



code design environment

Schedule

- **1. Introduction**
- **2. System Modeling Language: System C ***
- **3. HW/SW Cosimulation ***
- **4. C-based Design ***
- **5. Data-flow Model and SW Synthesis**
- **6. HW and Interface Synthesis**
(Midterm)
- **7. Models of Computation**
- **8. Model based Design of Embedded SW**
- **9. Design Space Exploration**
(Final Exam)
(Term Project)

■ Model of Computation

- [21] E. A. Lee and A. Sangiovanni-Vincentelli, "A Framework for Comparing Models of Computation," IEEE TCAD, 17(12), December, 1998.

■ STATEMATE

- [22] D. Harel, "Statecharts: A Visual Formalism for Complex Systems," Science of Computer Programming, Vol. 8, pp. 231-274, 1987
- [23] David Harel, et. al., "STATEMATE: A Working Environment for the Development of Complex Reactive Systems," IEEE TSE, 16(4), April 1990.

■ POLIS & ACFSM

- [24] Marco Sgroi, Luciano Lavagno, A. Sangiovanni-Vincentelli, "Formal Models for Embedded System Design," IEEE Design & Test of Computers, 17(2), 14-27, June 2000.

■ COSY

- [25] H.J.H.N.Kenter et. al, "Designing Digital Video Systems: Modeling and Scheduling," CODES Workshop, May 1999.

■ Simulink

- [26] Paul Caspi, et. al, "Translating Discrete-Time Simulink to Lustre," LNCS, Vol. 2855/2003, pp. 84-99, 2003.

■ Ptolemy

- [27] J. T. Buck, et. al., "Ptolemy: A Framework for Simulating and Prototyping Heterogeneous Systems," Int. Journal of Computer Simulation, special issue on Simulation Software Development, vol. 4, pp. 155-182, April, 1994.

■ Ptolemy II: *-chart

- [28] A. Girault, B. Lee, and E. A. Lee, "Hierarchical Finite State Machines with Multiple Concurrency Models," IEEE Transactions On Computer-aided Design Of Integrated Circuits And Systems, Vol. 18, No. 6, June 1999.

■ Process Networks

- [29] E. A. Lee and T. M. Parks, "Dataflow Process Networks," Proceedings of the IEEE, vol. 83, no. 5, pp. 773-801, May, 1995

■ PeaCE

- [30] Dohyung Kim, Minyoung Kim and Soonhoi Ha, "A Case Study of System Level Specification and Software Synthesis of Multi-mode Multimedia Terminal", Workshop on Embedded Systems for Real-Time Multimedia (ESTIMedia), Newport Beach, CA, USA Oct 2003

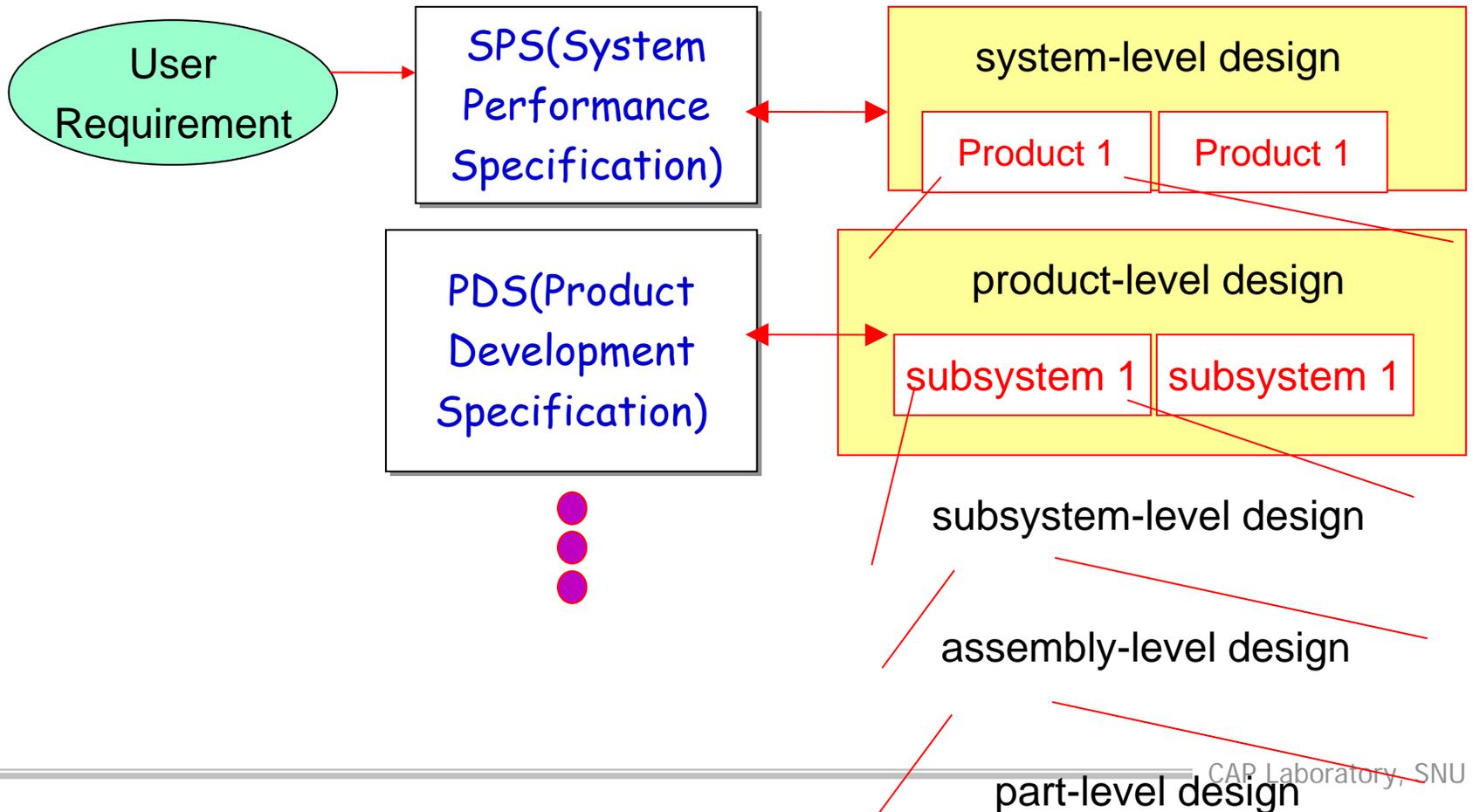
Outline

- **Introduction**
- **State-oriented model**
- **Activity-oriented model**
- **Process network model**
- **Heterogeneous approach**
- **System-level specification experiment**

System Design Flow

■ System Design Flow (System Engineering view)

- Top-down approach



System Specification

■ Definition

- Specification is a document that describes the essential **requirements** for items, materials, processes, or services of a prescribed solution space, data required to **implement** the requirements, and **methods of verification** to satisfy specific criteria for formal acceptance

■ Specification Tree

- Define System Performance Specification based on user requirement.
- Specification tree
 - *Product 1 development specification*
 - *Product 2 development specification*
 - Subsystem B1 Development specification
 - SW requirement specification
 - HW requirement specification

Understanding Specification Requirements

■ Specification requirements

- Operational requirements: basic system function
- Capability requirements: system performance
- Nonfunctional requirements: operating condition, outlook
- Interface requirement
- Verification requirement

■ Modes of operation

- Normal mode
- Abnormal mode
- Emergency mode
- Catastrophic mode

Specification Development Approaches

■ Feature-based approach

- Popular method for non-expert: ad hoc list of requirements
 - *Energy consumption, real-time performance, cost, functions, etc.*
- Poorly organized, prone to missing, misplaced, conflicting, and duplicate requirements

