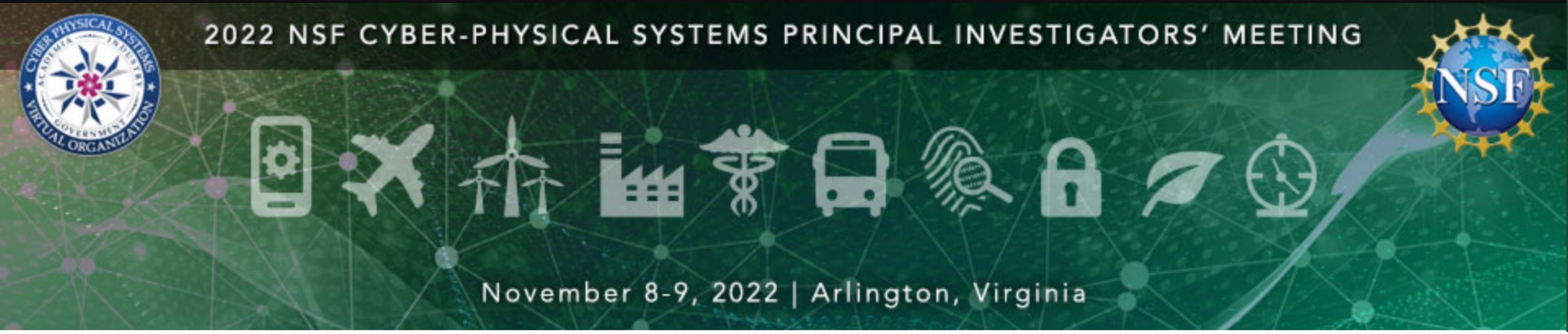

2022 NSF CYBER-PHYSICAL SYSTEMS PRINCIPAL INVESTIGATORS' MEETING
November 8-9, 2022 | Arlington, Virginia

TickTalk: Timing API for Federated Cyber-physical Systems

- Bob Iannucci (CMU), **Aviral Shrivastava** (ASU)
- Carlee Joe-Wong (CMU), Jonathan Aldrich (CMU)
- http://ccsg.ece.cmu.edu/wp/index.php/home/ticktalk/
- Aviral.Shrivastava@asu.edu

- CNS-1645578 (ASU), CNS-1646235 (CMU)

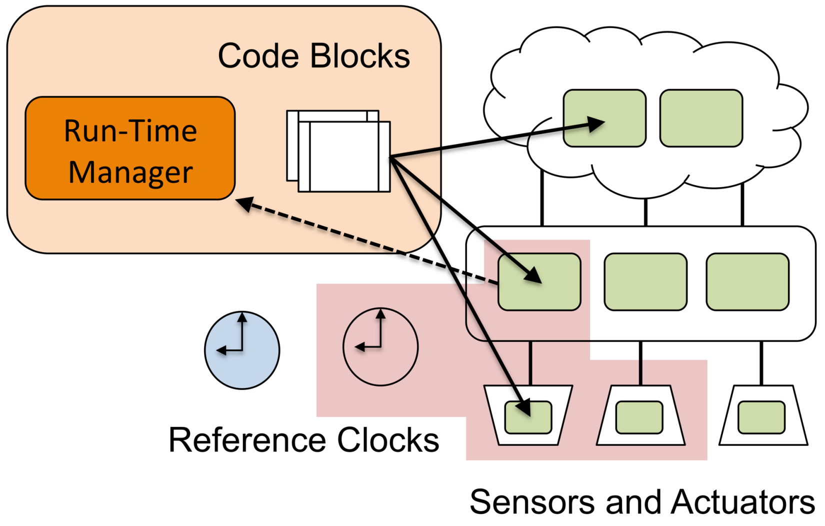
TickTalk: Timing API for Federated Cyber-physical Systems

The economic **potential** of large-scale CPS (*e.g.*, smart cities and environments) will be enabled in part through simplification of programming (like app development by non-specialists).

