DYNAMIC RECONFIGURATION AND IMPLEMENTATION IN ADAPTIVE AUTOSAR

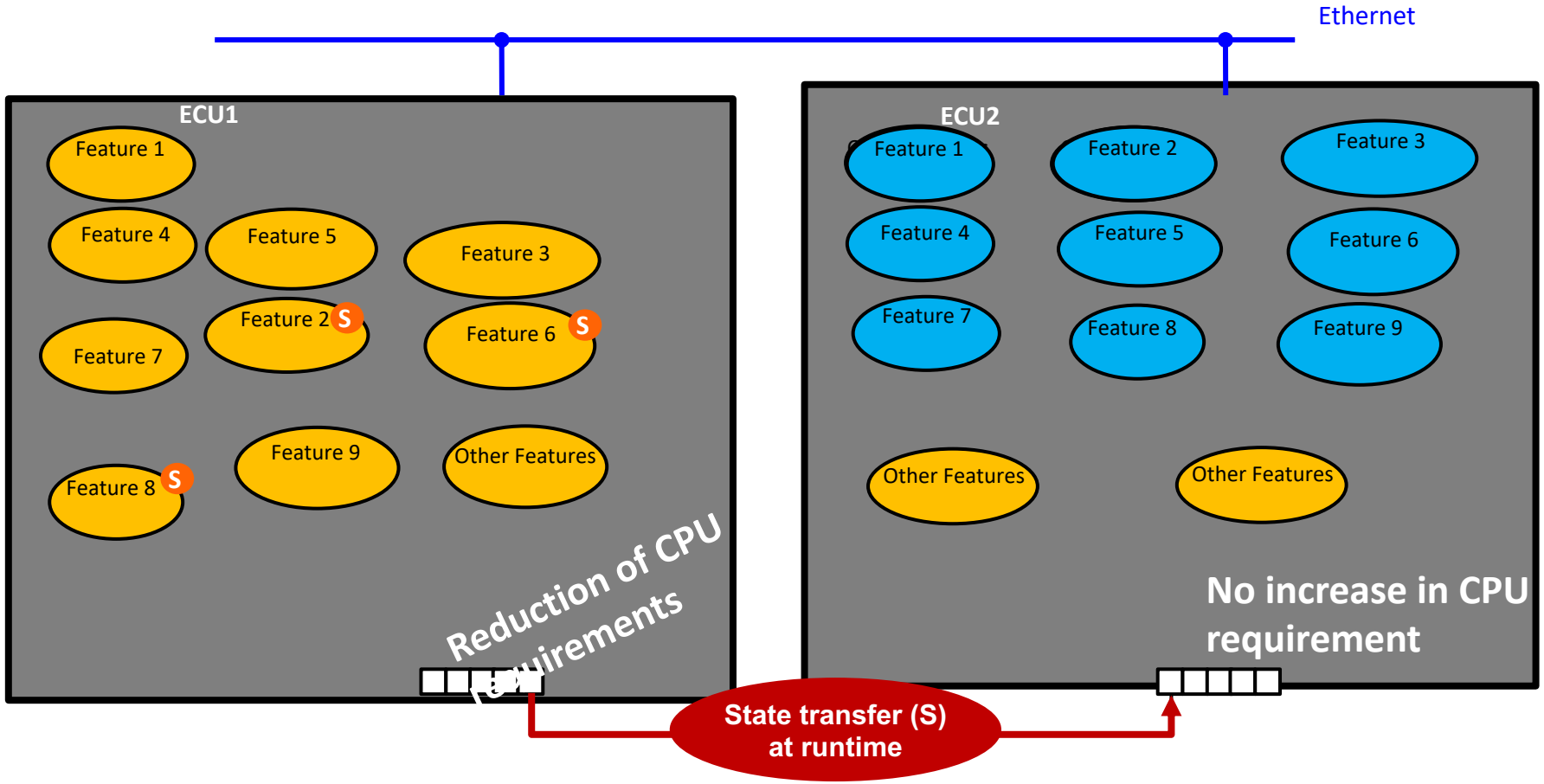
YUCHEN ZHOU

GM R&D

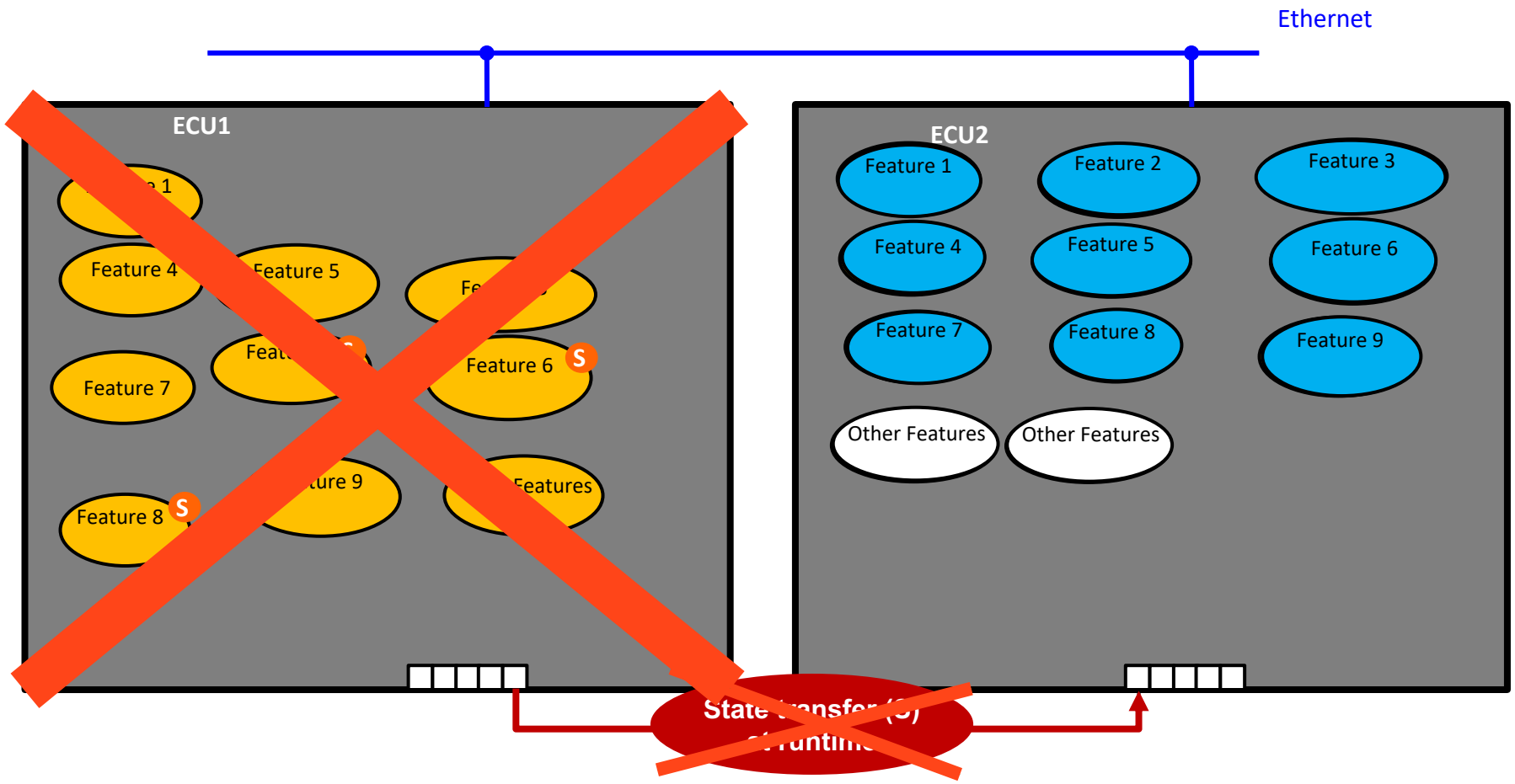
DYNAMIC RECONFIGURATION

- SAE level 3 and above automated driving can have varying redundancy needs for backup when driver is not in the loop
- In case of a failure of its main computing system a redundant computing system can serve as a backup for executing control tasks.
- Dynamic reconfiguration is a way to reduce costly execution of duplicates all the time, while maintaining situation awareness for the cold standby applications.
- The method:
  - Passing state information between main execution of an application to the cold standby duplicates without actually running the whole application
  - At the time of fault, launch the redundant application and establish sensor and actuation communication to the new application

