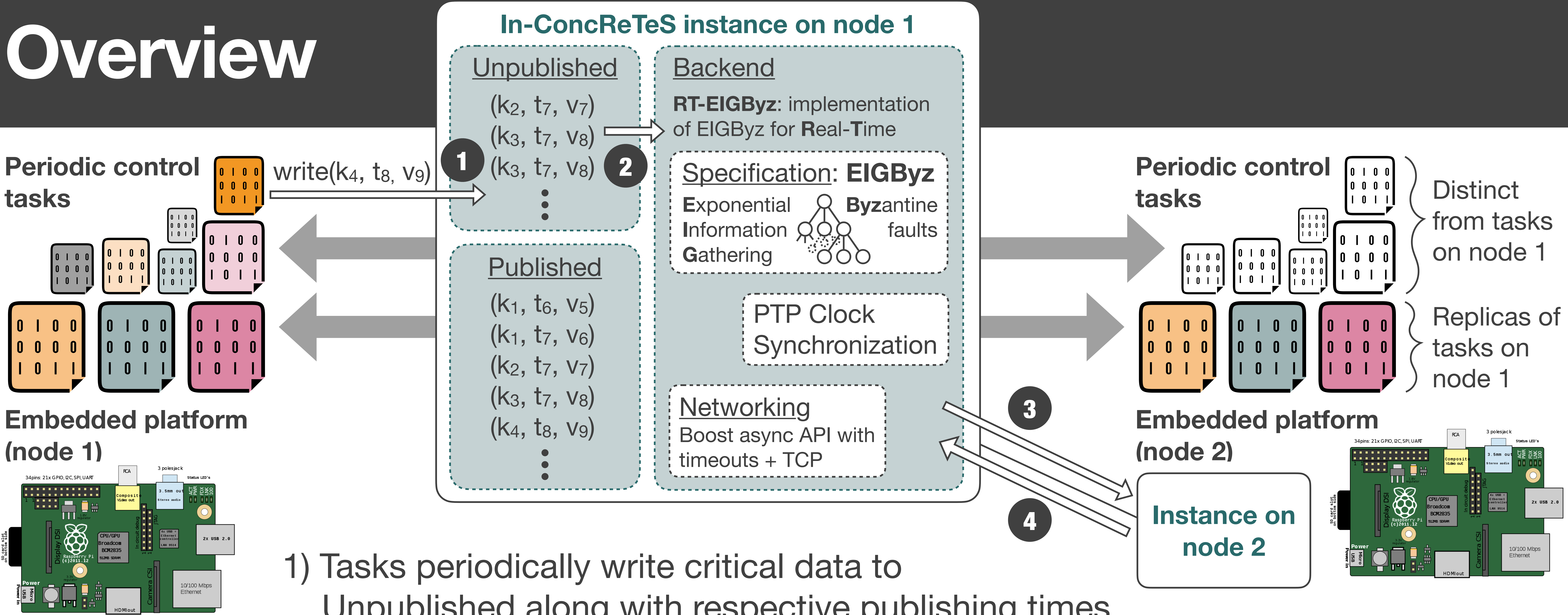
Instance on node 2



1) Tasks periodically write critical data to
Unpublished along with respective publishing times

2) Backend periodically empties Unpublished

Overview

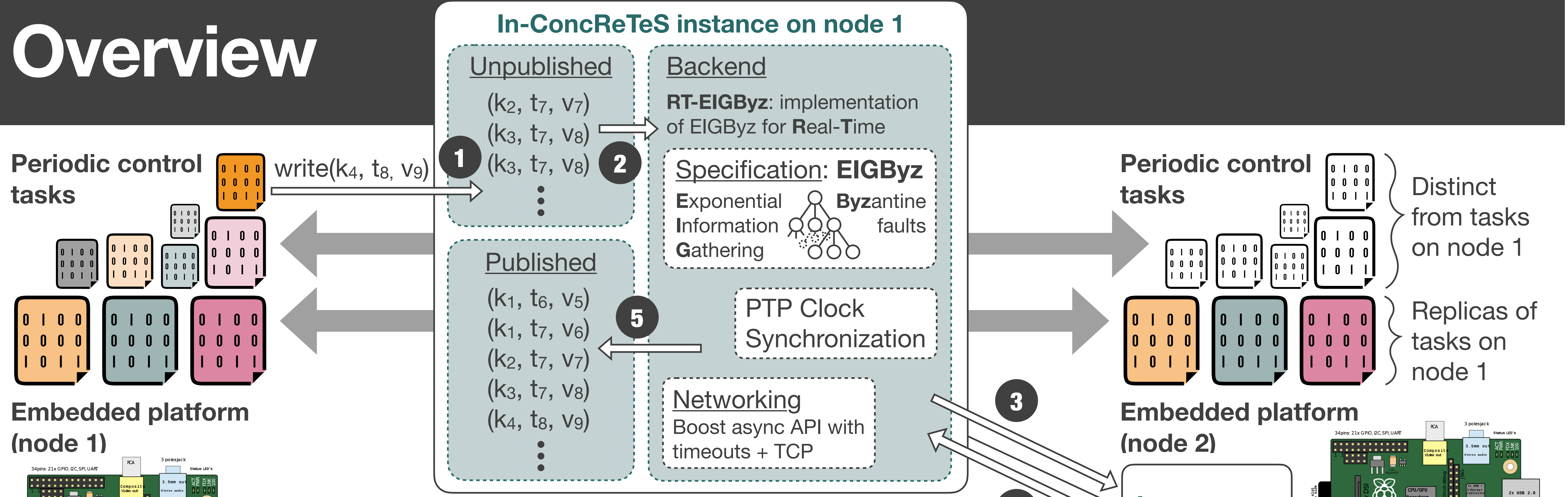


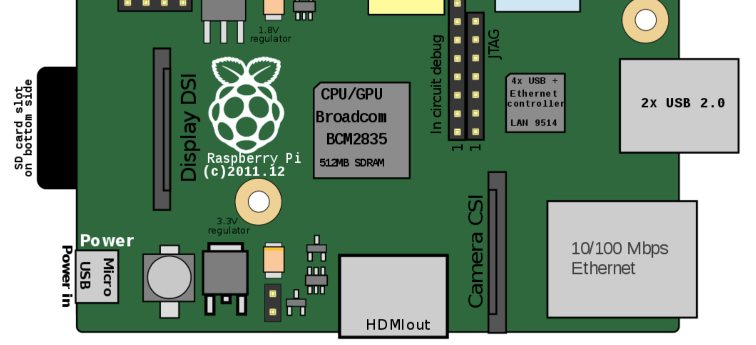
1) Tasks periodically write critical data to Unpublished along with respective publishing times

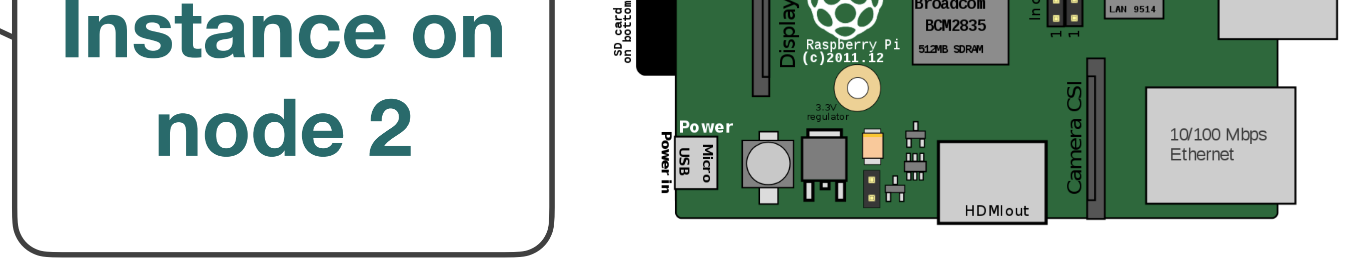
2) Backend periodically empties Unpublished

3, 4) Backend coordinates with remote nodes so that all replicas agree on the same set of values

Overview







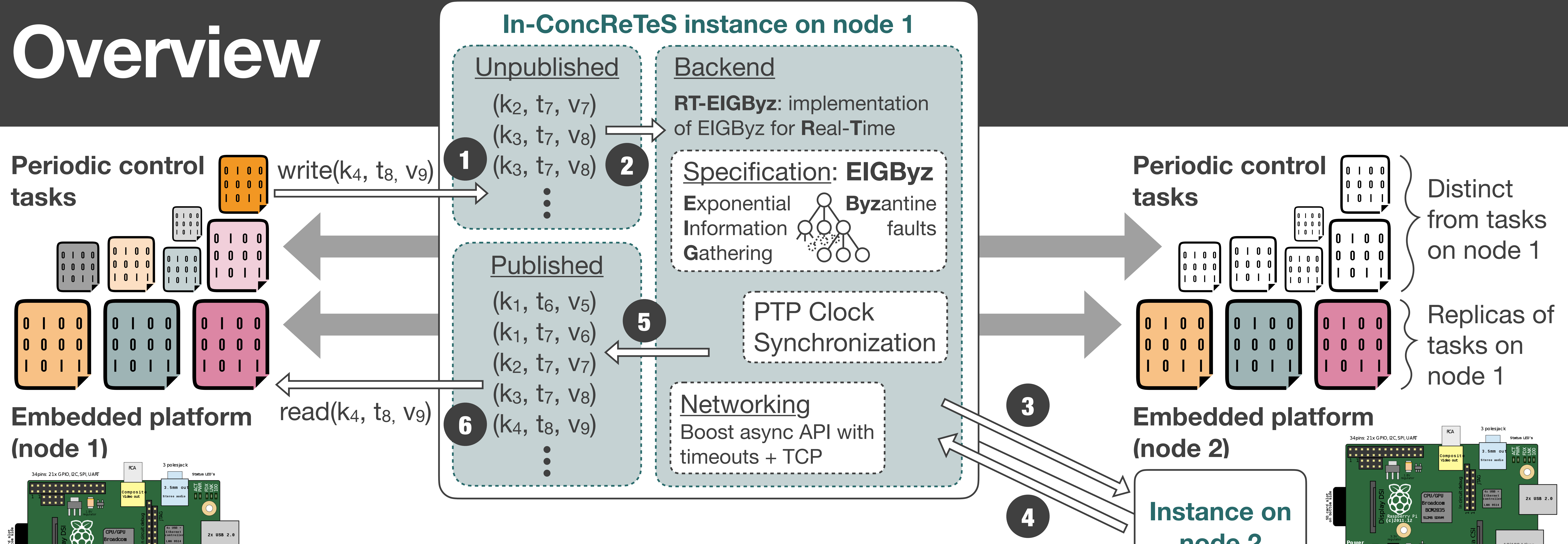
1) Tasks periodically write critical data to
Unpublished along with respective publishing times

2) Backend periodically empties Unpublished

3, 4) Backend coordinates with remote nodes so that all
replicas agree on the same set of values

5) Backend adds final “agreed-upon-by-all” values to Published

Overview



1) Tasks periodically write critical data to Unpublished along with respective publishing times

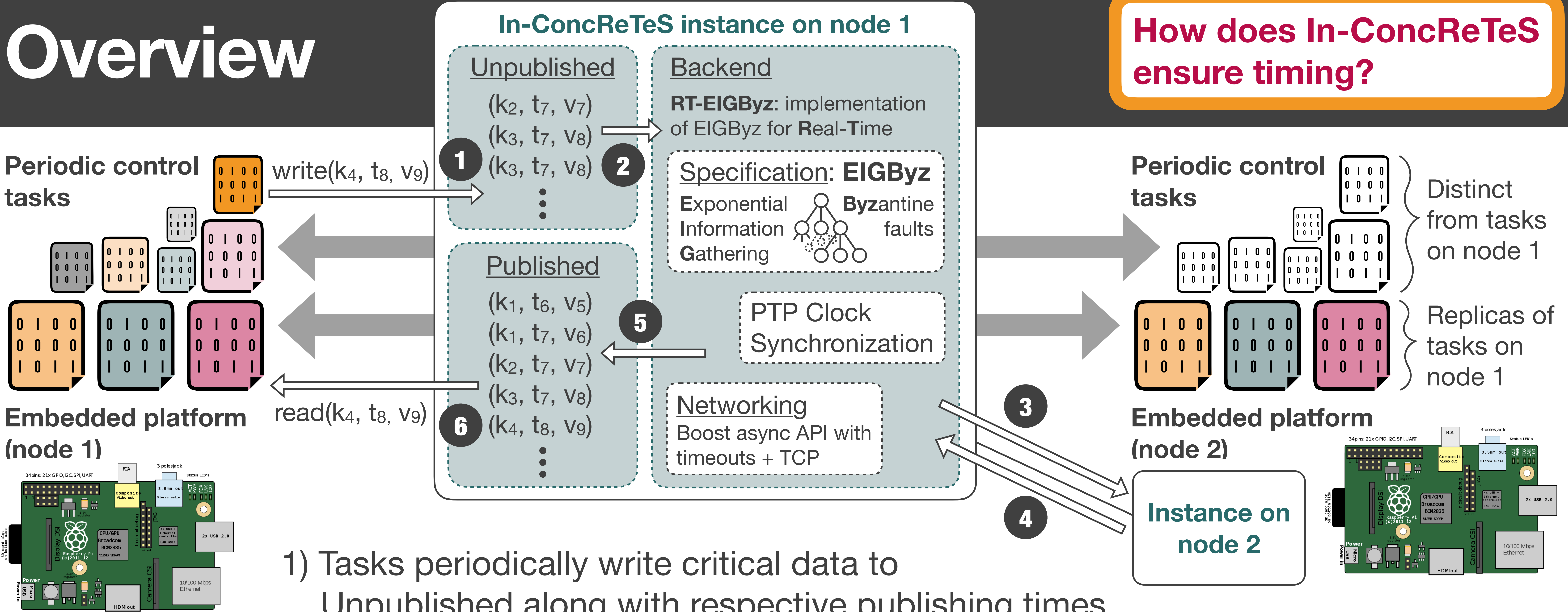
2) Backend periodically empties Unpublished

3, 4) Backend coordinates with remote nodes so that all replicas agree on the same set of values

5) Backend adds final “agreed-upon-by-all” values to Published

6) Tasks read values from Published in their next periodic iteration

Overview



1) Tasks periodically write critical data to Unpublished along with respective publishing times

