Validation of Safety-Critical Systems with AADL

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213

Peter H Feiler
April, 2008

Outline

Multiple aspects of system validation

System & software engineers working together

Multi-fidelity model-based analysis

Property preserving transformations

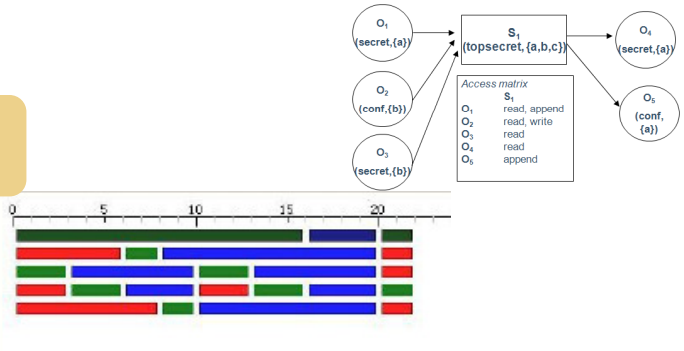
Conclusions

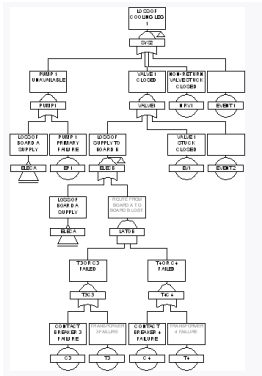
Dimensions of System Validation

Validation of models against system

The system

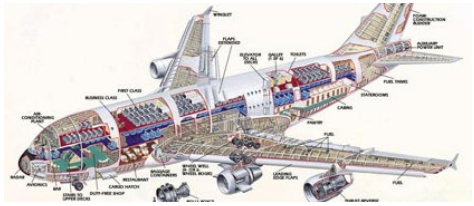
Model-based validation of system





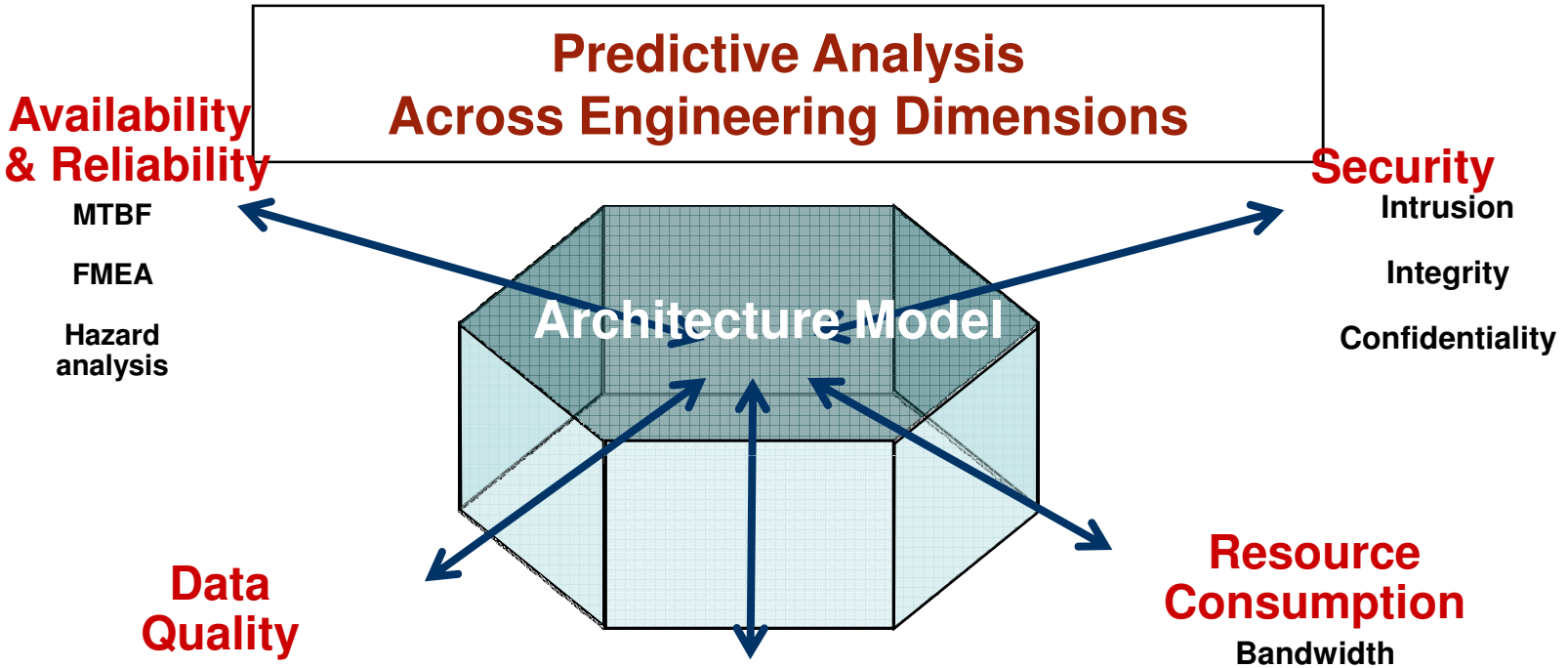
System models

Validation of implementation against system models



System implementation

Single Source Annotated Architecture Model

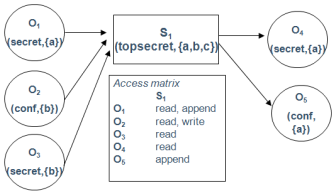


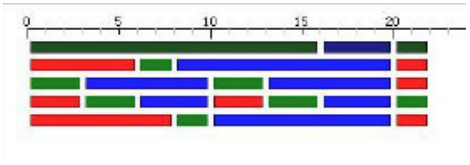
Fewer independently developed models reduces model validation

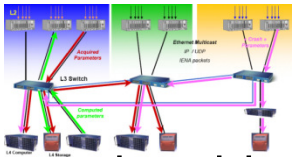
Low incremental cost for additional analyses & simulations

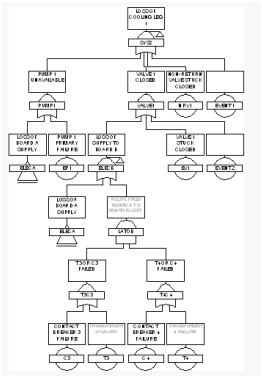
Architecture-Driven Modeling

Automatically derived analytical models



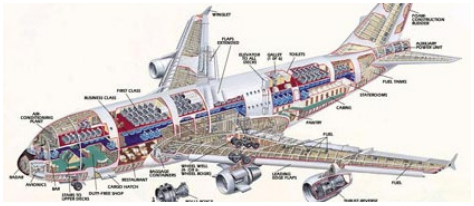






Annotated architecture

System generation from validated models



Validation of generators

AADL and Safety-Criticality

Fault management

- Architecture patterns in AADL
  - Redundancy, health monitoring, …
- Fault tolerant configurations & modes

Dependability

- Error Model Annex
- Specification of fault occurrence and fault propagation information
- Use for hazard and fault effect modeling
- Reliability & fault tree analysis

