Schedule Management Framework for Cloud-based Future Automotive Software Systems

Licong Zhang[1], Debayan Roy[1], Philipp Mundhenk[2], Samarjit Chakraborty[1]
[1] TU Munich, Germany
[2] TUM CREATE, Singapore

RTCSA 2016
Aug. 17, 2016, Daegu

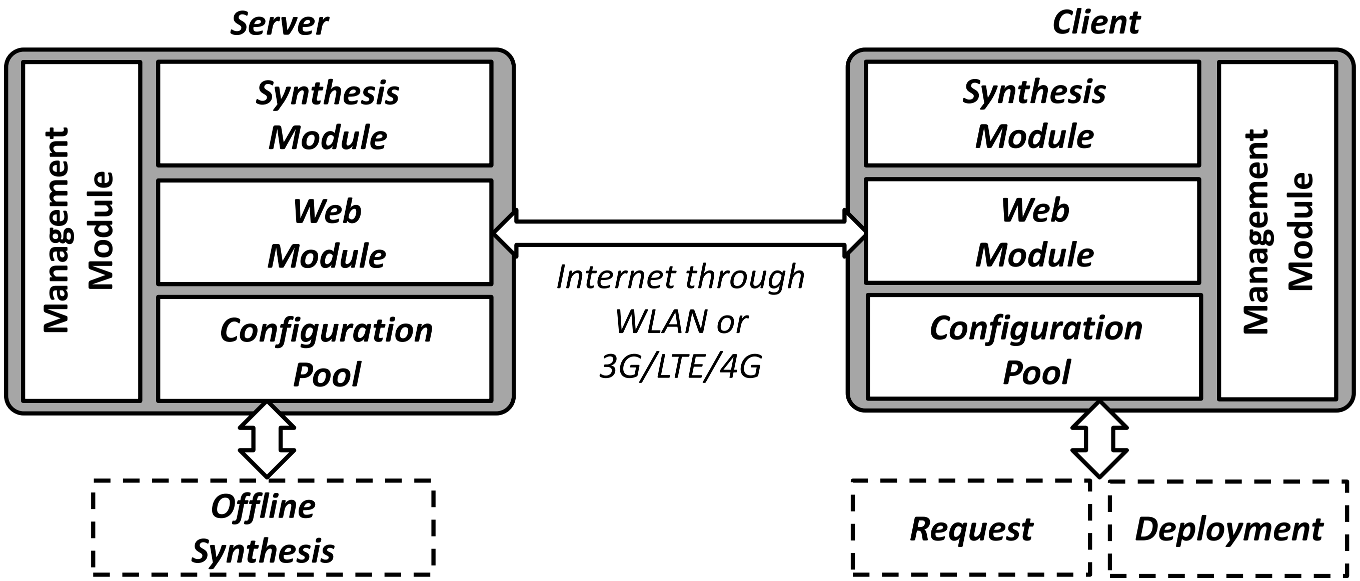
Overview

- **Problem**
  - Schedule synthesis for Ethernet-based time-triggered system
  - Online schedule generation and management for Plug-and-Play scenario

Overview

- **Problem**
  - Schedule synthesis for Ethernet-based time-triggered system
  - Online schedule generation and management for Plug-and-Play scenario

- **Approach**
  - Software framework for schedule management
  - Utilization of both local computation and cloud-computing
  - Four-stage scheduling strategy, online schedule synthesis, configuration pool

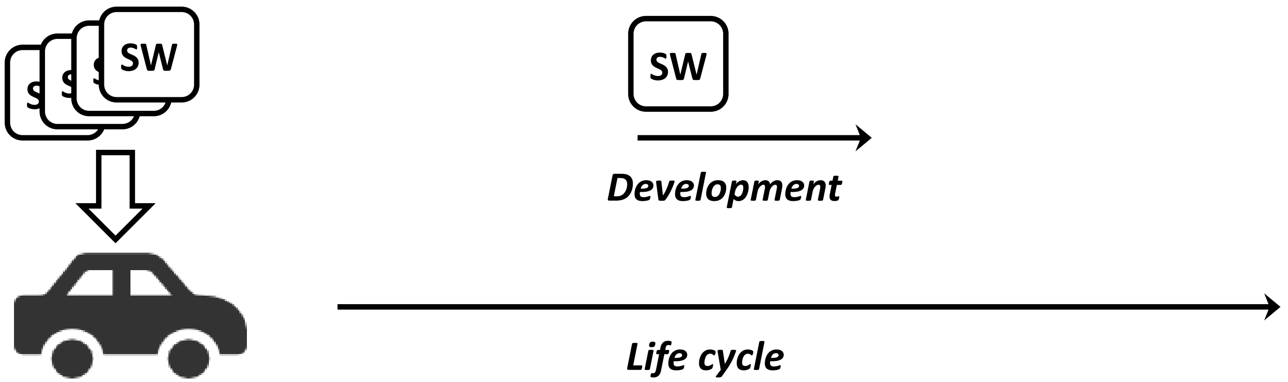


Outline

- Motivation
- Background
- Problem Formulation
- Proposed Framework
- Experimental Results
- Concluding Remarks

Motivation

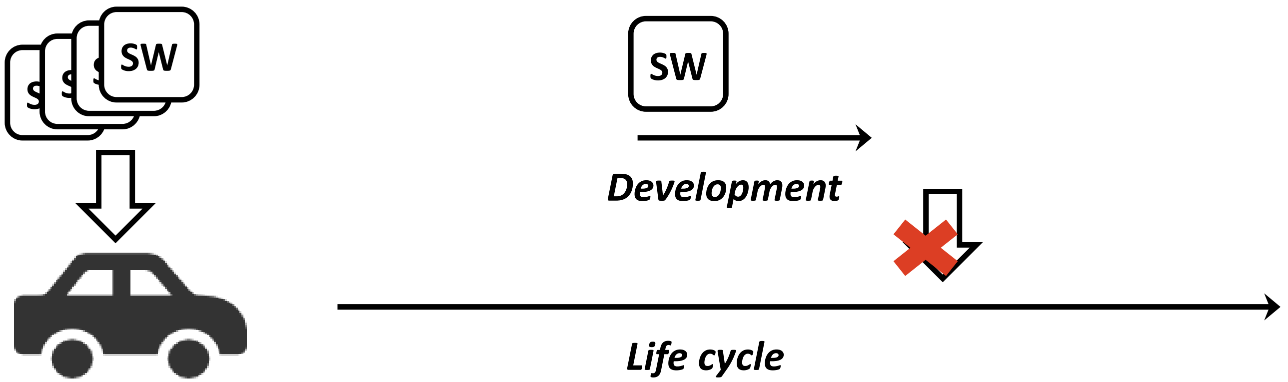
- **Software update and installation after sales**
  - Shift of innovation in automotive domain to Electrical/Electronics systems and software
  - Development cycle of electronic system and software is much shorter than vehicle life cycle
  - Increasingly more new software functions, e.g., in driver assistance, autonomous driving and infotainment domain



Motivation

- **Software update and installation after sales**
  - Shift of innovation in automotive domain to Electrical/Electronics systems and software
  - Development cycle of electronic system and software is much shorter than vehicle life cycle
  - Increasingly more new software functions, e.g., in driver assistance, autonomous driving and infotainment domain

➢ Functionality of a vehicle becomes ‚outdated' easily

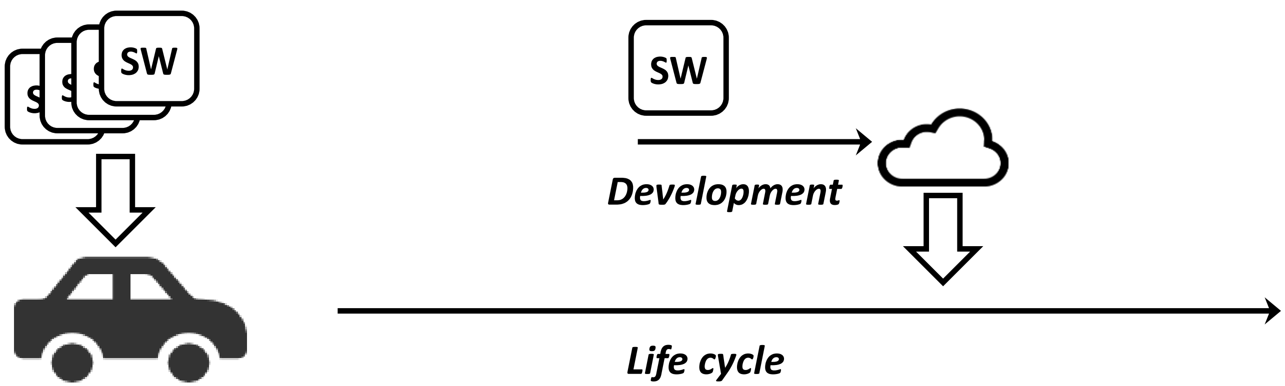


Motivation

- **Software update and installation after sales**
  - Shift of innovation in automotive domain to Electrical/Electronics systems and software
  - Development cycle of electronic system and software is much shorter than vehicle life cycle
  - Increasingly more new software functions, e.g., in driver assistance, autonomous driving and infotainment domain

  ➢ Functionality of a vehicle becomes ‚outdated‘ easily

  - In the future it would advantageous for both car manufacturers and customers that
    - Software functions can be constantly updated
    - Newly developed software can be deployed after sales



Motivation

- **Software update and installation after sales**
  - Shift of innovation in automotive domain to Electrical/Electronics systems and software
  - Development cycle of electronic system and software is much shorter than vehicle life cycle
  - Increasingly more new software functions, e.g., in driver assistance, autonomous driving and infotainment domain

  ➢ Functionality of a vehicle becomes ‚outdated‘ easily

  - In the future it would advantageous for both car manufacturers and customers that
    - Software functions can be constantly updated
    - Newly developed software can be deployed after sales

- **Cloud-based future automotive software systems**
  - Internet connection for cars
  - Vehicle is becoming increasingly autonomous

  ➢ Autonomous detection of driving condition and download software applications on demand

Motivation

- **Plug-and-play of software applications**
  - Software applications can be installed or updated through storage devices (e.g., USB stick) or through cloud service

Motivation

- **Plug-and-play of software applications**
  - Software applications can be installed or updated through storage devices (e.g., USB stick) or through cloud service

- **Challenges and issues to be addressed**
  - Supply chain – ECU with single independent function
  - ➢ ECU consolidation – ECUs are becoming computing platforms and multiple applications can be mapped on a single ECU

Motivation

- **Plug-and-play of software applications**
  - Software applications can be installed or updated through storage devices (e.g., USB stick) or through cloud service

- **Challenges and issues to be addressed**
  - Supply chain – ECU with single independent function
  - ➢ ECU consolidation – ECUs are becoming computing platforms and multiple applications can be mapped on a single ECU

  - Deployment of software components – ECU software into single binary file
  - ➢ Related research works in developing operating systems and runtime environment to accommodate new software without flashing the whole ECU

Motivation

- **Plug-and-play of software applications**
  - Software applications can be installed or updated through storage devices (e.g., USB stick) or through cloud service

- **Challenges and issues to be addressed**
  - Supply chain – ECU with single independent function
  - ➢ ECU consolidation – ECUs are becoming computing platforms and multiple applications can be mapped on a single ECU

  - Deployment of software components – ECU software into single binary file
  - ➢ Related research works in developing operating systems and runtime environment to accommodate new software without flashing the whole ECU

  - Reallocation of communication and computation resources
  - ➢ The problem to be addressed in this work

Motivation

- **Resource reallocation**
  - Change of configuration for processor scheduling and network scheduling
  - Issues like schedule generation, schedule deployment and safe reconfiguration process need to be addressed

Motivation

- **Resource reallocation**
  - Change of configuration for processor scheduling and network scheduling
  - Issues like schedule generation, schedule deployment and safe reconfiguration process need to be addressed

  ➢ This work focuses on the schedule generation and management problem

Motivation

- **Resource reallocation**
  - Change of configuration for processor scheduling and network scheduling
  - Issues like schedule generation, schedule deployment and safe reconfiguration process need to be addressed

  ➢ This work focuses on the schedule generation and management problem

- **Time-triggered System**
  - Task and network transmission are triggered according to pre-calculated schedules
  - In the case of adding new applications or the update changes requirements, new schedule set needs to be calculated.

Motivation

- **Resource reallocation**
  - Change of configuration for processor scheduling and network scheduling
  - Issues like schedule generation, schedule deployment and safe reconfiguration process need to be addressed

  ➢ This work focuses on the schedule generation and management problem

- **Time-triggered System**
  - Task and network transmission are triggered according to pre-calculated schedules
  - In the case of adding new applications or the update changes requirements, new schedule set needs to be calculated.

- **Requirements**
  - Obtain schedules online in relatively short time
  - As many as possible new applications can be accommodated
  - Facilitation of schedule reuse and minimization of disturbance to existing schedules

Motivation

- **Related Works**
  - Schedule synthesis problem for Ethernet-based time-triggered systems [10,11,13,15,16]
  - Incremental scheduling [11,12]
  - Configuration and reconfiguration of time-triggered Ethernet networks [18,19]
  - Plug-and-Play in the automotive setting [2,3,5,6]

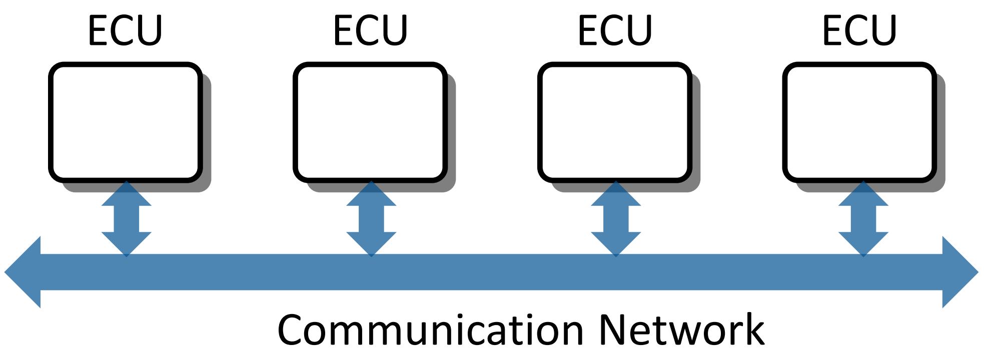
Motivation

- **Related Works**
  - Schedule synthesis problem for Ethernet-based time-triggered systems [10,11,13,15,16]
  - Incremental scheduling [11,12]
  - Configuration and reconfiguration of time-triggered Ethernet networks [18,19]
  - Plug-and-Play in the automotive setting [2,3,5,6]

- **Contributions**
  - Software framework for schedule generation and management for Plug-and-Play
  - Online schedule synthesis based on a mixture of embedded and cloud computing
  - Configuration pool for schedule reuse
  - A four-stage scheduling strategy offering trade-off between chance of accommodating new applications and synthesis time and disturbance to existing schedules