FROM ASYMMETRIC ALLOCATION TO DYNAMIC RECONFIGURATION



ECU1 FAILURE AND RECONFIGURATION



IMPLEMENTATION OBJECTIVES

QUALITY GOALS

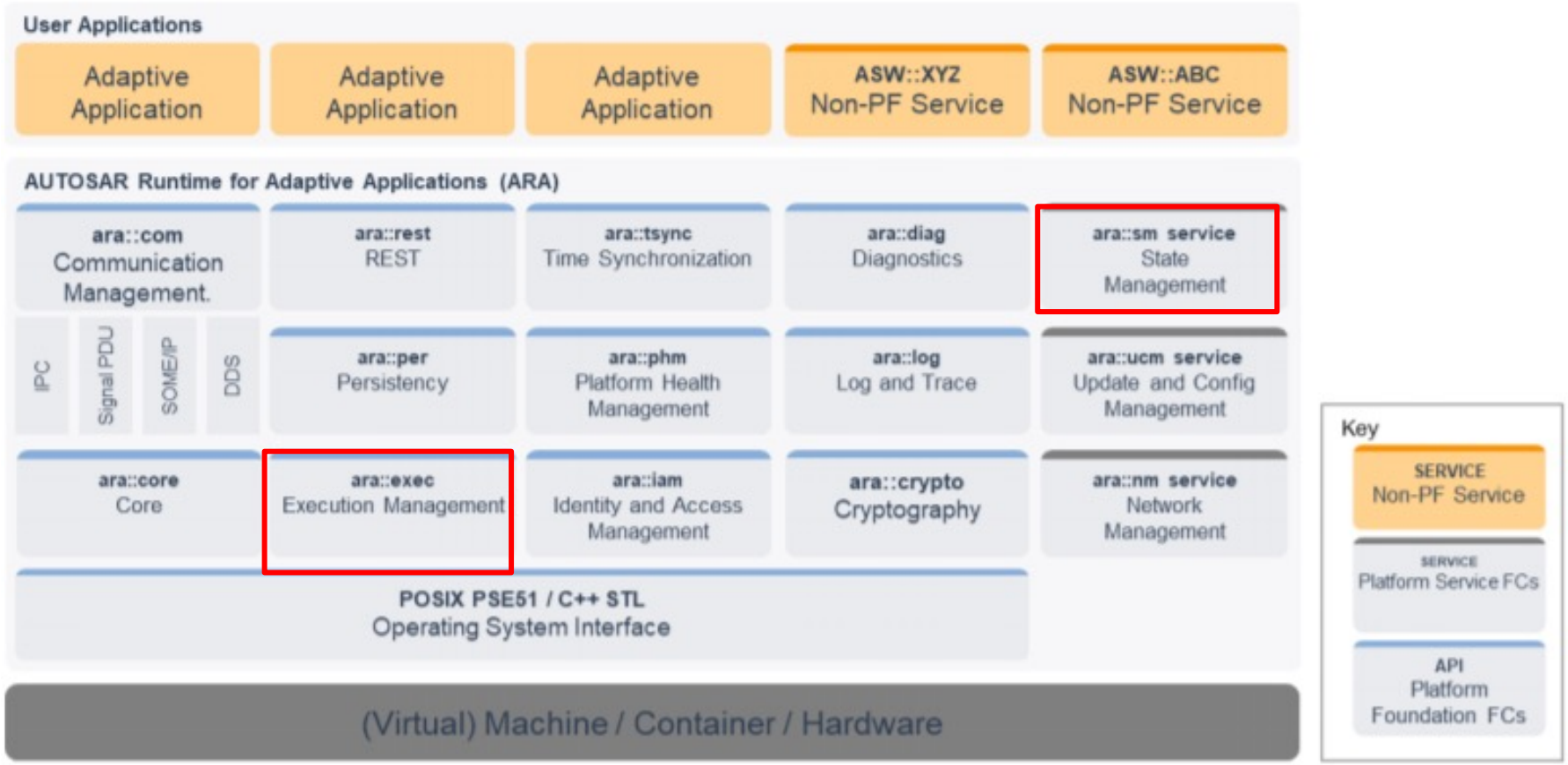
- Performance
  - Need to meet the fault handling time
- Resource consumption
  - To reduce cost

SOLUTION IDEA:

