COMPUTERSCIENCE

☆Functional Reactive Programming for Real-Time and Cyber-Physical Systems

Albert M. K. Cheng

- Outline
  - Embedded Real-Time Systems (RTS)
  - Functional Reactive Systems (FRS)
  - Cyber-Physical Systems (CPS)
  - Haskell and Functional Reactive Programming (FRP)
  - Priority-based FRP (P-FRP)
  - Response time analysis and scheduling

* Supported in part by the National Science Foundation under Awards No. 0720856 and No. 1219082.

UNIVERSITY of HOUSTON

COMPUTERSCIENCE

Real-Time Systems Group

- **Director:** Prof. Albert M. K. Cheng
- **PhD students:** Yong Woon Ahn, Yu Li, Xingliang Zou, Behnaz Sanati, Zeinab Kazemi, Carlos Rincon, Hassan Hafiz
- **MS student:** Chonghua Li
- **Undergraduate student (NSF-REU):** Daniel Underwood
- **Visiting scholars 2013-2016:** Yu Jiang (Heilongjiang U.), Qiang Zhou (Beihang U.), Yufeng Zhao (Xi'an Tech. U.), Qiao-Ling Wang (Jiangxi Agricultural U.), Yunfeng Peng (UST-Beijing)
- **Recent graduates and their positions:** Yuanfeng Wen (MS, Microsoft, then Facebook), Daxiao Liu (Uber), Chaitanya Belwal (PhD, Halliburton and Visiting Assistant Professor, UHCL), Jim Ras (PhD), Jian Lin (PhD, Assistant Professor, UHCL)

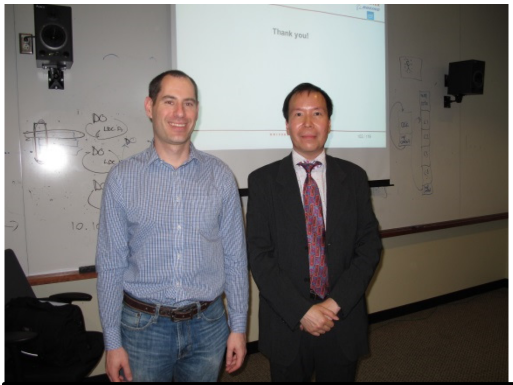
Yu Li (Best Junior PhD Student Awardee and Friends of NSM Graduate Fellow) and Prof. Albert Cheng visit the NSF-sponsored Arecibo Observatory after their presentation at the flagship RTSS 2012 in Puerto Rico.

Real-time systems research group at Yuanfeng Wen's graduation party in May 2013. Yuanfeng is now at Facebook.

Fall 2014 (9/3) group meeting - from left to right: Dr. Qiang Zhou, Qiong Lu, Carlos Rincon, Chonghua Li, Prof. Yu Jiang, Xin Liu, Prof. Yufeng Zhao, Prof. Albert Cheng, Xingliang (Jeffrey) Zou, Daxiao Liu, Yu Li, Yong Woon Ahn, and Behnaz Sanati. Zeinab Kazemi in class.

UNIVERSITY of HOUSTON

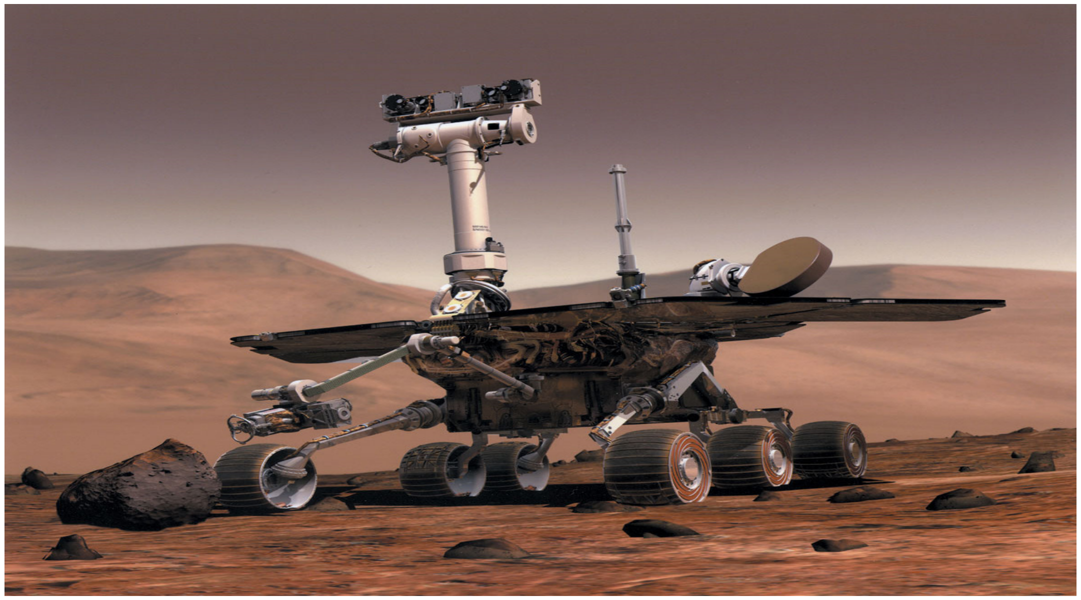
Recent Seminar Visits



With Prof. Dan Grossman Audience at the University of Washington (4/2015)

With Prof. Enrico Tronci, U Rome (12/2014) University of Oxford With Prof. Joel Ouaknine (4/2014)

☆ Real-Time Systems Theory



Pathfinder mission to Mars: best known Priority Inversion problem.
Failure to turn on priority Inheritance (PI) - Most PI schemes complicate and slow down the locking code, and often are used to compensate for poor application designs.
http://research.microsoft.com/en-us/um/people/mbj/mars_pathfinder/mars_pathfinder.html

Real-Time Systems Theory

- **The more components a real-time system has, the more difficult it is to build and maintain.**
  - In such systems, **preemptive scheduling** may not be suitable, since it is likely to create runtime overheads which can result in worst-case task execution times of up to **40%** greater than fully **non-preemptive** execution.
    - Yao G., Buttazzo G., Bertogna M., "Feasibility analysis under fixed priority scheduling with limited preemptions," Real-Time Systems, Volume 47 Issue 3, pages: 198-223, May 2011.

Real-Time Systems Theory

- However, **preemptive** scheduling allows for more feasible schedules than **non-preemptive** scheduling.
- **Non-preemptive** scheduling automatically prevents unbounded priority inversion, which avoids the need for a concurrency control protocol, leading to a less complex scheduling model.
- However, fully **non-preemptive** scheduling is too inflexible for some real-time applications, and has the added disadvantage of potentially introducing large blocking times that would make it impossible to guarantee the schedulability of the task set.

Real-Time Systems Theory

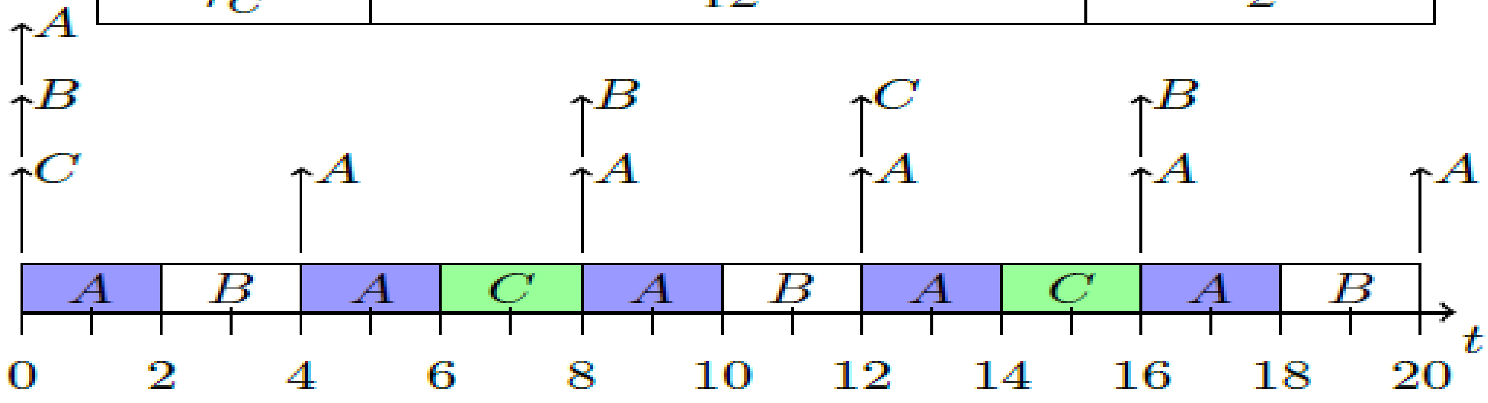
- Simplify the design and scheduling
  - Avoid priority inheritance
  - Use functional programming
  - Use abort-and-restart
  - Use harmonic task sets
    - However, harmonic tasks sets may be too restrictive for some situations. For example, one sensor needs to be serviced every 9 seconds and another (because of its design / physical characteristics) 10 seconds.

Real-Time Systems Theory

- Example (1) - Harmonic task sets
  - Can achieve 100% CPU utilization
  - Can avoid preemption and context switches costs

V. Bonifaci, A. Marchetti-Spaccamela, N. Megow, and A. Wiese, "**Polynomial-Time Exact Schedulability Tests for Harmonic Real-Time Tasks,**" **RTSS 2013.**

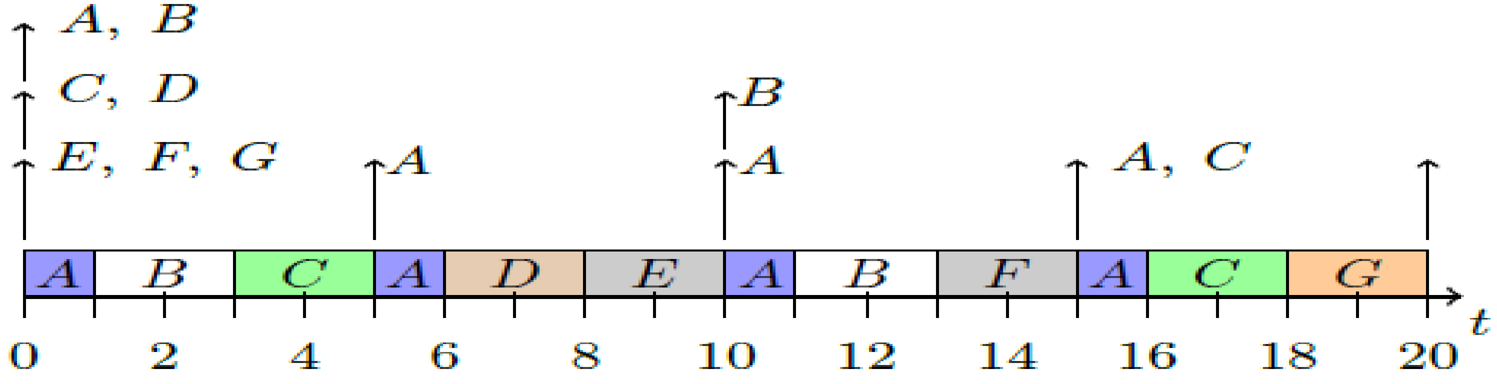
Task ID	Period=Deadline (T = D)	WCET (C)
τ_A	4	2
τ_B	8	2
τ_C	12	2



Real-Time Systems Theory

- Example (2) - Harmonic task sets

Task ID	Period=Deadline (T = D)	WCET (C)
τ_A	5	1
τ_B	10	2
τ_C	15	2
τ_D	20	2
τ_E	25	2
τ_F	30	2
τ_G	35	2



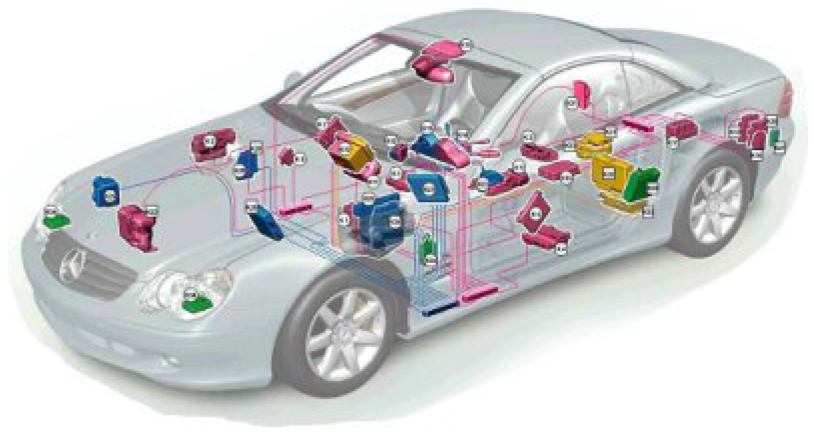
CS@UH

Embedded Real-Time Systems

- An embedded system is a computer system designed for specific control functions within a larger system

  (**A** is embedded into **B** for control)

- Often with such systems there are constraints such as deadlines, memory, power, size, etc.



