STOMP

Agile Evaluation of Scheduling Policies in Heterogeneous Multi-Processors

Augusto Vega
John-David Wellman
Hubertus Franke
Alper Buyuktosunoglu
Pradip Bose

IBM T. J. Watson Research Center

Aporva Amarnath
Hiwot Kassa
Subhankar Pal
Ronald Dreslinski

University of Michigan

IBM

Acknowledgment

- Thanks to the many IBM colleagues who contribute to and support different aspects of this work + our esteemed university collaborators at Harvard, Columbia, and UIUC (Profs. David Brooks, Vijay Janapa Reddi, Gu-Yeon Wei, Luca Carloni, Ken Shepard, Sarita Adve, Vikram Adve, Sasa Misailovic) + many brilliant graduate students and postdocs!

- Special thanks to **Dr. Thomas Rondeau**, Program Manager of the DARPA MTO DSSoC Program

This research was developed, in part, with funding from the Defense Advanced Research Projects Agency (DARPA). The views, opinions and/or findings expressed are those of the authors and should not be interpreted as representing the official views or policies of the Department of Defense or the U.S. Government. This document is approved for public release: distribution unlimited.

Domain-Specific Systems – It All Starts Here

Highly-heterogeneous applications

+

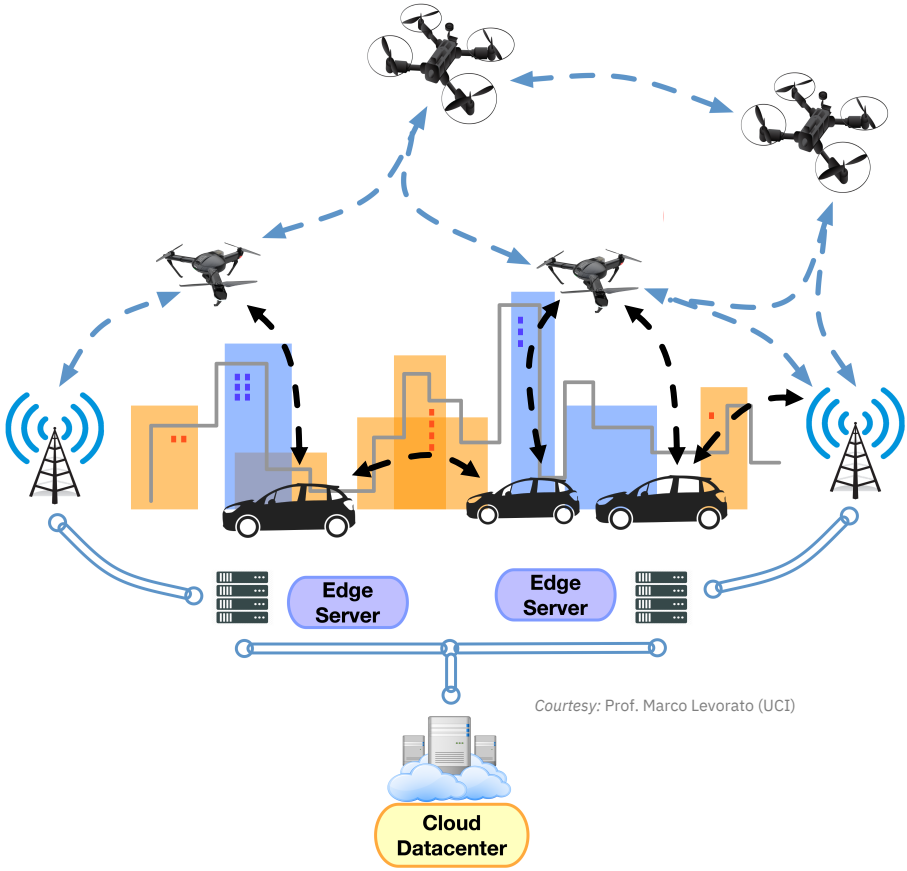
Performance needs

+

Power/energy-efficiency needs

+

Security/privacy needs



Courtesy: Prof. Marco Levorato (UCI)

Domain-Specific Systems – It All Starts Here

Highly-heterogeneous applications

+

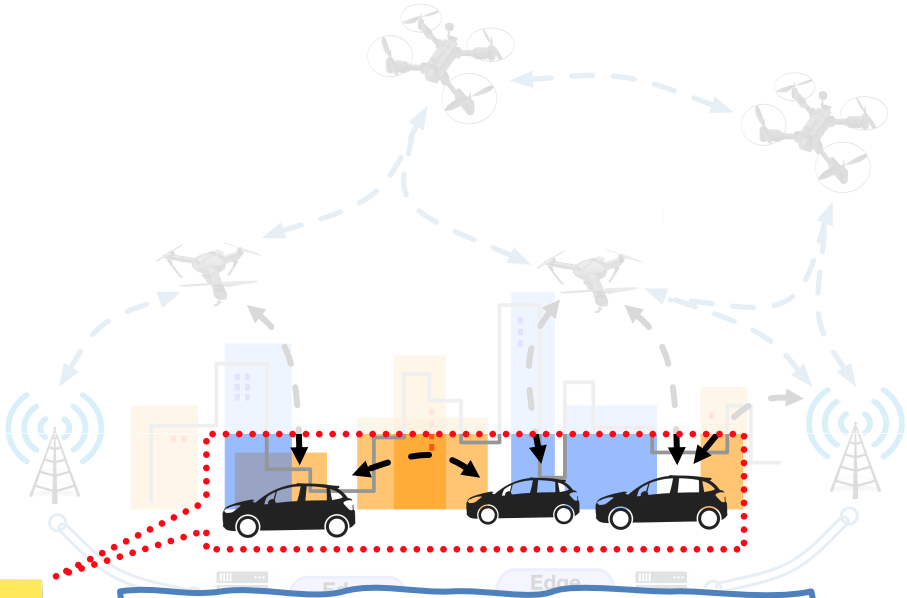
Performance needs

+

Power/energy-efficiency needs

+

Security/privacy needs



SW+HW support for Cooperative Perception

Conventional schedulers are **not optimized** for the characteristics of heterogeneous chips which calls for more intelligent and efficient scheduling

Outline

1. DARPA's Domain-Specific System on Chip (DSSoC) Program
- Agile development of efficient and programmable SoCs

2. EPOCHS Agile Flow Methodology
- Target: embedded processors for AVs

3. STOMP for Agile Evaluation of Scheduling Policies
- And the AVSched suite of policies

Outline

1. DARPA's Domain-Specific System on Chip (DSSoC) Program
- Agile development of efficient and programmable SoCs

2. EPOCHS Agile Flow Methodology
- Target: embedded processors for AVs

3. STOMP for Agile Evaluation of Scheduling Policies
- And the AVSched suite of policies

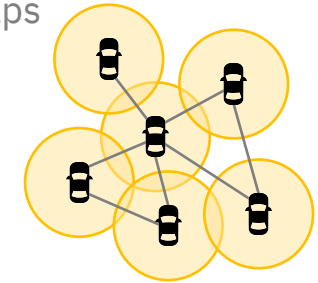
DARPA's Domain-Specific System on Chip (DSSoC) Program*