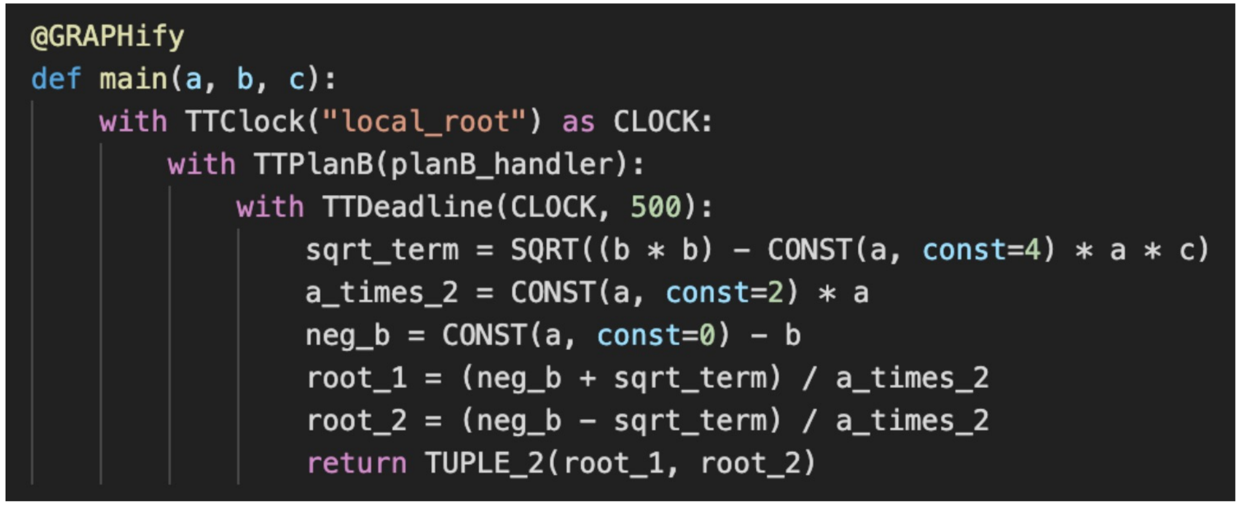
The need to meet the timing specifications makes programming large-scale distributed CPS **difficult**.

Proposal:

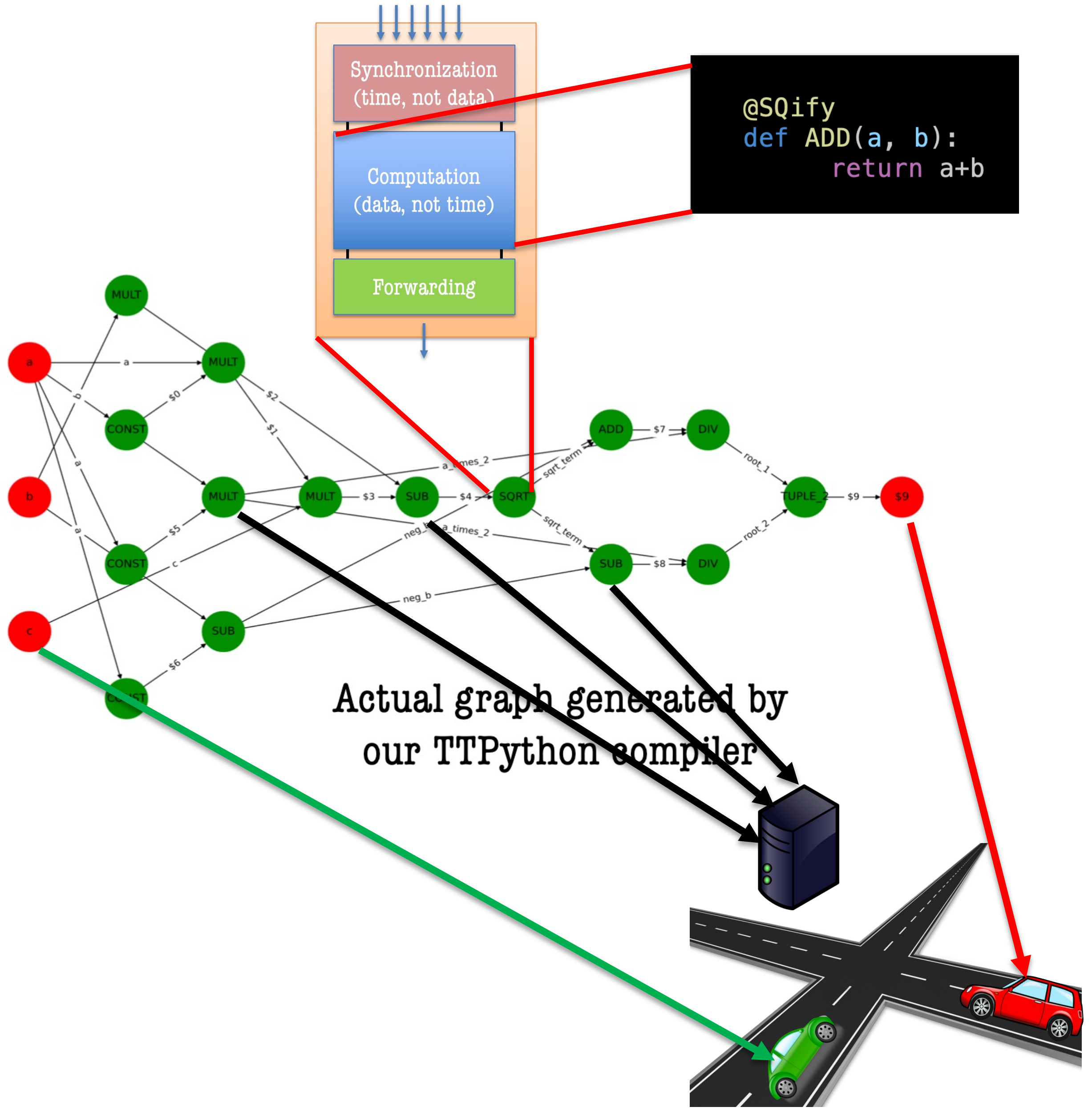
- Create a **programming language** that abstracts timing, timing-fault handling and related power management issues
- Develop **hardware extensions** that support low-power sync, timing-related power reporting, and multi-tenancy
- Create an **end-to-end demonstration** including a compiler and runtime; deploy in a real-world testbed