Embedded Real-Time Systems

- **Real-time systems (RTS)** are reactive systems that are required to respond to an environment in a bounded amount of time.
- **Functional reactive systems (FRS)**
- **Cyber-physical systems (CPS)**
  - Challenges
    - Complexity
    - Reliability
      - Fault-tolerant design
      - Meeting deadlines (Response Time Analysis (RTA))
    - Security/Privacy

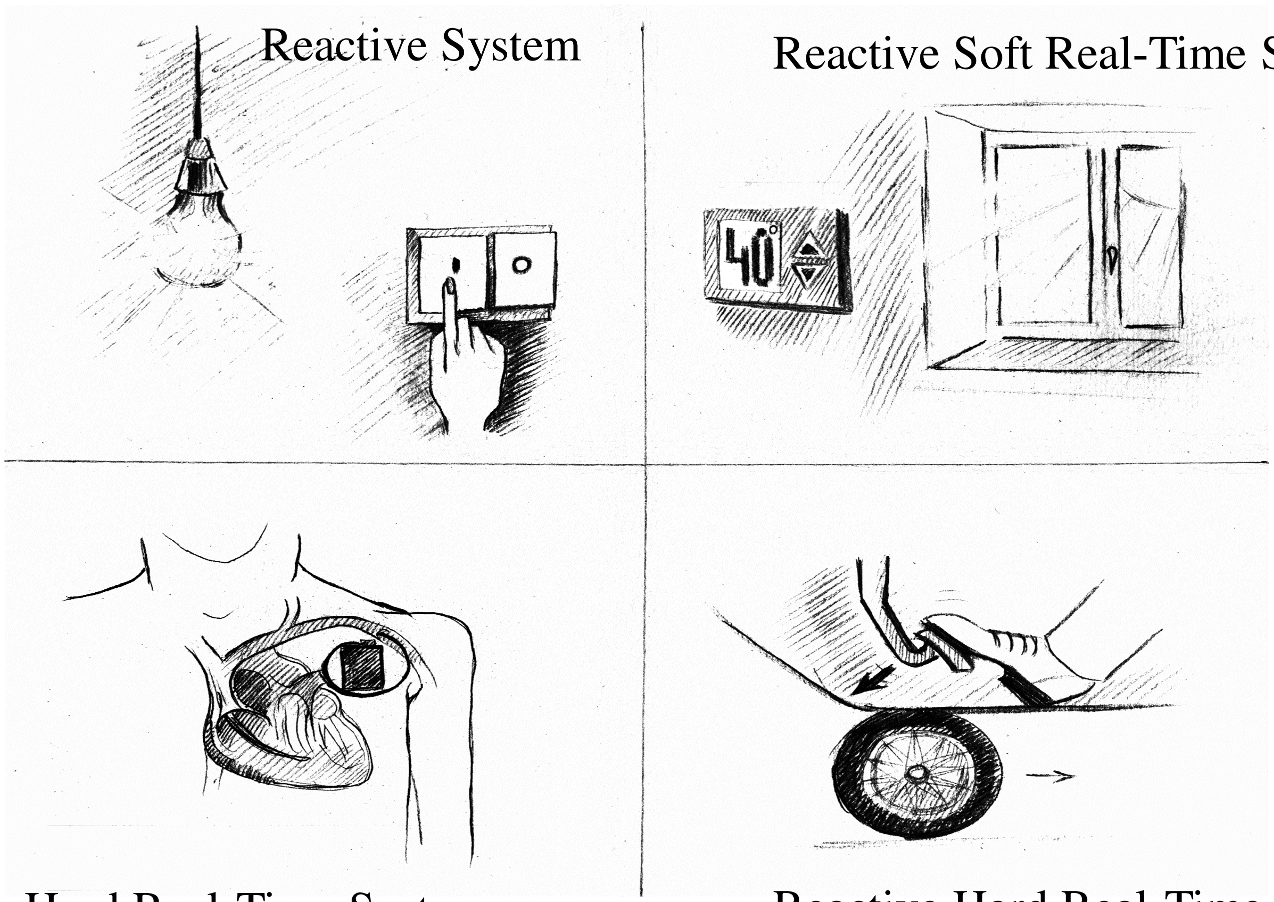
Functional Reactive Systems (FRS)

Systems that react to the environment being monitored and controlled in a timely fashion using functional (reactive) programming are known as Functional Reactive Systems (FRS).

These systems can range from small devices (which are not a CPS) to distributed and complex components (similar to a CPS).

CS@UH

Functional Reactive Systems (FRS)



Reactive Hard Real-Time System

Reactive Hard Real-Time System

Cyber-Physical Systems (CPS)

- Systematic integration of computation/information processing and physical processes and devices.
- Communication and sensing are components of CPS

Cyber-Physical Systems (CPS)

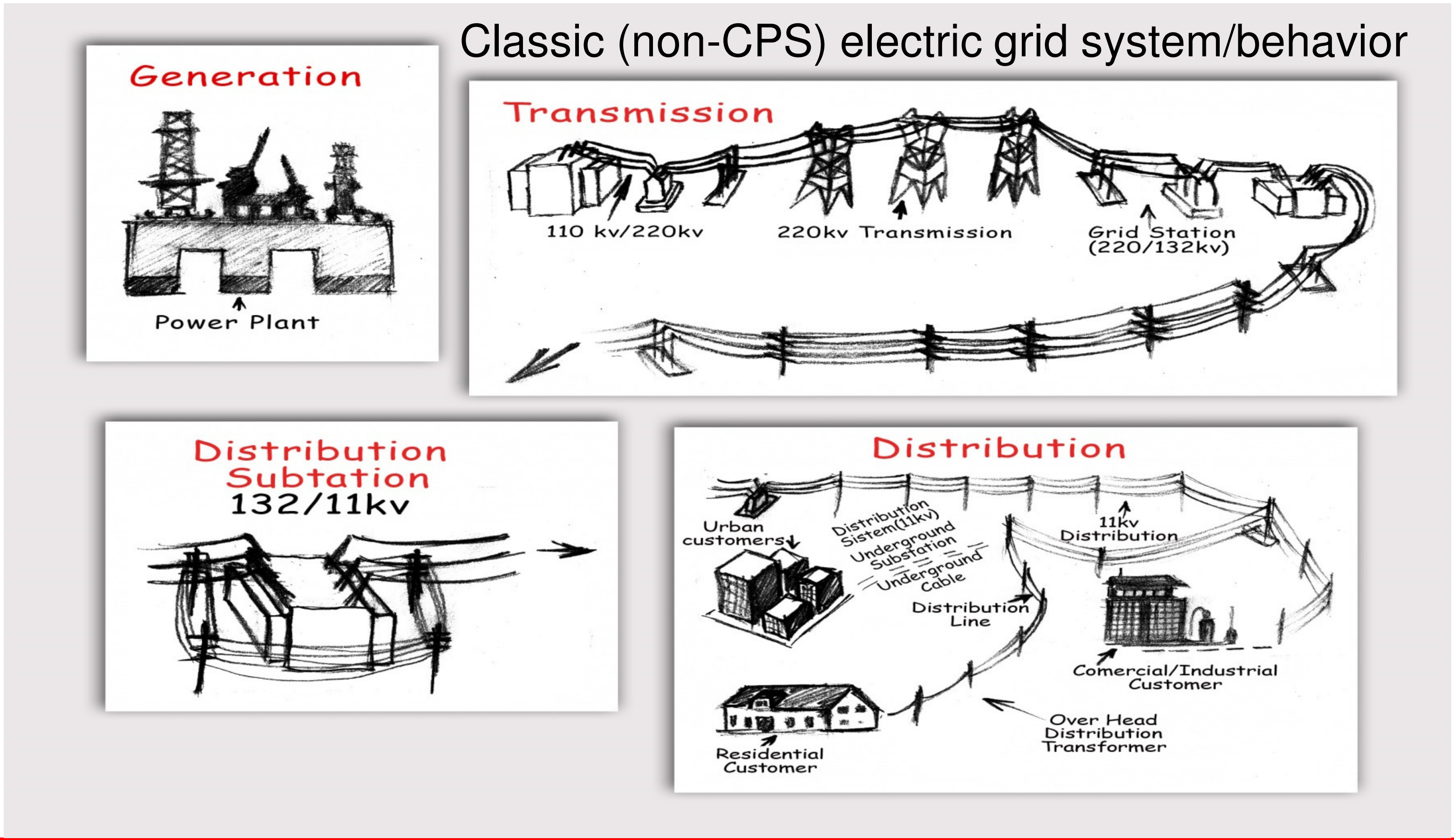
The current set of tools available for analysis cannot handle the complexity of CPS and thus are unable to predict system behavior with high degree of accuracy.

The consequences of these shortcomings:

Consider the electric power grid -- Massive failures leading to blackouts can be triggered by minor events.

Cyber-Physical Systems (CPS)

Classic (non-CPS) electric grid system/behavior



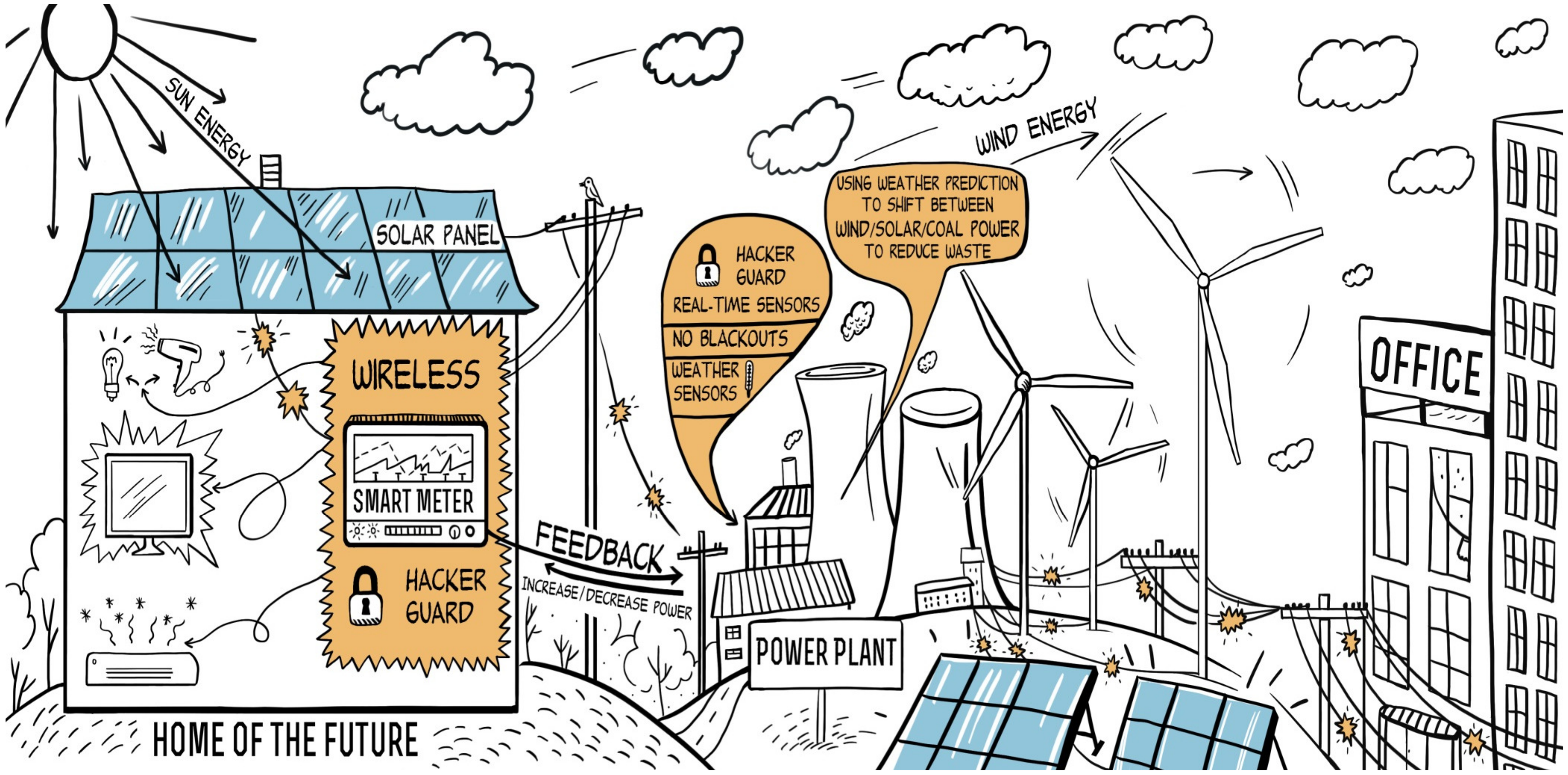
CS@UH

Cyber-Physical Systems (CPS)

- In a CPS, wireless/wired smart meters measuring real-time electricity usage and historical data (state) feedback (communication) to the generation station to better manage and distribute electricity.
- Current and predicted weather condition data can also further inform the decision-making in where to distribute electricity (very hot or very cold weather increase electricity demand).
- There is also a need to guard against intrusion into the system.
- Advocate formal verification to ensure satisfaction of safety properties.

CS@UH

Cyber-Physical Systems (CPS)

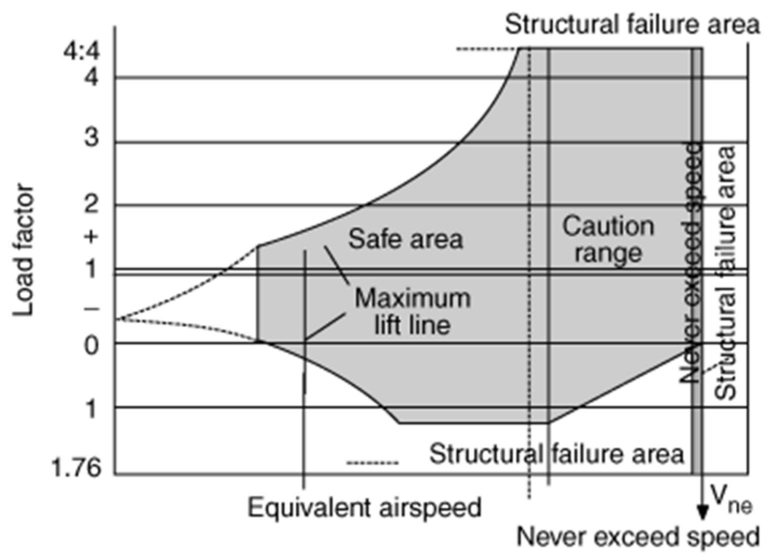


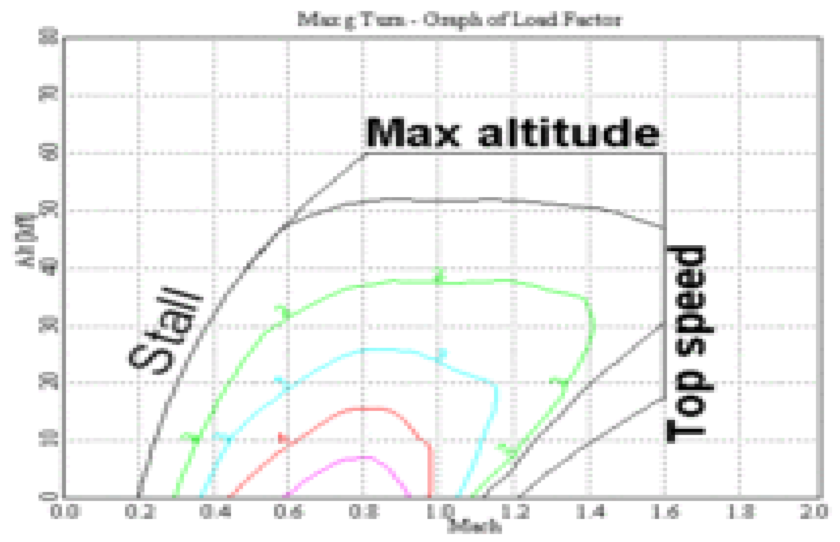
Cyber-Physical Systems (CPS)

Example 2: Imagine an airplane that refuses to crash. While preventing all possible causes of a crash is not possible, a well-designed flight control system can prevent certain causes. The systems that do this are good examples of cyber-physical systems.