Program Manager: Dr. Tom Rondeau

- **Goal:** to develop programable heterogeneous SoCs to significantly improve performance of applications within a **domain**

We target the domain of embedded chips for **connected/autonomous cars**

- **Cooperative perception**
  - Cars exchange locally-generated maps
  - Each vehicle merges its local map and the received ones in real time



computer vision **software radio**

24 Jul 2018 | 17:00 GMT

DARPA Picks Its First Set of Winners in Electronics Resurgence Initiative

Teams announced in design, architecture, and materials and integration programs under the $1.5 billion effort to remake U.S. electronics

By **Samuel K. Moore**

Source: IEEE Spectrum (July 2018)

* https://www.darpa.mil/program/domain-specific-system-on-chip

Outline

1. DARPA's Domain-Specific System on Chip (DSSoC) Program
   - Agile development of efficient and programmable SoCs

2. **EPOCHS Agile Flow Methodology**
   - Target: embedded processors for AVs

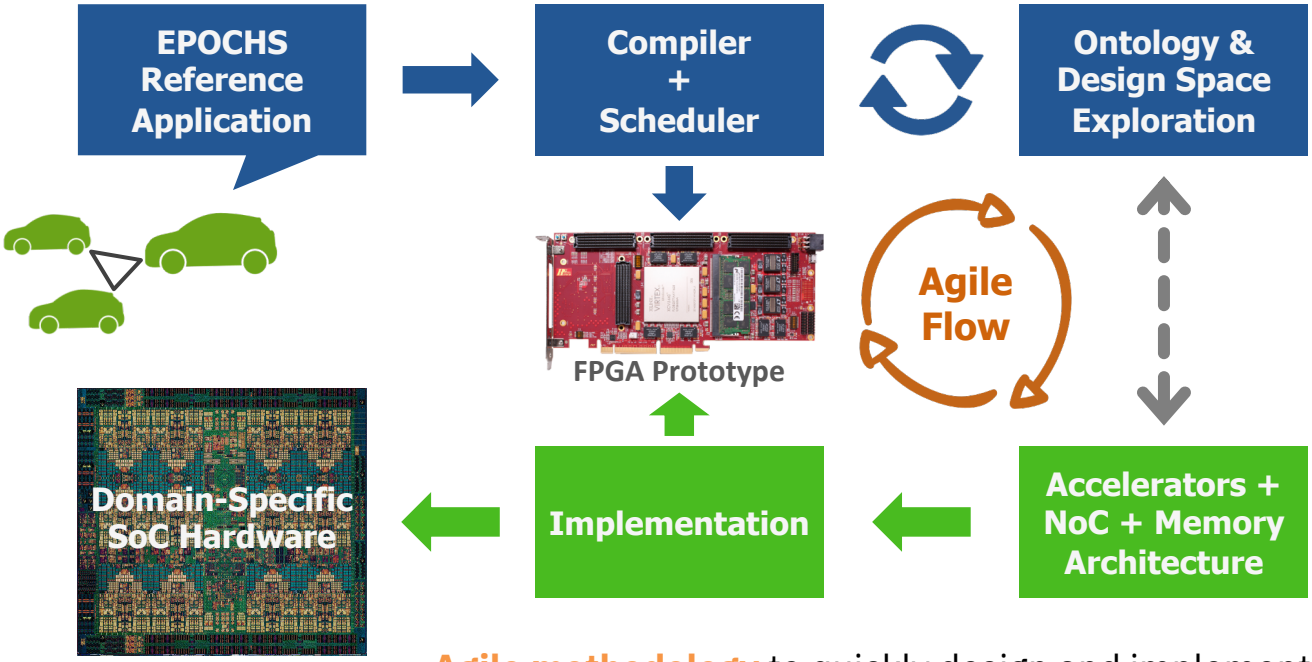
3. STOMP for Agile Evaluation of Scheduling Policies
   - And the AVSched suite of policies

Efficient Programmability Of Cognitive Heterogeneous Systems

“EPOCHS” → our proposed solution for the design challenge presented by the DSSoC program

Efficient Programmability Of Cognitive Heterogeneous Systems

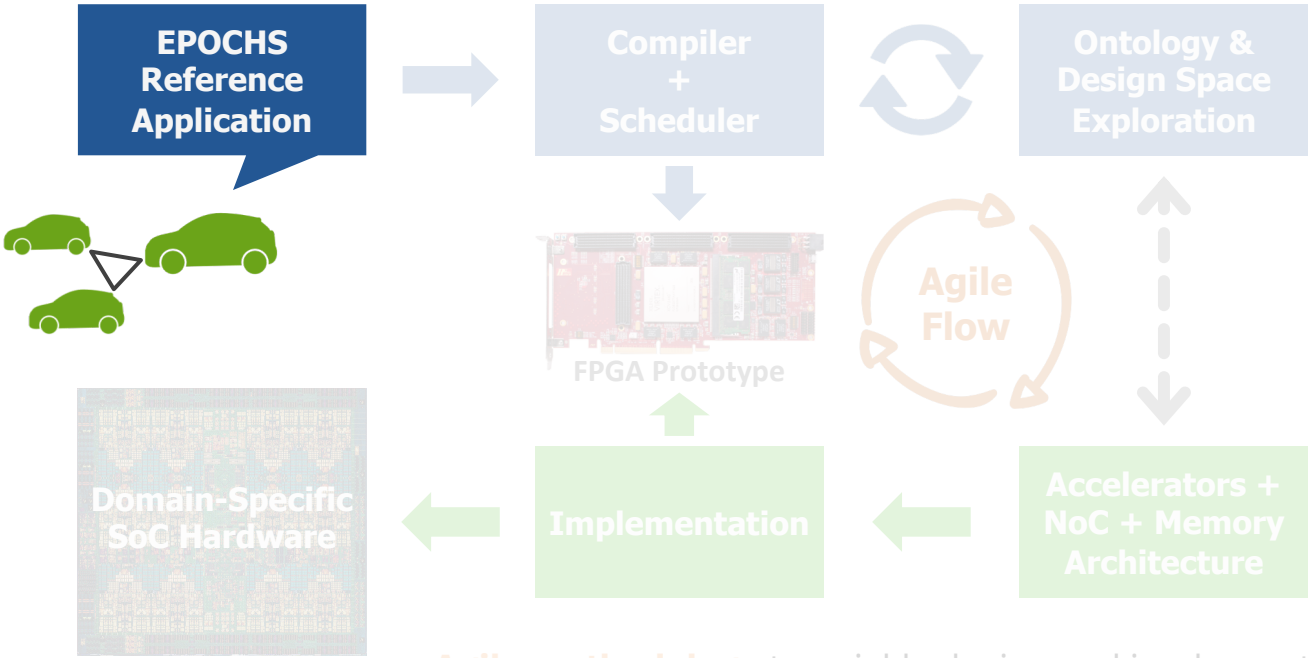
EPOCHS Agile Flow Methodology



Agile methodology to quickly design and implement an easily programmed domain-specific SoC for real-time cognitive decision engines in connected vehicles