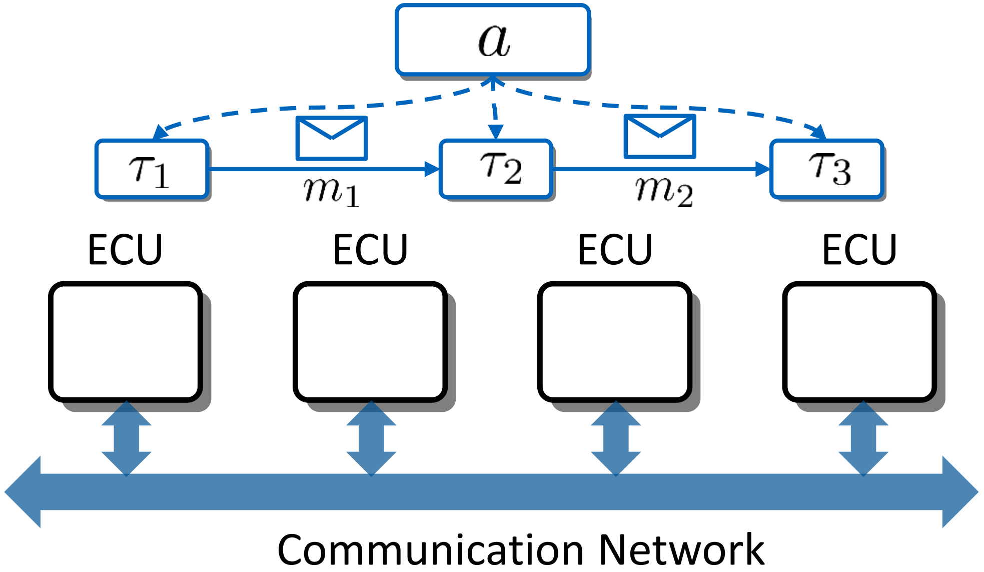
Background

- **Distributed embedded systems**
  - Hardware architecture



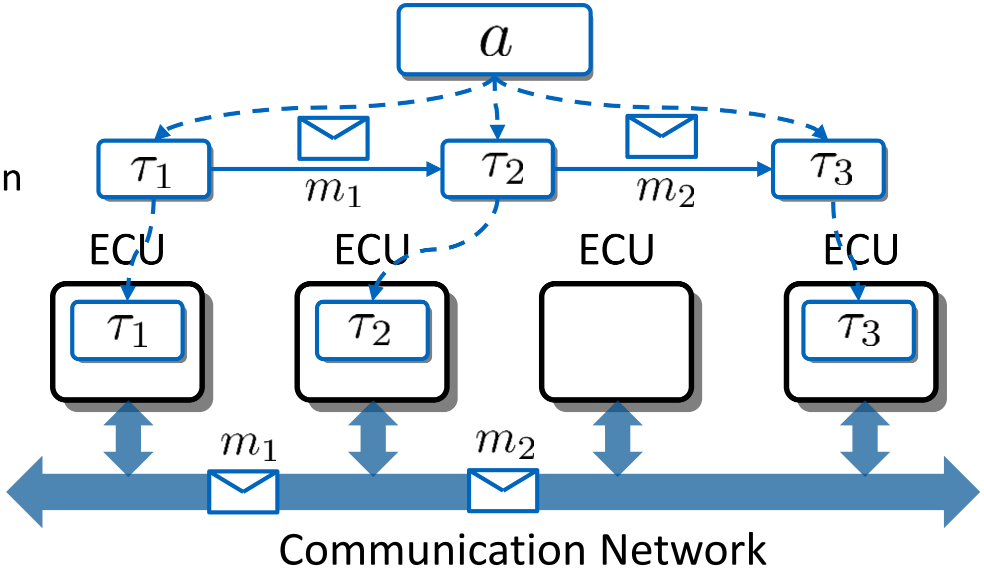
Background

- **Distributed embedded systems**
  - Hardware architecture
  - Distributed applications



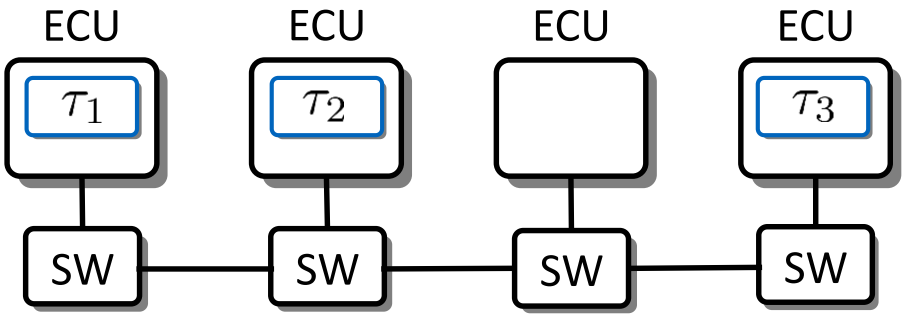
Background

- **Distributed embedded systems**
  - Hardware architecture
  - Distributed applications
  - Task mapping / network communication



Background

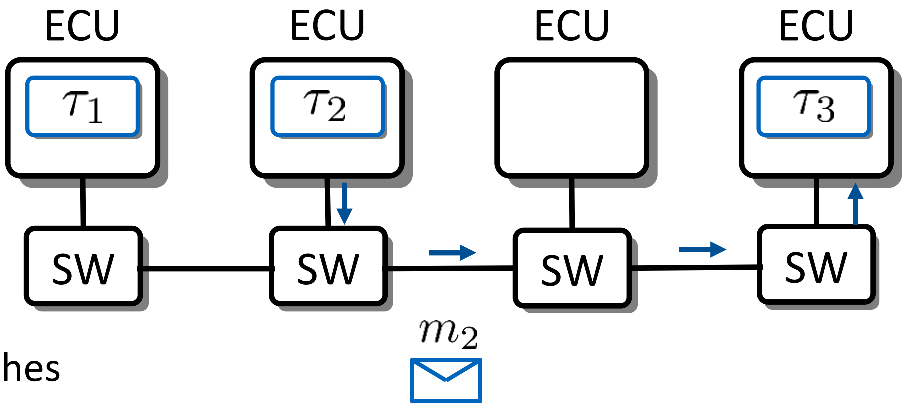
- **Distributed embedded systems**
  - Hardware architecture
  - Distributed applications
  - Task mapping / network communication



- **Switched Ethernet**
  - Processing units connected through switches
  - Commonly with full-duplex links

Background

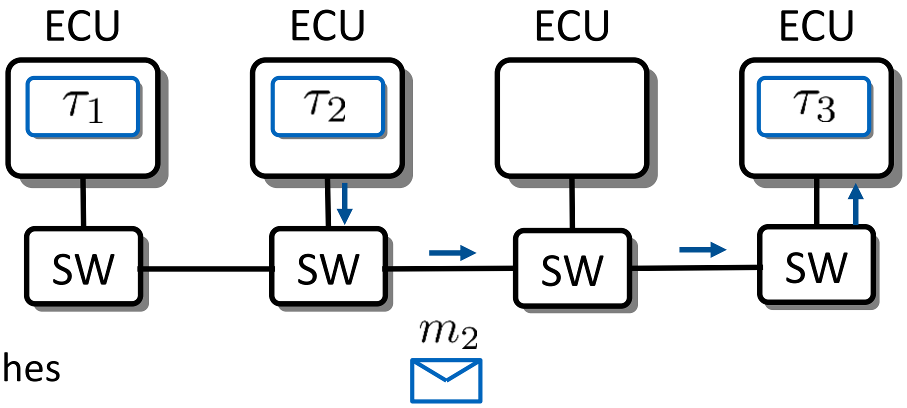
- **Distributed embedded systems**
  - Hardware architecture
  - Distributed applications
  - Task mapping / network communication



- **Switched Ethernet**
  - Processing units connected through switches
  - Commonly with full-duplex links
  - Ethernet frames forwarded switch by switch

Background

- **Distributed embedded systems**
  - Hardware architecture
  - Distributed applications
  - Task mapping / network communication

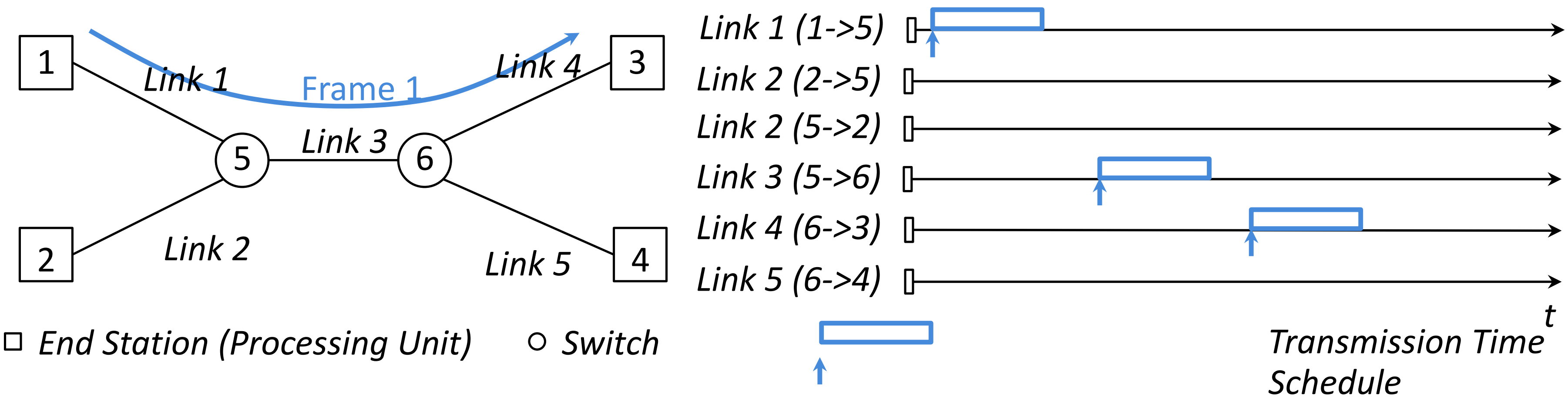


- **Switched Ethernet**
  - Processing units connected through switches
  - Commonly with full-duplex links
  - Ethernet frames forwarded switch by switch

➢ Queueing delay at each switch
  - Not deterministic
  - Can be relatively large

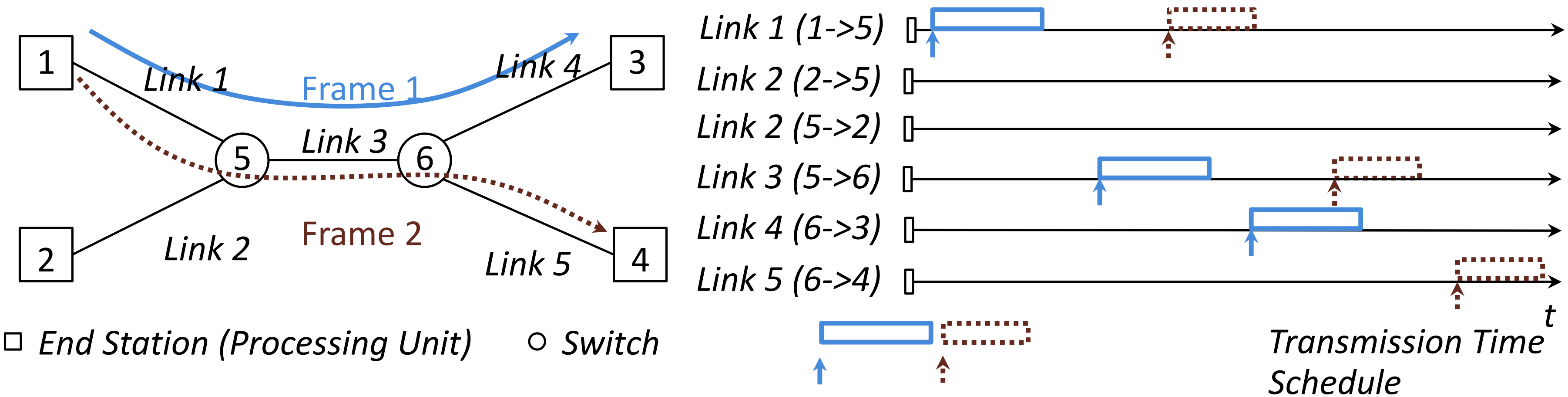
Background

- **Time-triggered Ethernet communication**
  - Frames are scheduled to avoid queueing delay
  - Frames transmission on each link according to static schedule



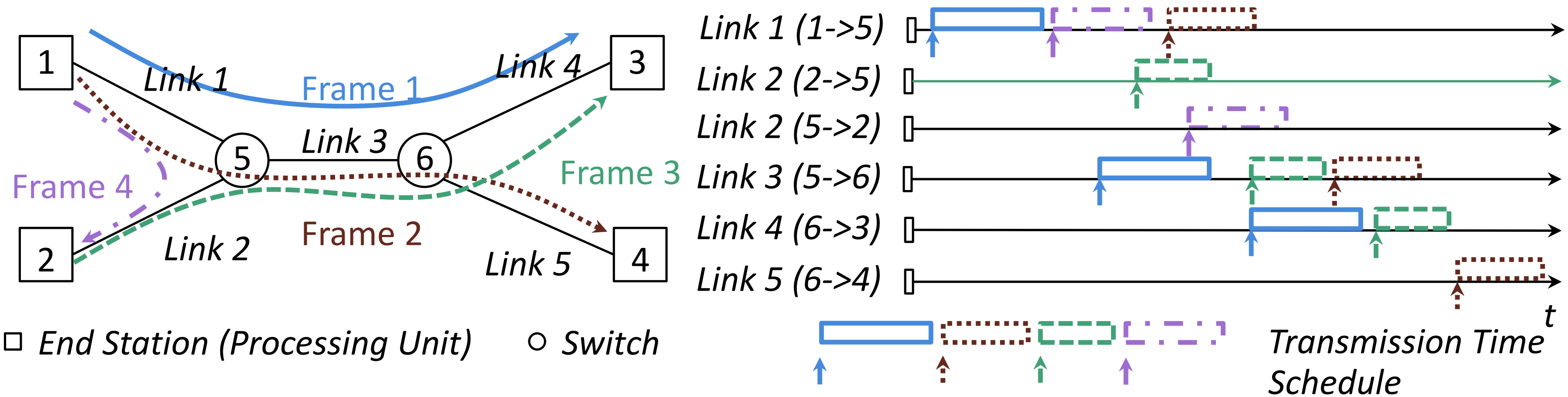
Background

- **Time-triggered Ethernet communication**
  - Frames are scheduled to avoid queueing delay
  - Frames transmission on each link according to static schedule



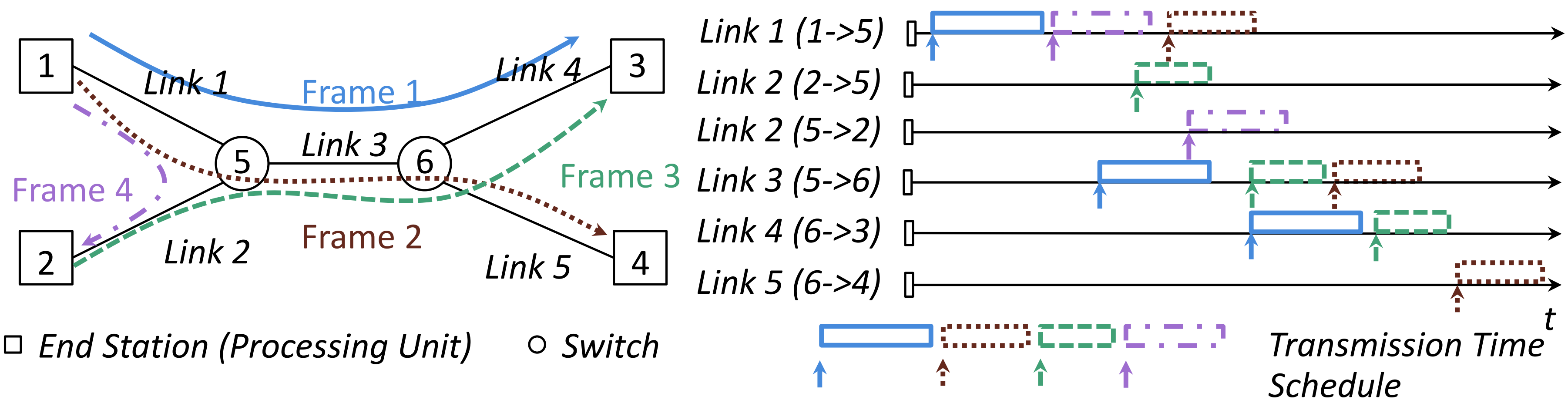
Background

- **Time-triggered Ethernet communication**
  - Frames are scheduled to avoid queueing delay
  - Frames transmission on each link according to static schedule



