NimbleNet

Serverless Computing for the Extreme Edge in Factory Environments

FAU | SIEMENS

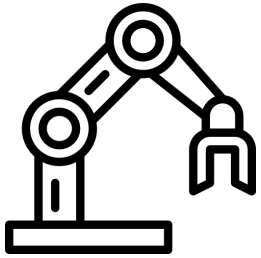
Siemens

FAU | SIEMENS

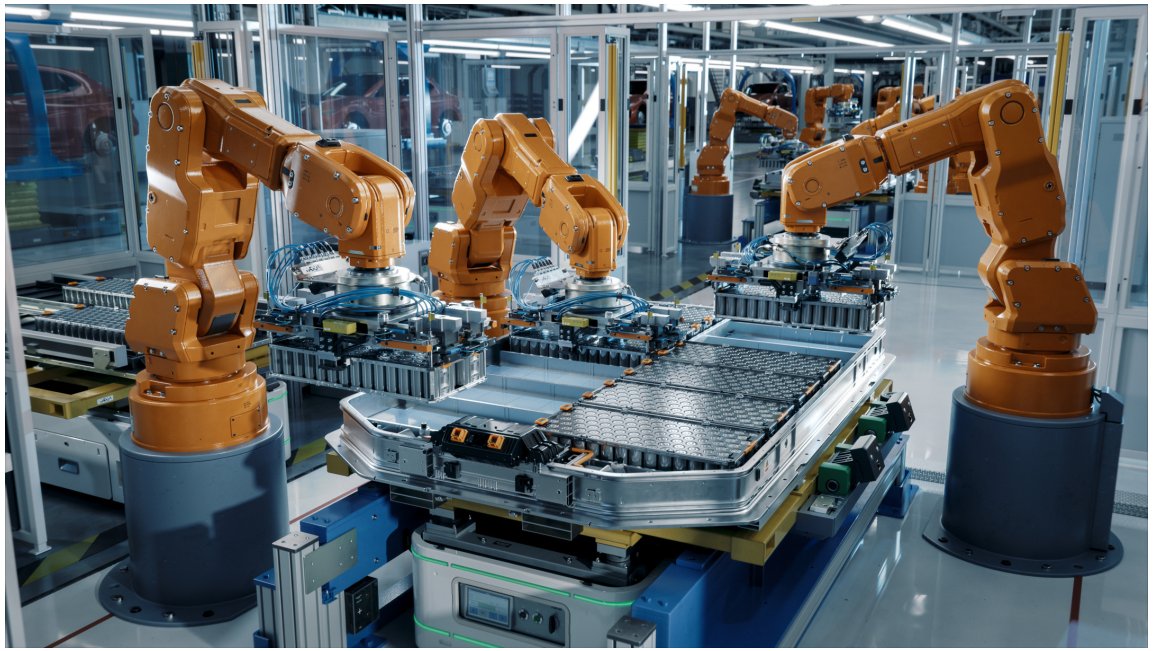
Siemens

FAU | SIEMENS

Siemens

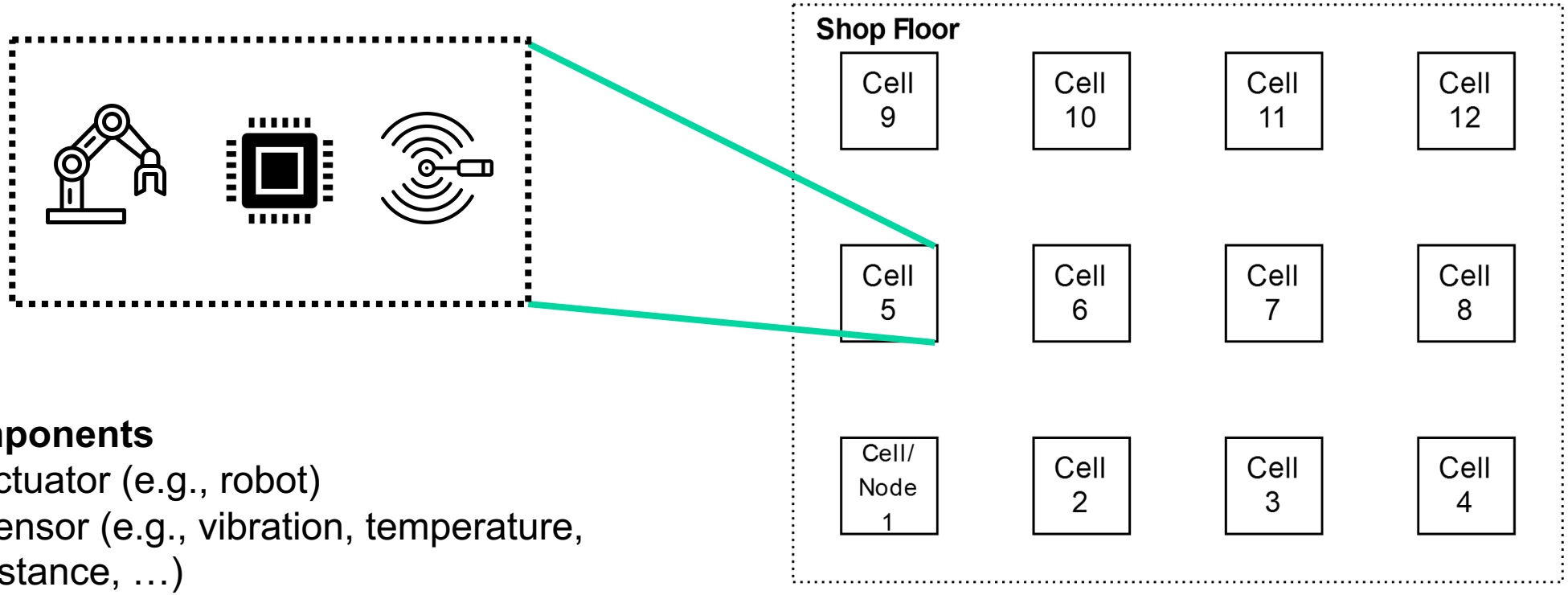


Increasing complexity



Line manufacturing → Modular manufacturing

Shop floor: Cell



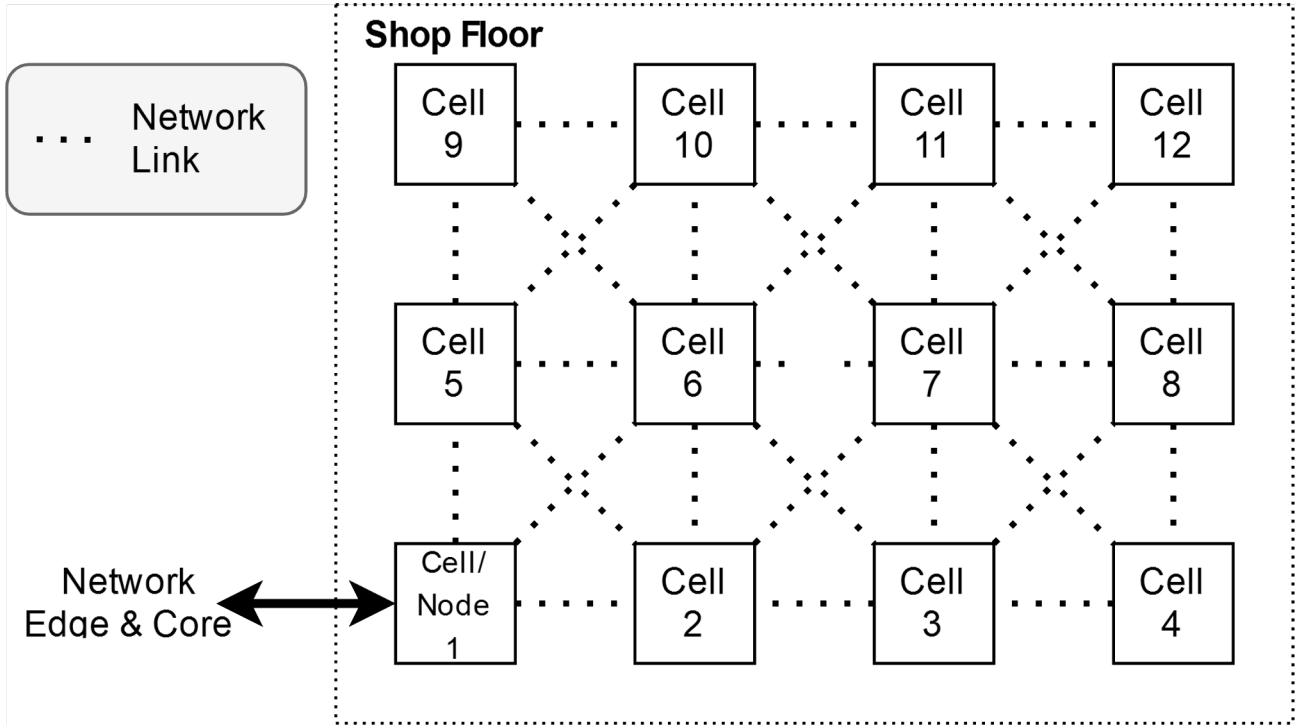
Components

- Actuator (e.g., robot)
- Sensor (e.g., vibration, temperature, distance, …)
- PLC: Programmable logic controller
- Connectivity