Efficient Programmability Of Cognitive Heterogeneous Systems

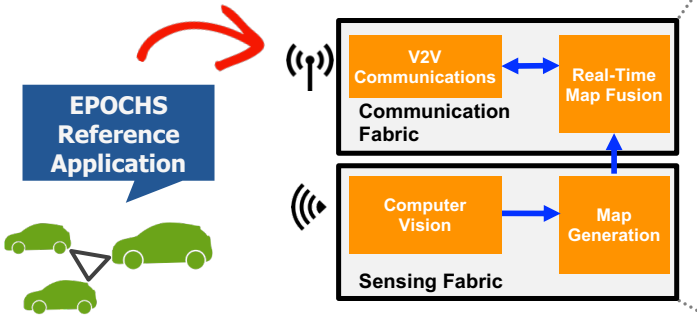
EPOCHS Agile Flow Methodology



Agile methodology to quickly design and implement an easily programmed domain-specific SoC for real-time cognitive decision engines in connected vehicles

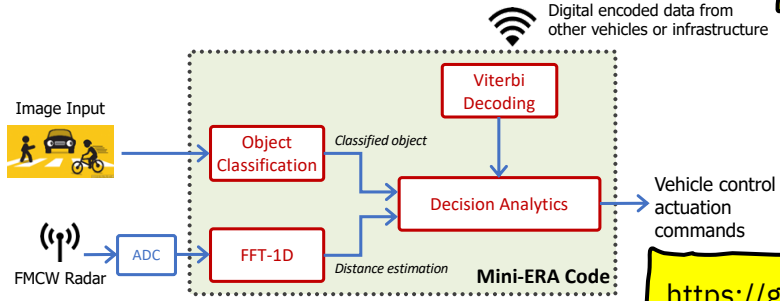
AV Application: ERA and Mini-ERA

- **ERA:** Cooperative perception for AVs
  - Occupancy map generation and fusion
  - DSRC-based V2V communication[1]



https://github.com/IBM/era

- **Mini-ERA:**
  - Scaled down-version of ERA
  - Intended to drive the initial demonstration of our FPGA SoC prototype



https://github.com/IBM/mini-era

1. https://github.com/bastibl/gr-ieee802-11

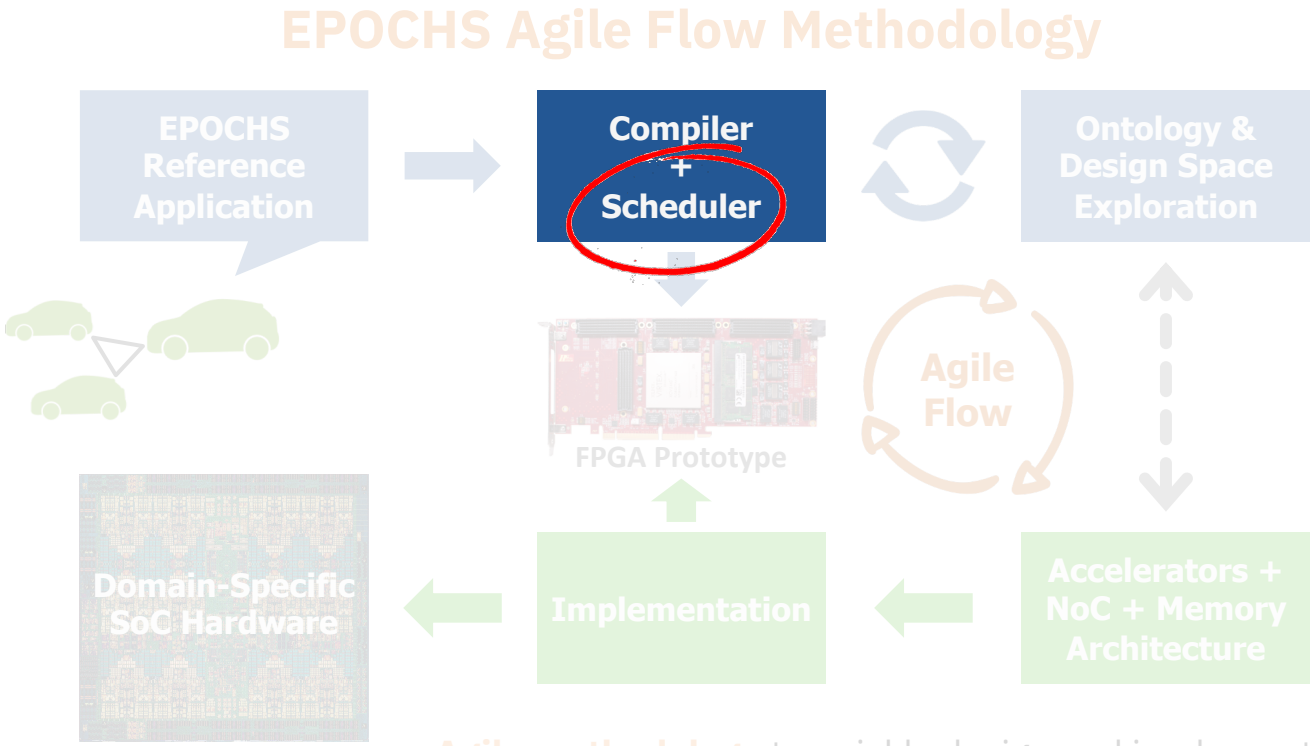
Outline

1. **DARPA's Domain-Specific System on Chip (DSSoC) Program**
   - Agile development of efficient and programmable SoCs