Cyber-Physical Systems (CPS)

For example, some airplanes use a technique called flight envelope protection to prevent a plane from going outside its safe operating range, and prevent a pilot from causing a stall.

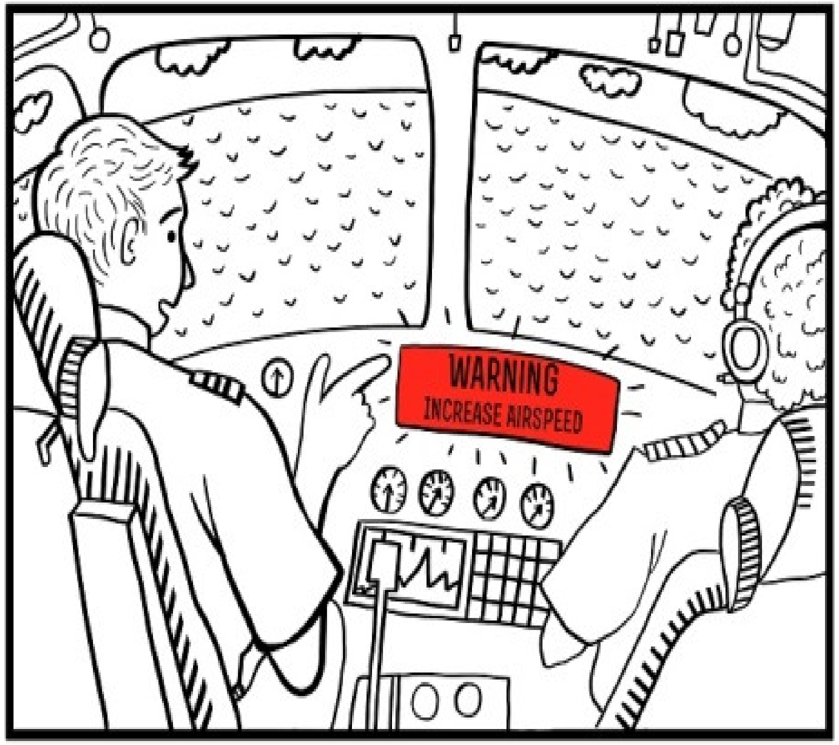


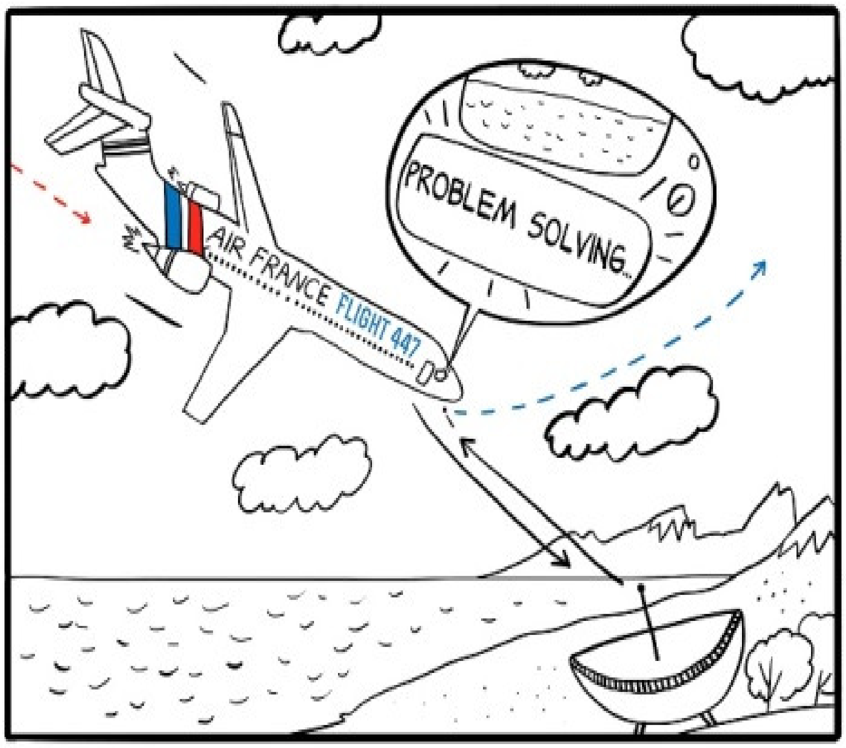


Cyber-Physical Systems (CPS)

- The embedded control system can over-ride erroneous operation that would lead to an accident.

http://en.wikipedia.org/wiki/Air_France_Flight_447





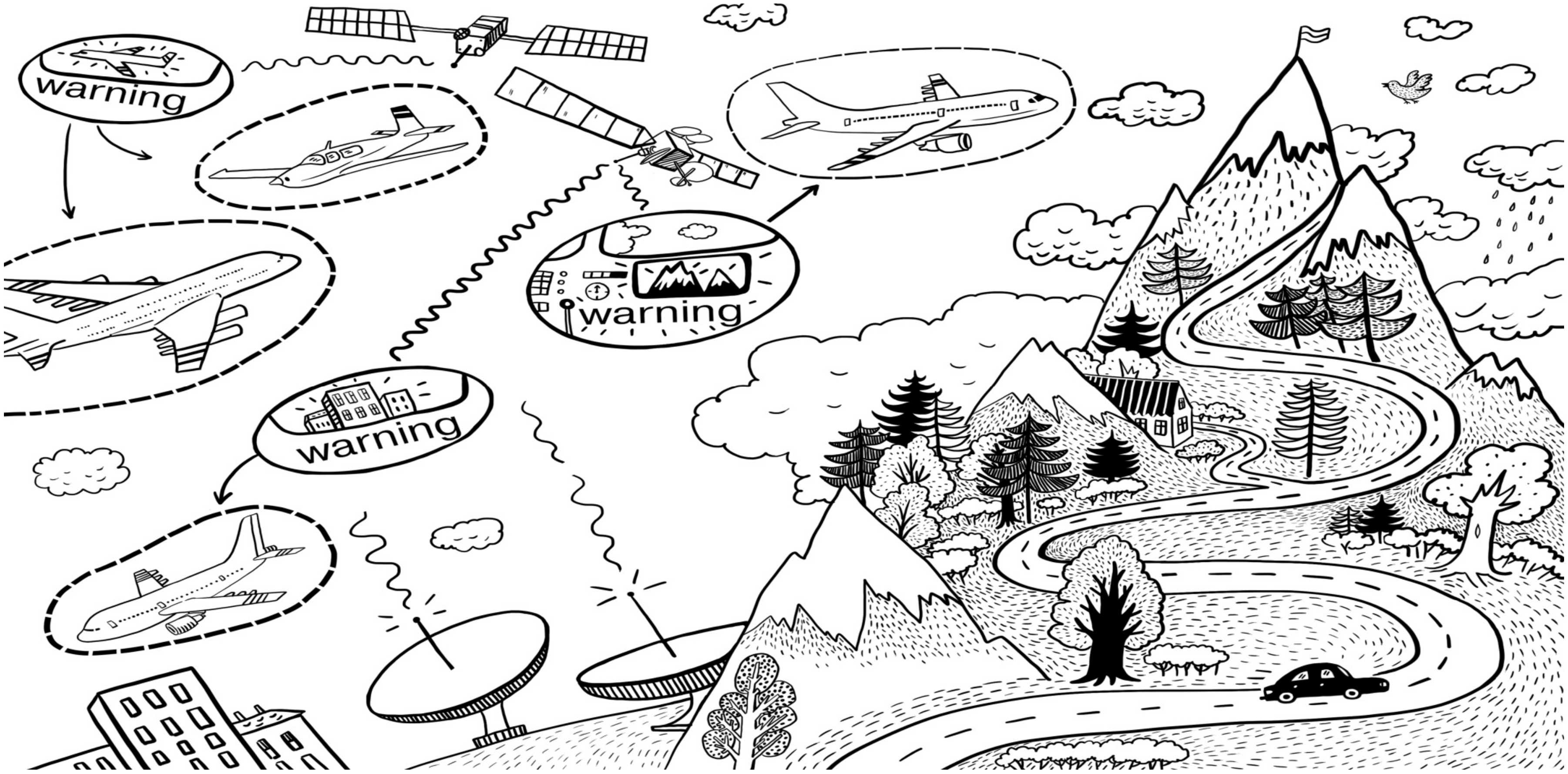
CS@UH

Cyber-Physical Systems (CPS)

- The concept of flight envelope protection could be extended to prevent other causes of crashes. For example, the soft walls system proposed by Prof. Edward Lee, if implemented, would track the location of the aircraft on which it is installed and prevent it from flying into obstacles such as mountains.
  - E. A. Lee, "Soft Walls - Modifying Flight Control Systems to Limit the Flight Space of Commercial Aircraft," EECS Department, University of California, Berkeley, Tech. Rep. UCB/ERL M01/31, 2001.

CS@UH

Cyber-Physical Systems (CPS)

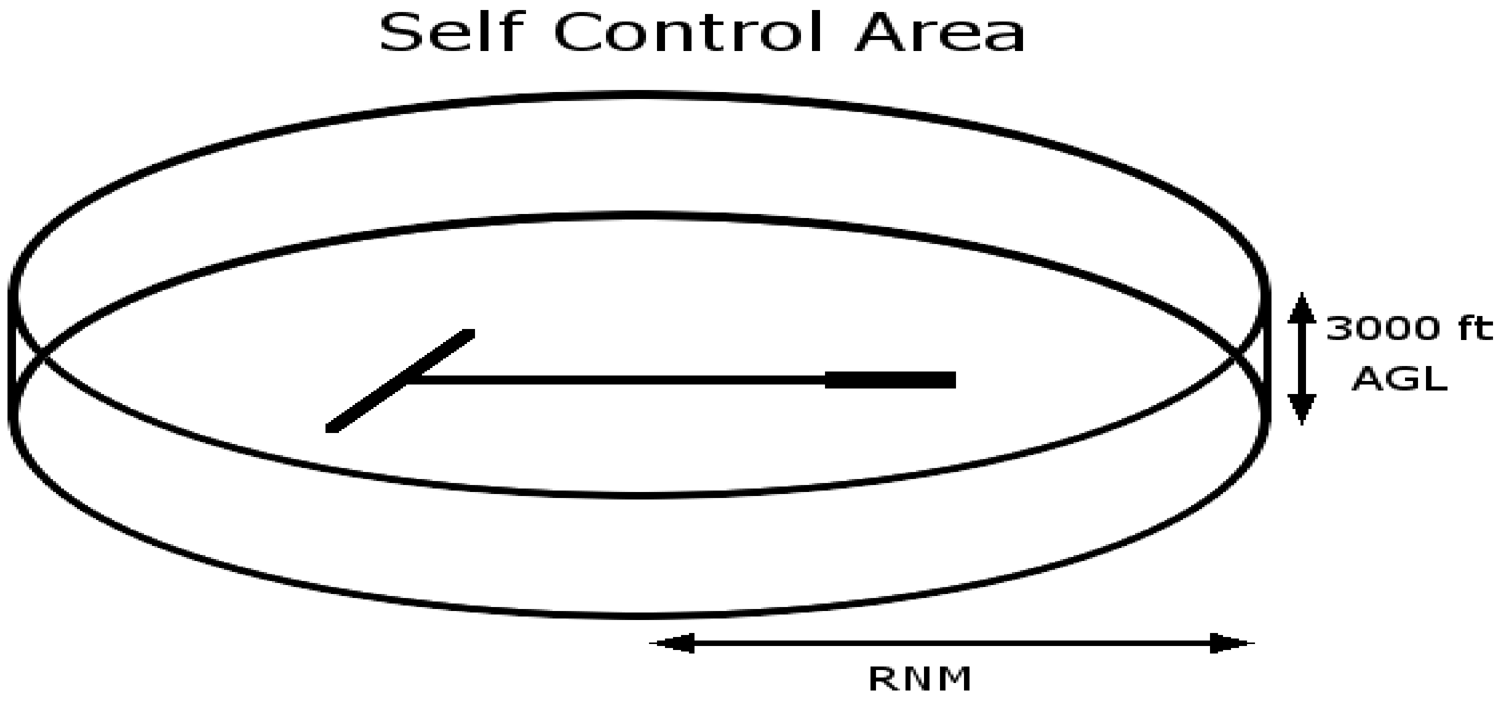


Cyber-Physical Systems (CPS)

- One of the key goals in our research is to develop the core tools that can be used to facilitate the analysis, design and engineering of highly-complex systems.
- With such tools, we can ensure that these systems are reliable, predictable, efficient, secure and resilient to multiple points of failure, and hence that their operation and safety can be depended upon with a high degree of confidence.
- We advocate formal verification to ensure safety of CPS's, but their complexity requires further research in verification tools.

