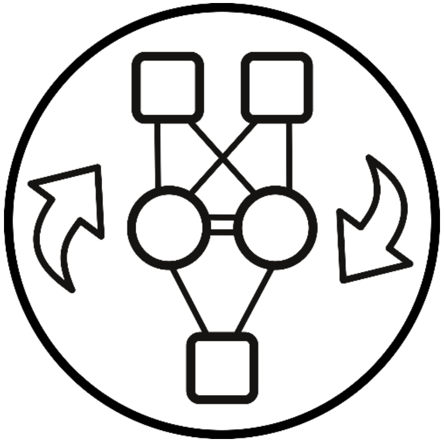
Towards a Self-Reconfigurable Infrastructure for Critical Adaptive Distributed Embedded Systems

Alberto Ballesteros
Julián Proenza
Manuel Barranco
Luís Almeida
Pere Palmer



MINISTERIO DE ECONOMÍA Y COMPETITIVIDAD
FEDER Fondo Europeo de Desarrollo Regional

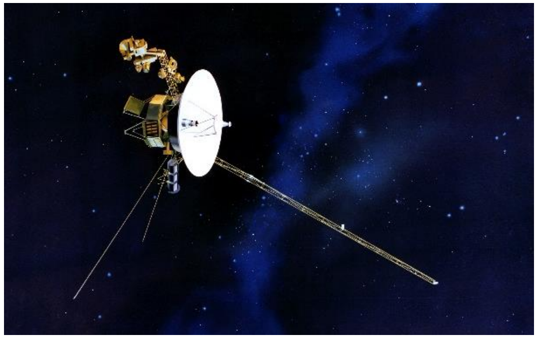
Introduction

Distributed Embedded Systems typically have stringent **real-time** and **dependability** requirements.

When they have to operate under **dynamic environments**, they must also be **flexible** to be able to **adapt** to the **changing operational requirements** and **conditions**.

Introduction

Adaptive Distributed Embedded Systems (ADES) can **rearrange** themselves **autonomously** and **dynamically**



Introduction

Adaptive Distributed Embedded Systems (ADES) can **rearrange** themselves **autonomously** and **dynamically**

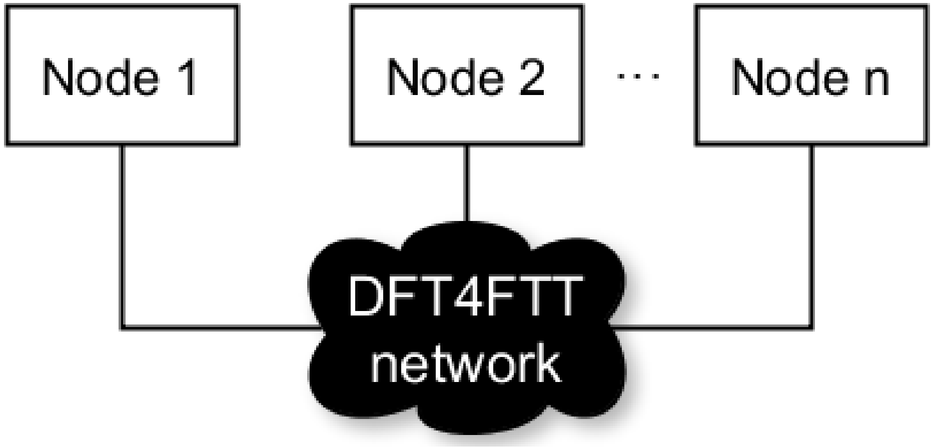
Adaptivity is and interesting feature in terms of:

- **Functionality** → Change the behaviour
- **Efficiency** → Load the necessary functionalities
- **Dependability** → Adaptive fault tolerance

Introduction

To **properly implement an ADES** it must be provided with the appropriate **architecture** and **mechanisms**, that make it possible to fulfil its **real-time**, **dependability** and **adaptivity** requirements

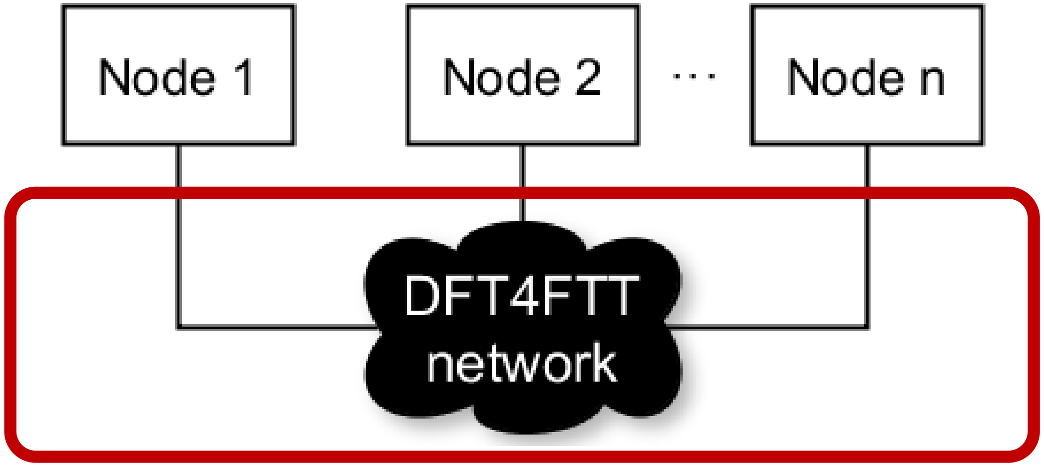
The DFT4FTT project



Introduction

To **properly implement an ADES** it must be provided with the appropriate **architecture** and **mechanisms**, that make it possible to fulfil its **real-time**, **dependability** and **adaptivity** requirements

The DFT4FTT project



Flexible Time-Triggered (FTT) communication paradigm

- Real-time
- Flexibility

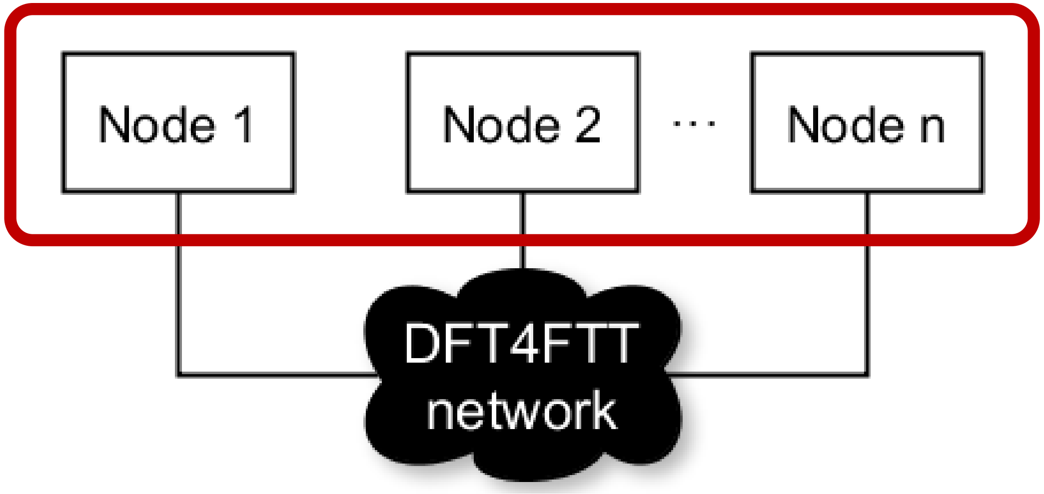
FTT Replicated Star (FTTRS)

- Reliability

Introduction

To **properly implement an ADES** it must be provided with the appropriate **architecture** and **mechanisms**, that make it possible to fulfil its **real-time**, **dependability** and **adaptivity** requirements