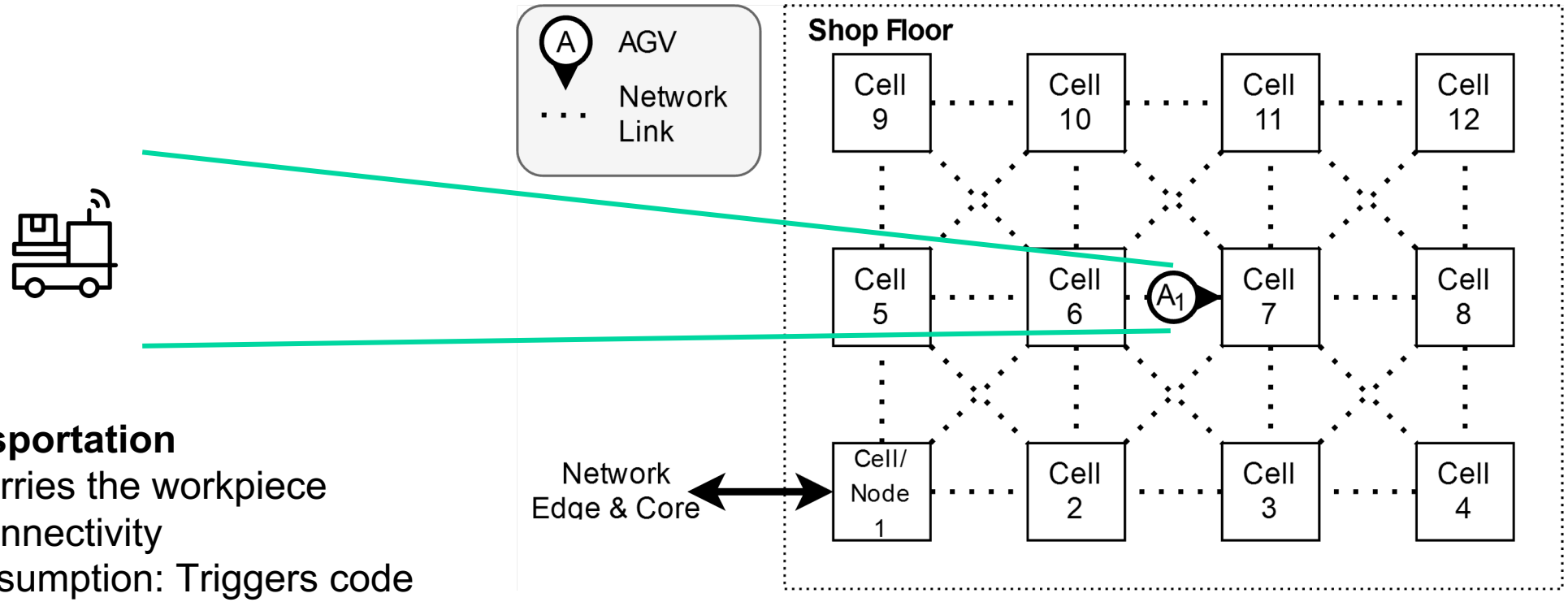
Shop floor: Connectivity

Connectivity

- Wireless mesh network



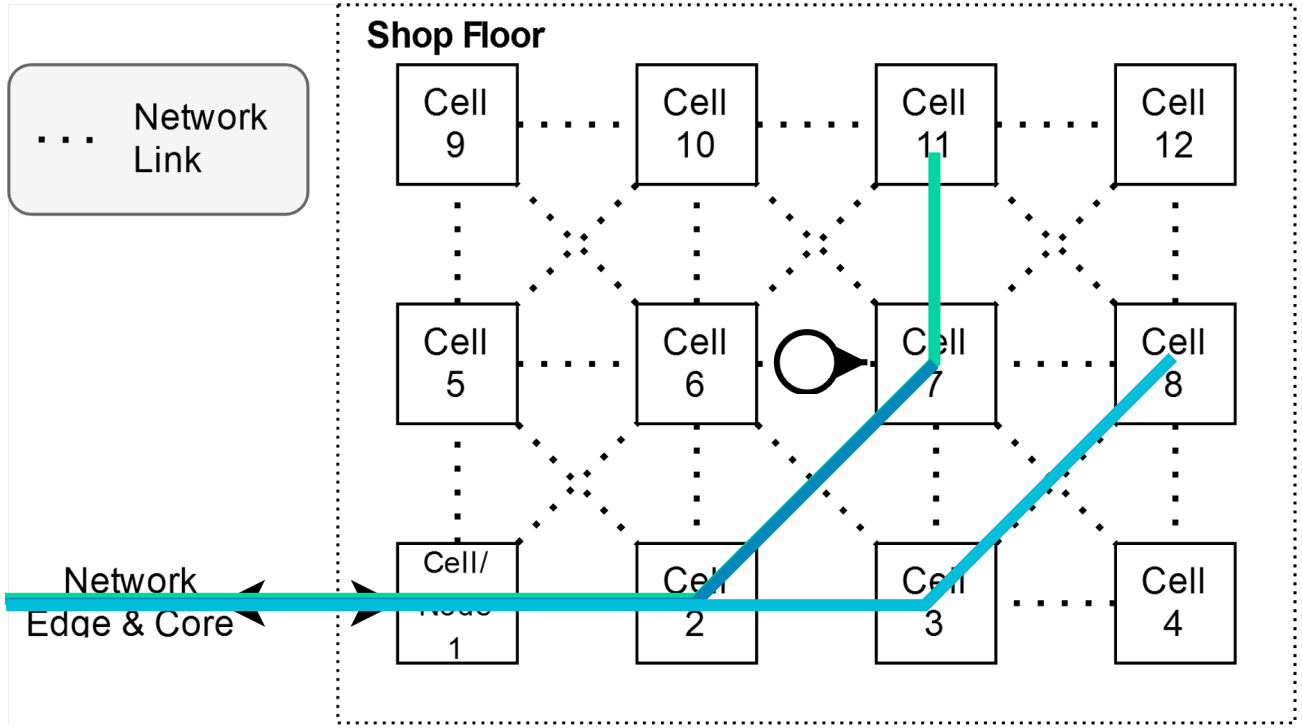
Shop floor: Transportation



Transportation

- Carries the workpiece
- Connectivity
- Assumption: Triggers code execution

Problem statement



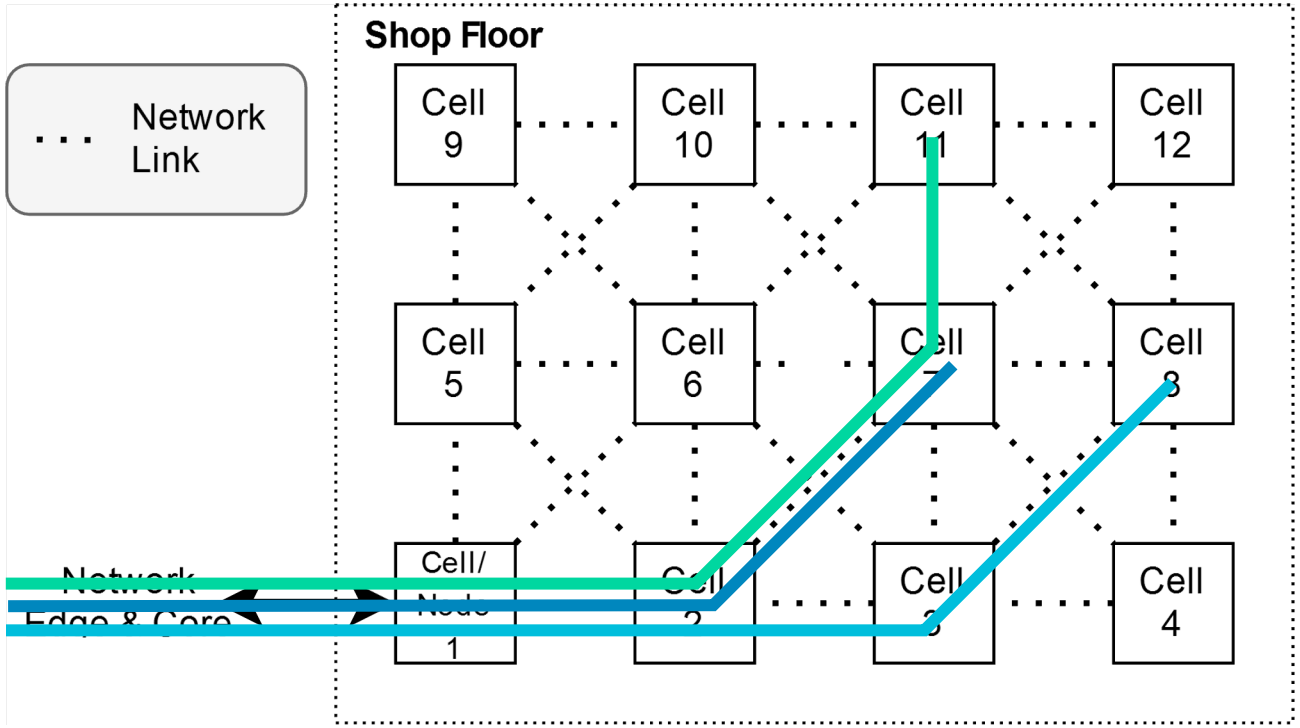
Problem statement

Problems

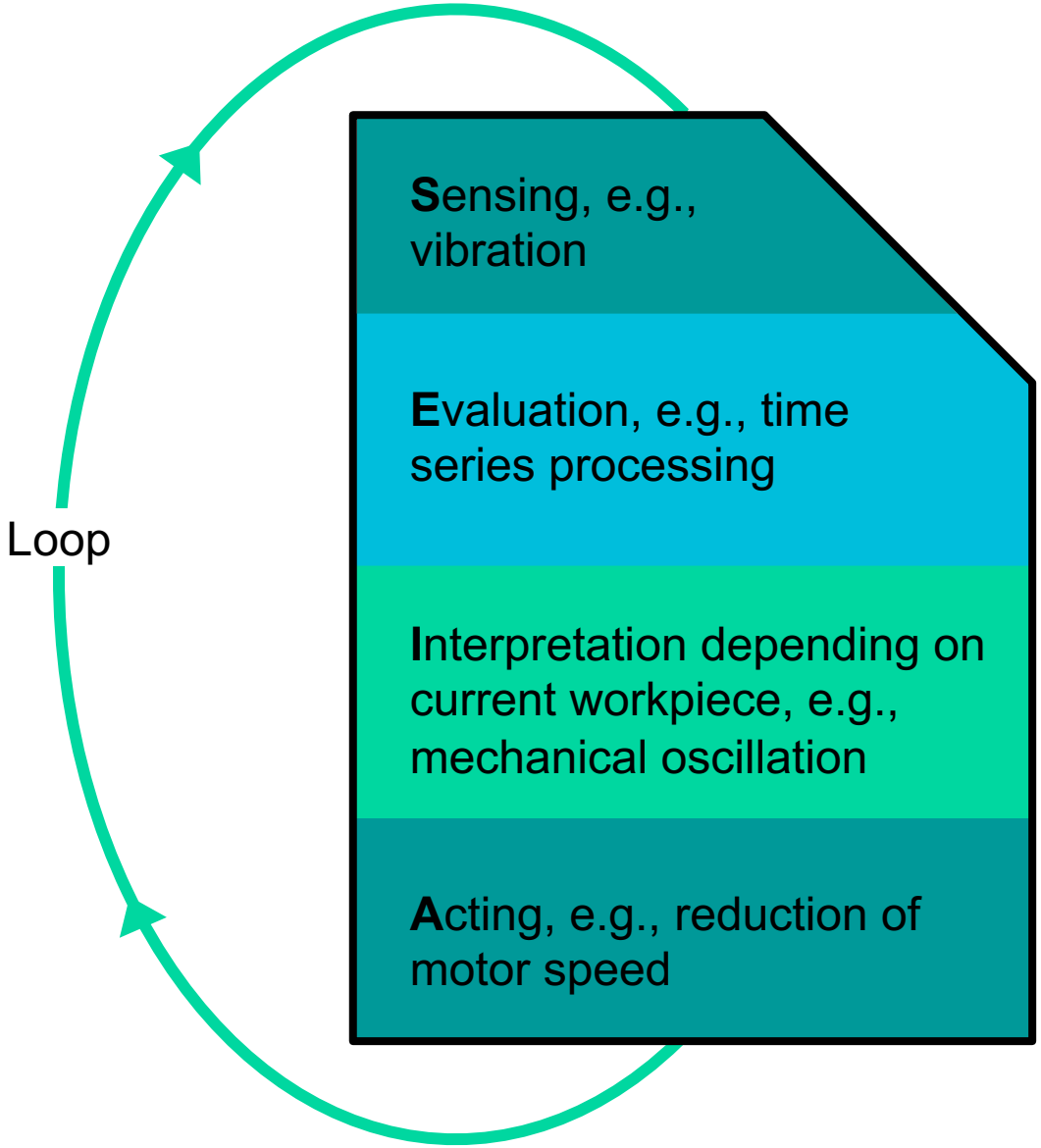
- Uneven load on nodes
- Long, redundant paths

Caching

… but how?



Caching domains



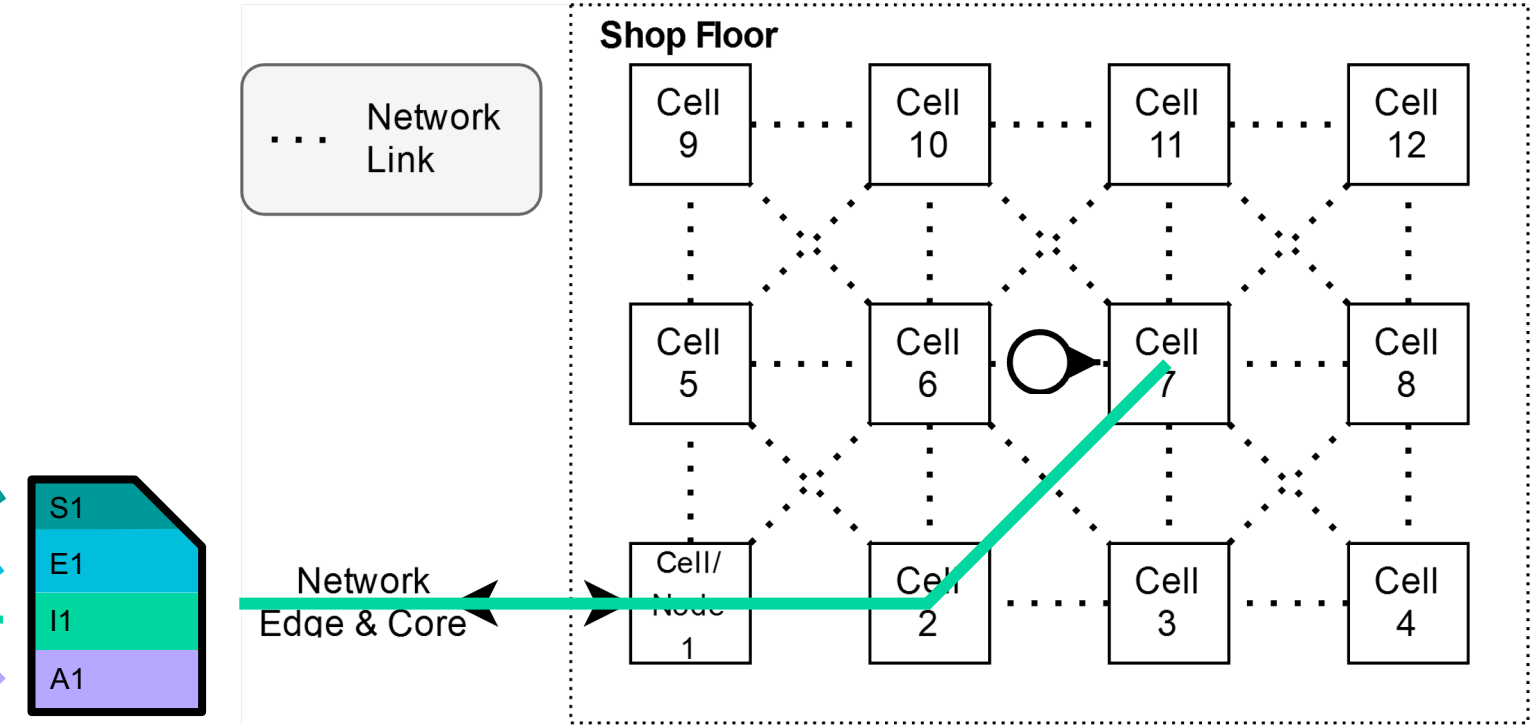
Caching domains

S1: Acceleration sensor

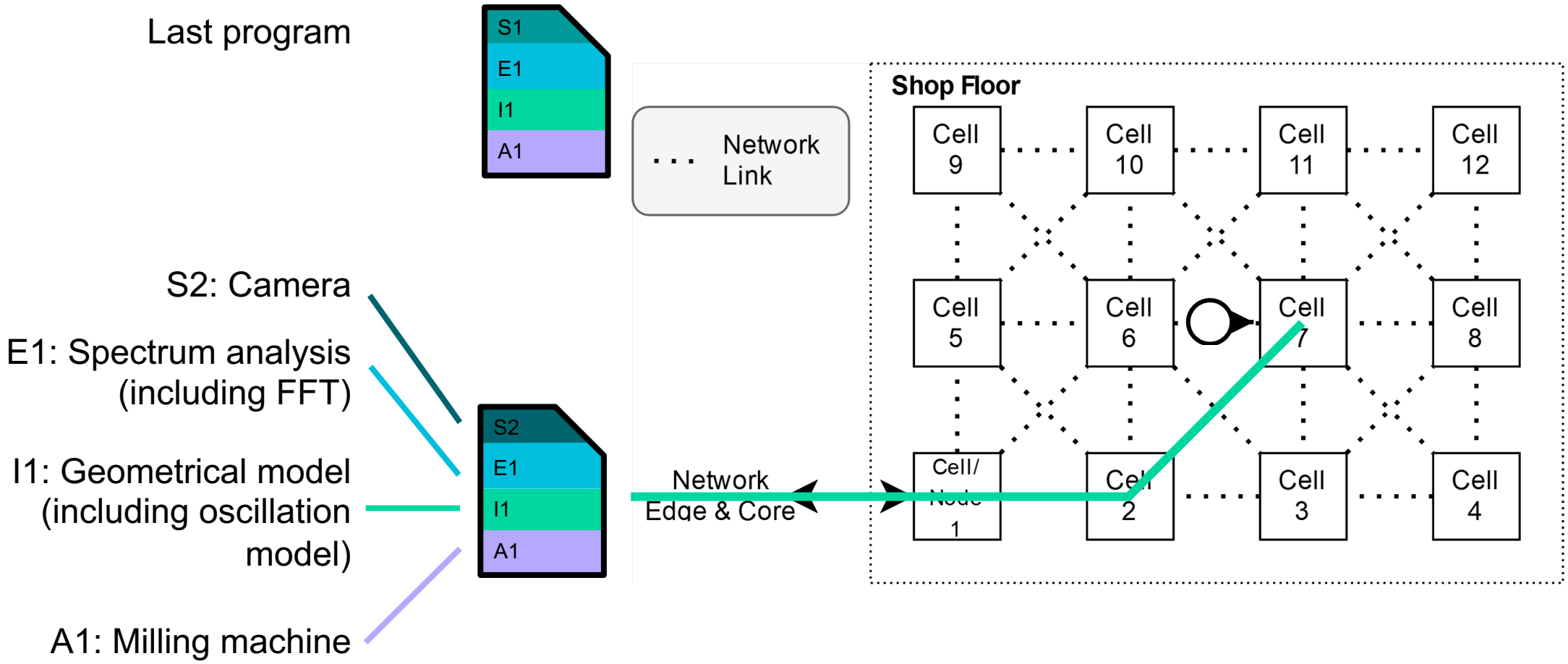
E1: Spectrum analysis (including FFT)

I1: Geometrical model (including oscillation model)