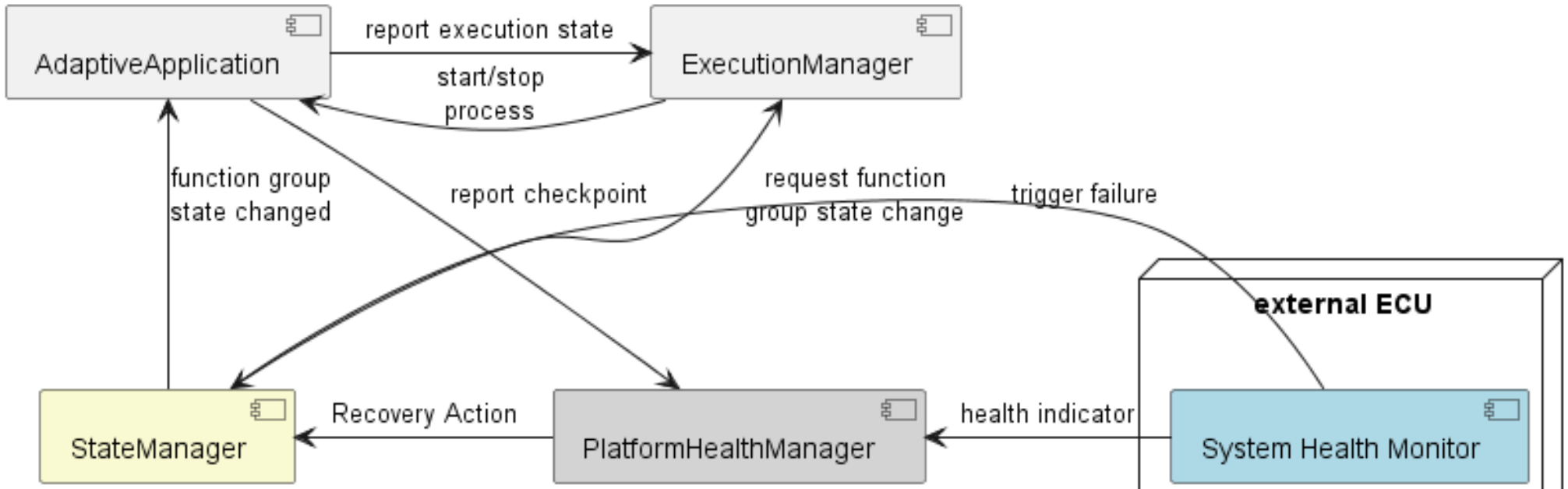
- Backup the *internal state* of an application on a different machine.
- In case of failure: Fallback application takes over
  - Fallback application standby modes

ADAPTIVE AUTOSAR IMPLEMENTATION

- Adaptive AUTOSAR provided essential runtime for managing execution state, i.e. standby, and activation and running applications through functional groups.



CONCEPT – INVOLVED CLUSTERS



STATE MANAGER (SM)

Manages running applications by using function groups

Manages system state

- Standby or running

PLATFORM HEALTH MANAGER (PHM)

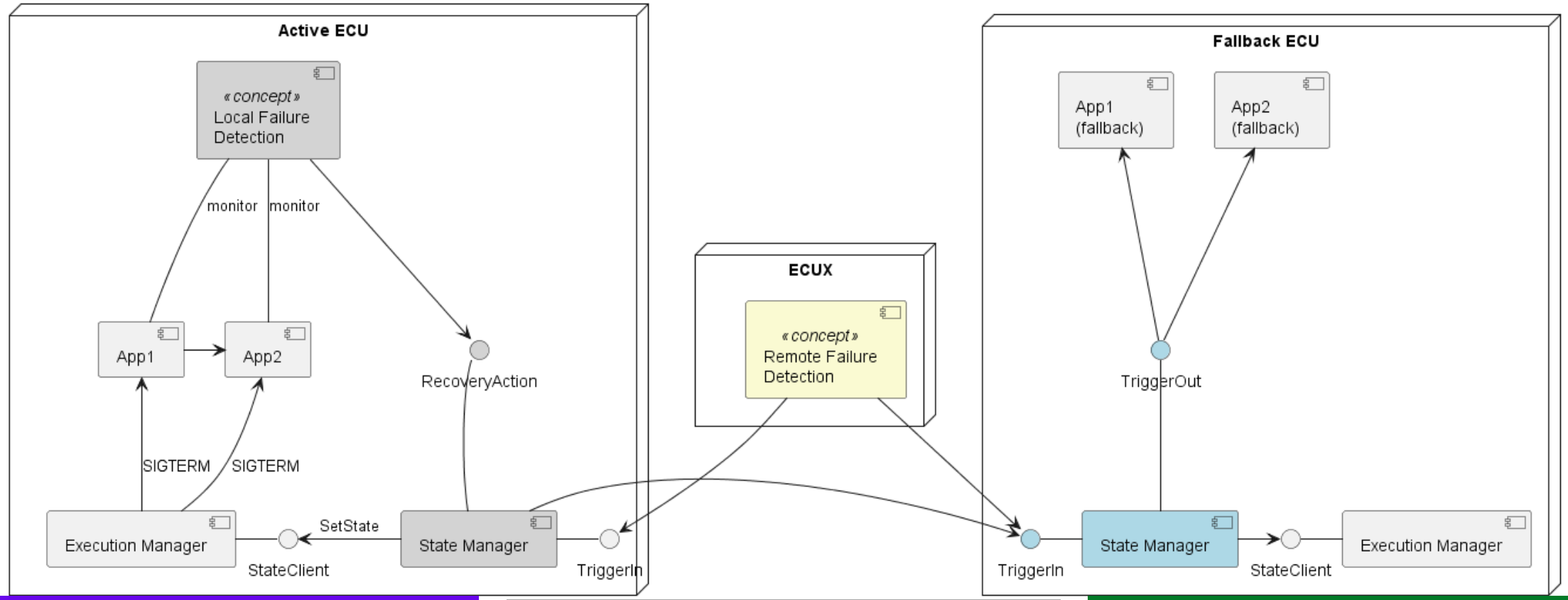
Monitors application health

Monitors health of machine

SYSTEM HEALTH MONITOR (SHM)