The DFT4FTT project

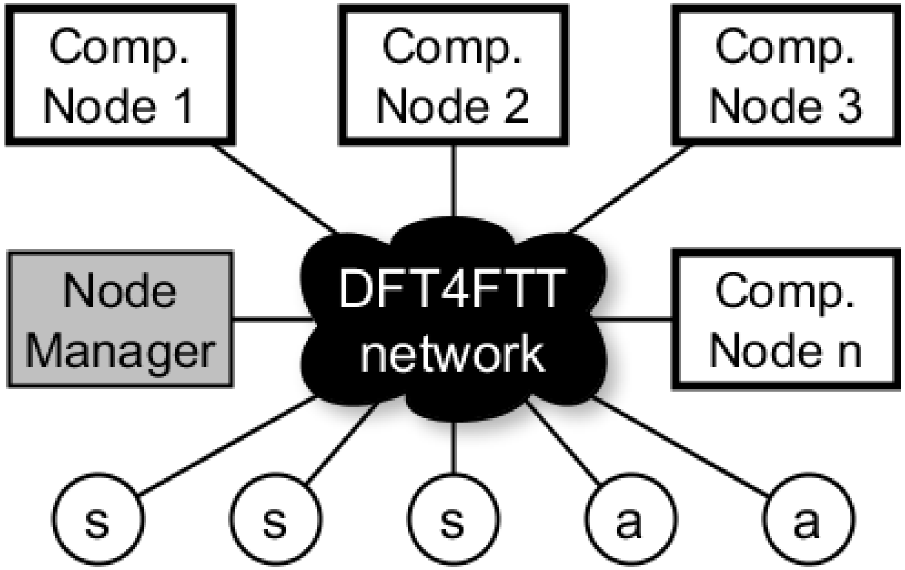


Dynamic task allocation
- Flexibility
- Real-time

Active replication with majority voting
- Reliability

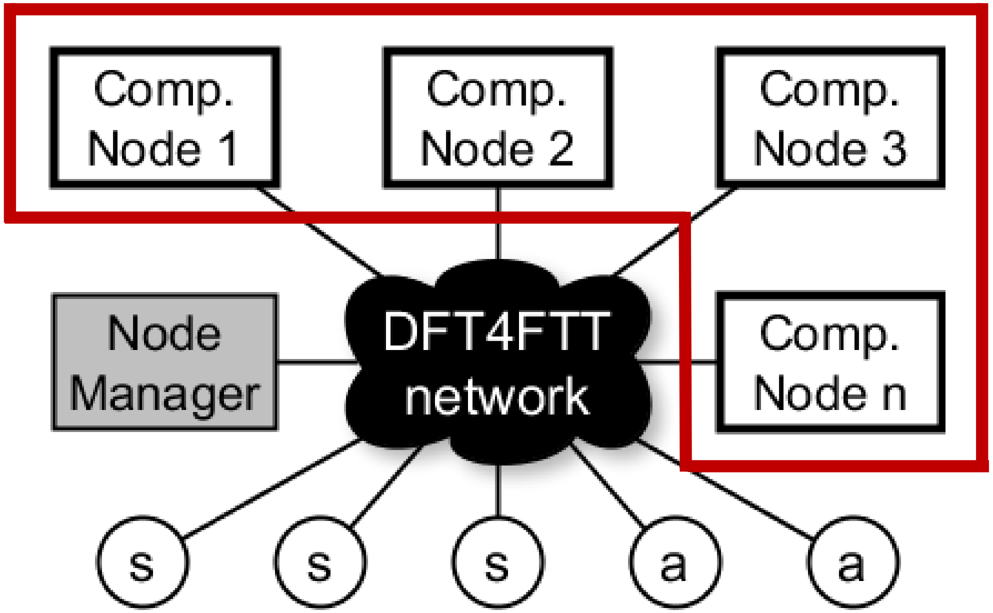
Introduction

At the **node level**, the DFT4FTT architecture is composed of **various components**



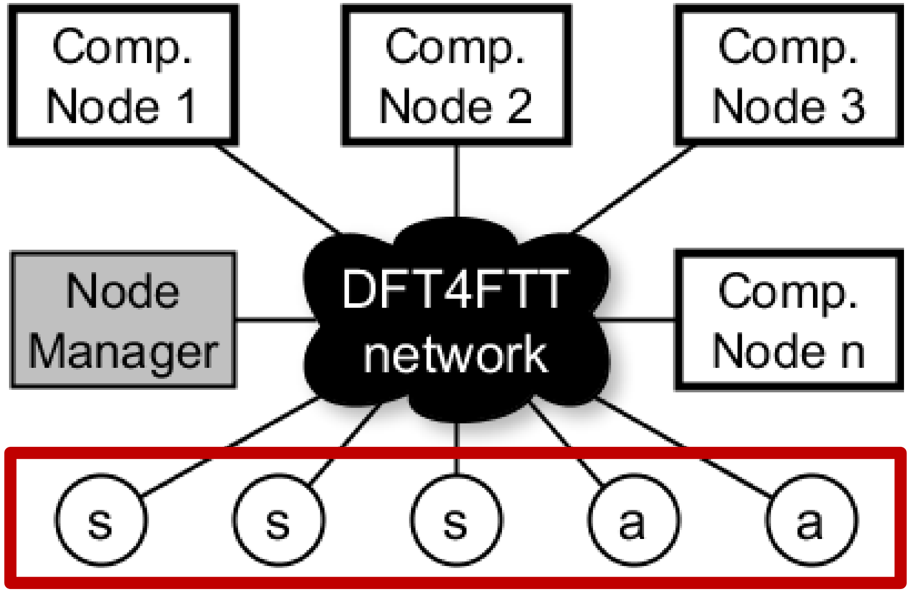
Introduction

At the **node level**, the DFT4FTT architecture is composed of **various components**



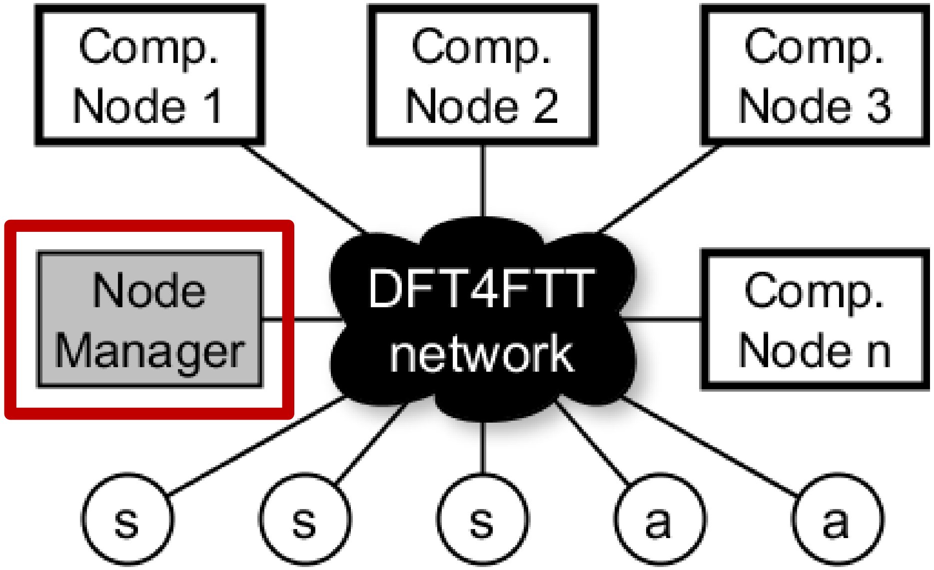
Introduction

At the **node level**, the DFT4FTT architecture is composed of **various components**



Introduction

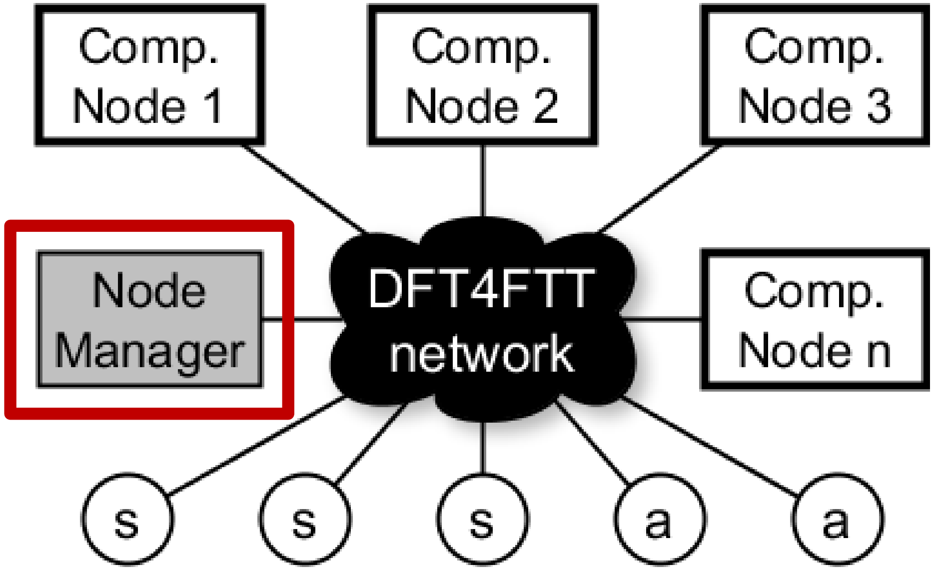
At the **node level**, the DFT4FTT architecture is composed of **various components**



Introduction

At the **node level**, the DFT4FTT architecture is composed of **various components**

- **Monitor**
- **Detect**
- **Configuration change**



Outline

Outline

1. The task model

2. The Self-Reconfiguration
    - 2.1 Monitoring Process
    - 2.2 Decision Process
    - 2.3 Configuration Change Process