Cyber-Physical Systems (CPS)

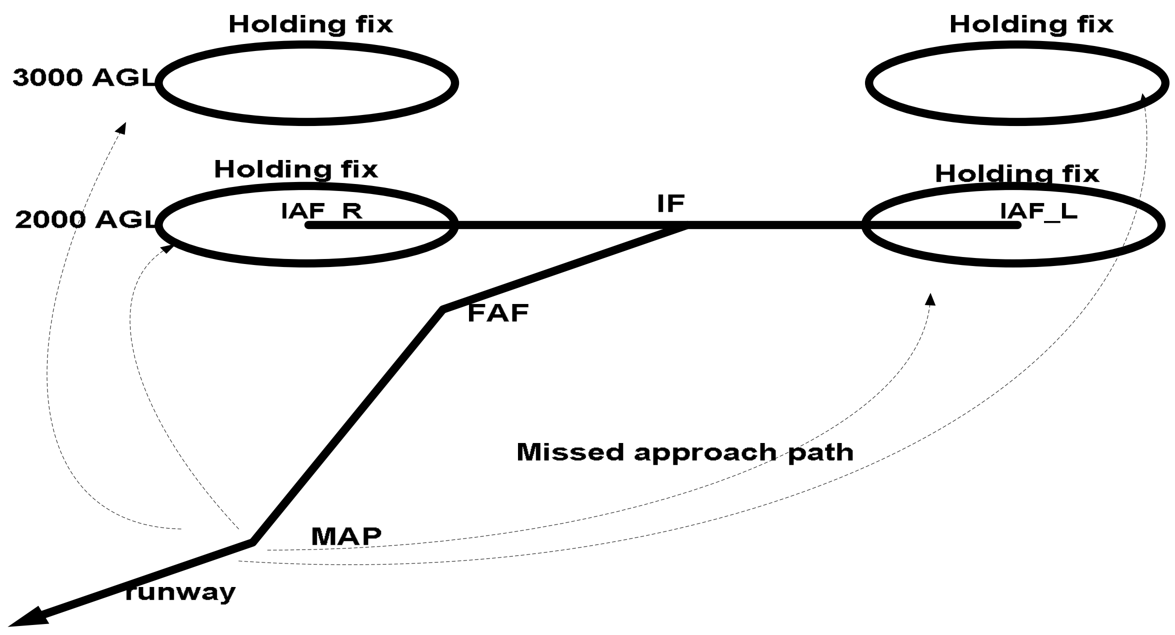
Small Aircraft Transportation System (SATS)



Self Control Area Airspace Volume

Cyber-Physical Systems (CPS)

Small Aircraft Transportation System (SATS)



3-D View of the SCA

Cyber-Physical Systems (CPS)

Small Aircraft Transportation System (SATS)

$$
\begin{array}{l}
\forall i @(\uparrow T_1\_ReqSeqNum\ , i) < @(\ T_1\_Approach\_IAF2\_L\ ,\\
i)\ -\ 5\ \wedge\\
@(\downarrow T_1\_ReqSeqNum, i) < @(T_1\_Approach\_IAF2\_L, i) \wedge\\
@(\uparrow T_1\_ReqSeqNum\ , i)\ < @(\uparrow T_2\_ReqSeqNum\ ,\ i) \wedge\\
@(\uparrow T_2\_ReqSeqNum\ , i) <\ @(\ T_2\_Approach\_IAF2\_R\ ,\\
i)\ -\ 5\ \wedge\\
@(\downarrow T_2\_ReqSeqNum, i) < (\ T_2\_Approach\_IAF2\_R, i) \wedge\\
@(\uparrow T_2\_ReqSeqNum, i) < @(\uparrow T_3\_ReqSeqNum\ , i) \wedge\\
@(\uparrow T_3\_ReqSeqNum\ ,\ i) < @(\ T_3\_Approach\_IAF3\_L\ ,\\
i)\ -\ 5\ \wedge\\
@(\downarrow T_3\_ReqSeqNum\ , i) < @(\ T_3\_Approach\_IAF3\_L\ , i) \wedge\\
@(\uparrow T_3\_ReqSeqNum\ , i) < @(\uparrow T_4\_ReqSeqNum\ , i) \wedge\\
@(\uparrow T_4\_ReqSeqNum\ , i)\ <\ @(\ T_4\_Approach\_IAF3\_R\ ,\\
i)\ -\ 5\ \wedge\\
\quad @(\downarrow T_4\_ReqSeqNum, i) < (\ T_4\_Approach\_IAF3\_R, i)
\end{array}
$$

- We will introduce RTL-based formal verification later in the tutorial.

Functional Reactive Programming

- Priority-based Functional Reactive Programming (P-FRP)
- P-FRP provides real-time guarantees using static priority assignment
- Higher-priority tasks preempt lower-priority ones; preempted tasks are aborted
- Multi-version commit model of execution
- Atomic execution – “all or nothing” proposition
- Execution different from ‘standard’ models

Other Examples of Functional Programming (FP) Languages:

- Haskell
- Atom - Domain Specific Language in Haskell
- Erlang - Developed at Ericsson for programming telecommunication equipment
- Esterel - Designed for reactive programming
- F# - Developed by Microsoft; available as a commercial platform

The Haskell Functional Programming Language

- The GHC compiler/interpreter uses a round-robin scheduler for Haskell threads
- No thread priorities yet for forkIO threads in GHC

A simple ABS example in Haskell:

```
import Control.Concurrent  -- For threading facilities: “forkIO”, “newEmptyMVar”, “takeMVar”, and “putMVar”.
import Control.Monad (forever) -- For "forever"

-- Type aliases help keep track of what values we are talking about.

type WheelSpeed    = Double  -- A "double" floating point value
type AverageSpeed = Double

-- | The ABS can either forcibly release, focibly engage, or stay neutral for each wheel.
-- The deriving clause creates the obvious Show instance for this ADT.

data BrakeSignal = Release | Engage | Neutral
   deriving Show
```

The Haskell Programming Language

```
-- | Compute the average speed by dividing the sum of the list of speeds by the length.
-- fromIntegral is there to convert the result of length (Int) into a Double
-- Note, this will traverse the list twice, ineffcient for vehicles with millions of wheels.
averageSpeed :: [WheelSpeed] -> AverageSpeed
averageSpeed speeds = sum speeds / (fromIntegral $ length speeds)

-- | This algorithm may be much more complicated, but the basic idea is present.
-- Given the average speed and a particular wheel speed, check to see if we are
-- within 5 (mph, kph, m/s, whatever) of the average. If we are below the minimum
-- send the release signal to the brakes. If we are within 5, remain neutral, otherwise
-- send a signal to engage the brakes.
ecuHelper :: AverageSpeed -> WheelSpeed -> BrakeSignal
ecuHelper average speed | speed < min = Release
                        | speed < max = Neutral
                        | otherwise = Engage
  where
    min = average - 5
    max = average + 5
```

The Haskell Programming Language

```
-- A list of wheel speeds are averaged and the speed of each wheel compared to it
-- and converted into an ABS signal.

ecu :: [WheelSpeed] -> [BrakeSignal]
ecu speeds = let avgS = averageSpeed speeds
             in map (ecuHelper avgS) speeds
-- The Main Function:
-- The first thread does all printing whenever information becomes available to the ABS.
-- The second thread waits for sensor data, sends it to the ECU and stores the result in the
ABS
-- The main thread waits for someone to type in a list of numbers and sends it to the “sensors”.

main = do
    -- Print initial instructions and an example.
    print "Enter wheel speeds: [45,46,45,47]"
    -- Create sensors represented as a list of WheelSpeeds, i.e., Doubles.
    sensors <- newEmptyMVar
    -- Create an ABS represented as as list of BrakeSignals.
    abs <- newEmptyMVar
```

The Haskell Programming Language

```
-- This thread handles all printing to the console.
  forkIO $ forever $ do
      putStr "Enter a Speed: "
      absOutput <- takeMVar abs          -- Read ABS status
      print absOutput                    -- Print ABS status.

  -- This thread is the ABS. It reads the sensors, then processes the data and updates the ABS.
  forkIO $ forever $ do
      sensorData <- takeMVar sensors              -- Read sensors.
      let brakeCommands = ecu sensorData          -- Calculate brake response
      putMVar abs brakeCommands                   -- Update ABS status.

  -- The main thread simply waits for users to enter data which is then written to the sensors.
  forever $ do
      input <- getLine                            -- User enters a line of text
      let wheelSpeedData = read input             -- Text is read as [WheelSpeed]
      putMVar sensors wheelSpeedData              -- wheelSpeedData is written to sensors
```

The Haskell Programming Language

To run program from the command prompt
In GHCi, type

```
*Main> main
"Enter wheel speeds: [45,46,45,47]"
Enter a Speed: [45,45,45,55]
[Neutral,Neutral,Neutral,Engage]

Enter a Speed: [45,45,45,45]
[Neutral,Neutral,Neutral,Neutral]

Enter a Speed: [45,45,55,45]
[Neutral,Neutral,Engage,Neutral]
```

The idea is that you keep entering new sensor data.
The system calculates the new ABS signals to send to the vehicle.
The session should look like this:

Functional Reactive Programming (FRP)

- Functional reactive programming (FRP) is a style of functional programming where programs are inherently stateful, but automatically react to changes in state.
- FRP allows intuitive specification and formal verification of safety-critical behaviors, thus reducing the number of defects during the design phase, and the stateless nature of execution avoids the need for complex programming involving synchronization primitives.
- Therefore, the program remains an algebraic description of system state, with the task of keeping the stated (unidirectional) relationships in sync left to the *language*.

Functional Reactive Programming (FRP)

- FRP is essentially (though rarely acknowledged as such) an extension to the old idea of dataflow programming.

- A key difference is that FRP supports higher-order functions, and modern FRP systems are generally well-integrated into broader languages.

- The original (modern) FRP work was built in the context of Haskell, though major FRP systems have also been built atop many other languages.

Functional Reactive Programming (FRP)

- Type-safe programming language
- Discrete and Continuous aspects
- Transactional model prevents priority inversion
- Synchronization primitives not required
- Ideal for parallel execution

Basic Abstractions

- FRP divides inputs into two basic classes:
  - Behaviors or signals: Functions of time.
  - Events: Temporal sequences of discrete values.
- An FRP language must include a means of altering or replacing a program based on event occurrences - this is the basis of FRP's reactivity.
- These abstractions may be reified in an FRP language or may form the basis of other abstractions, but they must be present.

Functional Reactive Programming (FRP)

- **Weaknesses**
  - FRP is still relatively new and the design space is still being explored.
- **Strengths**
  - FRP makes writing reactive programs easier to reason about and to avoid common errors
  - It is easier to expand and create new behaviors. Once the program becomes more complex, **forkIO** and multiple threads might start interfering with each other, or there would be odd interleaving, blocking, or other bad concurrency behavior.

FRP is still Haskell. It is just a different style.

Functional Reactive Programming (FRP)

- Using FRP makes the controllers (the computational components of CPS) more amenable to analysis and verification.
- We can treat components (programed in FRP) as mathematical functions, which can be composed and synthesized to form a much larger, complex system.
- More resistant to faults since there are no intermediate states. They can be connected and composed more easily.
- With procedural programs, there are more uncertainties, for example, intermediate states if faults/interruptions occur that need to be specified/modeled, making developing a CPS with guaranteed safety and response much more complex and potentially intractable.
- In the electric grid example, different generating stations have control components which analyze real-time data from smart meters, weather data, and industrial plants' energy usage to determine optimal or near-optimal generation and distribution of electricity.

Priority-based FRP (P-FRP)

- P-FRP aims to improve the programming of reactive real-time systems.
  - Supports assignment of different priorities to events
  - Benefits of using P-FRP over the imperative styles
    - P-FRP allows the programmer to intuitively describe safety-critical behaviors of the system, thus lowering the chance of introducing bugs in the design phase.
    - Its stateless nature of execution does not require the use of synchronization primitives like mutexes and semaphores, thus reducing the complexity in programming.

Priority-based FRP (P-FRP)

- **To preserve data consistency, shared resources must be accessed in mutual exclusion:**

Priority-based FRP (P-FRP)

- **However, mutual exclusion introduces extra delays:**

Priority-based FRP (P-FRP)

Example: The Car Controller

* C = worst case execution time
* T = (sampling) period = D (deadline)

- Speed Measurement: C=4ms, T=20ms, D=20ms
- ABS control: C=10ms,T=40ms, D=40ms
- Fuel injection: C=40ms,T=80ms, D=80ms
- Other software with soft deadlines, audio, air condition, etc.

Try any method to schedule the tasks

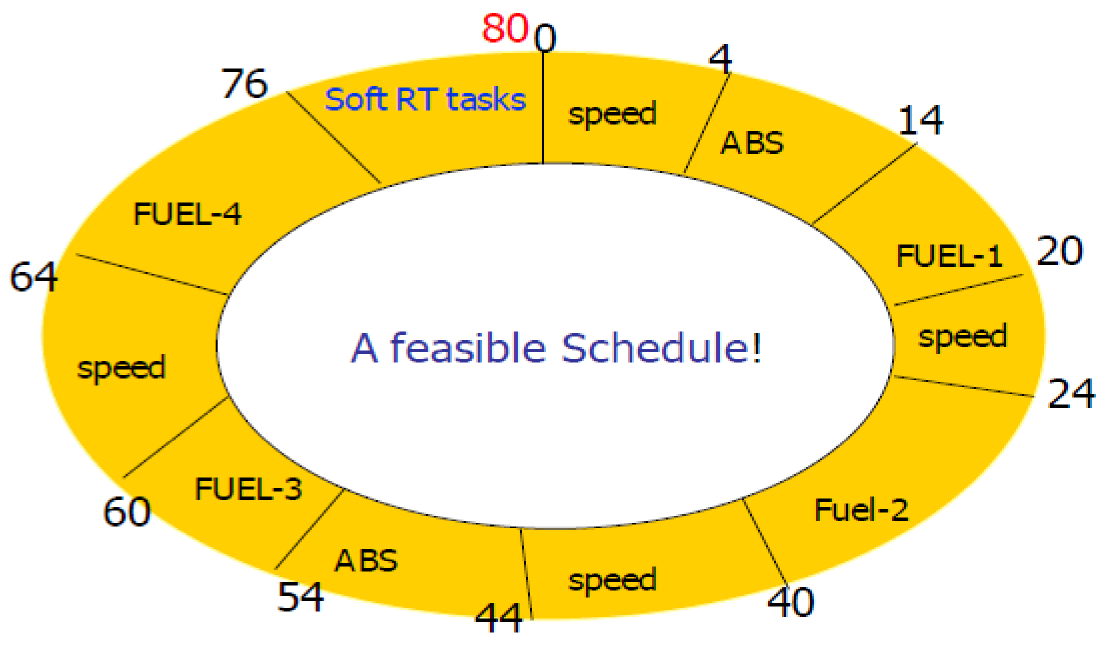
Priority-based FRP (P-FRP)

Static cyclic scheduling: + and –

- Deterministic: predictable (+)
- Easy to implement (+)
- Inflexible (-)
  - Difficult to modify, e.g., adding another task
  - Difficult to handle external events
- The table can be huge (-)
  - Huge memory-usage
  - Difficult to construct the time table

Priority-based FRP (P-FRP)

The Car Controller (Time table constructed with EDF)



Can use the Stack Resource Policy (SRP) or the Priority Ceiling Protocol (PCP) for concurrency control.