Background

- **Time-triggered Ethernet communication**
  - Frames are scheduled to avoid queueing delay
  - Frames transmission on each link according to static schedule



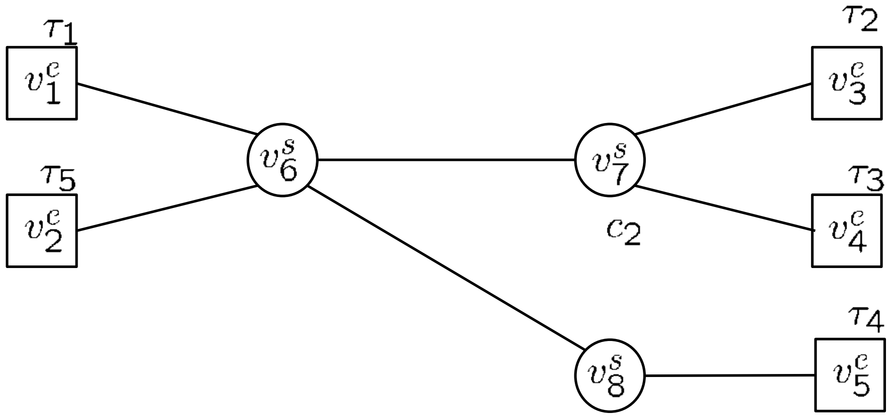
- **Ethernet-based time-triggered systems**
  - Processor: time-triggered non-preemptive task scheduling
  - Network: time-triggered Ethernet communication

Problem Formulation

- **The scheduling problem**
  - Application task

$$\tau_i = \{\tau_i.p, \quad \tau_i.o, \quad \tau_i.e\}$$

↓ *period* ↓ *offset* ↓ *WCET*



Problem Formulation

- **The scheduling problem**
  - Application task

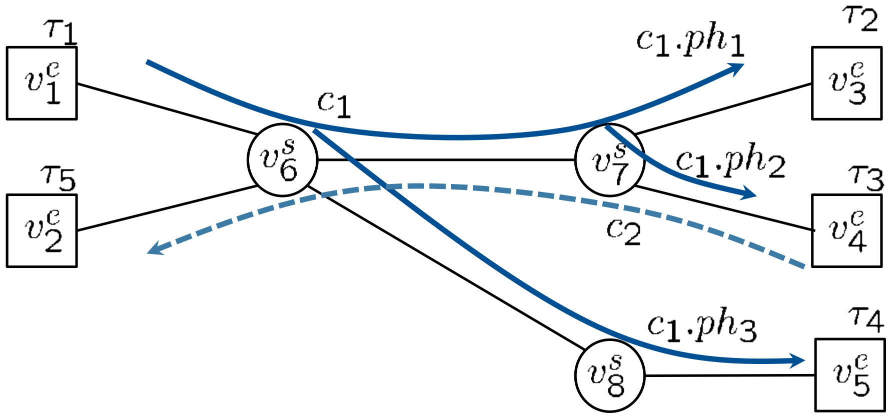
$$\tau_i = \{\tau_i.p, \quad \tau_i.o, \quad \tau_i.e\}$$

↓ *period* ↓ *offset* ↓ *WCET*

  - Communication task

$$c_i = \{f_i, \quad c_i.tr, \quad c_i.o, \quad c_i.p\}$$

↓ *frame length* ↓ *path tree* ↓ *offsets* ↓ *period*



Problem Formulation

- **The scheduling problem**
  - Application task

$$\tau_i = \{\tau_i.p, \quad \tau_i.o, \quad \tau_i.e\}$$

period *offset* *WCET*

  - Communication task

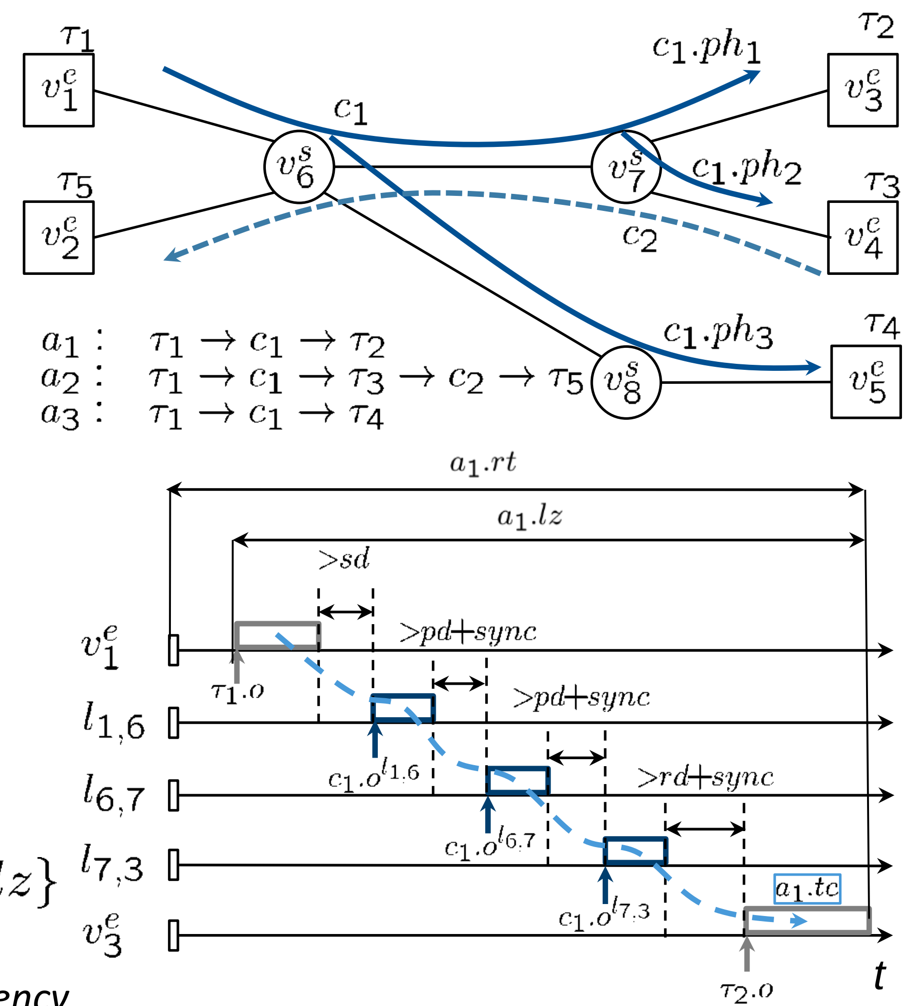
$$c_i = \{f_i, \quad c_i.tr, \quad c_i.o, \quad c_i.p\}$$

frame length *path tree* *offsets* *period*

  - Application

$$a_i = \{a_i.tc, \quad a_i.p, \quad a_i.rt, \quad a_i.lz\}$$

period *task chain* *response time* *latency*



Problem Formulation

- **The scheduling problem**
  - Hardware specific parameters $hw.\omega$
    - System topology
    - Timing parameters including network bandwidth, synchronization precision, etc.
  - Application parameters $\mathcal{A}.\omega$
    - Task mapping, period, WCET
    - Communication frame length, path tree, latency and response time constraints
  - Application schedules $\mathcal{A}.\mathbf{o}$
    - Task schedules and frame transmission schedules on each link

Problem Formulation

- **The scheduling problem**
  - Hardware specific parameters $hw.\omega$
    - System topology
    - Timing parameters including network bandwidth, synchronization precision, etc.
  - Application parameters $\mathcal{A}.\omega$
    - Task mapping, period, WCET
    - Communication frame length, path tree, latency and response time constraints
  - Application schedules $\mathcal{A}.\mathbf{o}$
    - Task schedules and frame transmission schedules on each link
- **Approach**
  - Formulation of the problem in SMT or MIP problem and use solvers to obtain the schedules, as in [10,11,13,15,16]

Problem Formulation

- **The schedule management problem**
  - Consider a system with $hw.\omega$ and existing application set $\mathcal{A}_o.\omega$, $\mathcal{A}_o.\mathbf{o}$
  - Obtain $\mathcal{A}_n.\mathbf{o}$ for the new application set $\mathcal{A}_n$ with $\mathcal{A}_n.\omega$, while addressing the requirements:
    - Obtain schedules in relatively short time
    - As many as possible new applications can be accommodated
    - Facilitation of schedule reuse and minimization of disturbance to existing schedules

Problem Formulation

- **The schedule management problem**
  - Consider a system with $hw.\omega$ and existing application set $\mathcal{A}_o.\omega$, $\mathcal{A}_o.\mathbf{o}$
  - Obtain $\mathcal{A}_n.\mathbf{o}$ for the new application set $\mathcal{A}_n$ with $\mathcal{A}_n.\omega$, while addressing the requirements:
    - Obtain schedules in relatively short time
    - As many as possible new applications can be accommodated
    - Facilitation of schedule reuse and minimization of disturbance to existing schedules

- **Alternatives and Challenges**
  - Synthesize schedules offline for a specific application -> conflicts with existing schedules
  - Synthesize all possible schedule sets offline -> possibly a huge number combinations
  - Online schedule synthesis on-board -> long synthesis time due to limited computing power

Problem Formulation

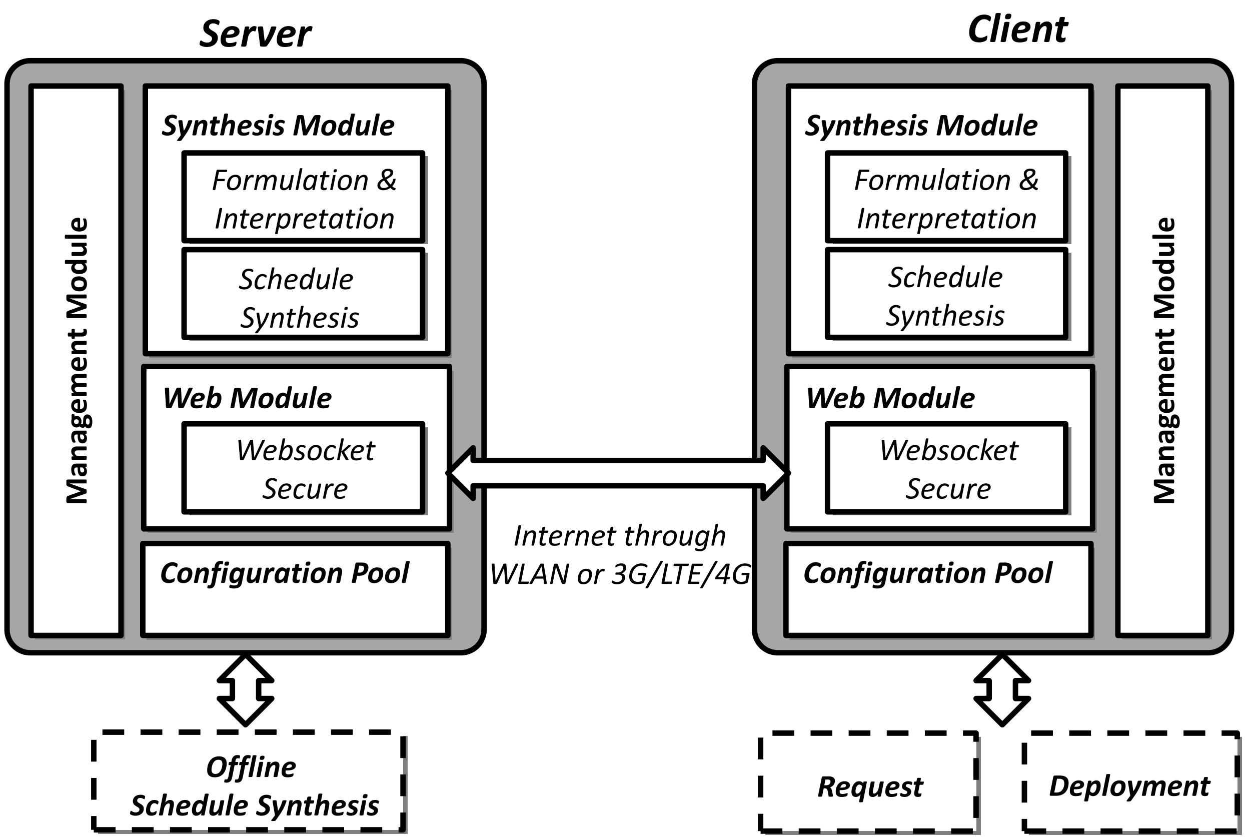
- **The schedule management problem**
  - Consider a system with $hw.\omega$ and existing application set $\mathcal{A}_o.\omega$, $\mathcal{A}_o.\mathbf{o}$
  - Obtain $\mathcal{A}_n.\mathbf{o}$ for the new application set $\mathcal{A}_n$ with $\mathcal{A}_n.\omega$, while addressing the requirements:
    - Obtain schedules in relatively short time
    - As many as possible new applications can be accommodated
    - Facilitation of schedule reuse and minimization of disturbance to existing schedules

- **Alternatives and challenges**
  - Synthesize schedules offline for a specific application -> conflicts with existing schedules
  - Synthesize all possible schedule sets offline -> possibly a huge number combinations
  - Online schedule synthesis on-board -> long synthesis time due to limited computing power

➢ **Need of an online schedule management framework**

Schedule Management Framework

- **Overview**
  - Client-server architecture
  - Utilization of both onboard processor and cloud-computing
  - Components: Synthesis Module, Web Module, Configuration Pool, Management Module



Schedule Management Framework

- **Configuration and request**
  - TTCON
    - An XML format containing $hw.\omega$, $\mathcal{A}.\omega$ and $\mathcal{A}.\mathbf{o}$
    - Configuration: all application schedules have valid values
    - Request: some application schedules are empty

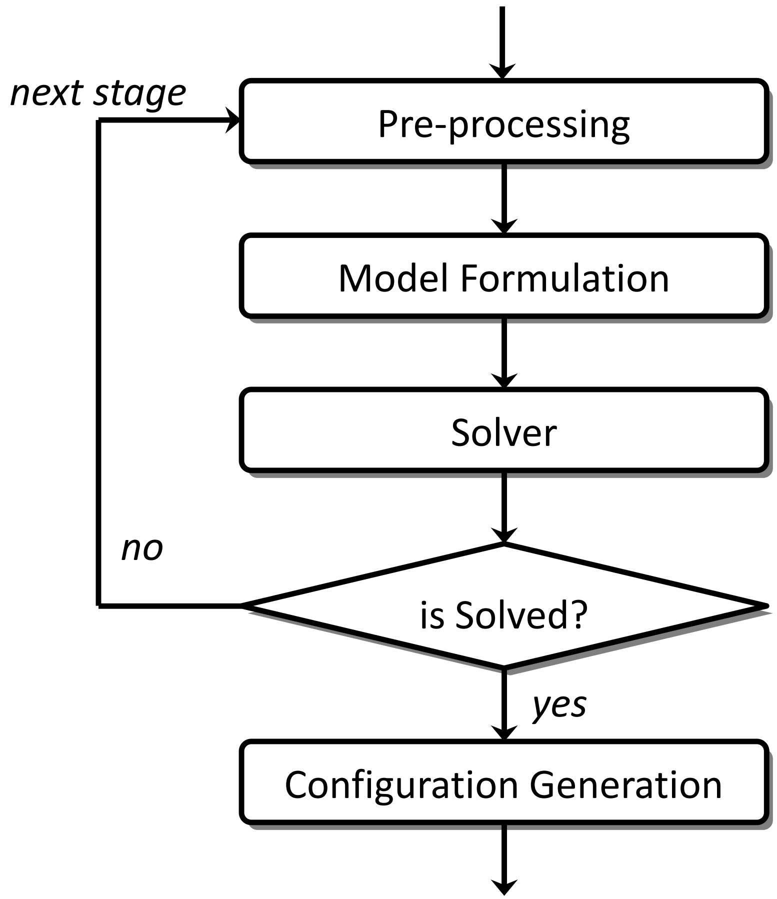
Schedule Management Framework

- **Configuration and request**
  - TTCON
    - An XML format containing $hw.\omega$, $\mathcal{A}.\omega$ and $\mathcal{A}.\mathbf{o}$
    - Configuration: all application schedules have valid values
    - Request: some application schedules are empty

