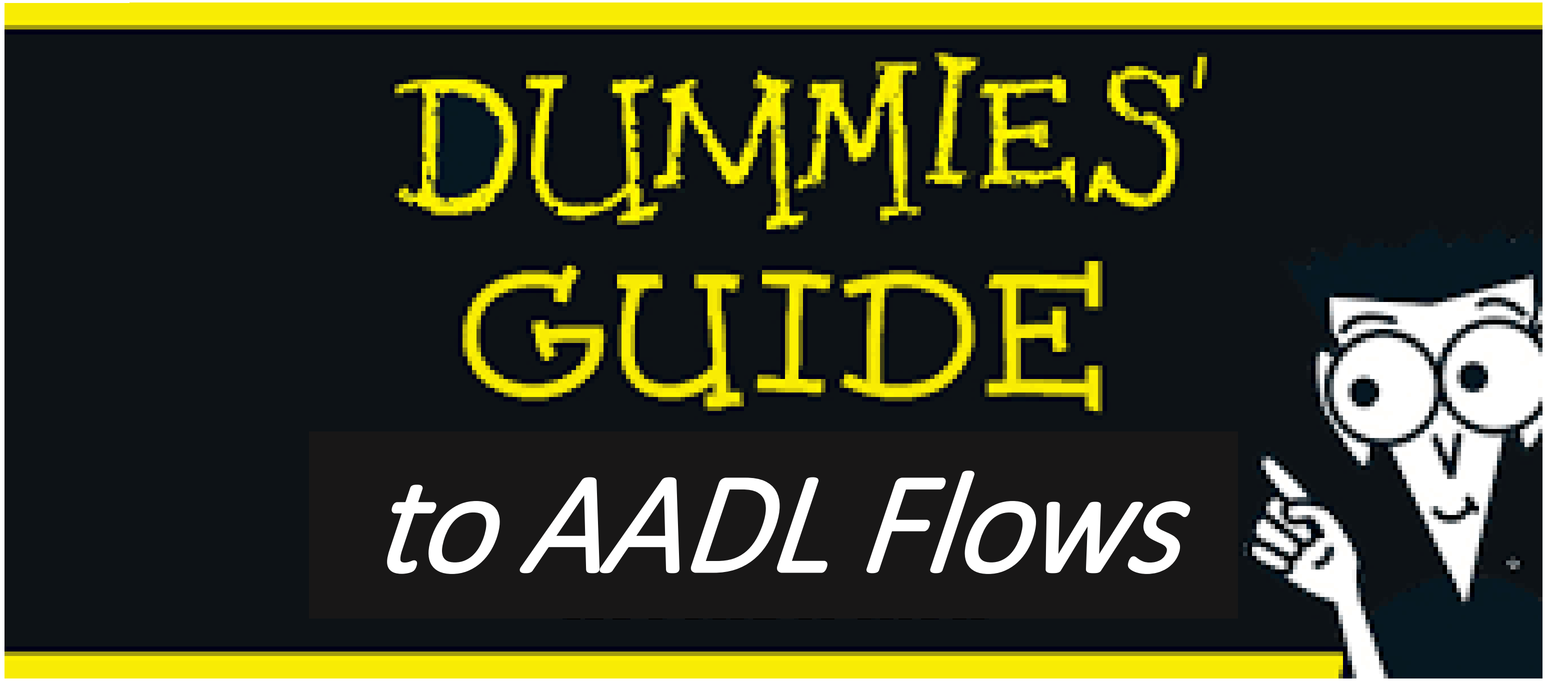


# DUMMIES' GUIDE to AADL Flows

*This is intended as a gentle introduction. The goal is a simple explanation of useful capabilities for tool users. This is our interpretation of the AADL standard that we believe provides a useful semantic model for users and tool developers.*

Adventium Labs
steve.vestal@adventiumlabs.com tyler.smith@adventiumlabs.com
November 2018

Contract # W911W6-17-D-0003

# AADL Flow Declarations

Here is an example AADL end to end flow implementation declaration:

```
myFlow: end to end flow
    threadA -> connectAB -> threadB -> connectBC -> threadC
    {Latency => 0ms .. 40ms;};[1]
```

AADL thread, connection, processor and bus declarations specify how a system is to be built, e.g., the period at which periodic threads are to be released, how a processor is to be scheduled, how a connection is to be routed through a network[2].

