Time Enough for seL4

Assured Systems III

Time Enough for seL4

Todd Carpenter - Chief Engineer at Adventium Labs
todd.carpenter@adventiumlabs.com

Acknowledgements to

John Shackleton, Jim Carciofini, Dr Steve Vestal, and Bruce Lewis from Adventium Labs

Dr. John Hatcliff, Dr. Robby, Jason Belt from Kansas State University

Collins CASE team

Data61 seL4 team

Kent McLeod from Kry10 Limited

DARPA, NASA, DHS, AFRL, US Army CCDC

To Submit Questions

Please enter questions into the discussion board on the seL4 Summit virtual hub page

You can also use this area to continue the discussion after the presentation as speakers will be checking for new comments throughout the summit

Overview

- Temporal Challenges
- seL4 verified kernel capabilities
- Model-based toolchain

Adventium Overview

- Solutions for safe and secure cyber-physical systems
- Secure separation architectures for virtualized environments
- Model-based system engineering

R&D Portfolio

- Cyber physical systems, safety, security, survivability
- Planning and scheduling, multi-core, secure virtualization, automated reasoning

CAMET Library

- Supported source of model-based engineering (MBE) tools
- Critical role in Army Digital Engineering Strategy
- Baselined for use by Army research teams

Broad capability to support Future Vertical Lift

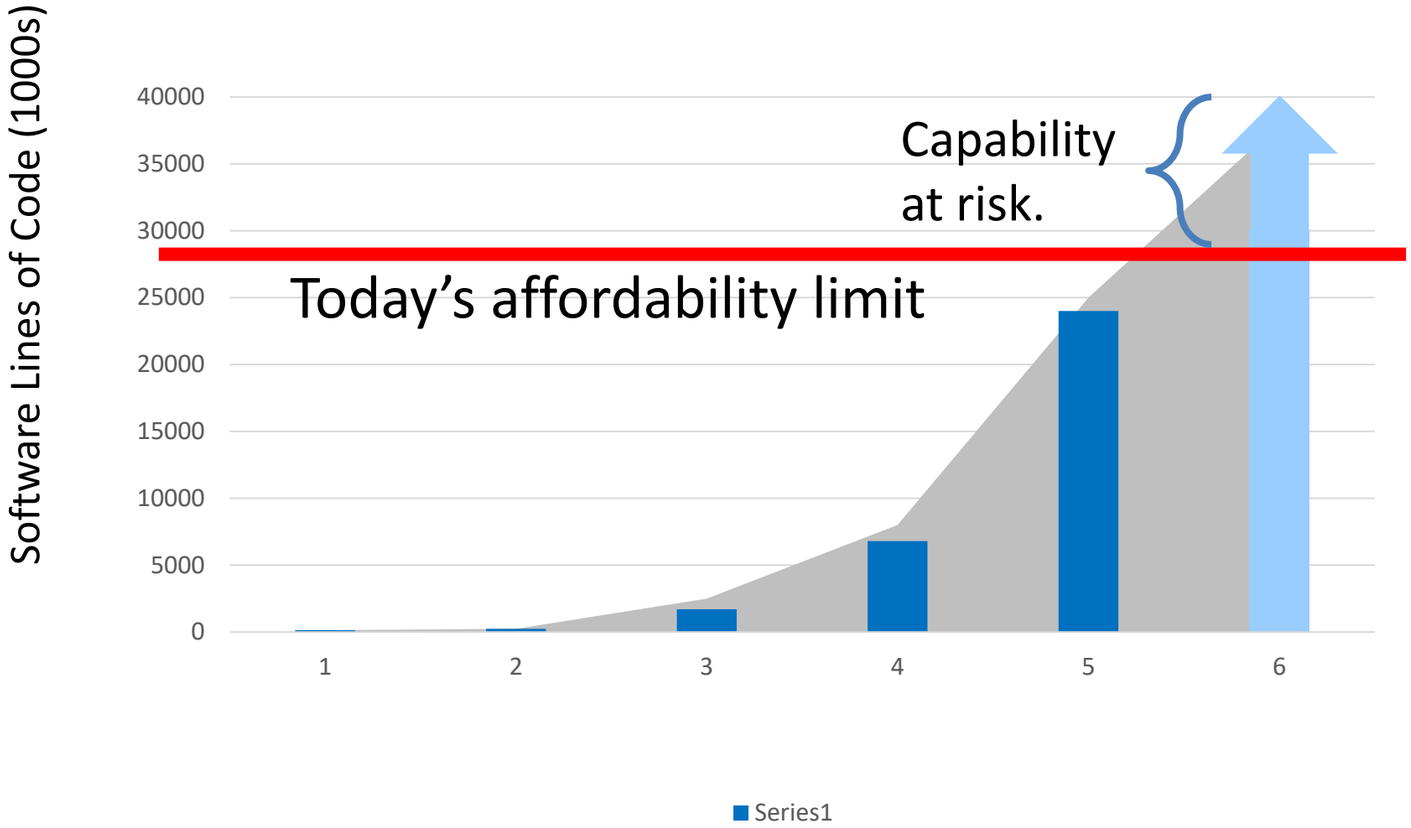
- **Leadership:** Defined ACVIP, Chairing AADL committee, Authored FACE/AADL annex, seL4 COE organizing committee
- **Expertise:** Architecture, Modeling, Design, Analysis, Training
- **Comprehensive:** System Modeling Language (SysML), Architecture Analysis and Design Language (AADL), Future Airborne Capability Environment (FACE), and others

ISOSCELES Embedded Platform

- Safe and secure embedded platform for medical devices and life-critical systems
- Highly disaggregated seL4-based design
- AADL-based specification – direct to code
- Code generation to secure separation kernel

The Problem

90% of modern Weapon System capability is provided by software

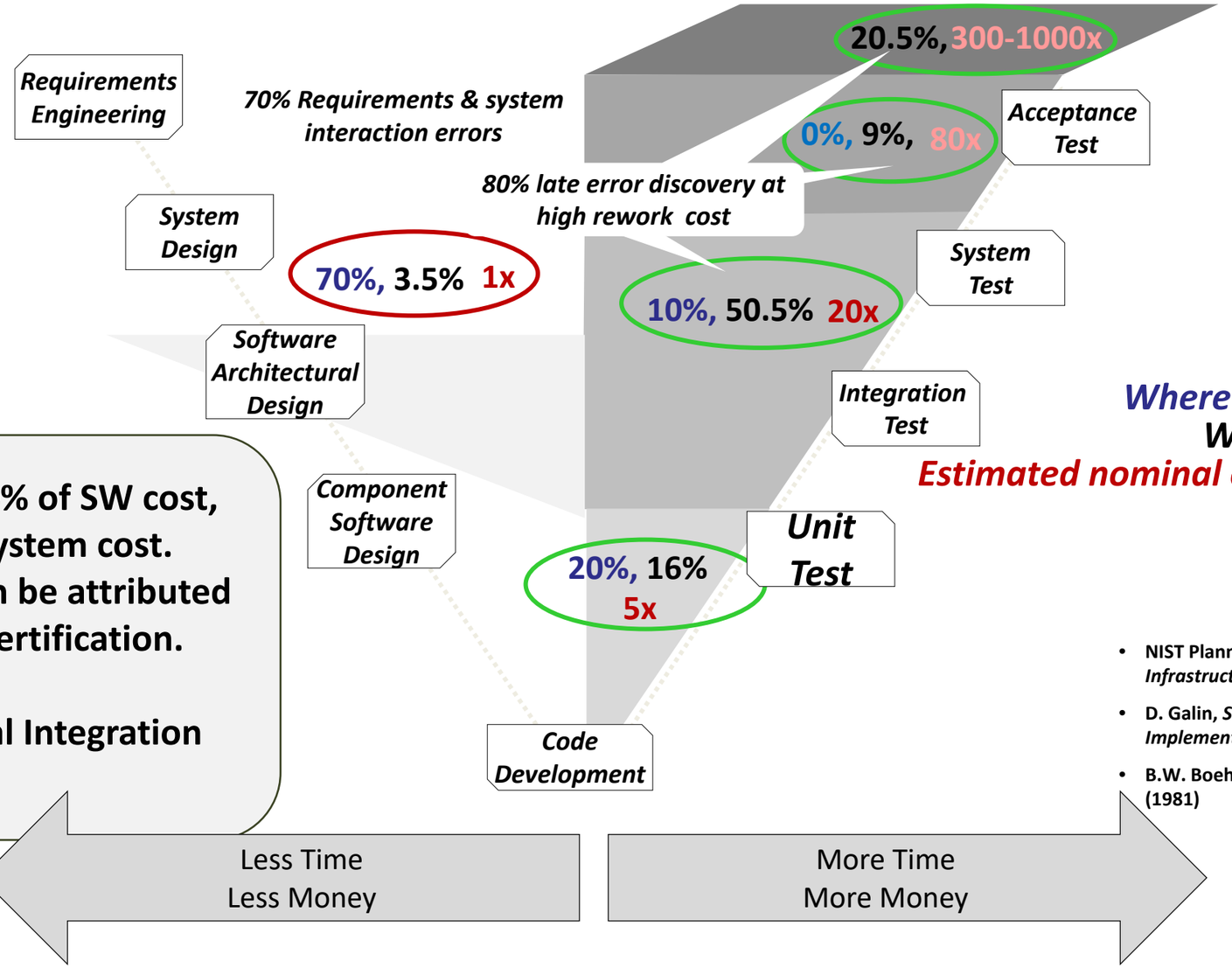


Affordability limit statistic from System Architecture Virtual Integration (SAVI) Study

Graphic adapted from Hagen/Sorenson "Delivering Military Software Affordably", Defense AT&L, Mar-Apr 2013

Current software practices seem to be hitting an affordability limit

Clues are in the Numbers



Rework & certification is 70% of SW cost, and software is 70% of system cost. Thus, 49% of system cost can be attributed to software rework and certification.