- **Configuration Pool**
  - Can be managed through different metrics like frequency of reuse
  - Retrieve a configuration
    - If the application set of request matches exactly or is a subset of a configuration in pool, the configuration can be retrieved
    - In the case of a subset, schedules of other applications are removed
  - Update the configuration pool
    - Add a configuration if it is not in pool
    - If the new configuration is a superset of an existing one, it replaces the existing one
  - It facilitates schedule reuse for a single vehicle or between vehicles of the same variant and request based configuration management

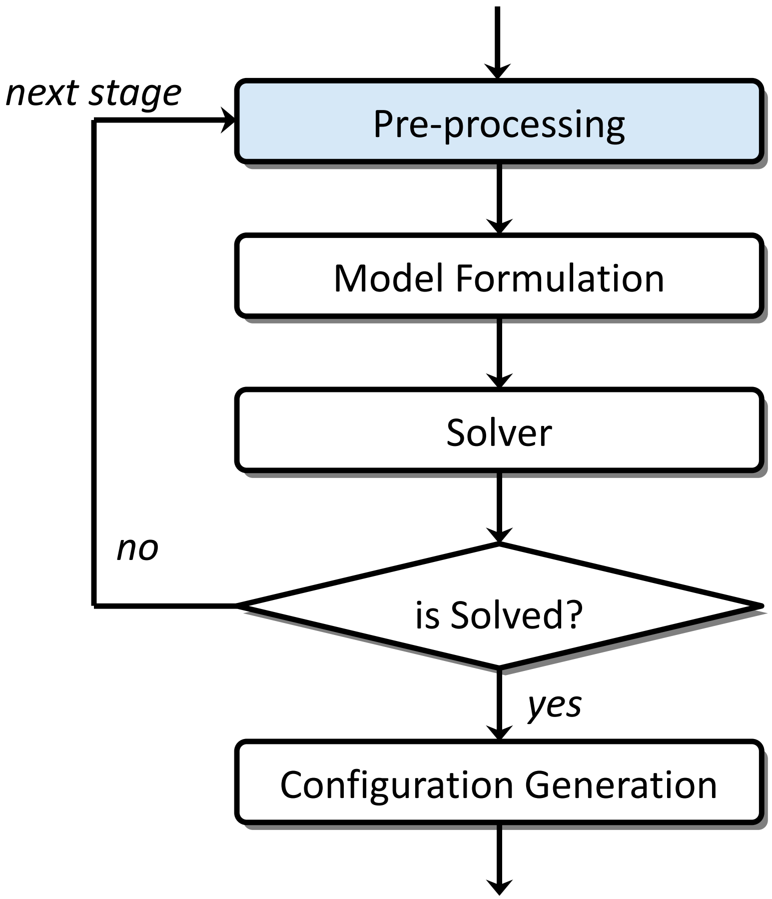
Schedule Management Framework

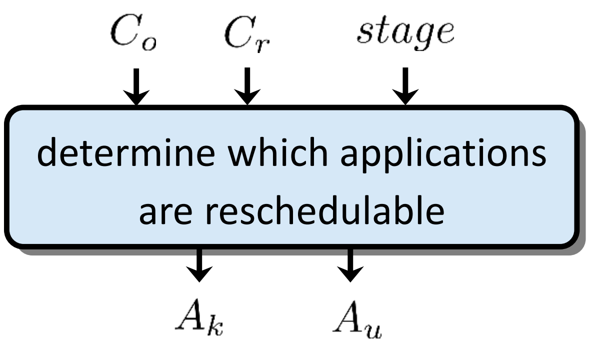
- **Synthesis module**



Schedule Management Framework

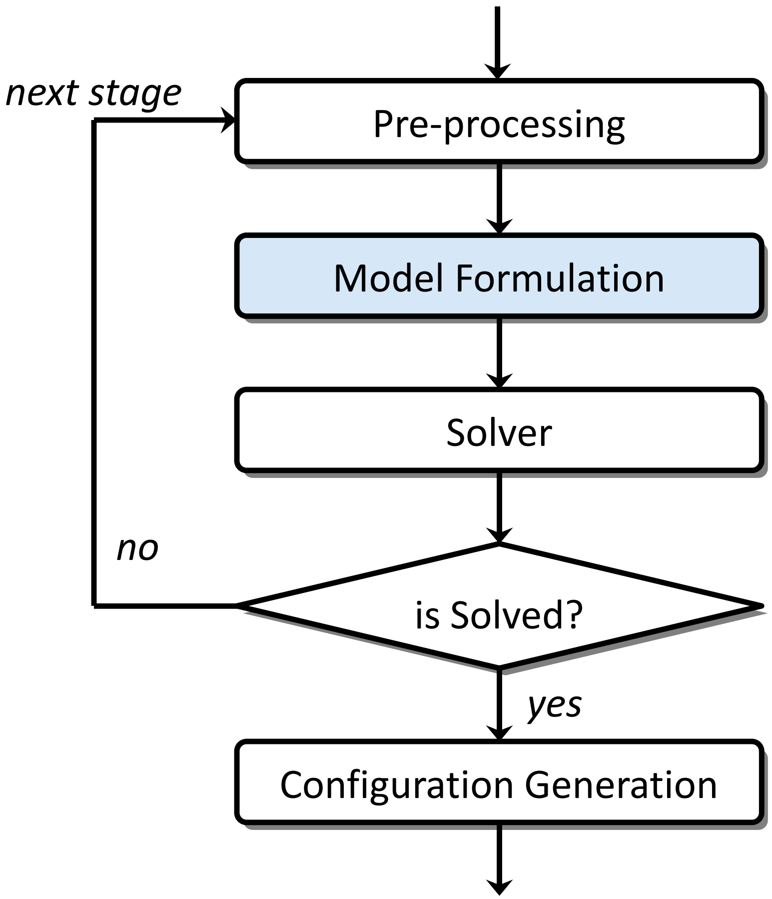
- **Synthesis module**

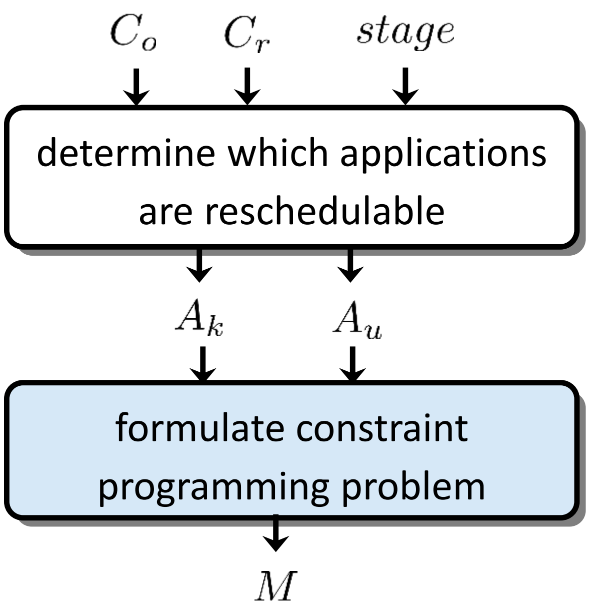




Schedule Management Framework

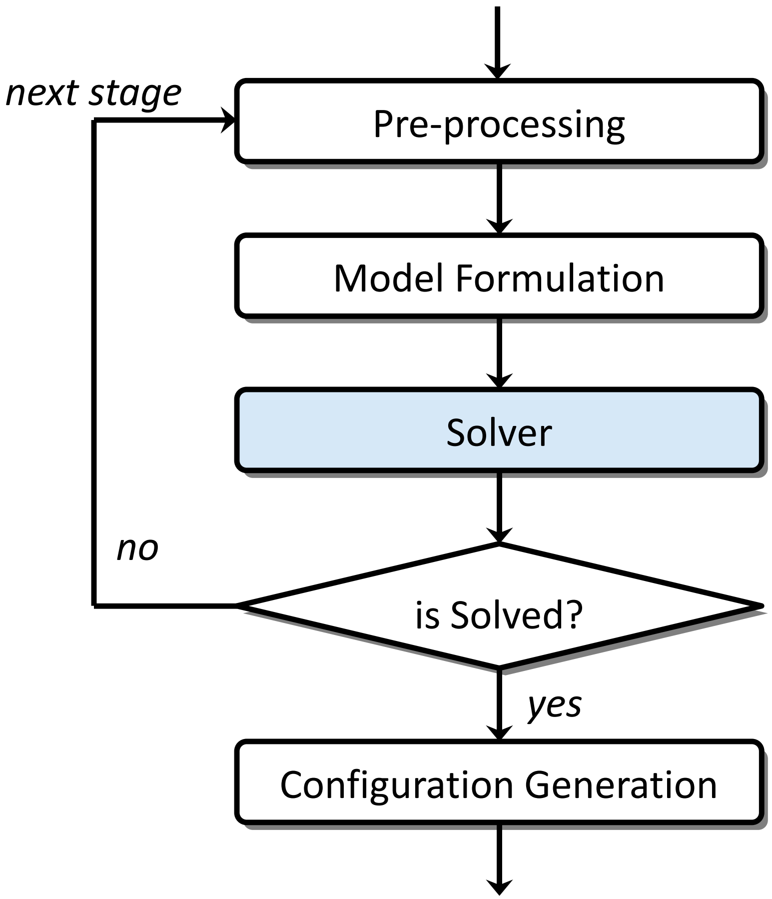
- **Synthesis module**

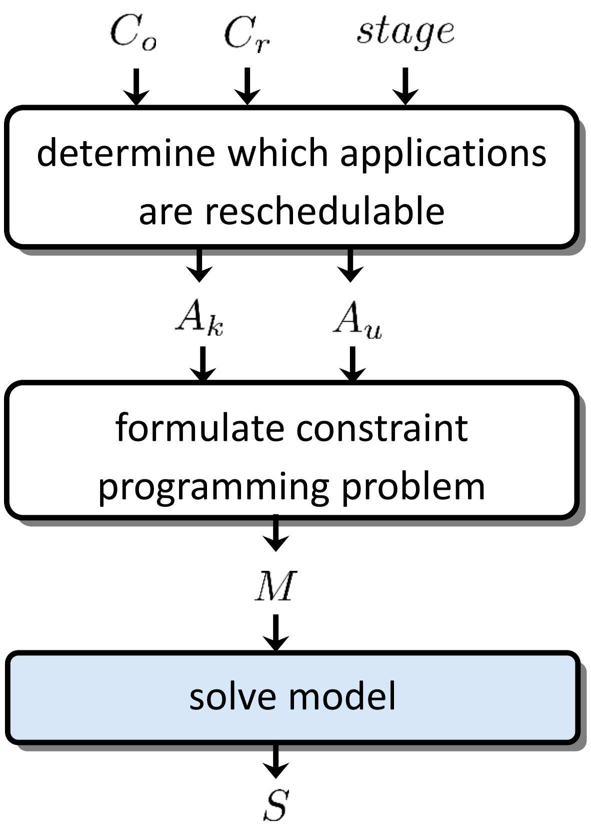




Schedule Management Framework

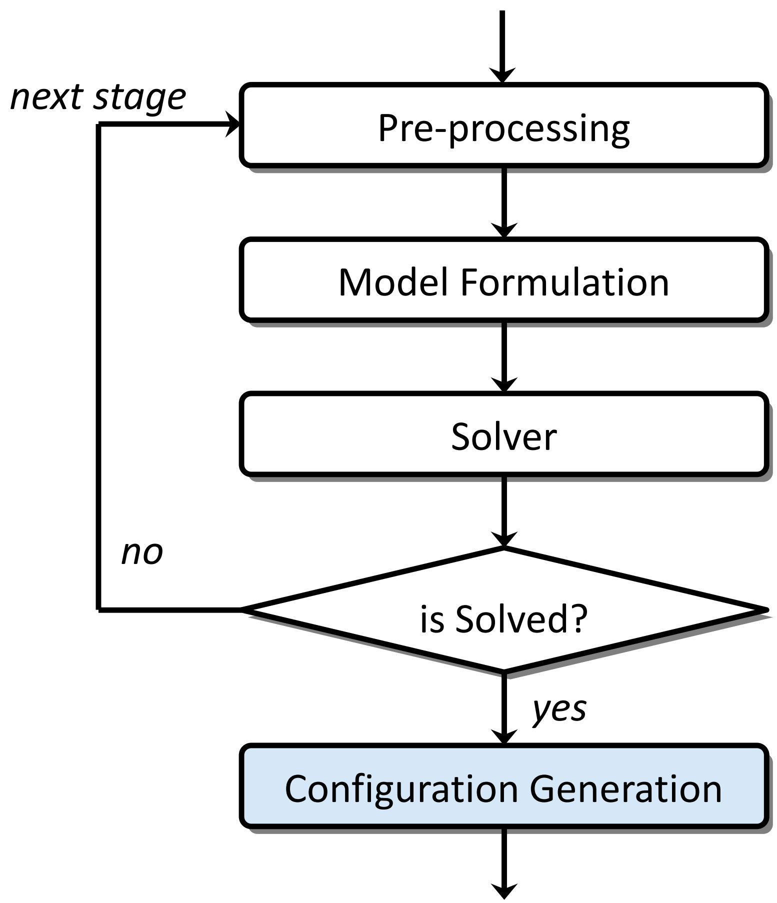
- **Synthesis module**

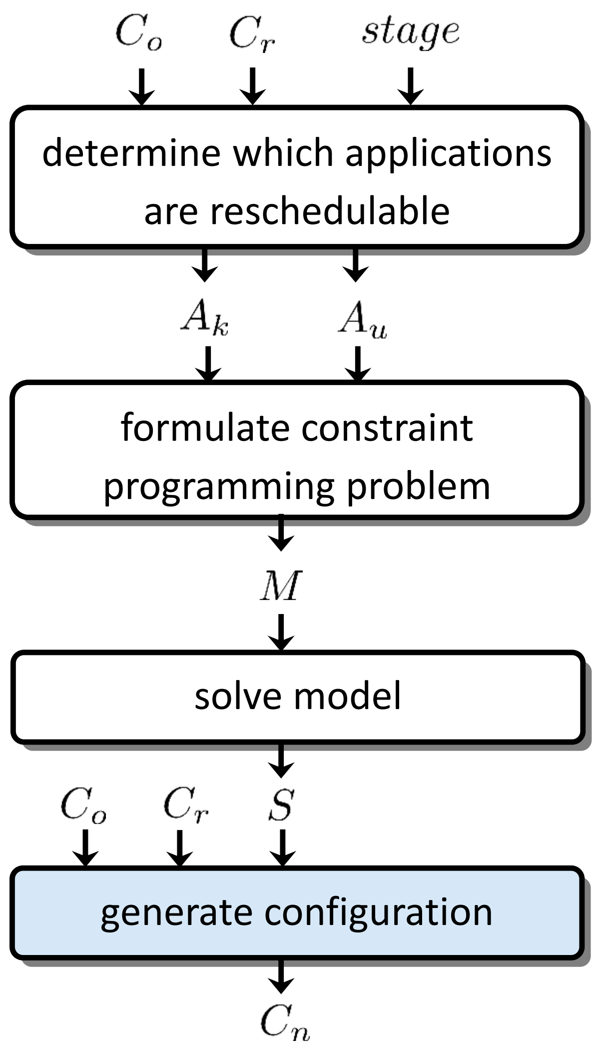




Schedule Management Framework

- **Synthesis module**





Schedule Management Framework

- **Four-stage scheduling strategy**
  - Stage 1 – Incremental scheduling
    - None of the existing applications are rescheduled

      $$\mathcal{A}_k = \mathcal{A}_o \cap \mathcal{A}_n, \mathcal{A}_u = \mathcal{A}_n \backslash \mathcal{A}_k$$