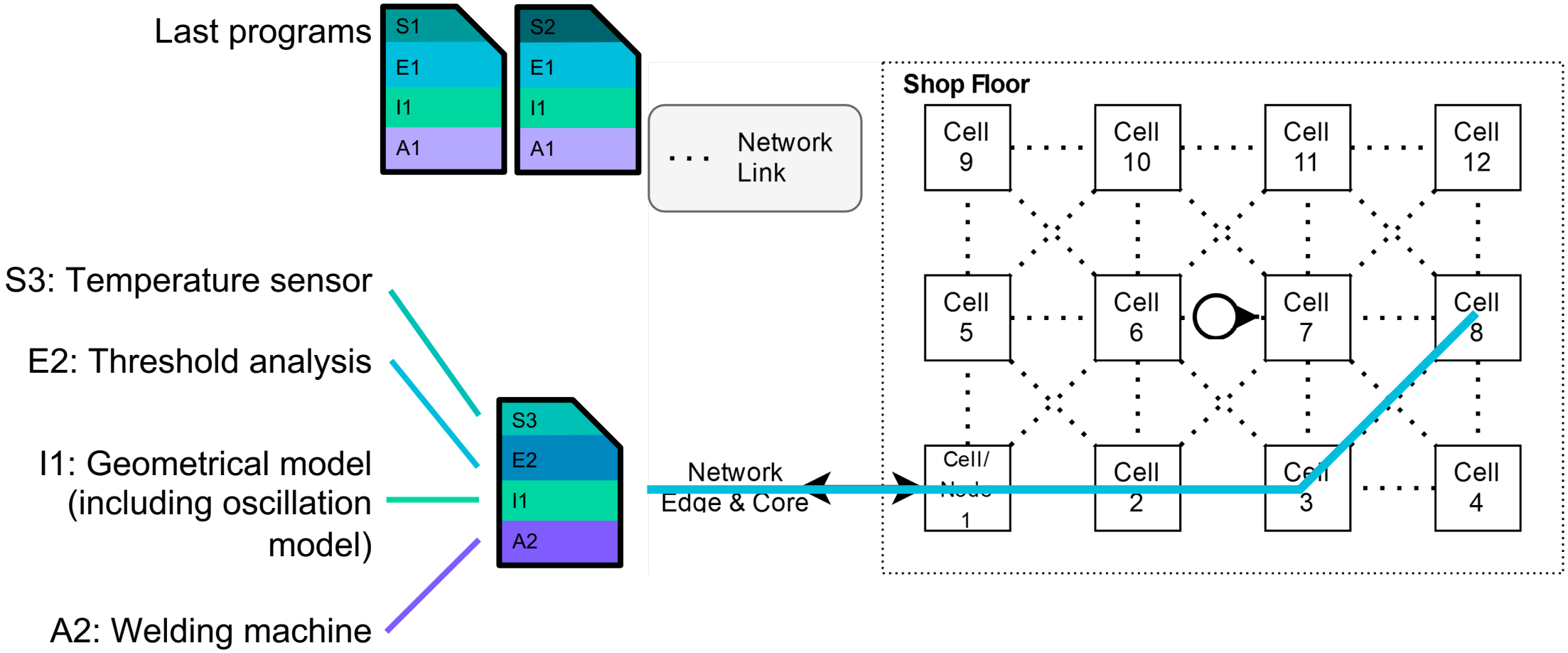
A1: Milling machine



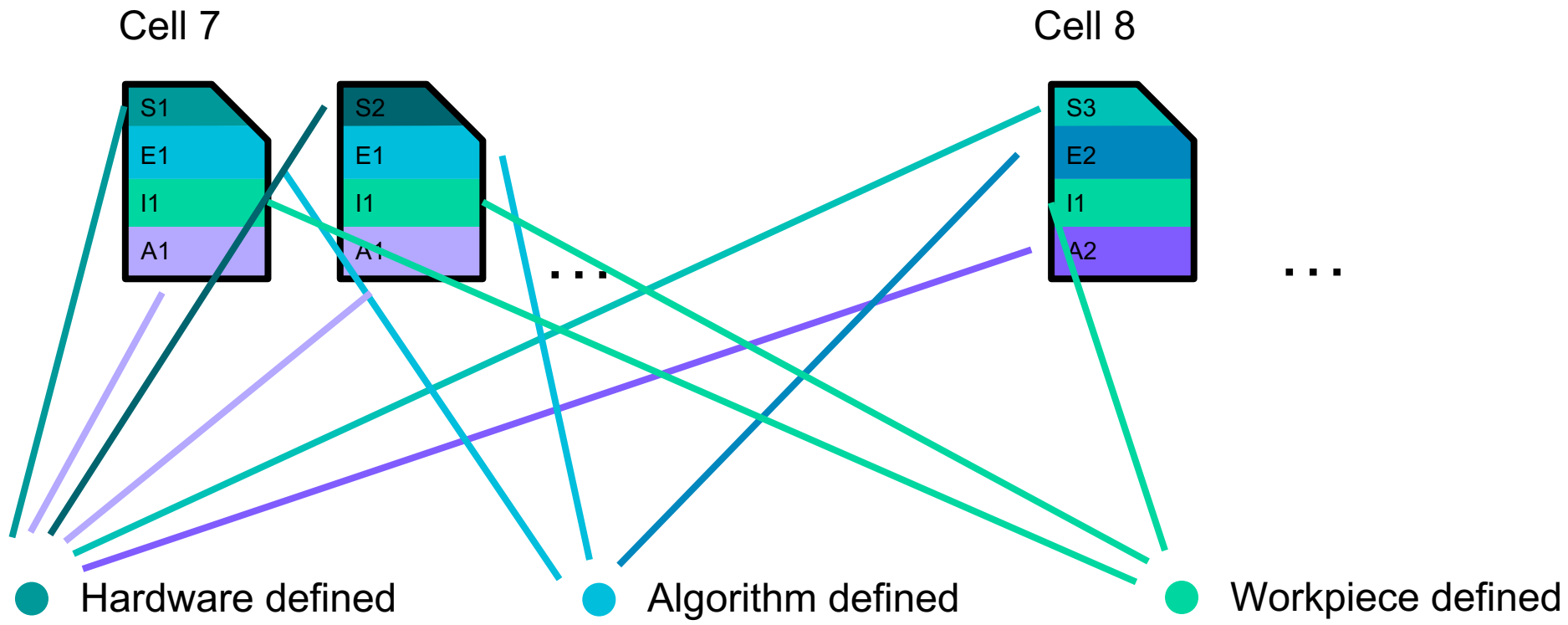
Caching domains



Caching domains



Caching domains



Caching domains

- Hardware defined
- Algorithm defined
- Workpiece defined

Related to node

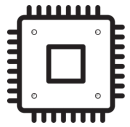
Related to cell

Related to AGV

Goal

Split programs into reusable parts

Employ local caching strategies



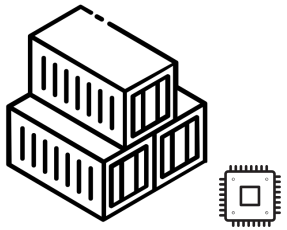
Be hardware independent

Enable simple compilation

Be transparent for the application

FAU | SIEMENS

Some Related Work

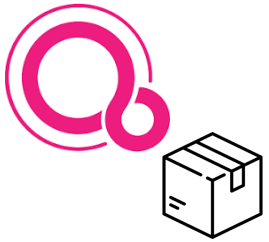


Micro-Containerization

- Platform specific features
- Often architecture specific

(Micro)Docker

- Does not run on constrained devices (IoT sensors)
- Large overhead



Fuchsia, APT, YUM, TUF…

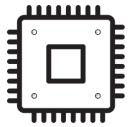
- Does not run on constrained devices (IoT sensors)
- Sophisticated dependency resolving causes overhead

Approach

Goal

Split programs into reusable parts

Employ local caching strategies



Be hardware independent

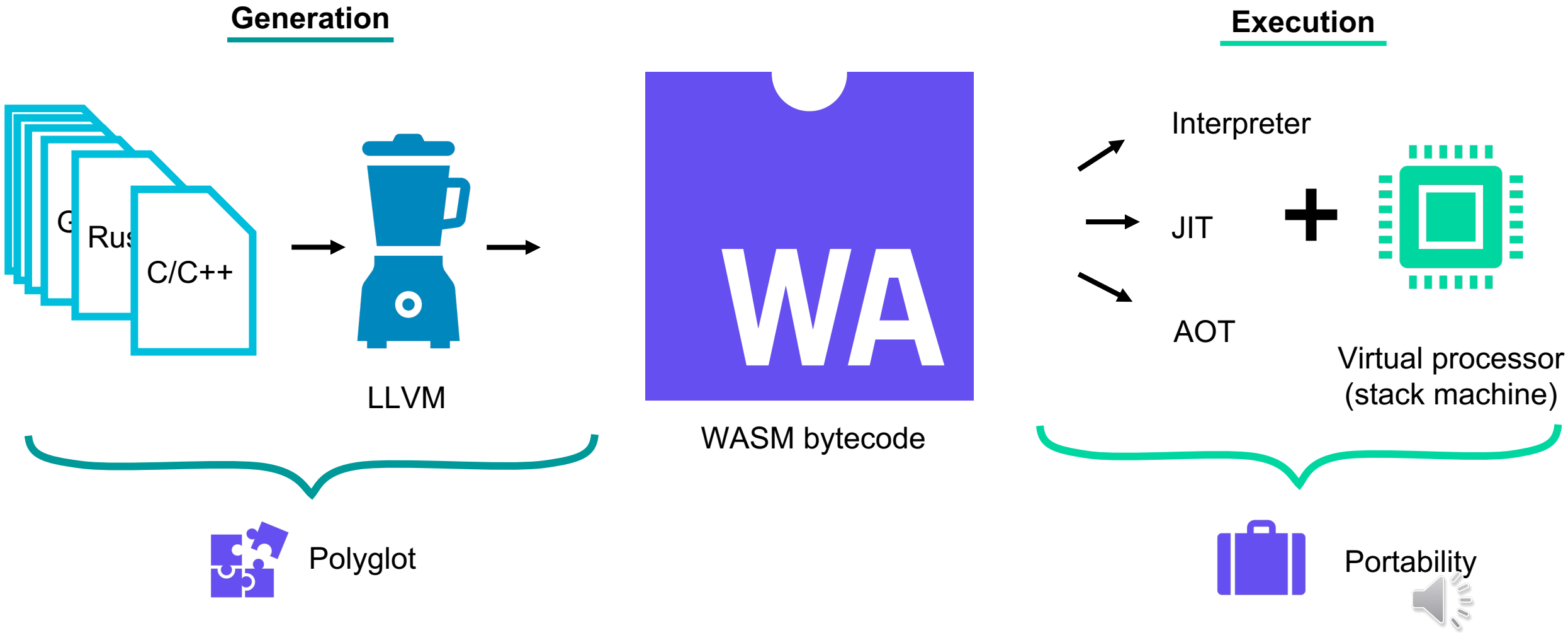
Enable simple compilation

Be transparent for the application

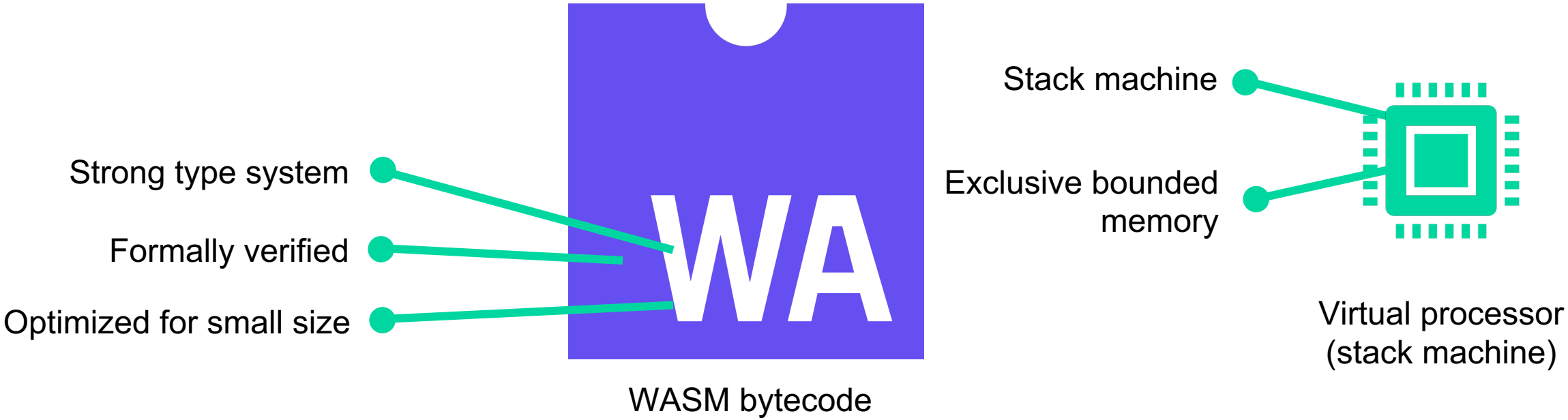
FAU | SIEMENS

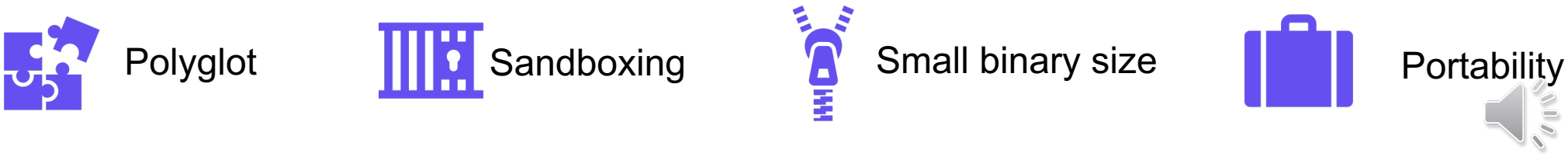
What is WASM?

What is WASM?



What is WASM?


Strong type system
Formally verified
Optimized for small size
WA
WASM bytecode
Stack machine
Exclusive bounded memory
Virtual processor (stack machine)


Polyglot
Sandboxing
Small binary size
Portability

What is WASM?