2. **EPOCHS Agile Flow Methodology**
   - Target: embedded processors for AVs

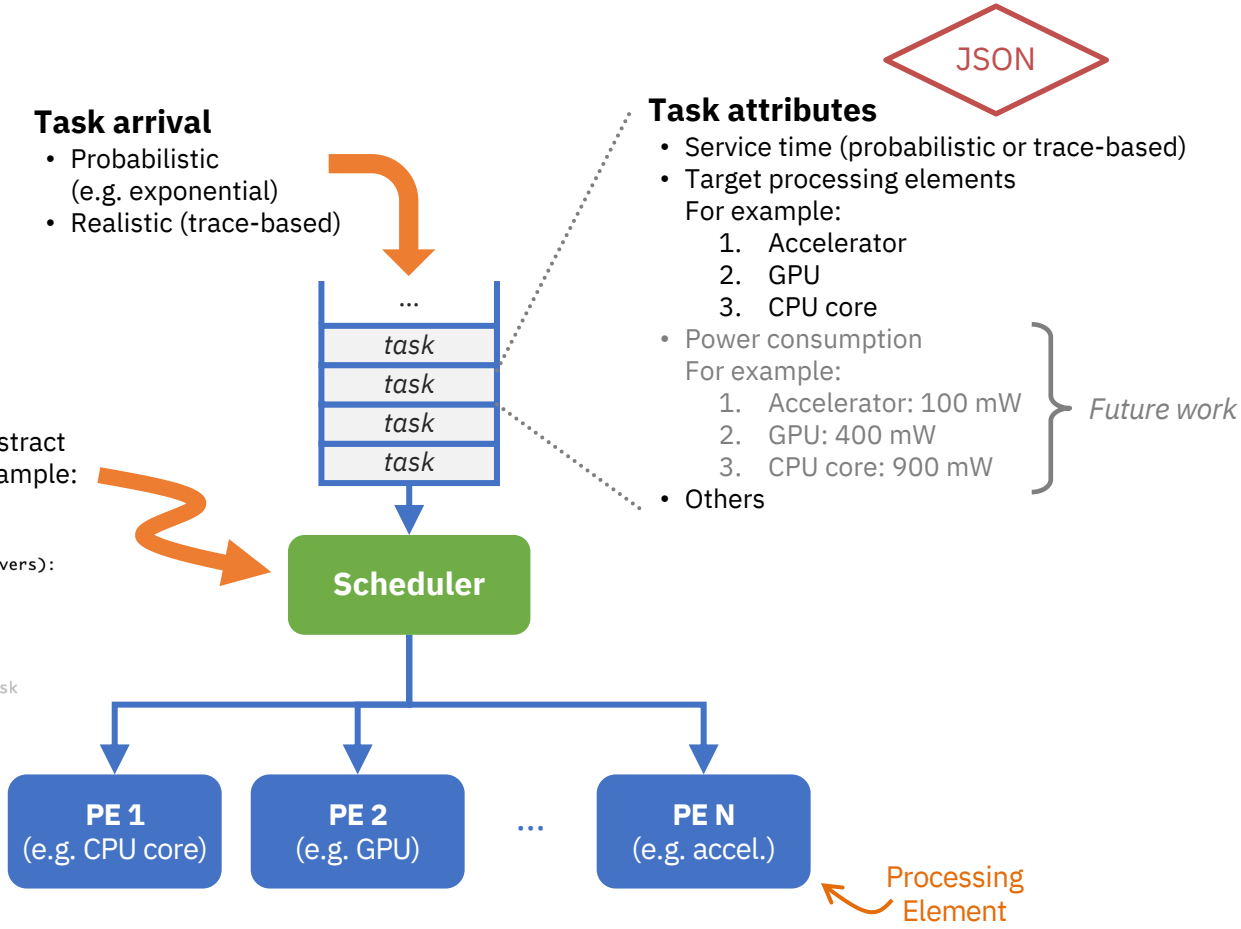
3. **STOMP for Agile Evaluation of Scheduling Policies** 
   - And the AVSched suite of policies

Efficient Programmability Of Cognitive Heterogeneous Systems



Agile methodology to quickly design and implement an easily programmed domain-specific SoC for real-time cognitive decision engines in connected vehicles

STOMP Overview



Python

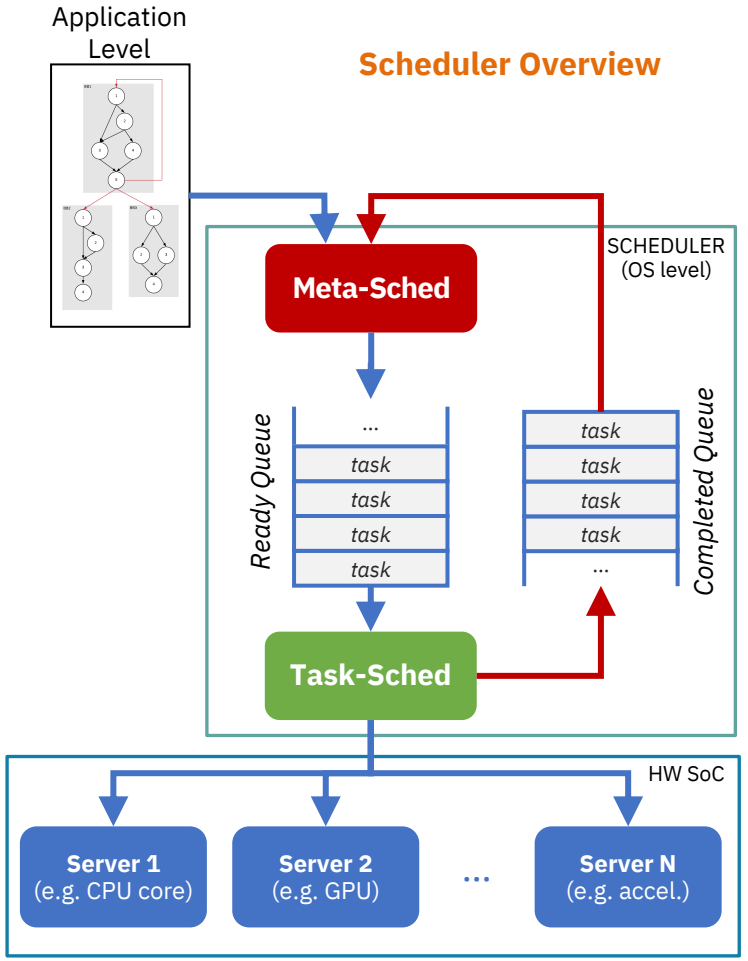
"Pluggable" Scheduling Policy

- The user is only required to implement the abstract Python class `BaseSchedulingPolicy` – for example:

```
class SchedulingPolicy(BaseSchedulingPolicy):

    def assign_task_to_server(self, sim_time, tasks, servers):

        if (len(tasks) == 0):
            # There aren't tasks to serve
            return None

        # Look for an available server to process the task
        for server in servers:

            if (not server.busy):

                # Pop task in queue's head
                task = tasks.pop()
                server.assign_task(sim_time, task)

                return server

        return None
```

STOMP Intrinsic Operation

- STOMP consists of two integral parts:
  - **Meta scheduler** → responsible for application (DAG) pre-processing
  - **Task scheduler** → assigns ready tasks to available PEs to optimize target metrics
- Meta-Sched and Task-Sched communicate via two queues: *ready* and *completed*
- **Input:** directed acyclic-graphs (DAGs) of multiple tasks with associated real-time constraints (priority and deadline)