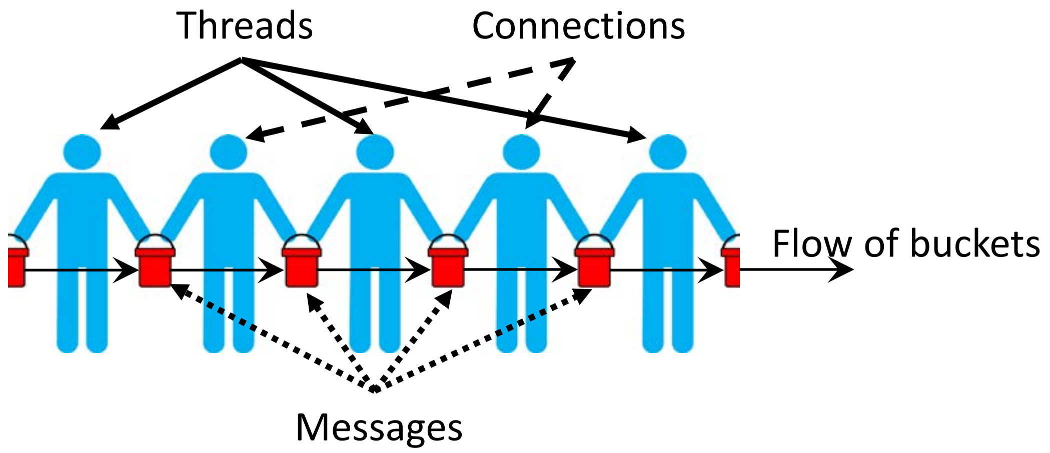
AADL flow declarations identify paths through the as-built system along which information flows, together with a requirement that end-to-end latencies along flows shall fall within specified bounds at run-time.

The thread and connection declarations are the build-to specifications. Flow declarations specify requirements to be verified about a system built-to-spec[3].

The FASTAR schedulability analyzer verifies that a system built to its model-based specification will satisfy flow latency requirements.

[1]A flow may also begin or end at an output or input port or data component. We won't discuss these variants here.
[2]The real system components should comply with these build-to specifications, e.g. thread execution times must not exceed their specified worst-case execution times. In this sense they are requirements on the real components.
[3]The exception occurs when a design synthesis tool, such as the SPICA schedule generator, generates a part of the AADL model in a way that satisfies user-specified requirements. Flow declarations affect how the system is built indirectly through the generated AADL specifications. Analysis can serve as an independent means of verification.

# The Bucket Brigade Analogy with Exceptions



- People will change the water in a bucket before passing it along (threads will compute new output values from their input values).
- Some people may be able to hold multiple buckets while waiting for the people around them (messages may be queued at thread ports).
- One person may work slower than another, and some buckets may be dropped (a thread may under-sample one of its input ports).
- One person may work faster than another, and some buckets may be cloned so there are multiple copies of that bucket (a thread may over-sample one of its input ports).
- Some people will stand there and only pass along a bucket when ordered to do so (a thread may be event-driven).
- Portions of a bucket brigade may be synchronized, so that one person turns to hand a bucket off at the same time another turns to receive it.
- Portions of a bucket brigade may not be synchronized, so that some people are left waiting to receive a bucket or holding a bucket for awhile.

# Some Real-Time Concepts and Terms

Real-time tasks (threads and connections) have some common and widely-accepted concepts and terms regardless of scheduling discipline.