■ Reuse-based approach

- Reuse and extend the specification of prebuilt component

■ Performance-based approach

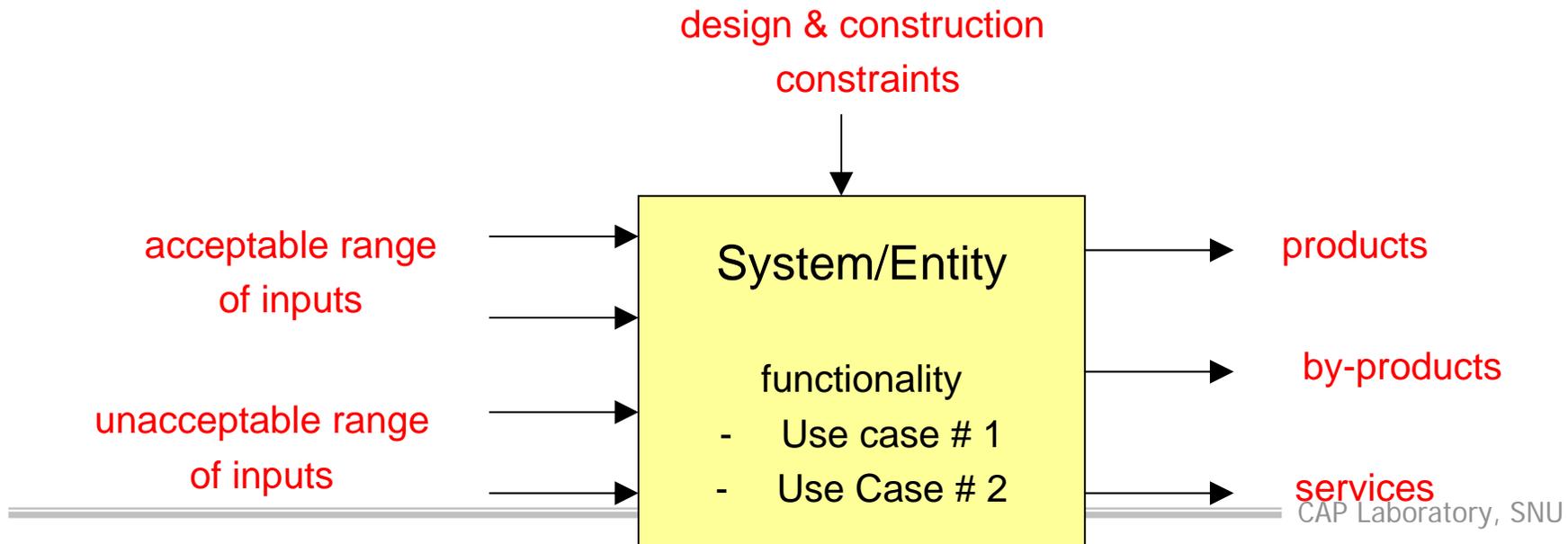
- Popular method to specify the requirement systematically

■ Model-based approach

- Build specification tree based on top-down approach

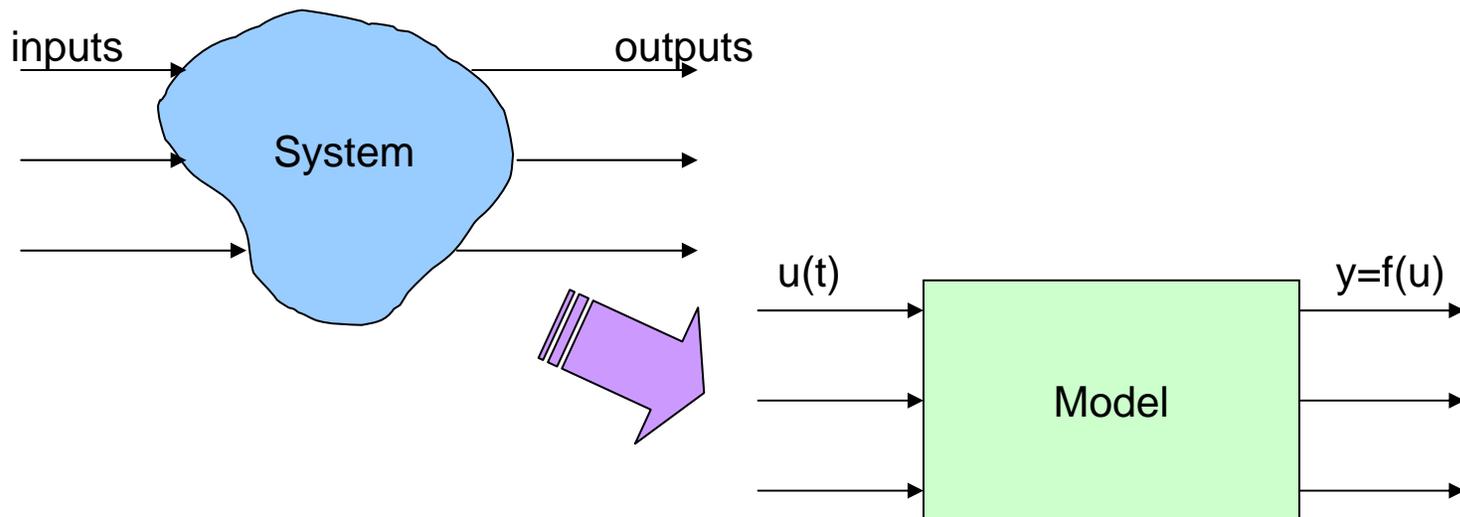
Performance Based Approach

- **Regard a system as a black box and specify the system function with performance boundary conditions.**
- **Systematic specification**
 - System modes, missions, use cases
 - Design and construction constraints
 - Range of Inputs and outputs



■ Model

- Abstract view of real entity
- Express the essential features as minimally as possible
- Better if executable, synthesizable, verifiable
- Separation of communication and computation
- Hierarchy for complexity management

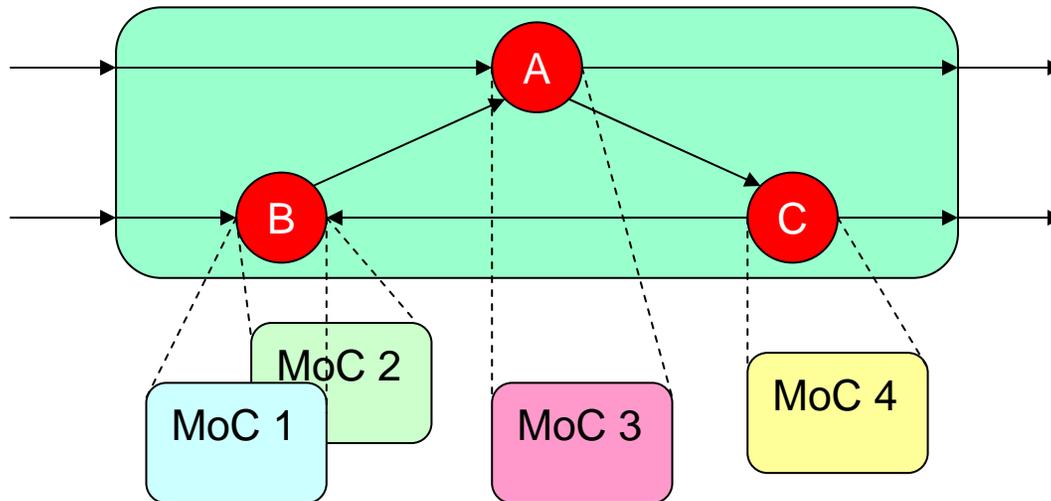


- **State-oriented models**
 - Finite-State Machines (FSM), statechart, Esterel
- **Object-oriented model**
 - UML
- **Activity-oriented models**
 - Dataflow model, Discrete-Event model
 - SystemC model
- **Concurrent process models**
 - CSP, Kahn process networks, Dataflow process network
- **Heterogeneous models**
 - Ptolemy, Metropolis, PeaCE
- **Others**
 - SIMULINK, Synchronous-Reactive model