Behavior validation

- Behavior Annex
- Model checking
- Source code validation

Outline

Multiple aspects of system validation

System & software engineers working together

Multi-fidelity model-based analysis

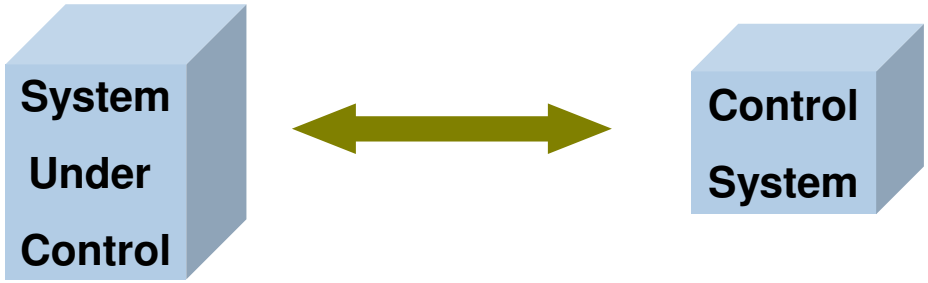
Property preserving transformations

Conclusions

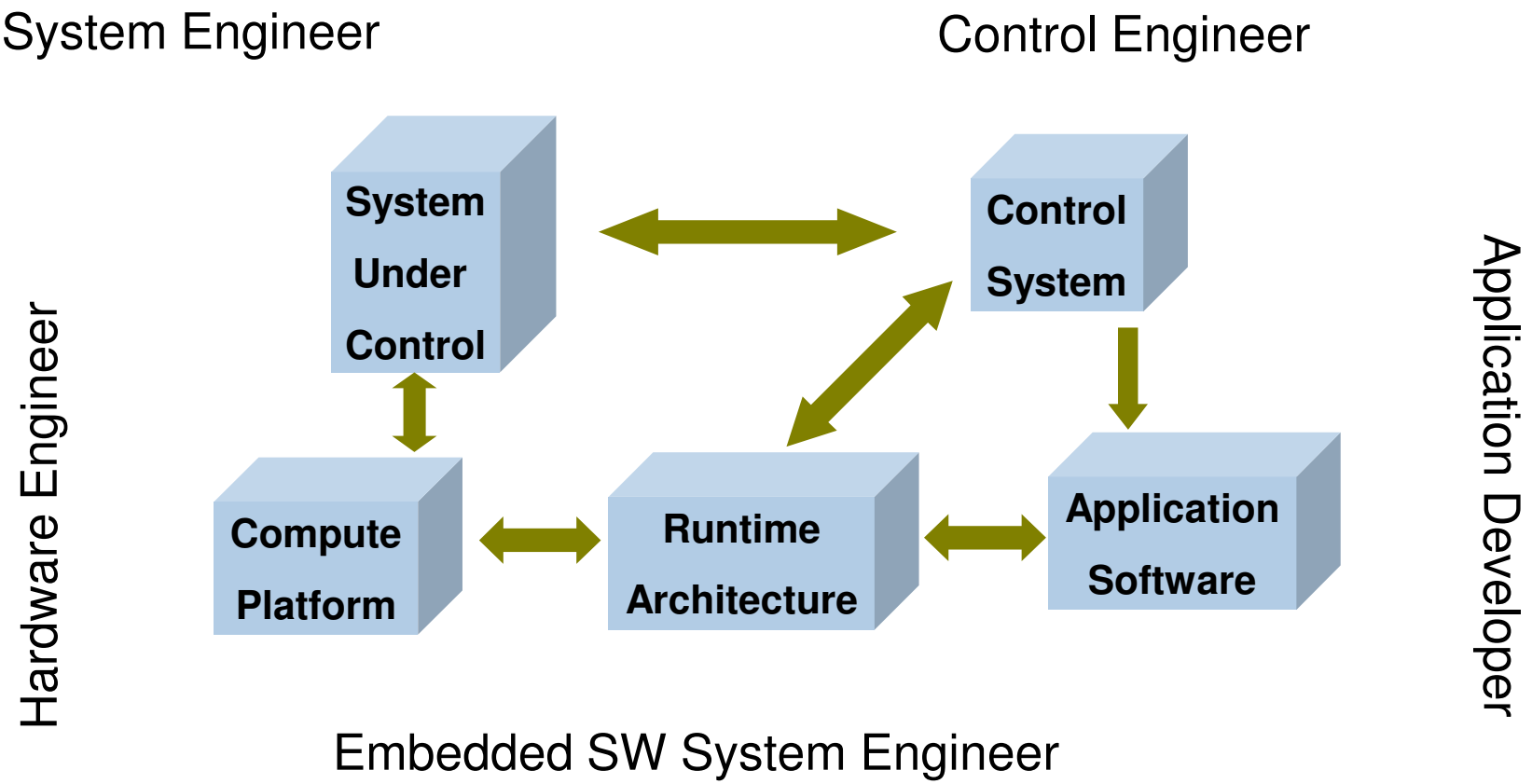
Traditional Embedded System Engineering

System Engineer

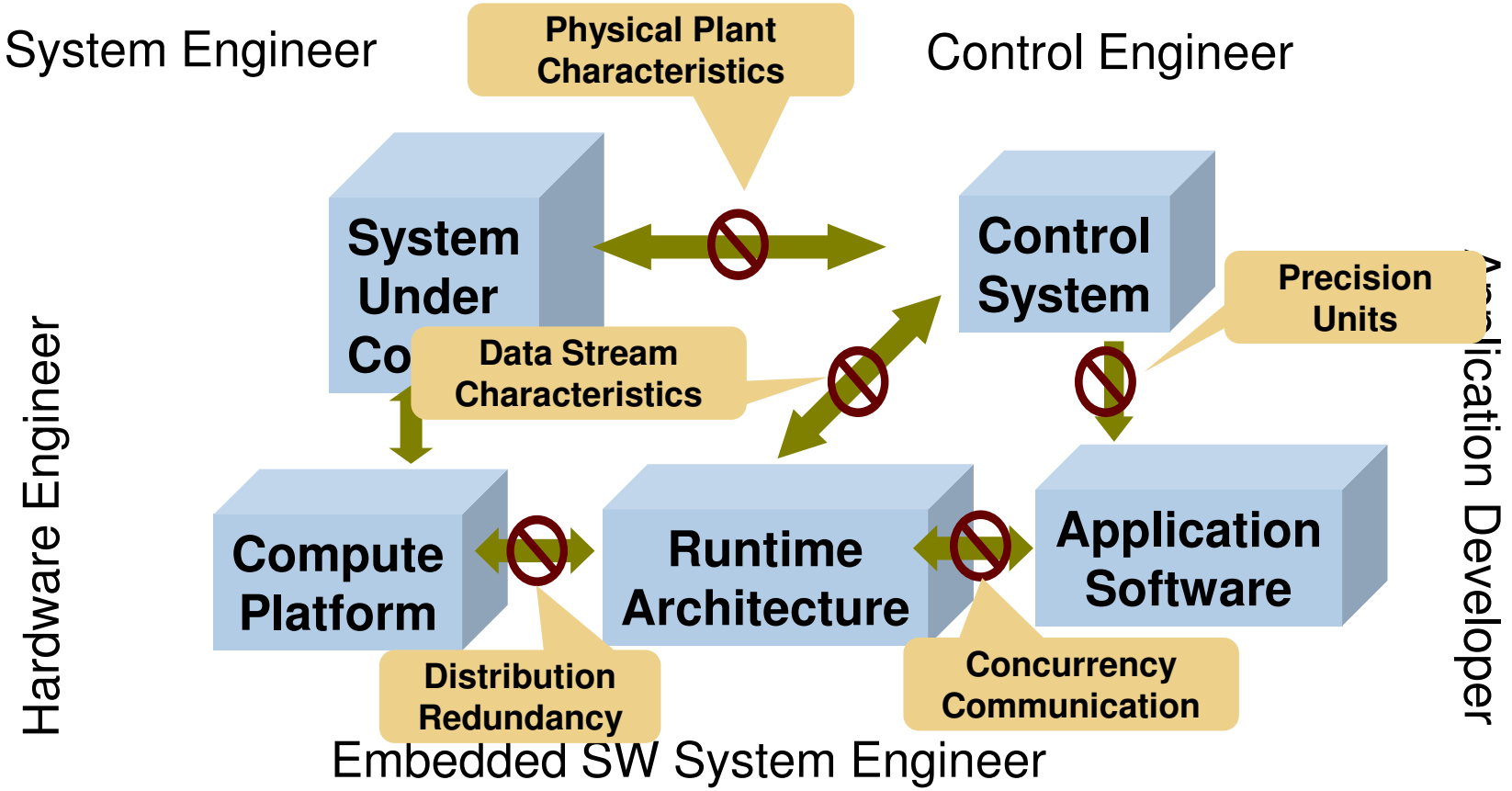
Control Engineer



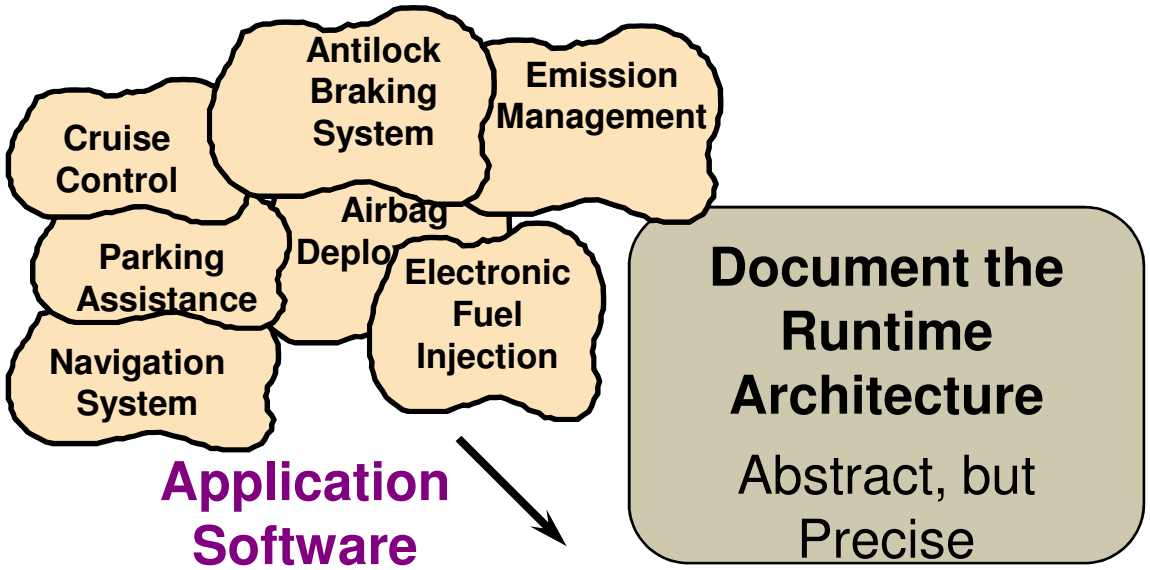
Software-Intensive Embedded Systems



Mismatched Assumptions



Predictable Embedded System Engineering



Application Software

Document the Runtime Architecture

Abstract, but Precise

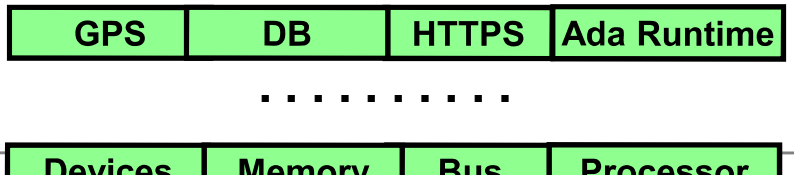
System Analysis

- Schedulability
- Performance
- Reliability
- Fault Tolerance
- Dynamic Configurability

SAE AADL

External Environment

Execution Platform



System Construction

- AADL Runtime System
- Application Software Integration

Working Together

Conceptual architecture

- UML-based component model
- Architecture views (DoDAF, IEEE1471)
- Platform independent model (PIM)

System engineering

- SysML as standardized UML profile
- Focus on system architecture and operational environment

Embedded software system engineering

- SAE AADL
- OMG MARTE profile based on AADL
- AADL as MARTE sub-profile
- Non-functional properties require deployment on platform