Schedule Management Framework

- **Four-stage scheduling strategy**
  - Stage 1 – Incremental scheduling
    - ➢ None of the existing applications are rescheduled

$$\mathcal{A}_k = \mathcal{A}_o \cap \mathcal{A}_n, \mathcal{A}_u = \mathcal{A}_n \backslash \mathcal{A}_k$$

  - Stage 2 – Rescheduling based on task conflict
    - ➢ Reschedule existing applications with common tasks with new ones

$$\mathcal{A}_k = (\mathcal{A}_o \cap \mathcal{A}_n) \backslash \mathcal{A}_\tau, \mathcal{A}_u = \mathcal{A}_n \backslash \mathcal{A}_k$$

$$\mathcal{A}_\tau = \{a_i | a_i \in \mathcal{A}_o^a \cap \mathcal{A}_n^a \wedge \underset{a_j \in A_n^a \backslash A_o^a}{\exists} \ \underset{\substack{\tau_k \in a_i.tc \\ \tau_l \in a_j.tc}}{\exists} \tau_k = \tau_l\}$$

Schedule Management Framework

- **Four-stage scheduling strategy**
  - Stage 1 – Incremental scheduling
    - None of the existing applications are rescheduled

$$\mathcal{A}_k = \mathcal{A}_o \cap \mathcal{A}_n, \mathcal{A}_u = \mathcal{A}_n \backslash \mathcal{A}_k$$

  - Stage 2 – Rescheduling based on task conflict
    - Reschedule existing applications with common tasks with new ones

$$\mathcal{A}_k = (\mathcal{A}_o \cap \mathcal{A}_n) \backslash \mathcal{A}_\tau, \mathcal{A}_u = \mathcal{A}_n \backslash \mathcal{A}_k$$

$$\mathcal{A}_\tau = \{a_i | a_i \in \mathcal{A}_o^a \cap \mathcal{A}_n^a \wedge \exists_{a_j \in A_n^a \backslash A_o^a} \exists_{\substack{\tau_k \in a_i.tc \\ \tau_l \in a_j.tc}} \tau_k = \tau_l\}$$

  - Stage 3 – Rescheduling based on computation resource conflict
    - Reschedule existing applications with tasks mapped on common ECU with new ones

$$\mathcal{A}_k = (\mathcal{A}_o \cap \mathcal{A}_n) \backslash \mathcal{A}_E, \mathcal{A}_u = \mathcal{A}_n \backslash \mathcal{A}_k$$

$$\mathcal{A}_E = \{a_i | a_i \in \mathcal{A}_o^a \cap \mathcal{A}_n^a \wedge \exists_{a_j \in A_n^a \backslash A_o^a} \exists_{\substack{\tau_k \in a_i.tc \\ \tau_l \in a_j.tc}} \tau_k.E = \tau_l.E\}$$

Schedule Management Framework

- **Four-stage scheduling strategy**
  - Stage 1 – Incremental scheduling
    - None of the existing applications are rescheduled

      $$\mathcal{A}_k = \mathcal{A}_o \cap \mathcal{A}_n, \mathcal{A}_u = \mathcal{A}_n \backslash \mathcal{A}_k$$
  - Stage 2 – Rescheduling based on task conflict
    - Reschedule existing applications with common tasks with new ones

      $$\mathcal{A}_k = (\mathcal{A}_o \cap \mathcal{A}_n) \backslash \mathcal{A}_\tau, \mathcal{A}_u = \mathcal{A}_n \backslash \mathcal{A}_k$$

      $$\mathcal{A}_\tau = \{a_i | a_i \in \mathcal{A}_o^a \cap \mathcal{A}_n^a \wedge \underset{a_j \in A_n^a \backslash A_o^a}{\exists} \underset{\substack{\tau_k \in a_i.tc \\ \tau_l \in a_j.tc}}{\exists} \tau_k = \tau_l\}$$
  - Stage 3 – Rescheduling based on computation resource conflict
    - Reschedule existing applications with tasks mapped on common ECU with new ones

      $$\mathcal{A}_k = (\mathcal{A}_o \cap \mathcal{A}_n) \backslash \mathcal{A}_E, \mathcal{A}_u = \mathcal{A}_n \backslash \mathcal{A}_k$$

      $$\mathcal{A}_E = \{a_i | a_i \in \mathcal{A}_o^a \cap \mathcal{A}_n^a \wedge \underset{a_j \in A_n^a \backslash A_o^a}{\exists} \underset{\substack{\tau_k \in a_i.tc \\ \tau_l \in a_j.tc}}{\exists} \tau_k.E = \tau_l.E\}$$
  - Stage 4 – Complete rescheduling
    - All existing applications are considered reschedulable

      $$\mathcal{A}_k = \mathcal{A}^b, \mathcal{A}_n = \mathcal{A}_n^a$$

Schedule Management Framework

- **Four-stage scheduling strategy**
  - Stage 1 – Incremental scheduling
    - ➢ None of the existing applications are rescheduled

      $\mathcal{A}_k = \mathcal{A}_o \cap \mathcal{A}_n, \mathcal{A}_u = \mathcal{A}_n \backslash \mathcal{A}_k$
  - Stage 2 – Rescheduling based on task conflict
    - ➢ Reschedule existing applications with common tasks with new ones

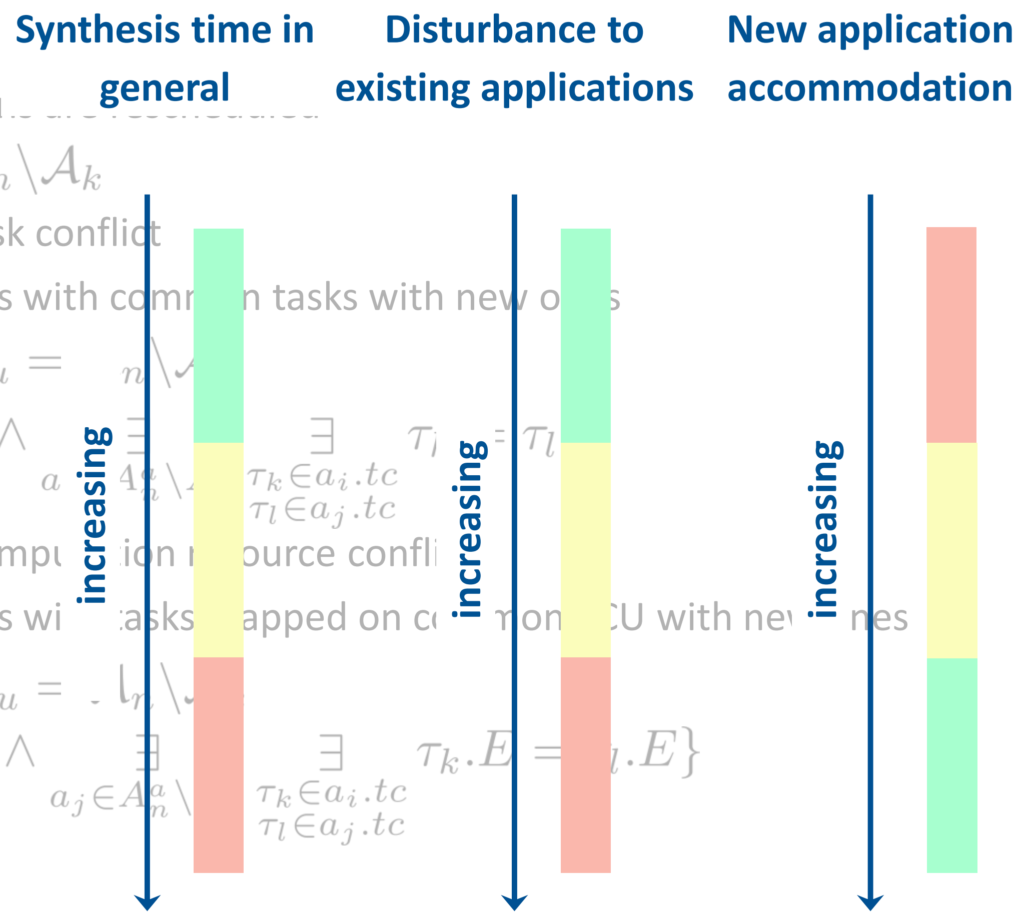
      $\mathcal{A}_k = (\mathcal{A}_o \cap \mathcal{A}_n) \backslash \mathcal{A}_\tau, \mathcal{A}_u = \mathcal{A}_n \backslash \mathcal{A}_k$

      $\mathcal{A}_\tau = \{a_i | a_i \in \mathcal{A}_o^a \cap \mathcal{A}_n^a \wedge \exists_{a_j \in \mathcal{A}_n^a \backslash \mathcal{A}_o^a} \exists_{\tau_k \in a_i.tc, \tau_l \in a_j.tc} \tau_k = \tau_l\}$
  - Stage 3 – Rescheduling based on computation resource conflict
    - ➢ Reschedule existing applications with tasks mapped on common ECU with new ones

      $\mathcal{A}_k = (\mathcal{A}_o \cap \mathcal{A}_n) \backslash \mathcal{A}_E, \mathcal{A}_u = \mathcal{A}_n \backslash \mathcal{A}_k$

      $\mathcal{A}_E = \{a_i | a_i \in \mathcal{A}_o^a \cap \mathcal{A}_n^a \wedge \exists_{a_j \in \mathcal{A}_n^a \backslash \mathcal{A}_o^a} \exists_{\tau_k \in a_i.tc, \tau_l \in a_j.tc} \tau_k.E = \tau_l.E\}$
  - Stage 4 – Complete rescheduling
    - ➢ All existing applications are considered reschedulable

      $\mathcal{A}_k = \mathcal{A}^b, \mathcal{A}_n = \mathcal{A}_n^a$



Schedule Management Framework

- **Web module**
  - Utilizes the Websocket Secure
    - Full-duplex communication between client and server
    - SSL/TLS layer for secure communication

Schedule Management Framework

- **Web module**
  - Utilizes the Websocket Secure
    - Full-duplex communication between client and server
    - SSL/TLS layer for secure communication

  - Methods for client-server communication
    - Request
    - ➢ Client sends request file to server

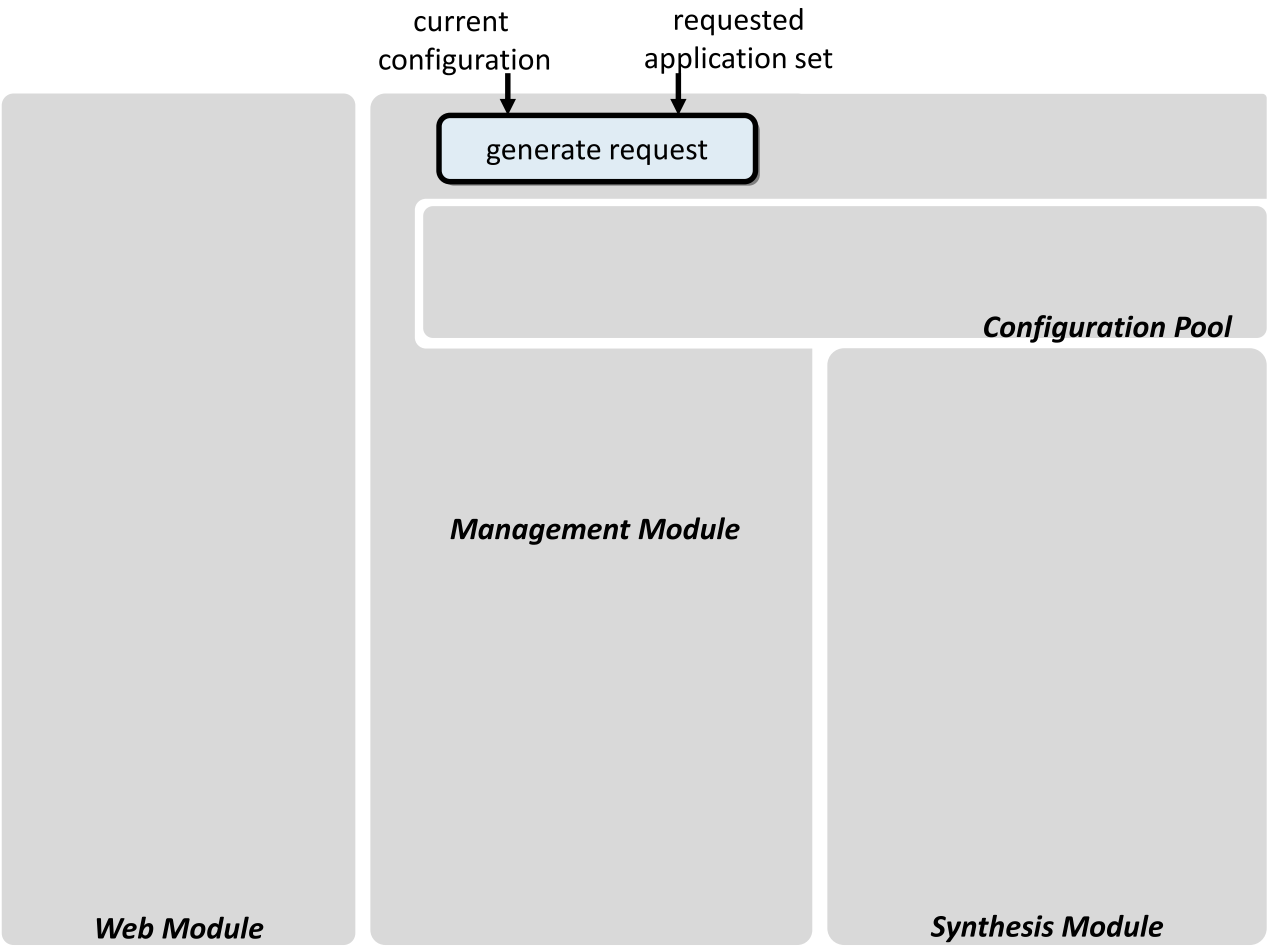
    - Response
    - ➢ Server sends response to client: either a valid configuration or a request denial

    - Abort
    - ➢ Client informs the server to abort operation, when a local result is obtained first

    - Update
    - ➢ Client sends the new configuration to the server to update the configuration pool