2) Backend periodically empties Unpublished

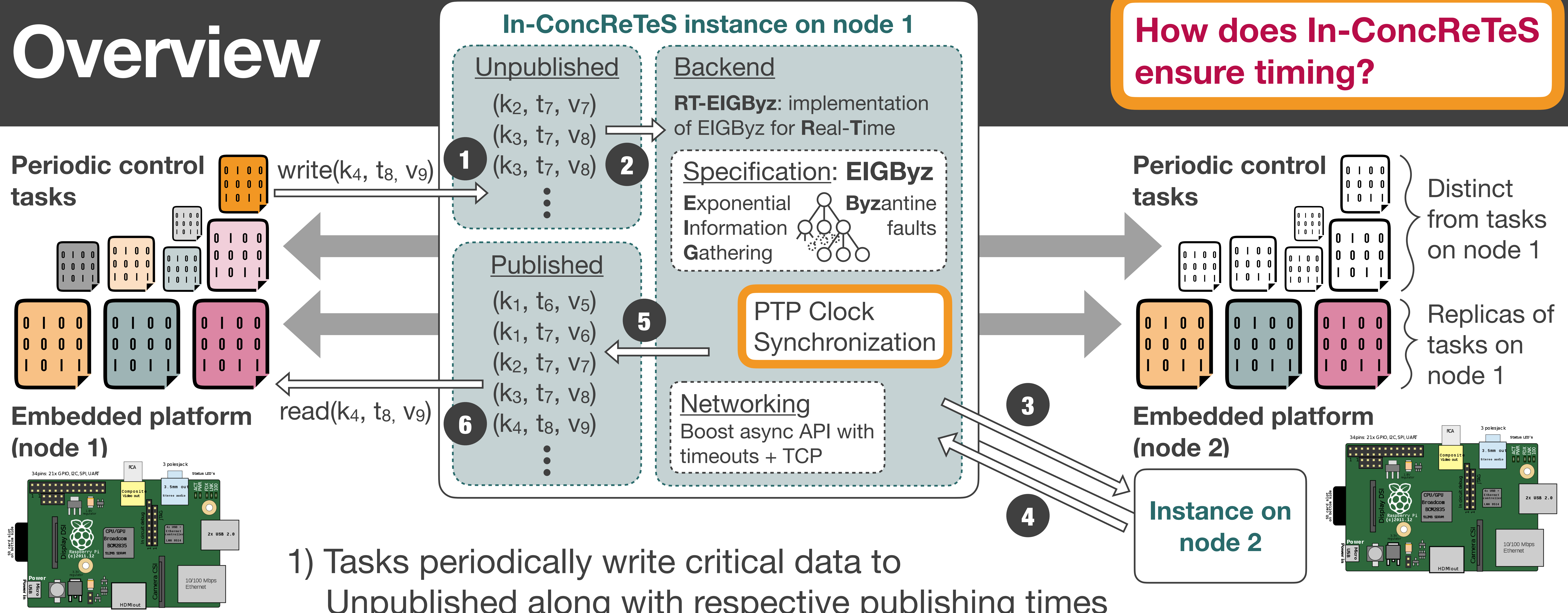
3, 4) Backend coordinates with remote nodes so that all replicas agree on the same set of values

5) Backend adds final “agreed-upon-by-all” values to Published

6) Tasks read values from Published in their next periodic iteration

Overview

How does In-ConcReTeS ensure timing?



1) Tasks periodically write critical data to <u>Unpublished</u> along with respective publishing times

2) <u>Backend</u> periodically empties <u>Unpublished</u>

3, 4) <u>Backend</u> coordinates with remote nodes so that all replicas agree on the same set of values

5) <u>Backend</u> adds final “agreed-upon-by-all” values to <u>Published</u>

6) Tasks read values from <u>Published</u> in their next periodic iteration

Key Idea 1: Time-aware Key-value API inspired by LET*

* Kirsch and Sokolova. "The Logical Execution Time Paradigm." Advances in Real-Time Systems (2012)

Key Idea 1: Time-aware Key-value API inspired by LET*

Algorithm 1 Controller interfaced with In-ConcReTeS

```
procedure KVSBackedInvertedPendulum
    time ← LastActivationAt()                              ▷ Compute freshness constraint
    globalTarget ← KVS.read("target", time)                ▷ Get globally consistent
    globalIntegral ← KVS.read("integral", time)            ▷ ... values of key
    globalError ← KVS.read("error", time)                  ▷ ... parameters
    current ← GetSensorData()
    error ← globalTarget − current
    integral ← globalIntegral + error
    derivative ← error − globalError
    force ← kp ∗ error + ki ∗ integral + kd ∗ derivative
    time ← timeOfNextActivation()                          ▷ Compute publishing time
    KVS.write("error", error, time)                        ▷ Globally synchronize state with
    KVS.write("integral", integral, time)                  ▷ ... other replicas
    actuate(force)
end procedure
```

* Kirsch and Sokolova. "The Logical Execution Time Paradigm." Advances in Real-Time Systems (2012)

Key Idea 1: Time-aware Key-value API inspired by LET*

Algorithm 1 Controller interfaced with In-ConcReTeS

```
procedure KVSBackedInvertedPendulum
    time ← LastActivationAt()                          ▷ Compute freshness constraint
    globalTarget ← KVS.read("target", time)            ▷ Get globally consistent
    globalIntegral ← KVS.read("integral", time)        ▷ ... values of key
    globalError ← KVS.read("error", time)              ▷ ... parameters
    current ← GetSensorData()
    error ← globalTarget − current
    integral ← globalIntegral + error
    derivative ← error − globalError
    force ← kp ∗ error + ki ∗ integral + kd ∗ derivative
    time ← timeOfNextActivation()                      ▷ Compute publishing time
    KVS.write("error", error, time)                    ▷ Globally synchronize state with
    KVS.write("integral", integral, time)              ▷ ... other replicas
    actuate(force)
end procedure
```

Reads impose **data freshness**

- "time" limits the age of the oldest value that can be accepted by a successful read

* Kirsch and Sokolova. "The Logical Execution Time Paradigm." Advances in Real-Time Systems (2012)

Key Idea 1: Time-aware Key-value API inspired by LET*

Algorithm 1 Controller interfaced with In-ConcReTeS

```
procedure KVSBACKEDINVERTEDPENDULUM
    time ← LastActivationAt()                          ▷ Compute freshness constraint
    globalTarget ← KVS.read("target", time)            ▷ Get globally consistent
    globalIntegral ← KVS.read("integral", time)        ▷ ... values of key
    globalError ← KVS.read("error", time)              ▷ ... parameters
    current ← GetSensorData()
    error ← globalTarget − current
    integral ← globalIntegral + error
    derivative ← error − globalError
    force ← kp ∗ error + ki ∗ integral + kd ∗ derivative
    time ← timeOfNextActivation()                      ▷ Compute publishing time
    KVS.write("error", error, time)                    ▷ Globally synchronize state with
    KVS.write("integral", integral, time)              ▷ ... other replicas
    actuate(force)
end procedure
```

Reads impose **data freshness**

- "time" limits the age of the oldest value that can be accepted by a successful read

Writes ensure **temporal determinism**

- "time" indicates the publishing time, when the value should become visible to all applications
- Decouples the time of data production from its availability in a predictable manner

* Kirsch and Sokolova. "The Logical Execution Time Paradigm." Advances in Real-Time Systems (2012)

Key Idea 1: Time-aware Key-value API inspired by LET*

Algorithm 1 Controller interfaced with In-ConcReTeS

```
procedure KVSBackedInvertedPendulum
    time ← LastActivationAt()                          ▷ Compute freshness constraint
    globalTarget ← KVS.read("target", time)            ▷ Get globally consistent
    globalIntegral ← KVS.read("integral", time)        ▷ ... values of key
    globalError ← KVS.read("error", time)              ▷ ... parameters
    current ← GetSensorData()
    error ← globalTarget − current
    integral ← globalIntegral + error
    derivative ← error − globalError
    force ← kp ∗ error + ki ∗ integral + kd ∗ derivative
    time ← timeOfNextActivation()                      ▷ Compute publishing time
    KVS.write("error", error, time)                    ▷ Globally synchronize state with
    KVS.write("integral", integral, time)              ▷ ... other replicas
    actuate(force)
end procedure
```

Reads impose **data freshness**

- "time" limits the age of the oldest value that can be accepted by a successful read

Writes ensure **temporal determinism**

- "time" indicates the publishing time, when the value should become visible to all applications
- Decouples the time of data production from its availability in a predictable manner

Clock synchronization ensures publishing times are meaningful across distributed nodes

* Kirsch and Sokolova. "The Logical Execution Time Paradigm." Advances in Real-Time Systems (2012)

Key Idea 1: Time-aware Key-value API inspired by LET*

Algorithm 1 Controller interfaced with In-ConcReTeS

```
procedure KVSBACKEDINVERTEDPENDULUM
    time ← LastActivationAt()                          ▷ Compute freshness constraint
    globalTarget ← KVS.read("target", time)            ▷ Get globally consistent
    globalIntegral ← KVS.read("integral", time)        ▷ ... values of key
    globalError ← KVS.read("error", time)              ▷ ... parameters
    current ← GetSensorData()
    error ← globalTarget − current
    integral ← globalIntegral + error
    derivative ← error − globalError
    force ← kp ∗ error + ki ∗ integral + kd ∗ derivative
    time ← timeOfNextActivation()                      ▷ Compute publishing time
    KVS.write("error", error, time)                    ▷ Globally synchronize state with
    KVS.write("integral", integral, time)              ▷ ... other replicas
    actuate(force)
end procedure
```

Reads impose **data freshness**

- "time" limits the age of the oldest value that can be accepted by a successful read

Writes ensure **temporal determinism**

- "time" indicates the publishing time, when the value should become visible to all applications
- Decouples the time of data production from its availability in a predictable manner

Clock synchronization ensures publishing times are meaningful across distributed nodes

API enables **static analysis** as it informs about the time budget available for replica coordination

* Kirsch and Sokolova. "The Logical Execution Time Paradigm." Advances in Real-Time Systems (2012)

Key Idea 2: Interactive Consistency[1] → EIGByz[2] → RT-EIGByz

[1] Kieckhafer *et al.* “The MAFT Architecture for Distributed Fault Tolerance.” IEEE Transactions on Computers (1988)

[2] Borran and Schiper. “A Leader-Free Byzantine Consensus Algorithm.” ICDCN (2010)

Key Idea 2: Interactive Consistency[1] → EIGByz[2] → RT-EIGByz

Byzantine agreement problem

- **Consensus over a vector of data**
- Enables **application-specific voting**, such as using median or weighted mean

Node 1 Node 2 Node 3 All nodes agree on this

$$\begin{bmatrix} v_{1,1} \\ v_{1,2} \\ v_{1,3} \end{bmatrix}, \begin{bmatrix} v_{2,1} \\ v_{2,2} \\ v_{2,3} \end{bmatrix}, \begin{bmatrix} v_{3,1} \\ v_{3,2} \\ v_{3,3} \end{bmatrix} \xrightarrow{\text{Interactive Consistency}} \begin{bmatrix} \mathbf{v_1} \\ \mathbf{v_2} \\ \mathbf{v_3} \end{bmatrix} \xrightarrow{\text{Application-specific voting function}} \mathbf{v_{final}}$$

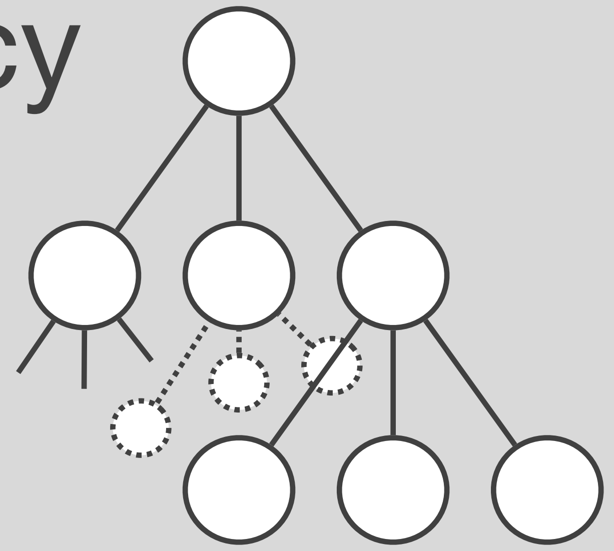
[1] Kieckhafer *et al.* “The MAFT Architecture for Distributed Fault Tolerance.” IEEE Transactions on Computers (1988)

[2] Borran and Schiper. “A Leader-Free Byzantine Consensus Algorithm.” ICDCN (2010)

Key Idea 2: Interactive Consistency[1] → EIGByz[2] → RT-EIGByz

Synchronous leader-free protocol for interactive consistency

- EIG trees
- Clock synchronization
- Deterministic rounds



Byzantine agreement problem

- **Consensus over a vector of data**
- Enables **application-specific voting**, such as using median or weighted mean

Node 1 Node 2 Node 3 All nodes agree on this

$$\begin{bmatrix} v_{1,1} \\ v_{1,2} \\ v_{1,3} \end{bmatrix}, \begin{bmatrix} v_{2,1} \\ v_{2,2} \\ v_{2,3} \end{bmatrix}, \begin{bmatrix} v_{3,1} \\ v_{3,2} \\ v_{3,3} \end{bmatrix} \xrightarrow{\text{Interactive Consistency}} \begin{bmatrix} \mathbf{v_1} \\ \mathbf{v_2} \\ \mathbf{v_3} \end{bmatrix} \xrightarrow{\text{Application-specific voting function}} \mathbf{v_{final}}$$

[1] Kieckhafer *et al.* “The MAFT Architecture for Distributed Fault Tolerance.” IEEE Transactions on Computers (1988)

[2] Borran and Schiper. “A Leader-Free Byzantine Consensus Algorithm.” ICDCN (2010)

Key Idea 2: Interactive Consistency[1] → EIGByz[2] → RT-EIGByz

Byzantine agreement problem

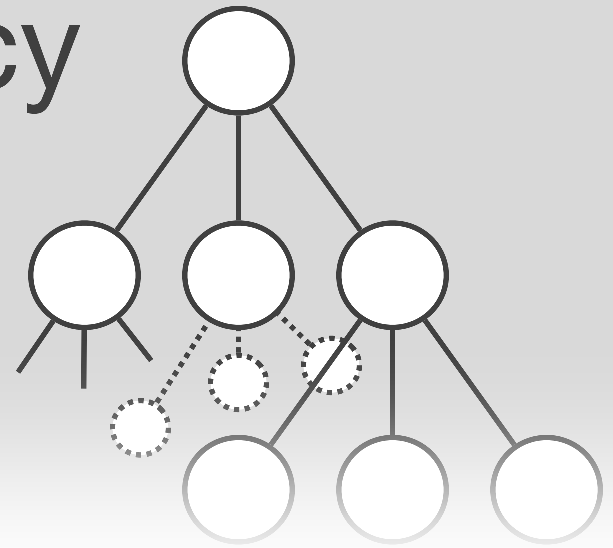
- **Consensus over a vector of data**
- Enables **application-specific voting**, such as using median or weighted mean

Node 1 Node 2

$$\begin{bmatrix} v_{1,1} \\ v_{1,2} \\ v_{1,3} \end{bmatrix}, \begin{bmatrix} v_{2,1} \\ v_{2,2} \\ v_{2,3} \end{bmatrix},$$

Synchronous leader-free protocol for interactive consistency

- EIG trees
- Clock synchronization
- Deterministic rounds



Predictable real-time friendly implementation of EIGByz

[1] Kieckhafer *et al.* “The MAFT Architec

[2] Borran and Schiper. “A Leader-Free Byzantine Consensus Algorithm.” ICDCN (2010)