Features

- Polyglot
- Portability
- Sandboxing
- Small binary size

Preemptable WASM

Split programs → Preemptive execution

```
(module
    (import "serverless" "function"
        (func $func))

    (func $execute
        ;; ...
        call $func
        ;; ...
    )
)
```

Host function definition

Host function call

Preemptable WASM

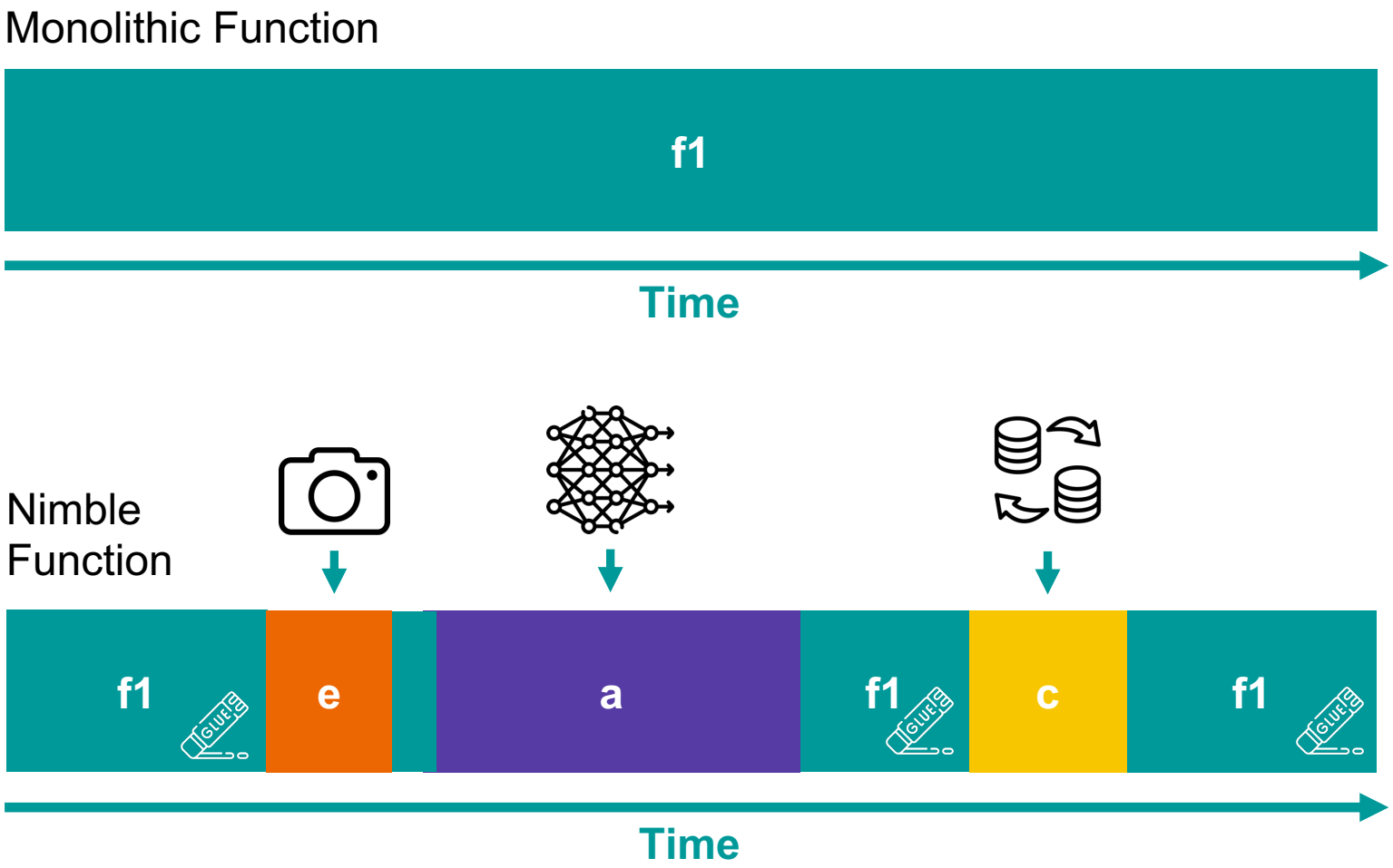
Split programs ➡ Preemptive execution

```
(module
    (import "serverless" "function"
        (func $func))

    (func $execute
        ;; ...
        call $func
        ;; ...
    )
)
```

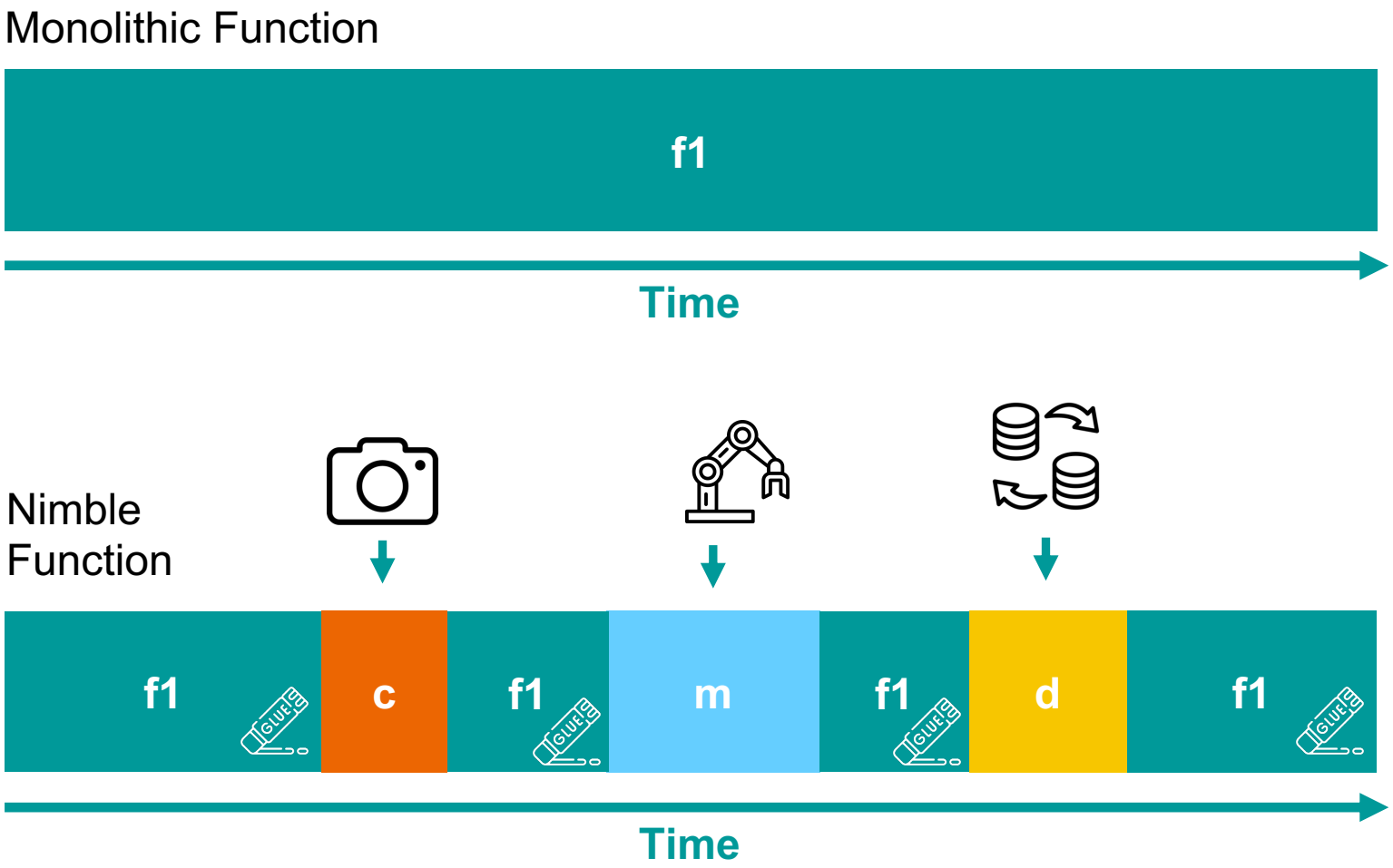
Host function call

1. Program state (➡ Constrained IoT nodes):
   - Operand stack + linear memory + global variables
   - “Program counter”
2. Trigger error
3. Execute host function
4. Restore before and recall