System Architecture Virtual Integration (SAVI) Study

Less Time
Less Money

More Time
More Money

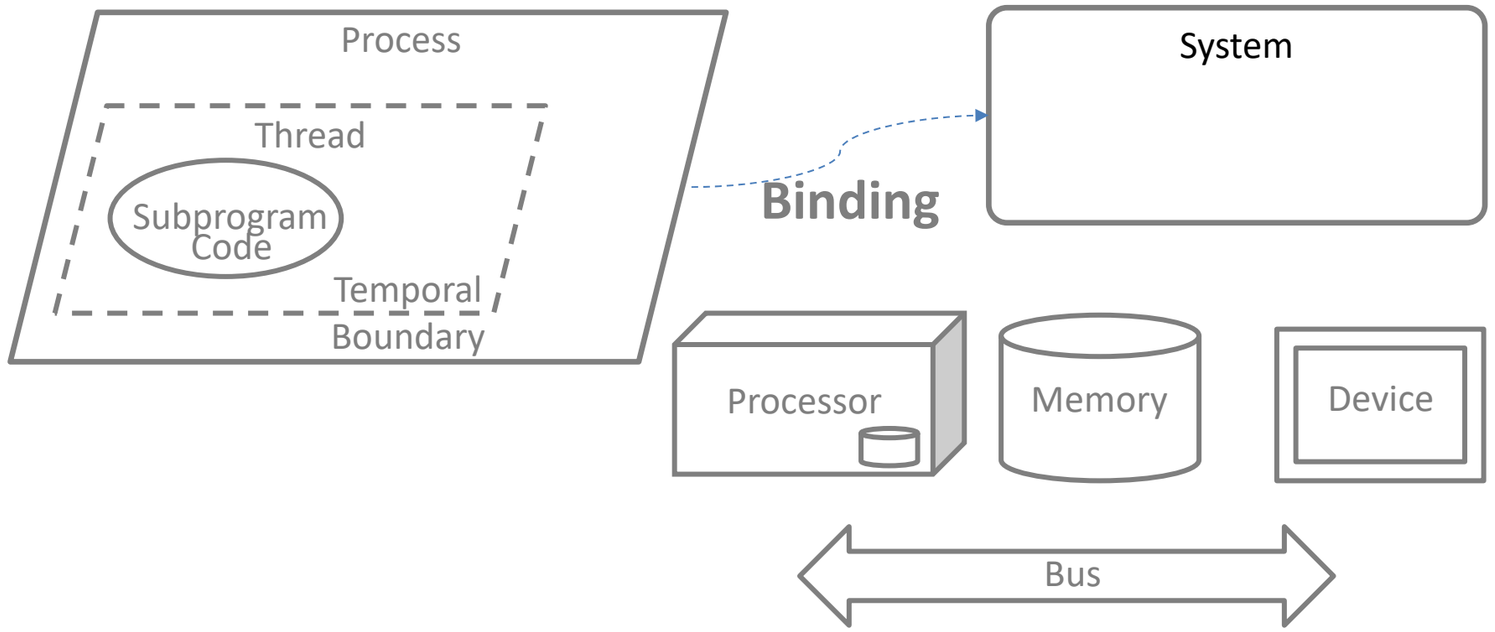
Sources:

- NIST Planning report 02-3, *The Economic Impacts of Inadequate Infrastructure for Software Testing,* May 2002.
- D. Galin, *Software Quality Assurance: From Theory to Implementation*, Pearson/Addison-Wesley (2004)
- B.W. Boehm, *Software Engineering Economics*, Prentice Hall (1981)

Dependable Systems and Model-based Systems Engineering help you “shift left”

Architecture Analysis & Design Language (AADL)

- Originated from DARPA, Standardized by SAE in 2004 (SAE AS 5506)
- Enables architectural analysis to predict the effects of integrating software, hardware and system components
- Strong, well-defined semantics promotes model exchange and reuse
- Deferred specification makes AADL easy to use throughout the design lifecycle
- Annexes address:
  - ARINC 653
  - Behavior
  - Communication
  - Code Generation
  - Error Modeling (Safety)
  - Requirements
  - Security



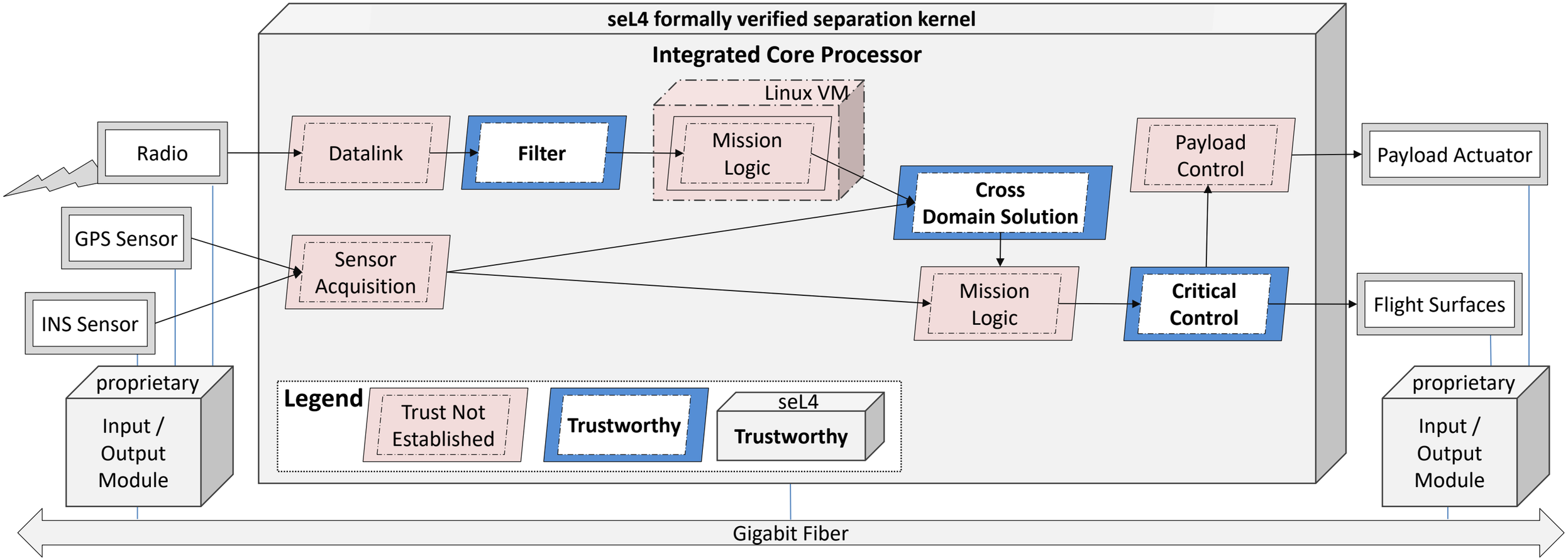
AADL picks up where FACE and SysML leave off

AADL Advantages

- It is **not vendor locked** - AADL models written in one tool will work in other AADL environments
- It is **human readable**
- It is designed to help **identify integration issues** in cyber-physical systems
- It enables analyses on **compositional architectures**