Monitors health of multiple machines

CONCEPT – FAILURE DETECTION



REMOTE FAILURE DETECTION

adaptive AUTOSAR System Health Monitor concept

Triggers fallback SM with TriggerIN Interface.

LOCAL FAILURE DETECTION

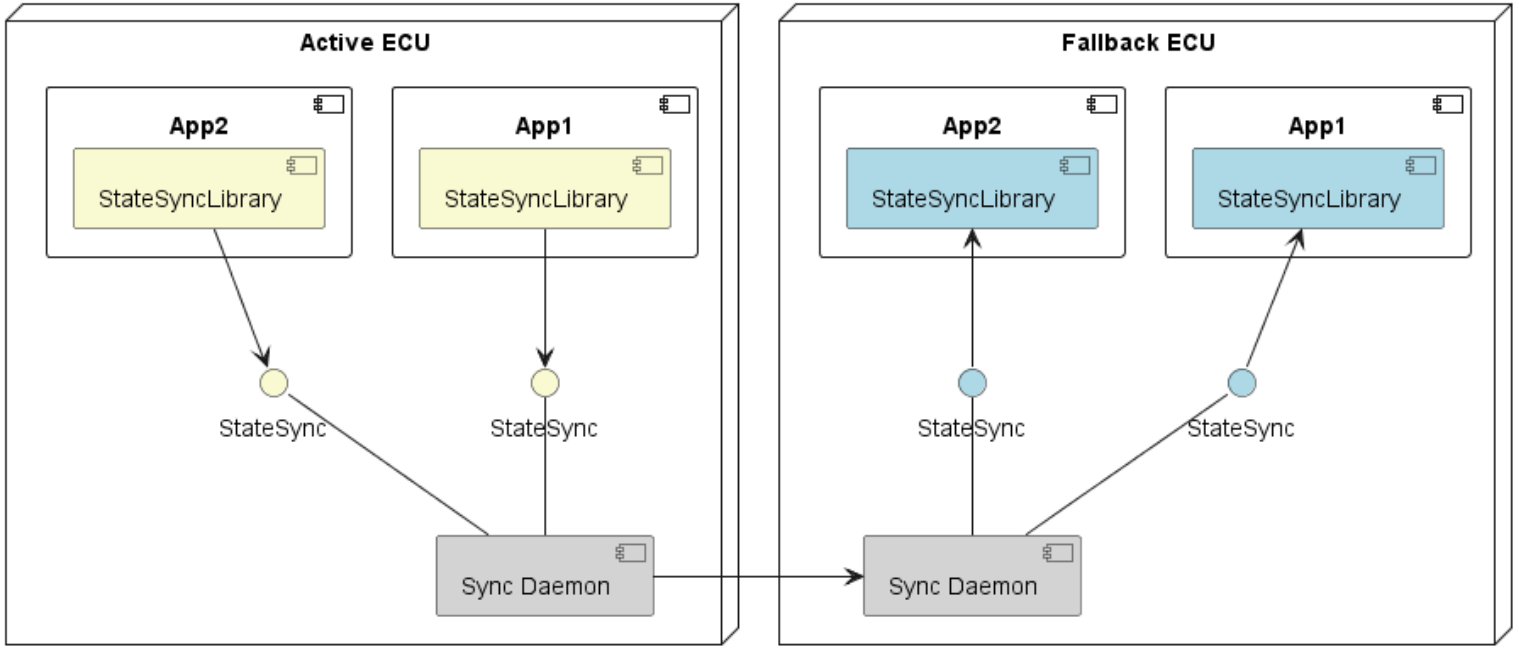
adaptive AUTOSAR Platform Health Manager concept

PHM calls recovery action of SM, that notifies the SM on the fallback ECU

RECONFIGURATION ACTIVATION

SM triggers running fallback Application

CONCEPT – STATE SYNCHRONIZATION



STATE SYNC TO DAEMON

State sync library writes internal state into shared memory.