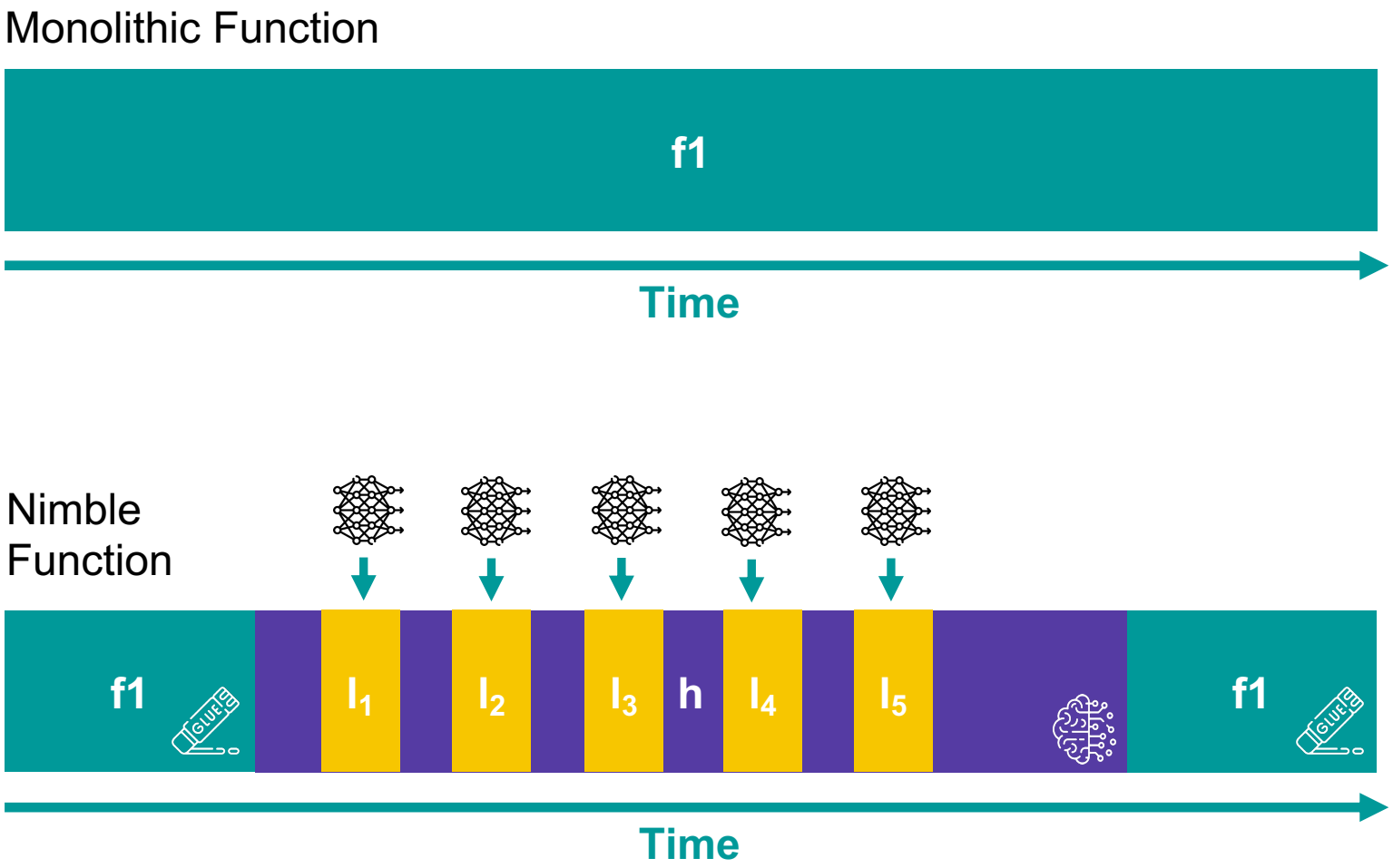
Function Overview



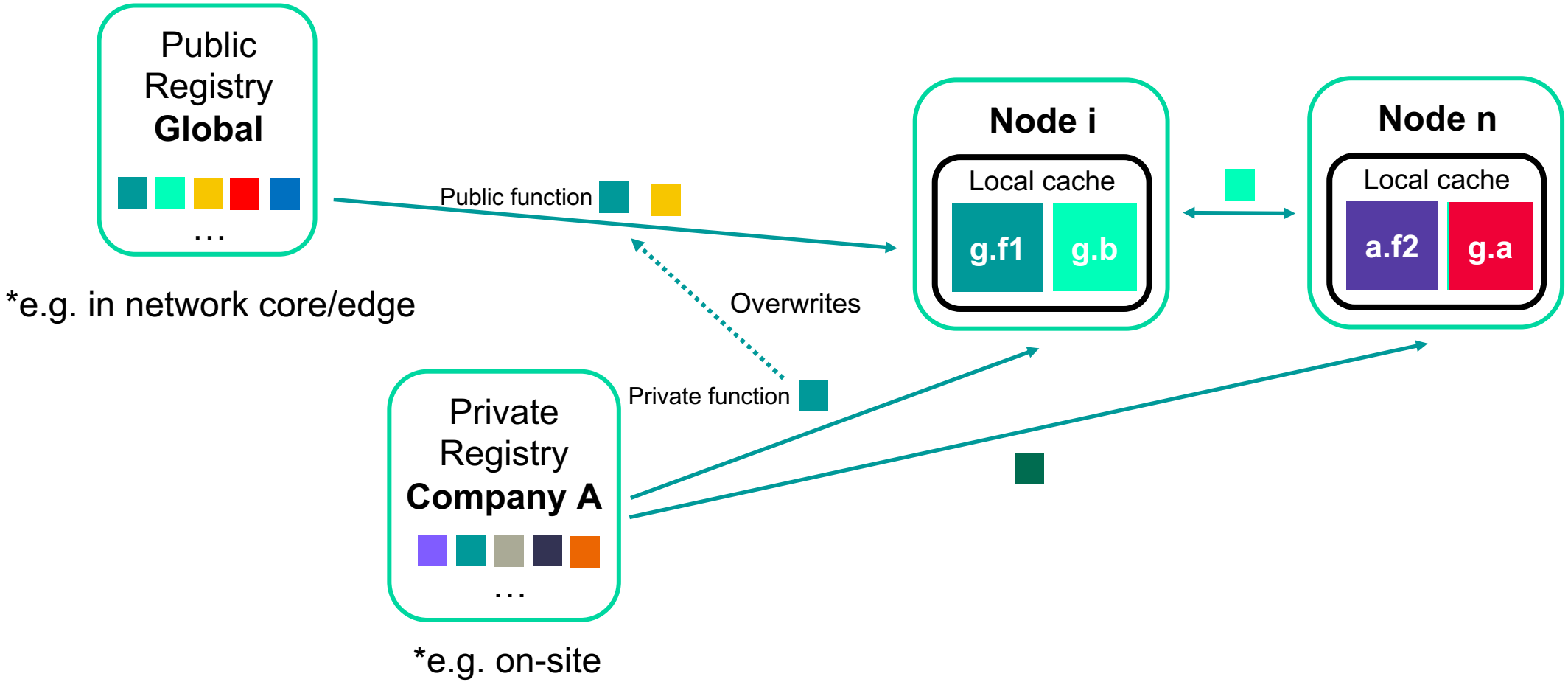
Function Overview



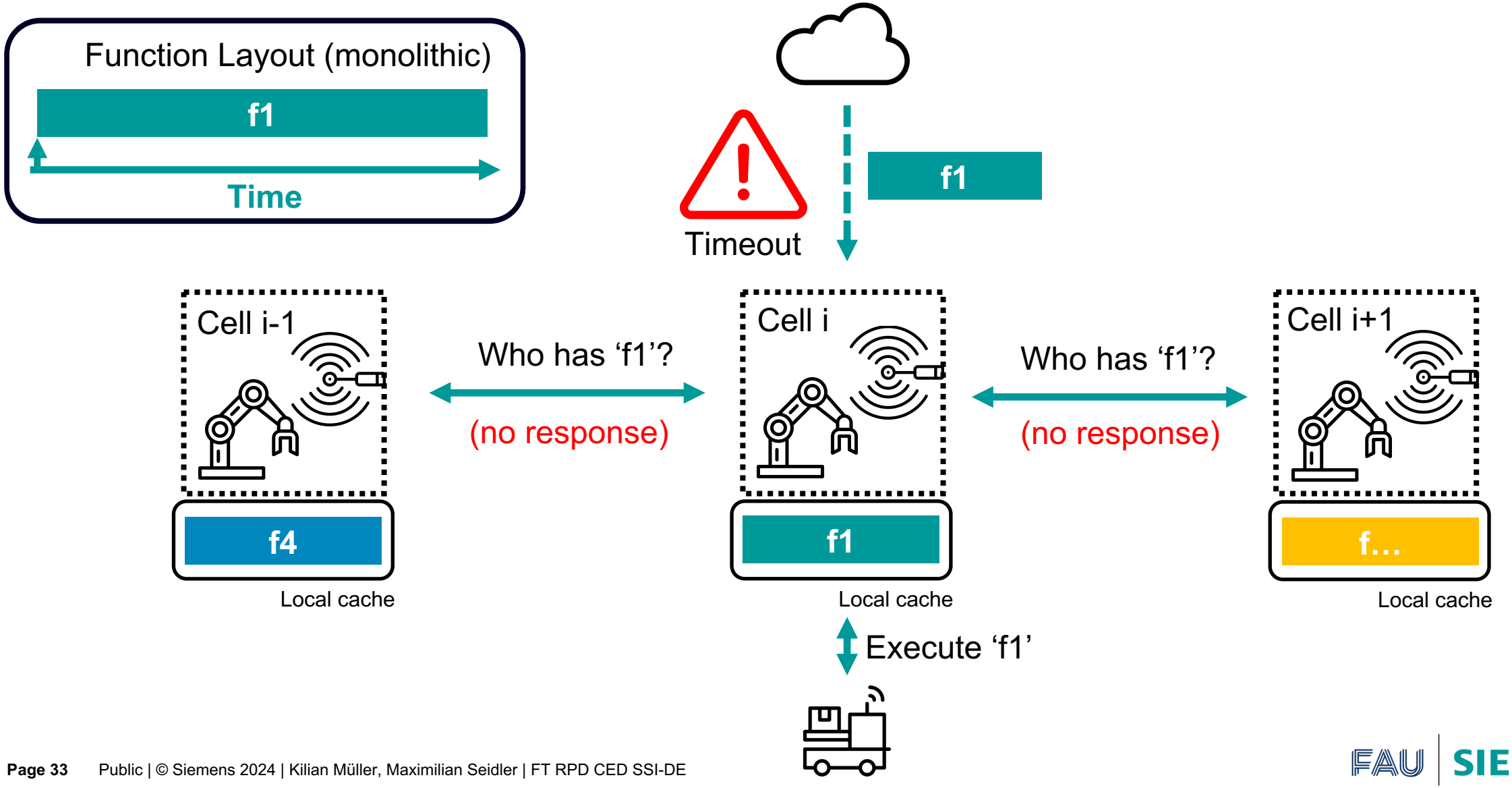
Hierarchical Function Calling



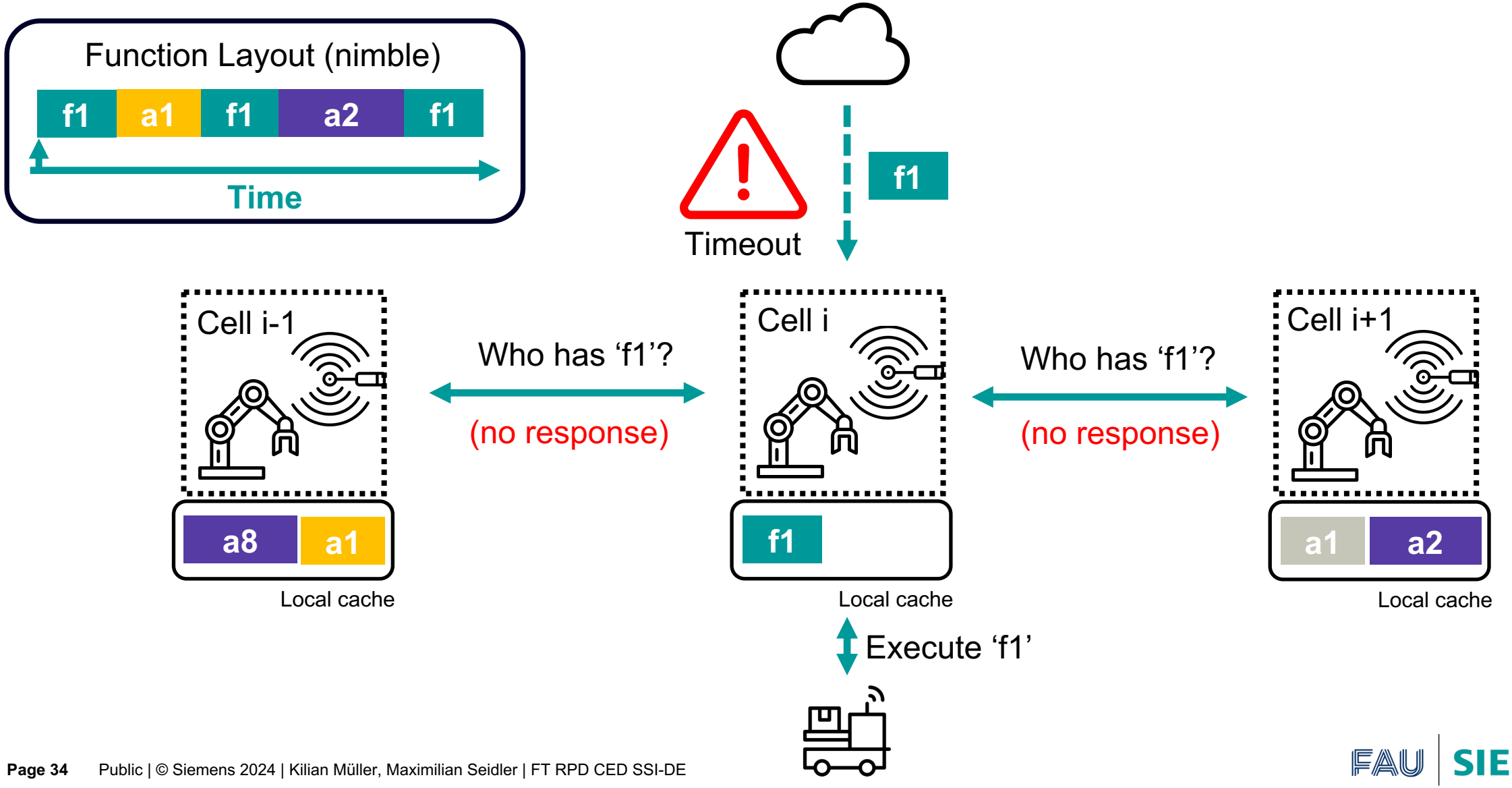
Function Stores & Local Caches



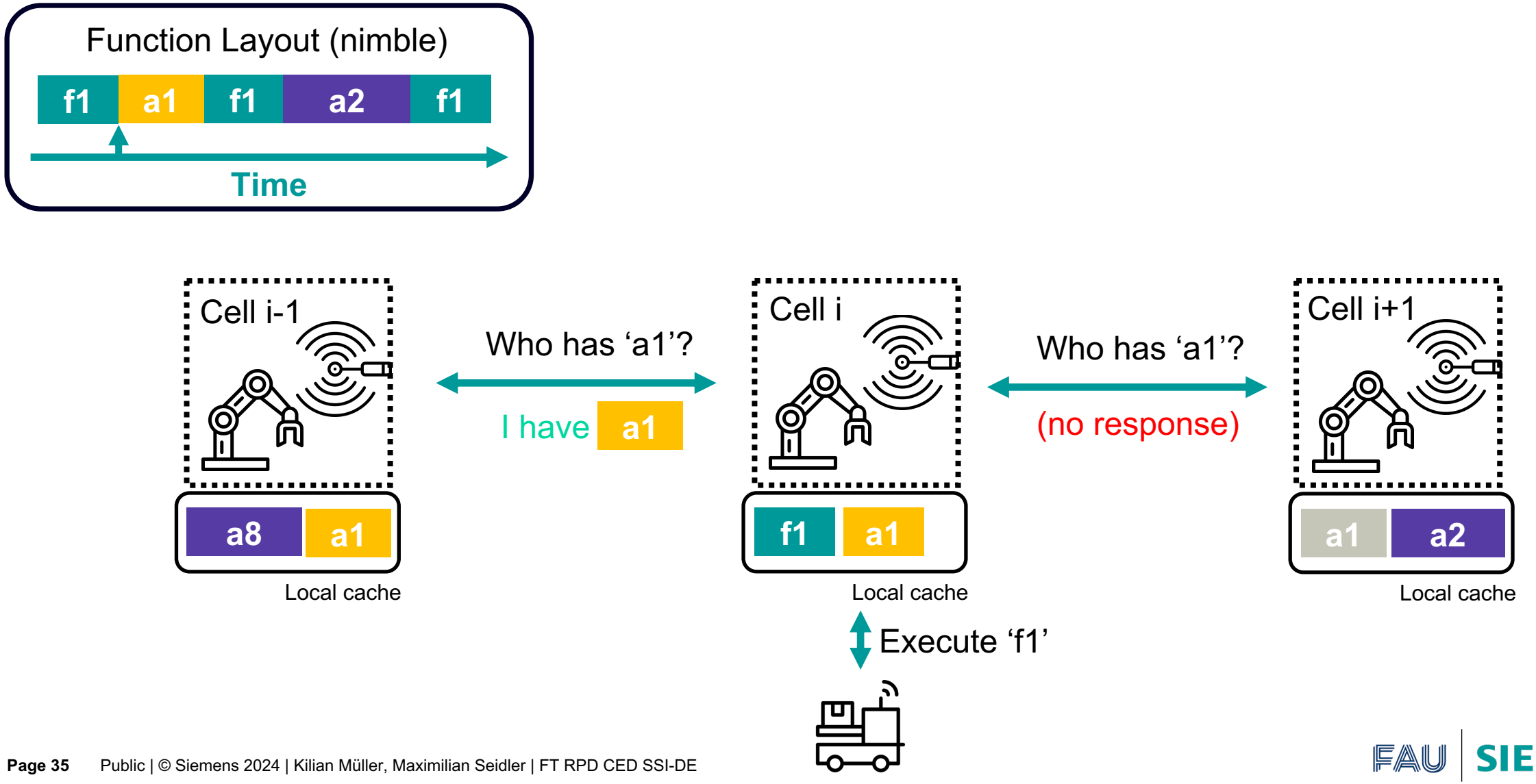
Examplary Function Acquisiton (Monolithic)



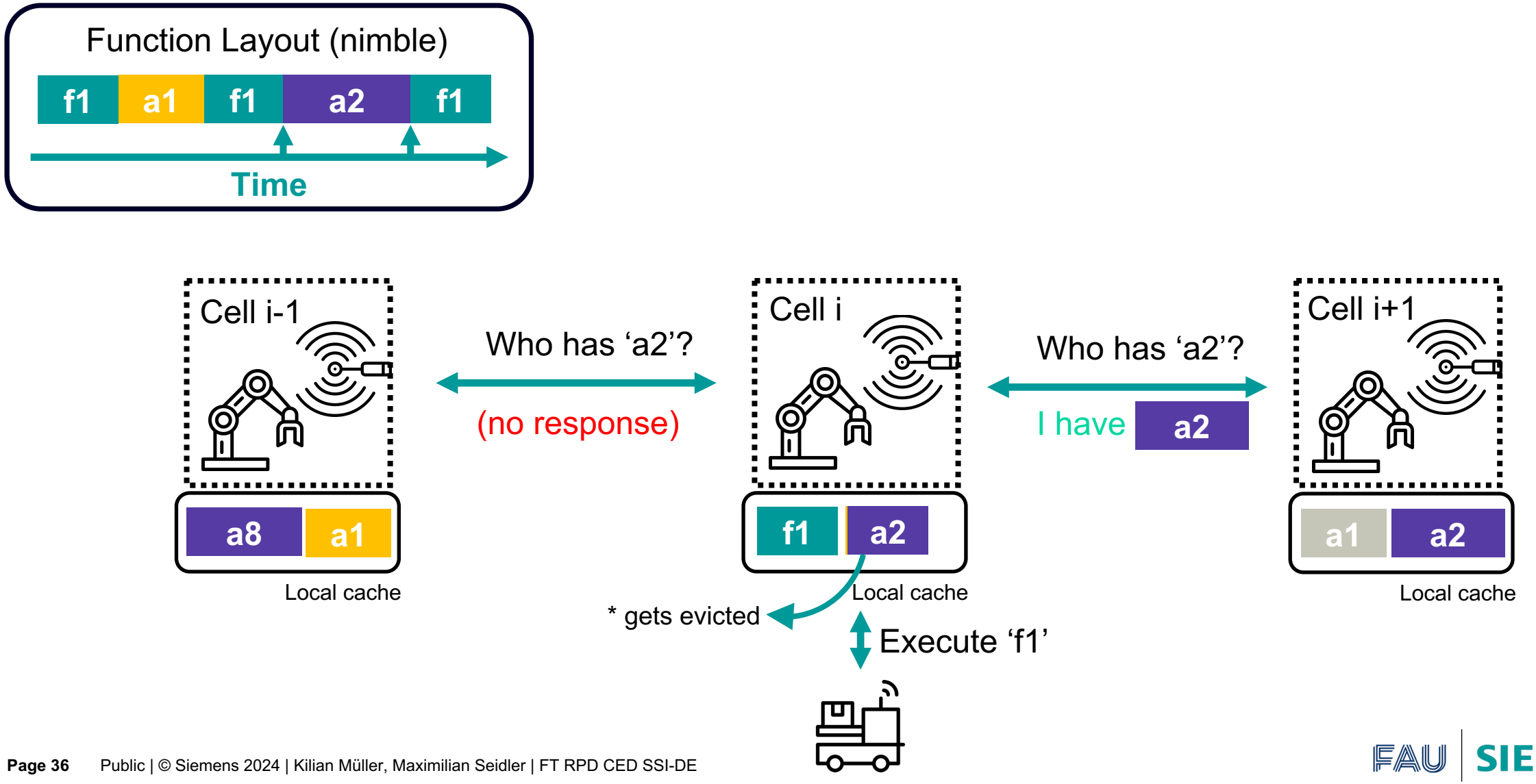
Exemplary Function Acquisiton (NimbleNet)