TTPython – Time-sensitive Macro-programming Language



```
@GRAPHify
def main(a, b, c):
    with TTClock("local_root") as CLOCK:
        with TTPlanB(planB_handler):
            with TTDeadline(CLOCK, 500):
                sqrt_term = SQRT((b * b) - CONST(a, const=4) * a * c)
                a_times_2 = CONST(a, const=2) * a
                neg_b = CONST(a, const=0) - b
                root_1 = (neg_b + sqrt_term) / a_times_2
                root_2 = (neg_b - sqrt_term) / a_times_2
                return TUPLE_2(root_1, root_2)
```


Synchronization (time, not data)
Computation (data, not time)
Forwarding
a
b
c
MULT
CONST
SUB
SQRT
ADD
DIV
TUPLE_2
$9

```
@SQify
def ADD(a, b):
    return a+b
```

Actual graph generated by our TTPython compiler

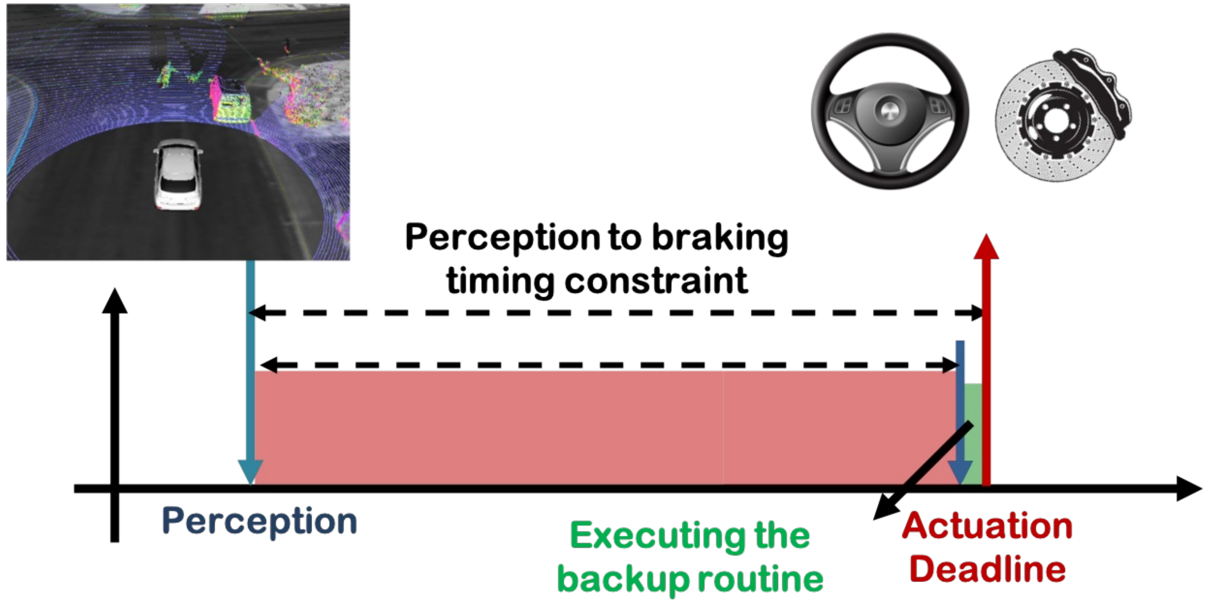
STEP 1: The programmer adds the @SQify annotation to function definitions. This says “expect the inputs to arrive as time-tagged values, but please compute the function body without having to think about that.”