Key Idea 2: Interactive Consistency[1] → EIGByz[2] → RT-EIGByz

Byzantine agreement problem

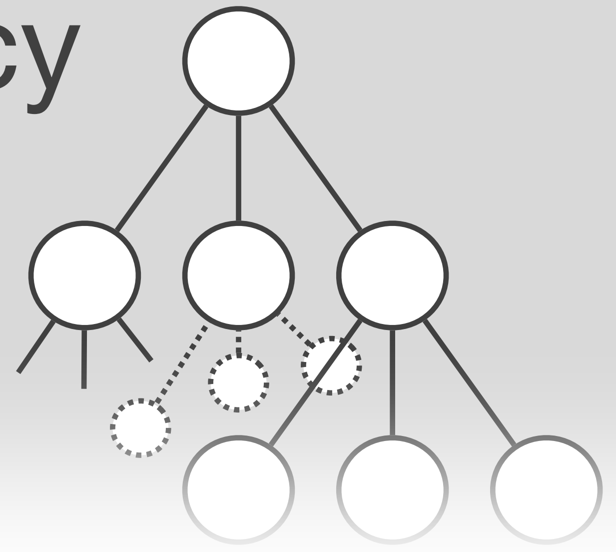
- **Consensus over a vector of data**
- Enables **application-specific voting**, such as using median or weighted mean

Node 1 Node 2

$$\begin{bmatrix} v_{1,1} \\ v_{1,2} \\ v_{1,3} \end{bmatrix}, \begin{bmatrix} v_{2,1} \\ v_{2,2} \\ v_{2,3} \end{bmatrix},$$

Synchronous leader-free protocol for interactive consistency

- EIG trees
- Clock synchronization
- Deterministic rounds



Predictable real-time friendly implementation of EIGByz

- Real-time periodic tasks → deterministic scheduling

[1] Kieckhafer *et al.* "The MAFT Architec

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

Key Idea 2: Interactive Consistency[1] → EIGByz[2] → RT-EIGByz

Byzantine agreement problem

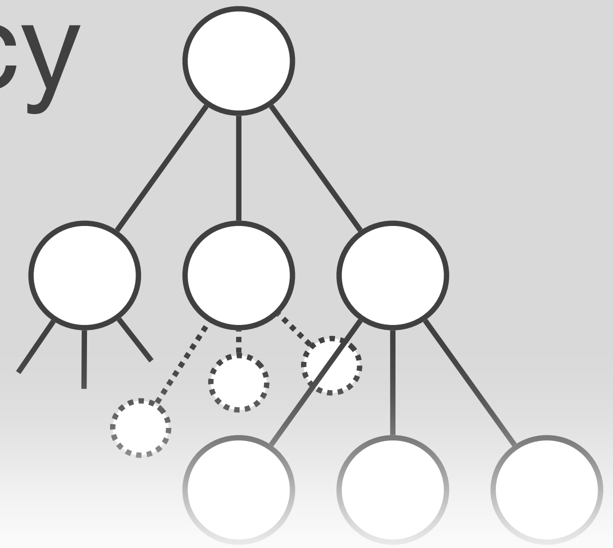
- **Consensus over a vector of data**
- Enables **application-specific voting**, such as using median or weighted mean

Node 1 Node 2

$$\begin{bmatrix} v_{1,1} \\ v_{1,2} \\ v_{1,3} \end{bmatrix}, \begin{bmatrix} v_{2,1} \\ v_{2,2} \\ v_{2,3} \end{bmatrix},$$

Synchronous leader-free protocol for interactive consistency

- EIG trees
- Clock synchronization
- Deterministic rounds



Predictable real-time friendly implementation of EIGByz

- Real-time periodic tasks → deterministic scheduling
- 1D, contiguous memory layout of EIG trees → fast reads and writes

[1] Kieckhafer *et al.* “The MAFT Architec

[2] Borran and Schiper. “A Leader-Free Byzantine Consensus Algorithm.” ICDCN (2010)

Key Idea 2: Interactive Consistency[1] → EIGByz[2] → RT-EIGByz

Byzantine agreement problem

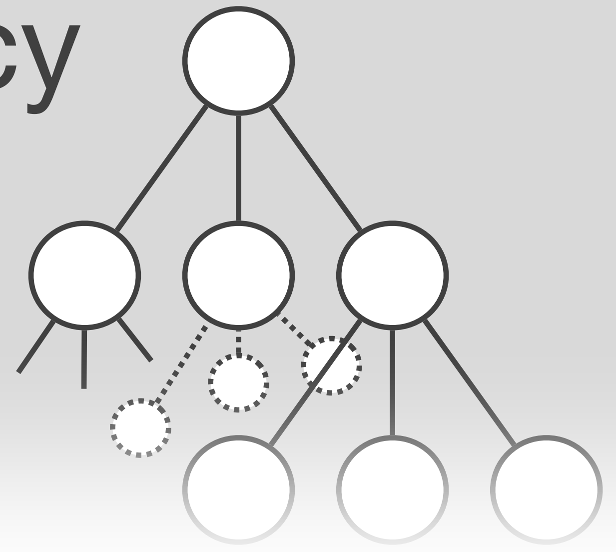
- **Consensus over a vector of data**
- Enables **application-specific voting**, such as using median or weighted mean

Node 1 Node 2

$$\begin{bmatrix} v_{1,1} \\ v_{1,2} \\ v_{1,3} \end{bmatrix}, \begin{bmatrix} v_{2,1} \\ v_{2,2} \\ v_{2,3} \end{bmatrix},$$

Synchronous leader-free protocol for interactive consistency

- EIG trees
- Clock synchronization
- Deterministic rounds



Predictable real-time friendly implementation of EIGByz

- Real-time periodic tasks → deterministic scheduling
- 1D, contiguous memory layout of EIG trees → fast reads and writes
- Static allocation parameterized in #nodes, #rounds → predictability

[1] Kieckhafer *et al.* "The MAFT Architec

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

Key Idea 2: Interactive Consistency[1] → EIGByz[2] → RT-EIGByz

Byzantine agreement problem

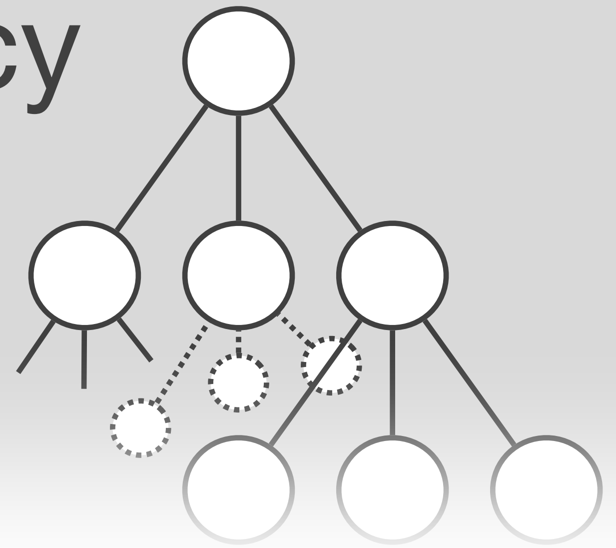
- **Consensus over a vector of data**
- Enables **application-specific voting**, such as using median or weighted mean

Node 1 Node 2

$$\begin{bmatrix} v_{1,1} \\ v_{1,2} \\ v_{1,3} \end{bmatrix}, \begin{bmatrix} v_{2,1} \\ v_{2,2} \\ v_{2,3} \end{bmatrix},$$

Synchronous leader-free protocol for interactive consistency

- EIG trees
- Clock synchronization
- Deterministic rounds



Predictable real-time friendly implementation of EIGByz

- Real-time periodic tasks → deterministic scheduling
- 1D, contiguous memory layout of EIG trees → fast reads and writes
- Static allocation parameterized in #nodes, #rounds → predictability
- TCP with timeouts → timeliness, prevents cascading failures

[1] Kieckhafer *et al.* “The MAFT Architec

[2] Borran and Schiper. “A Leader-Free Byzantine Consensus Algorithm.” ICDCN (2010)

Key Idea 2: Interactive Consistency[1] → EIGByz[2] → RT-EIGByz

Byzantine agreement problem

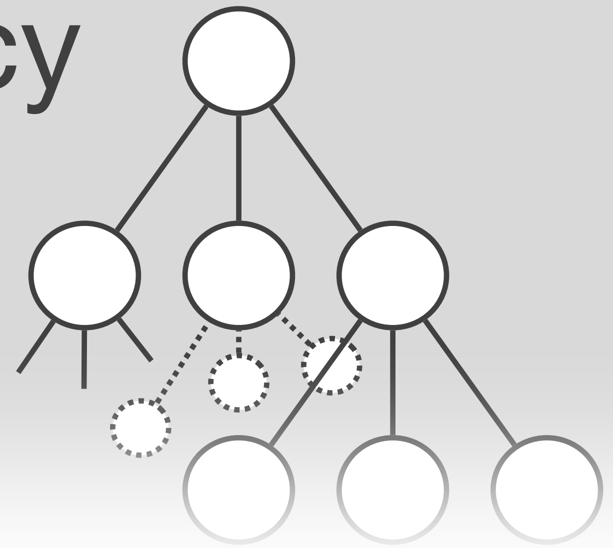
- **Consensus over a vector of data**
- Enables **application-specific voting**, such as using median or weighted mean

Node 1 Node 2

$$\begin{bmatrix} v_{1,1} \\ v_{1,2} \\ v_{1,3} \end{bmatrix}, \begin{bmatrix} v_{2,1} \\ v_{2,2} \\ v_{2,3} \end{bmatrix},$$

Synchronous leader-free protocol for interactive consistency

- EIG trees
- Clock synchronization
- Deterministic rounds



Predictable real-time friendly implementation of EIGByz

- Real-time periodic tasks → deterministic scheduling
- 1D, contiguous memory layout of EIG trees → fast reads and writes
- Static allocation parameterized in #nodes, #rounds → predictability
- TCP with timeouts → timeliness, prevents cascading failures
- Batching → multiples keys

[1] Kieckhafer *et al.* “The MAFT Architec

[2] Borran and Schiper. “A Leader-Free Byzantine Consensus Algorithm.” ICDCN (2010)

Implementation

Implementation

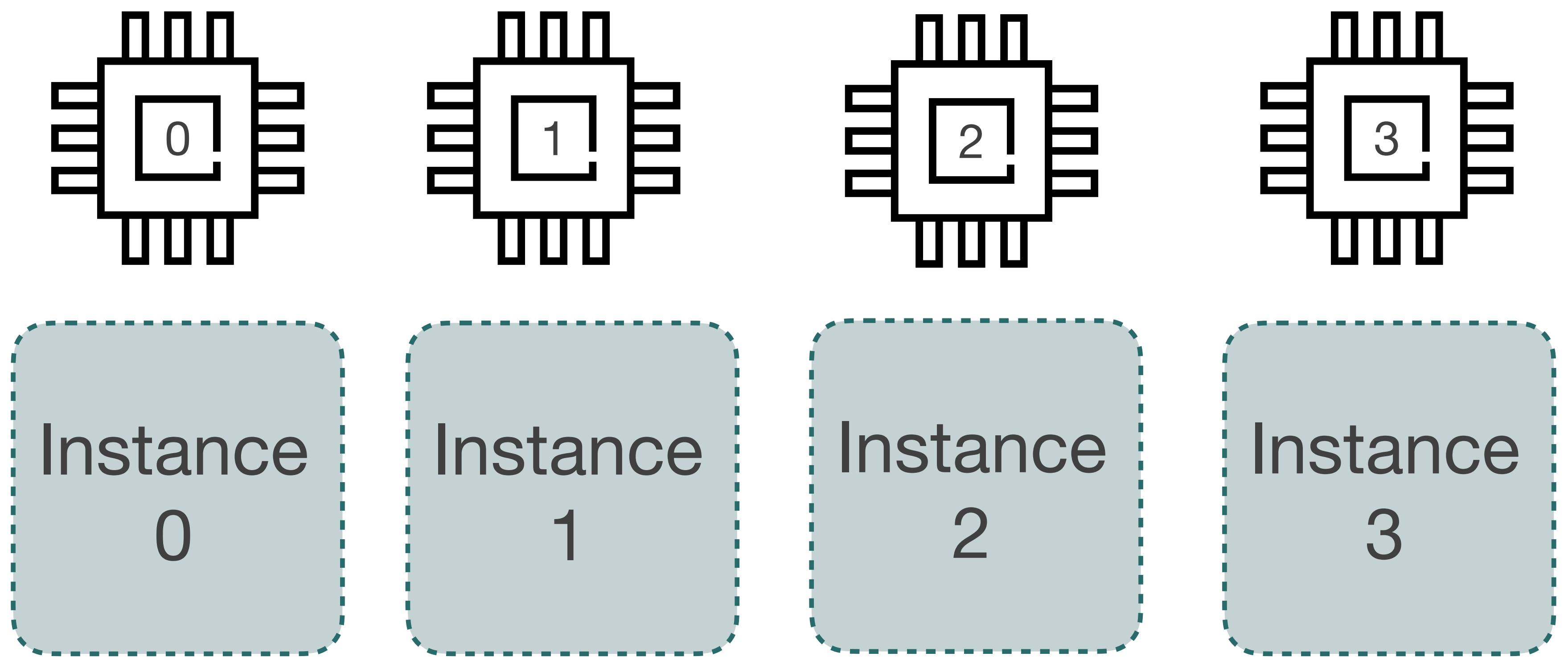
RT-EIGByz realized as a **single-threaded real-time periodic task**

- Implemented in C++ using Linux's `clock_gettime` and `clock_nanosleep` APIs

Implementation

RT-EIGByz realized as a **single-threaded real-time periodic task**

- Implemented in C++ using Linux's `clock_gettime` and `clock_nanosleep` APIs
- Unique In-ConcReTeS instance per core