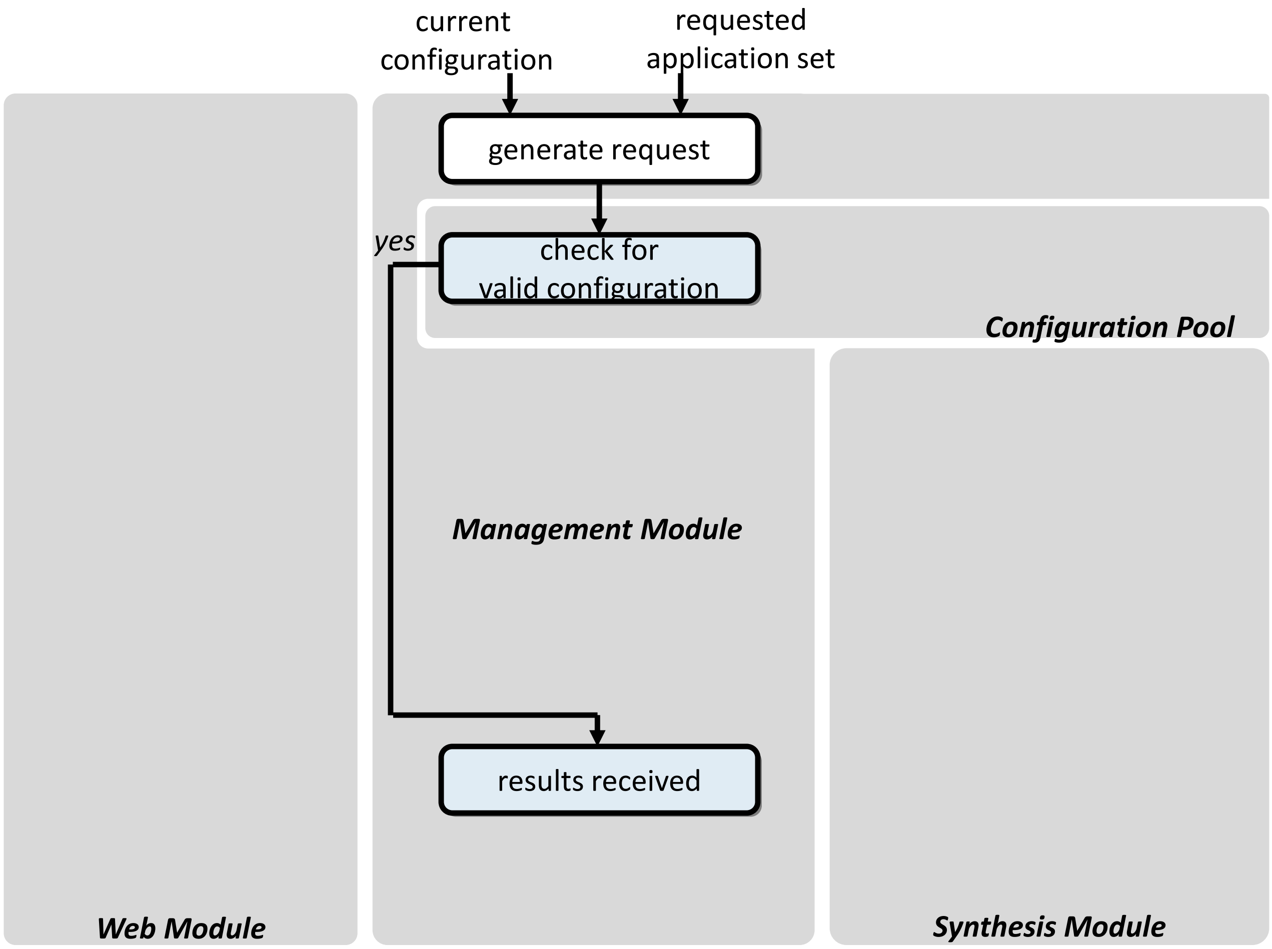
Schedule Management Framework

- **Management module – client side**



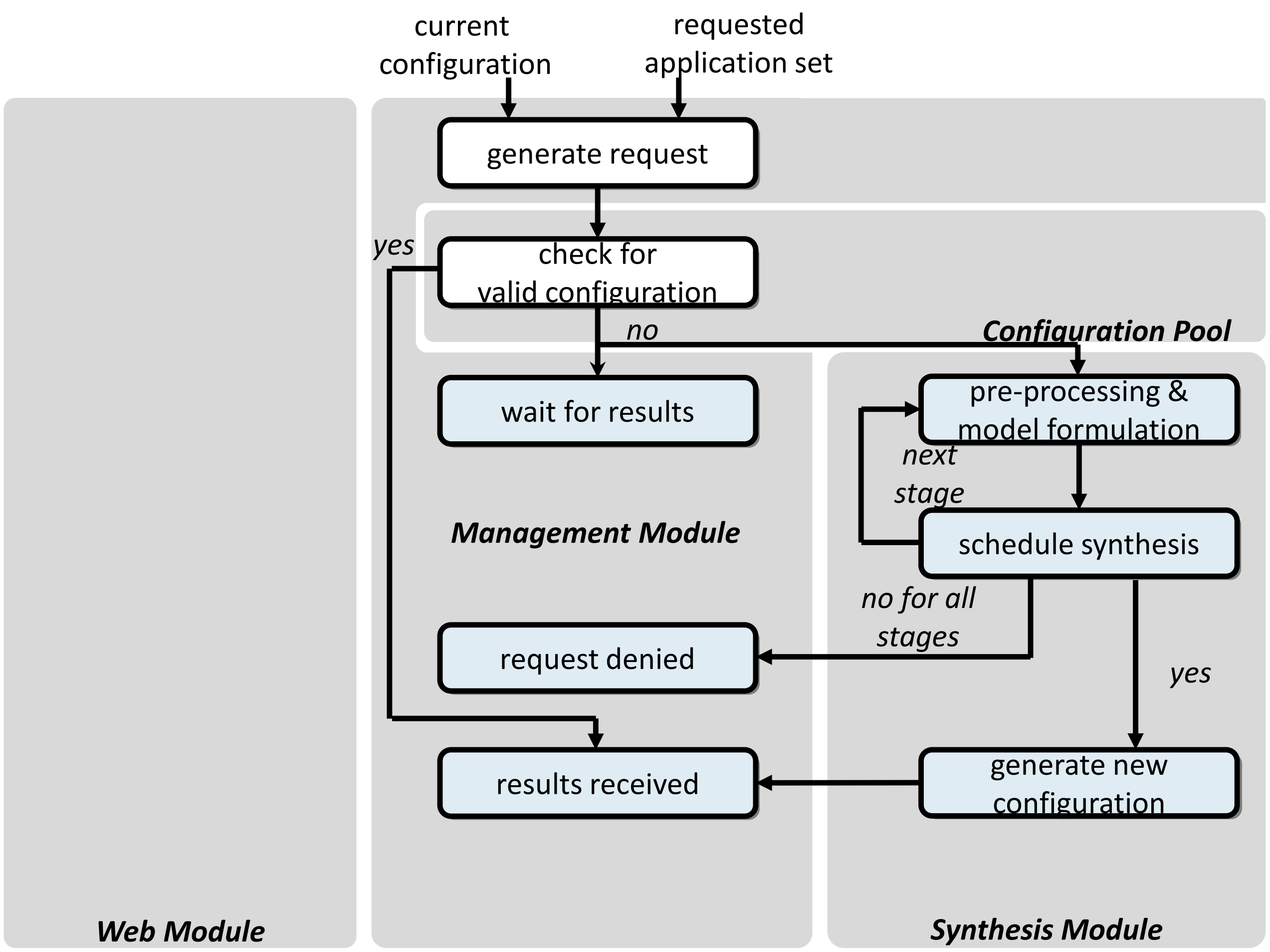
Schedule Management Framework

- **Management module – client side**



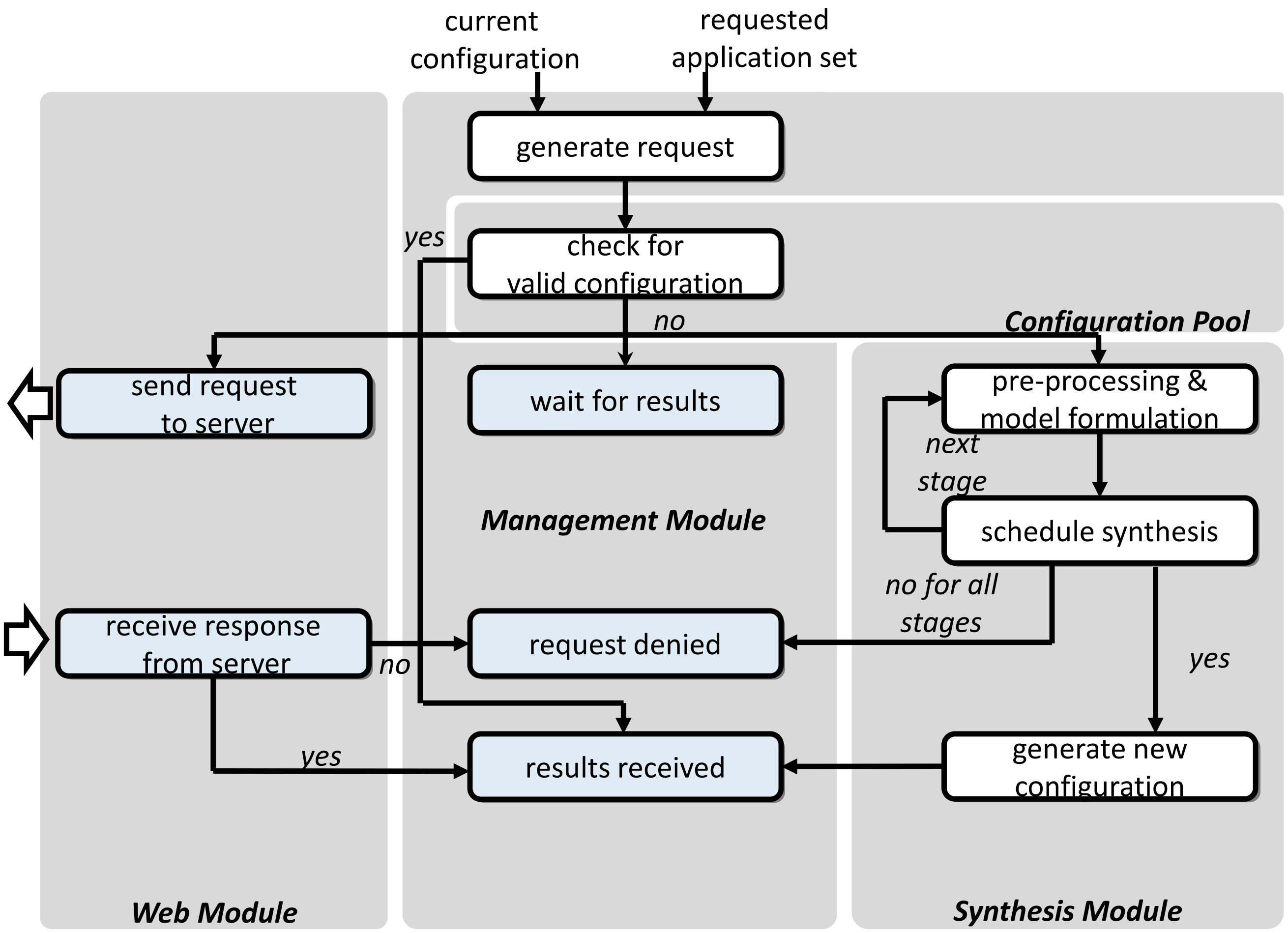
Schedule Management Framework

- **Management module – client side**



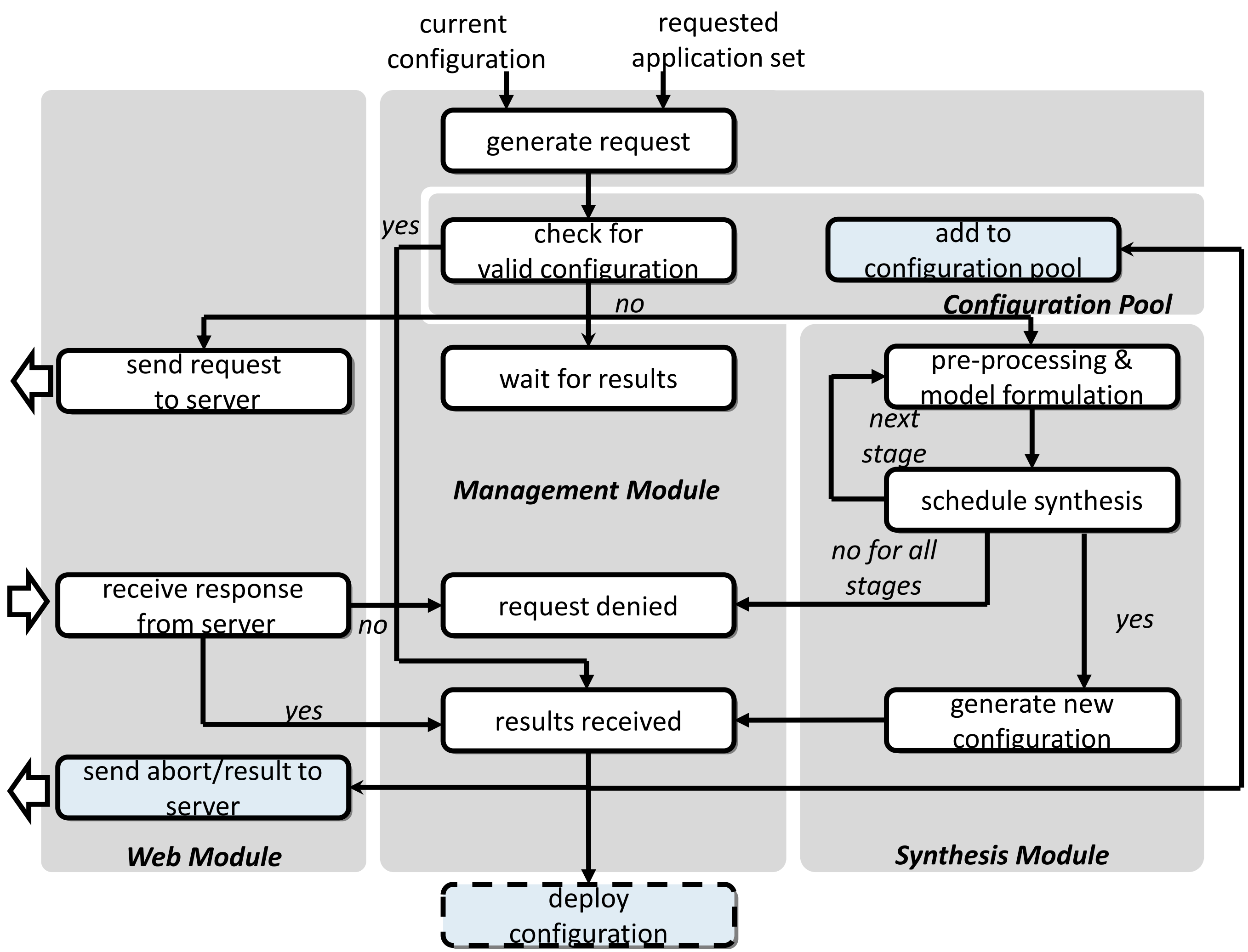
Schedule Management Framework

- **Management module – client side**



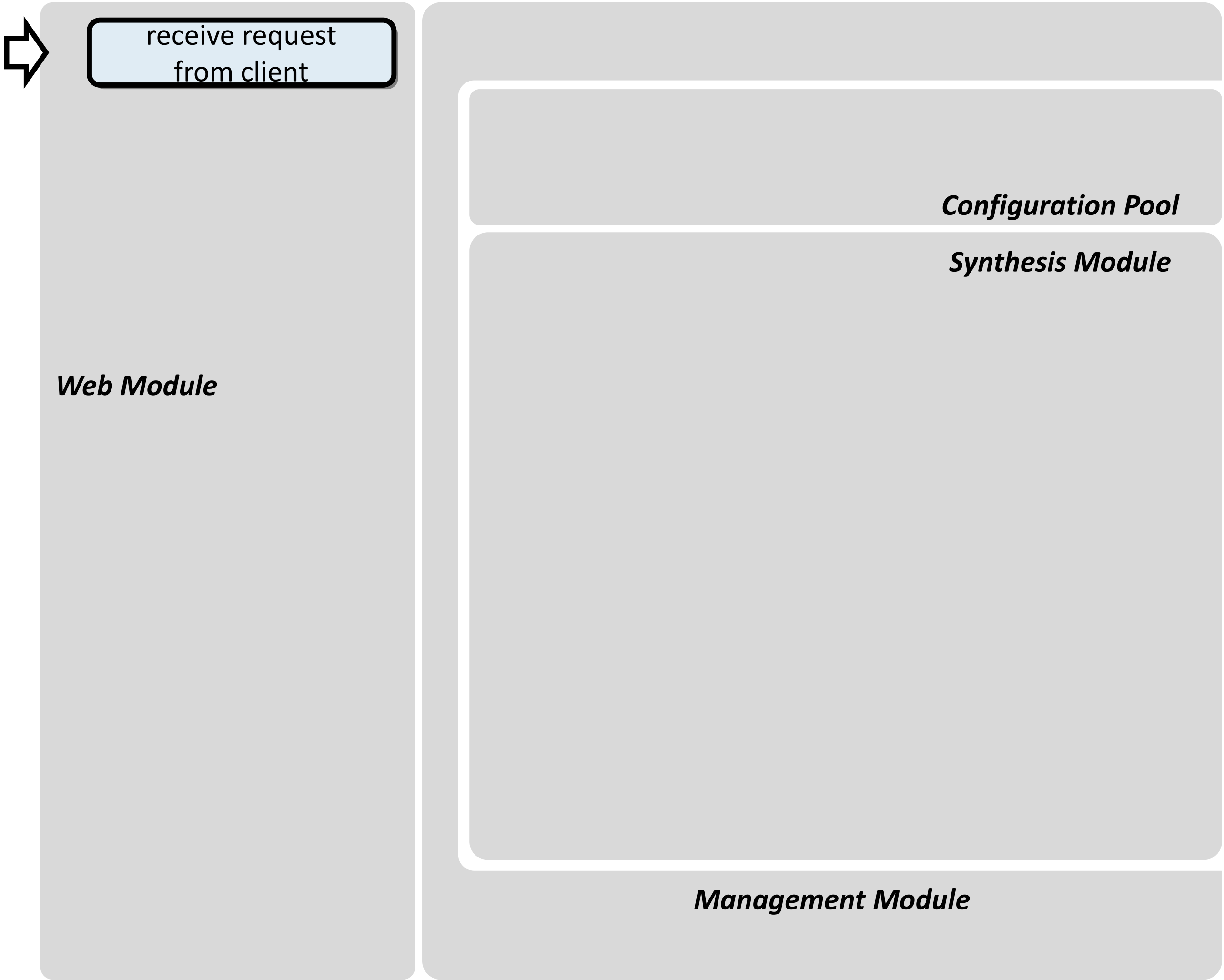
Schedule Management Framework

- **Management module – client side**