STEP 2: The programmer writes a Python-like program that is made up of calls to these SQified functions. Normal infix operations can be used for basic arithmetic operations. Importantly, time annotations like deadlines can be added.

Under the covers: the TTCompiler translates this program into a dataflow graph made up of instances of the SQified functions hooked together with arcs and triggered by tokens.

PlanB – To Design Timing-Robust Cyber-Physical Systems

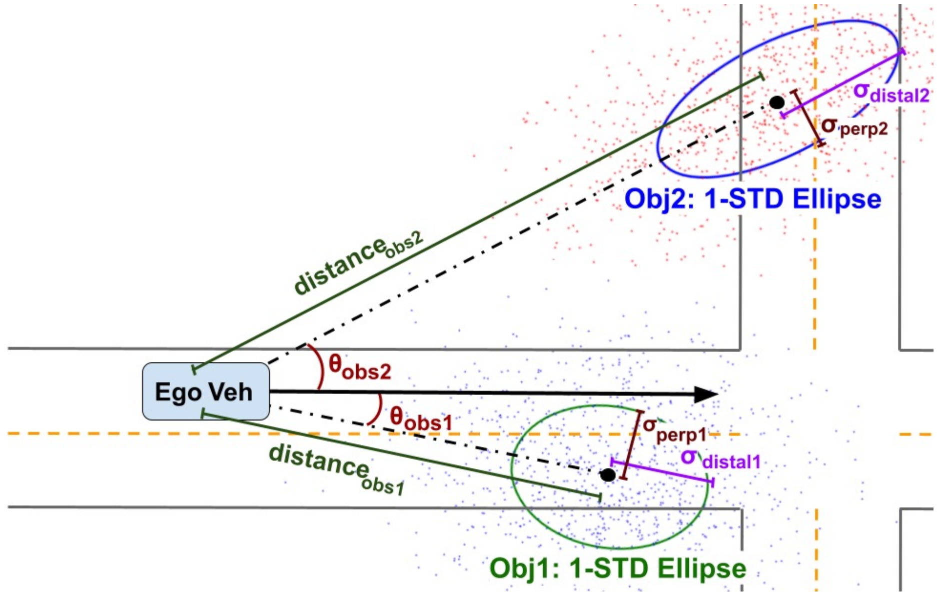
- Many Cyber-Physical Systems (CPS) have timing constraints that must be met by the cyber components (software and the network) to ensure safety.
- It is a tedious job to check if a CPS meets its timing requirement especially when they are distributed and the software and/or the underlying computing platforms are complex.
- Furthermore, the system design is brittle since a timing failure can still happen e.g., network failure, soft error bit flip, etc.
- We propose a new design methodology called PlanB where timing constraints of the CPS are monitored at the runtime, and a proper backup routine is executed when a timing failure happens to ensure safety.
- We provide a model on how to express the desired timing behavior using a set of timing constructs in a C/C++ code and how to efficiently monitor them at the runtime.
- The system remains safe and stable even when intentional faults are injected to cause a timing failure. We also demonstrate that the system can achieve graceful degradation when a less extreme timing failure happens.