Exemplary Function Acquisiton (NimbleNet)



Exemplary Function Acquisiton (NimbleNet)



Function Example (Micro)Python

siemens.temp_alert.1.0.1 **+ WASM**

```
(module
  (import "bosch" "bmp280" (func $bosch_bmp280 (result i32)))
  (import "rpirp2" "led_blink" (func $led_blink (param i32)))

  (func $execute

    call $bosch_bmp280
    i32.const 21
    i32.lt_s

    if
      i32.const 5
      call $led_blink
    else
      i32.const 1
      call $led_blink
    end
  )

  (export "execute" (func $execute))
)
```

* get_dependencies() only required for preloading

bosch.bmp280

```
import nimblenet
from machine import I2C, Pin
import bmp280

# BMP280 tasklet

def setup():

    # Initialize the I2C interface
    i2c = I2C(0, scl=Pin(9), sda=Pin(8))
    # Initialize the BMP280 sensor
    sensor = bmp280.BMP280(i2c)

def execute()

    # Read and print temperature and pressure
    temperature = sensor.temperature
    pressure = sensor.pressure
    # Return temperature and pressure
    return temperature, pressure


def get_dependencies():
    return []
```

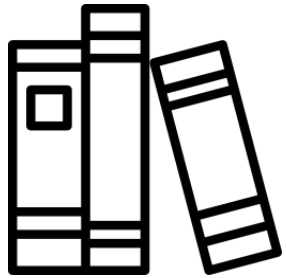
rpirp2.led_blink

```
import nimblenet
from machine import Pin
import time

#Blink tasklet

def setup():

    led = Pin(25, Pin.OUT)

def execute(period: float)
    # Turn the LED on
    led.on()
    # Keep the LED on for 0.5 seconds
    time.sleep(period/2)
    # Turn the LED off
    led.off()
    # Keep the LED off for 0.5 seconds
    time.sleep(period/2)

def get_dependencies():
    return []
```

*can also be a remote call

Supported platforms & languages

Supported Hardware



Native C/C++ Library

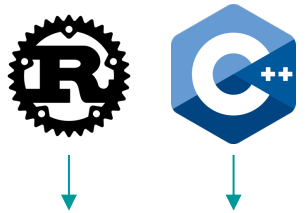
Embedded devices
e.g. ARM Cortex M0+

docker

Docker Container

Everywhere else ….

Supported Languages



… and many more

WebAssembly

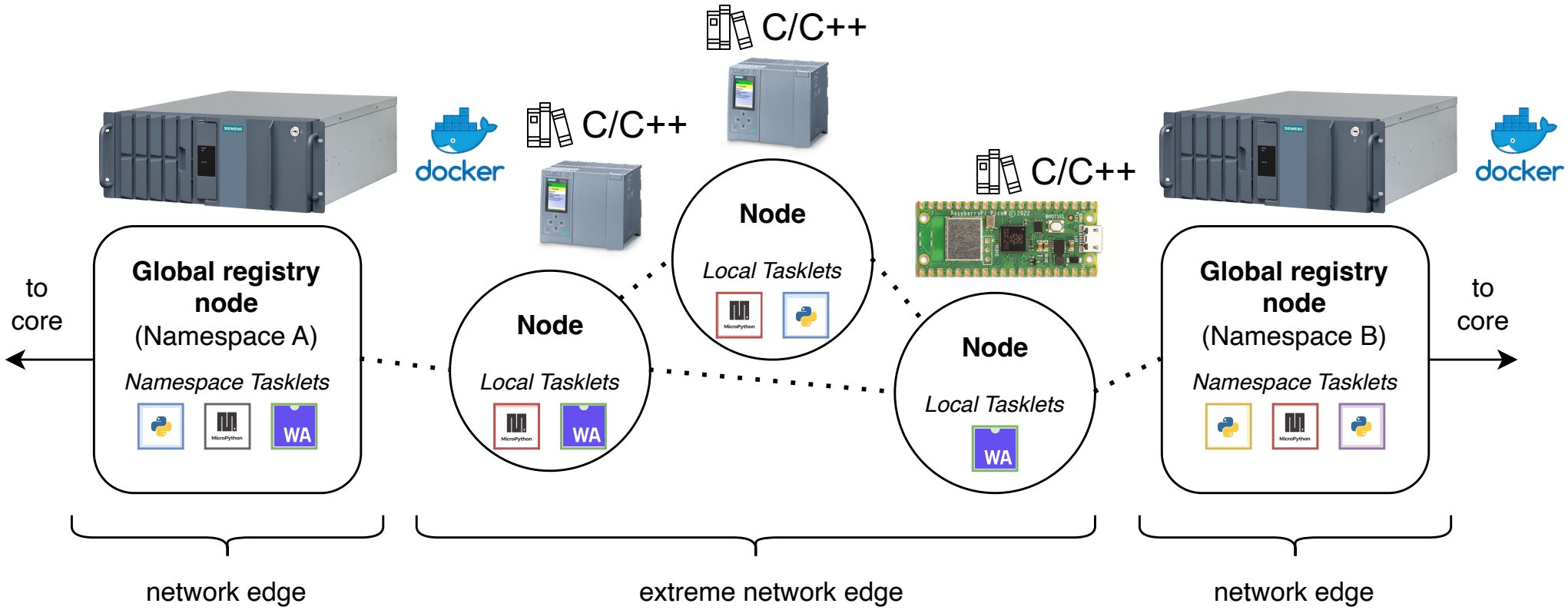
*MicroPython

*Python

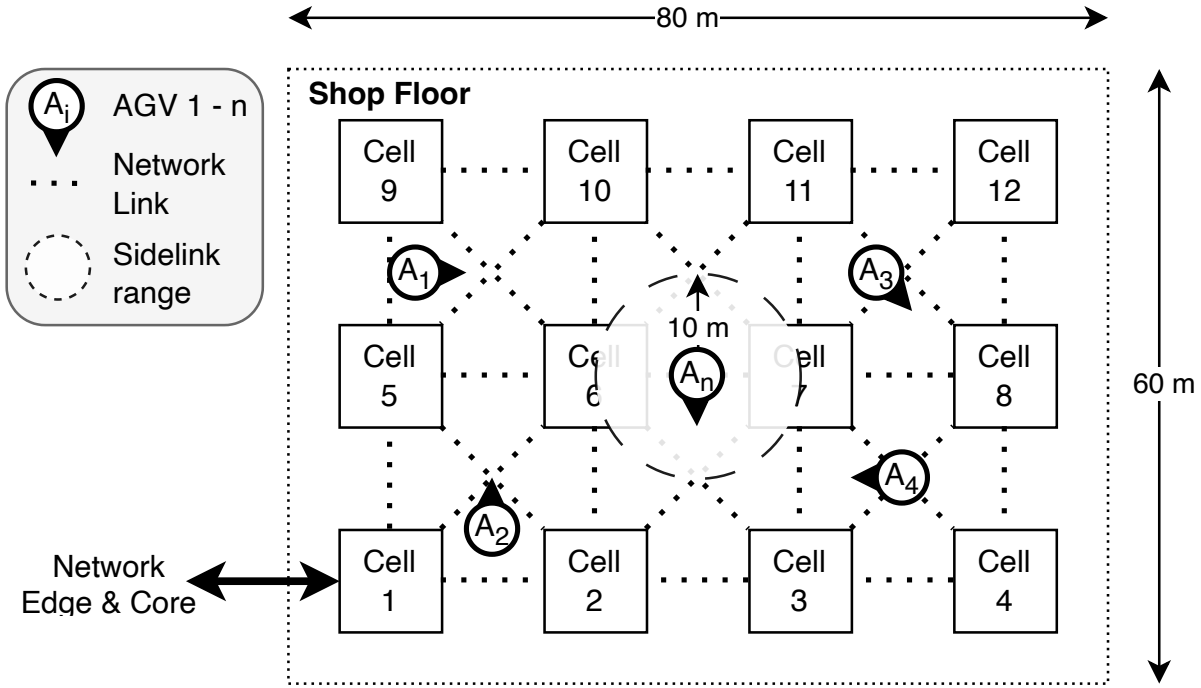
*not supported on all device classes

FAU | SIEMENS

Implementation Overview

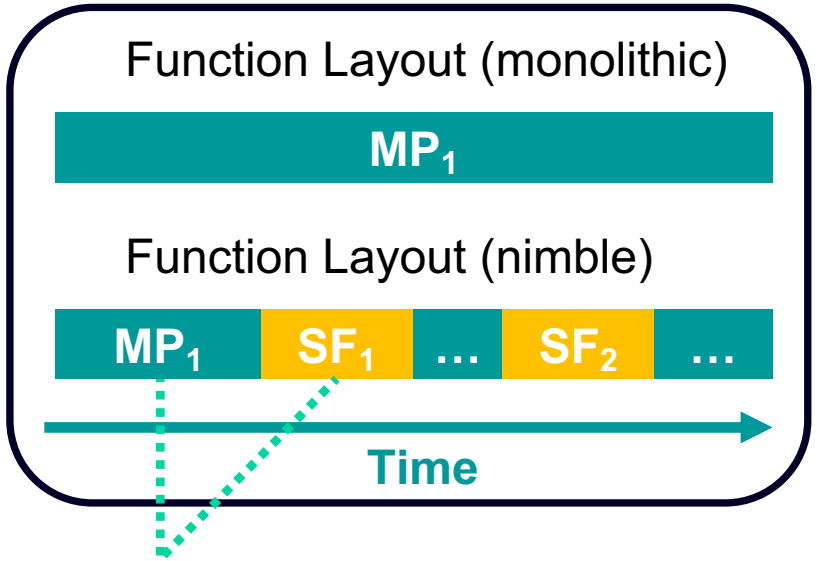


Simulation Setup



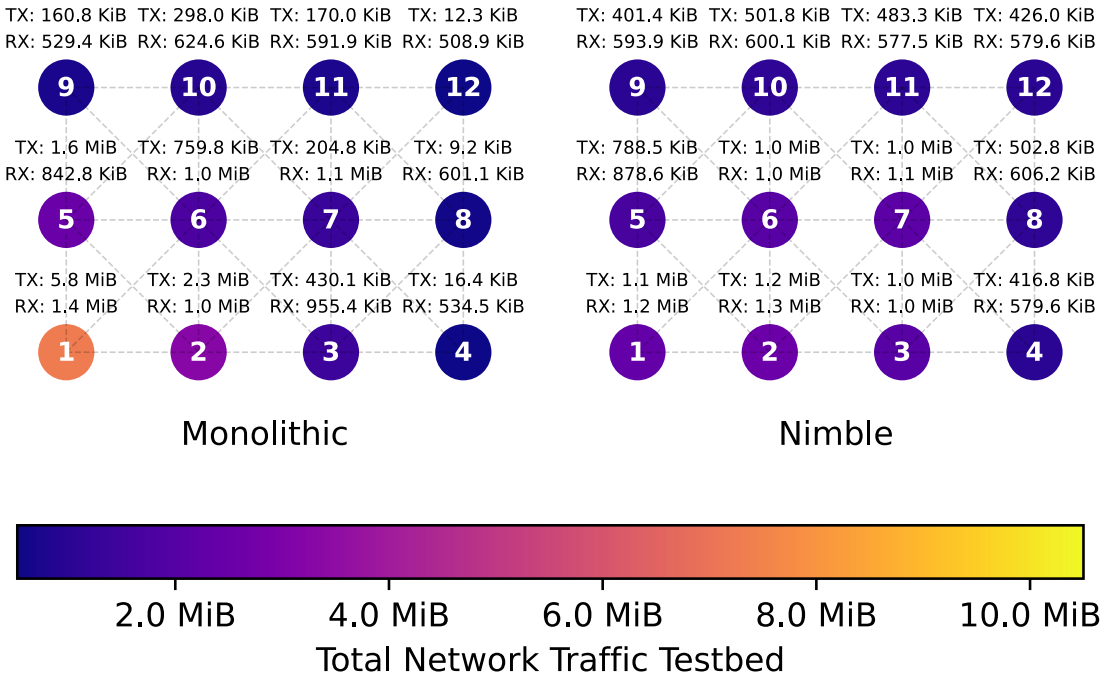
- 20 AGVs (Speed: 0 – 1 m/s)
- Random movement between cells
- Local Cache per Cell: 32 kB
- Schedule random MP every 9 – 11s

*MP: manufacturing process (step)
*SF: serverless function



- 20 MPs per AGV
- # SFs per MP: 3 - 7
- Unique MPs (SFs): 200 (50)
- MP & SF (bytes): 100 – 500 bytes
- MP (SF) response time: 0.01 – 2s (0.1 – 0.5s)