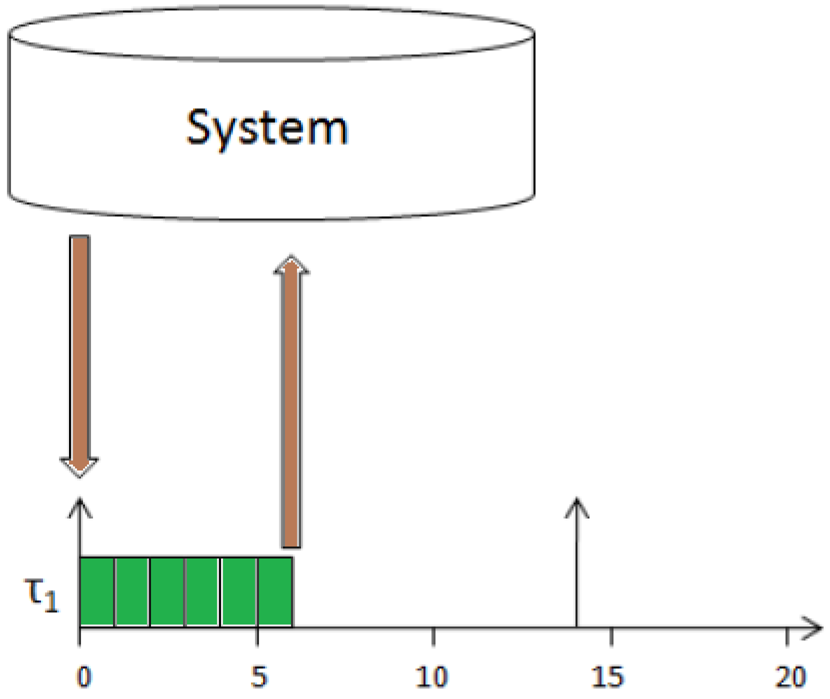
- Inheritance algorithms are complicated and difficult to program correctly.

Priority-based FRP (P-FRP)

- **In P-FRP, the scheduling model is called** Abort-and-Restart **(ANR)**
  - **Copy and restore operations**
    - **To allow for correct restarting of handlers, compilation is extended to generate statements that store variables modified in an event handler into fresh *temporary* (or *scratch*) variables in the beginning of the handler while interrupts are turned off, and to restore variables from the temporary variables at the end of the handler while interrupts are turned off.**

Priority-based FRP (P-FRP)

τ_1 starts at 0 and it copies a set of data from the system. After six ticks, its work is done and then it restore the updated data into the system.



Copy-and-Restore Operation

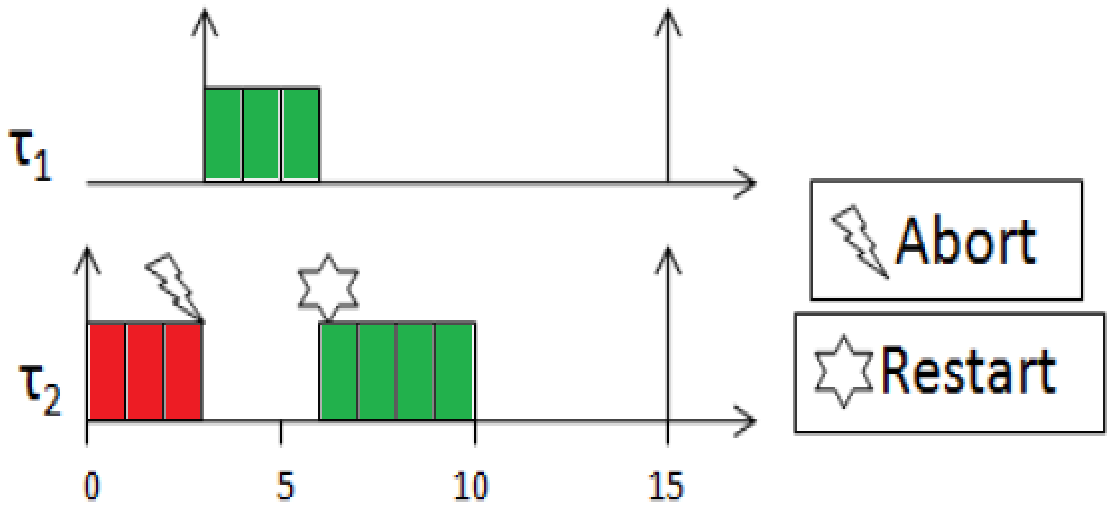
Priority-based FRP (P-FRP)

- **The Abort-and-Restart (ANR) Scheduling Model**
  - The idea of the ANR model is that a lower-priority task is aborted when a higher priority task arrives into the system. Once the higher-priority task is done, the lower priority task restarts as new.

Priority-based FRP (P-FRP)

Task	Period	WCET
τ_1	12	3
τ_2	15	4

(τ_1 has the highest priority)



Priority-based FRP (P-FRP)

- **Advantages of Abort-and-Restart (ANR)**
  - A simpler programming model
  - Tasks execute atomically so no task is blocked by another task
    - The priority inversion problem is removed
    - No overheads caused by priority inheritance
    - Closer adherence to priority scheduling

Priority-based FRP (P-FRP)

Limited Work on Scheduling and Schedulability Analysis

- While there is an extensive understanding of the theory and proof-carrying capability of functional programs and their reactive versions, relatively little work is available on the scheduling of primitives in the corresponding imperative code.
- Also, performance studies of the computational platforms on which these functional programs execute are mostly absent.

Priority-based FRP (P-FRP)

- The worst-case response time of a task is the length of the longest interval from a release of that task till its completion.
- With ANR, interference from higher-priority tasks induces both an interference cost and an abort cost on the response time of the preempted lower-priority task.
- Current focus is on response time analysis with abstract memory and I/O access times. Next challenges include accounting for precise memory and I/O access times.

Priority-based FRP (P-FRP)

- Response time analysis is an **exact** schedulability test to calculate the worst-case response time of a task which includes the time of interference from other higher priority tasks and blocking from lower priority tasks.
- RTA is **not** exact unless **blocking** is exact - which it is not. If the worst-case response time of a task is longer than its deadline (***D***), it means the task will not meet its deadline. The opposite situation is that if the worst-case response time of the task is less than or equal to its deadline, the task will meet its deadline.
- The analysis can be applied for ***D* = *T*** (task's period), ***D* < *T***, or ***D* > *T***.

CS@UH

Priority-based FRP (P-FRP)

Response time Analysis for ANR

- For the highest-priority task, its worst response time will be equal to its own computation time, that is $R = C$.

- If task j has the highest arrival rate, then the execution time of a task i cannot exceed $T_j - C_j$ or task i will suffer **interference** (I) and **aborts** (α). So for a general task i :

$$R_i = C_i + I_i + \alpha_i$$

Priority-based FRP (P-FRP)

Interference Cost

- If the execution time of some task ***i*** exceeds $\boldsymbol{T_j - C_j}$, then task ***i*** will **never** be able to complete execution.
- A simple expression for obtaining this Interference Cost is using the ceiling function:

$$I_i = \left\lceil \frac{R_i}{T_j} \right\rceil \cdot C_j$$

Priority-based FRP (P-FRP)

Maximum Interference

- Each task of higher-priority is interfering with task ***i***, *and so:*

$$I_i = \left\lceil \frac{R_i}{T_{i+1}} \right\rceil \cdot C_{i+1} + \left\lceil \frac{R_i}{T_{i+2}} \right\rceil \cdot C_{i+2} + \ldots + \left\lceil \frac{R_i}{T_n} \right\rceil \cdot C_n$$

- This gives us the following equation:

$$I_i = \sum_{j=i+1}^{n} \left\lceil \frac{R_i}{T_j} \right\rceil \cdot C_j$$

Priority-based FRP (P-FRP)

Maximum Abort Costs

- Each higher-priority task is interfering with task *i*, so the maximum Abort Costs are as follows*:*

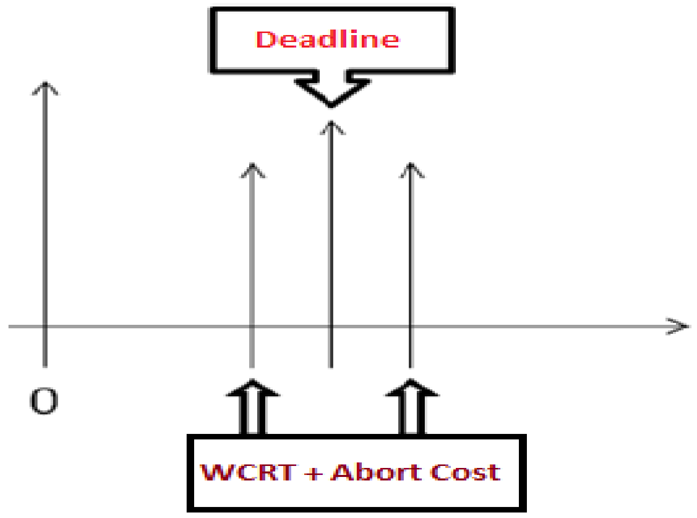
$$\alpha_i = \sum_{j=i+1}^{N} \left\lceil \frac{R_i}{T_j} \right\rceil \cdot \max_{k=i}^{j-1} C_k$$

- C_k is the maximum execution time between *i* and the highest-priority task.

Priority-based FRP (P-FRP)

Maximum Abort Costs

- The maximum abort cost equation is sensible and simple but overly pessimistic. Therefore, the test is said to be **sufficient** but not **necessary**.



Priority-based FRP (P-FRP)

- Abort-and-Restart with a limit on the number of aborts

$$R_i \leq C_i + I_i + B_i + \alpha_i$$

$$I_i = \sum_{j=i+1}^{n} min(MaxNumberAborts, \left\lceil \frac{R_i}{T_j} \right\rceil) \cdot C_j$$

$$B_i = \max_{j \in LowerPriority(i))} C_j \quad (Blocking\ Costs)$$

$$\alpha_i = MaxNumberAborts \cdot \sum_{j=i}^{n-1} C_j$$

Priority-based FRP (P-FRP) Example

Antilock braking system in a car is a simple example of an embedded hard real-time system with real-time constraints.

The ABS is expected to release a vehicle's brakes, preventing dangerous wheel locking, in a predictably short time frame.

ABS uses a kind of an Abort-and-Restart Scheme.

Kaleb R. Christoffersen and Albert M. K. Cheng, ``Model-Based Design: Anti-lock Brake System with Priority-Based Functional Reactive Programming,'' submitted to RTSS WIP 2013.

Priority-based FRP (P-FRP)

Anti-Lock Brake Types

ABS uses different schemes depending on the type of brakes in use.

- **Four-channel, four-sensor ABS** (the best scheme) - there is a speed sensor on all 4 wheels and a separate valve for all four wheels. With this setup, the controller monitors each wheel individually to make sure it is achieving maximum braking force.
- **Three-channel, three-sensor ABS -** this scheme found often on pickup trucks. It has a speed sensor and a valve for each of the front wheels, with one valve and one sensor for both rear wheels.
- **One-channel, one-sensor ABS -** this system found also often on pickup trucks with rear-wheel ABS. It has one valve, which controls both rear wheels, and one speed sensor.

Priority-based FRP (P-FRP)

Example: ABS Controller

- Activities of an ABS control system
  1. C = worst case execution time
  2. T = (sampling) period = D (deadline)

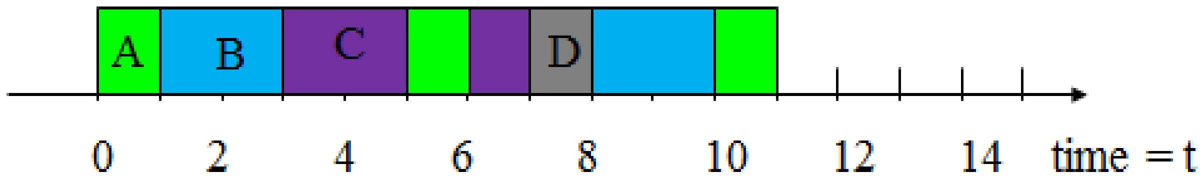
Prof. Cheng with undergrad students Mozahid and Kaleb

- (A) Car speed measurement: C= 1 ms, T= 5 ms
- (B) Wheel speed measurement: C= 2 ms,T=8 ms
- (C) Analysis and computation task : C= 3 ms,T=20 ms
- (D) Brakes (Abort (release) /Retry (pressure)) : C= 1 ms,T=25 ms

Priority-based FRP (P-FRP)

Example: ABS Controller - with RM scheduling

- The shortest repeating cycle / LCM = 200 ms



A's response time = 1 (Same as its own Computation Time)

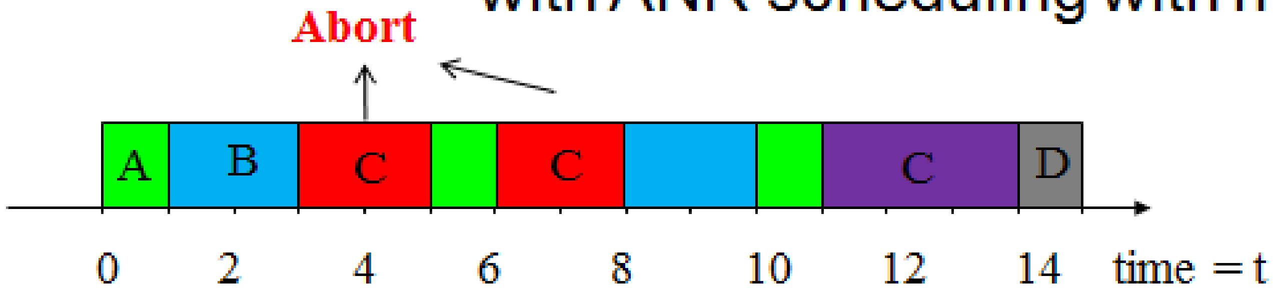
B's = 2 + 1 (time to execute task A) = 3

C's = 3 + 1 (A's) + 2 (B's) = 6

Priority-based FRP (P-FRP)

Example: ABS Controller –

with ANR scheduling with max 2 aborts.