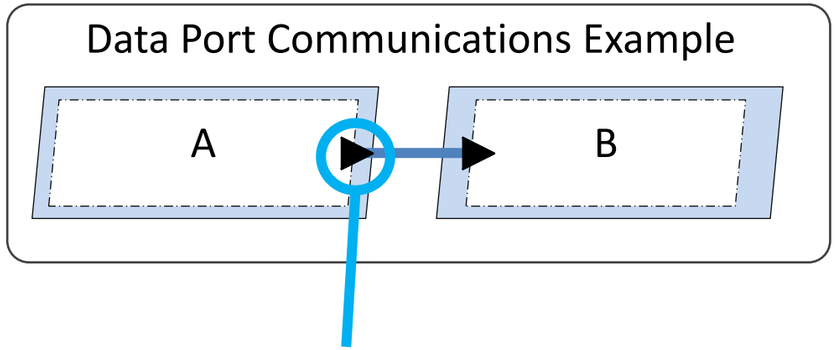
AADL is a well defined language for description and analysis of real time, embedded systems

Motivating Example



Consider re-engineering an existing mission system

Communications Abstraction

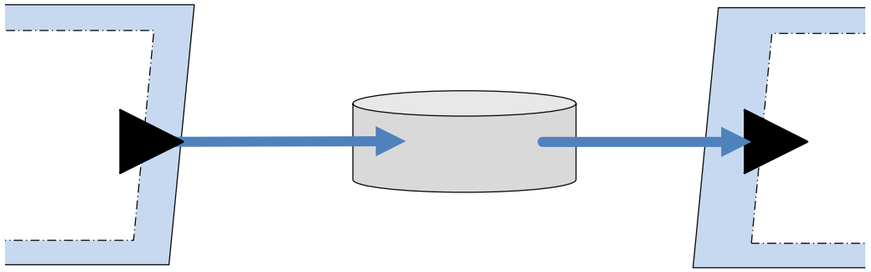


What does it really mean when “A sends a message to B?” What behavior is allowable?

Connections are unidirectional – no back-channel communications allowed

Implemented as seL4 shared memory

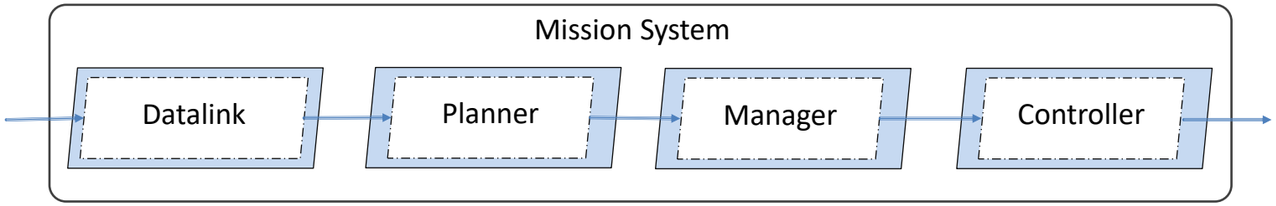
No RPC, which mixes control concepts



Writer (sender) has write access, and writing occurs during writer’s execution time
Reader (receiver) has read access, and reading occurs during reader’s execution time
Reads and writes are non-blocking

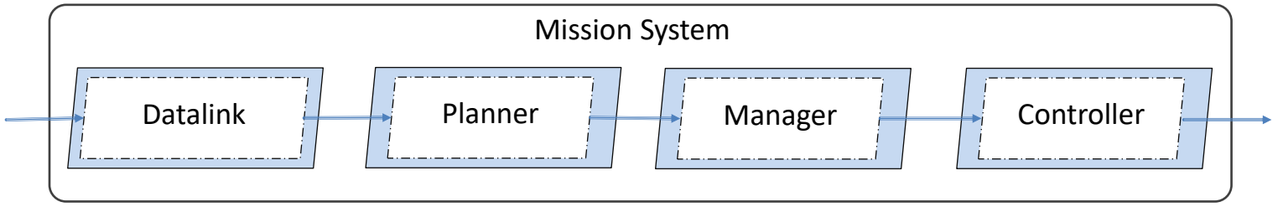
Communications bindings for CAmkES automatically generated for C by KSU’s HAMR

Basic System



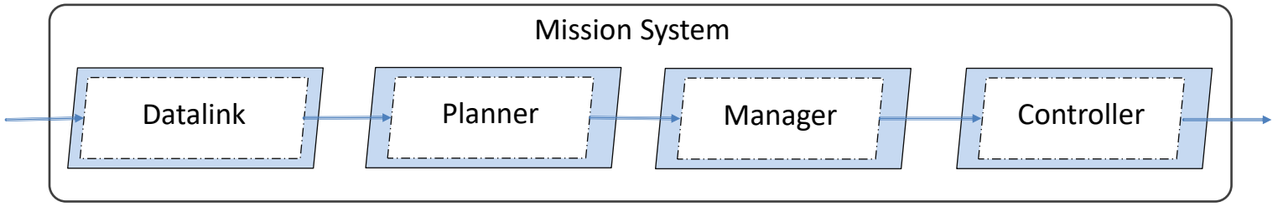
We can quickly prototype this in Linux, and things just work