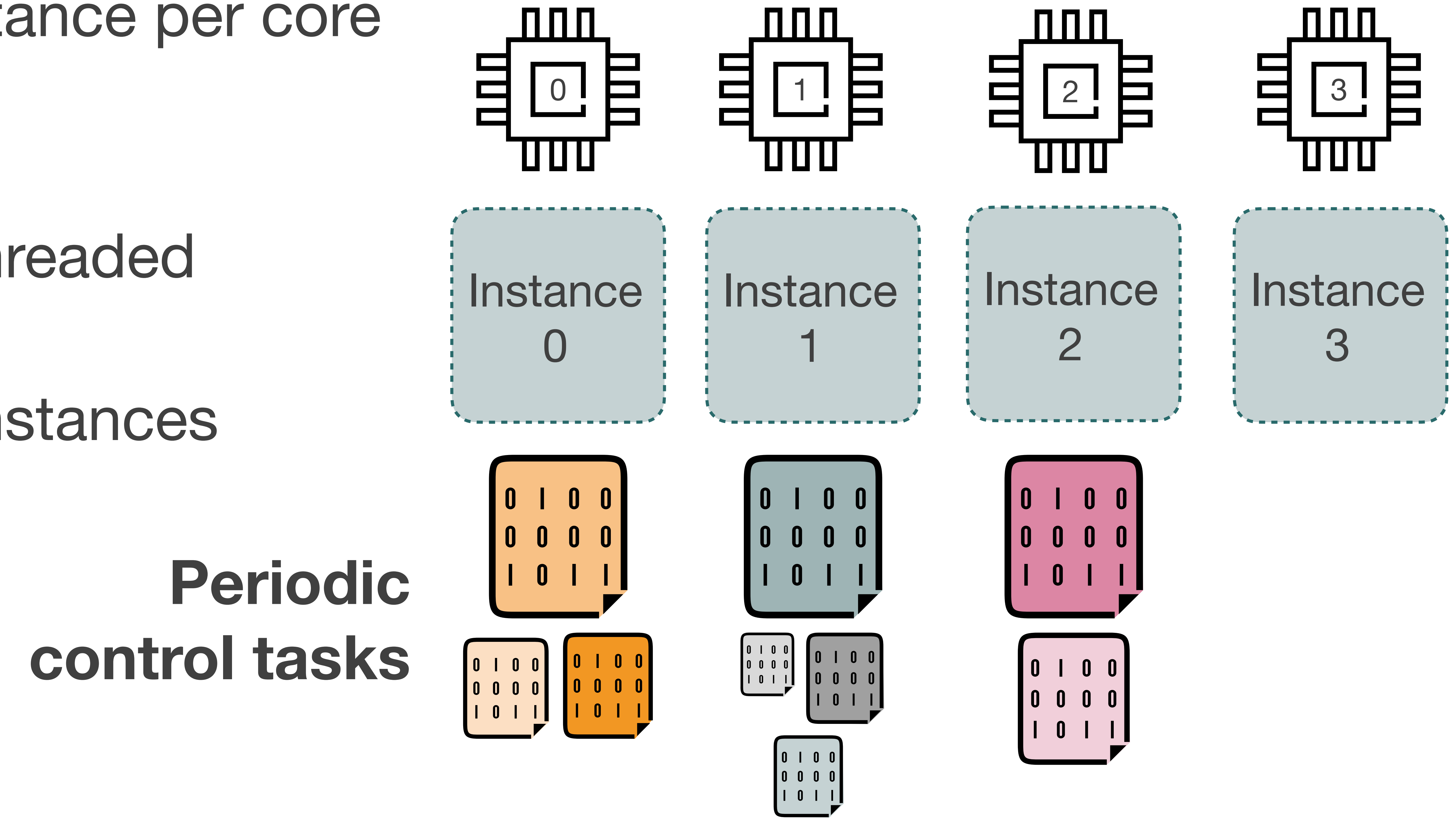
Implementation

RT-EIGByz realized as a **single-threaded real-time periodic task**

- Implemented in C++ using Linux's `clock_gettime` and `clock_nanosleep` APIs
- Unique In-ConcReTeS instance per core

Control applications

- Also modelled as single-threaded real-time periodic tasks
- Interface with core-local instances



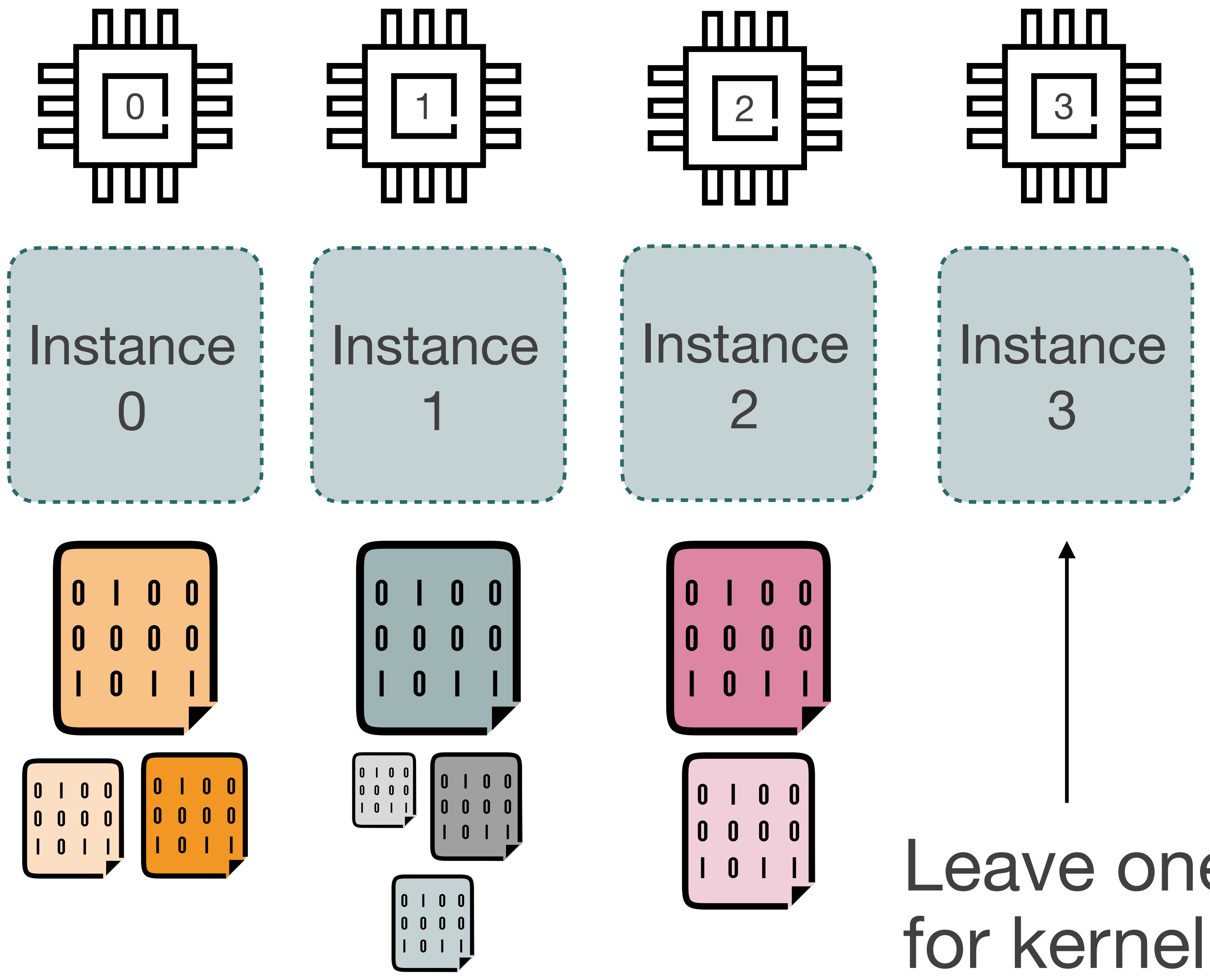
Implementation

RT-EIGByz realized as a **single-threaded real-time periodic task**

- Implemented in C++ using Linux's `clock_gettime` and `clock_nanosleep` APIs
- Unique In-ConcReTeS instance per core

Control applications

- Also modelled as single-threaded real-time periodic tasks
- Interface with core-local instances



Periodic control tasks

Leave one core **idle** for kernel and networking utilities

Evaluation

Questions

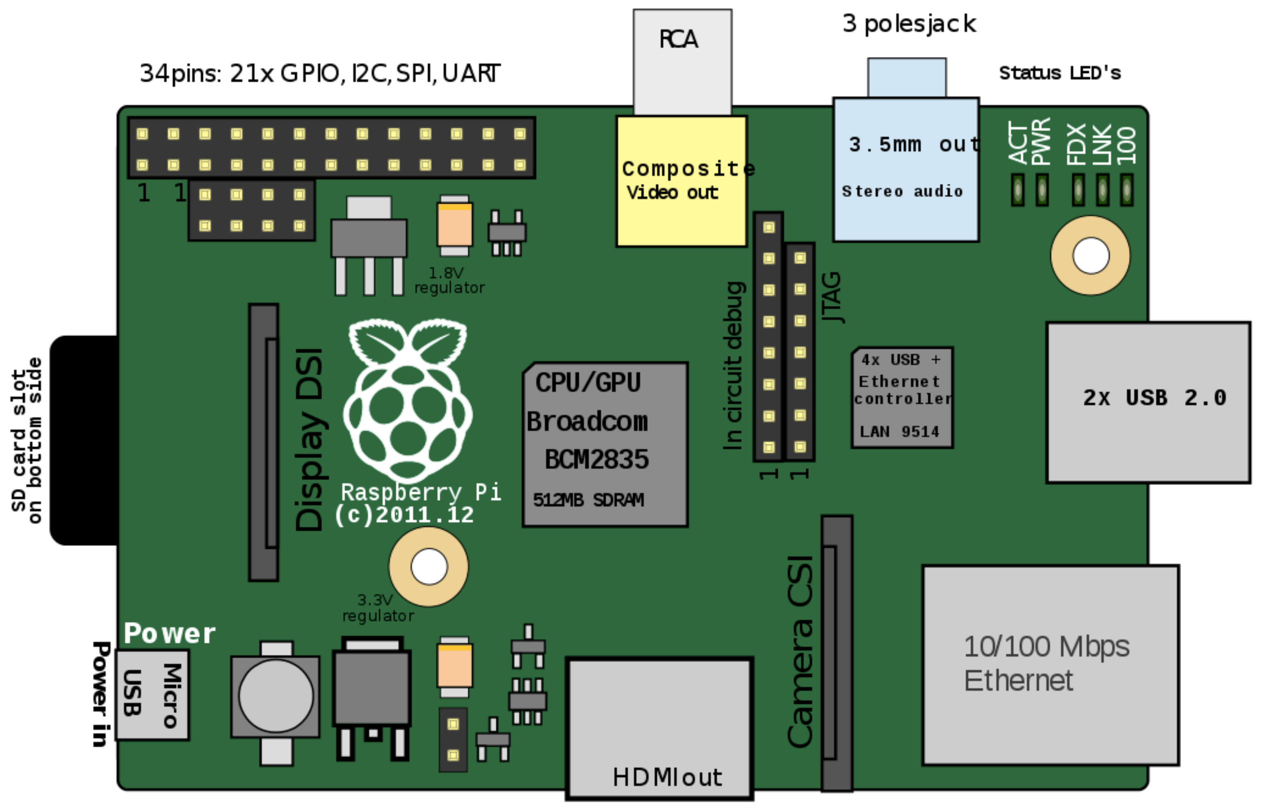
How does In-ConcReTeS compare against well-known KVS?

Can In-ConcReTeS deal with complex distributed real-time workloads?

Setup

Four Raspberry Pi 4 Model B units

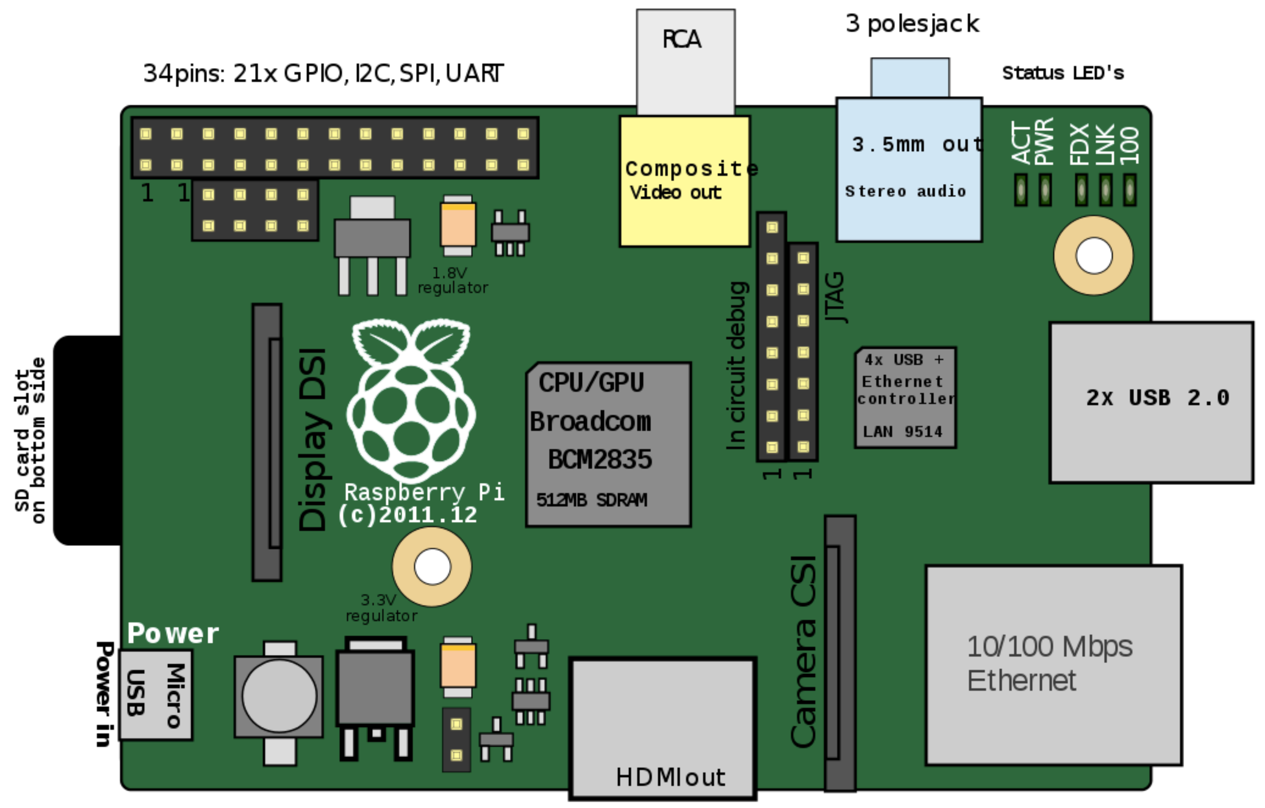
- Cortex A72 **quad-core** processor
- 4GB memory
- Raspbian GNU/Linux 10
- **Ethernet**