Heterogeneous Models

■ Motivation

- Application itself has heterogeneous nature: control and computation
- Different modeling methods have been developed for different class of application



* MoC : Model of Computation

Model Classification

- **Criteria: Abstraction Level of Time**

- **Untimed Model of Computation**
 - Data does not carry timing information
 - Only ordering of signal is expressed
 - (example) SDF, Process networks (Kahn, Dataflow)

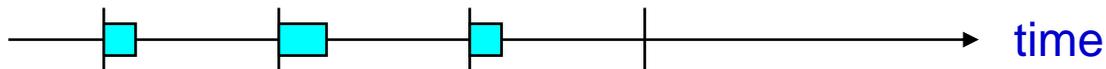
- **Synchronous Model of Computation**
 - Synchronous/reactive model
 - Statechart

- **Timed Model of Computation**
 - Discrete-event model
 - Continuous time model

Synchronous Model

■ Meaning of “Synchronous”

- Time axis is divided into timing slots
- Events in each slot are considered as completed instantaneously.
- A system only needs to be “fast enough” to simulate synchronous behavior



■ Classification

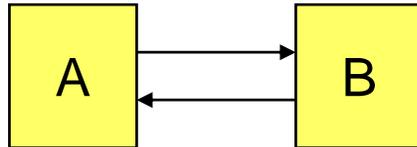
- Perfectly synchronous MoC
 - *Zero processing time: outputs are produced immediately after inputs are consumed.*
 - *Events are delivered to the destination instantaneously*
 - *(ex) SR model*
- Clocked synchronous MoC
 - *All processes have delay to produce outputs*
 - *Delay is equal to the evaluation cycle*
 - *(ex) FSM model*

■ Example

- Control-oriented languages: StateCharts, Esterel
- Data-flow-oriented languages: Signal, Lustre

Synchronous Reactive Model

■ Fixed-point semantics



fixed point \Leftrightarrow stable state
determinism \Leftrightarrow unique solution

■ Cycle-driven simulation

■ Characteristics

- Good match for control-intensive systems, clocked synchronous circuits
- Tightly synchronized
- Determinate in most cases
- Maps well to hardware and software

■ Limitation

- inefficient for irregular multirate systems
- Computation-intensive systems are overspecified
- Causality loops are hard to detect
- SW implementation is not easy

Timed Model of Computation

■ Timed Model of Computation

- Timing information is conveyed on the signals (in synchronous model, transmitting absent events at regular time intervals.)
- Events are processed in the chronological order

■ Difference from Synchronous MoC

- Finer timing granularity
 - *Minimum timing resolution can be defined arbitrarily (ns or ps)*
- Processes should preserve causality constraints

■ Example

- Discrete event model
- Continuous time model: (ex) SIMULINK

- **“Design is to represent”**
 - Design procedure depends on the initial specification model

- **Importance of model**
 - A design environment has its input model
 - Input model determines the limitation of the design environment as well as its capability

- **Status**
 - Diverse models of computation are used in reality.
 - No design environment exists supporting all design procedure from specification to implementation.

- **Introduction**
- **State-oriented model**
 - Finite State Machine
 - Statechart and fFSM
 - Esterel
 - STATEMATE: statechart-based design environment
- **Activity-oriented model**
- **Process network model**
- **Heterogeneous approach**
- **System-level specification experiment**

State-oriented Model

- **Main application: reactive systems**

- **Intricate control activities**

- errors in the control sequence can have fatal consequences
- intuitive and clean specification for designers
- precise and rigorous specification for analysis

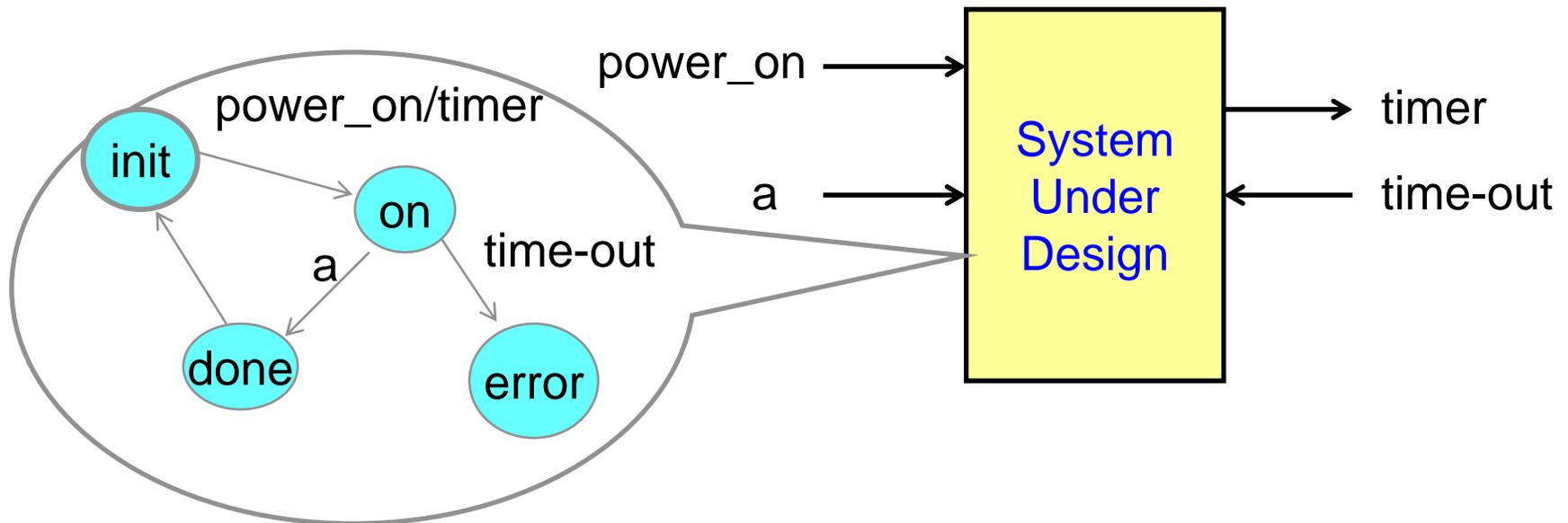
 **FSM**

- **Concurrency**

- autonomous agents interact each other
- pure FSM model suffers from state explosion problem

 **Statechart, CFSM, fFSM**

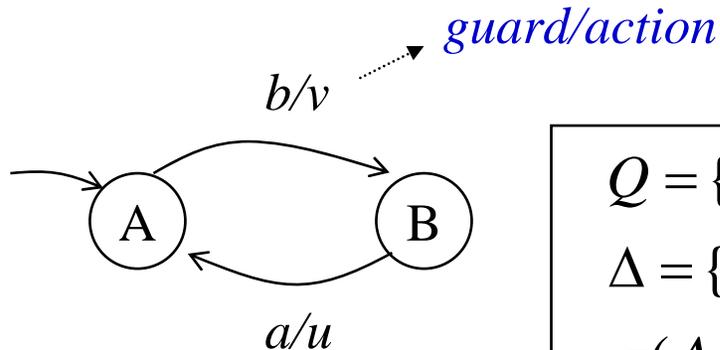
- **Well-known semantics**
- **Rich analytical properties**
 - Reachability analysis, liveness