Basic System



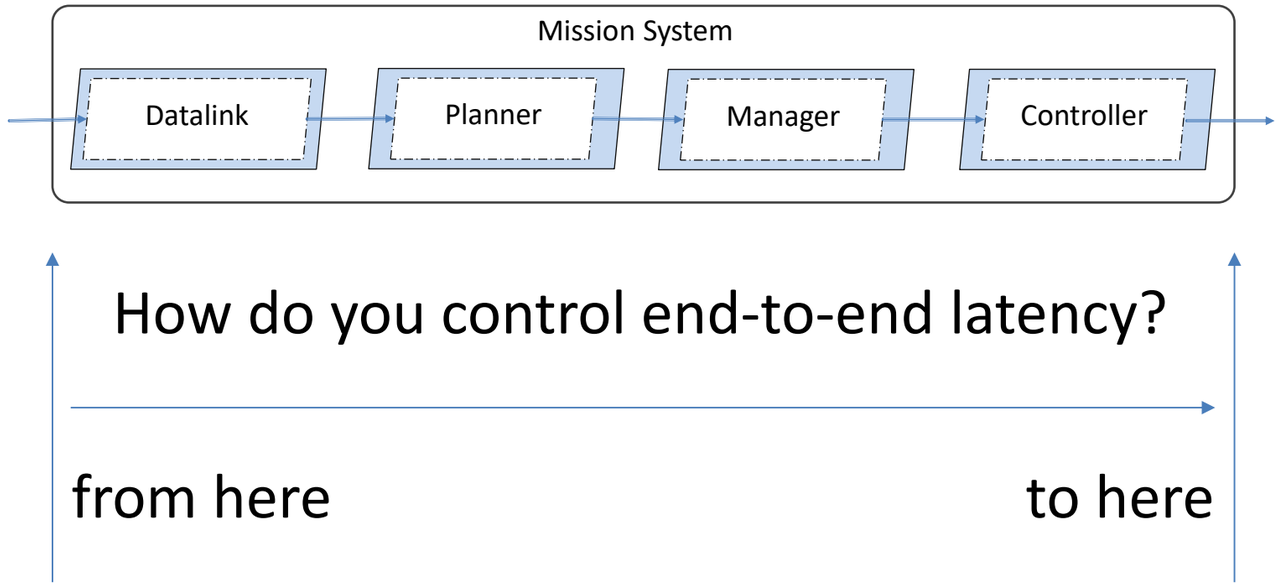
Usually

Basic System



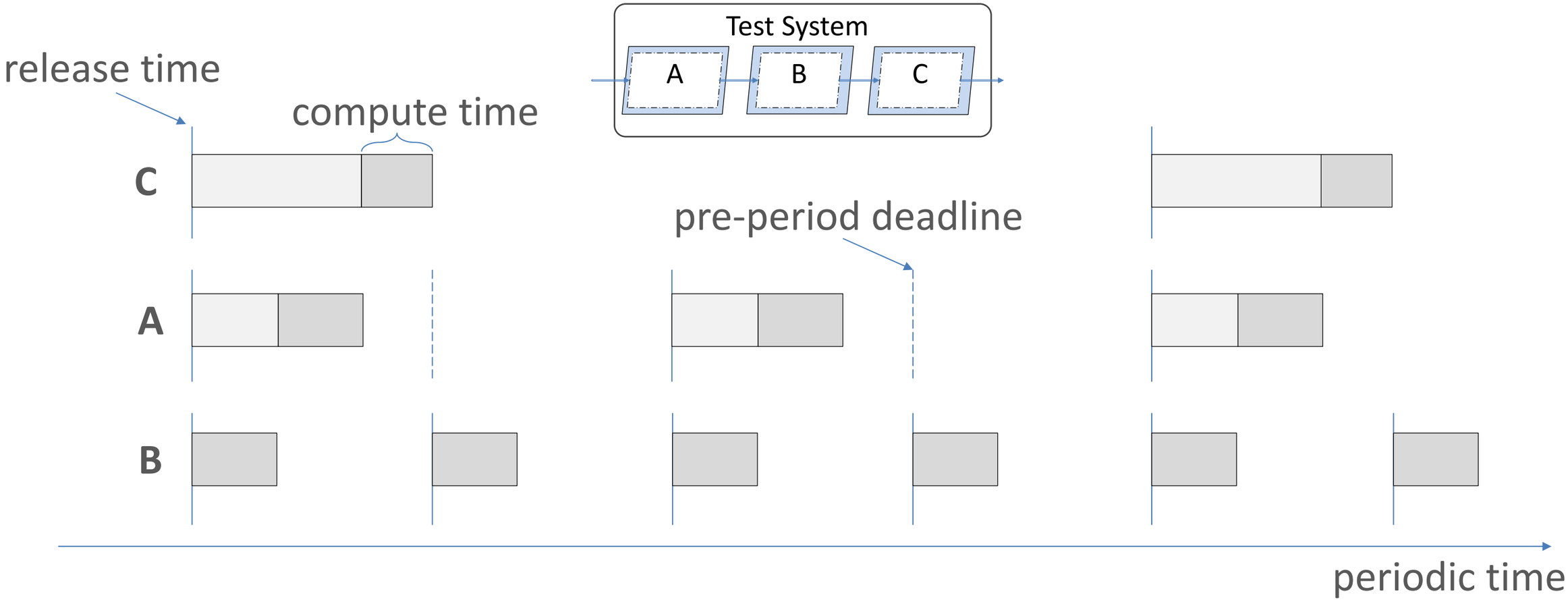
Except when they don’t

Example Temporal Property: Latency



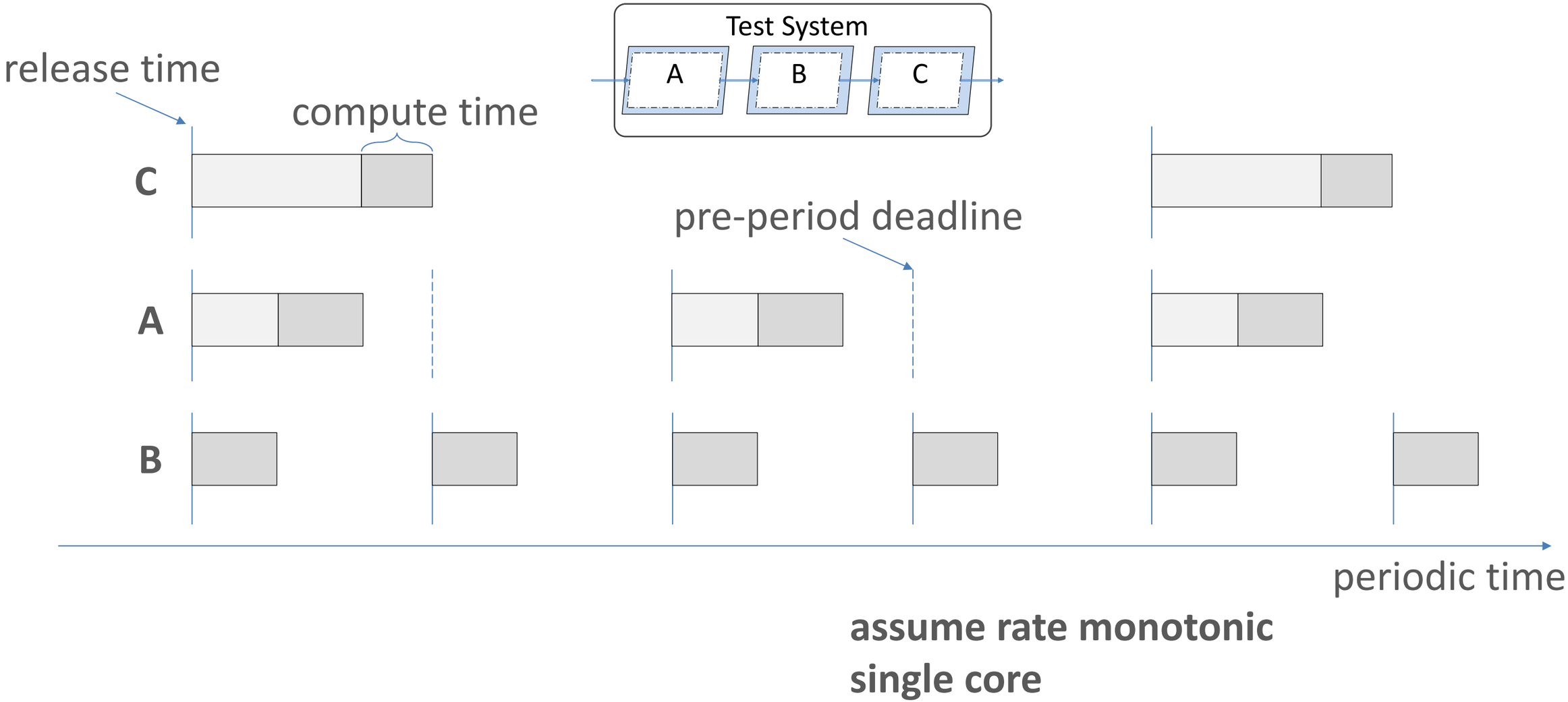
How do you know how this system will behave during runtime?

Latency Challenge



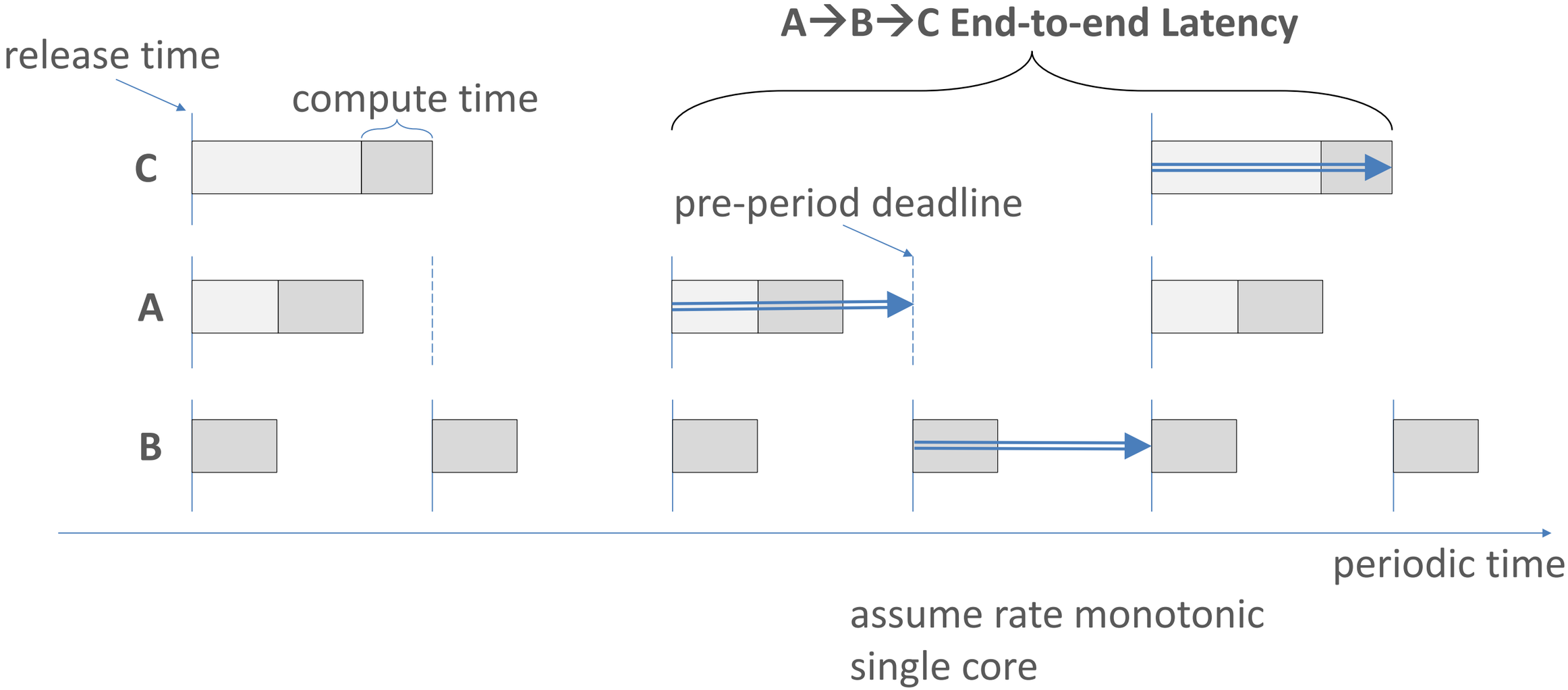
What is the end-to-end latency of A→B→C?