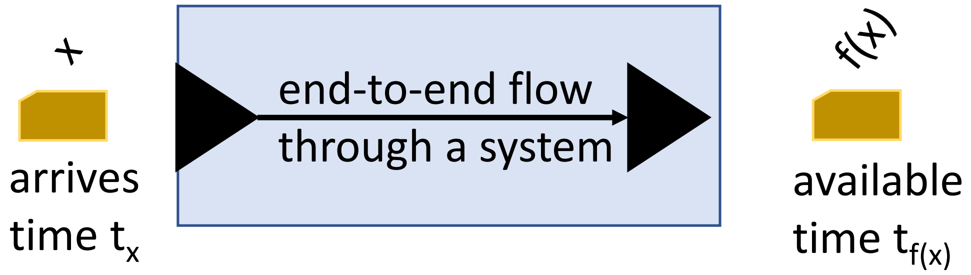
- **Release times**: wall-clock times at which a thread is placed on a ready queue. An execution begins at or after each release time. (For periodic threads, these are specified as a fixed interval between release times; for sporadic threads, as a minimum inter-arrival time.)
- **Completion times**: wall-clock times at which thread executions complete; times at which a thread is removed from the ready queue until its next release time (typically because the thread executes a "suspend until next release" service call).
- **Response times**: intervals of (wall-clock) time between the release times of a thread and the subsequent completion times of the resulting thread executions.

All of these depend on the run-time behavior of the integrated system and may vary from thread-to-thread and execution-to-execution.

The following are often (but not always) considered run-time errors:

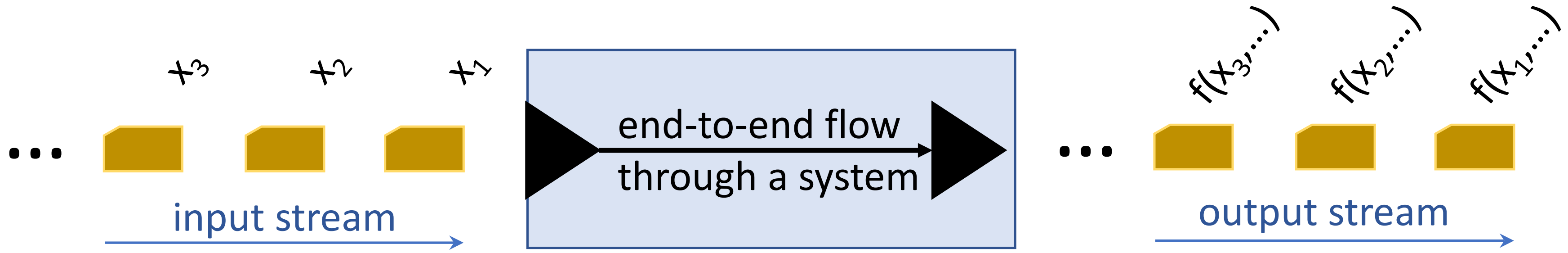
- release times do not conform to specified period or inter-arrival times
- a response time exceeds a specified deadline requirement
- a release time occurs before the previous thread execution has completed

# System Input/Output View of a Flow



If an input value x arrives at time $t_x$, and an output value f(x) computed using x becomes available at time $t_{f(x)} > t_x$, then the flow latency for input value x is $t_{f(x)} - t_x$.

We further restrict this to one specific output f(x), the earliest one that could have been computed using x as an input (the earliest one with a causal relation from x)[1].