Data modeling

- UML, ASN,, …

Outline

Multiple aspects of system validation

System & software engineers working together

Multi-fidelity model-based analysis

Property preserving transformations

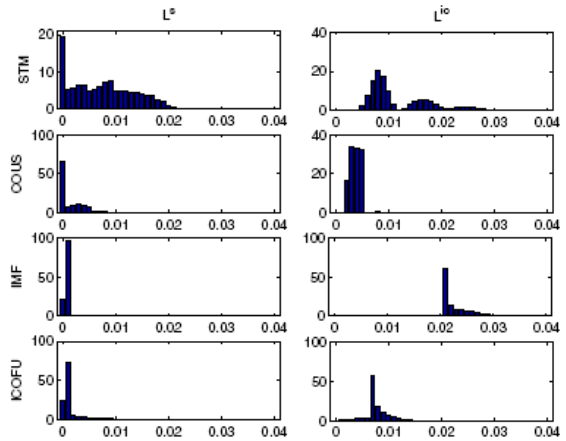
Conclusions

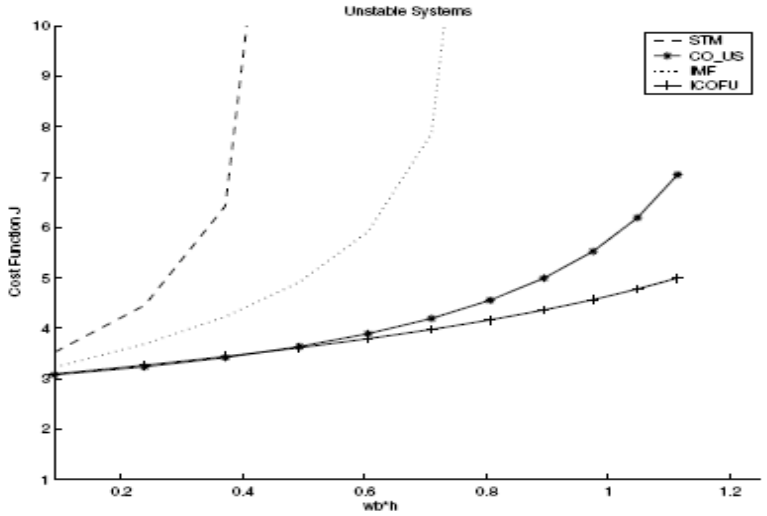
Impact of Sampling Latency Jitter

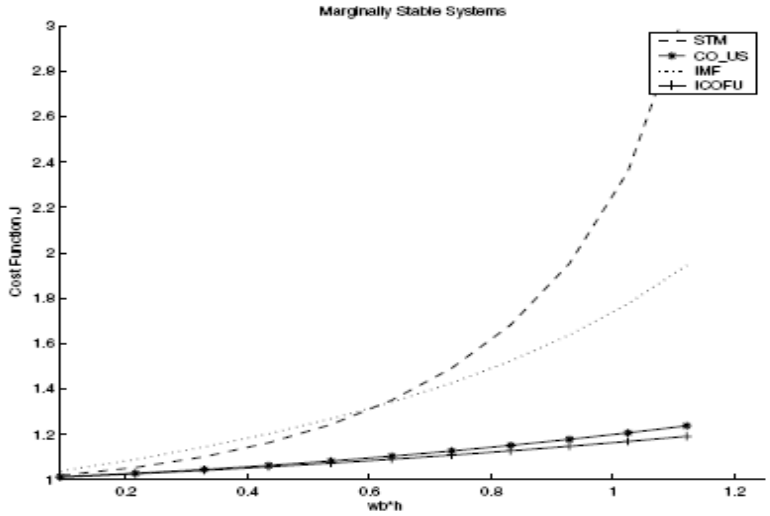
Impact of Scheduler Choice on Controller Stability

- A. Cervin, Lund U., CCACSD 2006

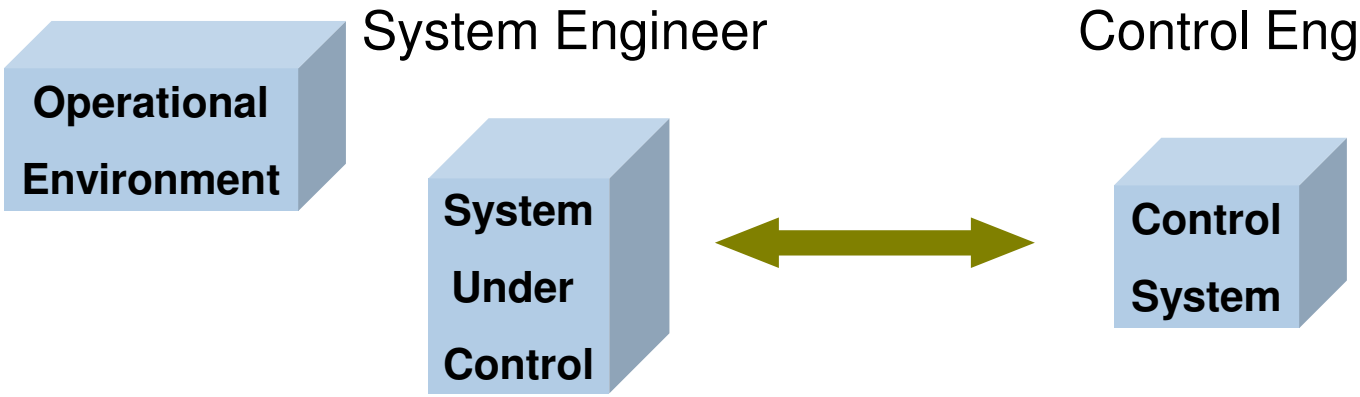
Sampling jitter due execution time jitter and application-driven send/receive