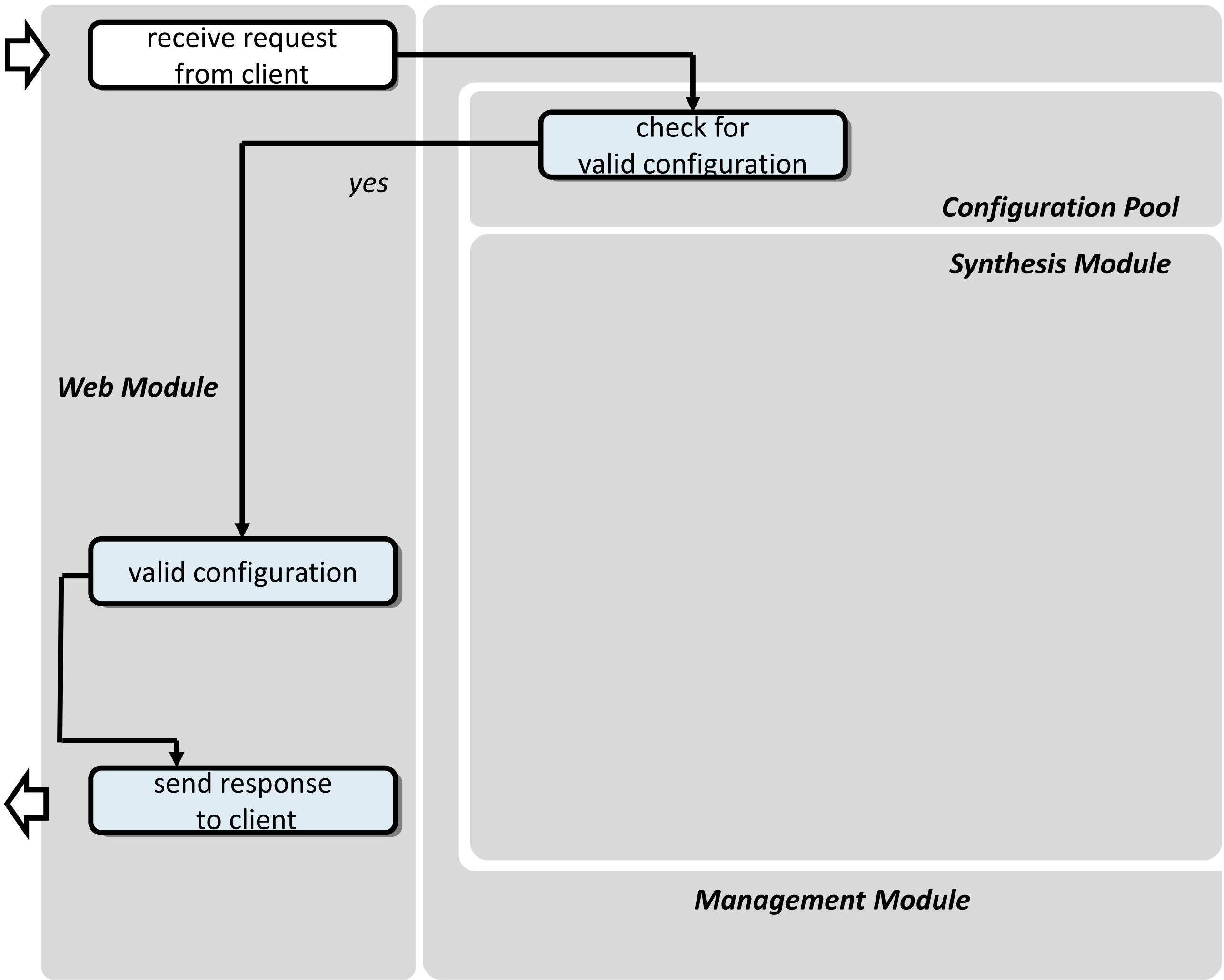
Schedule Management Framework

- **Management module –server side**



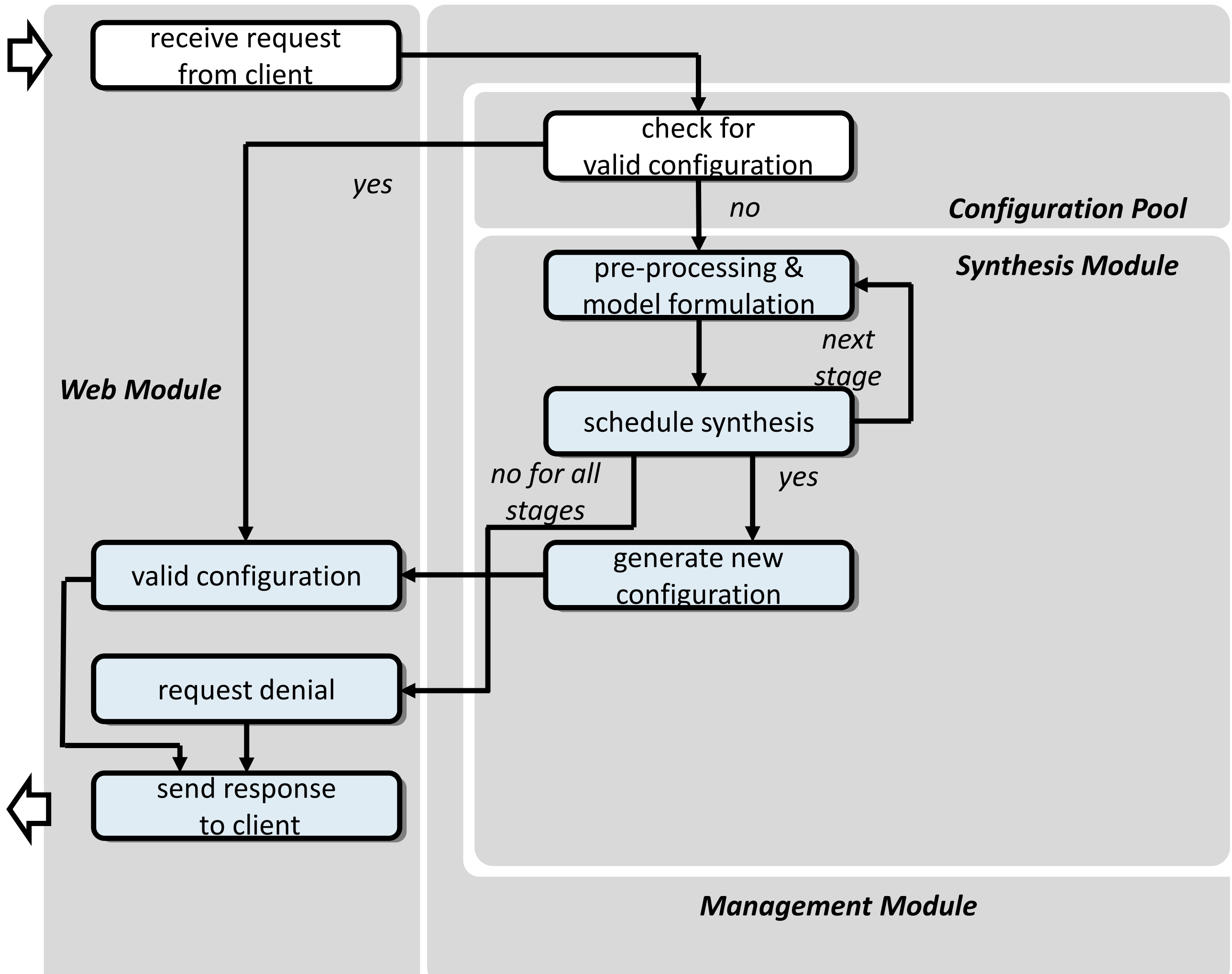
Schedule Management Framework

- **Management module –server side**



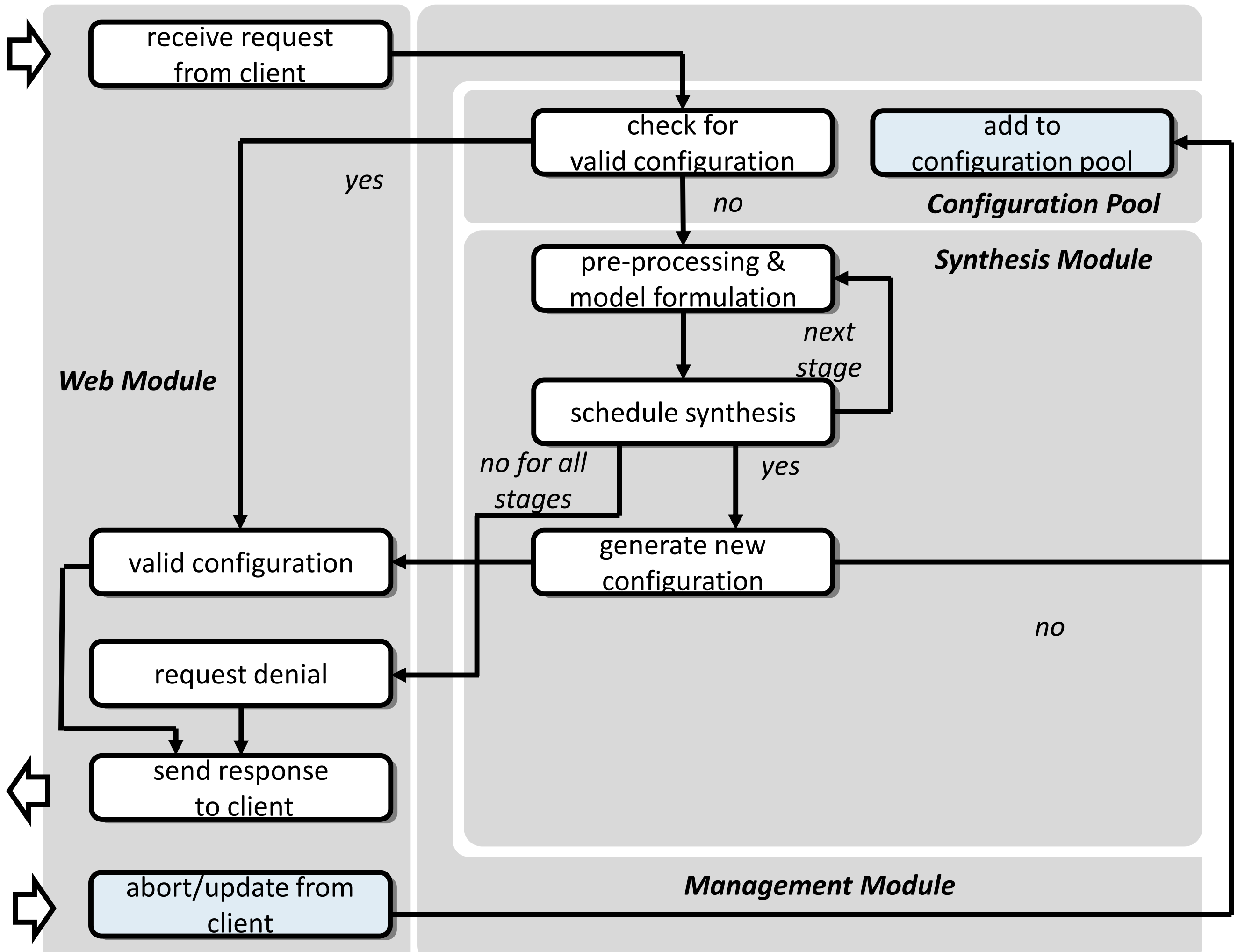
Schedule Management Framework

- **Management module –server side**



Schedule Management Framework

- **Management module –server side**



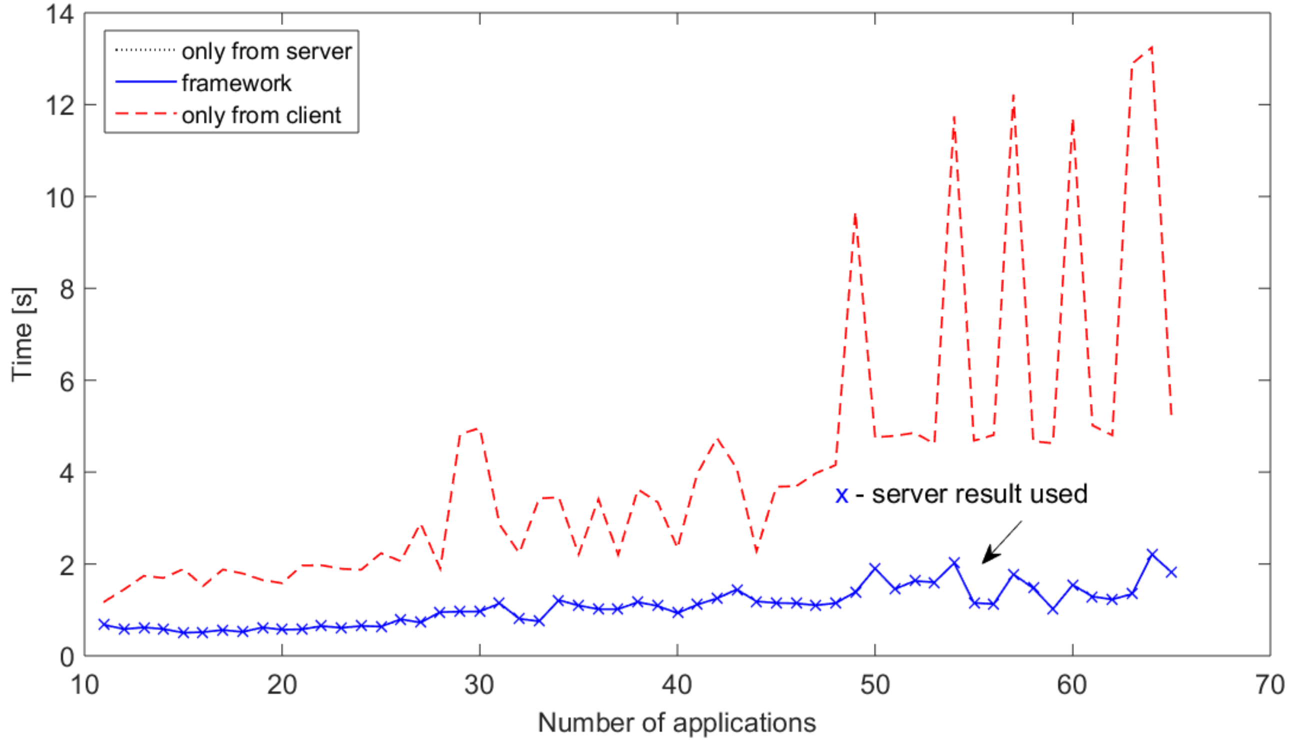
Results

- **Implementation**
  - Client on a Raspberry PI 2 Model B
  - Server on a PC
  - Connection through WLAN