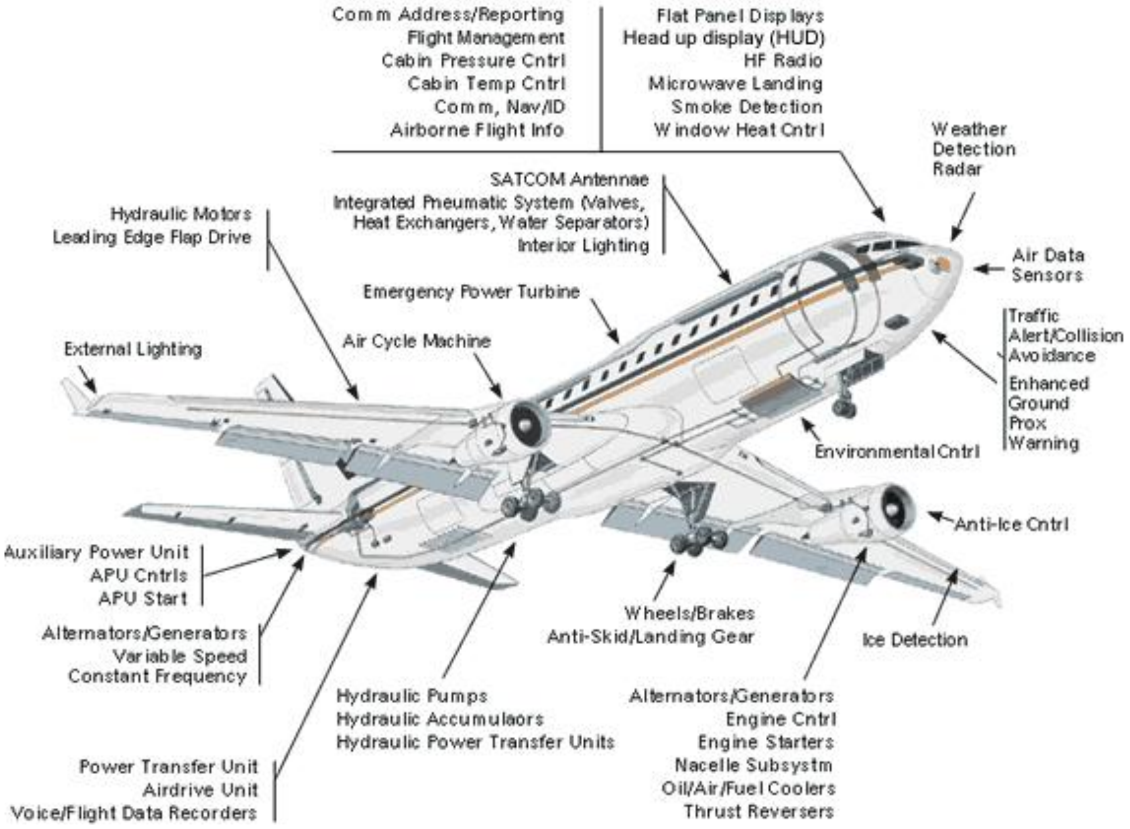
3. Reconfiguration for Reliability

4. Conclusions and On-going Work

Functionality

Task Model

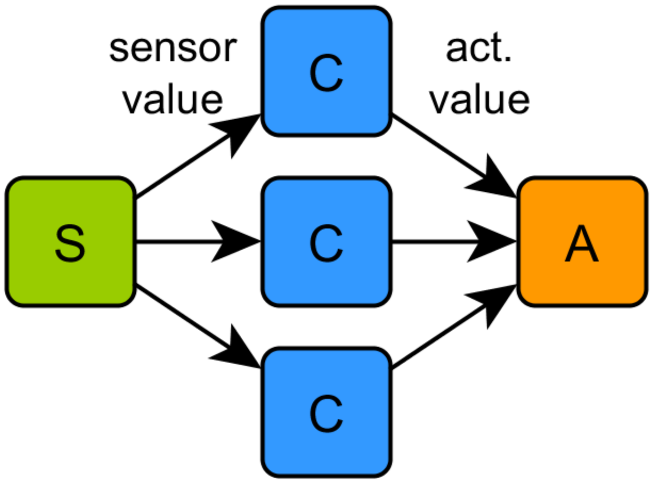
Functionality



Task Model

Functionality → Application

Application: Set of **distributed** and **interconnected tasks** that are executed in a **sequential** or **parallel** manner



Task Model

Functionality → Application

Application: Set of **distributed** and **interconnected tasks** that are executed in a **sequential** or **parallel** manner

Task Model

Functionality → Application

Application: Set of **distributed** and **interconnected tasks** that are executed in a **sequential** or **parallel** manner

Determine a **sequence** of **task executions** and **message transmissions** that allow to **meet the deadlines**

- Critical tasks are replicated
- Message replicas pro-actively transmitted

Outline

1. The task model

2. The Self-Reconfiguration

   2.1 Monitoring Process

   2.2 Decision Process

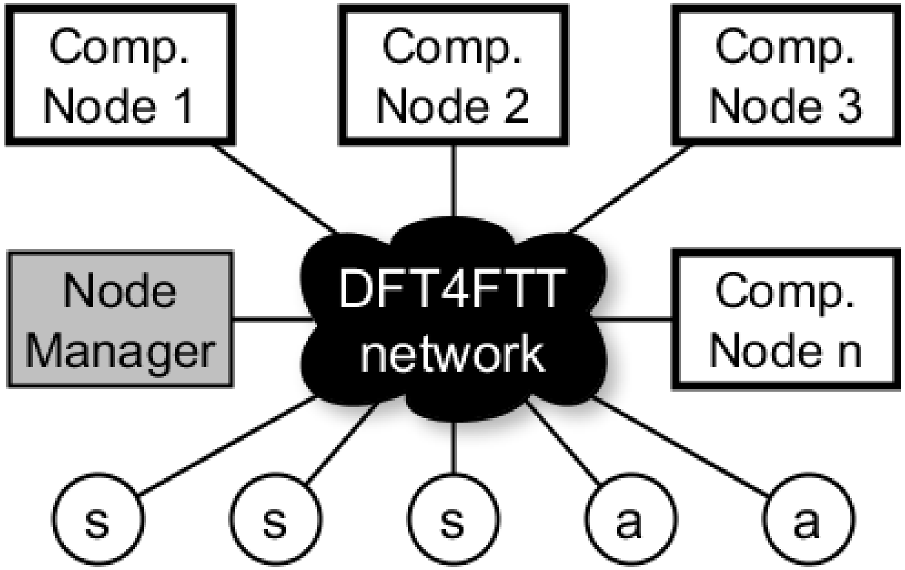
   2.3 Configuration Change Process

3. Reconfiguration for Reliability

4. Conclusions and On-going Work

The Self-Reconfiguration
Introduction

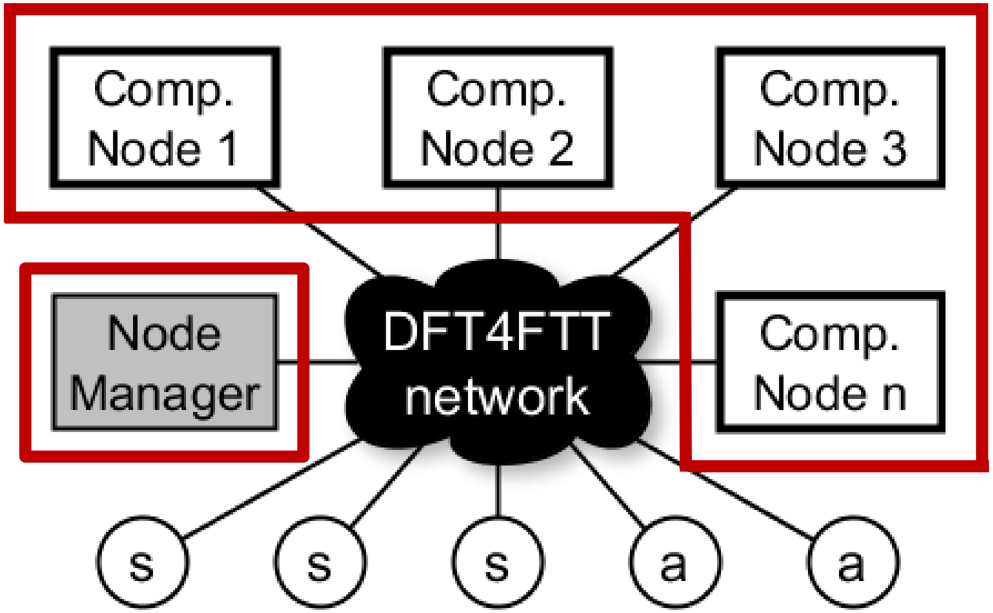
At the **node level**, the DFT4FTT architecture is composed of **various components**