Latency Challenge



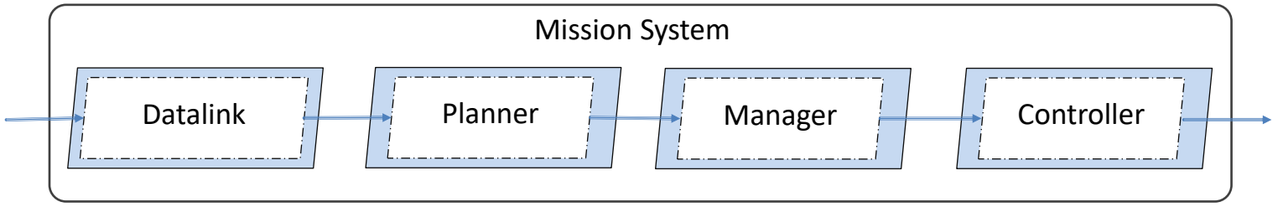
What is the end-to-end latency of A→B→C?

Latency Challenge



What about other temporal properties?

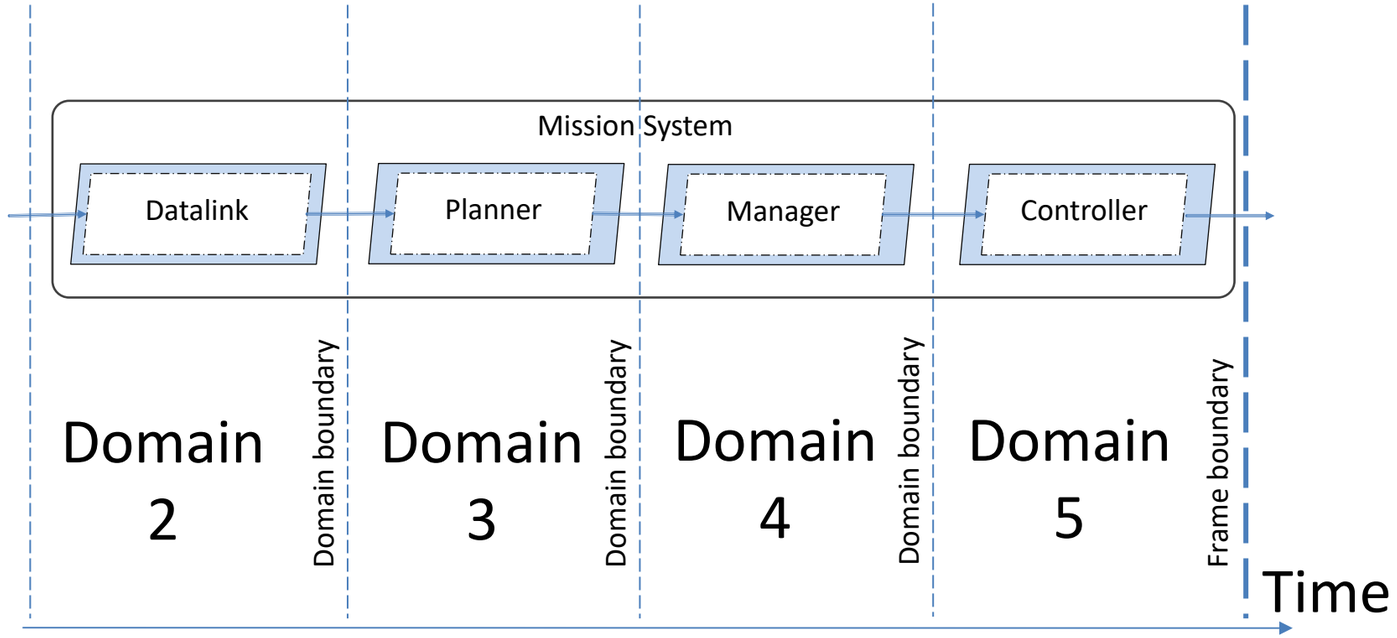
Example Temporal Concerns



- How do you control
  - end-to-end latency?
  - rate and throughput?
  - jitter and minimize race conditions?
- What happens when a failed or compromised task
  - dominates the CPU?
  - induces subtle jitter?

The seL4 **verified kernel** provides basic mechanisms for temporal isolation.

Put Each Thread in an seL4 Domain

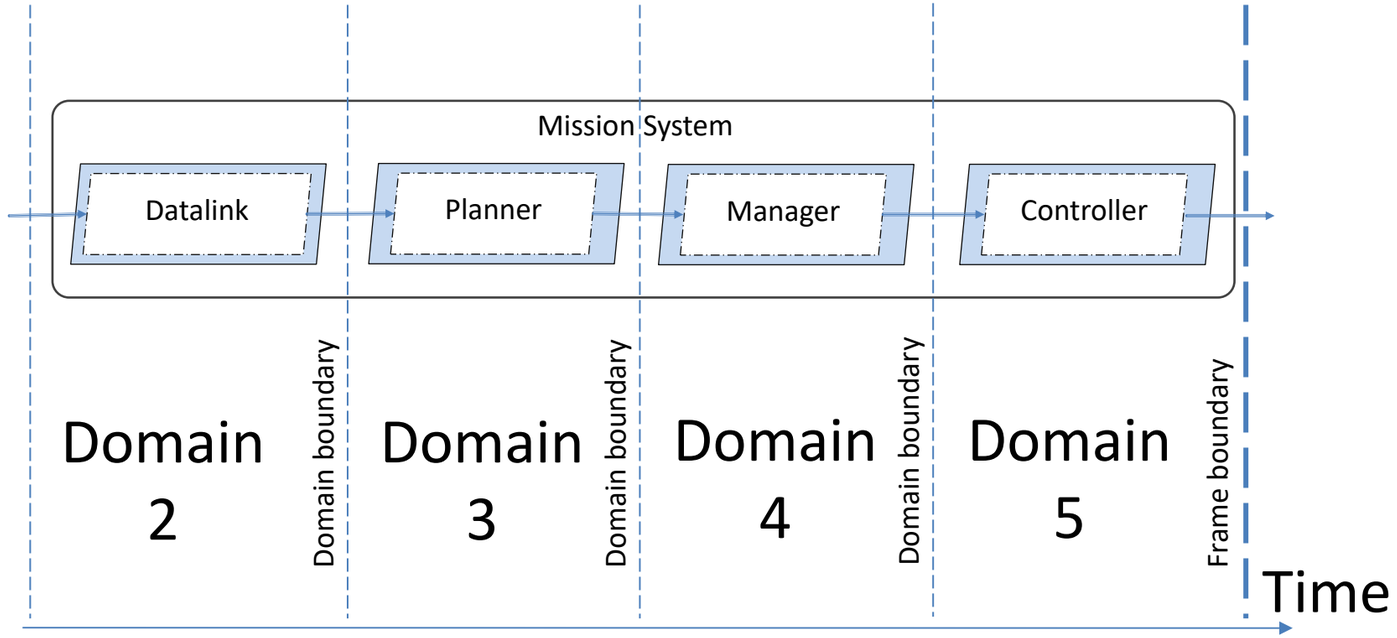


Temporal requirements specification:

```
thread Planner_thread
  properties
    Dispatch_Protocol => Periodic;
    Period => 40ms;
    Compute_Execution_Time => 8ms .. 8ms;
...
process Planner_process
  properties
    CASE_Scheduling::Domain => 3;
```

Put each thread in its own seL4 domain

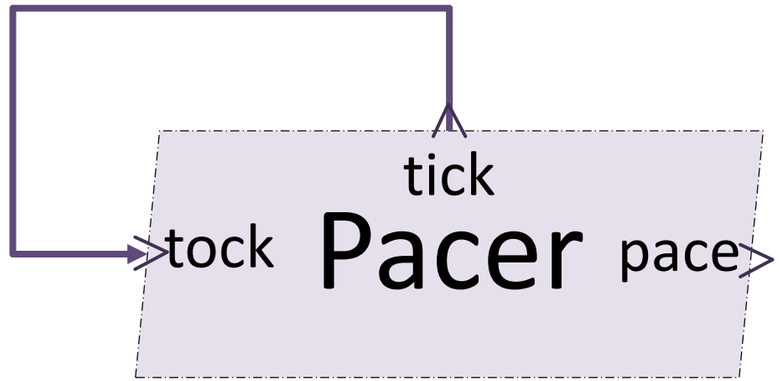
When To Sample the Data Port?