Latency Contributors

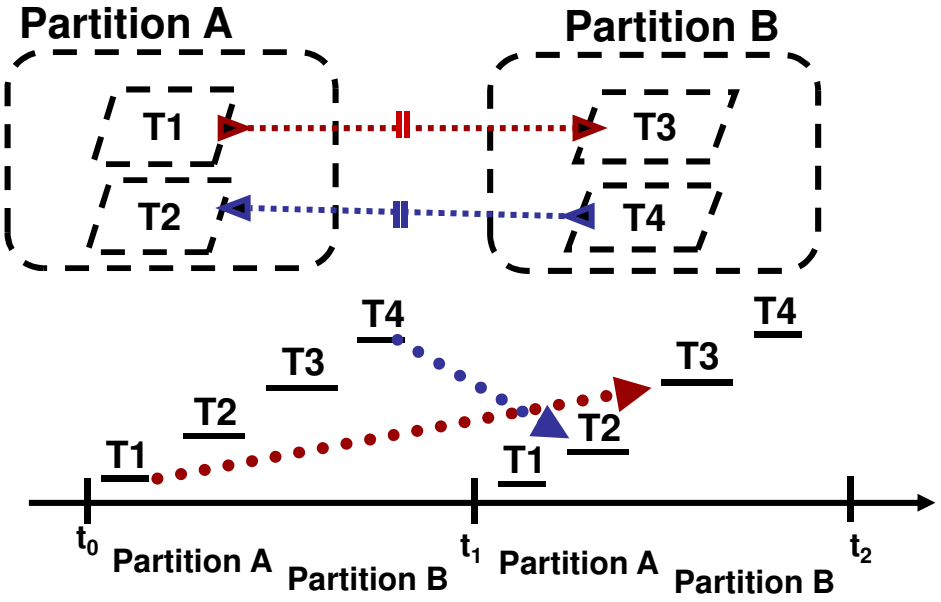


- Processing latency
- Sampling latency
- Physical signal latency

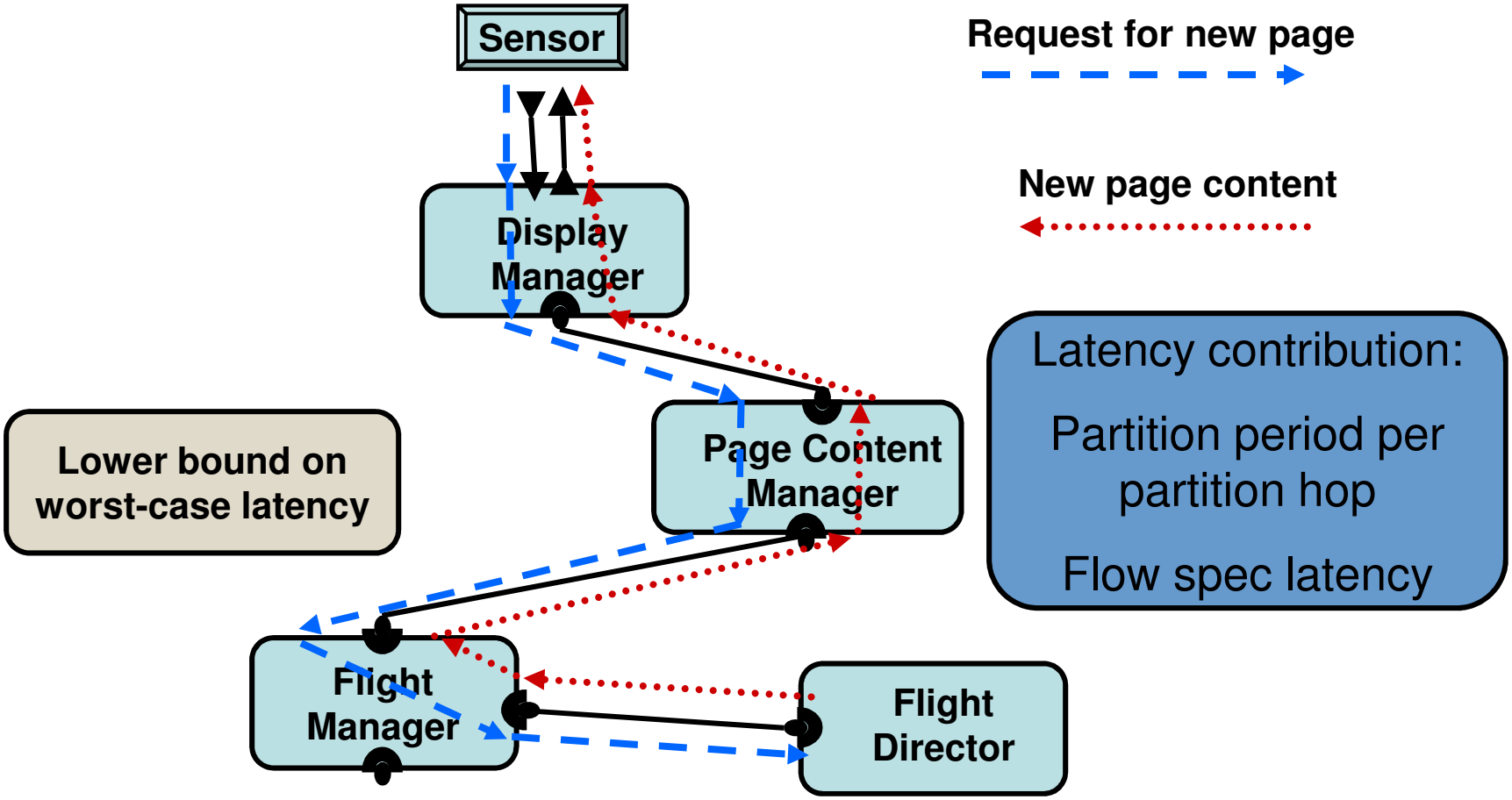
ARINC 653 Partitions & Communication

Frame-delayed inter-partition communication

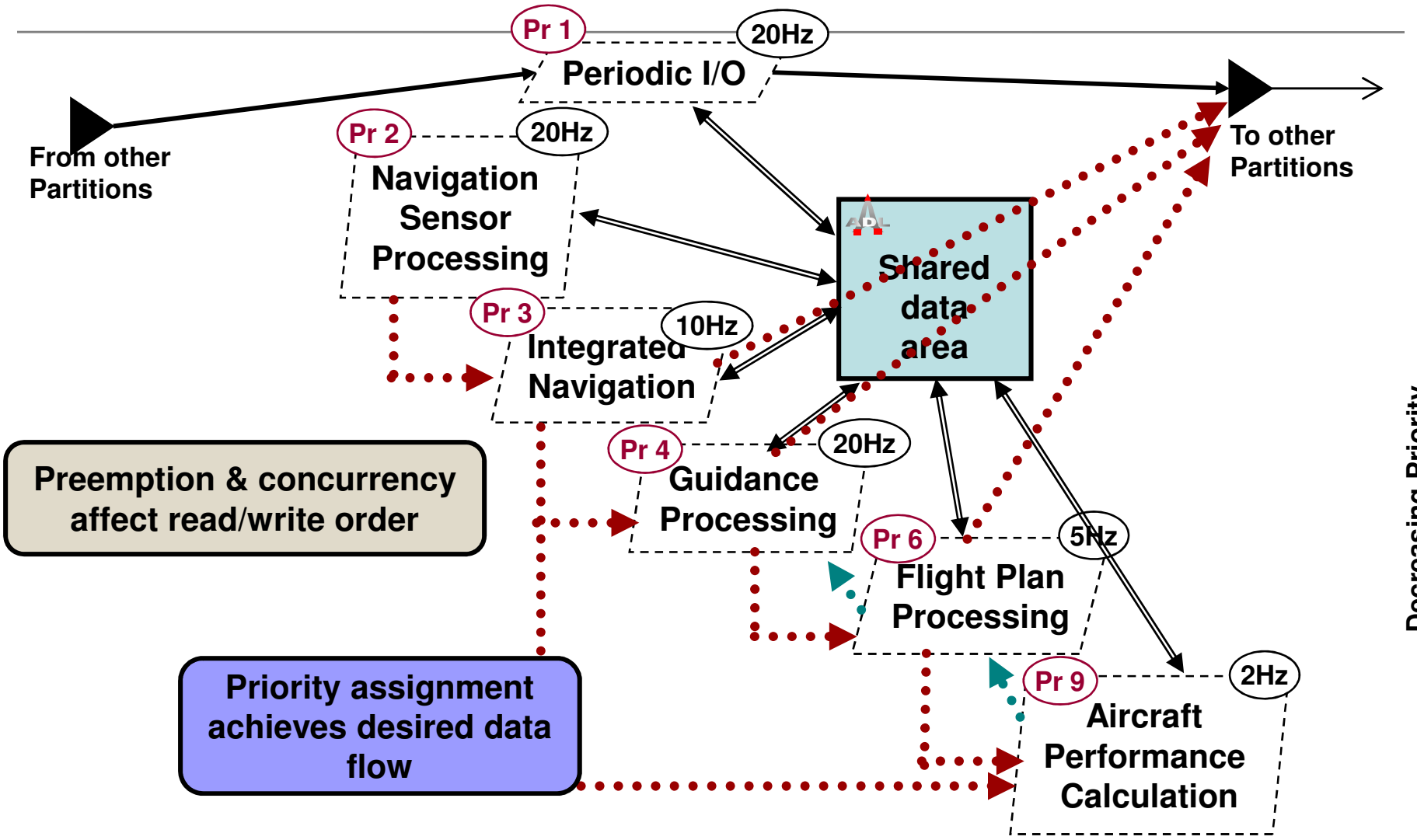
Timing semantics are insensitive to partition order