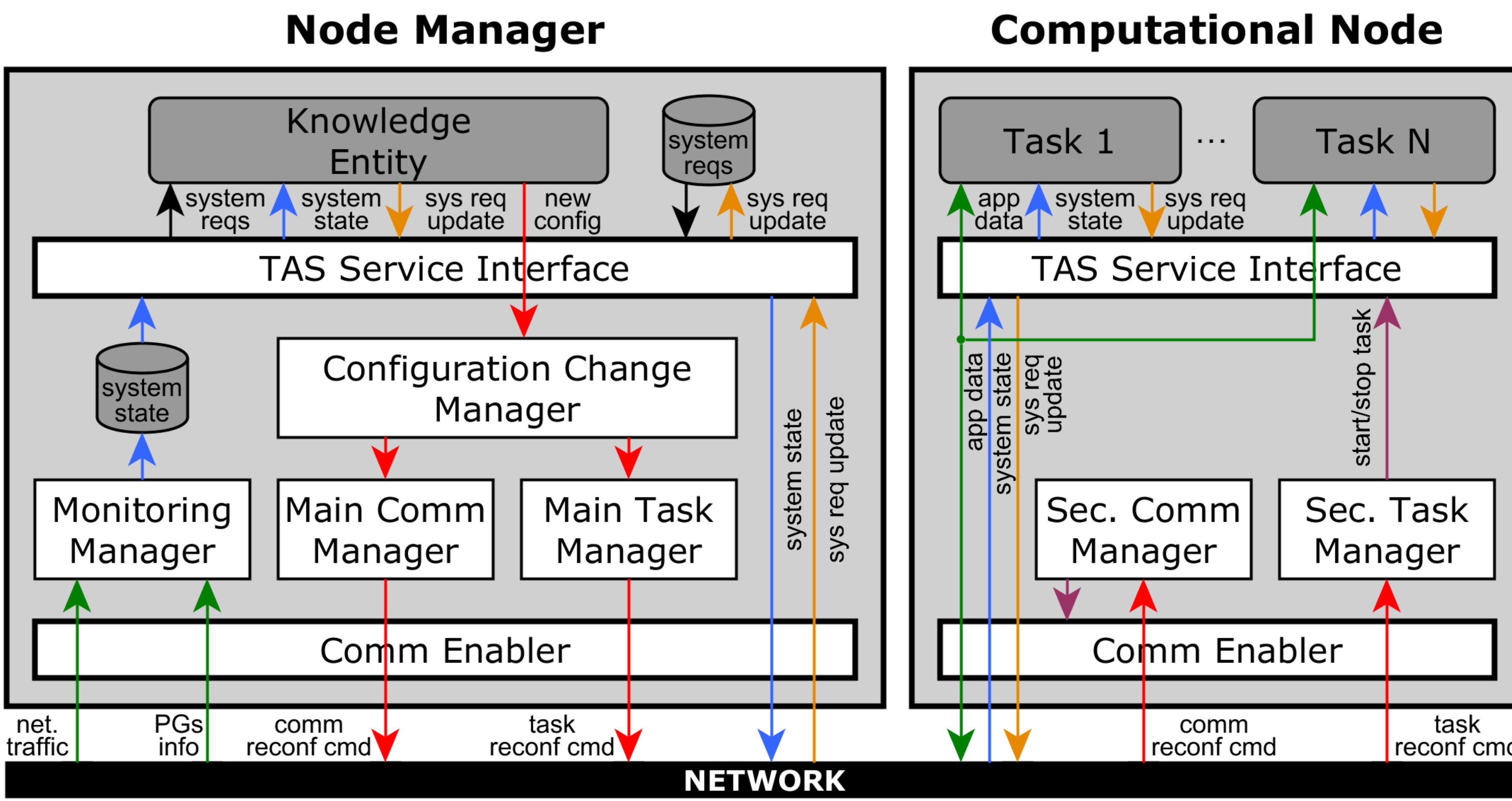
The Self-Reconfiguration

Introduction

At the **node level**, the DFT4FTT architecture is composed of **various components**