Finite State Machines

■ **Five-tuple** $(Q, \Sigma, \Delta, \sigma, q_0)$

- Q : finite set of symbols denoting states
- Σ : set of symbols denoting the possible inputs $\Sigma = \Sigma_1 \times \Sigma_2 \times \dots$
- Δ : set of symbols denoting the possible outputs
- σ : transition function mapping $Q \times \Sigma \rightarrow Q \times \Delta$
- $q_0 \in Q$ is the initial state



$$Q = \{A, B\}, \Sigma = \{a, b\}$$

$$\Delta = \{\varepsilon, u, v\}$$

$$\sigma(A, b) = (B, v), \sigma(A, a) = ?$$

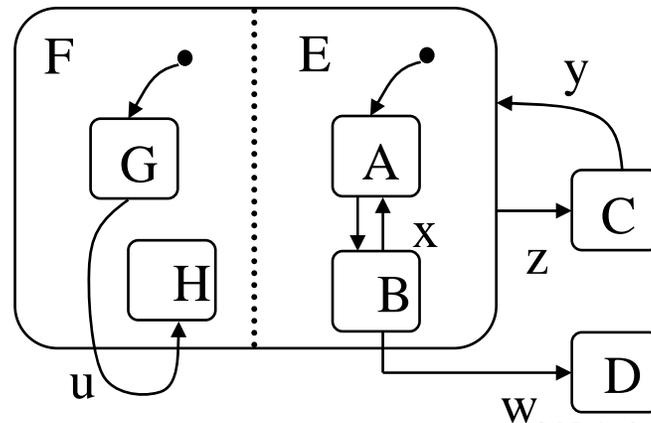
Statechart

■ Proble of (Pure) FSM Model

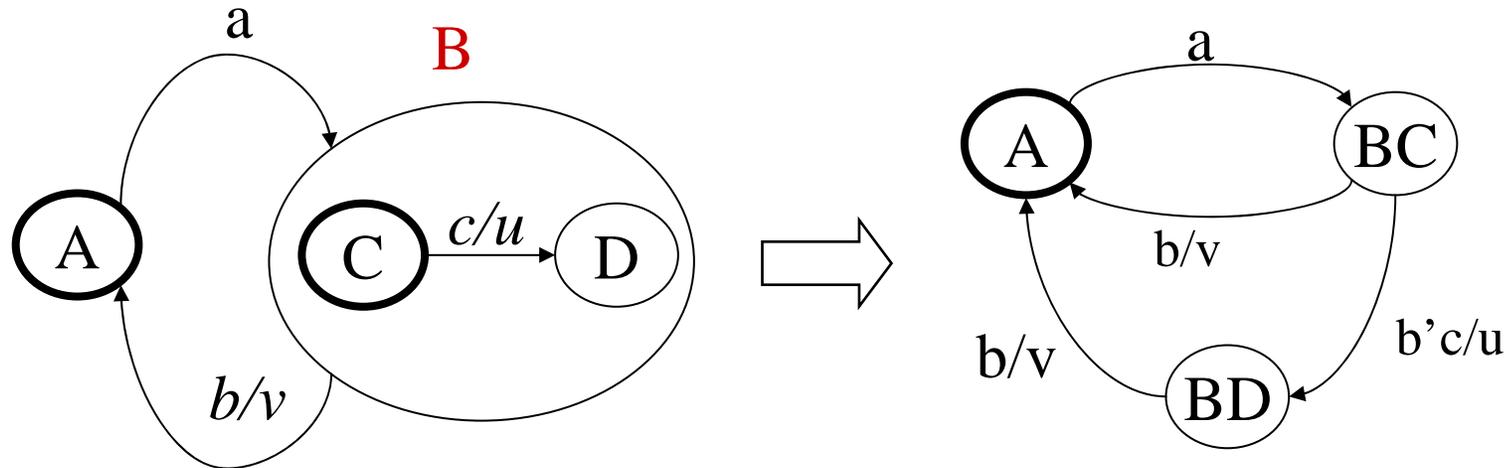
- Flat and unstructured: not easy to understand
- Inherently sequential: can not express concurrency
 - *very large number of states and transitions*

■ Harel's statechart

- D. Harel, "Statecharts: A Visual Formalism for Complex Systems," Science of Computer Programming, Vol. 8, pp. 231-274, 1987
- **hierarchy**
(OR decomposition)
 - Transitions are **NOT** level restricted
- **Concurrency**
(AND decomposition)

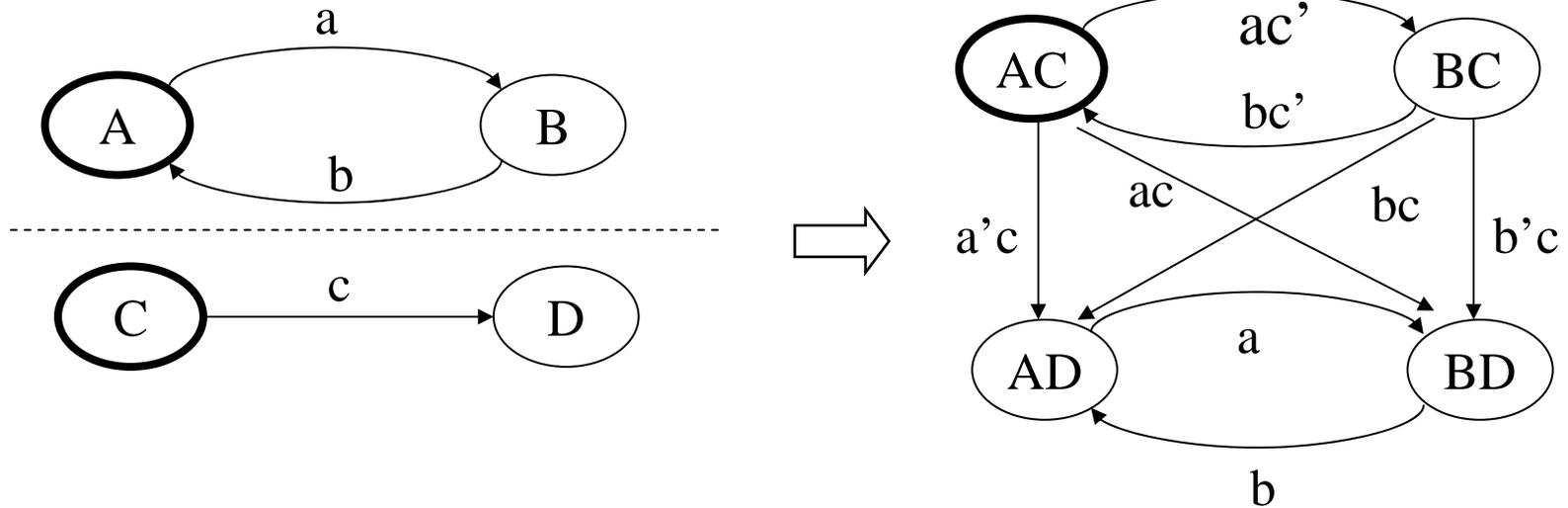


FSM Hierarchy



- **Structural description: does not reduce the number of states**
- **Does reduce the number of transitions**
- **What happens when input c and b both exist in state C ?**
 - (one solution) external FSM take actions first.