- **Case study**
  - Hardware architecture: 10 ECUs connected by 4 switches
  - 100 applications are randomly generated (10 basic applications, 90 plug-in applications)
  - 20 request series of incrementally adding applications
  - Different overhead provision for possible authentication and security process on server

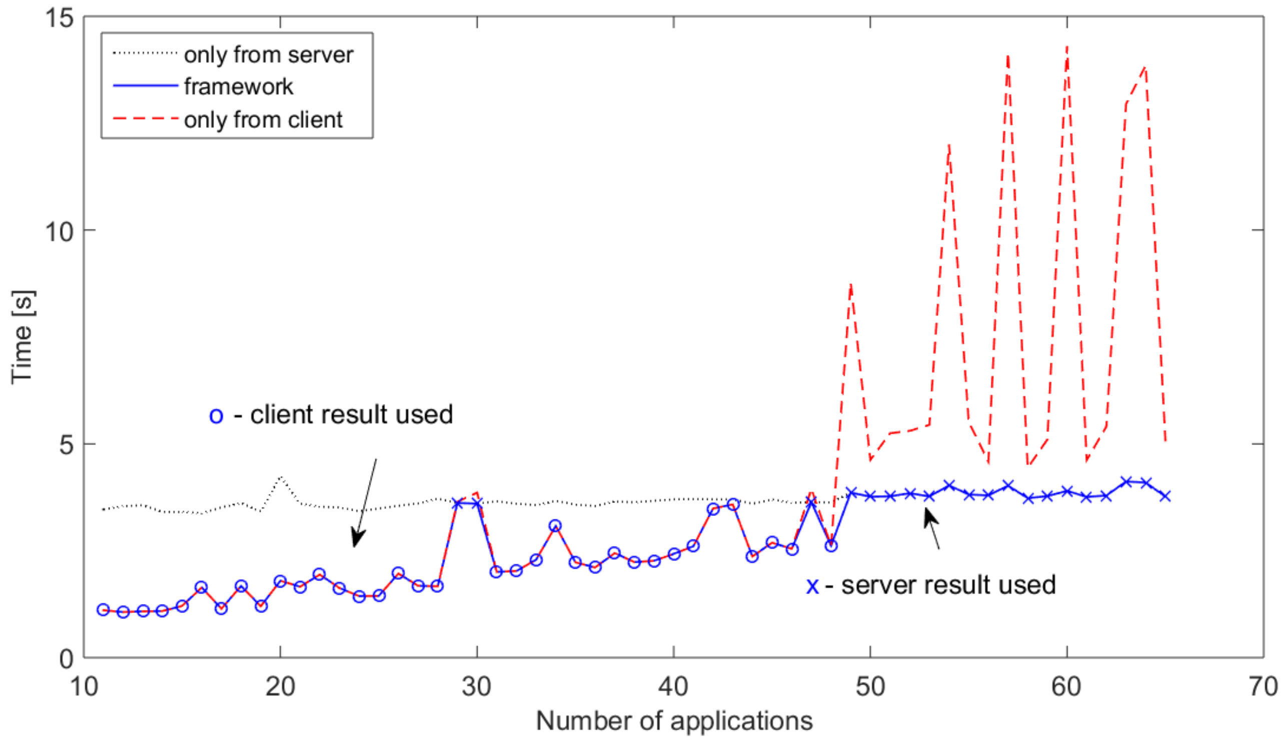
Results

- **Synthesis time**
  - Case 0 s overhead provision for server



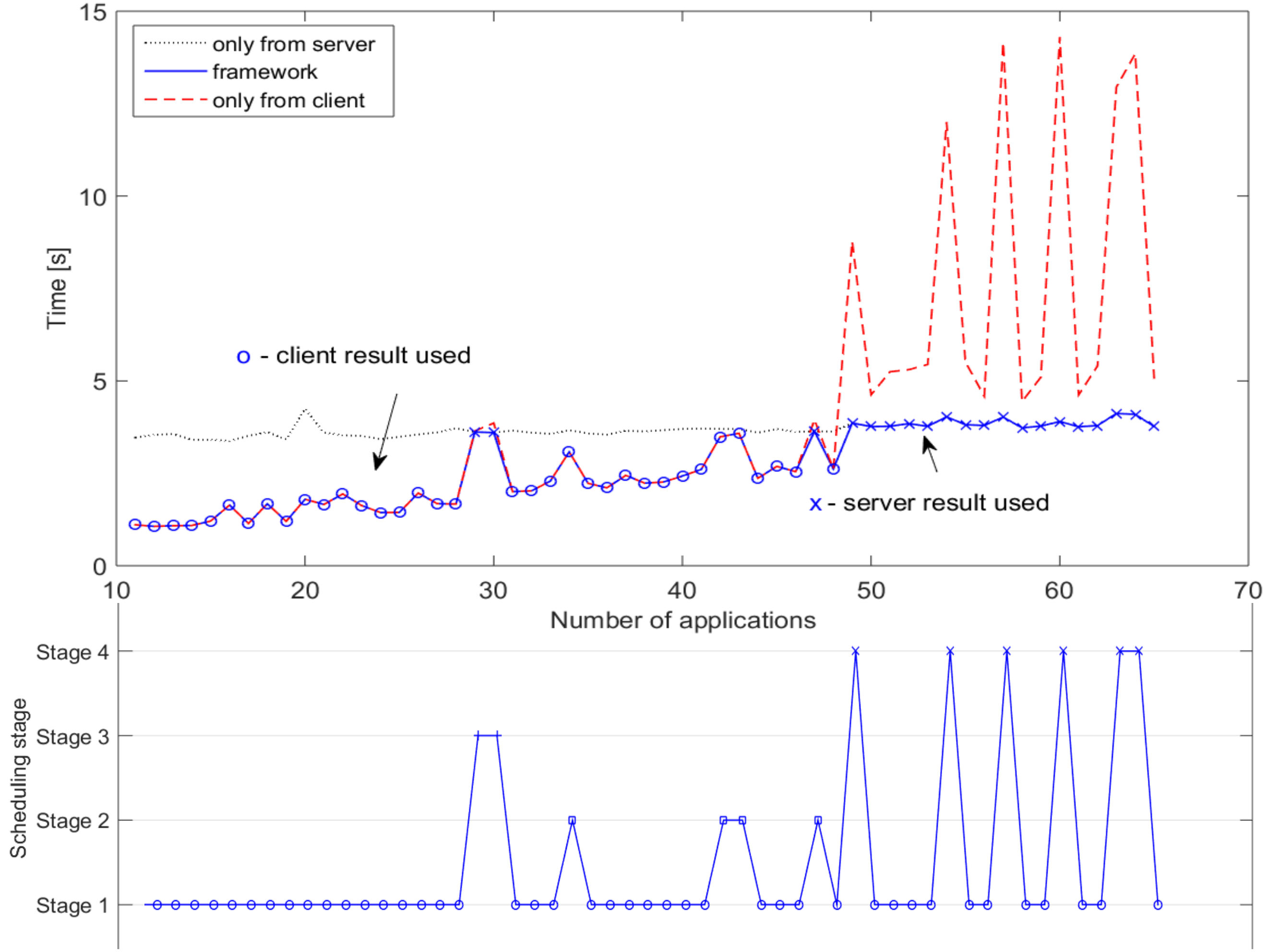
Results

- **Synthesis time**
  - Case 3 s overhead provision for server



Results

- **Scheduling stages**



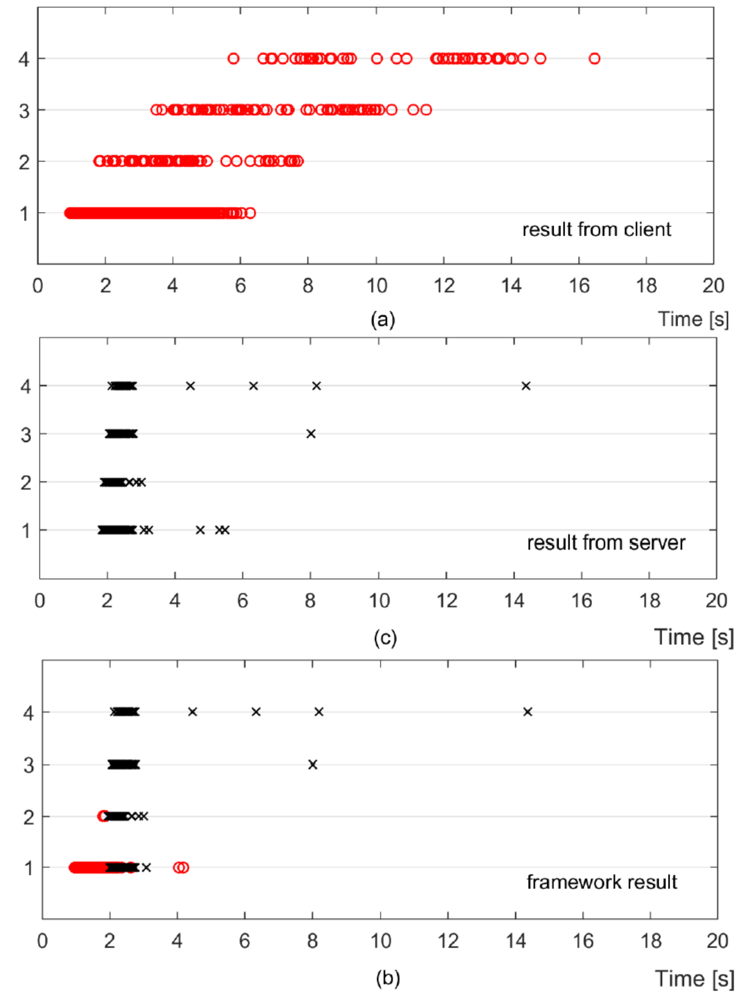
Results

- **Comparison of synthesis time for client, server and proposed framework**
  - Case 1.5 s overhead provision for server

client

server

framework



Concluding Remarks

- **Problem**
  - Ethernet-based time-triggered automotive system
  - Resource reallocation for accommodating new software applications in a Plug-and-Play manner

Concluding Remarks

- **Problem**
  - Ethernet-based time-triggered automotive system
  - Resource reallocation for accommodating new software applications in a Plug-and-Play manner

- **Approach**
  - Client-server based software framework for schedule management
  - Use of local computation and cloud-computing for online schedule synthesis and management
  - Four-stage scheduling strategy for trade-off between synthesis time, disturbance to existing applications and the chances of accommodating new ones

Concluding Remarks

- **Problem**
  - Ethernet-based time-triggered automotive system
  - Resource reallocation for accommodating new software applications in a Plug-and-Play manner
- **Approach**
  - Client-server based software framework for schedule management
  - Use of local computation and cloud-computing for online schedule synthesis and management
  - Four-stage scheduling strategy for trade-off between synthesis time, disturbance to existing applications and the chances of accommodating new ones
- **Future work**
  - Utilize the multi-core architecture to parallelize synthesis methods to reduce synthesis time
  - Explore extensibility-aware scheduling to provision resources for future applications so more applications can be accommodated using incremental design

References

[1] M. Di Natale, and A. Sangiovanni-Vincentelli, "Moving from federated to integrated architectures in automotive: the role of standards, methods and tools," Proc. of the IEEE, 2010.

[2] J. Yoo, Y. Lee, D. Kim, and K. Park, "An Android-based automotive middleware architecture for Plug-and-Play of applications," ICOS, 2012.

[3] H. Martorell, J. Fabre, M. Roy, and R. Valentin, "Towards Dynamic Updates in AUTOSAR," Workshop, International Conference on Computer Safety, Reliability and Security, 2013.

[4] S. Sommer, A. Camek, K. Becker, C. Buckl, A. Zirkler, L. Fiege, M. Armbruster, G. Spiegelberg, and A. Knoll, "RACE: A centrialized platform computer based architecture for automotive applications," IEVC, 2013.

[5] J. Frtunikj, V. Rupanov, A. Camek, C. Buckl, and A. Knoll, "A safety aware run-time environment for adaptive automotive control systems," ERTS2, 2014.

[6] C. Buckl, M. Geisinger, D. Gulati, F. Ruiz-Bertol, and A. Knoll, "CHROMOSOME: A run-time environment for plug\&play-capable embedded real-time system," APRES, 2014.

[7] A. Zeeb, "Plug and play solution for Autosar software components," ATZelektronik worldwide, 2012.

[8] AS6802: Time-triggered Ethernet. SAE International, 2011.

[9] IEEE 802.1 Time-Sensitive Networking Task Group: http://www.ieee802.org/1/pages/tsn.html.

[10] W. Steiner, "An evaluation of SMT-based schedule synthesis for time-triggered multi-hop networks," RTSS 2010.

[11] W. Steiner, "Synthesis of static communication schedules for mixed-criticality systems," ISORCW, 2011.

[12] P. Pop, P. Eles, Z. Peng, and T. Pop, "Scheduling and mapping in an incremental design methodology for distributed real-time embedded systems," IEEE Transactions on VLSI Systems, 2004.

References

[13] D. Tamas-Selicean, P. Pop, and W. Steiner, "Synthesis of communication schedules for TTEthernet-based mixed-criticality systems," CODES+ISSS, 2012.

[14] R. S. Oliver, S. S. Craciunas, and G. Stoger, "Analysis of deterministic Ethernet scheduling for the industrial Internet of Things," CAMAD, 2014.

[15] L. Zhang, D. Goswami, R. Schneider, and S. Chakraborty, "Task- and network-level schedule co-synthesis of Ethernet-based time-triggered systems," ASP-DAC, 2014.

[16] S. S. Craciunas and R. S. Oliver, "SMT-based task- and network-level static schedule generation for time-triggered networked systems," RTNS, 2014.

[17] F. Sagstetter, P. Waszecki, S. Steinhorst, M. Lukasiewycz, and S. Chakraborty, "Multischedule synthesis for variant management in automotive time-triggered systems," TCAD, 2015.

[18] M. Gutierrez, W. Steiner, R. Dobrin, and S. Punnekkat, "A configuration agent based on the time-triggered paradigm for real-time networks," WFCS, 2015.

[19] M. Gutierrez, W. Steiner, R. Dobrin, and S. Punnekkat, "Learning the parameters of periodic traffic based on network measurements," IEEE International Workshop on Measurements \& Networking, 2015, pp 1-6.

[20] W. Steiner, M. Gutierrez, Z. Matyas, F. Pozo, and G. Rodriguez-Navas, "Current techniques, trends and new horizons in avionics networks configuration," DASC, 2015.

[21] Gecode: http://www.gecode.org.

[22] https://github.com/eidheim.

[23] G. C. Necula, "Proof-carrying code," POPL, 1997.

Thanks for your attention!

Q/A