Sync Daemon manages shared memory

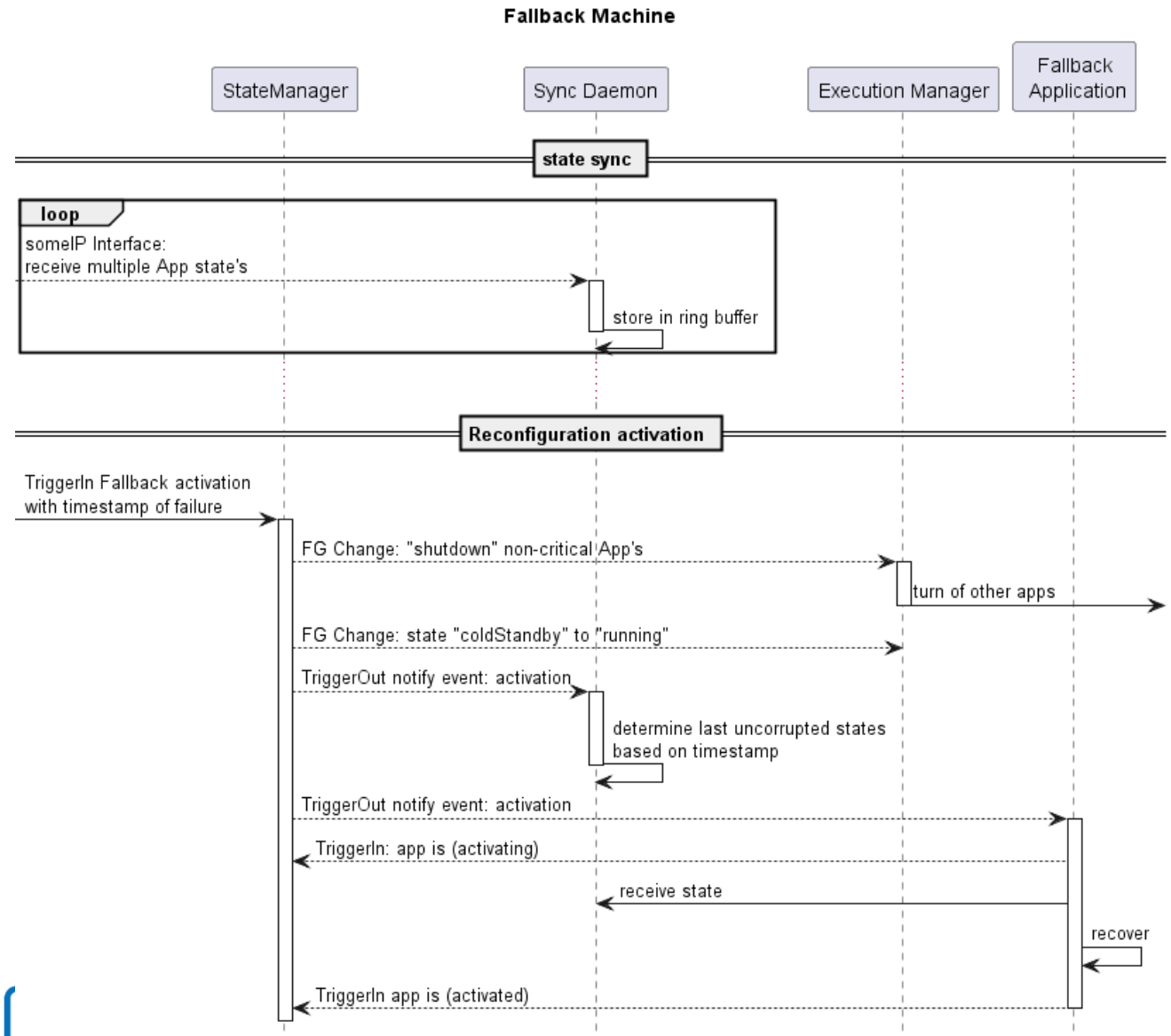
SYNC BETWEEN DAEMON'S (ECU'S)

ara:com Events with state collection as backup

RECONFIGURATION ACTIVATION

SM triggers Sync Daemon at failure time to provide the state's to the fallback applications