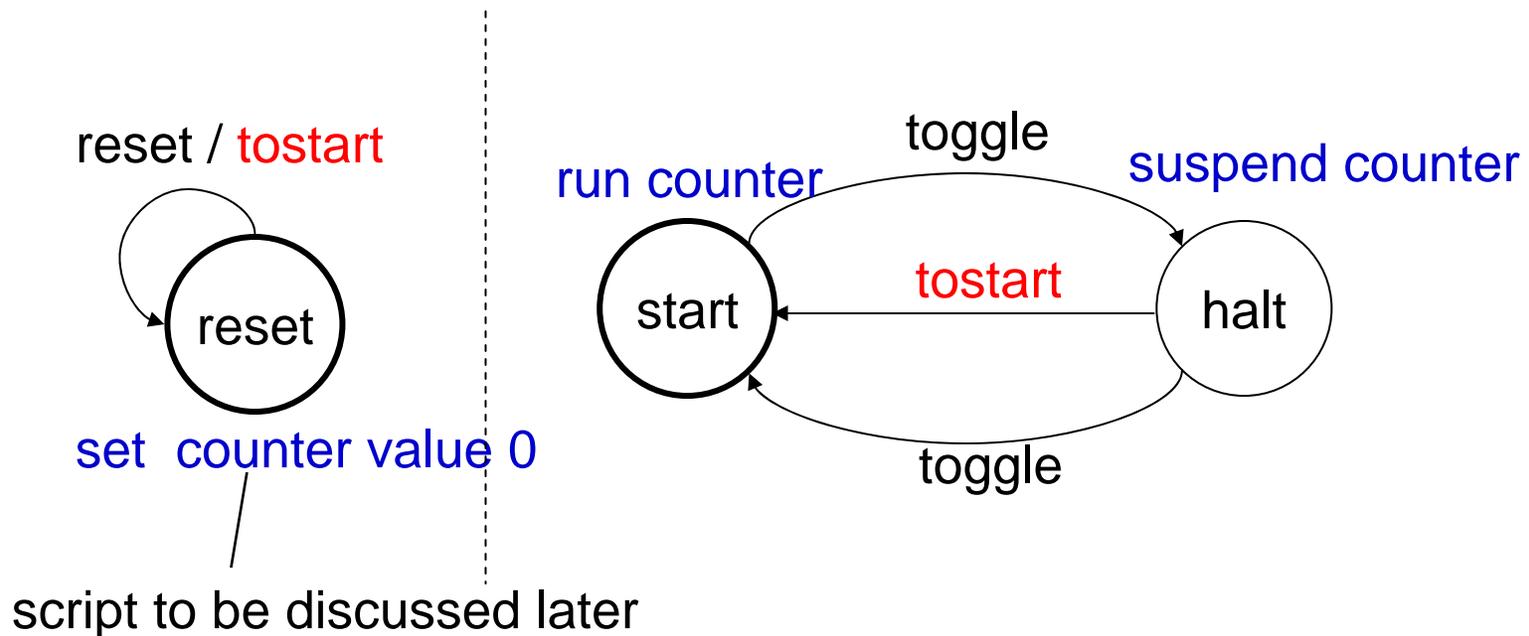
AND Decomposition



- Concurrent execution of multiple FSMs
- Solve state explosion problem
- How to support inter-FSM communication?
 - Internal events

Internal Event

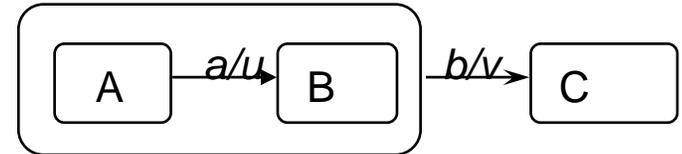
- Support communications between concurrent states and across hierarchy



Comments on statechart

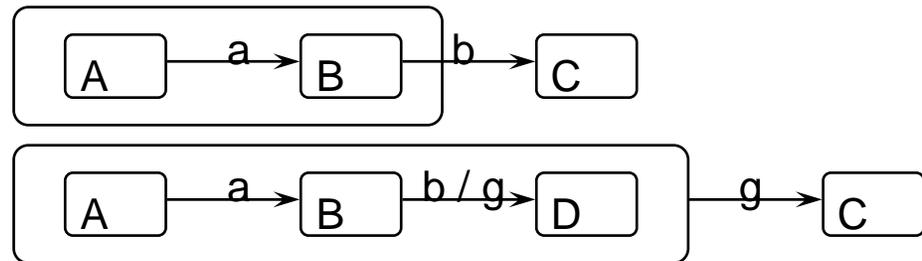
- **Incomplete and imprecise semantics**

- (ex) Transition priority is not well-defined



- **Rich syntax: transition is not level-restricted**

- Not compositional
- Not easy to analyze

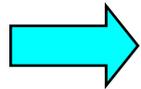


- **Perfect synchrony hypothesis**

- **More than 20 variants exist.**

Problems with Statecharts

- **Incomplete and imprecise semantics**
- **Rich syntax**
- **Synchronous approach**



More than 20 Statechart variants

- **Not compositional**
 - inter-level transitions
 - state references

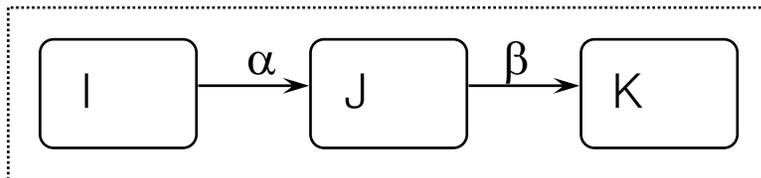
Variants of Statecharts

■ Plan 1 : Prohibit instantaneous transition

- A state must not be simultaneously entered and exited
- transitions indifferent parallel components may be simultaneously executed.
- Help solving many non-deterministic problem
- But, there is difficult to achieve compositionality

■ Plan 2 : Allow instantaneous transition with some limitation

- Not to make non-determinism and infinite sequence of transition
- Not permit to enter the same state twice
- distinguishing internal from external events

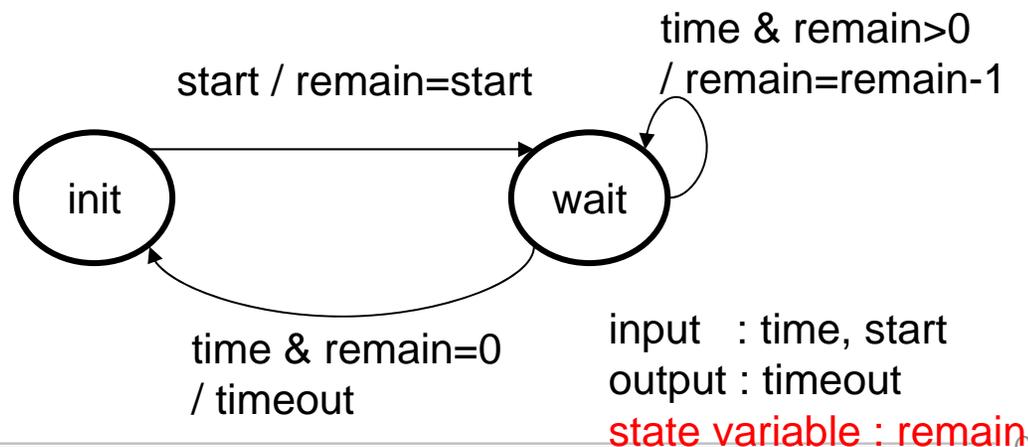


What happens when α , β exist simultaneously?