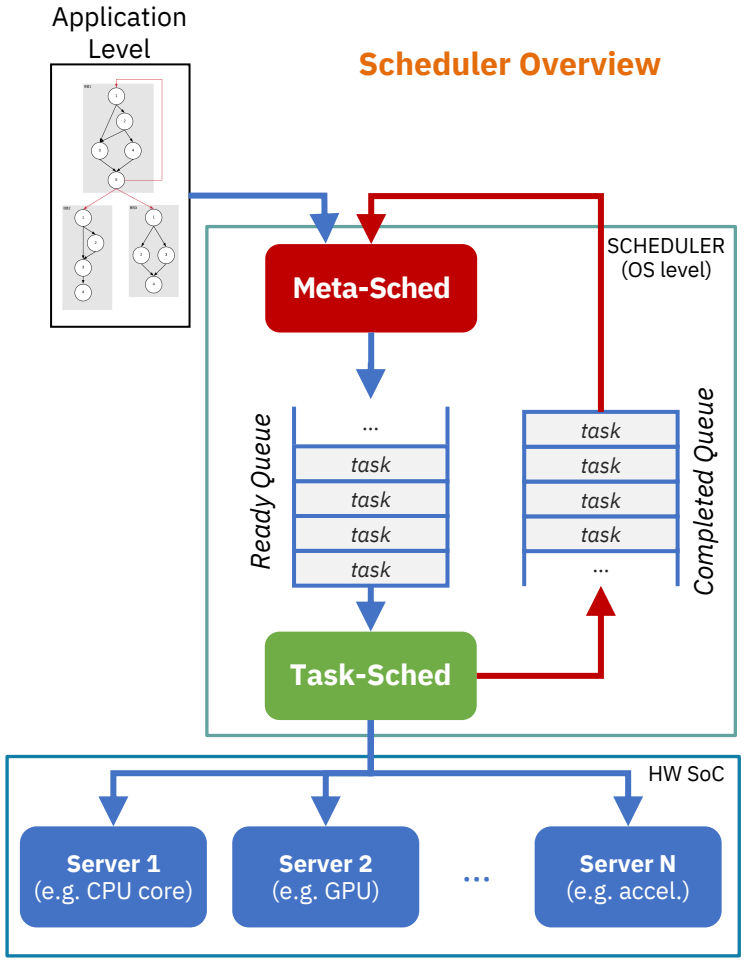


Meta Scheduler

- Responsible for application (DAG) pre-processing:
  - Track dependencies, assign task ranks, etc.
- Keeps *ready* tasks ordered by **"rank"**
  - A task's rank can be computed in different ways
  - E.g. as a function of its criticality and deadline

$$\boldsymbol{Rank}_i = \frac{Criticality_i}{Deadline_i}$$

- Drops non-critical DAGs if deadline is missed
  - All remaining tasks in the DAG are dropped
  - Help reduce task traffic in the system

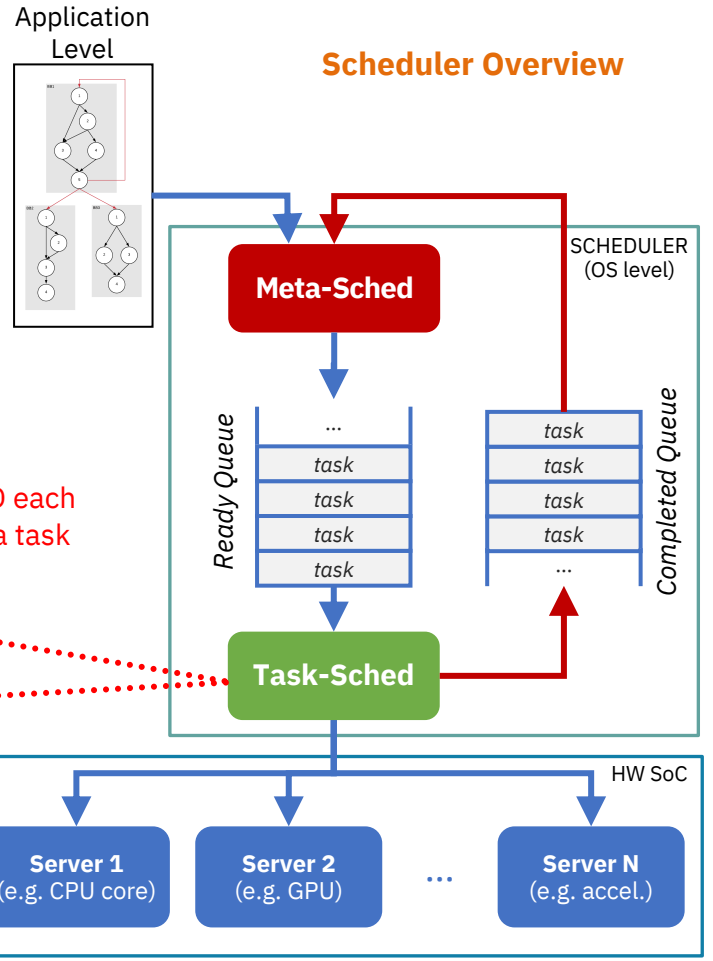


Task Scheduler

The user primarily defines the assignment actions:
(here the task is scheduled to the fastest server type)

```python
from stomp import BaseSchedulingPolicy

class SchedulingPolicy(BaseSchedulingPolicy):

  def init(self, servers, stomp_stats, stomp_params):
    ...


  def remove_task_from_server(self, sim_time, server):
    ...

  def assign_task_to_server(self, sim_time, tasks):

    if (len(tasks) == 0):
    # There aren't tasks to serve
      return None

    # Determine task's best scheduling option (target server)
    target_server_type = tasks[0].mean_service_time_list[0][0]

    # Look for an available server to process the task
    for server in self.servers:
      if (server.type == target_server_type and not server.busy):
        # Pop task in queue's head and assign it to server
        server.assign_task(sim_time, tasks.pop(0))
        return server

    return None
```

Invoked by SCHED each time it schedules a task to a server



Simulation Parameters and Configuration

- Example JSON configuration file:

```
"general" : {
     "logging_level":        "INFO",
     "random_seed":          0,
     "working_dir":          ".",
     "basename":             "",
     "pre_gen_arrivals":     false,
     "input_trace_file":     "",
     "output_trace_file":    ""
  },

  "simulation" : {
     "sched_policy_module":  "policies.simple_policy_ver3",
     "max_tasks_simulated":  10000,
     "mean_arrival_time":    50,
     "distribution":         "Poisson",
     "power_mgmt_enabled":   false,
     "max_queue_size":       1000000,
```