A's response time = 1 (Same as its Computation Time)

B's = 2 + 1 (time to execute task A) = 3

C's = 3 + 1 (A's) + 2 (B's) = 14 < Deadline

Priority-based FRP (P-FRP)

Typically ABS includes

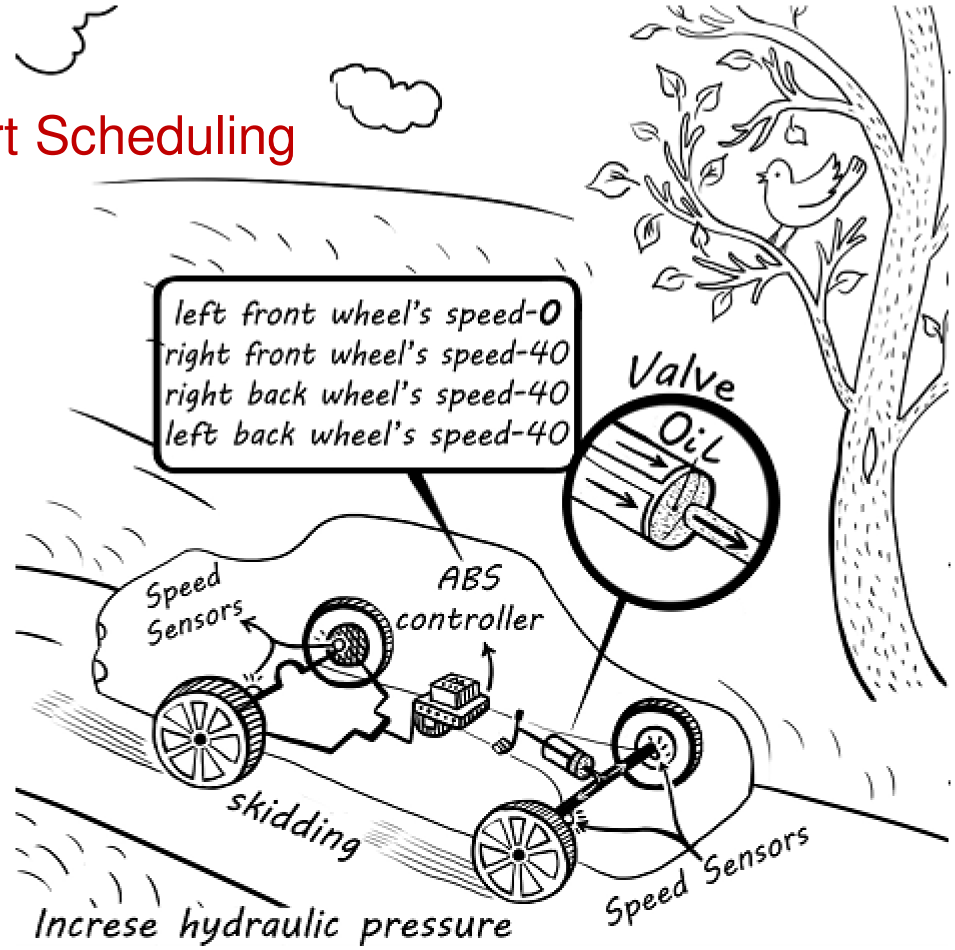
- Electronic control unit (ECU)
- Wheel speed sensors
- At least two hydraulic valves within the brake hydraulics

- The ECU constantly monitors the rotational speed of each wheel; if it detects a wheel rotating significantly slower than the others, a condition indicative of impending wheel lock, it actuates the valves to reduce hydraulic pressure to the brake at the affected wheel, thus reducing the braking force on that wheel; the wheel then turns faster.

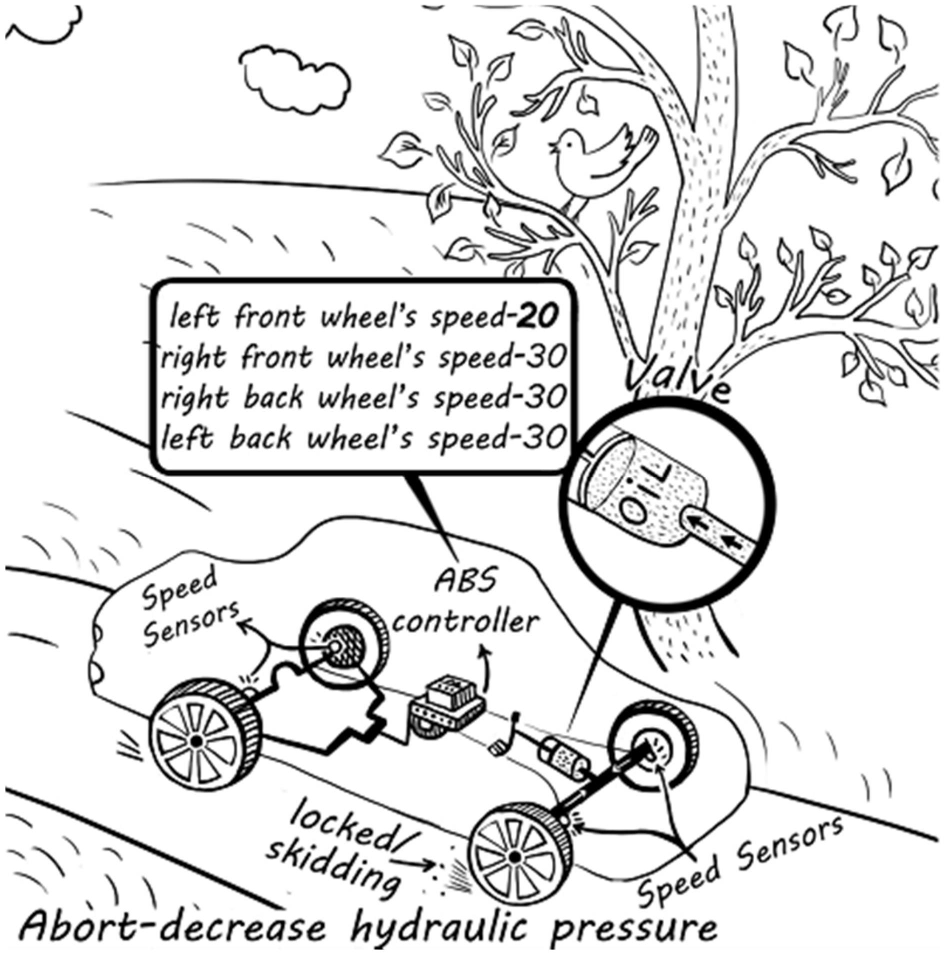
CS@UH

Priority-based FRP (P-FRP)

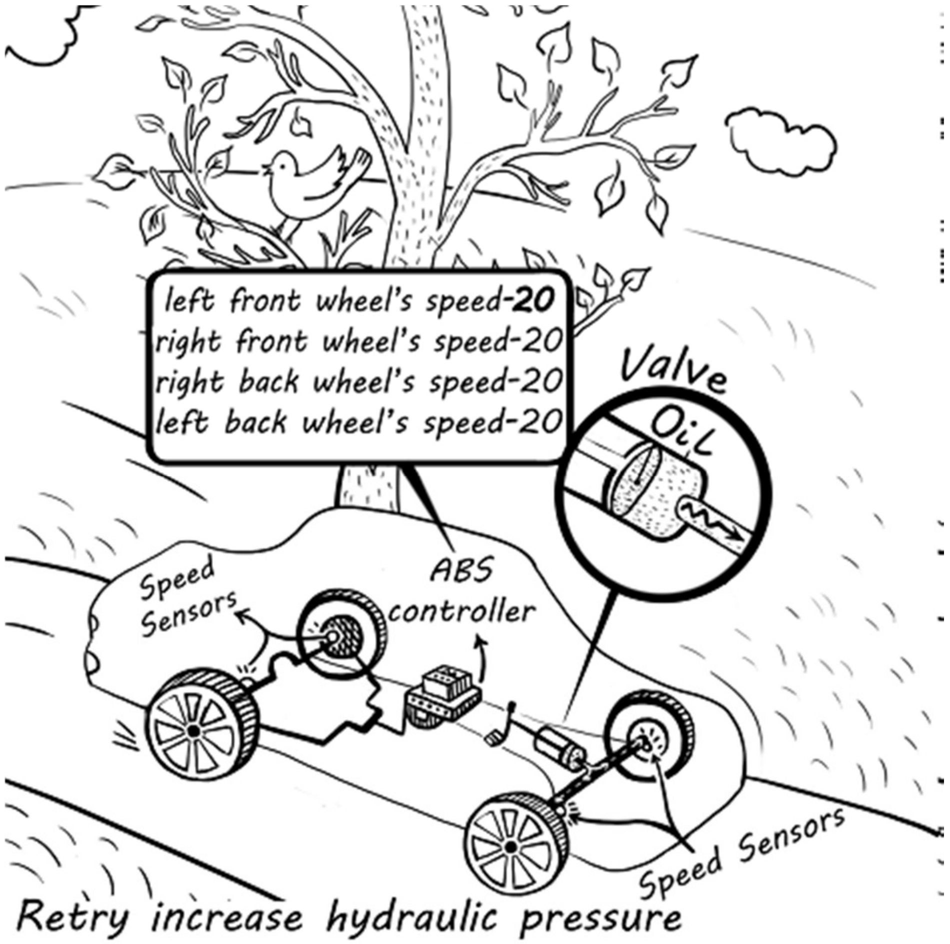
Abort-and-Restart Scheduling


left front wheel's speed-0
right front wheel's speed-40
right back wheel's speed-40
left back wheel's speed-40
Valve
Oil
Speed Sensors
ABS controller
skidding
Increse hydraulic pressure
Speed Sensors

Priority-based FRP (P-FRP)

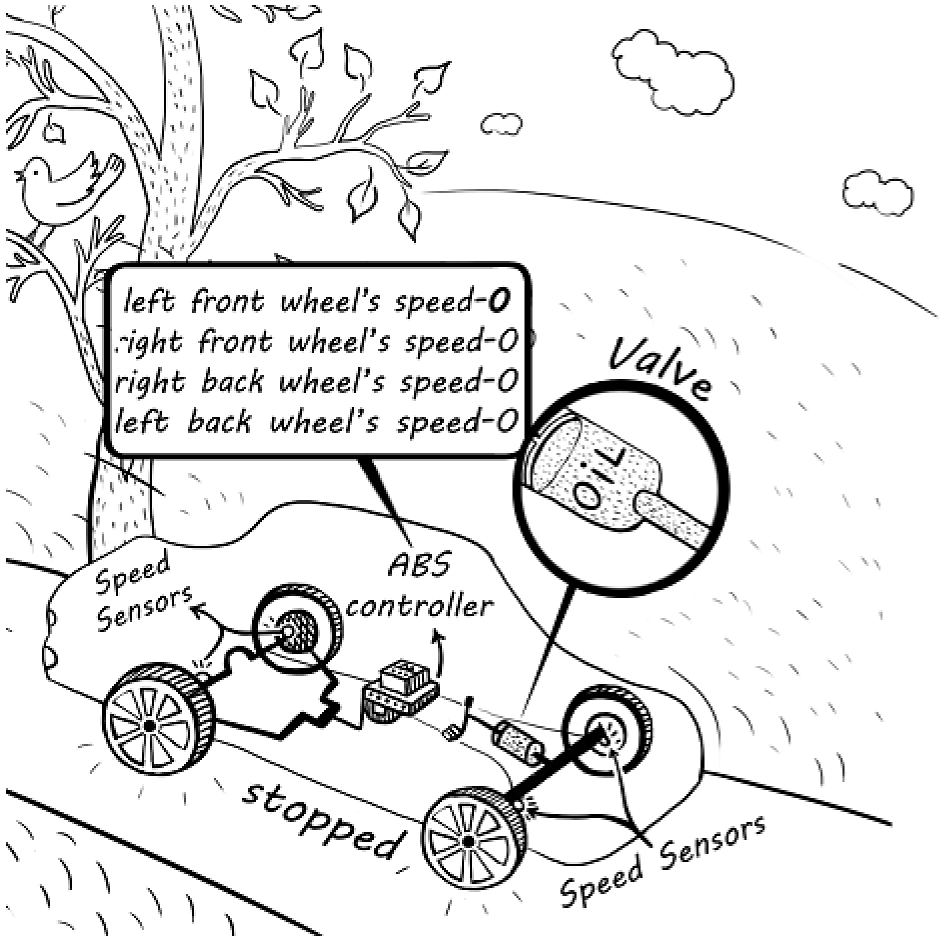

left front wheel's speed-20
right front wheel's speed-30
right back wheel's speed-30
left back wheel's speed-30
Valve
OIL
Speed Sensors
ABS controller
locked/ skidding
Speed Sensors
Abort-decrease hydraulic pressure

Priority-based FRP (P-FRP)

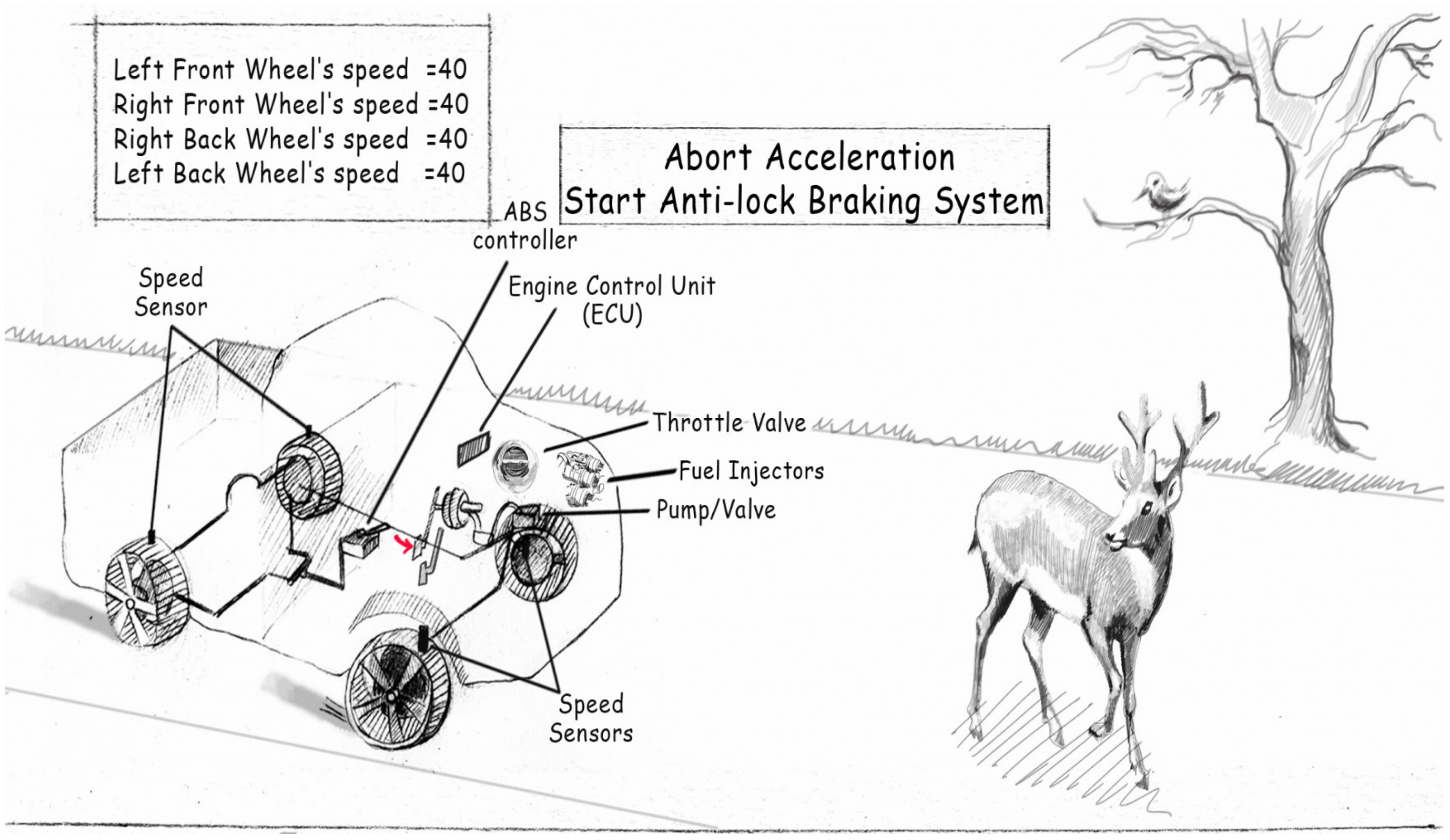

left front wheel's speed-20
right front wheel's speed-20
right back wheel's speed-20
left back wheel's speed-20
Valve
Oil
Speed Sensors
ABS controller
Speed Sensors
Retry increase hydraulic pressure