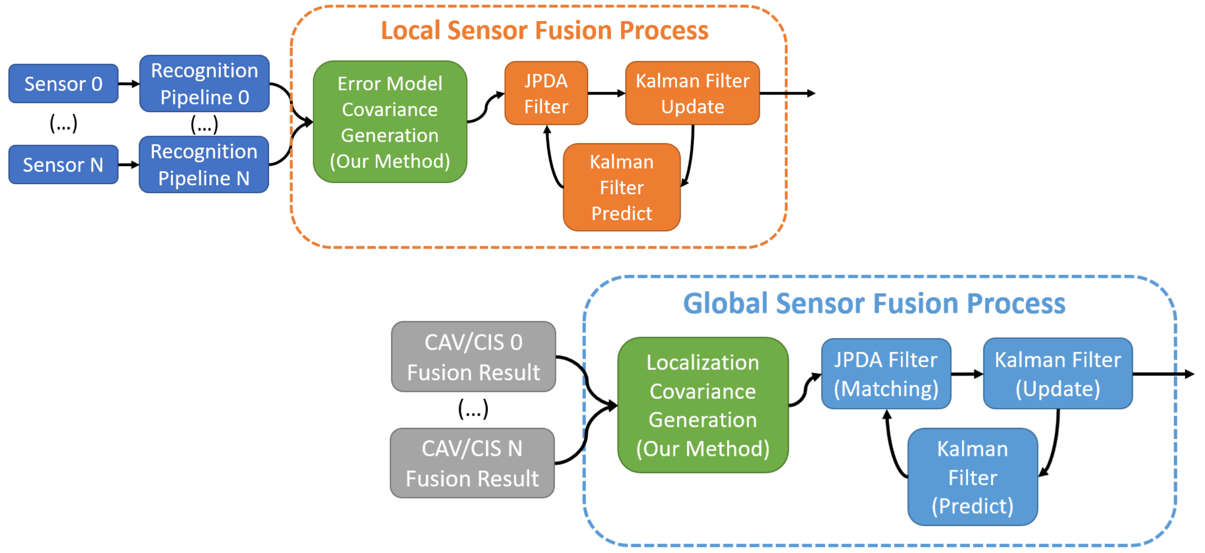

Perception to braking timing constraint
Perception
Executing the backup routine
Actuation Deadline

[ACM TCPS] PlanB – Design Methodology for Cyber-Physical Systems Robust to Timing Failures, Mohammad Khayatian, Mohamamdreza Mehrabian, Edward Andert, Reese Grimsley, Kyle Liang, Yi Hu, Ian McCormack, Carlee Joe-Wong, Jonathan Aldrich, Bob Iannucci and Aviral Shrivastava, in the ACM Transactions on Cyber Physical Systems (TCPS), Jan 2022, Article 21, pages 39, https://doi.org/10.1145/3516449

Accurate Cooperative Sensor Fusion

- A major challenge in cooperative sensing is to weight the measurements taken from the various sources to get an accurate result.
- Previous cooperative sensor fusion approaches for autonomous vehicles use a fixed error model, in which the covariance of a sensor and its recognizer pipeline is just the mean of the measured covariance for all sensing scenarios.
- Our approach estimates error using key predictor terms that have high correlation with sensing and localization accuracy for accurate covariance estimation of each sensor observation.
- We adopt a tiered fusion model consisting of local and global sensor fusion steps.
    - At the local fusion level, we add in a covariance generation stage using the error model for each sensor and the measured distance to generate the expected covariance matrix for each observation.
    - At the global sensor fusion stage we add an additional stage to generate the localization covariance matrix from the key predictor term velocity and combines that with the covariance generated from the local fusion for accurate cooperative sensing.
- Results show an average and max improvement in RMSE when detecting vehicle positions of 1.42x and 1.78x respectively in a four-vehicle cooperative fusion scenario when using our error model versus a typical fixed error model.





[ITSC] Accurate Cooperative Sensor Fusion by Parameterized Covariance Generation for Sensing and Localization Pipelines in CAVs, Edward Andert and Aviral Shrivastava, in the IEEE Intelligent Transportation Systems Conference (ITSC), 2022

Demo: Traffic Intersection for Autonomous Cars

- Crossroads Algorithm for Managing Traffic Intersection for Autonomous Vehicles.
- When vehicles approach the intersection, they send their position and velocity to the intersection manager.
- Intersection manager knows the schedule of all the vehicles crossing the intersection and assigns a speed to the incoming vehicle for safe and efficient crossing.
- Crossroads algorithm considers the computation and communication delays as safety buffer around the vehicle.

Advantages of our approach

- Macro-programming – Program the whole system as one application
- Explicitly the specify timing constraints – using the “with” decorator
- Timing constraints can be distributed
- Automatic clock synchronization, timestamp translation
- Dataflow execution and token passing under the hood
- Portable timing and code