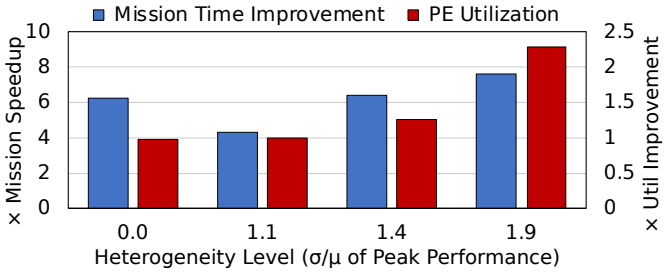
```
"servers" : {
          "cpu_core" :  { "count" : 8 },
          "gpu" :       { "count" : 2 },
          "fft_accel" : { "count" : 1 }
      },

"tasks" : {
          "fft" : {
              "mean_service_time" : {
                  "cpu_core"  : 500,
                  "gpu"       : 100,
                  "fft_accel" : 10
              },

              "stdev_service_time" : {
                  "cpu_core"  : 5.0,
                  "gpu"       : 1.0,
                  "fft_accel" : 0.1
              }
          },
```

Running STOMP

AVSched: Scheduling Policies

- Policies conceived to optimize response time while meeting **real-time** *and* **criticality (safety)** constraints
  - Real-time-aware-only schedulers are not enough for AV applications
  - AVSched also considers task criticality, showing significant benefits over real-time-aware-only schedulers

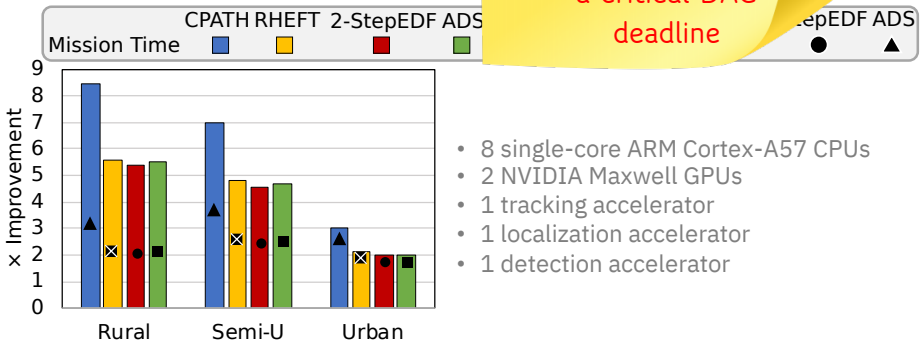
Learned lesson



Baseline schedulers complete only a fraction of the mission at the maximum safe speed of AVSched before missing a critical DAG deadline

- Comparison against existing real-time-aware schedulers:
  - **Workload:** AV pipeline including object detection/tracking, localization, and mission/motion planning
  - **Metrics:** mission time and PE utilization
  - **Scenarios:** rural, semi-urban, urban

- **AVSched achieves significant improvements over state-of-the-art real-time schedulers**



- 8 single-core ARM Cortex-A57 CPUs
- 2 NVIDIA Maxwell GPUs
- 1 tracking accelerator
- 1 localization accelerator
- 1 detection accelerator

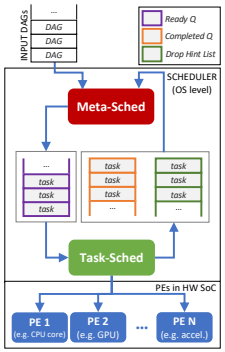
Task criticality plays a key role driving AVSched's decision

Scheduler Library: STOMP Deployment into a Real SoC

AV Application

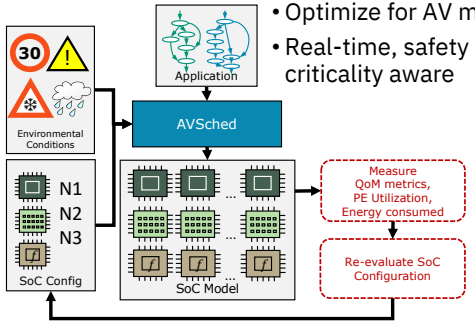
ERA and Mini-ERA

STOMP: Simulation Infrastructure



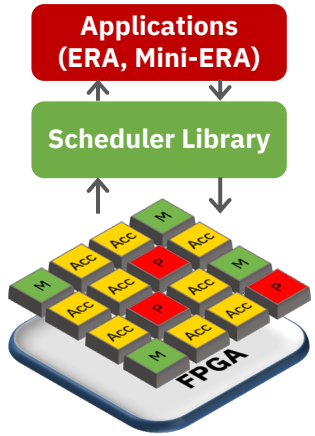
- "Plug & play" approach
- Fast prototyping of scheduling policies in domain-specific SoC

AVSched: Scheduling Policies



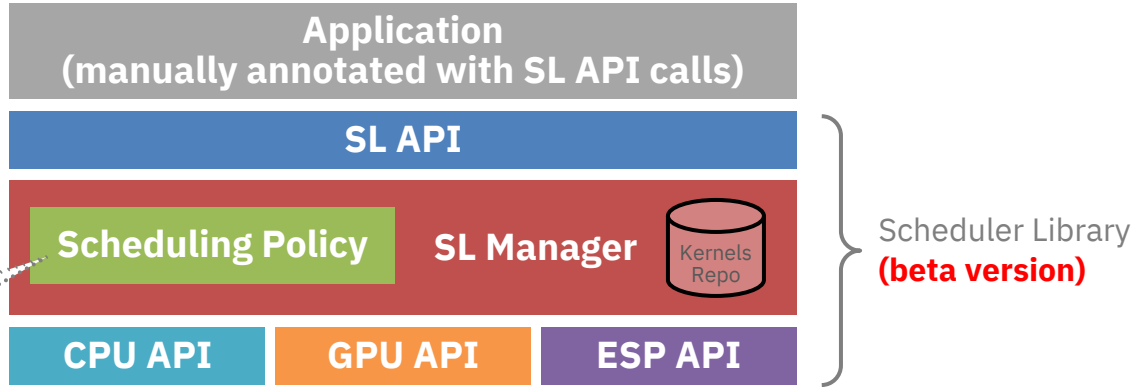
- Optimize for AV mission
- Real-time, safety and criticality aware

EPOCHS-0 SoC



Scheduler Library and HPVM Compiler

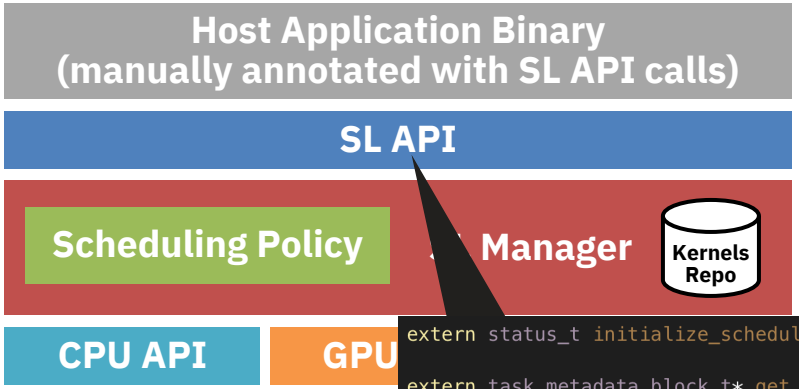
- First beta version available: https://github.com/IBM/scheduler-library
  - Tailored to Mini-ERA
  - Extension to other applications in progress