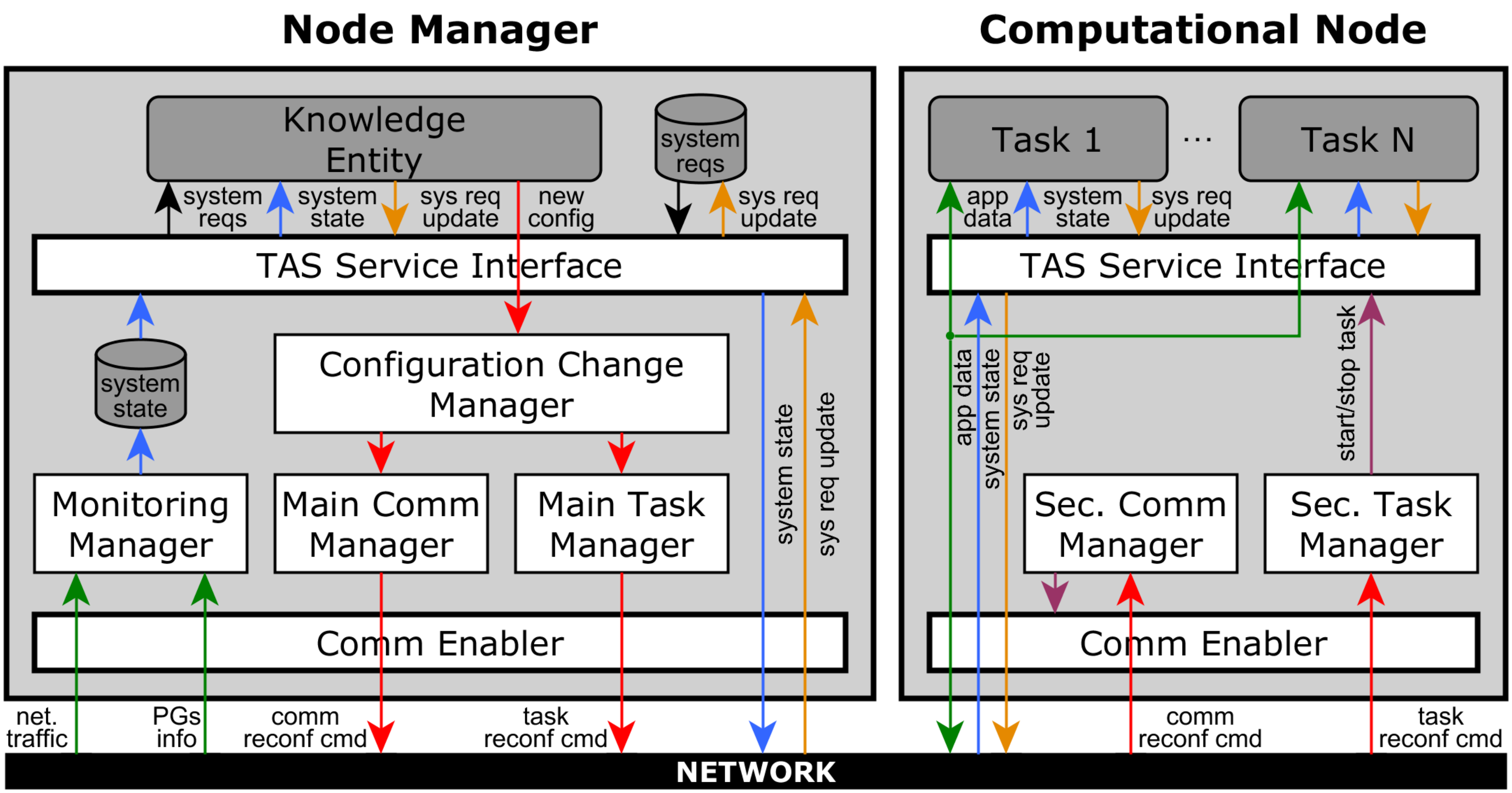


The Self-Reconfiguration
Introduction

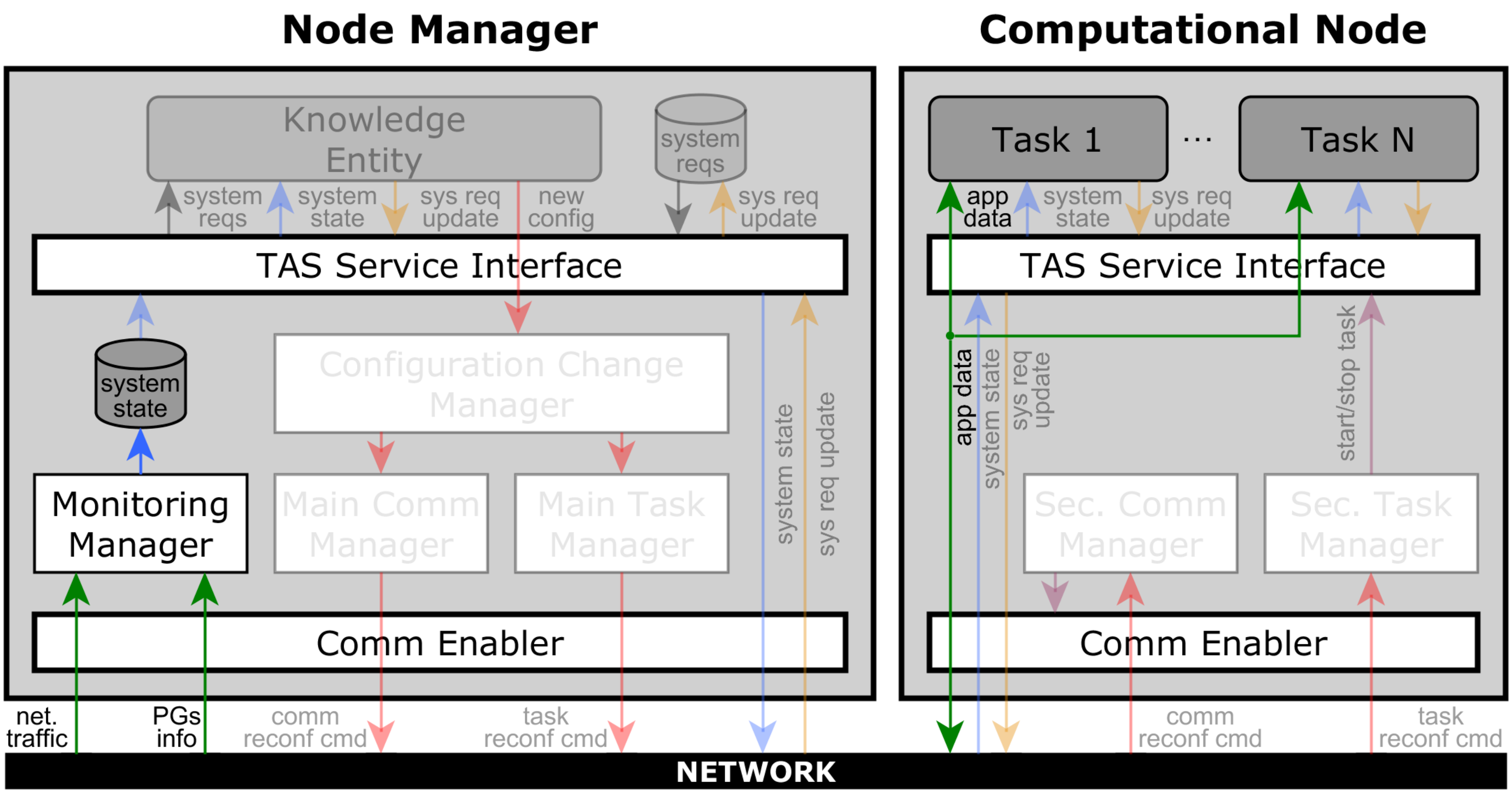


Outline

The Self-Reconfiguration
Introduction



The Self-Reconfiguration
Monitoring Process



The Self-Reconfiguration
Monitoring Process

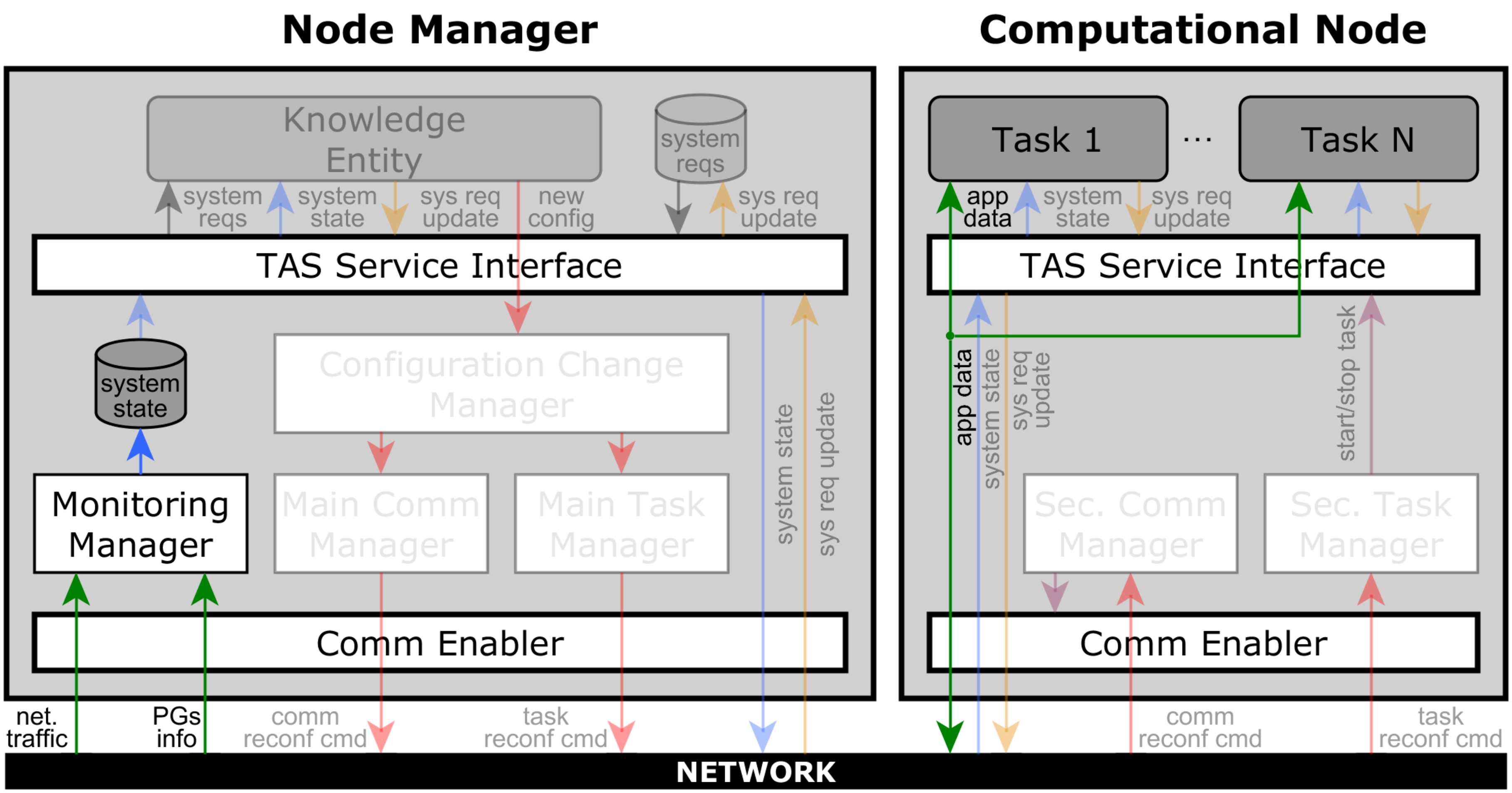
Monitor the **environment** and the **system itself**

Obtain the **system status**:

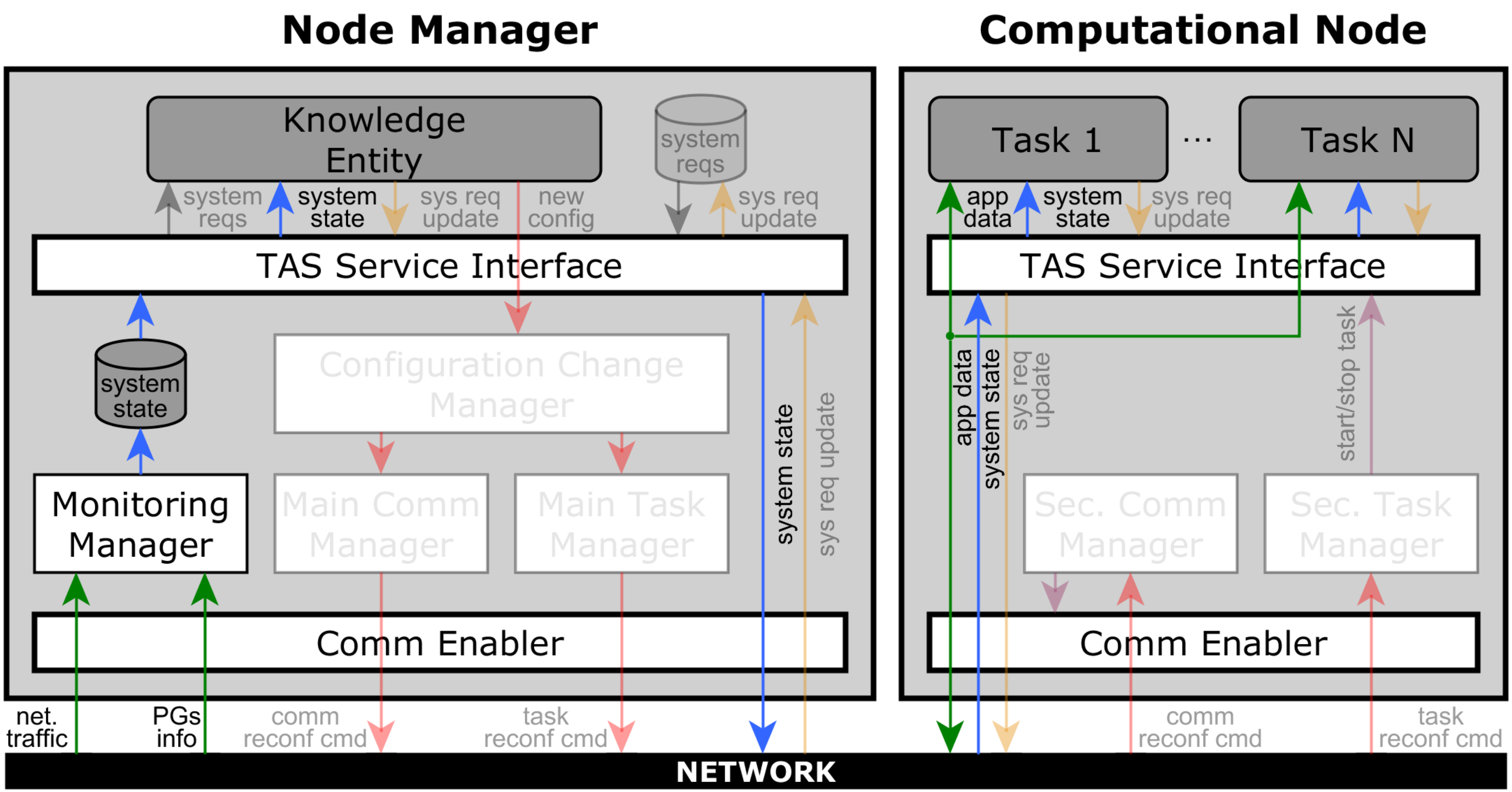
- **Status of the architecture** → Port Guardians (PGs)
- **Failure Rate and Bit Error Rate** → FR model, PGs and sensors
- **Status of the execution** → Messages sent by applications
- **Status of the resources** → Amount of application resources