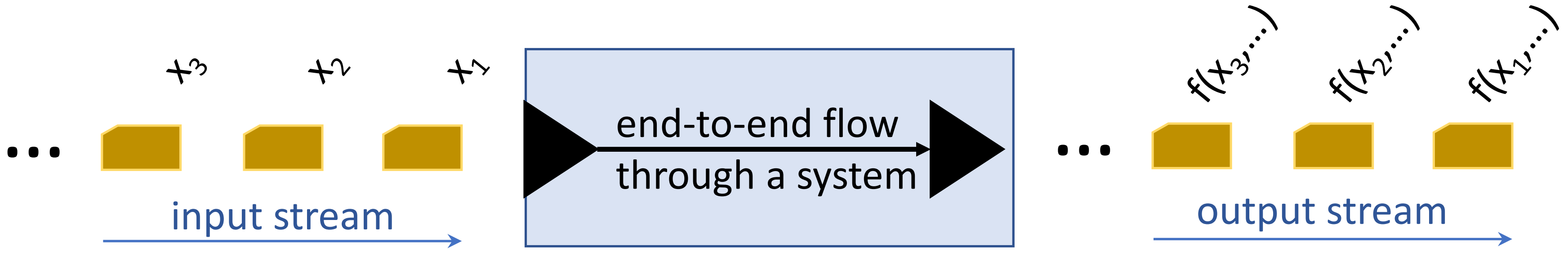


A flow has a sequence of inputs (an input stream). Each input has its own latency.

The system may contain state, an output is in general a function of all previous inputs.

[1]In the presence of over/under-sampling, asynchrony, etc. within a flow, there may be multiple outputs that share a most recent input. Only one of them will be the corresponding earliest output for a specific input.

# Flow Releases, Executions, Completions



We define flow release times, executions, and response times analogous to those of threads.

Inputs are release times for executions of the end-to-end flow. These are typically at or proximately preceding the release times of the first thread named in the flow declaration.

Earliest (proximately succeeding) outputs computed from arriving inputs are completion times for flow executions. These are typically completion times for the last thread named in the flow declaration[1].

Flow execution response times are intervals of time between an input and its corresponding earliest output (flow release times to flow completion times)[2].

Flow latency bounds are the min and max of all flow execution response times.

[1]In the presence of internal asynchrony or oversampling, not every final thread completion time is necessarily a flow execution completion time.

[2]An alternative is to define requirements based on outputs: for each output, what are the most recent inputs used to compute that output? Some literature refers to this as the "age" rather than latency or response time for an output.

## Schedulability Analysis of AADL Models

Assume the AADL model specifies for each thread and connection

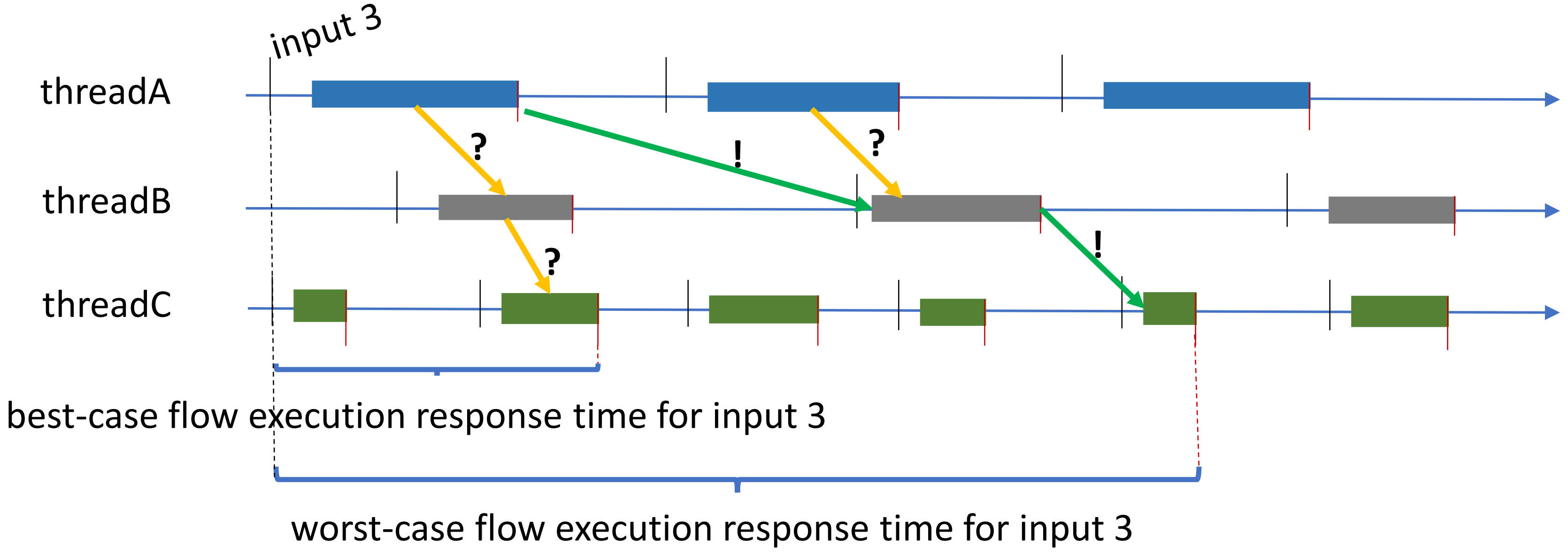
- hosting processors and  busses and their scheduling disciplines and overheads
- release periods, inter-arrival times, or events (release event streams)
- worst-case execution times[1]