Setup

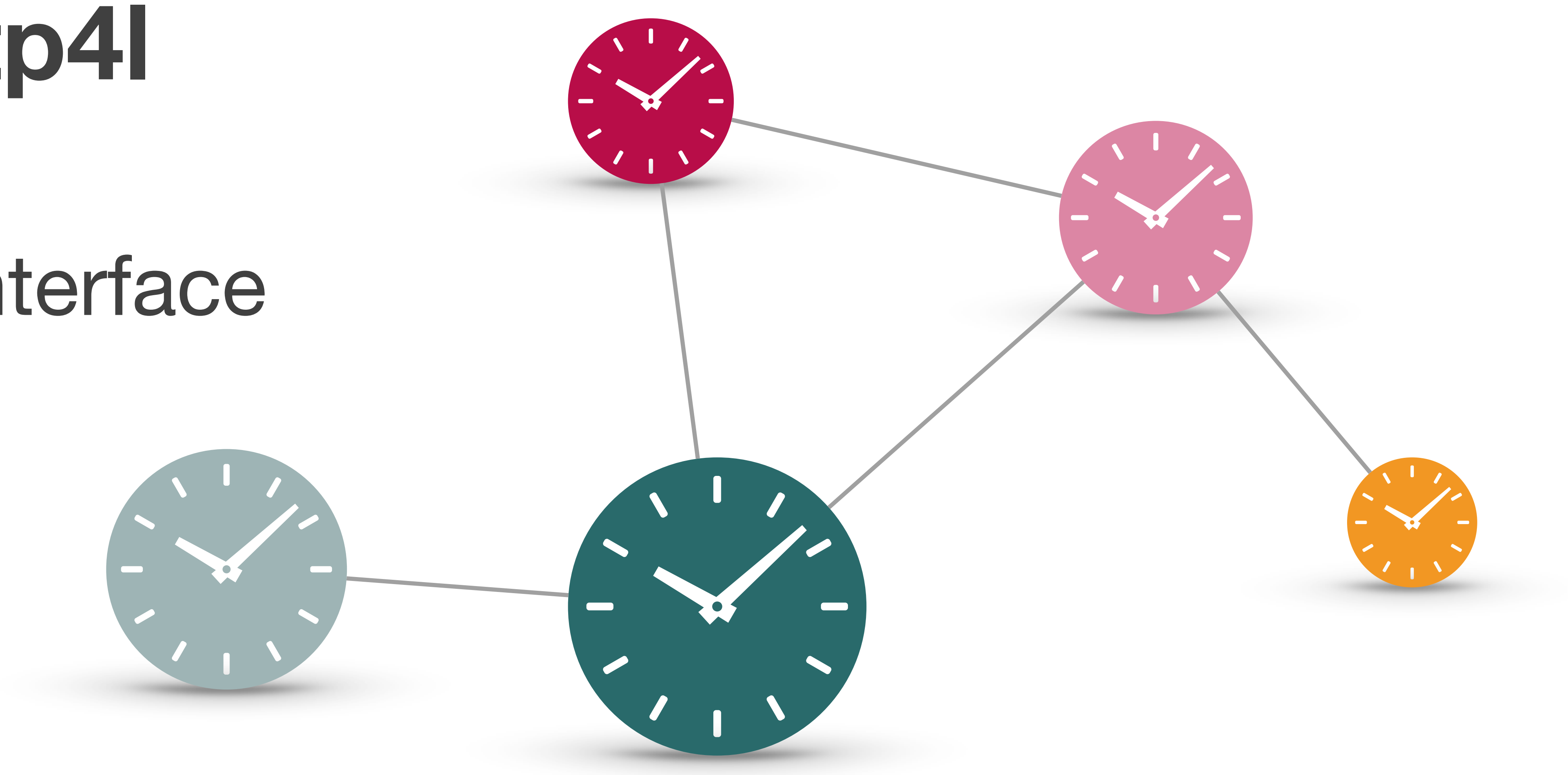
Four Raspberry Pi 4 Model B units

- Cortex A72 **quad-core** processor
- 4GB memory
- Raspbian GNU/Linux 10
- **Ethernet**



Clock synchronization using **ptp4l**

- Highest real-time priority
- Uses a separate virtual network interface
- **Software time-stamping**



Questions

How does In-ConcReTeS compare against well-known KVS?

Can In-ConcReTeS deal with complex distributed real-time workloads?

Baselines

Baselines

redis

In-memory data store

Baselines

redis

In-memory data store

Single-threaded C server

- Not designed to benefit from multiple cores
- Like Achal, **one instance per each core, on each Pi**

Baselines

redis

Query all
nodes
at read time

In-memory data store

Single-threaded C server

- Not designed to benefit from multiple cores
- Like Achal, **one instance per each core, on each Pi**

No fault tolerance by default

- "Replicated" config has lazy semantics, not useful for CPS
- For fault tolerance, we **query Redis instances on all nodes at read time**

Baselines

redis

Query all nodes at read time

In-memory data store

Single-threaded C server

- Not designed to benefit from multiple cores
- Like Achal, **one instance per each core, on each Pi**

No fault tolerance by default

- "Replicated" config has lazy semantics, not useful for CPS
- For fault tolerance, we **query Redis instances on all nodes at read time**

etcd

Strongly consistent, distributed

- Written in Go
- Raft Consensus for fault tolerance
- **Single instance on each Pi**

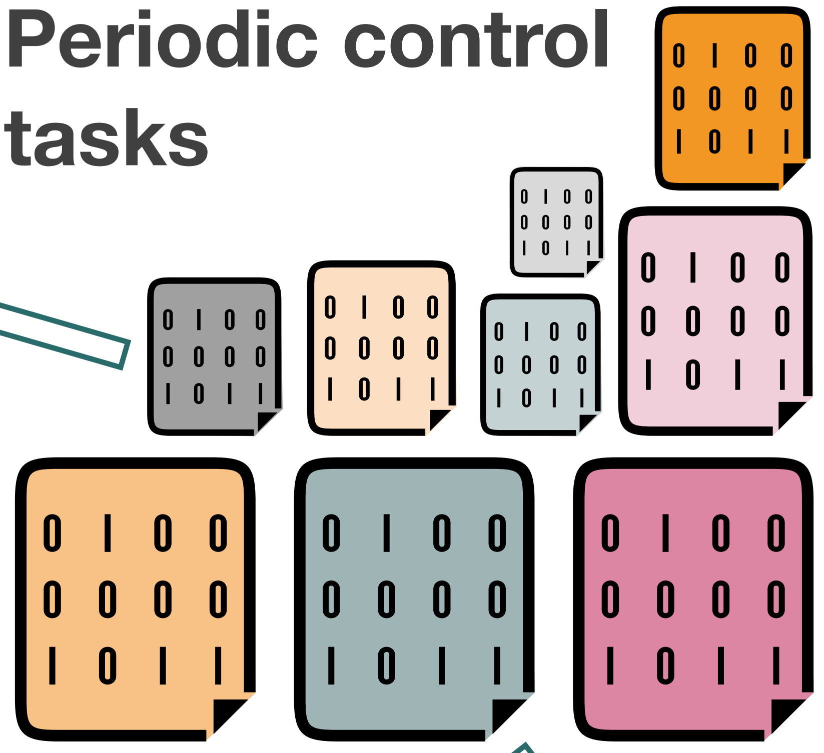
Baselines

redis

Query all nodes at read time

Core-local proxies

Periodic control tasks



Mimic In-ConcReTeS's real-time periodic task model, time-aware API

In-memory data store

Single-threaded C server

- Not designed to benefit from multiple cores
- Like Achal, **one instance per each core, on each Pi**

No fault tolerance by default

- "Replicated" config has lazy semantics, not useful for CPS
- For fault tolerance, we **query Redis instances on all nodes at read time**

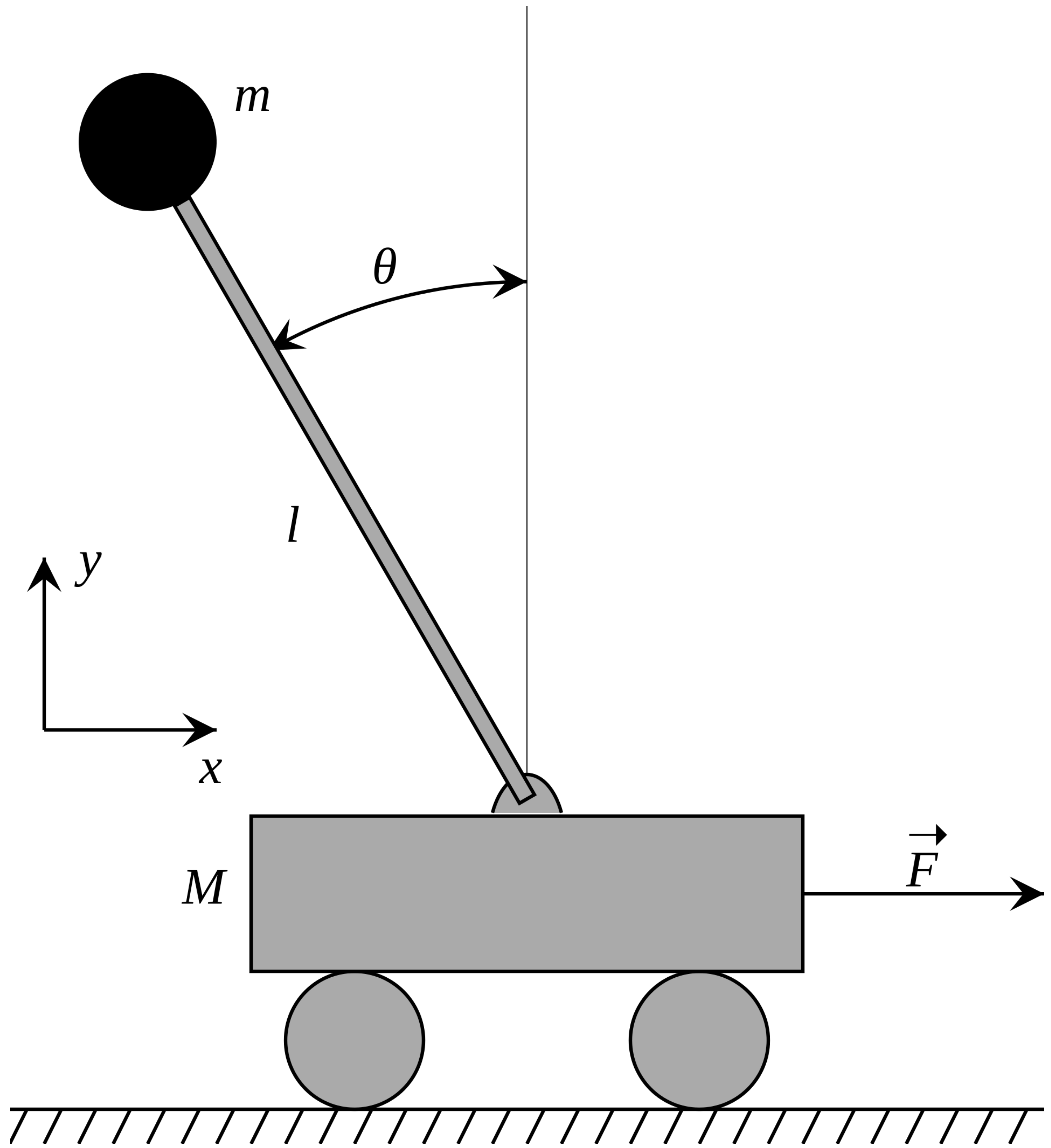
Core-local proxies

etcd

Strongly consistent, distributed

- Written in Go
- Raft Consensus for fault tolerance
- **Single instance on each Pi**

Workload: Inverted Pendulum Simulation*



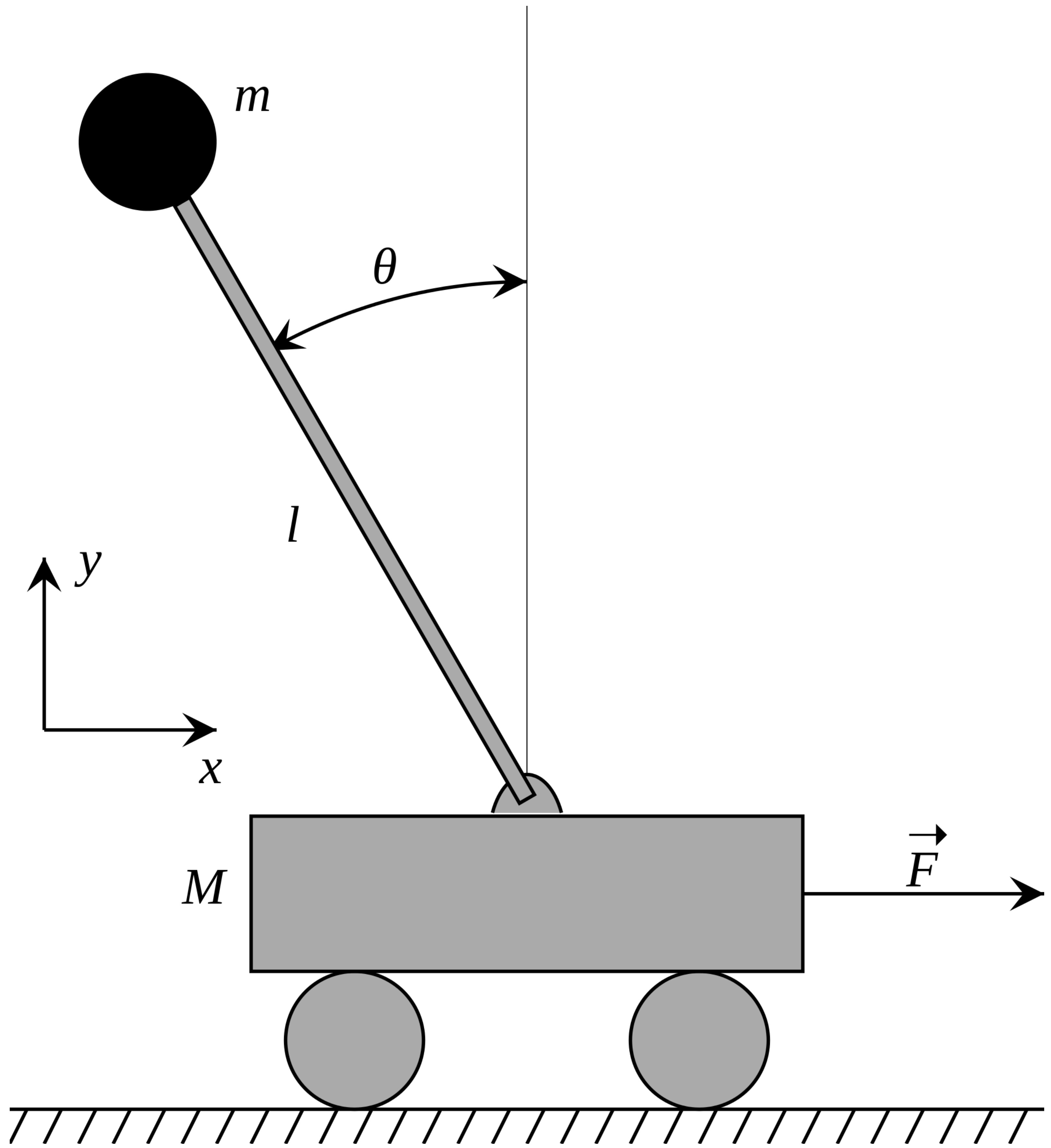
* Morgado. https://gmagno.users.sourceforge.net/InvertedPendulum.htm (2011)

Workload: Inverted Pendulum Simulation*

Prototypical control application

IvPSim

- Periodic real-time task
- Reads and writes 19 floats to the datastore
- Time period can be adjusted



* Morgado. https://gmagno.users.sourceforge.net/InvertedPendulum.htm (2011)