Latency Impact of Partitions



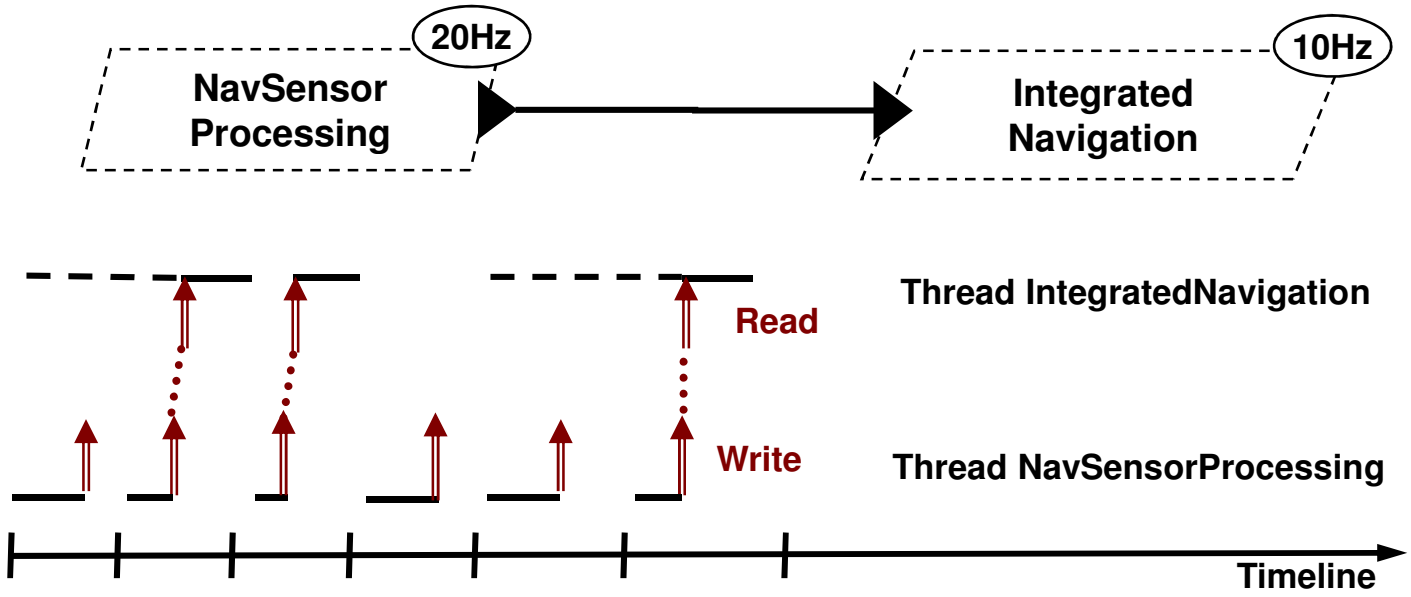
Intended Data Flow in Task Architecture



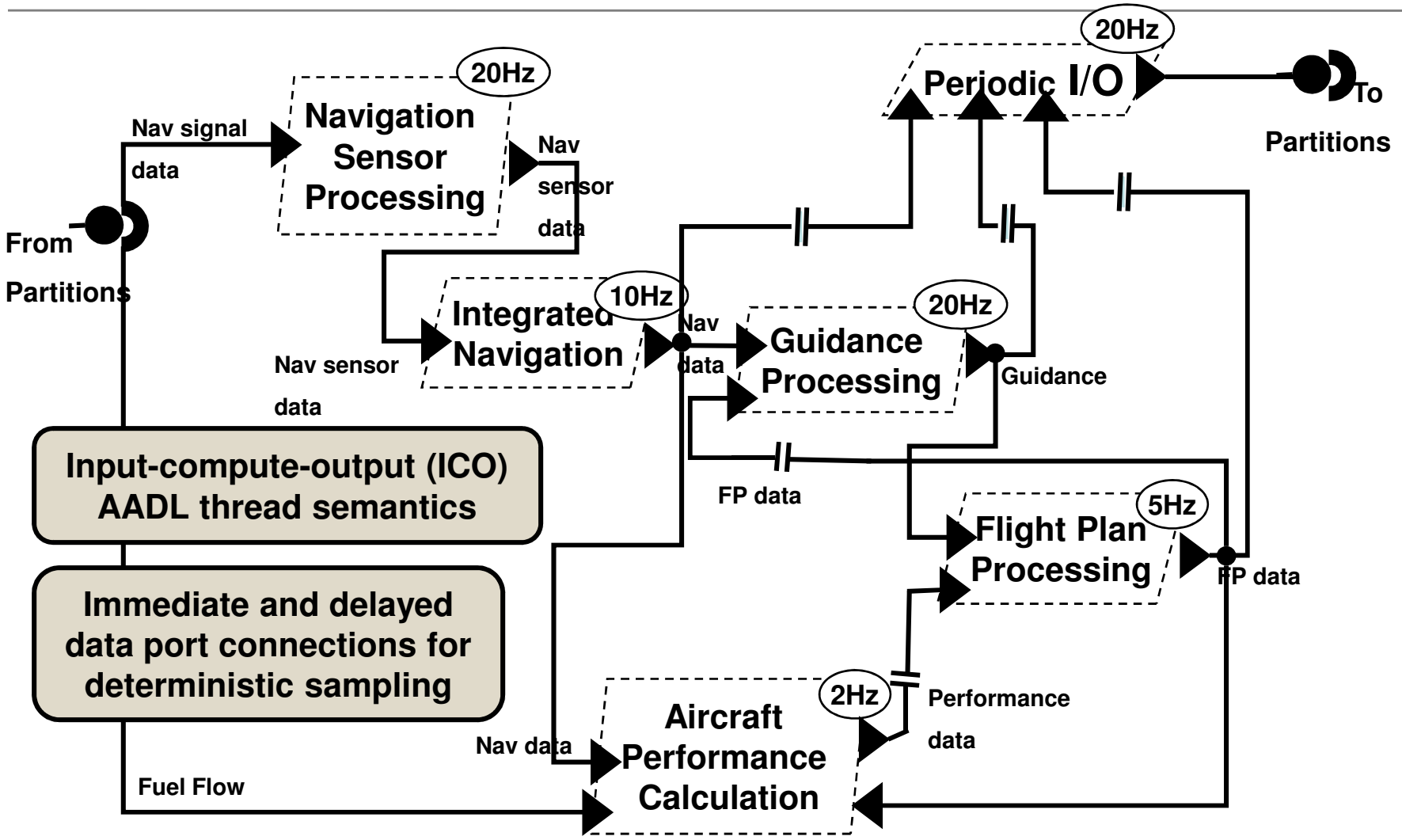
Frame-level Latency Jitter of Data Stream

Example: Non-deterministic downsampling

- Desired sampling pattern 2X: n, n+2, n+4 (2,2,2,…)
- Worst-case sampling pattern: n, n+1, n+4 (1,3,…)



Managed Latency Jitter through Deterministic Sampling



Rate Group Optimization

Logical threads to execute at a specific rate

Multiple logical threads to execute with the same rate

Placement of units with same rate in same operating system thread

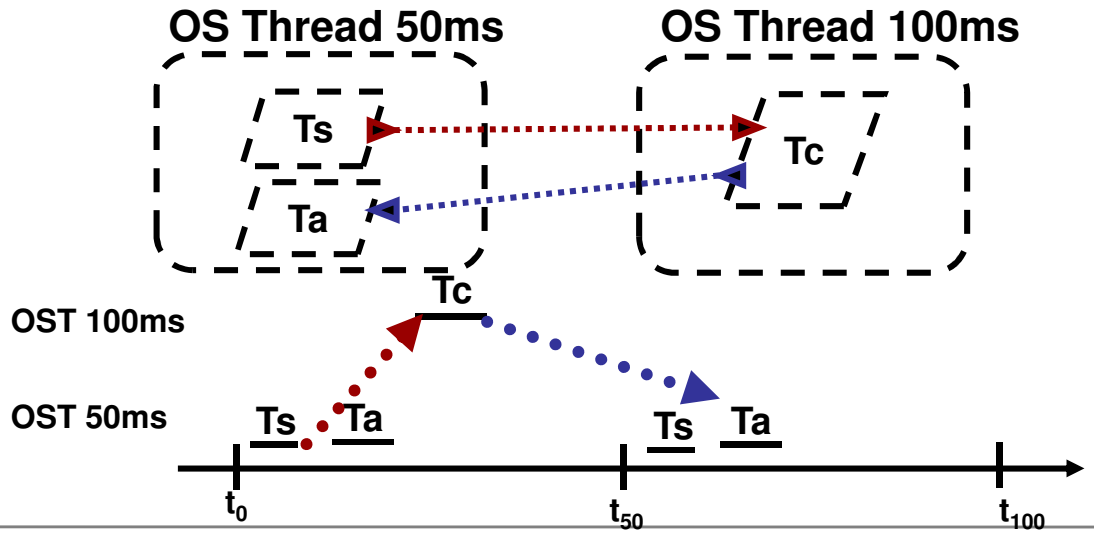
Reduced number of threads and context switches

Rate Group Order Can Affect Latency

Data flow from sensor T_s to control T_c to actuator T_a with mid-frame communication

Effect of rate groups: T_c to T_a becomes delayed

Occurs when pairwise immediate connections in opposite direction



Software-Based Latency Contributors

Execution time variation: algorithm, use of cache

Processor speed

Resource contention

Preemption

Legacy & shared variable communication

Rate group optimization

Protocol specific communication delay

Partitioned architecture

Migration of functionality

Fault tolerance strategy

Latency and Age of Data

Latency: the amount of time between a sensor reading and an output to an actuator based on the sensor reading

Age: amount of time that has passed since the sensor reading

Age Contributors

- Oversampling
- Missing sensor readings
- Failed processing
- Missed deadlines

Outline

Multiple aspects of system validation

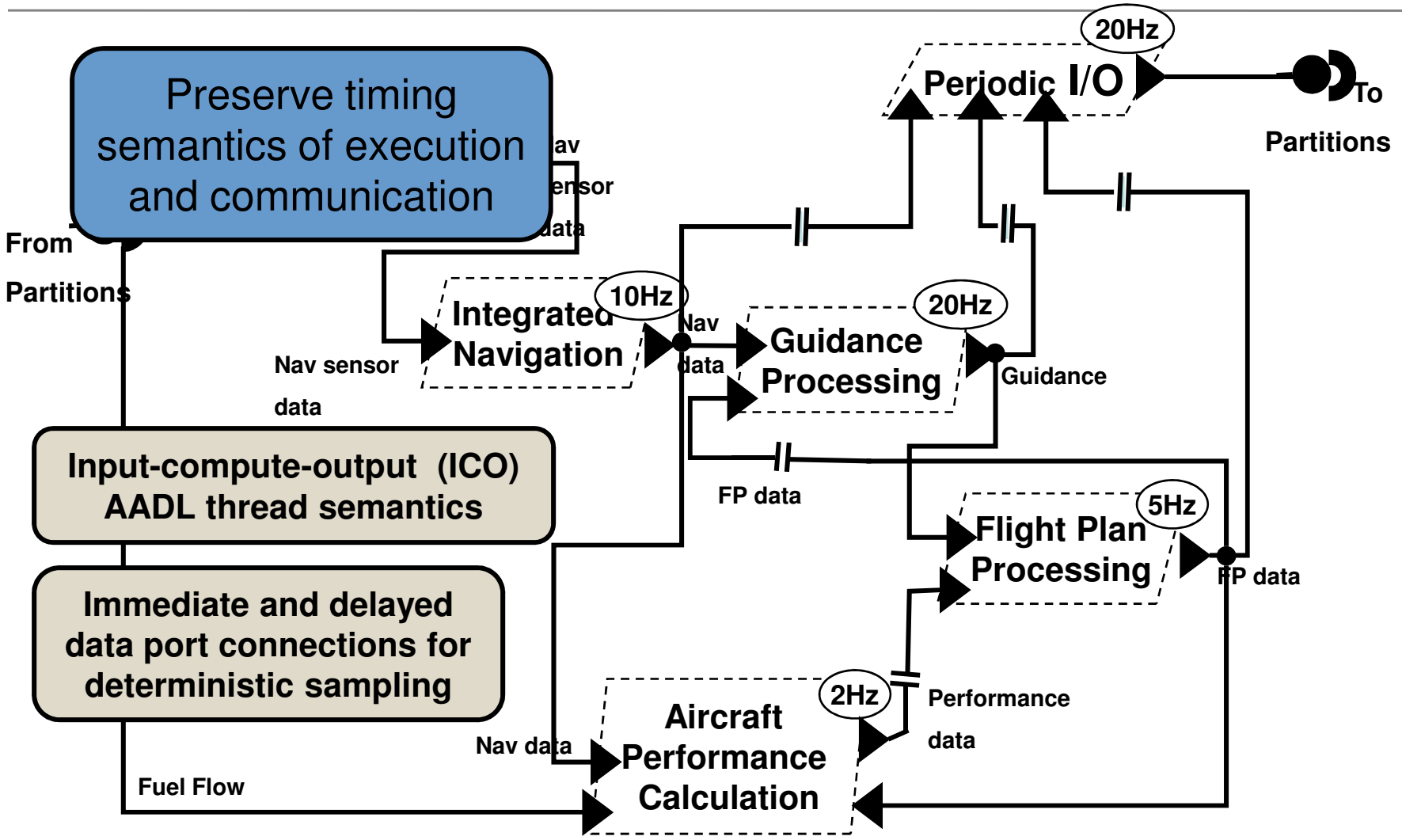
System & software engineers working together