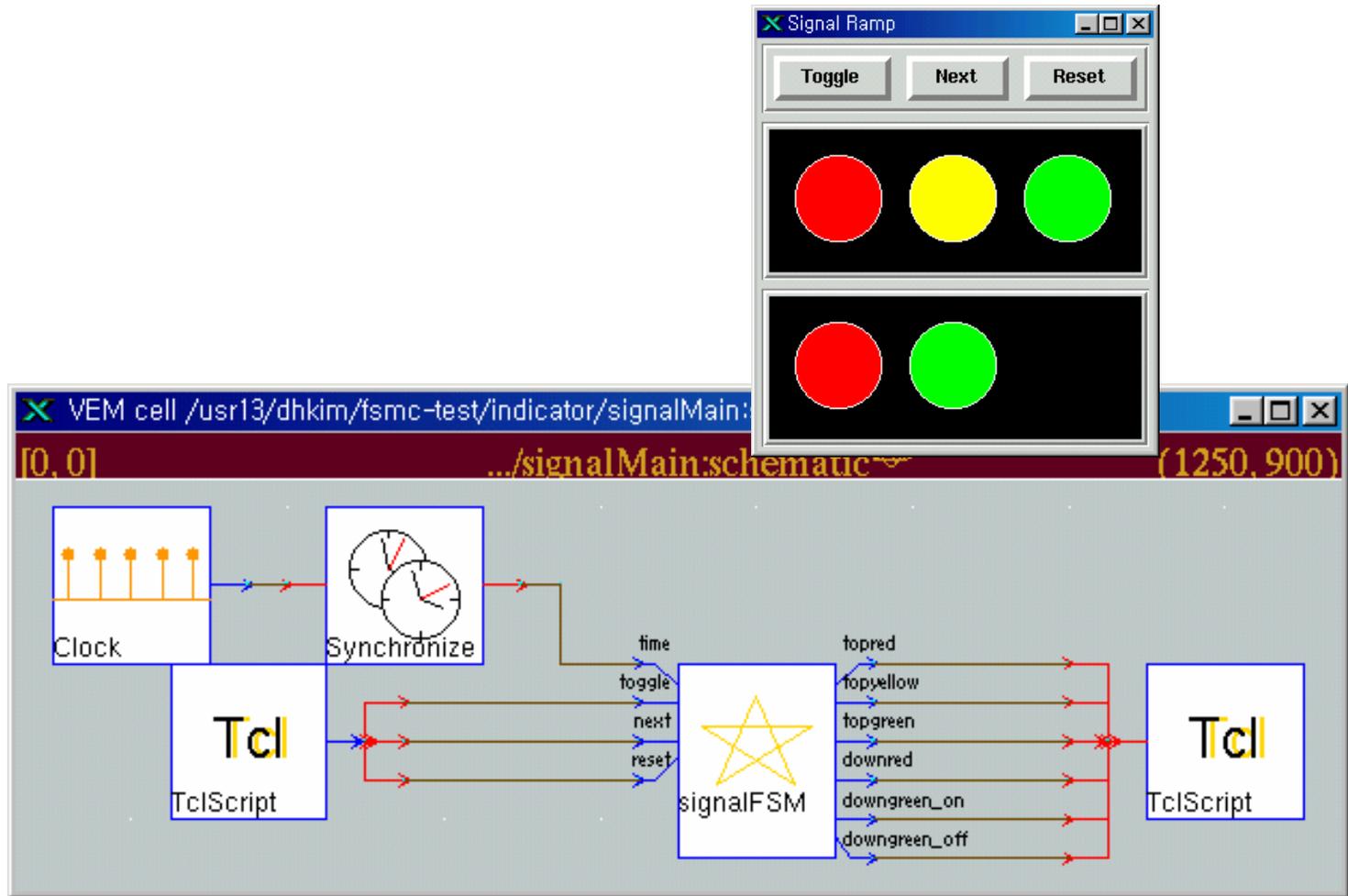
flexible FSM (f FSM)

- **New FSM extension used in PeaCE**
 - A Statechart derivative
 - Used in a heterogeneous modeling environment

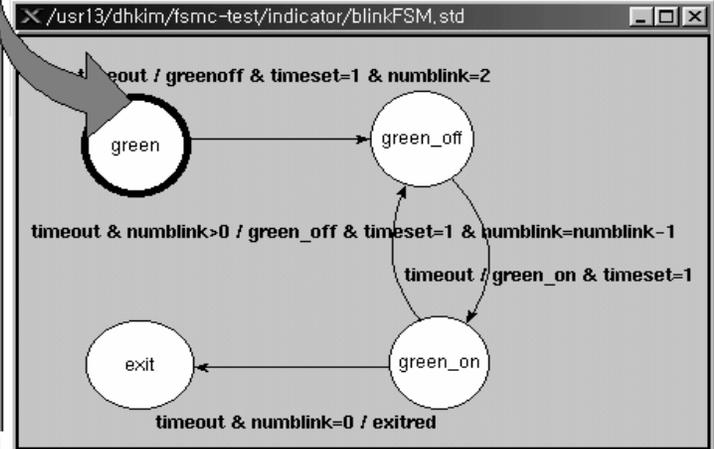
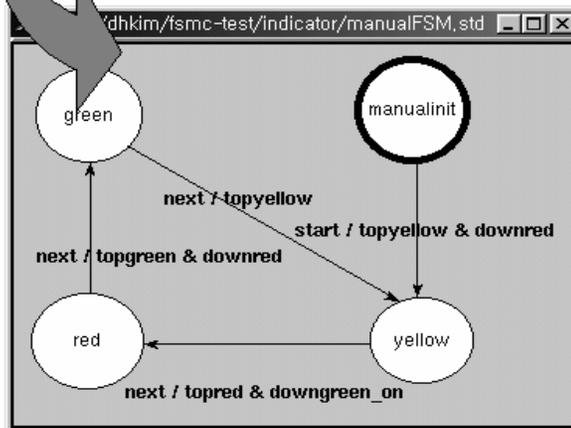
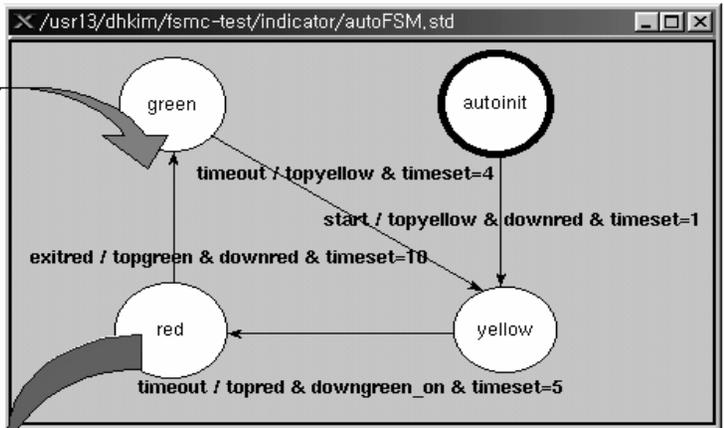
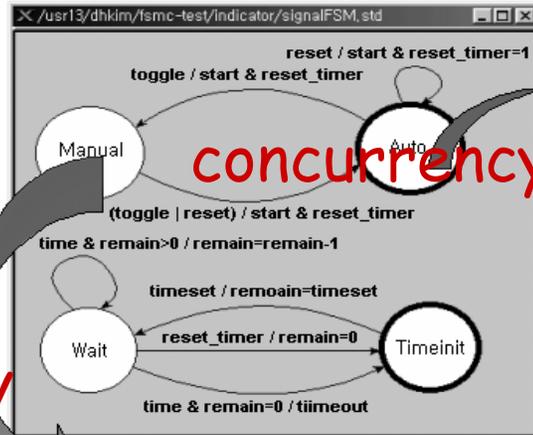
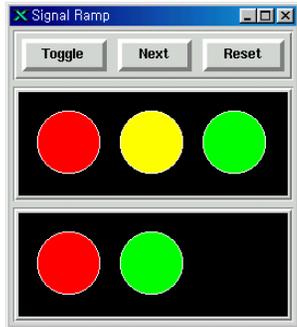
- **fFSM properties**
 - concurrency
 - hierarchy
 - internal event
 - **variable state**: support memory in FSM, a kind of concurrent FSM