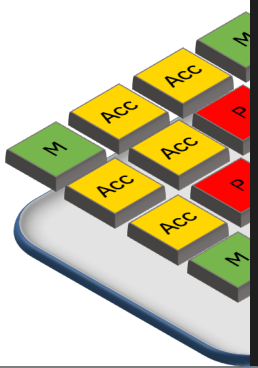


- Where all the magic happens
- User-specified (plug & play)
- Previously defined and evaluated using STOMP

EPOCHS-0
FPGA

Scheduler Library and HPVM Compiler

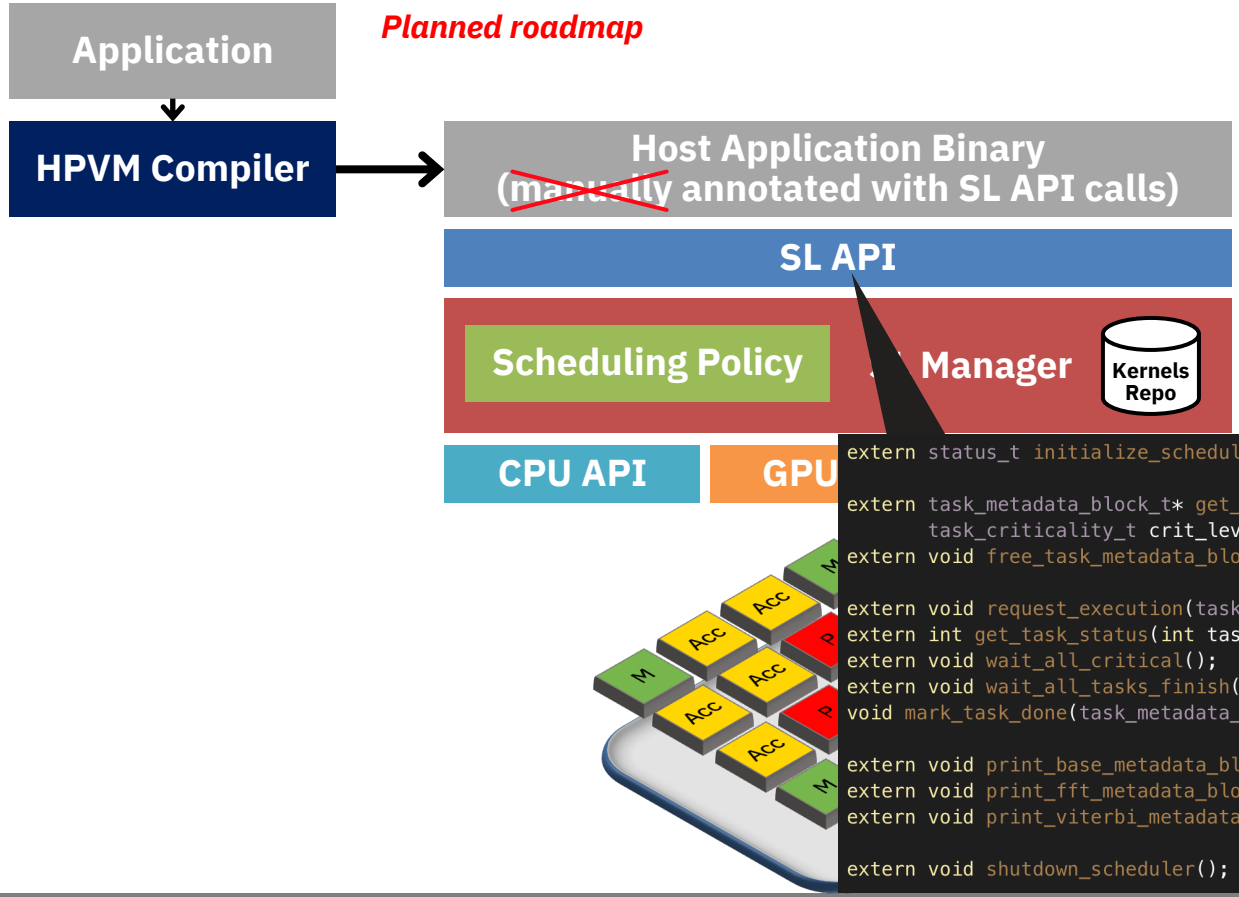




API Summary

```
extern status_t initialize_scheduler();

extern task_metadata_block_t* get_task_metadata_block(scheduler_jobs_t task_type,
       task_criticality_t crit_level, float * task_profile);
extern void free_task_metadata_block(task_metadata_block_t* mb);

extern void request_execution(task_metadata_block_t* task_metadata_block);
extern int get_task_status(int task_id);
extern void wait_all_critical();
extern void wait_all_tasks_finish();
void mark_task_done(task_metadata_block_t* task_metadata_block);

extern void print_base_metadata_block_contents(task_metadata_block_t* mb);
extern void print_fft_metadata_block_contents(task_metadata_block_t* mb);
extern void print_viterbi_metadata_block_contents(task_metadata_block_t* mb);

extern void shutdown_scheduler();
```