The verified kernel does not yet have a “start of period”

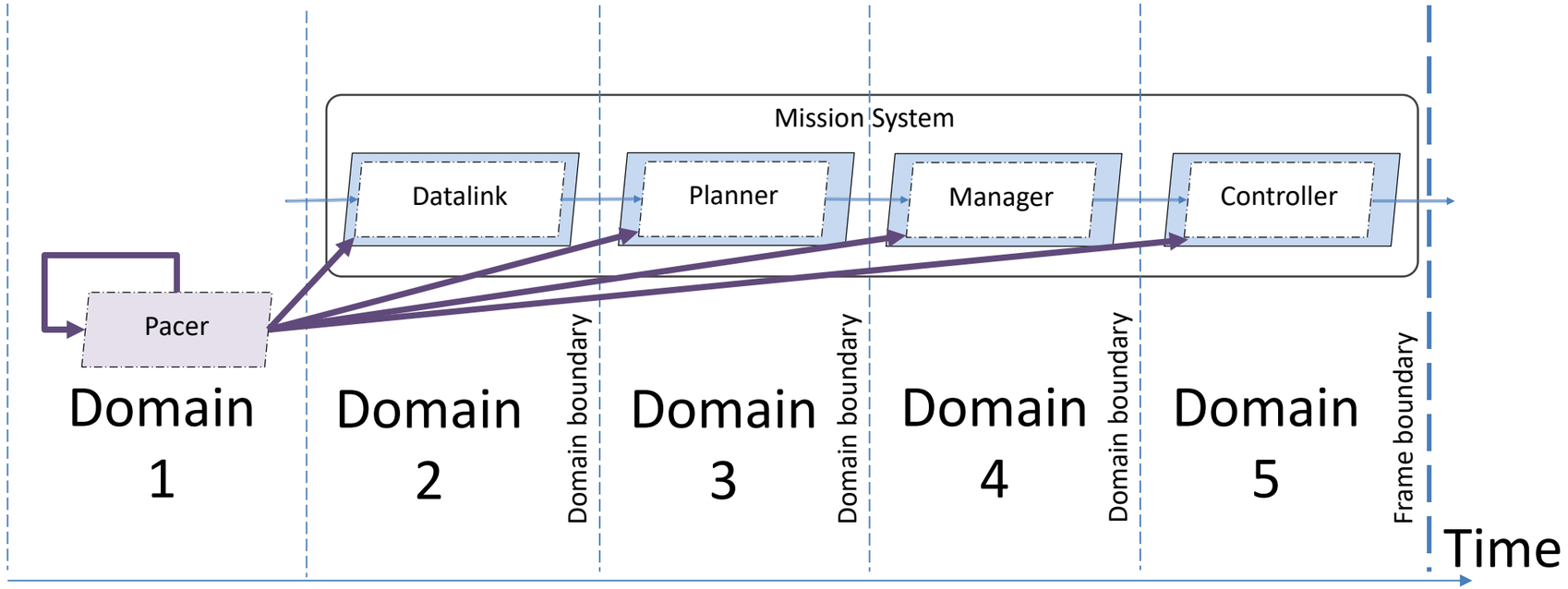
Pacer


```
Initialize
emit pacing_event on output “tick”,
  which is connected to input “tock”
while true {
  wait for pacing_event
  emit pacing_event
}
```



Since cross-domain IPC is delayed until a domain switch, this fires once per domain slice

Pacing Signals “start of period”

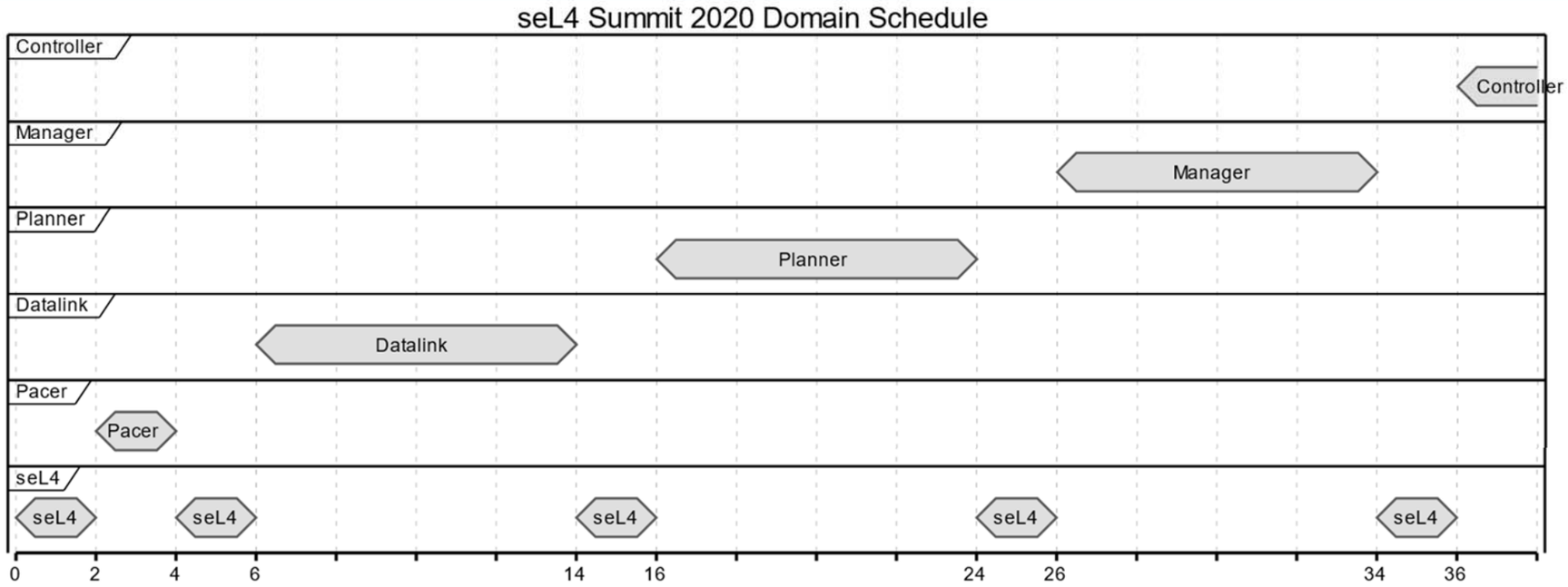


Eliminates busy-wait spin

Supports “background” processing in each domain slice

HAMR generates all the infrastructure

seL4 Domain Schedule




```
const dschedule_t ksDomSchedule[] = {
    { .domain =  0, .length = 1 },    // 000 - 002 ms: 2 ms seL4
    { .domain =  1, .length = 1 },    // 002 - 004 ms: 2 ms Pacer
    { .domain =  0, .length = 1 },    // 004 - 006 ms: 2 ms seL4
    { .domain =  2, .length = 4 },    // 006 - 014 ms: 8 ms Datalink
    { .domain =  0, .length = 1 },    // 014 - 016 ms: 2 ms seL4
    { .domain =  3, .length = 4 },    // 016 - 024 ms: 8 ms Planner
    { .domain =  0, .length = 1 },    // 024 - 026 ms: 2 ms seL4
    { .domain =  4, .length = 4 },    // 026 - 034 ms: 8 ms Manager
    { .domain =  0, .length = 1 },    // 034 - 036 ms: 2 ms seL4
    { .domain =  5, .length = 2 },    // 036 - 040 ms: 4 ms Controller
};
```

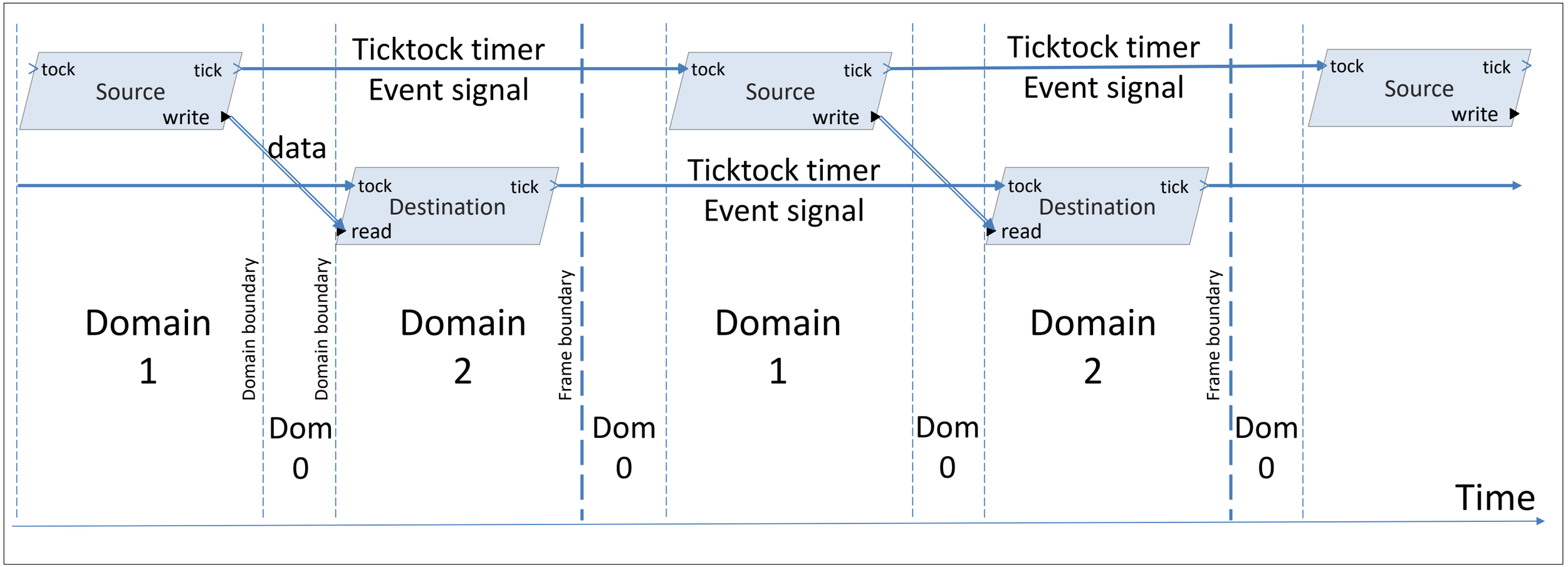
Self Pacing Infrastructure


```
int run(void) {
  {
    int64_t sb_dummy;
    sb_entrypoint_source_thread_impl_initializer(&sb_dummy);
  }
  tick_emit();
  for(;;) {
    tock_wait();
    {
      int64_t sb_dummy = 0;
      sb_entrypoint_period_source_thread_impl(&sb_dummy);
    }
    tick_emit();
  }
  return 0;
```