For most uni-processor scheduling disciplines there are schedulability analysis algorithms that will determine, for each thread and connection task, an upper bound on all its response times.  This assumes the run-time system behavior conforms to its AADL model-based specification.

Flow schedulability analysis determines an upper bound on flow response times for all flow executions and compares that analytic bound against the Latency requirement specified in the flow declaration.  (Declared flow Latency is thus analogous to declared thread Deadline.)

[1]WCET is a scalar input parameter used in schedulability analysis whose value is, in practice, challenging to tightly bound with acceptable assurance.

# Flow Execution Response Times (Latencies)


input 3
threadA
threadB
threadC
?
!
?
?
!
best-case flow execution response time for input 3
worst-case flow execution response time for input 3

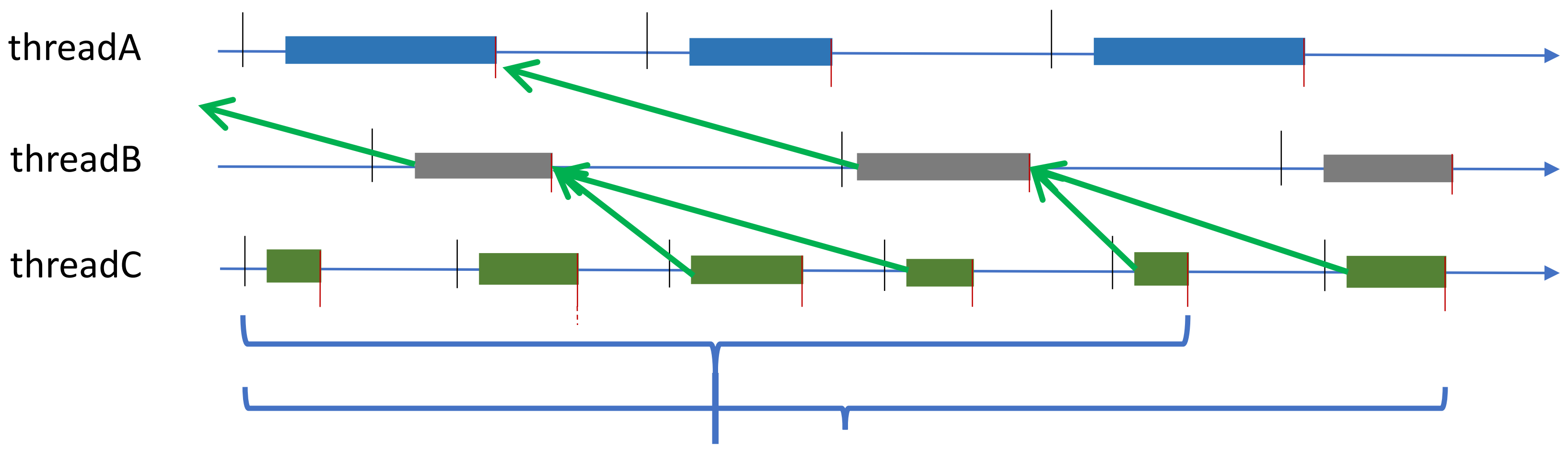
Individual flow response times depend on implementation details and run-time variability.