Workload: Inverted Pendulum Simulation*

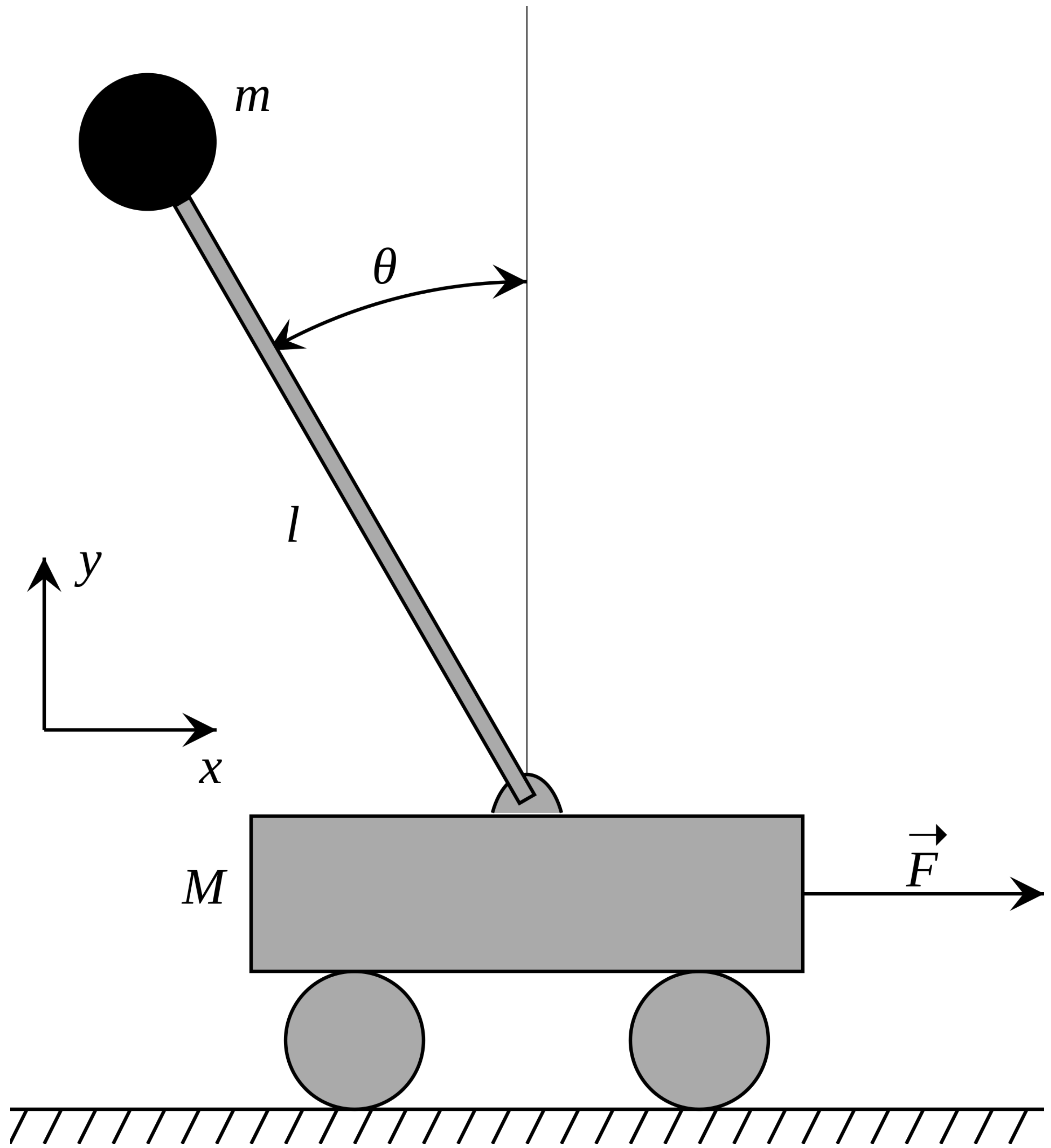
Prototypical control application

IvPSim

- Periodic real-time task
- Reads and writes 19 floats to the datastore
- Time period can be adjusted

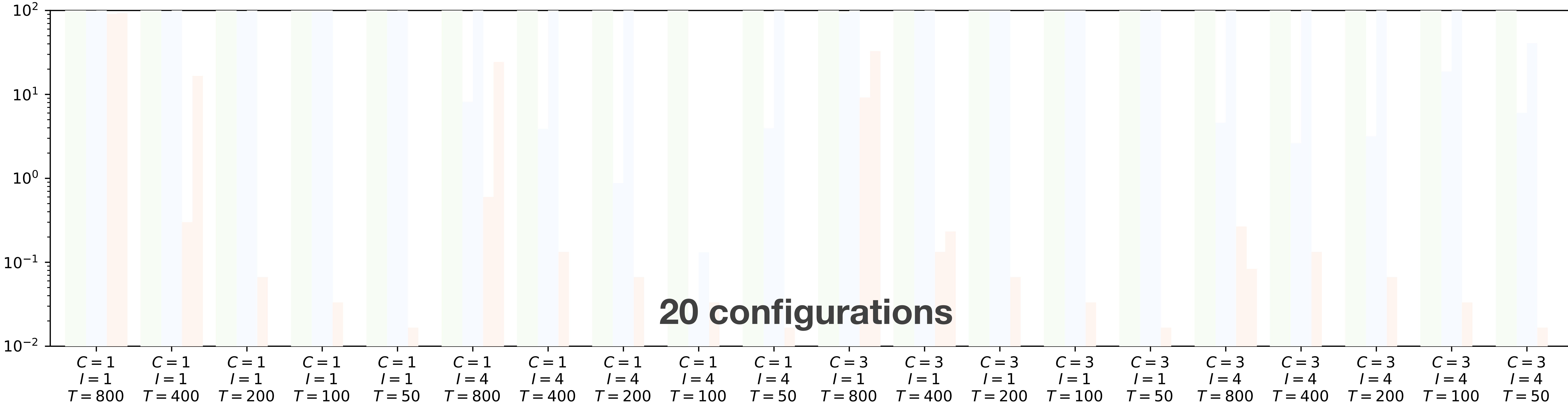
Objective

- Run replicas of IvPSim on separate nodes synchronously



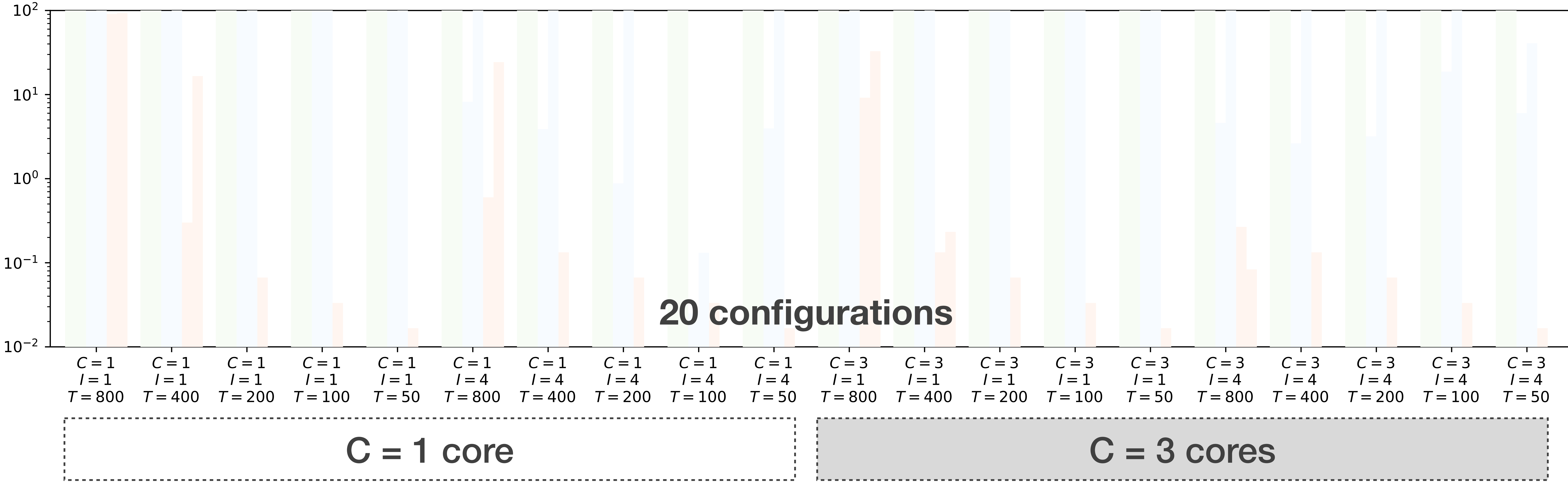
* Morgado. https://gmagno.users.sourceforge.net/InvertedPendulum.htm (2011)

Results



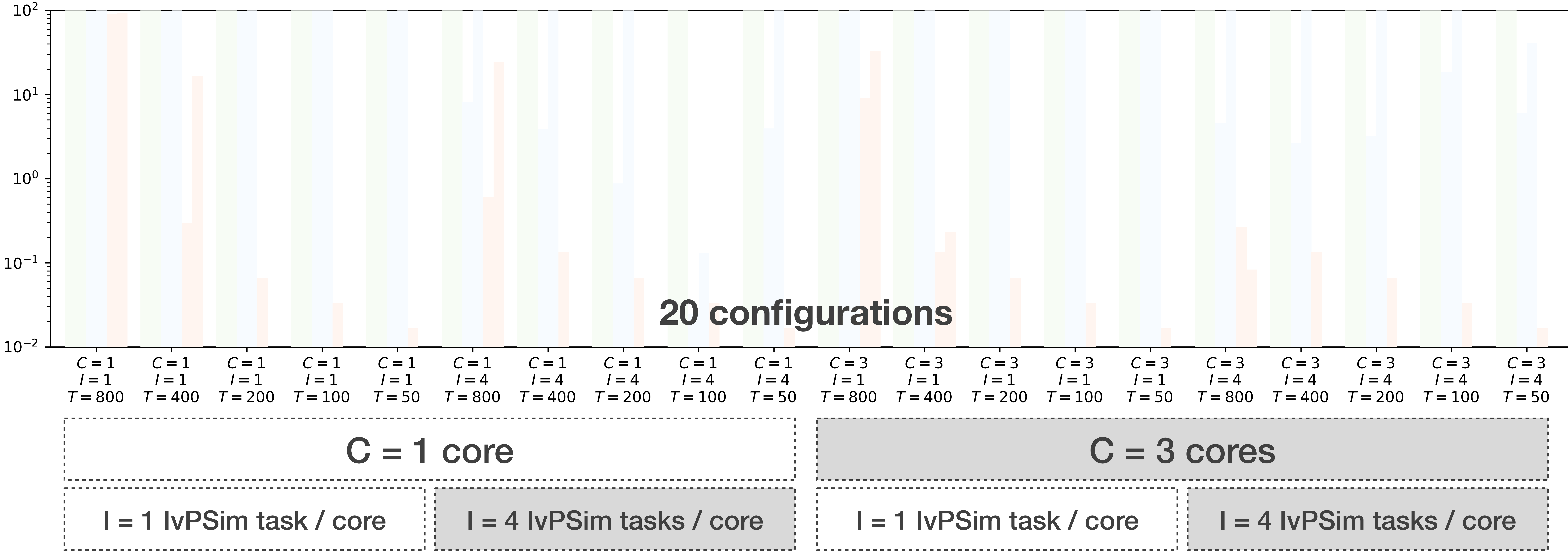
10^2
10^1
10^0
10^{-1}
10^{-2}
20 configurations
$C=1$, $I=1$, $T=800$
$C=1$, $I=1$, $T=400$
$C=1$, $I=1$, $T=200$
$C=1$, $I=1$, $T=100$
$C=1$, $I=1$, $T=50$
$C=1$, $I=4$, $T=800$
$C=1$, $I=4$, $T=400$
$C=1$, $I=4$, $T=200$
$C=1$, $I=4$, $T=100$
$C=1$, $I=4$, $T=50$
$C=3$, $I=1$, $T=800$
$C=3$, $I=1$, $T=400$
$C=3$, $I=1$, $T=200$
$C=3$, $I=1$, $T=100$
$C=3$, $I=1$, $T=50$
$C=3$, $I=4$, $T=800$
$C=3$, $I=4$, $T=400$
$C=3$, $I=4$, $T=200$
$C=3$, $I=4$, $T=100$
$C=3$, $I=4$, $T=50$

Results

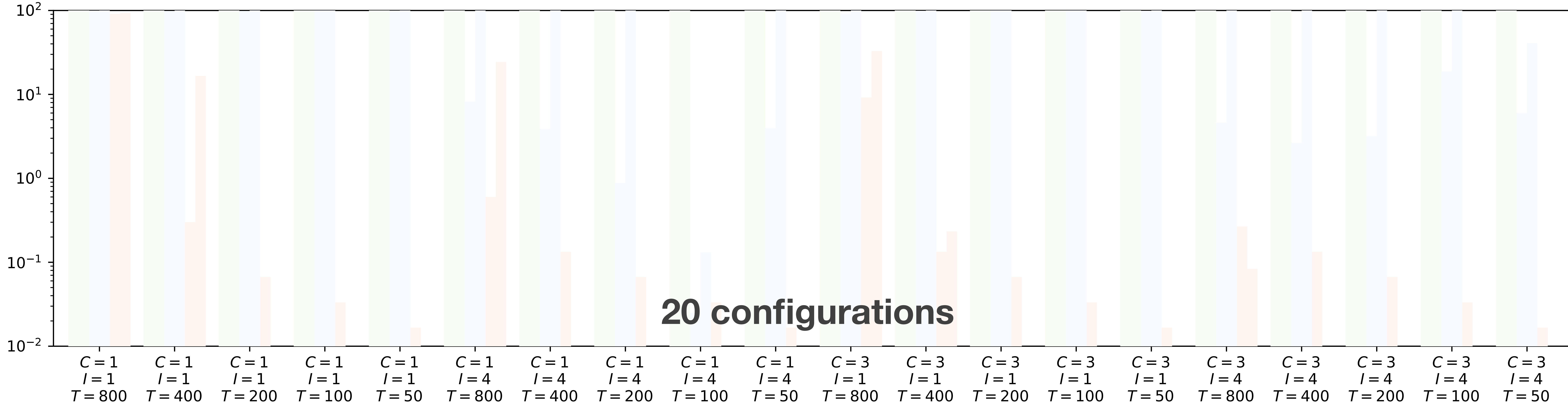


10^2
10^1
10^0
10^{-1}
10^{-2}
20 configurations
$C=1$ $I=1$ $T=800$
$C=1$ $I=1$ $T=400$
$C=1$ $I=1$ $T=200$
$C=1$ $I=1$ $T=100$
$C=1$ $I=1$ $T=50$
$C=1$ $I=4$ $T=800$
$C=1$ $I=4$ $T=400$
$C=1$ $I=4$ $T=200$
$C=1$ $I=4$ $T=100$
$C=1$ $I=4$ $T=50$
$C=3$ $I=1$ $T=800$
$C=3$ $I=1$ $T=400$
$C=3$ $I=1$ $T=200$
$C=3$ $I=1$ $T=100$
$C=3$ $I=1$ $T=50$
$C=3$ $I=4$ $T=800$
$C=3$ $I=4$ $T=400$
$C=3$ $I=4$ $T=200$
$C=3$ $I=4$ $T=100$
$C=3$ $I=4$ $T=50$
C = 1 core
C = 3 cores

Results


10^2
10^1
10^0
10^{-1}
10^{-2}
20 configurations
$C=1$ $I=1$ $T=800$
$C=1$ $I=1$ $T=400$
$C=1$ $I=1$ $T=200$
$C=1$ $I=1$ $T=100$
$C=1$ $I=1$ $T=50$
$C=1$ $I=4$ $T=800$
$C=1$ $I=4$ $T=400$
$C=1$ $I=4$ $T=200$
$C=1$ $I=4$ $T=100$
$C=1$ $I=4$ $T=50$
$C=3$ $I=1$ $T=800$
$C=3$ $I=1$ $T=400$
$C=3$ $I=1$ $T=200$
$C=3$ $I=1$ $T=100$
$C=3$ $I=1$ $T=50$
$C=3$ $I=4$ $T=800$
$C=3$ $I=4$ $T=400$
$C=3$ $I=4$ $T=200$
$C=3$ $I=4$ $T=100$
$C=3$ $I=4$ $T=50$
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