The Self-Reconfiguration

Monitoring Process



The Self-Reconfiguration
Monitoring Process



Outline

1. The task model

2. The Self-Reconfiguration

    2.1 Monitoring Process

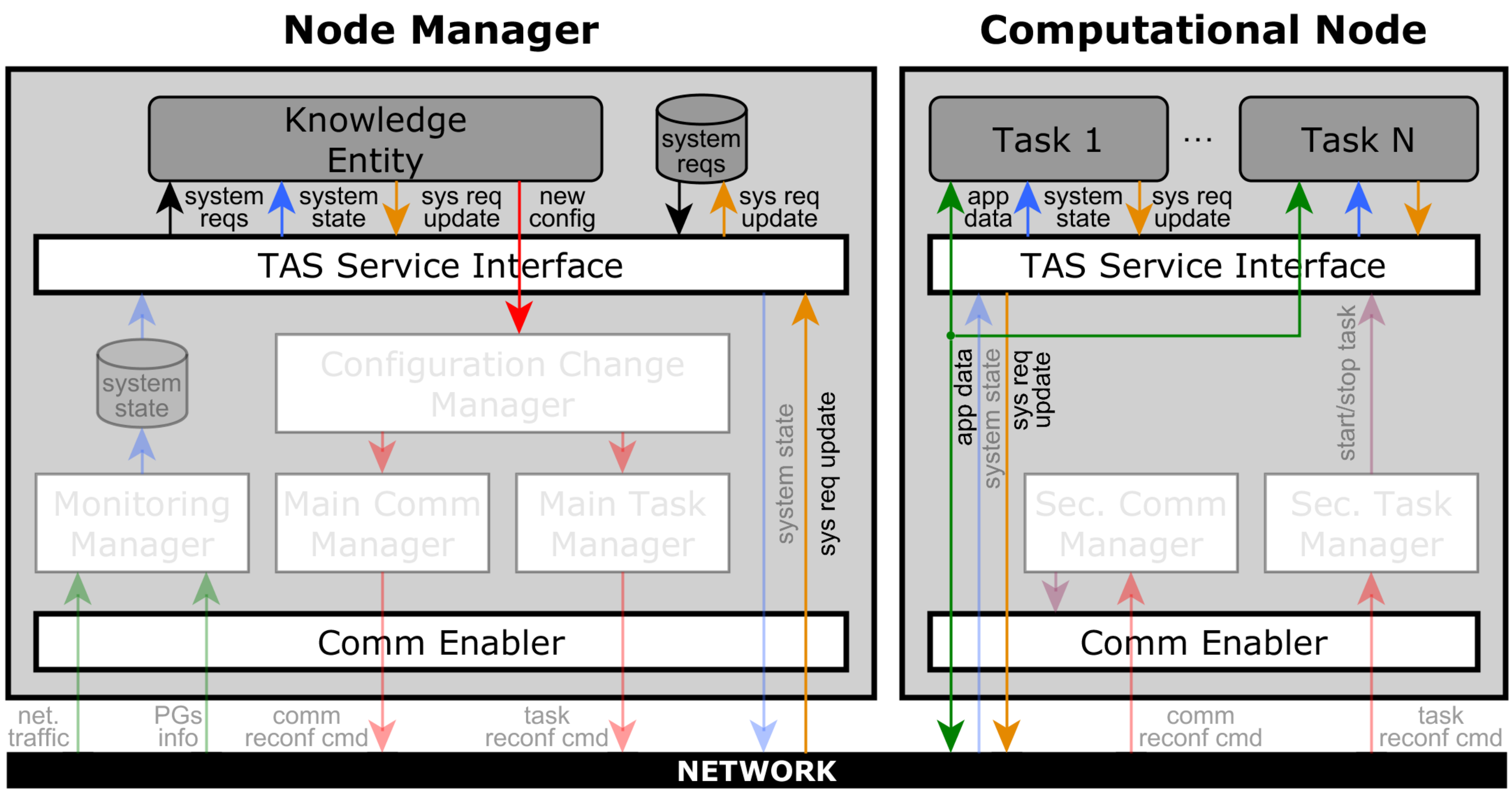
    2.2 Decision Process

    2.3 Configuration Change Process

3. Reconfiguration for Reliability

4. Conclusions and On-going Work

The Self-Reconfiguration
Decision Process



The Self-Reconfiguration
Decision Process

System requirements

List of applications, together with their **real-time and reliability requirements**, that have to be **executed**

- **Phase-related applications**

Indispensable applications needed to fulfil the **functional requirements** of a given **phase of the mission**. Maintained by the KE.

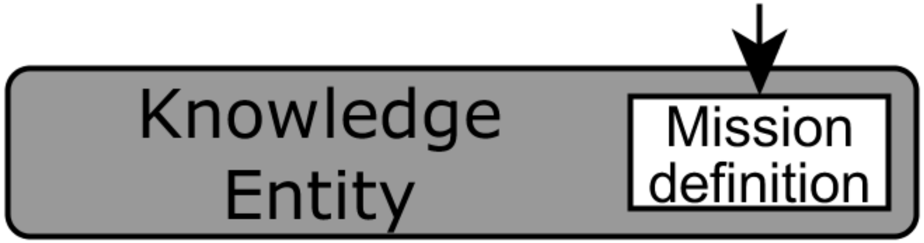
- **On-demand-related applications**

Indispensable and **non-indispensable applications** started as a result of a **new functional requirement**, **not related** to the **phase of the mission**. Maintained by the tasks.

The Self-Reconfiguration
Decision Process

Knowledge Entity

The KE **determines** when a **new phase** starts and **updates** the **system requirements** accordingly.



The KE **constantly consults** the **system state** and checks if the **conditions** associated to any of the **phases** are **met**

The Self-Reconfiguration
Decision Process

Tasks

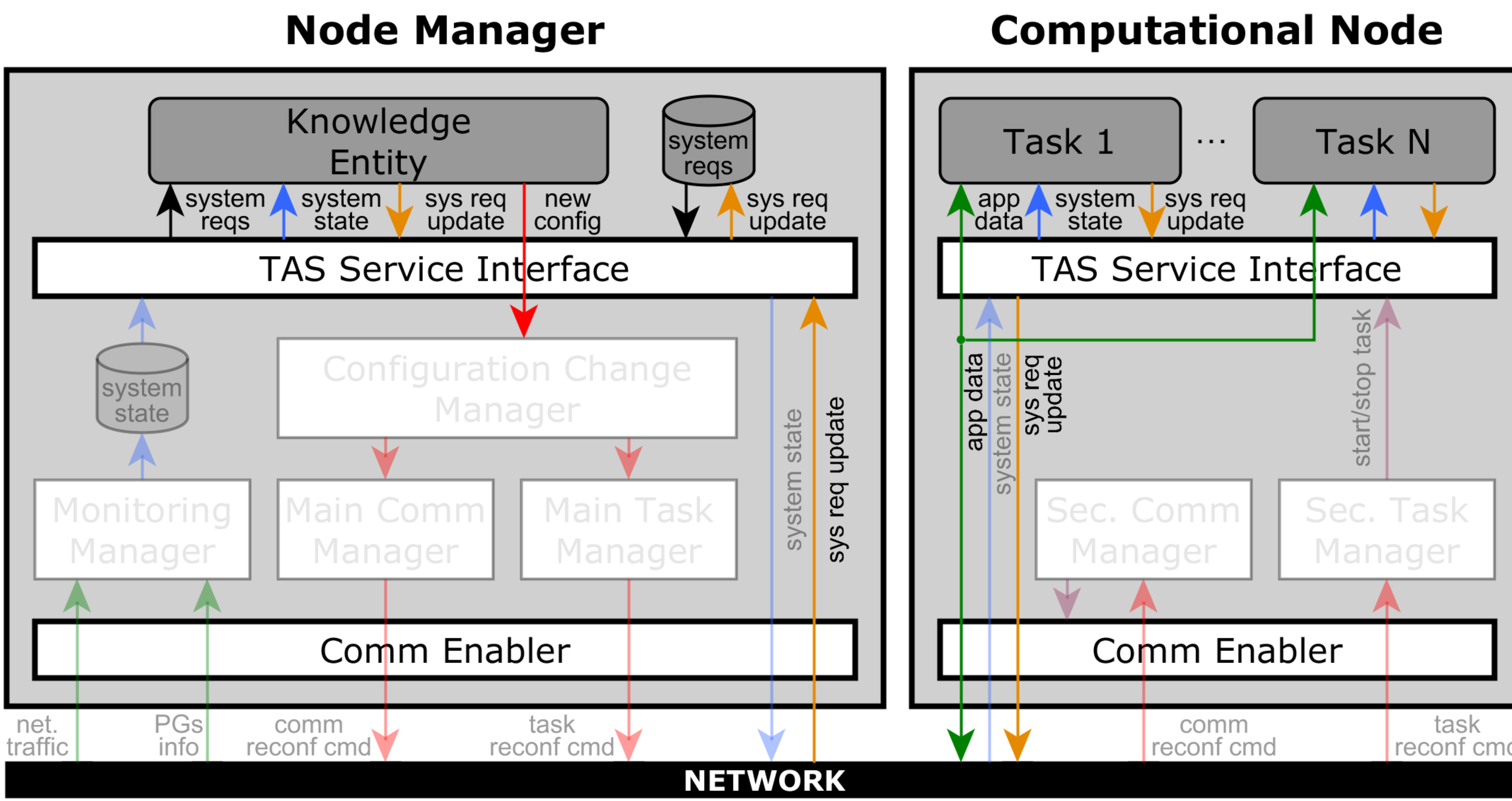
Tasks are the only system modules that know the **dynamic operational requirements** derived from **human commands** or the **tasks themselves**.

Start and **stop applications**, as well as to **modify** their **real-time** and **reliability requirements**.