- When do release times of different threads occur relative to each other? Even for strictly periodic threads, this may vary in asynchronous systems.
- When are messages transferred from sending to receiving threads? Send and receive calls may occur anytime between thread start time and thread completion time (assumed above). Systems may statically schedule transmissions, e.g. sends at deadlines and receives at release times. This may depend on message release and completion times on networks and asynchronous sampling latencies.

Unlike threads, a new flow execution may be released before an earlier flow execution has completed (flows may pipeline through a system).

Flow schedulability analysis determines an upper bound for all possible worst-case flow execution response times for each flow declaration.

# Flow Execution Ages



worst-case ages of output values

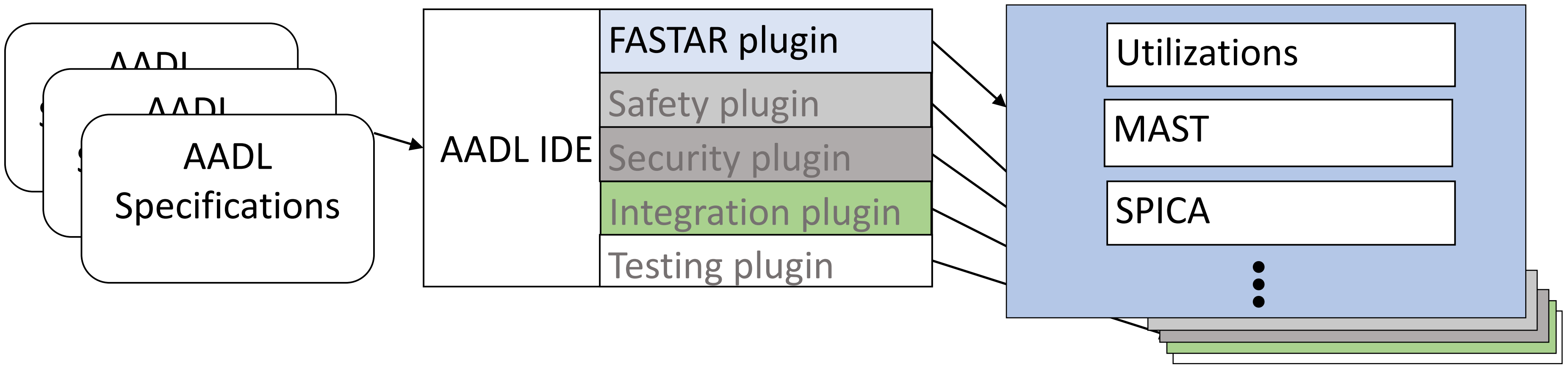
A different metric that is sometimes useful  is the age of each individual completion of the final task (each final output value) of a flow[1]. This is an upper bound on the time interval between each output event and the most recent preceding arrival of the input value used to compute that output value[2].

Latency is a metric for each input value.  Age is a metric for each output value.

[1]The real-time database community uses this metric.

[2]The AADL 2.2 standard does not define an Age property.