Results



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 →

Results

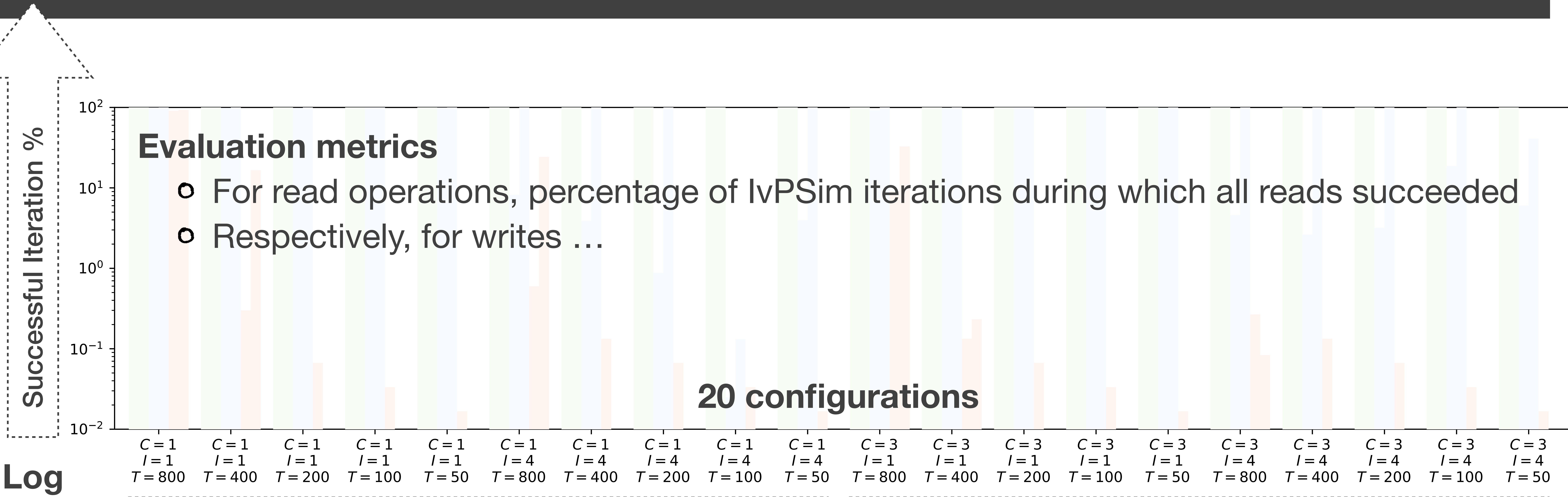
Evaluation metrics

- For read operations, percentage of IvPSim iterations during which all reads succeeded
- Respectively, for writes …

Results

Evaluation metrics

- For read operations, percentage of IvPSim iterations during which all reads succeeded
- Respectively, for writes …



Log scale

Higher is better!

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 →

Results

- **In-ConcReTeS success rate is 100%**

Results

- In-ConcReTeS success rate is 100%
- Redis underperforms when I = 4

Results

- In-ConcReTeS success rate is 100%
- Redis underperforms when I = 4
- etcd almost always underperforms

Results

- In-ConcReTeS success rate is 100%
- Redis underperforms when I = 4
- etcd almost always underperforms

Redis and etcd are not build for timeliness on the timescales encountered in CPS

Questions

How does In-ConcReTeS compare against well-known KVS?

Can In-ConcReTeS deal with complex distributed real-time workloads?

Workload: Real-Automotive Benchmark from Bosch*

* Kramer *et al.* “Real World Automotive Benchmarks for Free.” WATERS (2015)

Workload: Real-Automotive Benchmark from Bosch*

TABLE I. DISTRIBUTION OF LABEL SIZES

Size (byte)	Share
1	35 %
2	49 %
4	
5 - 8	
9 - 16	
17 - 32	
33 - 64	
> 64	

TABLE II. INTER-TASK COMM

Period	1 ms	2 ms	5 ms	10 ms	20 ms	50 ms				
1 ms				I	I					
2 ms				I	I					
5 ms		I	IV	IV	II	II				
10 ms	II	II	II	VI	IV	II	IV	II	III	IV
20 ms	I	I	I	IV	VI	II	IV	I	II	IV
50 ms			II	II	II	III	I			
100 ms		I	I	V	IV	II	VI	II	III	IV
200 ms				I	I		I	I	I	
1000 ms				III	II		III	I	IV	I
Angle-sync	I	I	I	IV	IV	I	III	I	I	V

TABLE III. RUNNABLE DISTRIBUTION AMONG PERIODS

Period	Share
1 ms	3 %
2 ms	2 %
5 ms	
10 ms	
20 ms	
50 ms	
100 ms	

TABLE IV. RUNNABLE AVERAGE EXECUTION

Period	Average Execution Tim		
	Min.	Avg.	
1 ms	0,34	5,00	
2 ms	0,32	4,20	
5 ms	0,36	11,04	
10 ms	0,21	10,09	
20 ms	0,25	8,74	
50 ms	0,29	17,56	
100 ms	0,21	10,53	
200 ms	0,22	2,56	
1000 ms	0,37	0,43	
angle-synchronous	0,45	6,52	
Interrupts	0,18	5,42	12,59

TABLE V. FACTORS FOR DETERMINING RUNNABLE BEST- AND WORST-CASE EXECUTION TIMES

Period	Best		Worst	
	f_{min}	f_{max}	f_{min}	f_{max}
1 ms	0,19	0,92	1,30	29,11
2 ms	0,12	0,89	1,54	19,04
5 ms	0,17	0,94	1,13	18,44
10 ms	0,05	0,99	1,06	30,03
20 ms	0,11	0,98	1,06	15,61
50 ms	0,32	0,95	1,13	7,76
100 ms	0,09	0,99	1,02	8,88
200 ms	0,45	0,98	1,03	4,90
1000 ms	0,68	0,80	1,84	4,75
angle-synchronous	0,13	0,92	1,20	28,17
Interrupts	0,12	0,94	1,15	4,54

* Kramer *et al.* “Real World Automotive Benchmarks for Free.” WATERS (2015)

Workload: Real-Automotive Benchmark from Bosch*

Generated multiple random workload instances

- Mix of tasks with time periods in {50, 100, 200, 1000} ms
- Number of keys between 100 and 1000
- Value sizes ranging from 1 to 16 bytes

* Kramer *et al.* “Real World Automotive Benchmarks for Free.” WATERS (2015)

Workload: Real-Automotive Benchmark from Bosch*

Generated multiple random workload instances

- Mix of tasks with time periods in {50, 100, 200, 1000} ms
- Number of keys between 100 and 1000
- Value sizes ranging from 1 to 16 bytes

Objective: Run workload replicas on separate nodes synchronously

* Kramer *et al.* “Real World Automotive Benchmarks for Free.” WATERS (2015)

Workload: Real-Automotive Benchmark from Bosch*

Generated multiple random workload instances

- Mix of tasks with time periods in {50, 100, 200, 1000} ms
- Number of keys between 100 and 1000
- Value sizes ranging from 1 to 16 bytes

Objective: Run workload replicas on separate nodes synchronously

Setup: Single core experiments, no redis or etcd

* Kramer *et al.* "Real World Automotive Benchmarks for Free." WATERS (2015)

Workload: Real-Automotive Benchmark from Bosch*

Generated multiple random workload instances

- Mix of tasks with time periods in {50, 100, 200, 1000} ms
- Number of keys between 100 and 1000
- Value sizes ranging from 1 to 16 bytes

Objective: Run workload replicas on separate nodes synchronously

Setup: Single core experiments, no redis or etcd

Metrics: Successful iteration % for read operations

- Writes to the local unpublished datastore are always successful

* Kramer *et al.* "Real World Automotive Benchmarks for Free." WATERS (2015)

Results

Keys	Read success %				KVS latency (ms)	
	Pi 1	Pi 2	Pi 3	Pi 4	ACET	WCET
270						
363						
476						
573						
689						
744						
849						
905						

Results

Keys	Read success %				KVS latency (ms)	
	Pi 1	Pi 2	Pi 3	Pi 4	ACET	WCET
270	100.00	100.00	100.00	100.00	8.34	28.97
363	100.00	100.00	100.00	100.00	12.21	23.64
476	100.00	100.00	100.00	100.00	14.87	32.88
573	100.00	100.00	100.00	99.85	17.50	35.36
689	100.00	100.00	100.00	100.00	17.89	35.62
744						
849						
905						

Successful replica coordination for more complex workloads

Results

Keys	Read success %				KVS latency (ms)	
	Pi 1	Pi 2	Pi 3	Pi 4	ACET	WCET
270	100.00	100.00	100.00	100.00	8.34	28.97
363	100.00	100.00	100.00	100.00	12.21	23.64
476	100.00	100.00	100.00	100.00	14.87	32.88
573	100.00	100.00	100.00	99.85	17.50	35.36
689	100.00	100.00	100.00	100.00	17.89	35.62
744	100.00	99.97	99.96	100.00	19.72	35.86
849	99.81	99.91	99.74	99.98	20.43	36.59
905	98.71	99.56	44.82	99.07	23.49	41.82

Successful replica coordination for more complex workloads

WCET closer to 50ms, which is also the T_{min}, results in little slack and increased deadline violations

Results

Keys	Read success %				KVS latency (ms)	
	Pi 1	Pi 2	Pi 3	Pi 4	ACET	WCET
270	100.00	100.00	100.00	100.00	8.34	28.97
363	100.00	100.00	100.00	100.00	12.21	23.64
476	100.00	100.00	100.00	100.00	14.87	32.88
573	100.00	100.00	100.00	99.85	17.50	35.36
689	100.00	100.00	100.00	100.00	17.89	35.62
744	100.00	99.97	99.96	100.00	19.72	35.86
849	99.81	99.91	99.74	99.98	20.43	36.59
905	98.71	99.56	44.82	99.07	23.49	41.82

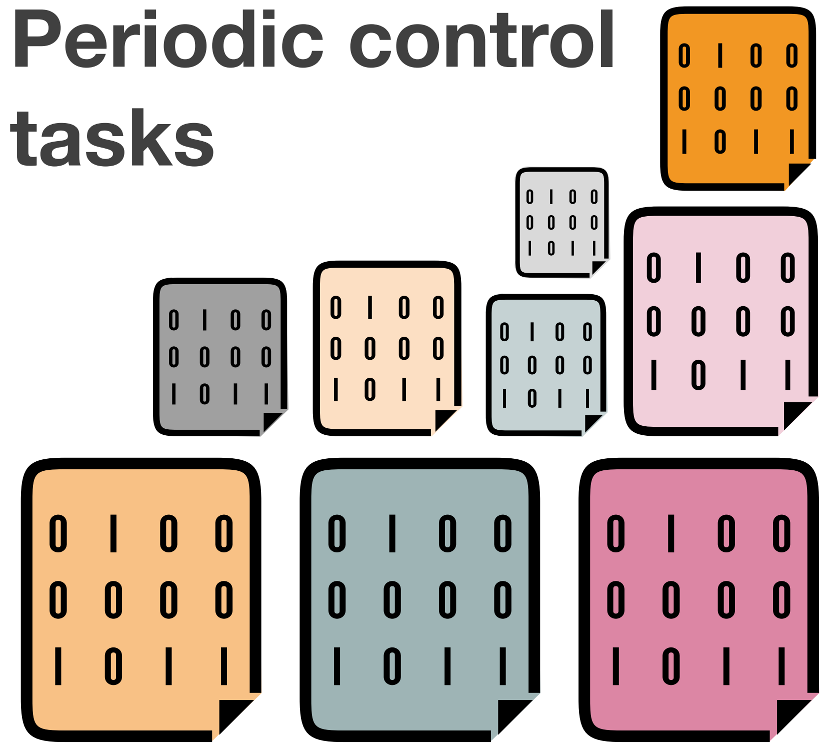
Successful replica coordination for more complex workloads

WCET closer to 50ms, which is also the T_{min}, results in little slack and increased deadline violations

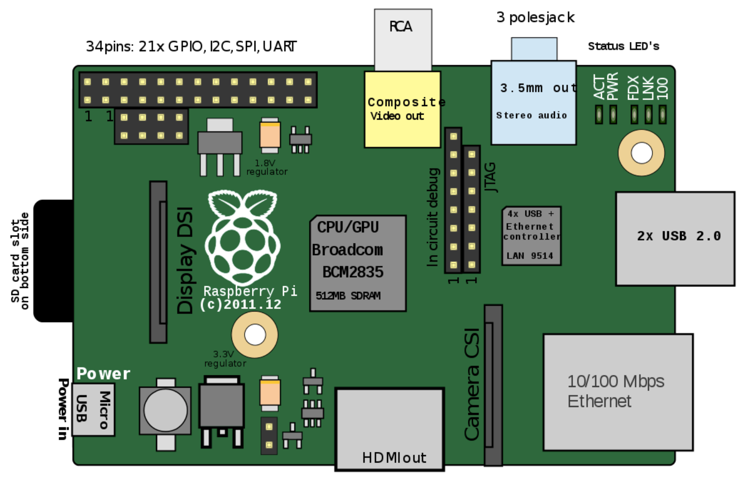
Increasing the #keys pushes In-ConcReTeS on the Raspberry Pi cluster to its limits!