Example: Traffic Light



Example : Traffic Light



hierarchy

concurrency

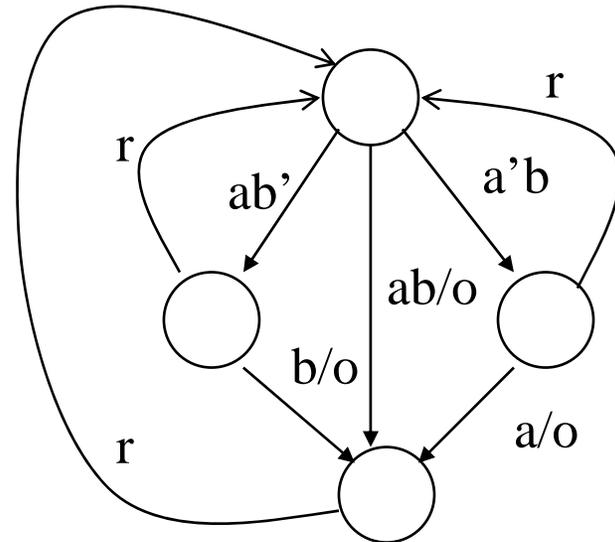
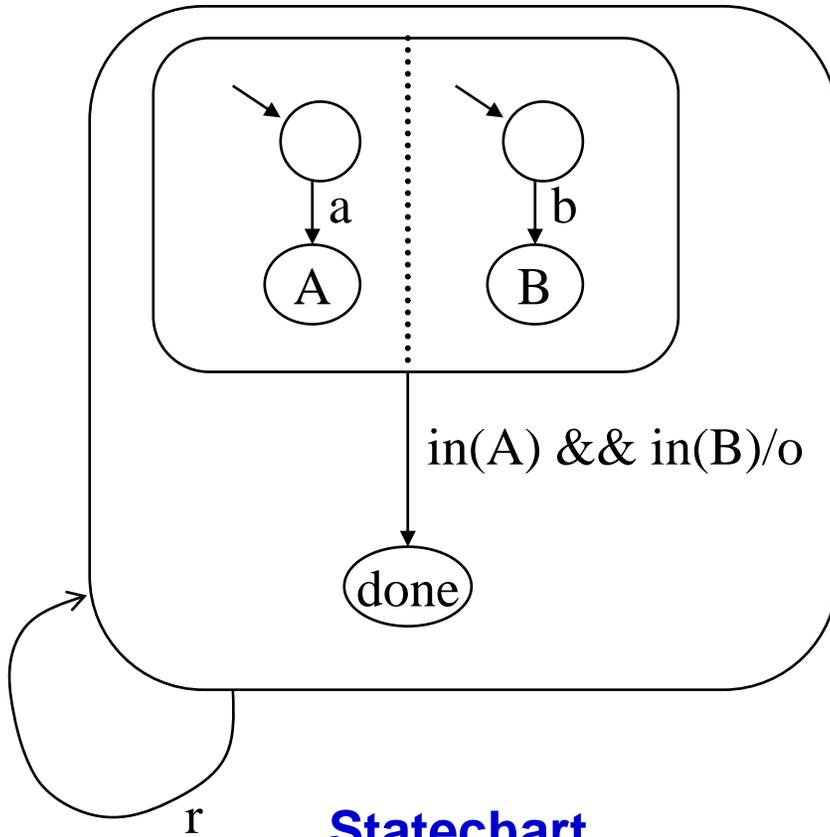


Esterel

- **A language with a textual syntax that describes synchronous automata**
- **Some semantics**
 - $S1;S2$ — sequential execution
 - $S1 \parallel S2$ — parallel execution
 - $\text{do } \dots \text{ watching } R$ — body is executed until either it terminates or an event occurs on the signal R

```
< example >
module EsTest
input a, b, r;
output o;
loop
  do
    [await a || await b];
    emit o;
    halt;
  watching r
end loop
end module
```

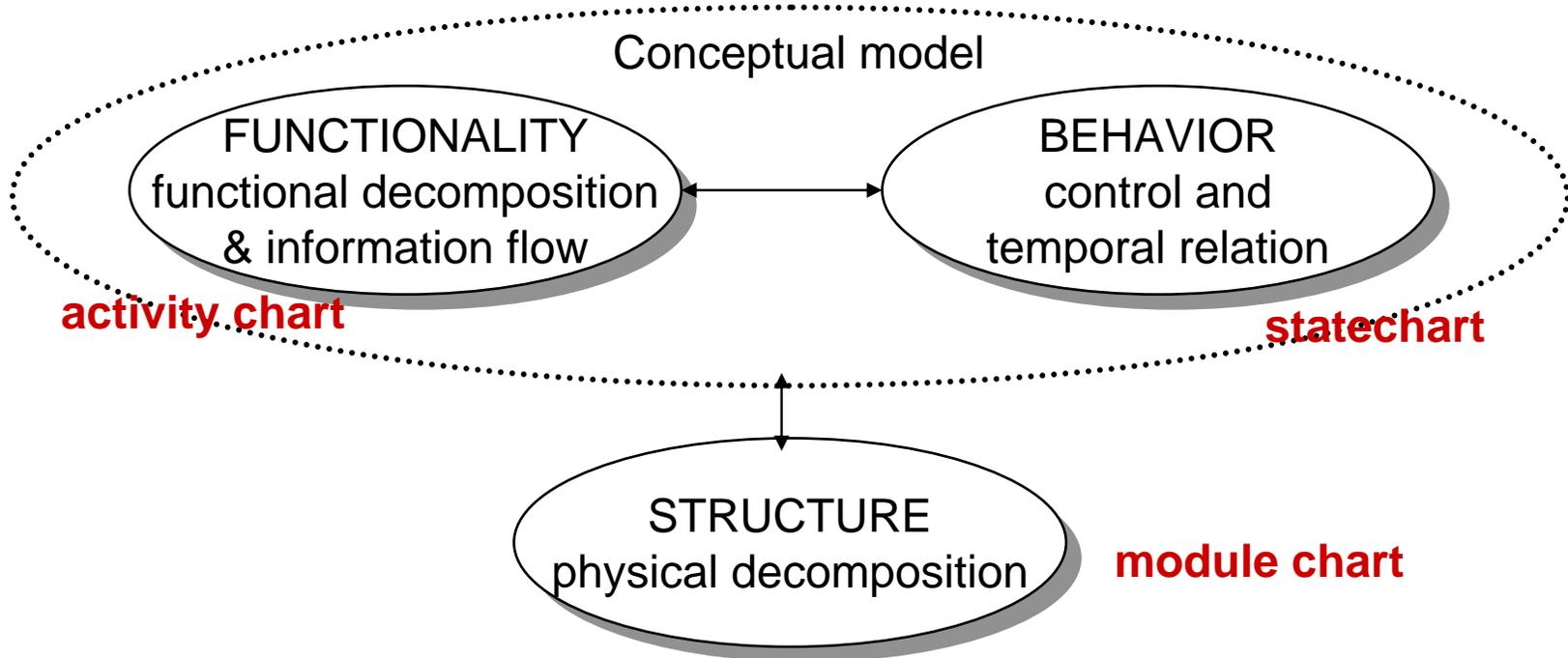
FSM and Statechart Examples



FSM

■ Telelogic (old, iLogix) STATEMATE

- Behavior specification: statechart + activity chart
- Architecture specification: module chart



Activity chart

■ 3 types of objects

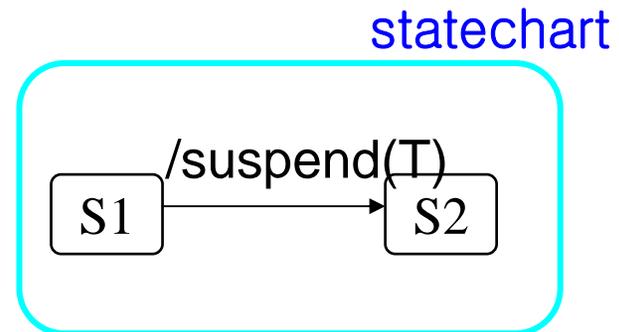
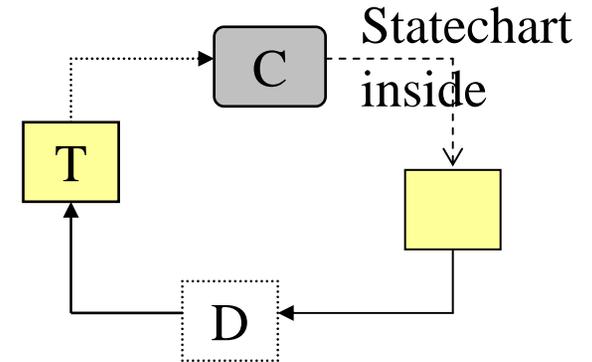
- transformation
- data-store
- control activity (content: Statechart)