CONCEPT – FALLBACK ACTIONS



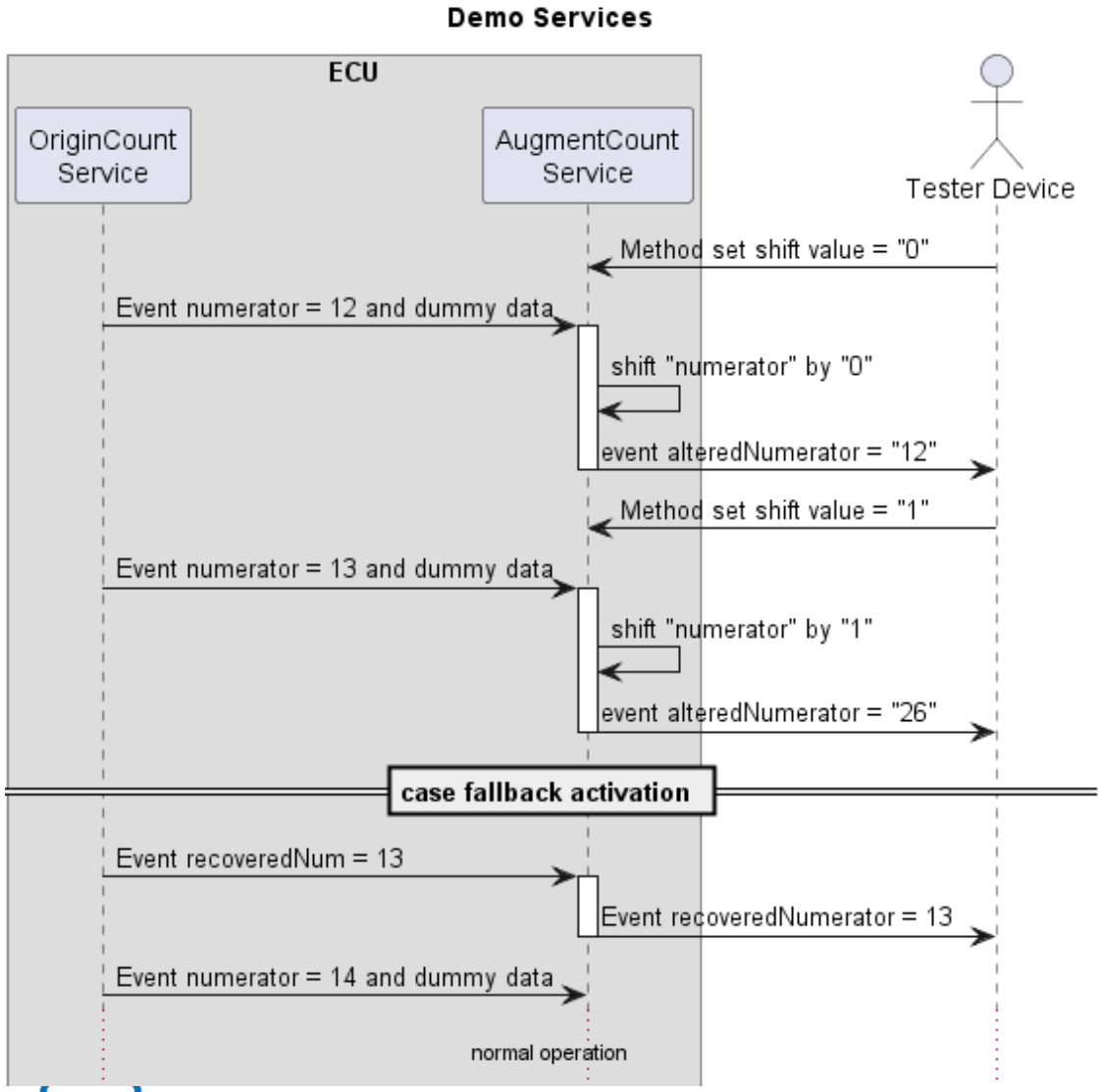
Process of Application already running.