Users write modular code and don’t have to see this behind-the-scenes magic

Simple Source → Destination Model

Self-Pacing Unrolled



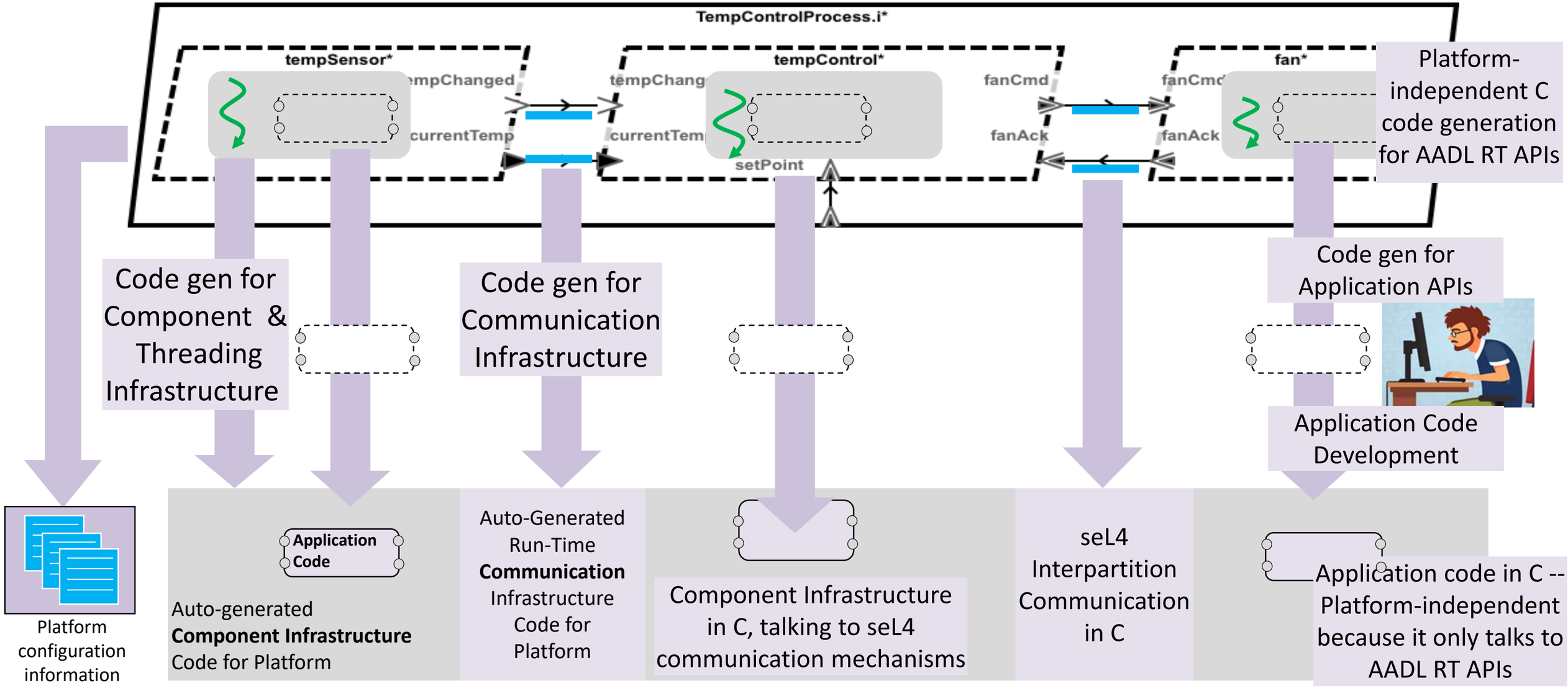
Currently verifying pacing code under new DARPA SBIR

Real World Considerations

- Tiny hardware buffers might require high service rate to prevent over/underflows → Use MCS
- Virtual Machines take a long time to initialize and require significant resources from Domain 0
- Frame overflow – domain provides containment
- Frame underflow – background processing opportunity

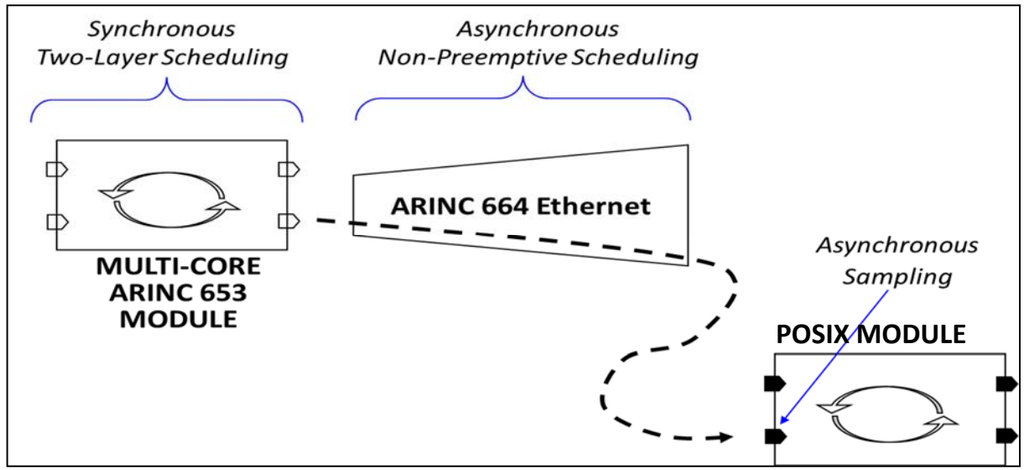
Useful approach to build deterministic systems with fewer integration challenges