Dependability issues:

- Use highly-reliable CN
- Replicate decision tasks and vote

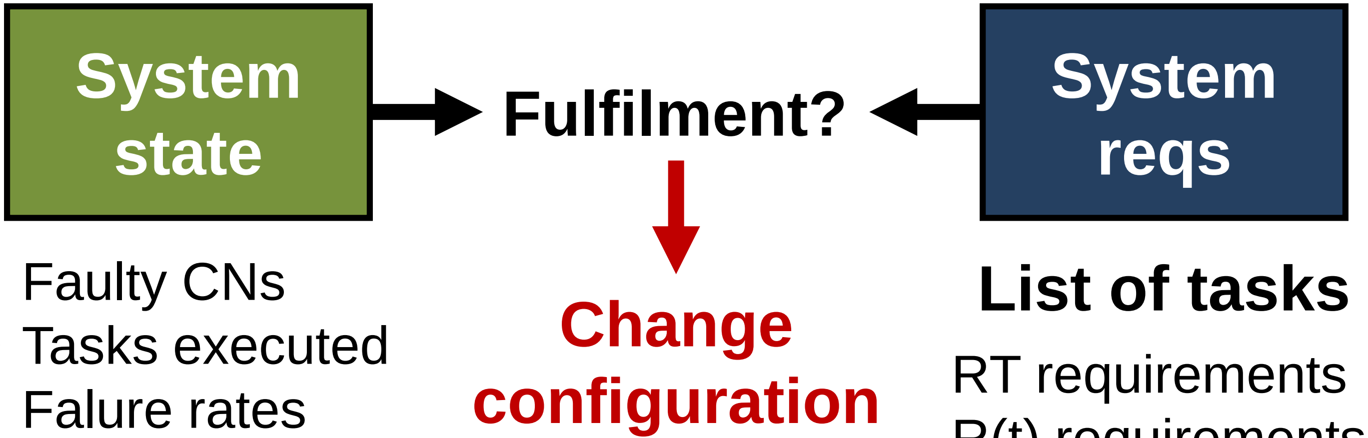
The Self-Reconfiguration
Decision Process



The Self-Reconfiguration

Decision Process

The KE **constantly verifies** that the **system reqs** are **fulfilled**



- Faulty CNs
- Tasks executed
- Falure rates
- …

List of tasks

RT requirements
R(t) requirements

The Self-Reconfiguration
Decision Process

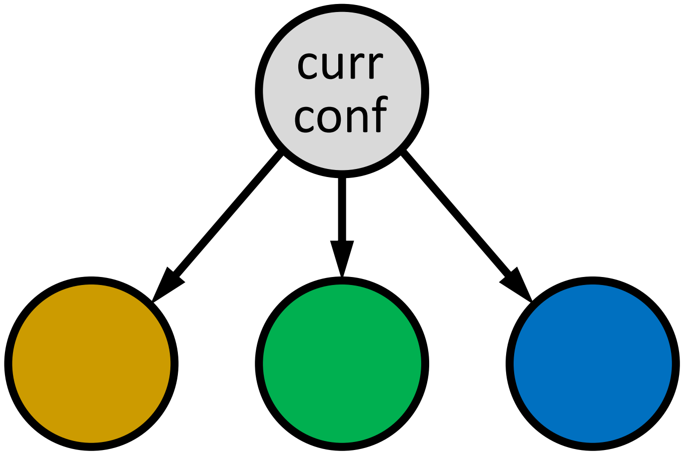
If the **system requirements** are **not fulfilled**, the **KE decides** on the **new configuration to apply**

Finding a new **proper configuration** can take **a lot of time**:

- Provide asap a good configuration for critical apps
- Provide asap a good configuration for non-critical apps
- Provide, while the system is running, a better configuration, i.e. good and optimal according to some specific policy
  - For instance: energy consumption, network performance, QoS, …
  - System designers specify the relevant policies
  - Score each configuration

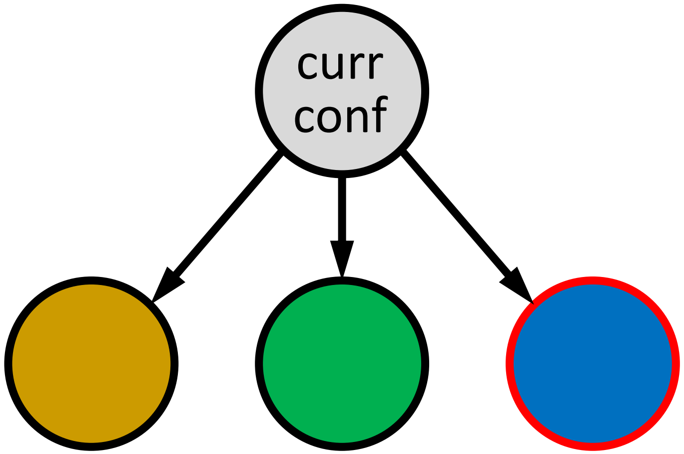
The Self-Reconfiguration
Decision Process

Branch and bound with a **greedy algorithm**



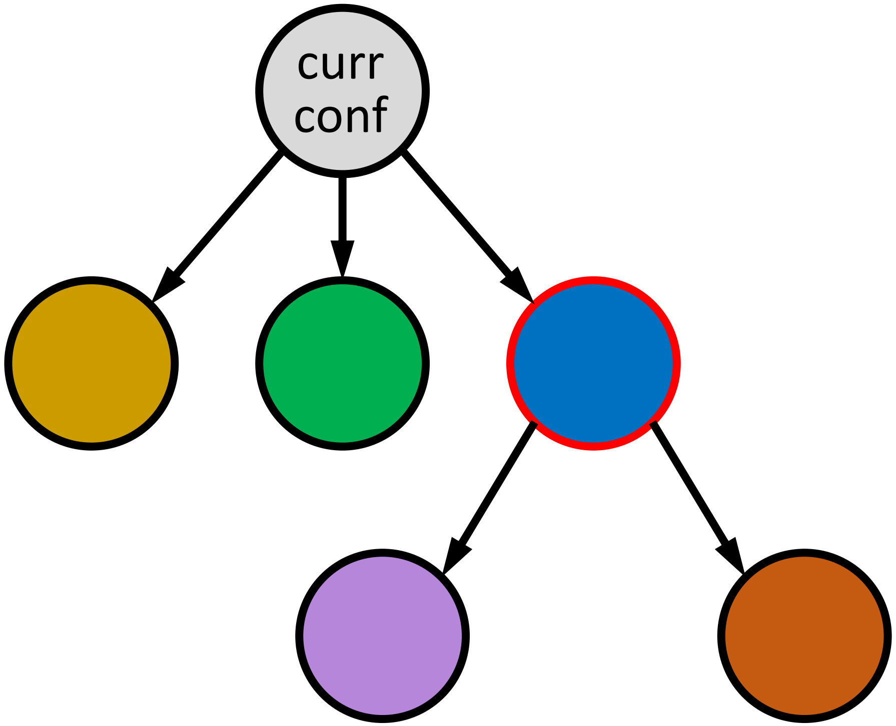
The Self-Reconfiguration
Decision Process

Branch and bound with a **greedy algorithm**



The Self-Reconfiguration
Decision Process

Branch and bound with a **greedy algorithm**



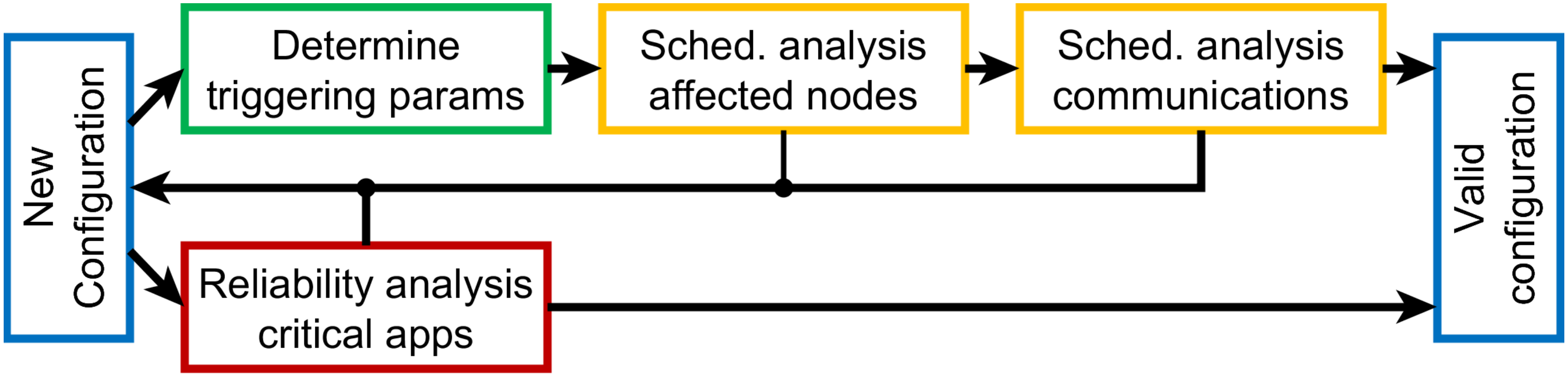
The Self-Reconfiguration
Decision Process