Application and “Sync Daemon” are subscribed to events from TriggerOut interface

Fallback function group state “running” could trigger PHM

TriggerIn from Application is used to rebuild execution dependencies

DEMO FUNCTIONALITY



DEMO SCENARIO

OriginCount is an IPC Service counting up

AugmentCount can bit-shift the value and provide it via some/IP protocol

Dummy Data of “OriginCount” service can be configured

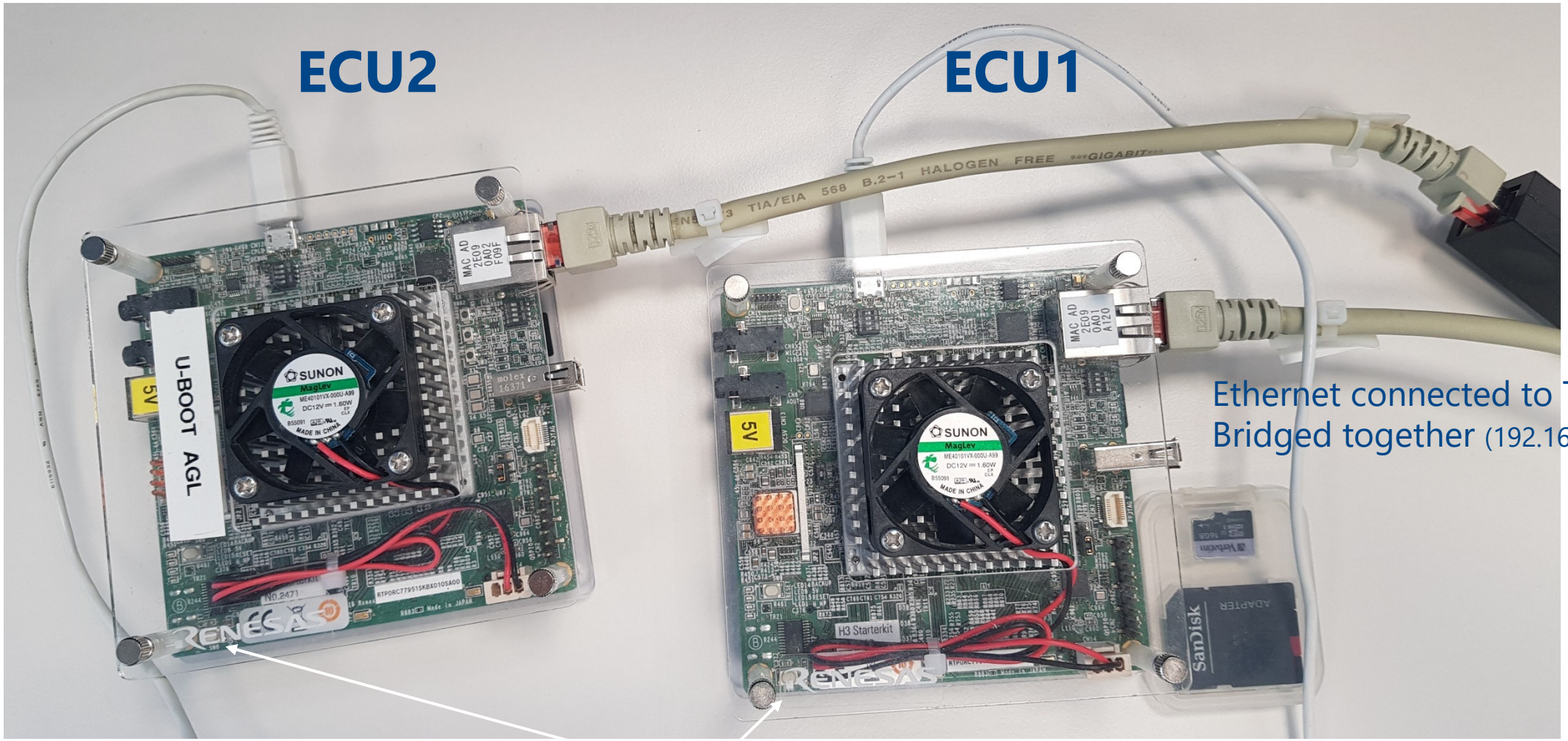
- 50 / 100 / 1000 bytes used for testing

Frequency of “OriginCount” event’s can be configured

- 100 / 50 / 25 milliseconds used for testing

Service AugmentCount is execution dependent on OriginCount

IMPLEMENTATION SETUP



Ethernet connected to Tester
Bridged together (192.168.56.1/24)

Power Button SW8

RESULTS

- State Backup latency between ECU'S took ~2.3 ms
- Recovery time
  - for App1 was ~3ms
  - for dependent app2 ~4.9ms

Timestamp	Ecuid	Apid	Ctid	Type	Subtype	Payload
54.5888	ECU2	FSDM	MEAS	log	fatal	BX2 --> State Backup Ev
52.5268	ECU1	SMA	MEAS	log	fatal	AE1 --> ECU1 time at dyn
52.8115	ECU1	SDMN	MEAS	log	fatal	AX2 --> sending StateBacku
55.5891	ECU2	FSDM	MEAS	log	fatal	BX2 --> State Backup Ever
56.9790	ECU2	FADA	MEAS	log	fatal	A1 --> detected failure
56.9820	ECU2	FAP1	MEAS	log	fatal	B1 --> successful acti
56.9838	ECU2	FAP2	MEAS	log	fatal	C1 --> successful act

- Memory consumption was about constant
  - SM: 6.5MB, Daemon: 4.6MB, App1: 4.5MB, App2: 6.4MB
- CPU usage below 1%