Multi-fidelity model-based analysis

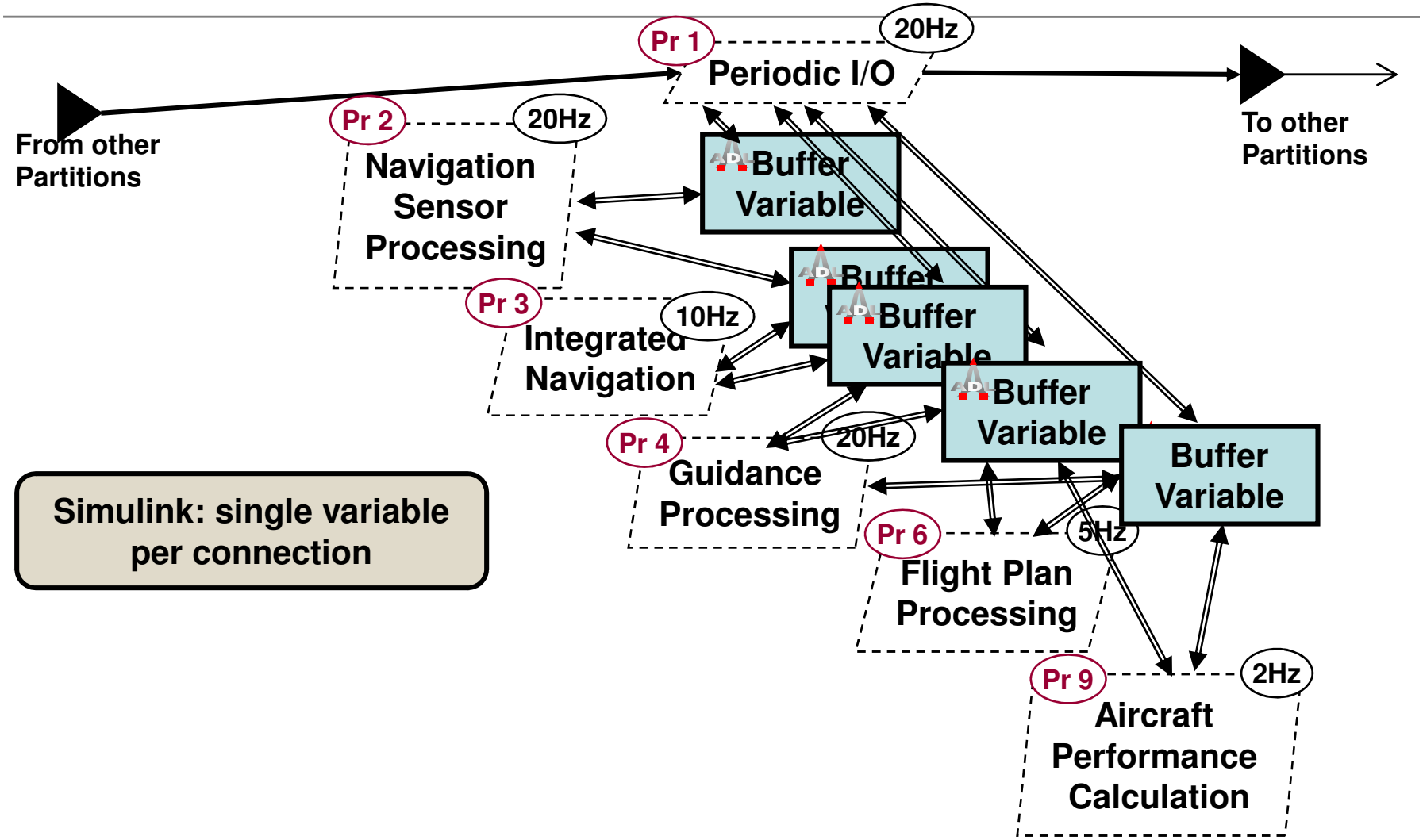
Property preserving transformations

Conclusions

Efficient Runtime System Generation



Will This Implementation Work?



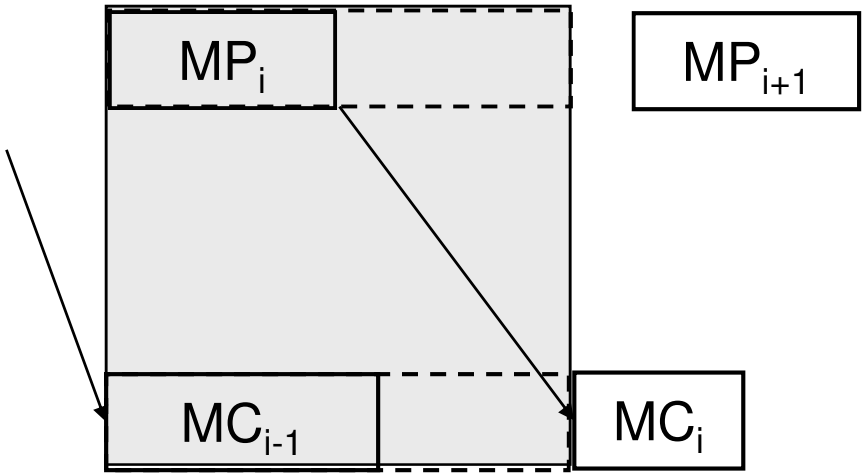
Overlapping Message Lifespan

Periodic thread MP and MC

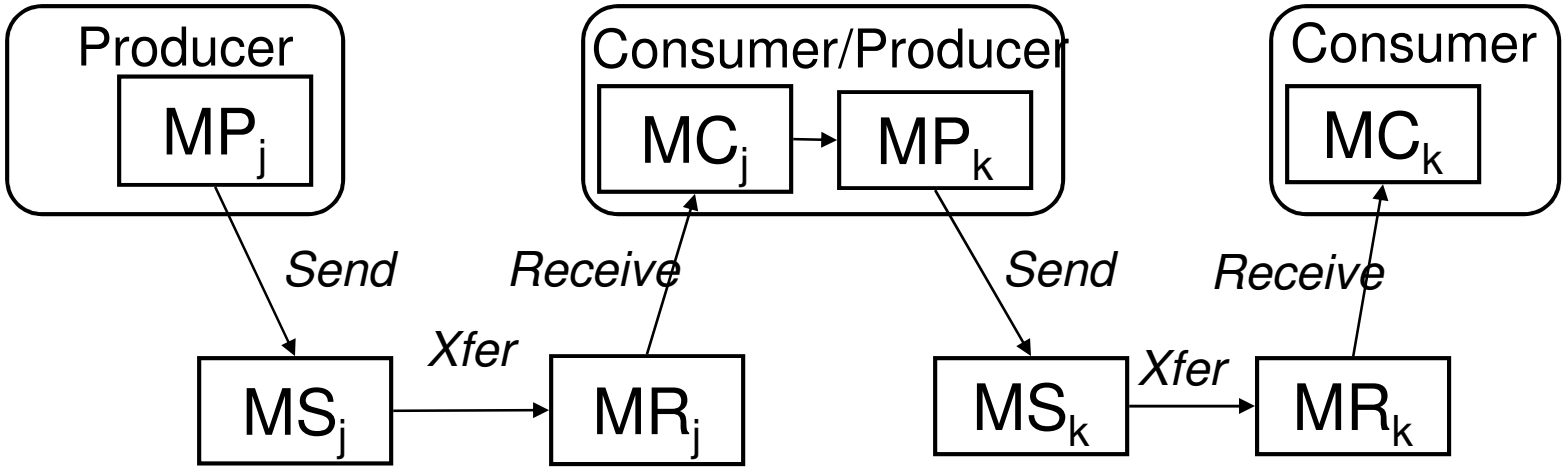
MP ->> MC

Need for double buffering

and timely transfer



Optimization of General Port Buffer Model



- Send/receive with or without copy
- Transfer with or without copy
- Processing with or without copy