# Framework for Analysis of Schedulability, Timing And Resources (FASTAR)



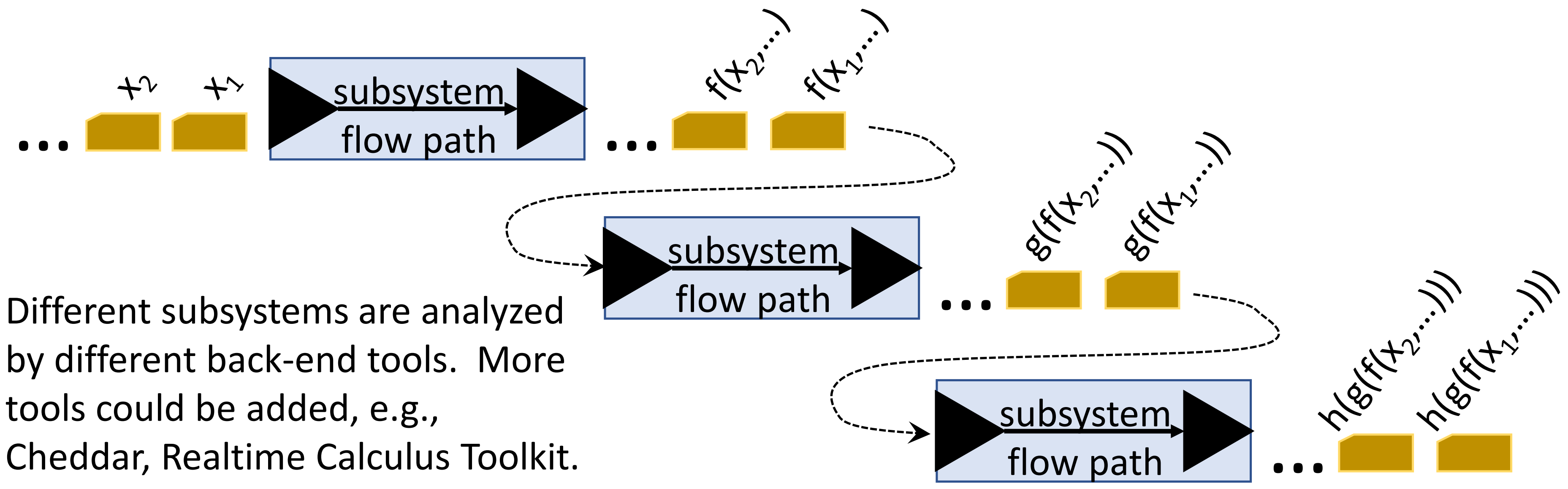
There are many scheduling and timing analysis tools specialized for specific types of equipment and workload patterns, ranging from open source to proprietary.

FASTAR is a framework that allows integration and collaborative use of a selected set of specialized tools for families of heterogeneous, distributed, layered systems. FASTAR performs collaborative/compositional schedulability analysis[1].

FASTAR currently has plugins for two analysis tools, MAST for priority-based scheduling and analysis and SPICA for ARINC 653 scheduling and analysis.

[1] See Kunzil, Hermann, Ernst, and Thiele, "Combined Approach to System Level Performance Analysis of Embedded Systems," CODES+ISS 2007; Perathoner, Wandeler, Thiele, Hamann, Schliecker, Henia, Racu, Ernst, and Harbour, "Influence of Different System Abstractions on the Performance Analysis of Distributed Real-Time Systems," EMSOFT 07; and R. Edman, H. Shackleton, J. Shackleton, T. Smith, and S. Vestal, A Framework for Compositional Timing Analysis of Embedded Computer Systems, HPCC 2015 .

# FASTAR Composition in a Nutshell



Different subsystems are analyzed by different back-end tools. More tools could be added, e.g., Cheddar, Realtime Calculus Toolkit.

$x_i$, $f_i$, $g_i$, $h_i$ are event streams. There can be multiple subclasses of event streams. The framework currently uses subclasses for periodic-with-jitter and sporadic streams. Examples: periodic hardware interrupt, periodic-with-jitter message arrivals and departures. Additional ones can be added. Conversions may be needed between different analyzers.

The model specifies which subsystems' periodic releases are synchronized and which are not. FASTAR adds sampling latencies, or delayed or immediate connection latencies, as needed.

Every back-end analyzer must report upper bounds on thread, connection, and sub-flow latencies such that the sum, plus any sampling or special connection latencies between subsystems, is an upper bound on the system end-to-end flow latency.