Validate functional requirements

- Check that all the tasks are in the configuration

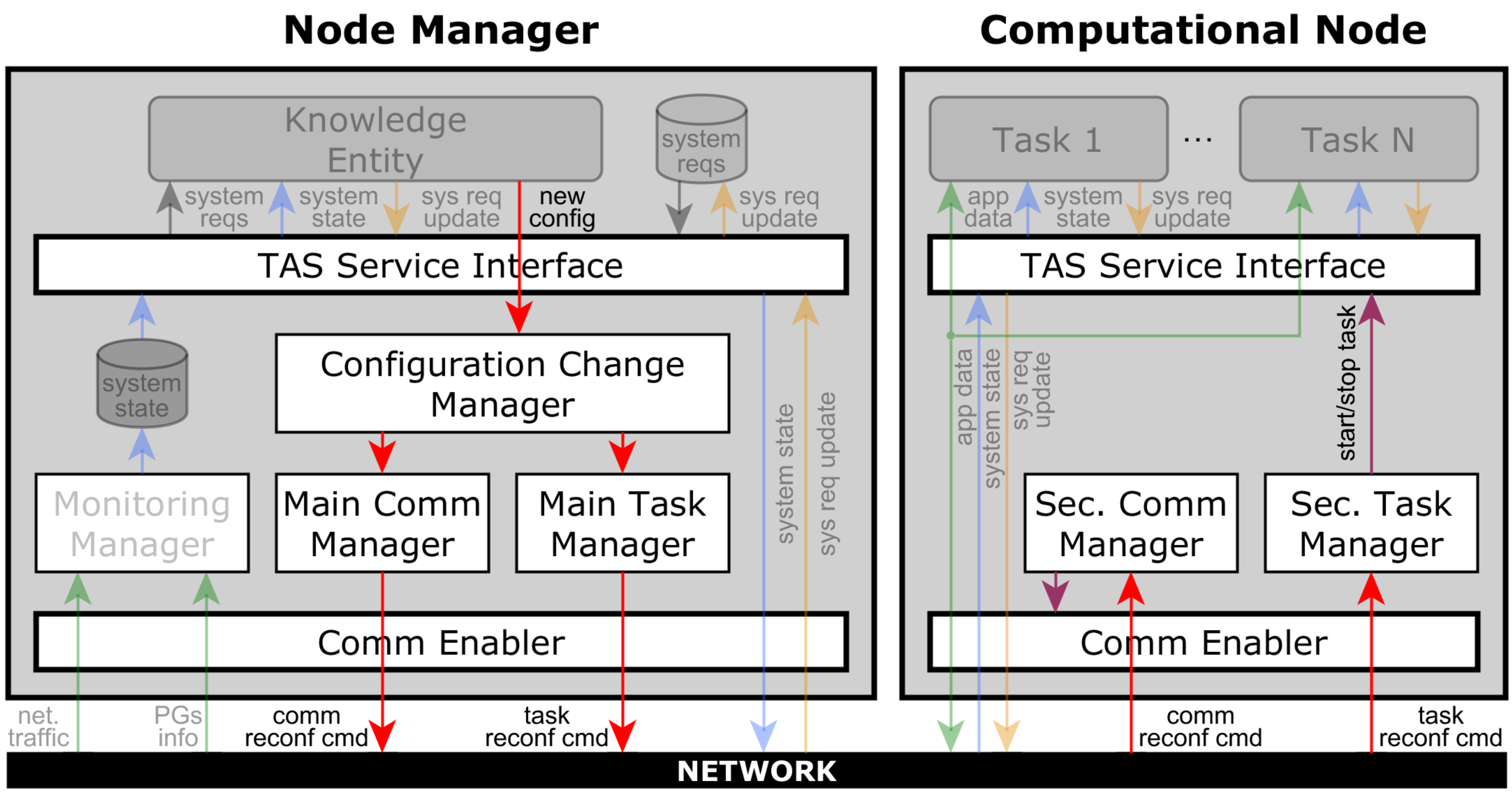
Validate non-functional requirements

- Check that the real-time and reliability requirements are met



Outline

The Self-Reconfiguration
Configuration Change Process



The Self-Reconfiguration
Configuration Change Process

Liberate the computational and communication **resources** of the **applications** that are **no longer required**

- Take into account the interdependencies
- Take into account the *termination condition*

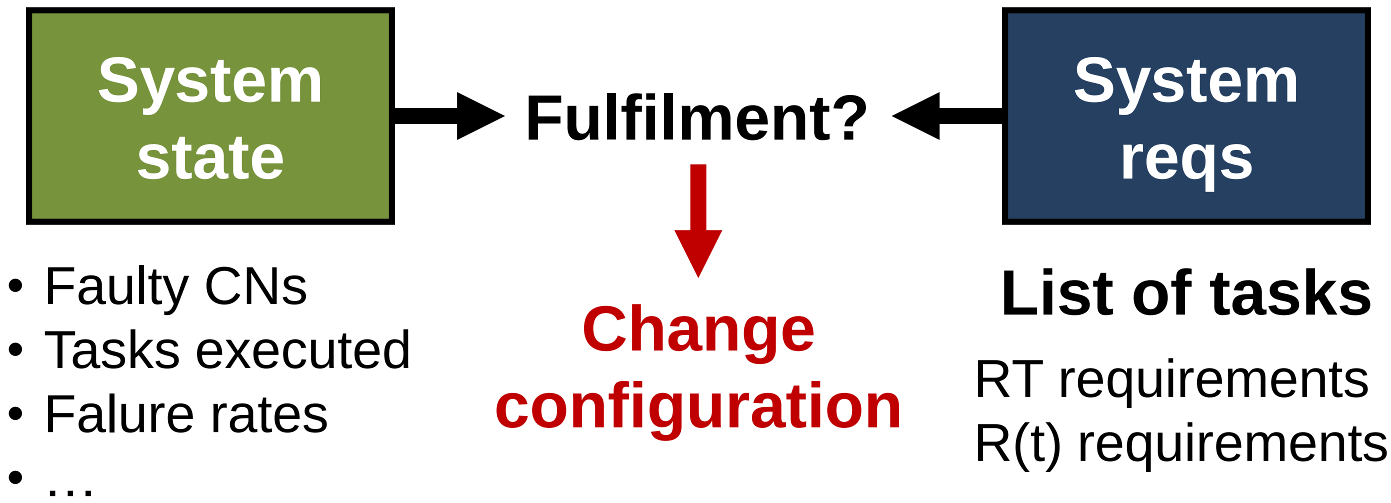
Reserve the computational and communication **resources** of the **new required applications**

Triggers the **execution** of the **tasks** and the **transmission** of **messages** in the **appropriate order**

Outline

Reconfiguration for Reliability

The **self-reconfiguration capabilities** of this infrastructure make it possible to **change the set of applications being executed** in the system, in response to **changes in the system state** or in the **system requirements**



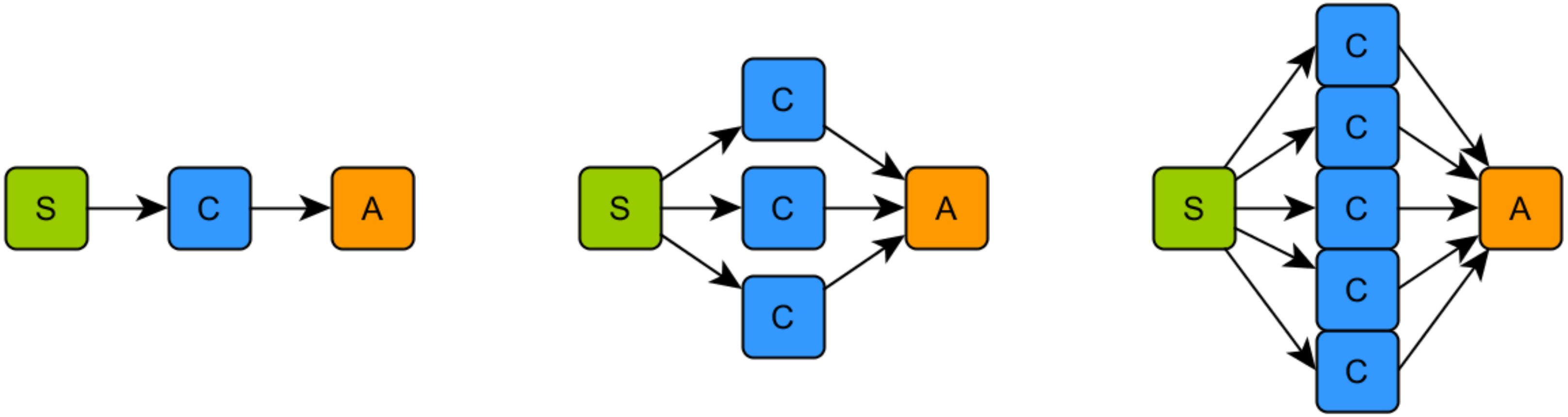
Reconfiguration for Reliability

Efficient use of the resources

Redundancy is a typical mechanism used to **tolerate faults**

- Is expensive
- Static redundancy suffers from redundancy attrition

The **level of task replication** is **managed automatically**



Reconfiguration for Reliability

Recovering of tasks

Reallocate the **tasks** being executed in one CN to another, when the first one suffers a **permanent failure**.

Non-critical tasks

- The service is restored after some downtime

Critical (replicated) tasks

- We have redundancy preservation
- Equivalent to N-Modular Redundancy scheme with spares

Outline

Conclusions

We described the **on-going work** we are carrying out to **construct** a **self-reconfigurable infrastructure** for systems with **real-time**, **reliability** and **adaptivity requirements**.

It allows to **dynamically modify** the **allocation of tasks in response to** a **changes** in the **system requirements** or in the **system state**.

- Real-time requirements
- Reliability requirements

This is **particularly interesting** for systems that use **redundancy**

- Efficient use of the resources
- Better recovering

On-going Work

- **Replicate** the **Node Manager**
- **Characterize** the **self-reconfiguration time**
  - Detect the need for reconfiguration
  - Determine a valid new configuration
  - Apply said configuration
- **Construct** a **prototype** to prove its feasibility
- **Evaluate** the **feasibility** of dynamically **changing the replication scheme**

Towards a Self-Reconfigurable Infrastructure for Critical Adaptive Distributed Embedded Systems

Alberto Ballesteros
Julián Proenza
Manuel Barranco
Luís Almeida
Pere Palmer

MINISTERIO DE ECONOMÍA Y COMPETITIVIDAD

FEDER
Fondo Europeo de Desarrollo Regional