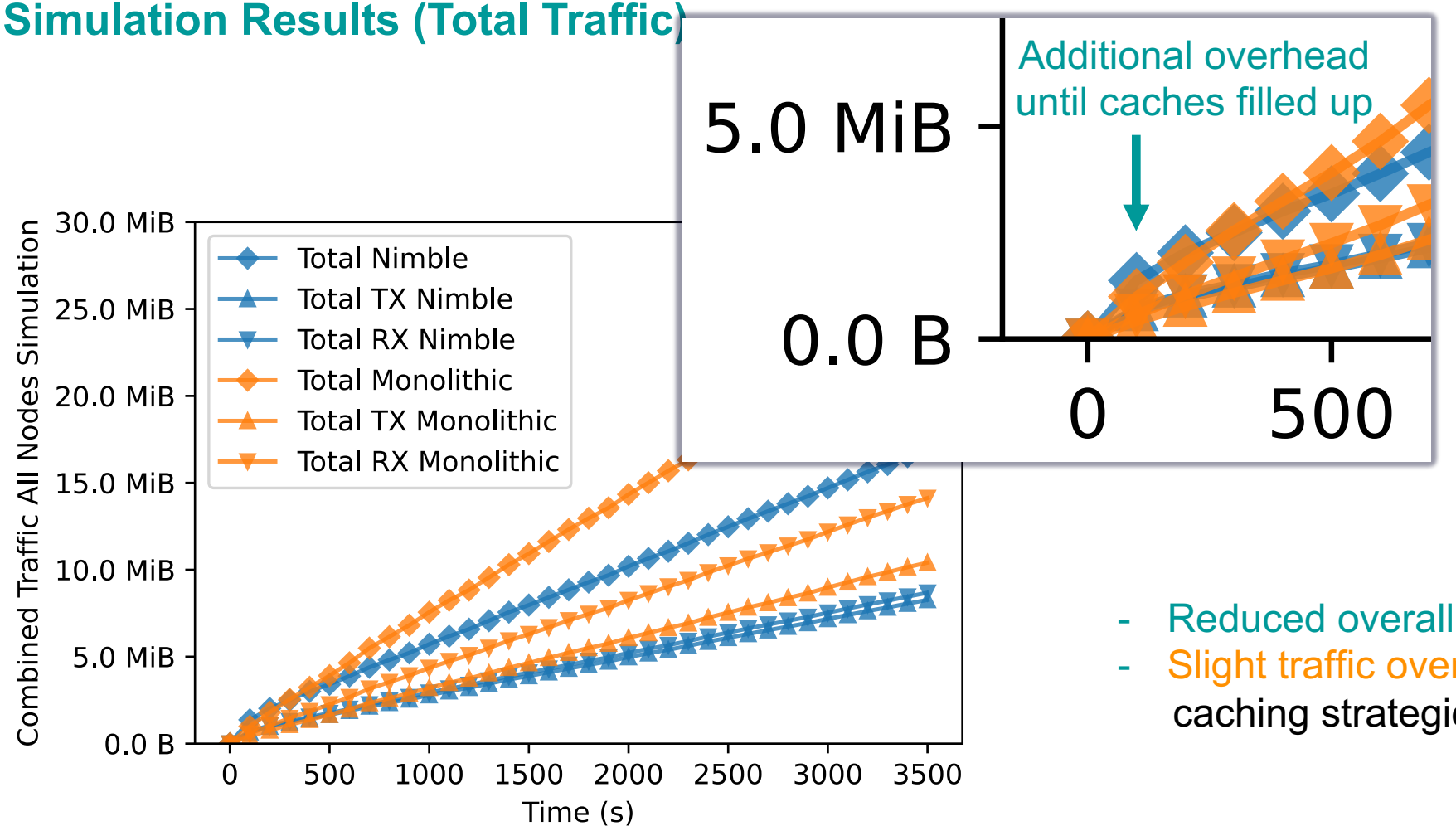
Simulation Results (Traffic Distribution)



Simulation after 3600 s

- Reduced overall traffic on ‘sink nodes’
  - See **‘energy hole effect’**
- More equal traffic distribution through out the whole network
- Traffic increase (mostly TX) on some nodes
  - Slightly increased binary size
  - Acquisition overhead

Simulation Results (Total Traffic)

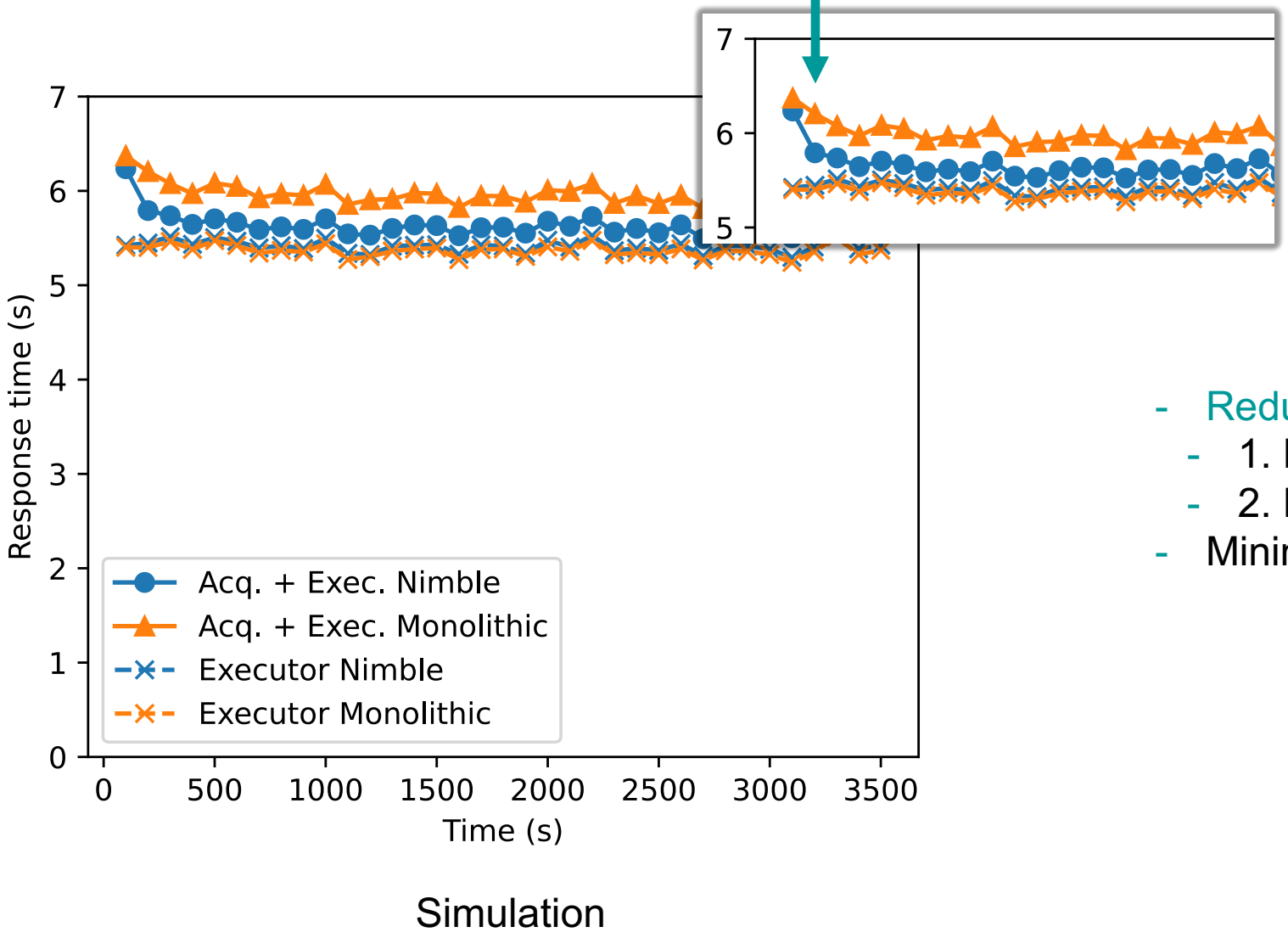


- Reduced overall network traffic
- Slight traffic overhead until caching strategies (LFU) start to have impact

Simulation

Simulation Results (Response Time)

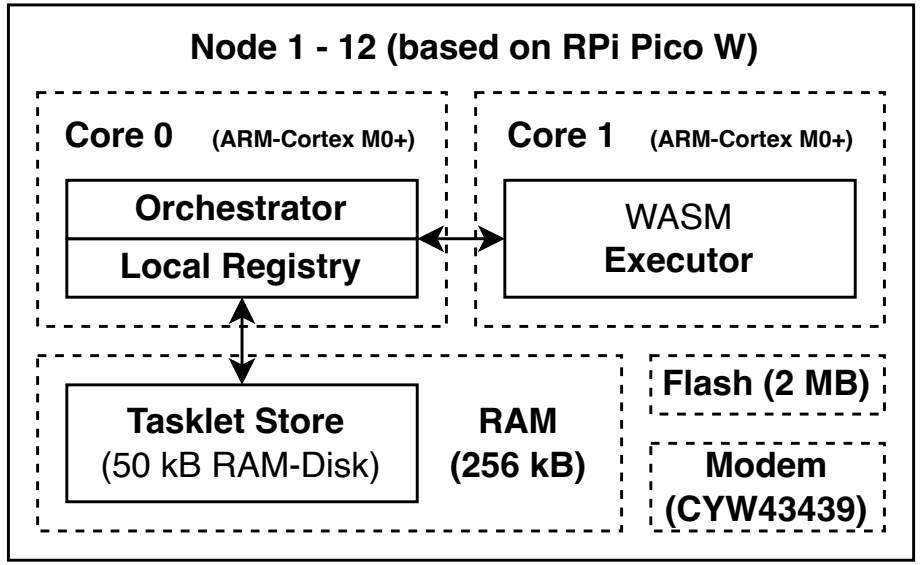
Additional overhead
until tasklets propagated into the network

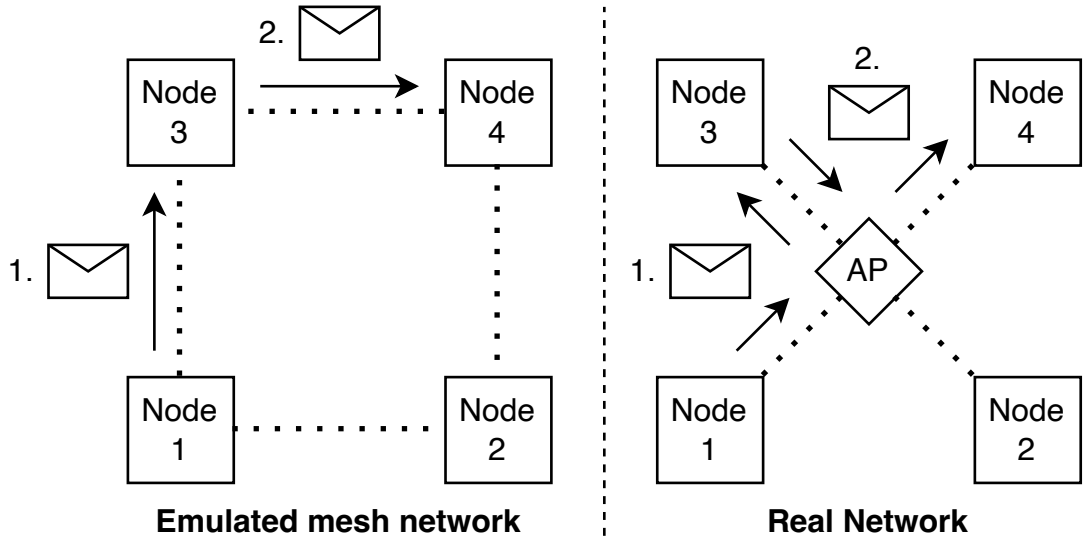


Simulation

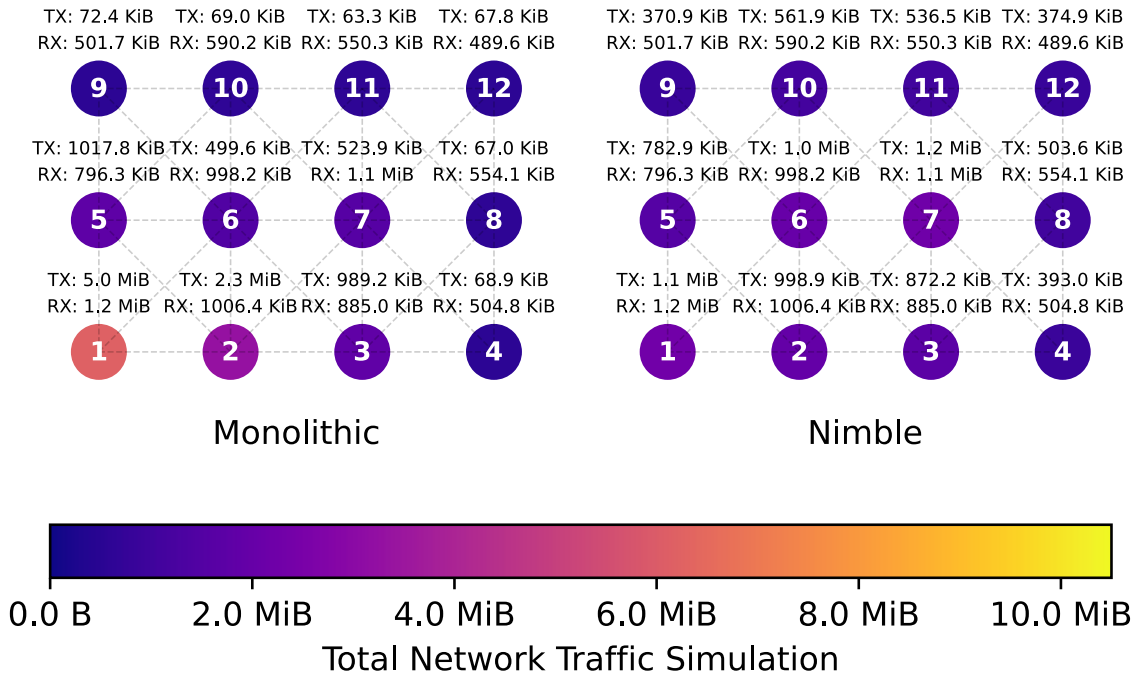
- Reduced response times due to
  - 1. Local **function reuse** (local cache)
  - 2. **Neighbour-first** caching (fewer hops)
- Minimal context switching time