# Some FASTAR Schedulability Assumptions[1]

## Framework assumptions apply to all integrated analysis methods.

- Port and feature connections are used to determine if information flows from one thread to another. Access connections to shared data components are not supported in flows.
- Transient effects at start-up, operating mode changes, and shut-down are not analyzed.
- Where a part of a model is analyzed by a selected method (e.g. MAST, SPICA), the AADL features supported and the analyzer assumptions made will be determined by that analysis method.
- Where a part of a model is declared to be BlackBox, user-declared latencies and message arrival and departure properties will be used when determining bounds on flow latencies.
- Every integrated analysis method reports analysis results to the framework that allow an upper bound on end to end flow latency to be determined by summing reported upper bounds on <release time, completion time> pairs along a flow, with sampling latencies inserted according to AADL connection and synchronous/asynchronous time domain semantics[2].

[1]To the maximum possible extend, the framework and integrated analyzers check for and issue error or warning messages for unsupported AADL language features or when run-time behaviors will or might violate analysis assumptions.

[2]This assumption must be satisfied when integrating an analysis method into the framework. Reported <release time, completion time> pairs may be for individual threads and connections. They may be for sub-sequences of threads and connections along a flow, e.g. an upper bound between the release times of a sending thread and the completion times of messages sent by that thread. The reported upper bounds are processed as edges in a directed graph. The framework assumes that the sum of results along any path from begin to end of an end to end flow, plus needed sampling latencies, is an upper bound on end to end flow latency. By default the framework will add sampling latencies as needed according to AADL semantics, but an analysis method may override this and incorporate sampling in its results (e.g. SPICA ARINC 653).

# Some MAST Schedulability Assumptions

Relaxing any of these assumptions will require tool enhancements.

- An AADL processor can be executing at most one thread at any point in time. Multi-cores are modeled as multiple processors. Interference effects are accounted for in specified thread execution time and processor overhead properties.
- All threads are periodic or sporadic, or aperiodic triggered by a periodic or sporadic event.
- Thread response times that exceed specified thread deadlines or periods are erroneous.
- Thread fixed priorities remain fixed at the specified values throughout system operation.
- Threads do not have unbounded blocking times. Any blocking times other than for real-time semaphores or thread preemption overheads (which are specified in the AADL model) are accounted for in the thread execution time property values.
- An input queueing port has all queued messages flushed during a receiver execution.
- Message release times occur at either sending thread deadline times (AADL default when sender and receiver are in the same synchronization domain) or completion times (sender and receiver are in different synchronization domains or Immediate connection type declared).
- Message transmission completion times occur at either the completion time of the sending thread (an unbound connection within a single processor) or completion times determined by analyzing a connection as a sequence of scheduled tasks whose binding declares a connection routing.

# Some SPICA ARINC 653 Schedulability Assumptions

Relaxing any of these assumptions will require tool enhancements.

- An ARINC 653 module is uni-processor (single core). Interference effects are accounted for in specified thread Compute_Execution_Times.
- All threads are periodic. Support for sporadics and aperiodics is TODO.
- Thread response times that exceed thread deadlines or periods are erroneous.
- Thread release times are strictly deterministic after system startup. Periodic_Processing_Start is true in only one window for each partition (thus phase-shift of release times at partition re-starts does not occur). Relaxation to multiple starts that satisfy determinism requirements is TODO.
- Thread priorities remain fixed at the specified values throughout system operation.
- Threads do not have unbounded blocking times. Compute_Execution_Time upper bounds account for all thread blocking times. Explicit modeling of intra-partition methods is TODO.
- Thread and partition preemption times are zero. Explicit modeling is TODO.
- All connections are between sampling ports. Queuing ports are not supported.
- ARINC 653 connection timing is used rather than AADL default semantics. Message transmission from an execution of sending thread A to an execution of receiving thread B is assumed to have completed if the sending execution completion time does not exceed the receiving execution release time (possibly before that in a running system, this is used to determine worst-case flow executions as defined earlier).