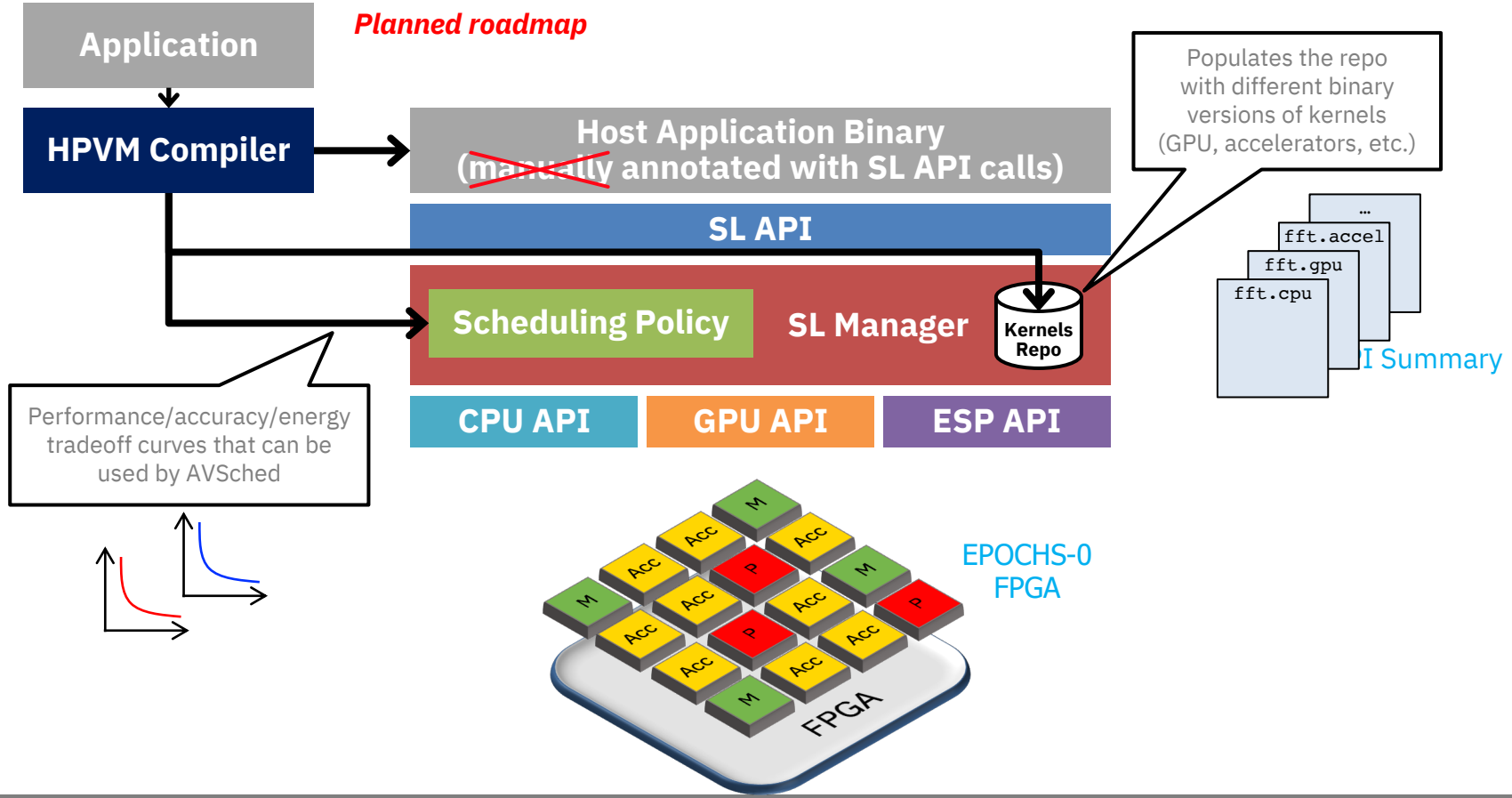
Scheduler Library and HPVM Compiler



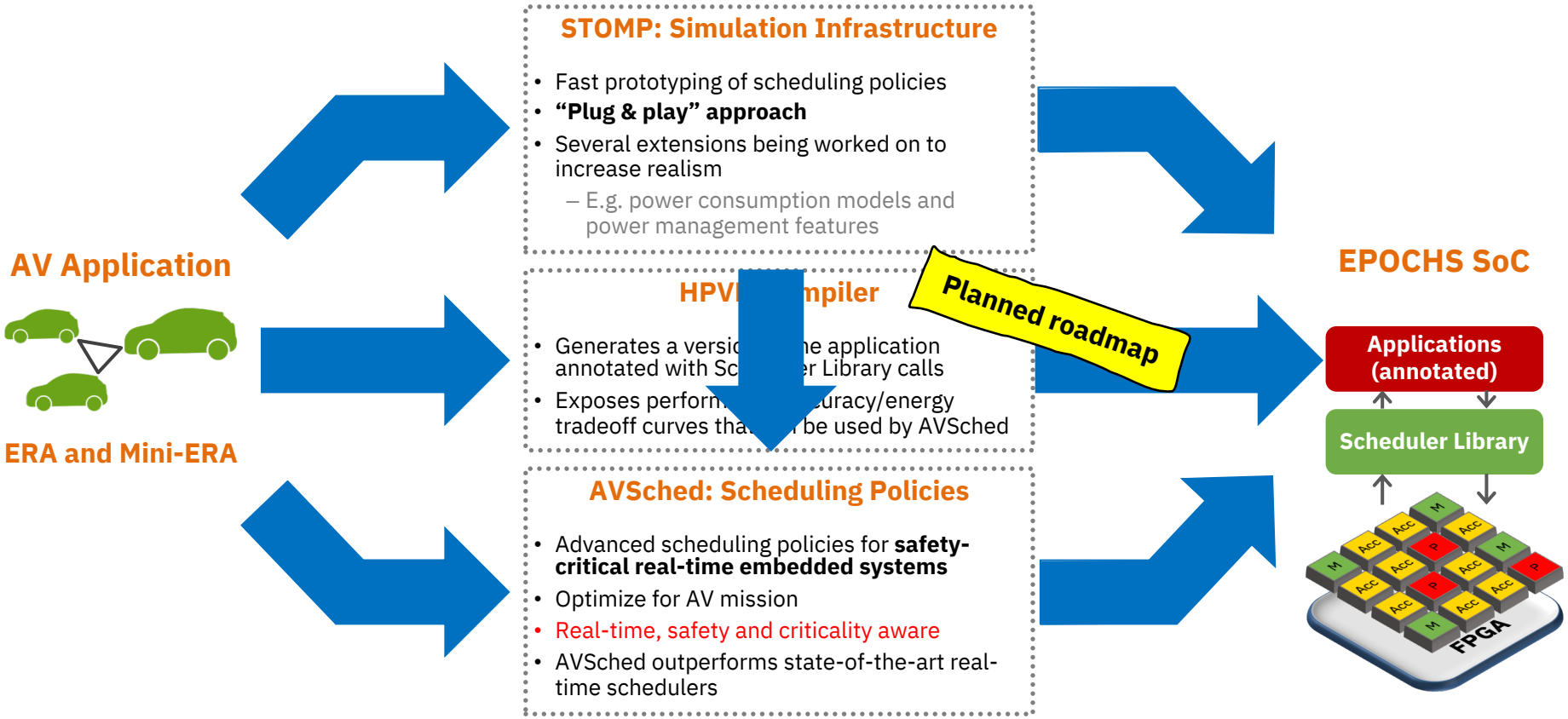
API Summary

```
extern status_t initialize_scheduler();

extern task_metadata_block_t* get_task_metadata_block(scheduler_jobs_t task_type,
       task_criticality_t crit_level, float * task_profile);
extern void free_task_metadata_block(task_metadata_block_t* mb);

extern void request_execution(task_metadata_block_t* task_metadata_block);
extern int get_task_status(int task_id);
extern void wait_all_critical();
extern void wait_all_tasks_finish();
void mark_task_done(task_metadata_block_t* task_metadata_block);

extern void print_base_metadata_block_contents(task_metadata_block_t* mb);
extern void print_fft_metadata_block_contents(task_metadata_block_t* mb);
extern void print_viterbi_metadata_block_contents(task_metadata_block_t* mb);

extern void shutdown_scheduler();
```

Scheduler Library and HPVM Compiler



Summary and Path Forward



Thank you

Dr. Augusto Vega
Research Staff Member
—
ajvega@us.ibm.com
@AugustoJVega
ibm.com