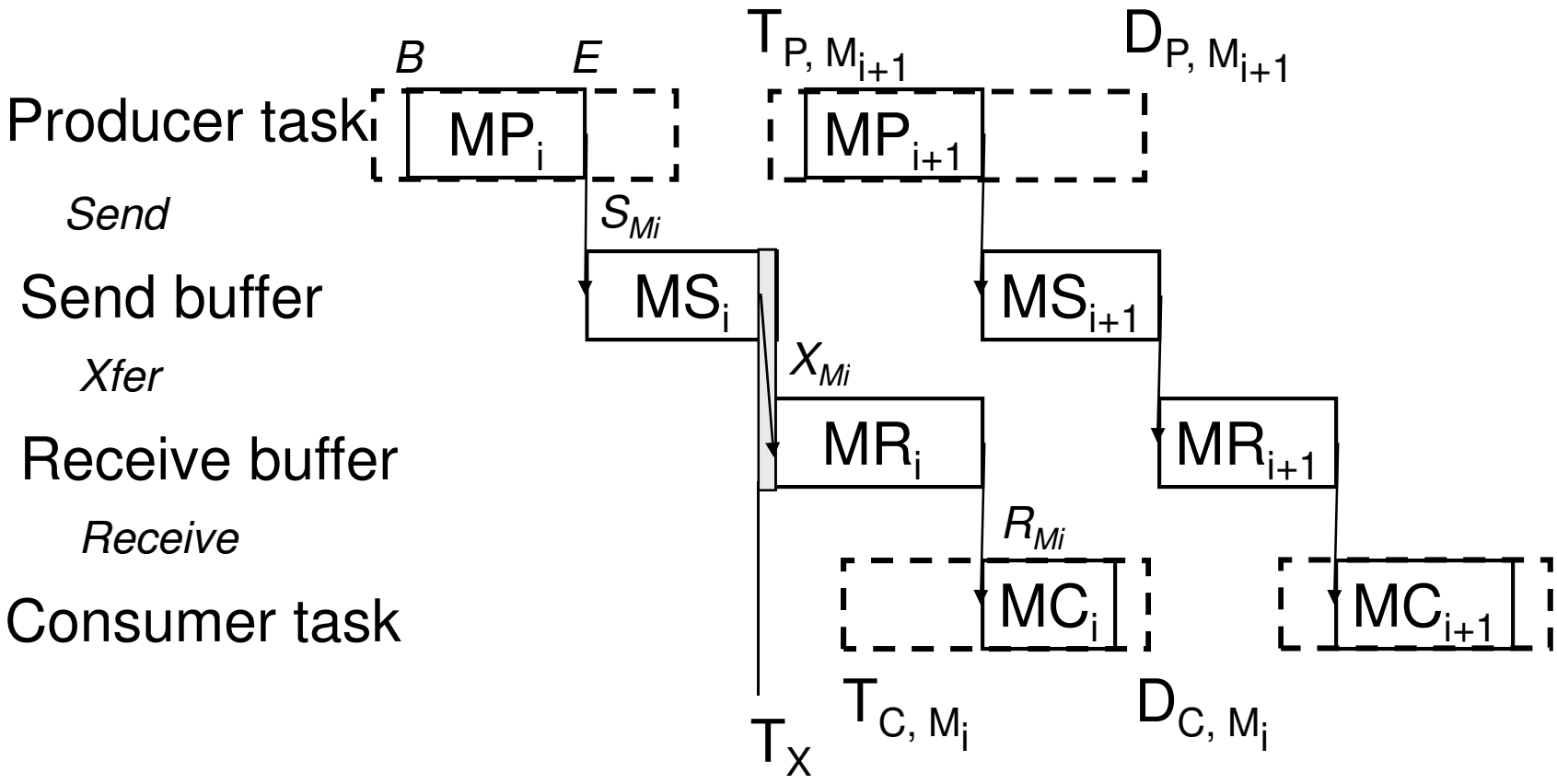
MP: producer copy

MS: send copy

MR: receive copy

MC: consumer copy

Message Streaming Lifespan Framework



Message Lifespan Properties

MC input-compute-output guarantee

$$T_{C,\,M_i} \le R_{M_i} = B_{MC_i} \le \mathbf{E_{MC_i}} \le T_{C,\,M_{i+1}} \le \mathbf{R_{m_{i+1}}}$$

Message operation ordering condition

$$S_{M_i} < X_{M_i} < R_{M_i}$$

MP bounded by producer dispatches

$$T_{P,\,M_i} \le B_{MP_i} \le E_{MP_i} = S_{M_i} \le T_{P,\,M_{i+1}}$$

MS bounded by sends and transfer

$$S_{M_i} = B_{MS_i} \le X^{*}_{M_i} \le E_{MS_i} < S_{M_{i+1}}$$

MR bounded by transfers and receive

$$X^{**}_{M_i} \le B_{MR_i} \le E_{MR_i} = R^{***}_{M_i} < X_{M_{i+1}}$$

* Completion of transfer

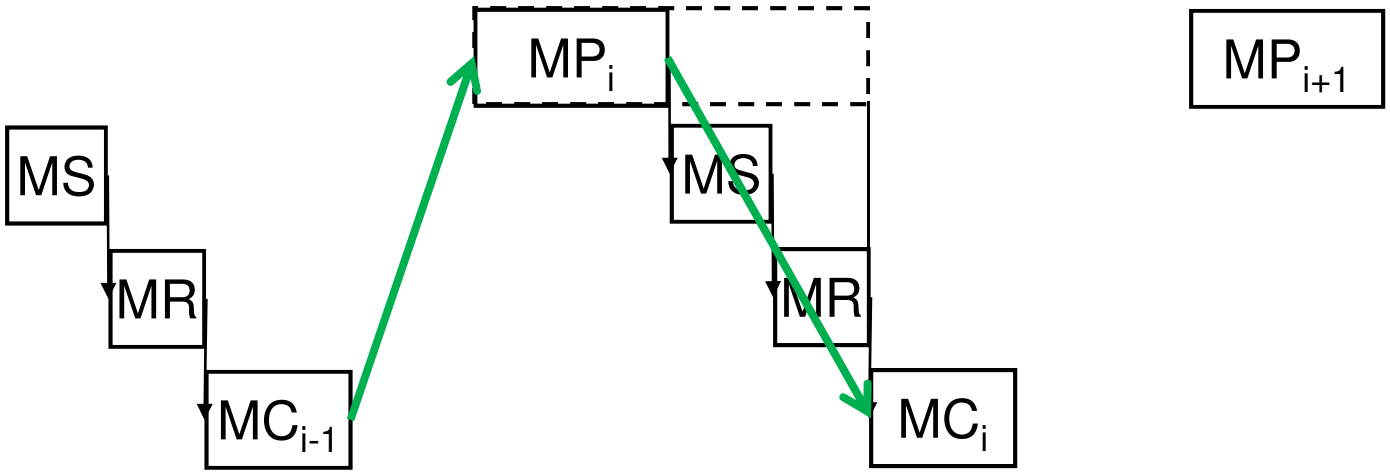
** Start of transfer

*** Latest of multiple receivers

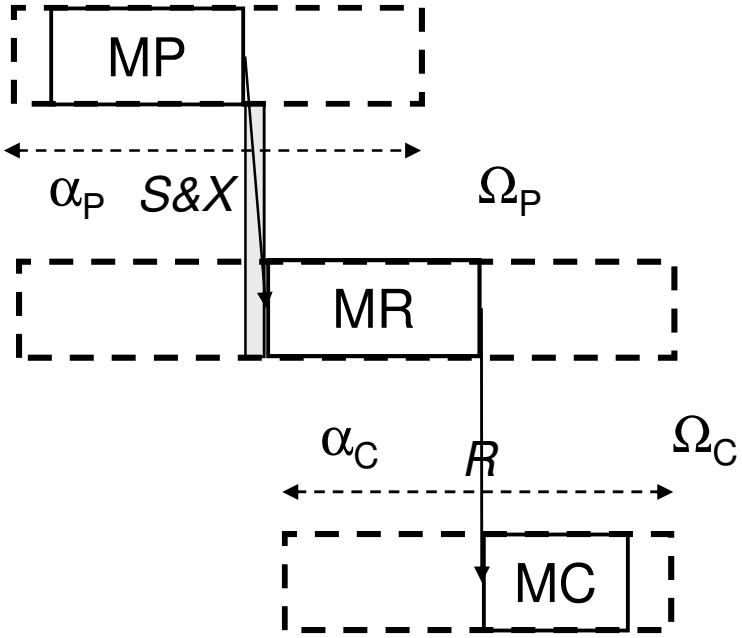
Sequential Execution of Periodic Tasks

$(\tau_P ; \tau_C)^*$

Collapse to single buffer



Application-based Send and Receive (ASR)



$(\tau_P \mid \tau_C)^*$

3 buffers for ICO

$T_P \le \alpha_P \le S \le \Omega_P \le D_P$

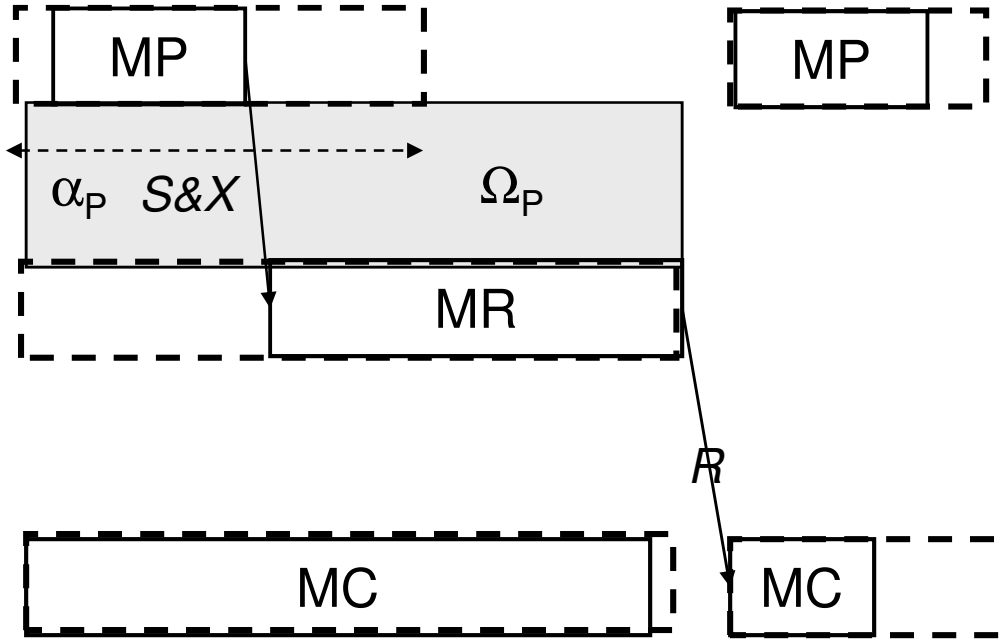
$T_C \le \alpha_C \le R \le \Omega_C \le D_C$

α : actual execution start time

Ω : actual completion time

$\alpha_P - \Omega_P \cap \alpha_C - \Omega_C \neq \varnothing \Rightarrow$ **non-deterministic S/R order**

Dispatch-based Send and Receive (DSR)



$(\tau_P \mid \tau_C)^*$

2 buffers for ICO

$T_P \le \alpha_P \le S \le \Omega_P \le D_P$

$D_P \le R \le T_C$

α : actual execution start time

Ω : actual completion time

$\alpha_P - \Omega_P \cap D_P - T_C = \varnothing \Rightarrow$ deterministic S/R

Buffer Optimization Considerations

Periodic & aperiodic task dispatch

Send and receive execution

- As part of application (ASR)
- As part of task dispatch/completion (DSR)