Summary

Periodic control tasks



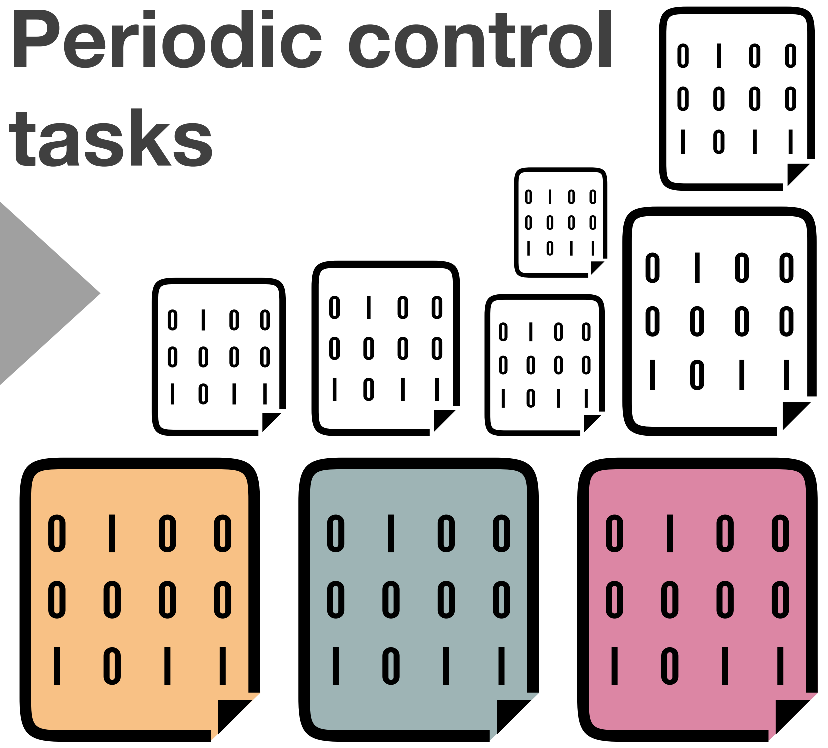
Embedded platform (node 1)



Data sharing

Fault tolerance / replica coordination

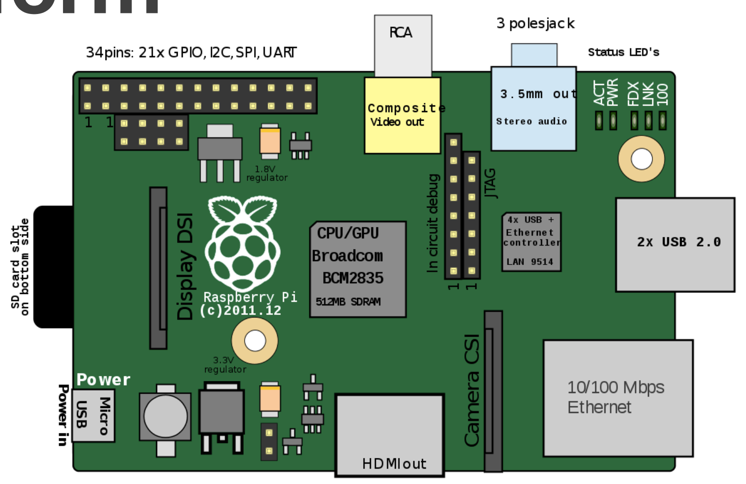
Periodic control tasks



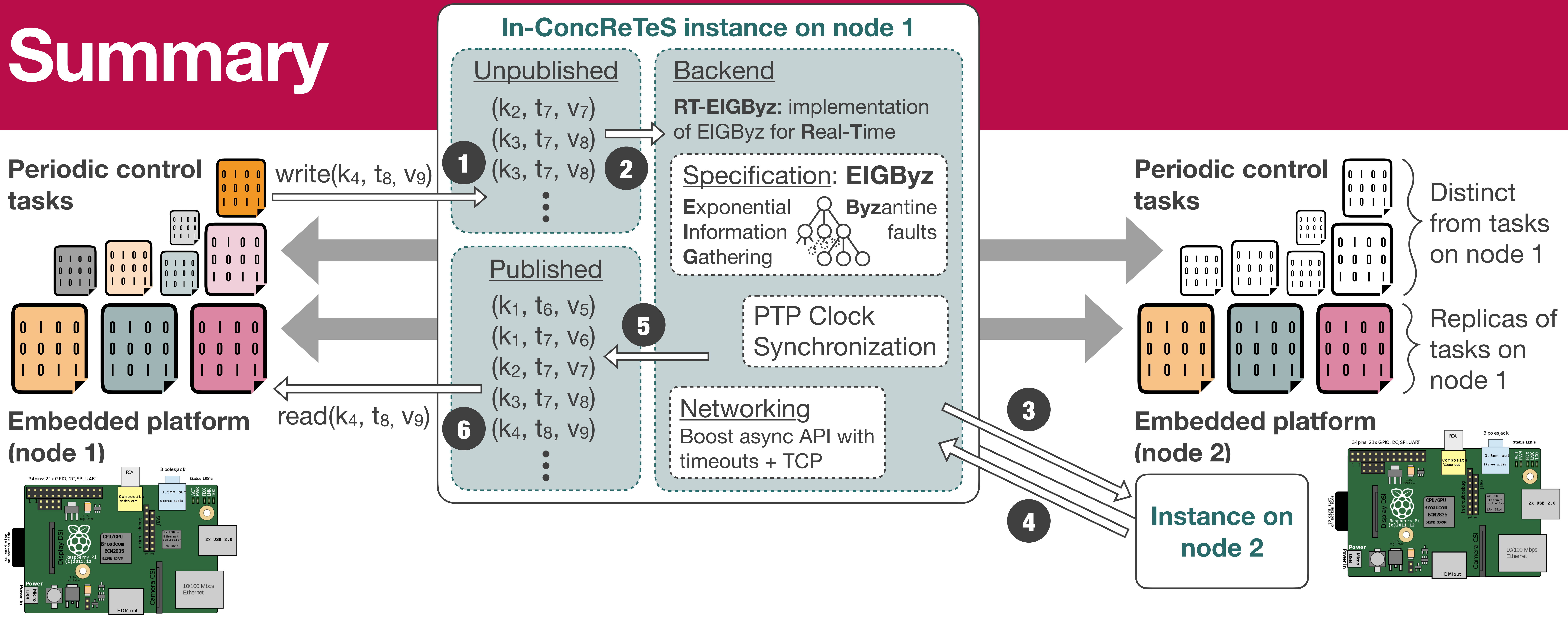
Distinct from tasks on node 1

Replicas of tasks on node 1

Embedded platform (node 2)

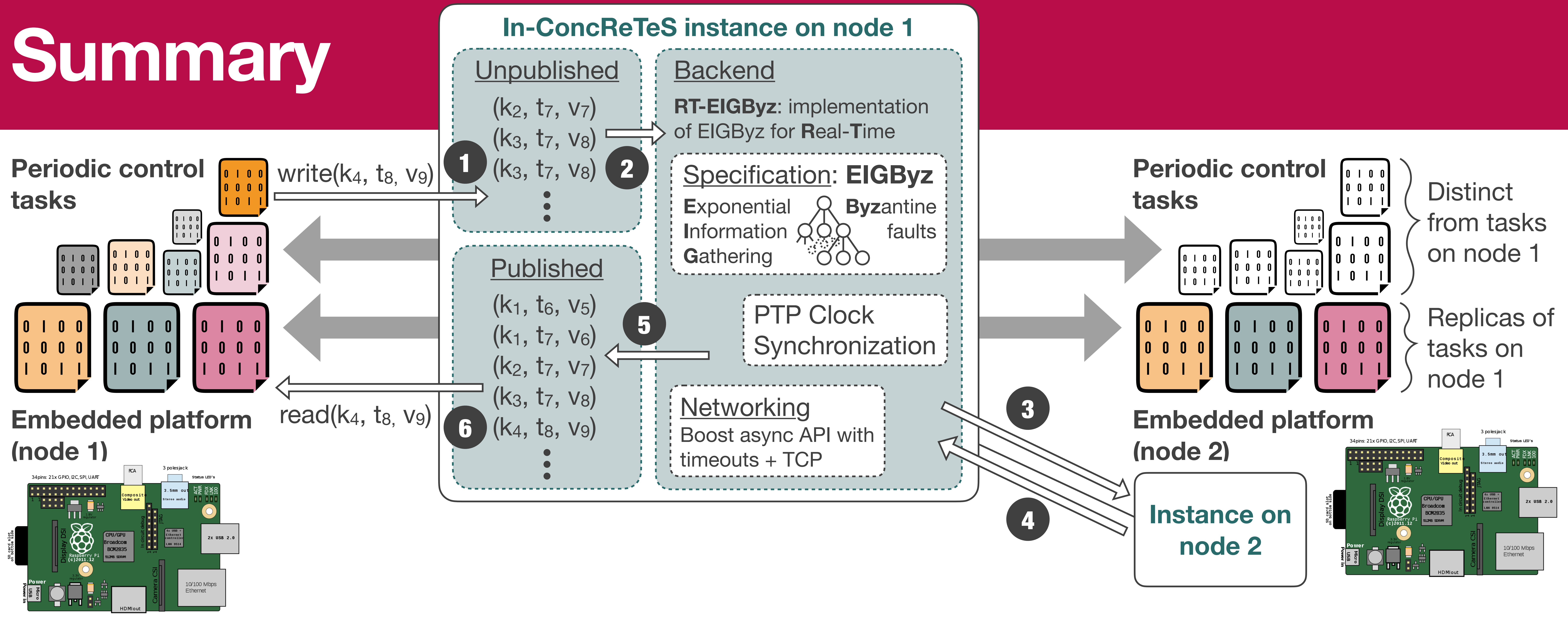


Summary



In-ConcReTeS instance on node 1
Unpublished
(k_2, t_7, v_7)
(k_3, t_7, v_8)
(k_3, t_7, v_8)
Backend
RT-EIGByz: implementation of EIGByz for Real-Time
Specification: EIGByz
Exponential Information Gathering
Byzantine faults
Published
(k_1, t_6, v_5)
(k_1, t_7, v_6)
(k_2, t_7, v_7)
(k_3, t_7, v_8)
(k_4, t_8, v_9)
PTP Clock Synchronization
Networking
Boost async API with timeouts + TCP
1
2
3
4
5
6
Periodic control tasks
write(k_4, t_8, v_9)
read(k_4, t_8, v_9)
Embedded platform (node 1)
Periodic control tasks
Distinct from tasks on node 1
Replicas of tasks on node 1
Embedded platform (node 2)
Instance on node 2

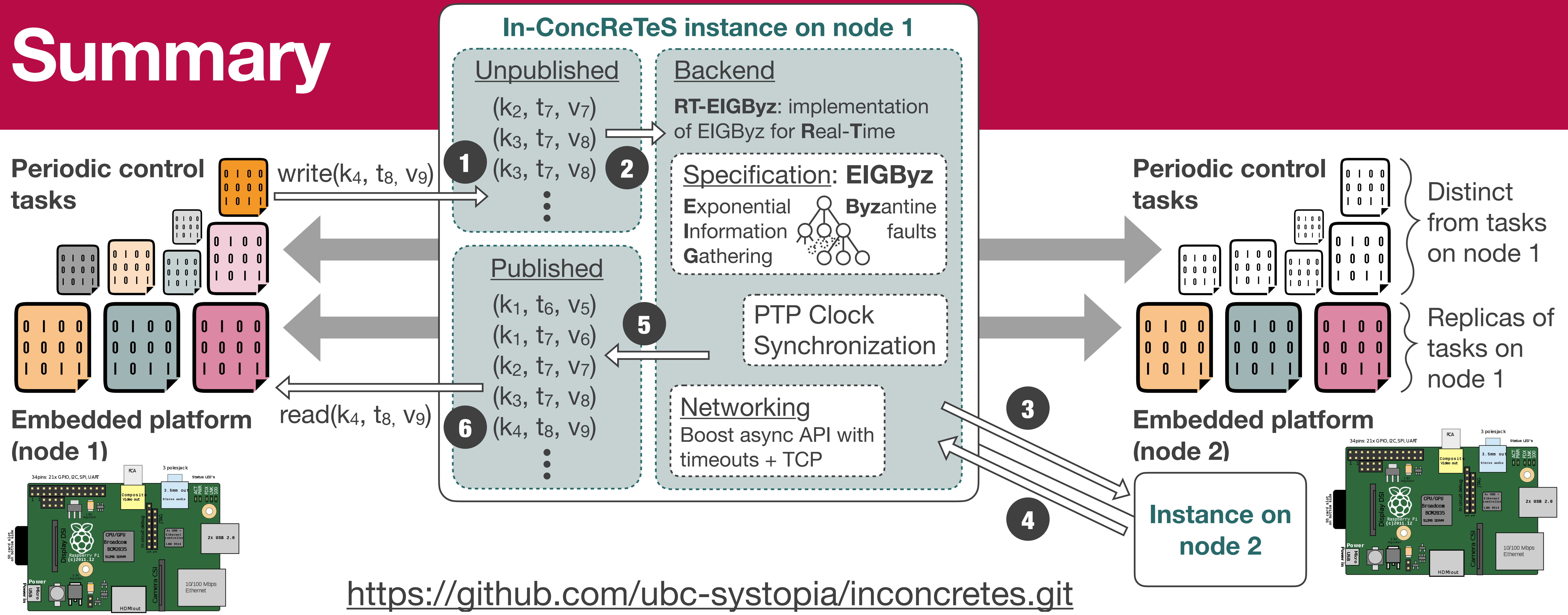
Summary



By making **design choices** around the needs of **periodic real-time applications**

- **In-ConcReTeS is able to outperform** generic datastores such as redis and etcd

Summary



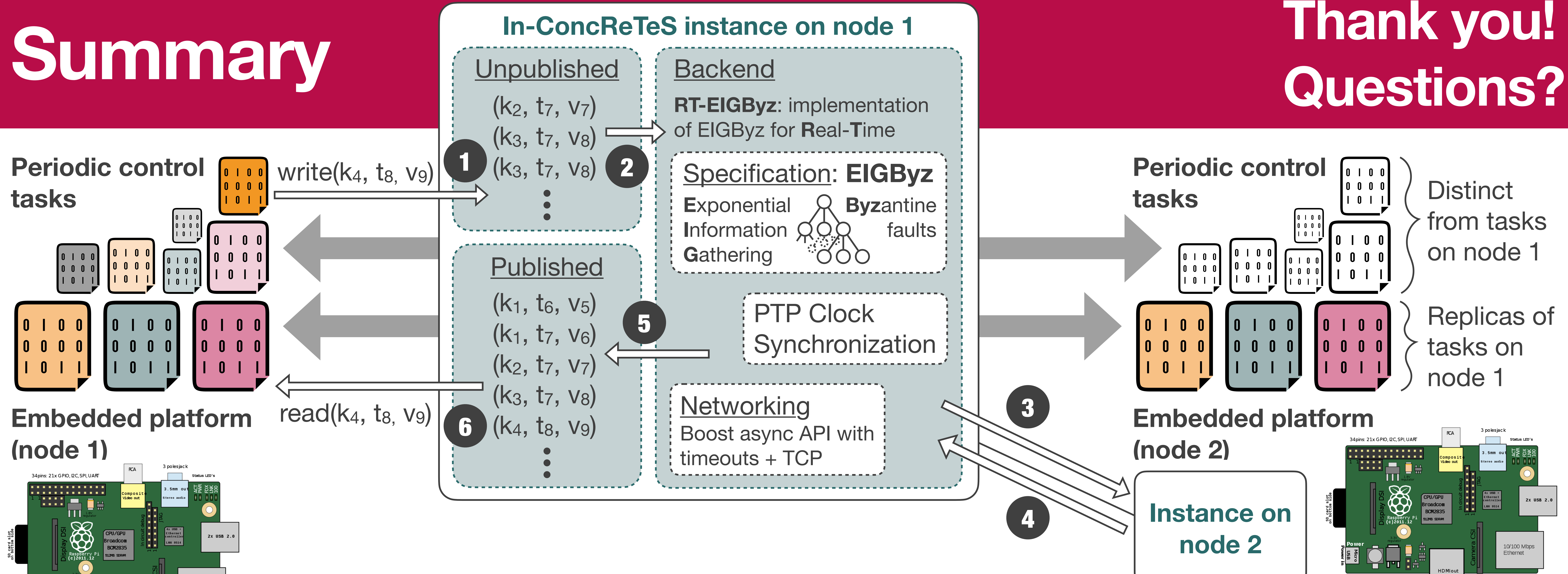
https://github.com/ubc-systopia/inconcretes.git

By making **design choices** around the needs of **periodic real-time applications**

- **In-ConcReTeS is able to outperform** generic datastores such as redis and etcd

Summary

Thank you! Questions?



https://github.com/ubc-systopia/inconcretes.git

By making **design choices** around the needs of **periodic real-time applications**

- **In-ConcReTeS is able to outperform** generic datastores such as redis and etcd