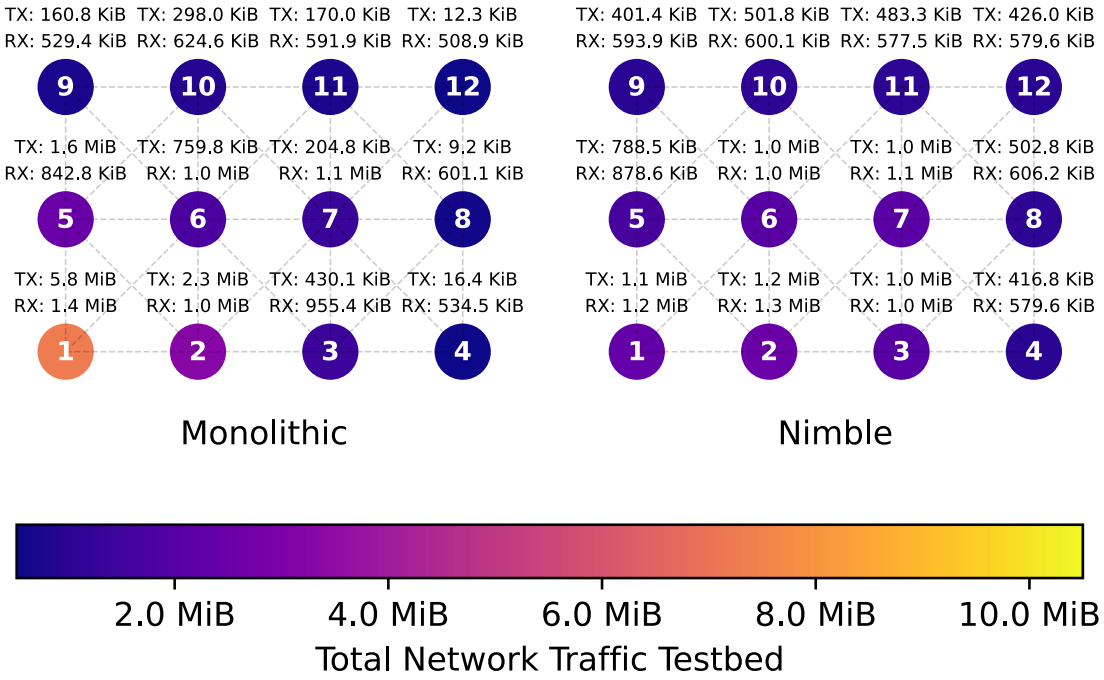
NimbleNet Testbed





Testbed Results (Traffic Distribution)



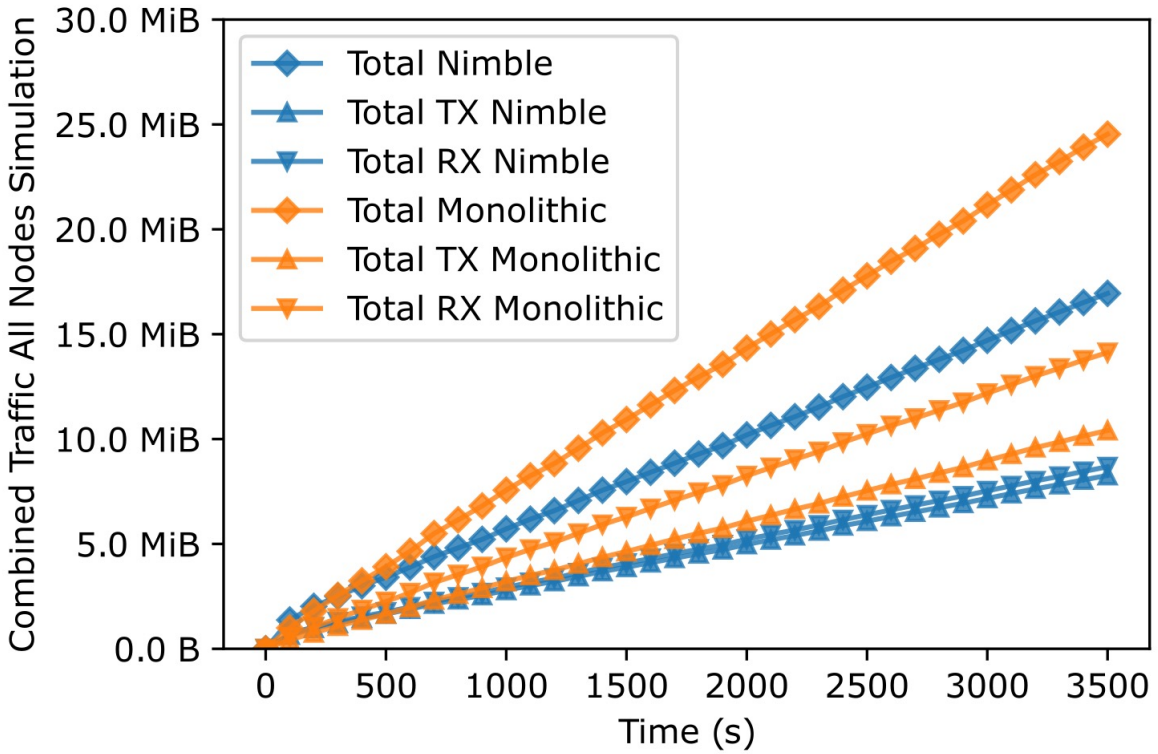


Simulation after 3600 s

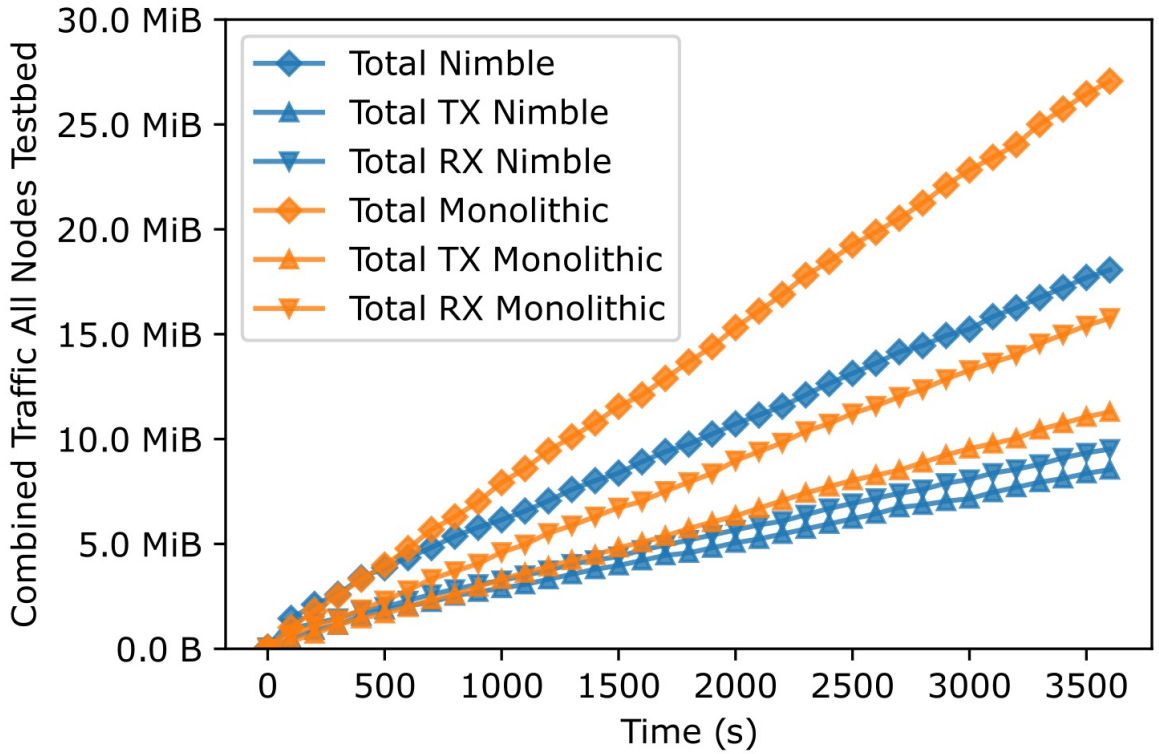
Testbed after 3600 s

FAU | SIEMENS

Testbed Results (Total Traffic)

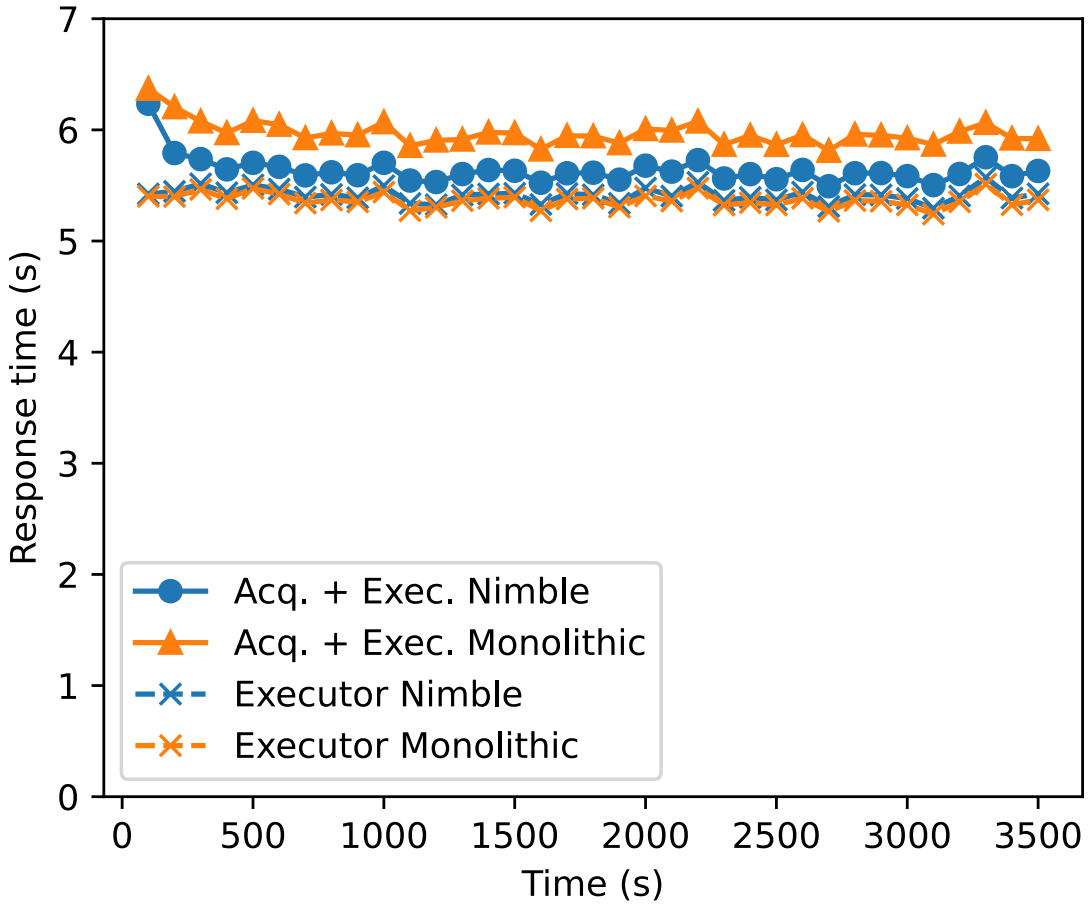


Simulation

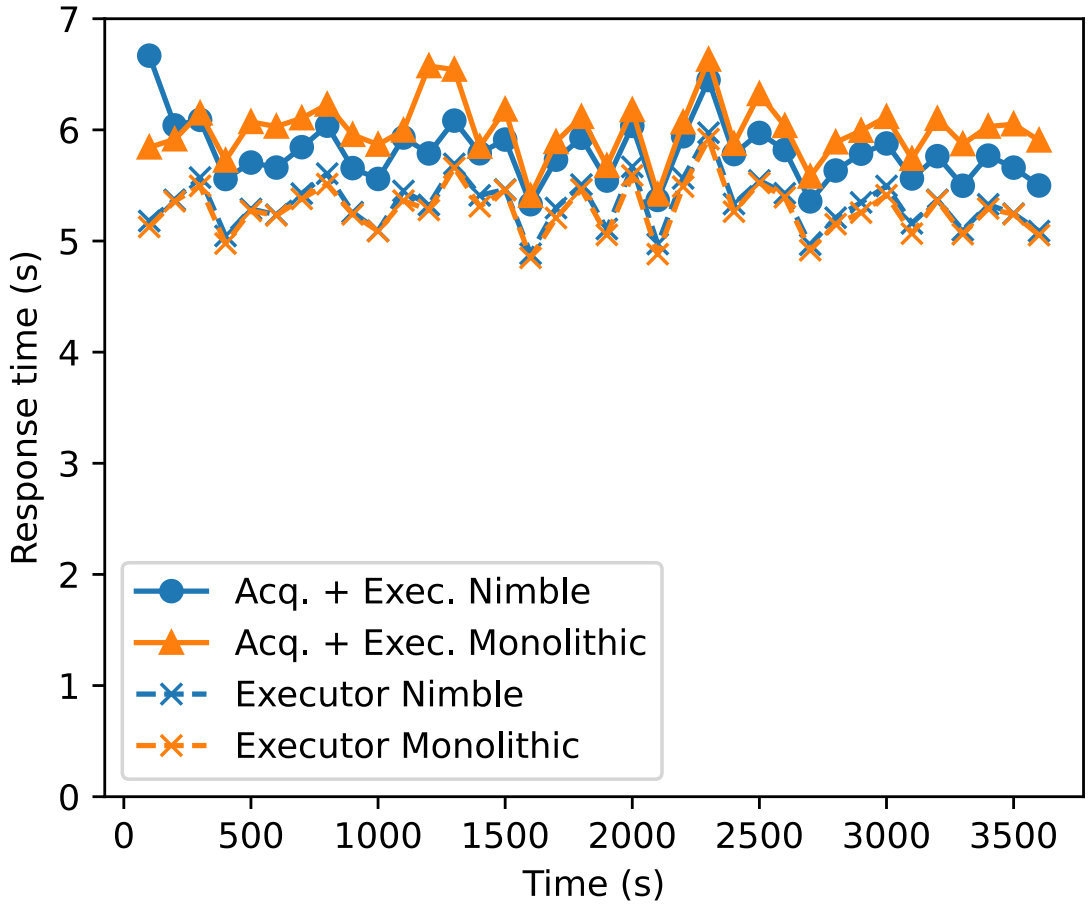


Testbed

Testbed Results (Response Time)

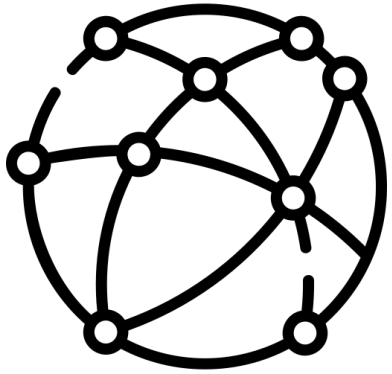


Simulation



Testbed

Additional Usecases & Potential Future Work



Efficient & Fast
In-network computing

Redundant & Fast
In-platoon computing

Contact

Kilian Müller
PhD Candidate
FT RPD SSI-DE
Schuckertstr. 2
91058 Erlangen
Germany
Email:
kilian.mueller@siemens.com
kilian.felix.mueller@fau.de

Maximilian Seidler
PhD Candidate
FT RPD SSI-DE
Schuckertstr. 2
91058 Erlangen
Germany
Email:
maximilian.seidler@siemens.com
maximilian.seidler@fau.de