Task execution order

- Concurrent: $\tau_C \mid \tau_P$
- Atomic non-deterministic: $\tau_C \neq \tau_P$
- Ordered: $\tau_C ; \tau_P$ or $\tau_P ; \tau_C$

Message transfer

- Immediate to consumer (IMT)
- Direct to delayed consumer (DMT)
- Period-delayed to consumer (PMT)

Periodic Task Communication Summary

Periodic Same period	ASR IMT | PMT		DSR IMT | PMT		DMT
τ_P ; τ_C	MF:1B	PD:2B S∨X∨R	PD:2B R	PD:2B S∨X/R	MF:1B
τ_C ; τ_P	PD:1B	PD:1B	PD:1B	PD:1B	PD:1B
$\tau_P \neq \tau_C$	ND:1B	PD:2B X	PD:2B R	PD:2B X/R	ND:1B
τ_P | τ_C	ND:3B S/X_C R_C	PD:2B X	PD:2B R	PD:2B X/R	NDI:2B $S/X/R_C$

MF: Mid-Frame
PD: Period Delay
ND: Non-Deterministic
NDI: No Data Integrity

1B: Single buffer
2B: Two buffers
3B: Three buffers
4B: Four buffers

S, X, R : data copy
S/X : IMT combined send/xfer
S/X/R : DMT combined S, X, R
X/R: DSR/PMT combined X, R
o1∨o2 : One operation copy

Outline

Multiple aspects of system validation

System & software engineers working together

Multi-fidelity model-based analysis

Property preserving transformations

Conclusions

Predictable Model-based Engineering

Reduce the risks

- Analyze system early and throughout life cycle
- Understand system wide impact
- Validate assumptions across system

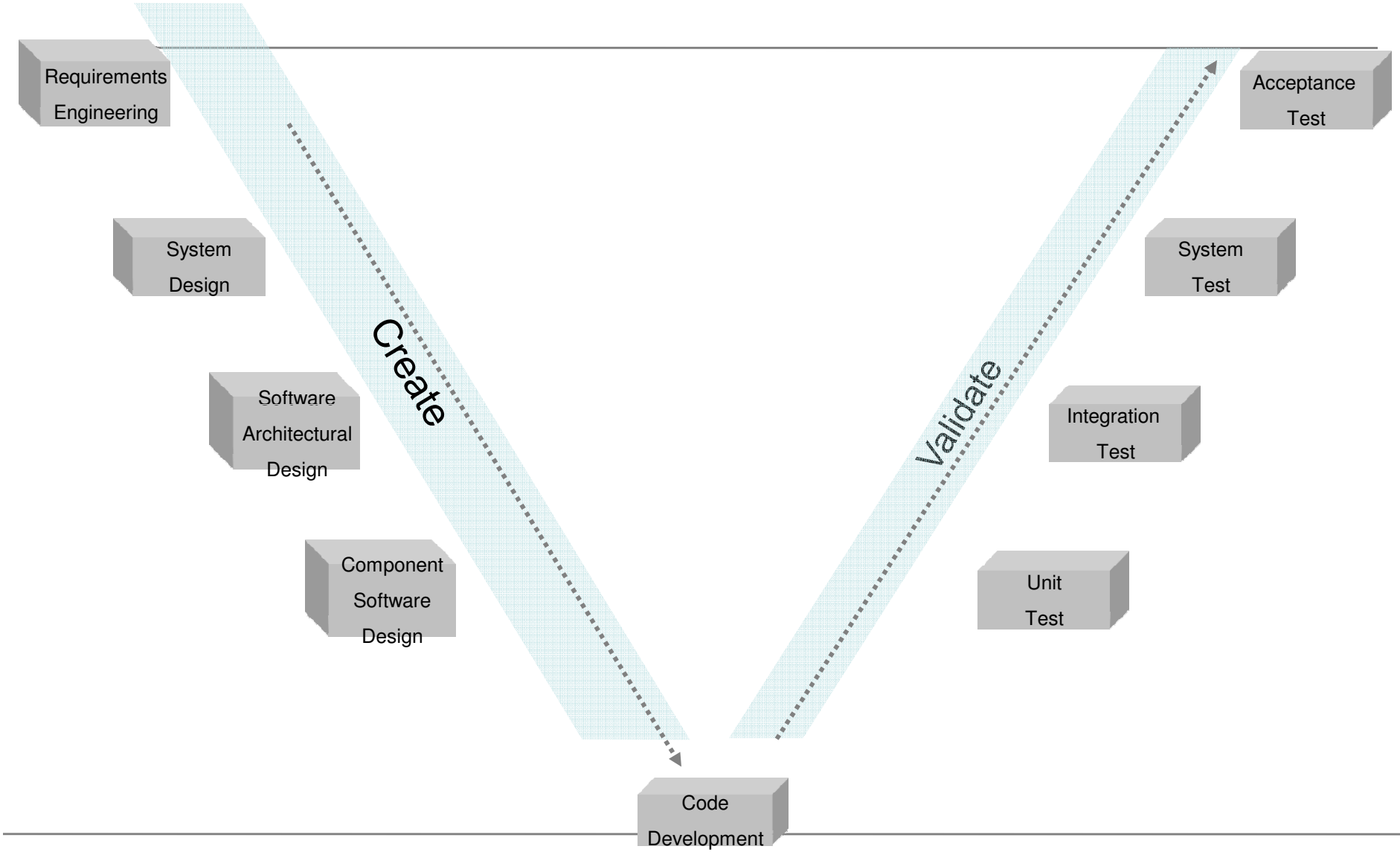
Increase the confidence

- Validate models to complement integration testing
- Validate model assumptions in operational system
- Evolve system models in increasing fidelity

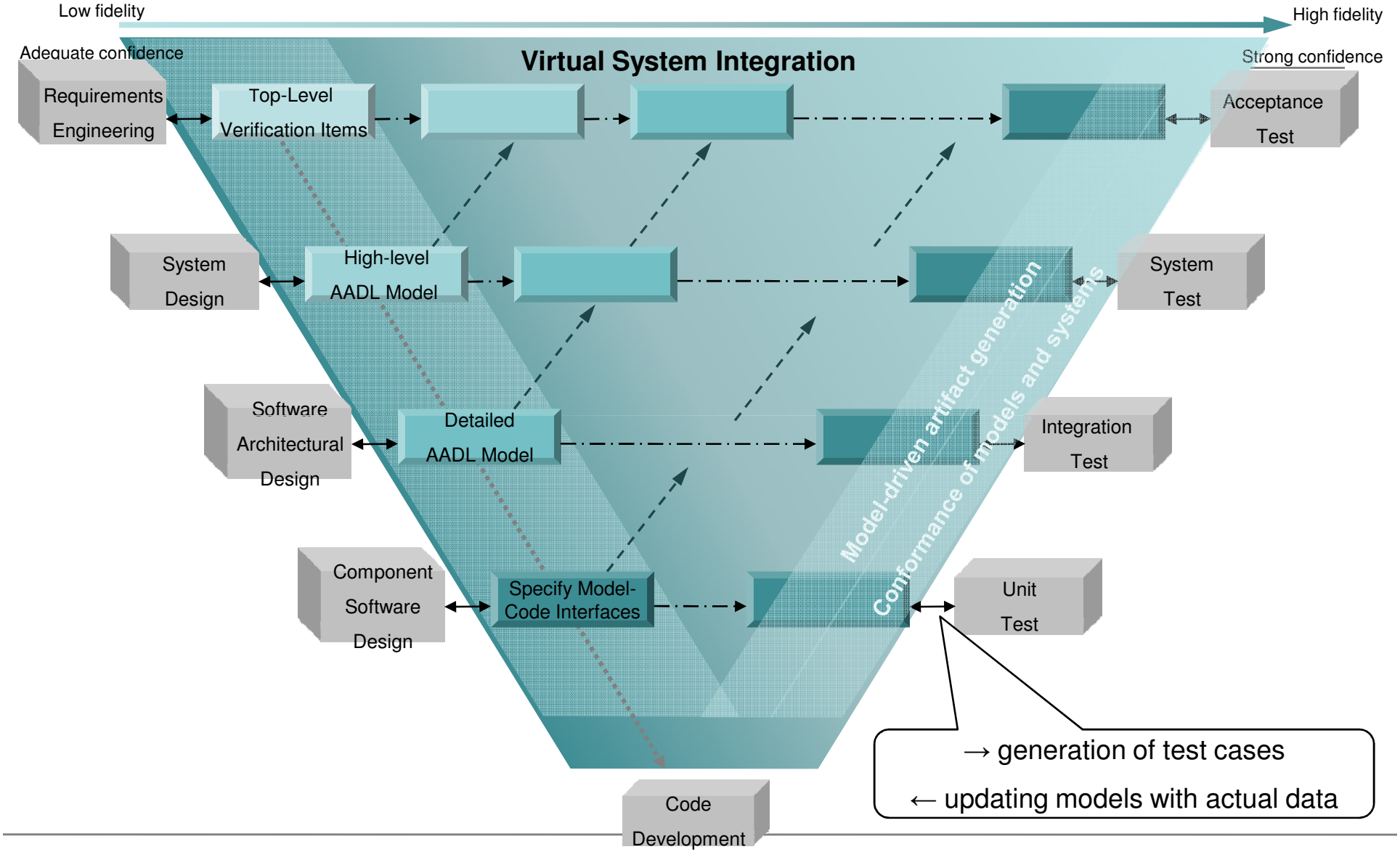
Reduce the cost

- Fewer system integration problems
- Fewer validation steps through use of validated generators

Traditional Development Model



Benefits of Predictive Architecting



Software Engineering Institute | Carnegie Mellon

Peter H Feiler

phf@sei.cmu.edu

at ENST until June 9, 2008