CS@UH

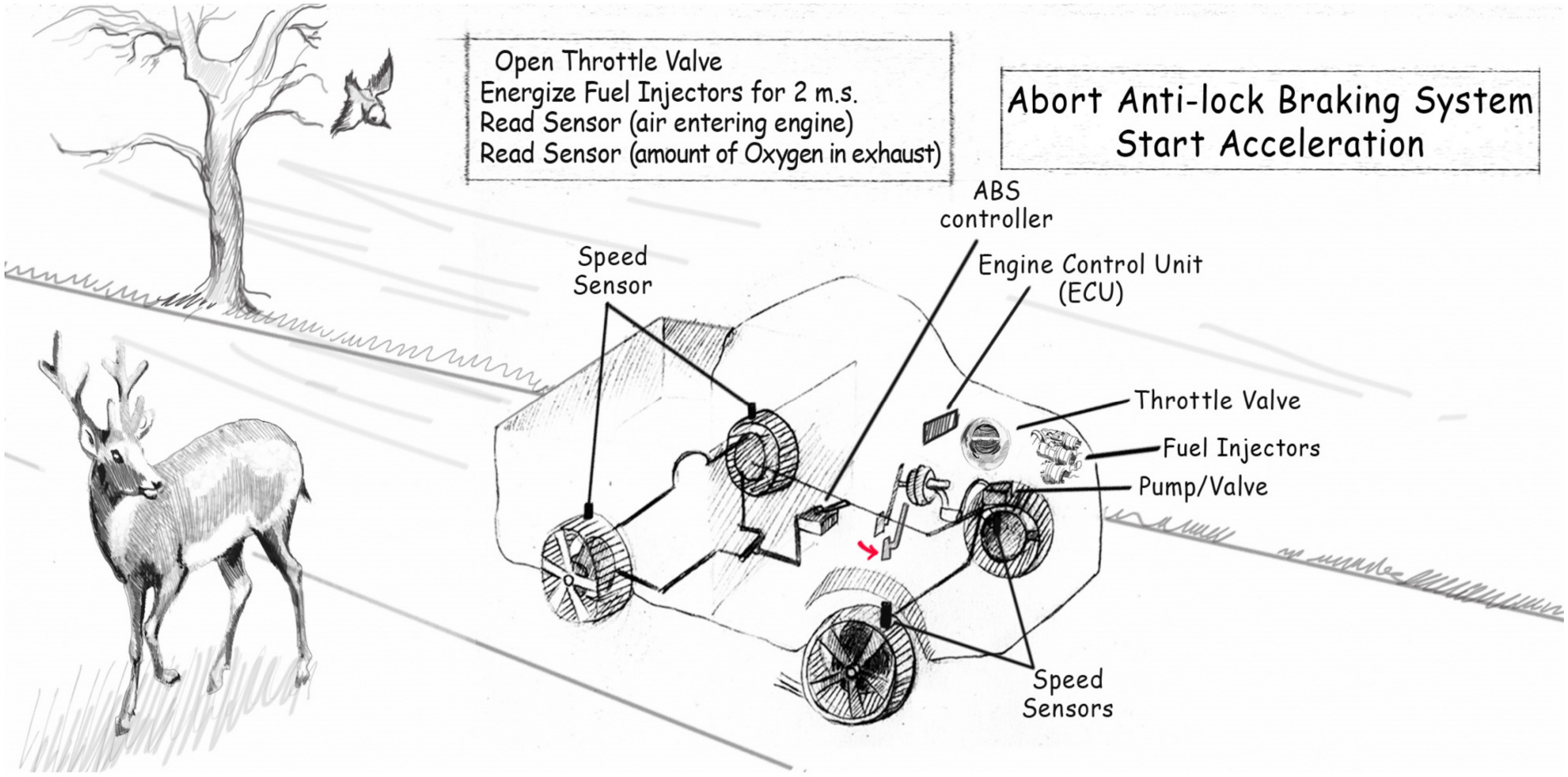
Priority-based FRP (P-FRP)


left front wheel's speed-0
right front wheel's speed-0
right back wheel's speed-0
left back wheel's speed-0
Valve
OIL
Speed Sensors
ABS controller
stopped
Speed Sensors

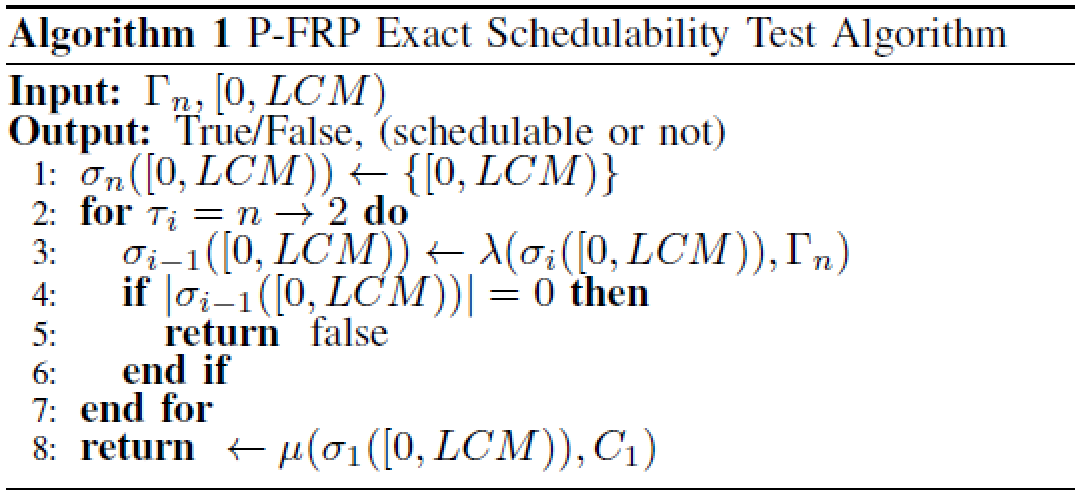
Priority-based FRP (P-FRP)


Left Front Wheel's speed =40
Right Front Wheel's speed =40
Right Back Wheel's speed =40
Left Back Wheel's speed =40
Abort Acceleration
Start Anti-lock Braking System
ABS controller
Speed Sensor
Engine Control Unit (ECU)
Throttle Valve
Fuel Injectors
Pump/Valve
Speed Sensors

Priority-based FRP (P-FRP)


Open Throttle Valve
Energize Fuel Injectors for 2 m.s.
Read Sensor (air entering engine)
Read Sensor (amount of Oxygen in exhaust)
Abort Anti-lock Braking System
Start Acceleration
ABS controller
Engine Control Unit (ECU)
Speed Sensor
Throttle Valve
Fuel Injectors
Pump/Valve
Speed Sensors

Priority-based FRP (P-FRP)



Algorithm 1 P-FRP Exact Schedulability Test Algorithm

Input: $\Gamma_n, [0, LCM)$

Output: True/False, (schedulable or not)

```
1: $\sigma_n([0, LCM)) \leftarrow \{[0, LCM)\}$
2: for $\tau_i = n \rightarrow 2$ do
3:     $\sigma_{i-1}([0, LCM)) \leftarrow \lambda(\sigma_i([0, LCM)), \Gamma_n)$
4:     if $|\sigma_{i-1}([0, LCM))| = 0$ then
5:         return false
6:     end if
7: end for
8: return $\leftarrow \mu(\sigma_1([0, LCM)), C_1)$
```

- On-line Schedulability Test returns the gap (the amount of execution time available) for the next lower-priority task.
- Precise (tight) timing characterization of the embedded controller software execution leads to faster physical system response compared with one designed without accurate controller timing analysis (and thus requires more tolerance of execution time variations).

Latest In-Progress Work

A Scratchpad Memory-Based Execution Platform for Functional Reactive Systems and its Static Timing Analysis

Zeinab Kazemi, **Albert M. K. Cheng**

IEEE RTAS WIP 2015, Seattle, WA

Worst Case Execution Time (WCET)

- Maximum length of time for a task to execute on a specific platform.
- Why is it necessary to compute WCET in P-FRP?
  - To increase responsiveness
    - To find a tight bound for response time analysis
  - To avoid unnecessary micro-architecture costs
    - Optimizing resources by a finding a tighter bound

Cache For Real-Time Systems

- A good solution to speed up for most programs and average execution time analysis
- Unpredictable cache miss penalty
- Timing predictability is required for Hard real-time systems
- Not suitable for P-FRP systems

Scratchpad Memory (SPM) vs. Cache

- Cache
  - Suitable for average execution time analysis
  - Write-back to memory
- SPM
  - Programmer can specify which data should be in SPM
  - Commit to memory
  - DMA
  - Suitable for WCET derivation
  - Predictable latency

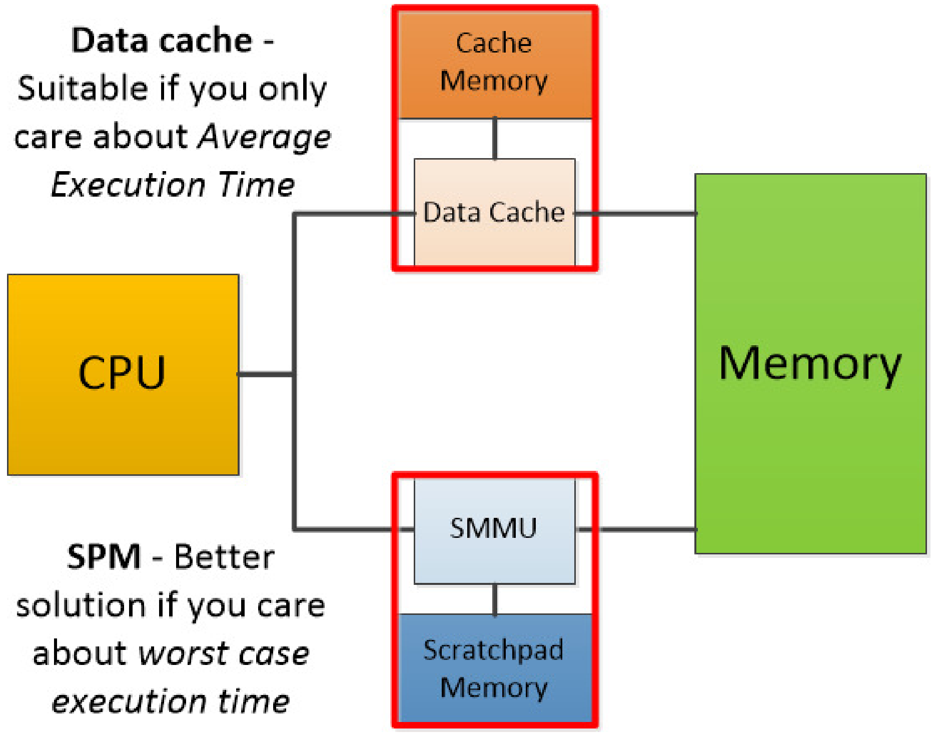

Data cache - Suitable if you only care about *Average Execution Time*
Cache Memory
Data Cache
CPU
Memory
SMMU
Scratchpad Memory
SPM - Better solution if you care about *worst case execution time*

Fig ref: http://jwhitham.org/c/smmu.html

SPM in P-FRP

- P-FRP system requires atomic execution of tasks
  - Discard results of preempted tasks
  - Commit results to main memory after end of execution
- Using DMA to read/write to/from main memory
- Avoid extra overhead
- Limited size - the same as cache

Tasks in P-FRP

- A task should be in SPM before execution
- A task’s data need to be read every time
- Old data will be replaced by the new one
- Multiple Tasks in SPM
- Dynamic Scheduling of Multiple Tasks in SPM
- Tasks bigger than SPM size

Dynamic Scheduling of Multiple Tasks in SPM

- Multiple tasks can fit into SPM
- Looking up a task in the SPM using hash table
- Need to use replacement policies if SPM is full
- Avoiding the overhead of reading existing task code
- Manageable at software level

Tasks Larger Than SPM Size

- Handling Multiple Memory Reads For Data of A Single Task
  - Pieces of data should be read multiple times
  - Using DMA to speedup data read
  - New chunk of data replaces the processed data
- Using hash table for fast data access
- The pointer to data chunk is the key