KSU HAMR Code Generation for seL4

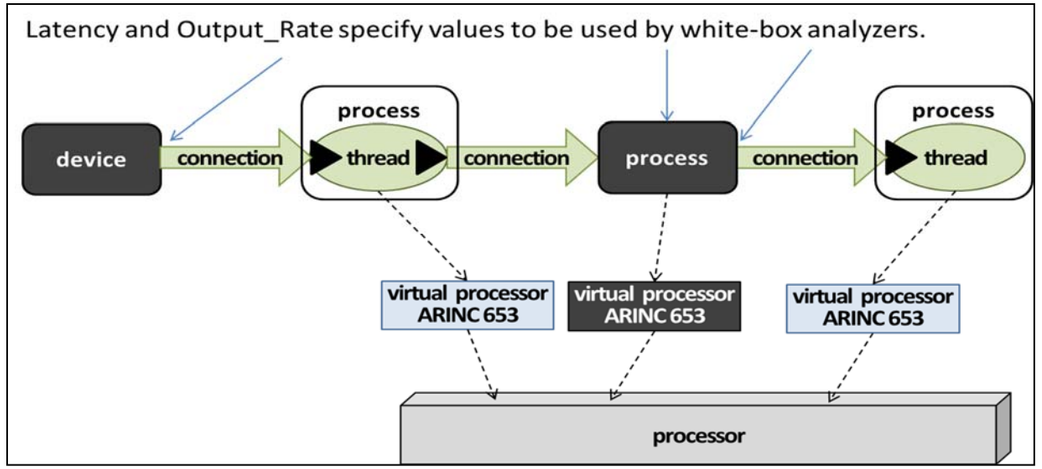


Multiple targets: CAmkES, Minix, Linux

FASTAR Schedule Generation

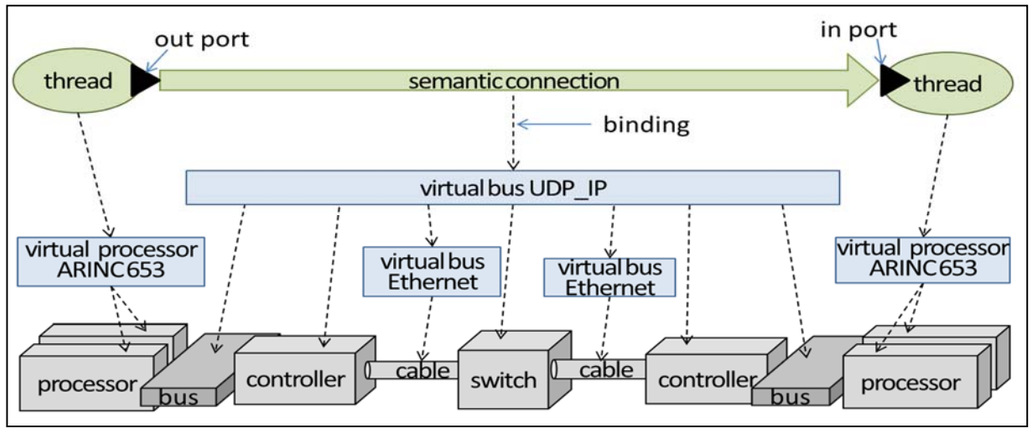


Heterogeneous Architectures



Mixed Fidelity Models

FASTAR analyzes resource needs and timing properties of complex architecture models as they evolve through multiple development phases.

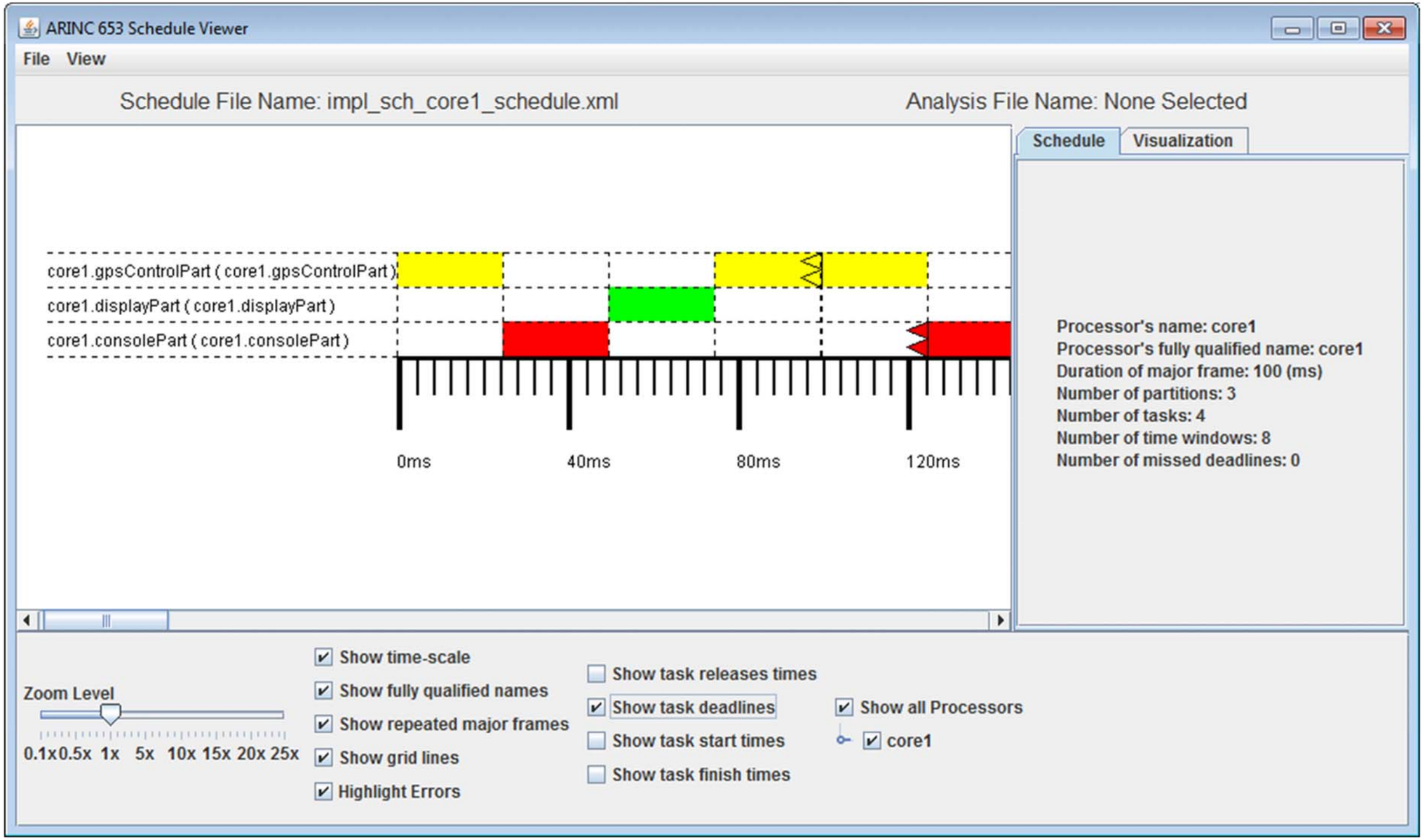


Layered Architectures

FASTAR is itself a framework that integrates multiple other back-end tools selected to address the needs of a specific product line.

FASTAR can also generate schedules

Prototype ARINC653 Schedule Display



Continuous Virtual Integration

Models of Systems
(e.g., hardware, software, requirements)

AADL Models

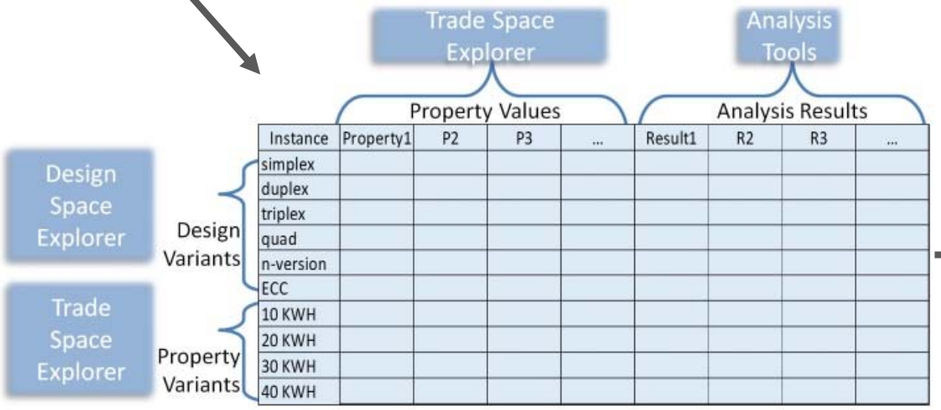
SysML Models

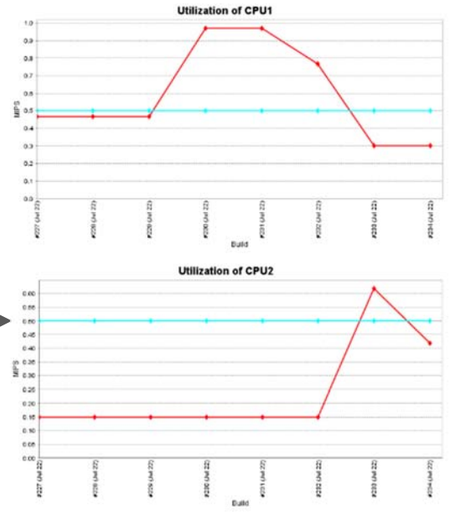
(1) Designer specifies:
- Top level design
- Design variants to evaluate
- Parametric studies
- Analysis tools to apply

- Risk Management Analysis
- Systems-Theoretic Process Analysis
- Reliability Analysis
- Power & Weight Analysis
- Latency Analysis
- Schedulability Analysis

CAFFMAD

(2) As models are subsequently updated and refined during the design process, CAFFMAD:
- Automatically rebuilds design variants
- Applies analysis tools to each variant
- Tabulates results for trade study





Track model updates over time and revalidate earlier decisions

Summary Capabilities

- Static cyclic slices with the verified seL4 kernel
- seL4 MCS, verification coming soon, has significant additional capability
- Scheduling models within the domain slice include
  - Paced deterministic behavior across platforms or
  - Event-triggered dynamic dispatch within slice
- Need to balance the following requirements in one schedule
  - Initialization
  - Long duration threads
  - High rate threads

The current domain scheduler is already straightforward to use