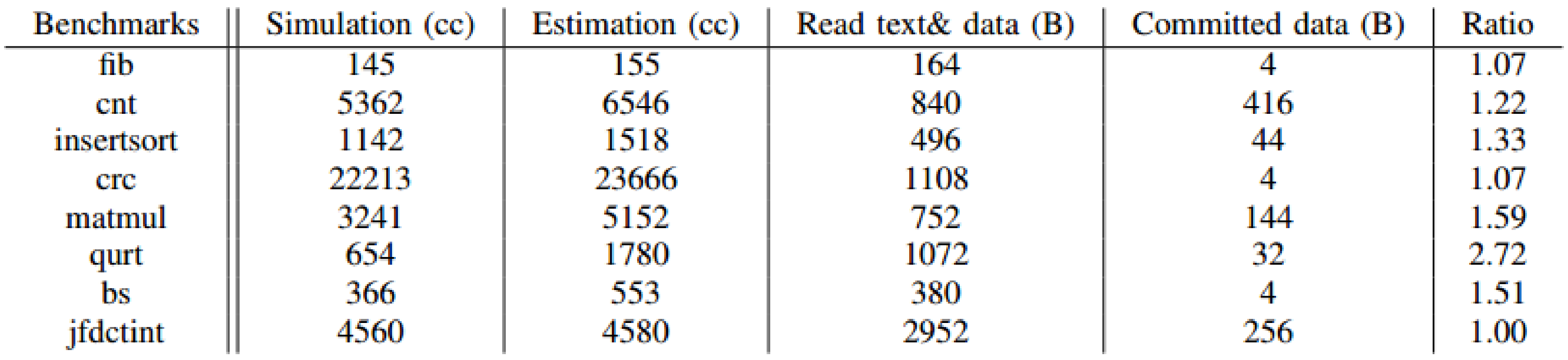
P-FRP Task's Characteristics

- As small as possible
  - Larger tasks mean more overhead caused by interference
- Every memory write operation is at the end
  - Guarantees lack of side effects
  - No need to keep the program state
- Committing data is done by DMA
- Need atomic DMA write operation
  - DMA write has the highest priority among tasks

CS@UH

Preliminary Results

- Comparison of Simulation and Estimation With One Task in SPM
- Based on Chronos Simulator [1]



Benchmarks	Simulation (cc)	Estimation (cc)	Read text& data (B)	Committed data (B)	Ratio
fib	145	155	164	4	1.07
cnt	5362	6546	840	416	1.22
insertsort	1142	1518	496	44	1.33
crc	22213	23666	1108	4	1.07
matmul	3241	5152	752	144	1.59
qurt	654	1780	1072	32	2.72
bs	366	553	380	4	1.51
jfdctint	4560	4580	2952	256	1.00

[1] Xianfeng Li, Yun Liang, Tulika Mitra, and Abhik Roychoudhury. Chronos: A timing analyzer for embedded software. Science of Computer Programming, 69(1):56–67, 2007

Ongoing Work

- A scratchpad execution platform for P-FRP systems.
- WCET analysis considering memory cost
- Dynamic SPM scheduling to reduce memory overhead
- A mathematical equation to model WCET analysis of P-FRP is produced
- Timing analysis of response time in P-FRP considering memory based on SPM

Evaluation

- Does precise timing characterization of the embedded controller software execution lead to faster physical system response compared with one designed without accurate controller timing analysis (and thus requires more tolerance of execution time variations)?
- How does the time to develop new control components with accurate response time analysis tools compare to doing the same with older methods?
- Automotive application: Do the new scheduling/execution such as AWR lead to safer physical system behaviors such as shorter stopping distance for ABS-equipped cars?
- Do optimizations to the runtime controller software such as reducing event-handler preemptions and better priority assignments result in faster controller response as measured by developed analytical methods, simulation, and actual physical system testing?

Evaluation

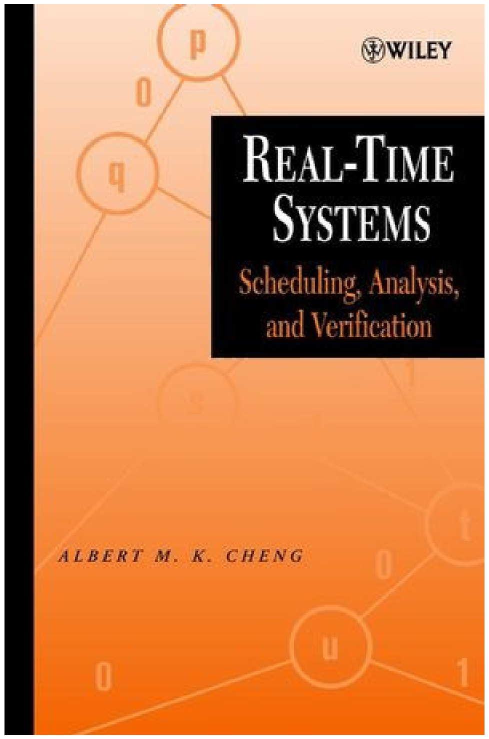
- Does the inclusion of power-aware and power-saving measures maintain the satisfaction of timing and space/memory constraints imposed on the embedded controller and controlled physical system behaviors? What is the amount of energy savings in the physical system and embedded controller achieved with these approaches compared with systems without them?
- Does the resulting approach make it easier and safer to make minor modifications to components of the control systems?
- Does this framework and toolset facilitate the design of the controller and its timing/safety verification? Is the time from design to actual implementation shortened and the development cost lowered?

Concluding Remarks

- Our goal: Enhance the safety and performance of a physical system controlled by an embedded controller consisting of single or networked control components with functional reactive programming (FRP) and real-time virtualization.
- FRP allows intuitive specification and formal verification of safety-critical behaviors, thus reducing the number of defects injected during the design phase, and the stateless nature of execution avoids the need for complex programming involving synchronization primitives.
- Accurate response time analysis tools (accounting for CPU execution, memory access, I/O, and sensor processing times), novel scheduling techniques, and new power-conserving methods are needed.
- Research impact: Facilitate the design and update of the embedded controller (or network of controllers) as well as its (their) timing and safety verification.
- Enhance and update embedded systems with real-time virtual resources.

CS@UH

☆ Thank you all!
☆ RTS Textbook and DPRTCPS 2015



DPRTCPS Workshop at RTSS 2015, San Antonio, Texas. Submissions due on October 9, 2015.

References (1)

- Andrei S., Mozahid H., and Cheng A.M.K., "Optimizing the Linear Real-Time Logic Verifier," 19th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS) WIP Session, Philadelphia, PA, April 8, 2013.
- Andrei S. and Cheng A.M.K., "Decomposition-based Verification of Linear Real-Time Systems Specifications," 2nd Workshop on Compositional Theory and Technology for Real-Time Embedded Systems (CRTS), Washington, D.C., USA (Co-located with IEEE RTSS 2009), December 1, 2009.
- Andrei S. and Cheng A.M.K., "Efficient Verification and Optimization of Real-Time Logic Specified Systems," IEEE Transactions on Computers, vol. 58, no. 12, pp. 1640-1653, December 2009.
- Andrei S., Chin W., Lupa M., Cheng A.M.K., "Automatic Debugging of Real-Time Systems Based on Incremental Satisfiability Counting," IEEE Transactions on Computers, Vol. 55, No. 7, pp. 830-843, July 2006. Selected as this issue's featured article.
- Andrei S., Mozahid H., and Cheng A.M.K., Optimizing the Linear Real-Time Logic Verifier," 19th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS) WIP Session, Philadelphia, PA, April 8, 2013.

References (2)

- Andrei S., Radulescu V., McNicholl T., Cheng A.M.K., "Toward an optimal power-aware scheduling technique," 14th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC), Timisoara, Romania, September 26-29, 2012.
- Belwal C. and Cheng A.M.K., "Chaitanya Belwal, Albert M. K. Cheng, and Bo Liu, `` Feasibility Interval for the Transactional Event Handlers of P-FRP," Special Issue on UbiSafe Computing and Communications, Elsevier's Journal of Computer and System Sciences, 2012.
- Belwal C. and Cheng A.M.K., "Determining Actual Response Time in P-FRP," Proc. Thirteenth International Symposium on Practical Aspects of Declarative Languages (PADL), Austin, Texas, USA, pages: 250-264, January 24-25, 2011.
- Belwal C. and Cheng A.M.K., "Determining Actual Response Time in P-FRP using Idle-Period Game Board," Proc. 14th IEEE International Symposium on Object, Component, and Service-Oriented Real-time Distributed Computing (ISORC), Newport Beach, CA, USA, pages: 136-143, March 28-31, 2011.

References (3)

- Belwal C. and Cheng A.M.K., "Response Time Bounds for Event Handlers in the Priority-based Functional Reactive Programming (P-FRP) Paradigm," ACM Research in Applied Computation Symposium (RACS), San Antonio, Texas, USA, October 23-26, 2012.
- Cheng A.M.K, Niktab H., and Walston M., "Timing Analysis of Small Aircraft Transportation System (SATS)," International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA), Seoul, Korea, August 2012.
- Ras J. and Cheng A.M.K., "Response Time Analysis of the Abort-and-Restart Model under Symmetric Multiprocessing." The 7th IEEE International Conferences on Embedded Software and Systems (ICESS-10)," pages: 1954-1961, 2010.
- Ras J. and Cheng A.M.K., "Response Time Analysis for the Abort-and-Restart Event Handlers of the Priority-Based Functional Reactive Programming (P-FRP) Paradigm," Embedded and Real-Time Computing Systems and Applications (RTCSA-09), pages: 305-314, 2009.
- Sha, L., Elements of Research, UIUC, 2014.

References (4)

- Ras J. and Cheng A.M.K., "An Evaluation of the Dynamic and Static Multiprocessor Priority Ceiling Protocol and the Multiprocessor Stack Resource Policy in an SMP System," Proc. IEEE-CS Real-Time and Embedded Technology and Applications Symposium (RTAS), San Francisco, California, April 2009.
- Wen Y., Liu Z., Shi W., Jiang Y., Yang F., Kohar A., Cheng A.M.K., "Support for Power Efficient Mobile Video Playback on Simultaneous Hybrid Display,“ 10th IEEE Symposium on Embedded Systems for Real-Time Multimedia Tampere, Finland, October 11-12, 2012.
- Wen Y., Belwal C., Cheng A.M.K., "Response Time Bounds for Event Handlers in the Priority based Functional Reactive Programming (P-FRP) Paradigm," ACM Research in Applied Computation Symposium (RACS), San Antonio, Texas, 2012.
- Wen Y., Belwal C., Cheng A.M.K., "Time Petri Nets for Schedulability Analysis of the Transactional Event Handlers of P-FRP," ACM Research in Applied Computation Symposium (RACS), San Antonio, Texas, USA, October 23-26, 2012.
- Li Y. and Cheng A.M.K., "Static Approximation Algorithms for Regularity-based Resource Partitioning," 33rd Real-Time Systems Symposium (RTSS), San Juan, Puerto Rico, USA, December 4-7, 2012.

References (5)

- Chaitanya Belwal, Yuanfeng Wen and Albert M. K. Cheng, ``Utilization Bounds of P-FRP Tasks," International Journal of Embedded Systems, 2014.
- Yong woon Ahn, Albert M. K. Cheng, Jinsuk Baek, Minho Jo, and Hsiao-Hwa Chen, ``An Auto-Scaling Mechanism for Virtual Resources to Support Mobile, Pervasive, Real-Time, Healthcare Applications in Cloud Computing," IEEE Network, Sept. 2013.
- Chaitanya Belwal, Albert M. K. Cheng, and Bo Liu, `` Feasibility Interval for the Transactional Event Handlers of P-FRP," Special Issue on UbiSafe Computing and Communications, Elsevier's Journal of Computer and System Sciences, Volume 79, Issue 5, pages 530-541, August 2013.
- Yuanfeng Wen, Chaitanya Belwal, and Albert M. K. Cheng, ``Towards Optimal Priority Assignments for the Transactional Event Handlers of P-FRP," ACM International Conference on Reliable And Convergent Systems (RACS), Montreal, QC, Canada, October 1-4, 2013.
- Chaitanya Belwal, Albert M. K. Cheng, J. Ras, and Yuanfeng Wen, ``Variable Voltage Scheduling with the Priority-based Functional Reactive Programming Language," ACM International Conference on Reliable And Convergent Systems (RACS), Montreal, QC, Canada, October 1-4, 2013.

References (6)

- Yu Jiang, Qiang Zhou, Xingliang Zou, and Albert M. K. Cheng, ``Feasibility Interval of Real-Time Tasks with Arbitrary Release Offsets Under Fixed Priority Scheduling,'' 11th IEEE International Conference on Embedded Software and Systems (ICESS), in conjunction with HPCC and CSS, Paris, France, August 20-22, 2014.
- Zeinab Kazemi Alamouti and Albert M. K. Cheng, ``Static Worst Case Execution Time Analysis of Functional Reactive Systems,'' 11th IEEE International Conference on Embedded Software and Systems (ICESS) WIP Session, in conjunction with HPCC and CSS, Paris, France, August 20-22, 2014.
- Qiang Zhou, Xingliang Zou, Albert M. K. Cheng, and Yu Jiang, ``An Integrated Analysis of the Worst Case Response Time for P-FRP,'' to appear in 35th IEEE-CS Real-Time Systems Symposium (RTSS) WIP Session, Rome, Italy, December 3-5, 2014. Jian (Denny) Lin, Albert M. K. Cheng, Douglas Steel, and Michael Yu-Chi Wu, ``Scheduling Mixed-Criticality Real-Time Tasks with Fault Tolerance,'' to appear in 2nd Workshop on Mixed Criticality (WMC), in conjunction with IEEE RTSS, Rome, Italy, December 2, 2014.

References (7)

- Kaleb Christoffersen and Albert M. K. Cheng, `` Model-Based Design: Anti-lock Brake System with Priority-Based Functional Reactive Programming," 34th IEEE-CS Real-Time Systems Symposium (RTSS) WIP Session, Vancouver, Canada, Dec. 2013.
- Yong woon Ahn and Albert M. K. Cheng, `` Automatic Resource Scaling for Medical Cyber-Physical Systems Running in Private Cloud Computing Architecture," Medical Cyber Physical Systems Workshop (MedicalCPS), Cyber-Physical Systems Week (CPSWeek), Berlin, Germany, April 14, 2014.
- Stefan Andrei, Albert M. K. Cheng, and Mozahid Haque, `` Mathematical Considerations of Linear Real-Time Logic Verification," 20th IEEE-CS Real-Time and Embedded Technology and Applications Symposium (RTAS) WIP Session, Berlin, Germany April 2014.
- Yu Jiang, Xingliang Zou, and Albert M. K. Cheng, ``On the Schedulability of P-FRP Tasks," 20th IEEE-CS Real-Time and Embedded Technology and Applications Symposium (RTAS) WIP Session, Berlin, Germany April 2014.
- Albert M. K. Cheng, ``An Undergraduate Cyber-Physical Systems Course," ACM Workshop on Design, Modeling, and Evaluation of Cyber-Physical Systems (CyPhy), Cyber-Physical Systems Week (CPSWeek), Berlin, Germany, April 14, 2014.