■ 2 types of arcs

- dataflow: solid line
- control flow: dashed line

■ connection between activity chart and statechart

- transition labeling (ex)
- forms language



Transition Labeling of Statechart

- **Not level restricted.**
- **General syntax: $A[C] / B$**
 - A: event: primitive or special
 - C: condition *entered(S) [in(T) and not active(C)]/*
 - B: action: primitive or special *suspend(C); x:=y+7*
- **Special events; conditions; actions**
 - **state F:** entered (F), exited(F); in(F)
 - **activity S:** started(S), stopped(S); active(S), hanging(S);
start(S), stop(S), suspend(S), resume(S)
 - **data items D,F:** read (D), written(D); D=F, D<F, etc; D:= exp
 - **condition C:** true(C), false(C); ; make_true(C), make_false(C)
 - **event E, n time untis:** timeout(E,n)
 - **action A, n time units:** ; ; schedule(A, n)

Executable Specification

■ Execution and Dynamic Analysis

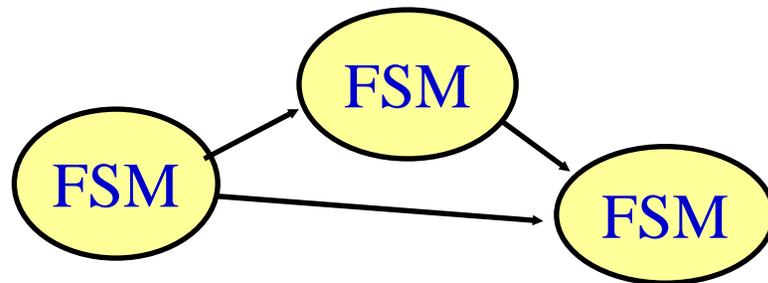
- The most basic way of “running” the SUD is in a “step-by-step” interactive fashion
 - *user generates external events and gives the “go” command*
- For non-interactive execution (or programmed execution), specially tailored *simulation control language (SCL)* has been designed.
- Essentially exhaustive, brute-force, dynamic tests
 - *reachability, non-determinism, deadlock, and usage of transitions*

■ Code Generation and Rapid prototyping

- prototype code: from activity chart to Ada and C program
 - *not as efficient as final real-time code.*
 - *originally for complied execution*

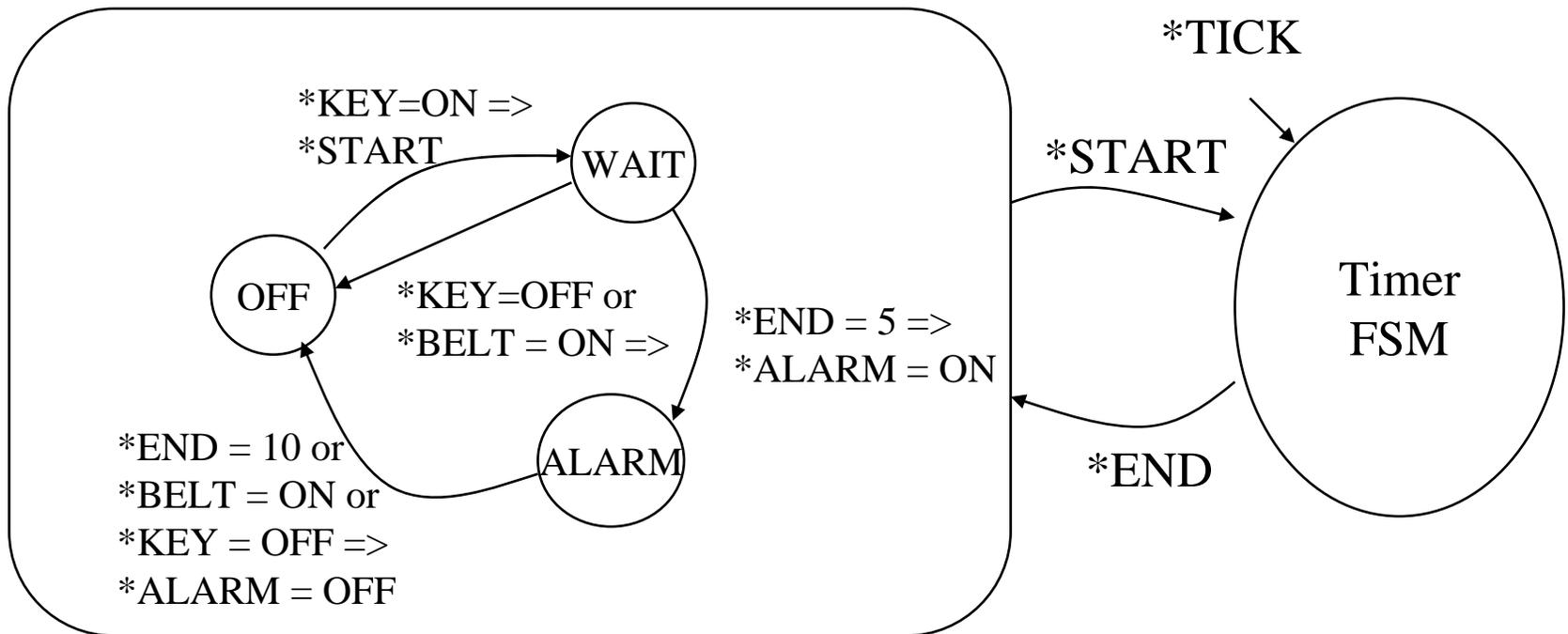
POLIS: Codesign FSM

- **U.C.Berkeley CAD group –A. Sangiovanni-Vincentelli et. al.**
- **CFSM**
 - Formal model as an intermediate representation during the system design process. (relatively small real-time control systems)
 - *Front-end specification: Esterel, StateCharts, a subset of VHDL*
 - Low level enough to be efficiently co-synthesized.
 - a network of interacting FSMs
 - each computing element takes a non-zero unbounded time



CFSM Example

- Five seconds after the key is turned on, if the belt has not been fastened, an alarm will beep for ten seconds or until the key is turned off.



Formality of CFSM

- “Asynchronous” behavior of the network of CFSMs can be translated into a network of “Synchronous” FSMs.

- Correctness of design
 - specification verification: implementation-independent
 - design verification: implementation-dependent

 - prototyping
simulation
formal verification

- Implementation verification is accomplished by construction

Outline

- **Introduction**
- **State-oriented model**
- **Activity-oriented model**
 - Discrete event Model
 - SIMULINK
 - SystemC Model
 - Dataflow Model (chapter 5 + more)
- **Process network model**
- **Heterogeneous approach**
- **System-level specification experiment**

Discrete-Event Model

■ Main applications

- Reveal the system dynamic characteristics
- (queueing) network simulation, hardware simulation

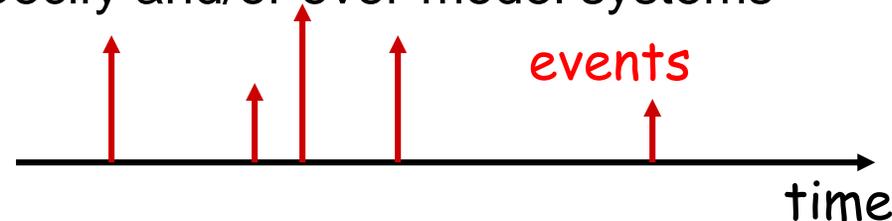
■ Module execution order is determined by incoming events

■ Characteristics

- Natural for asynchronous digital hardware
- Global synchronization
- Timing simulation

■ Limitation

- Expensive to implement in software
- May over-specify and/or over-model systems



Execution Policy of Discrete Event Model

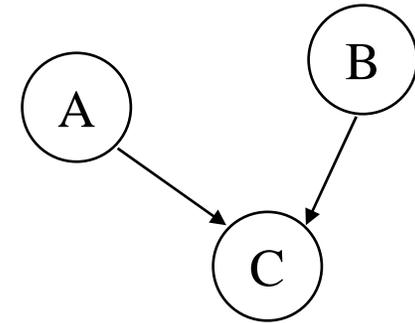
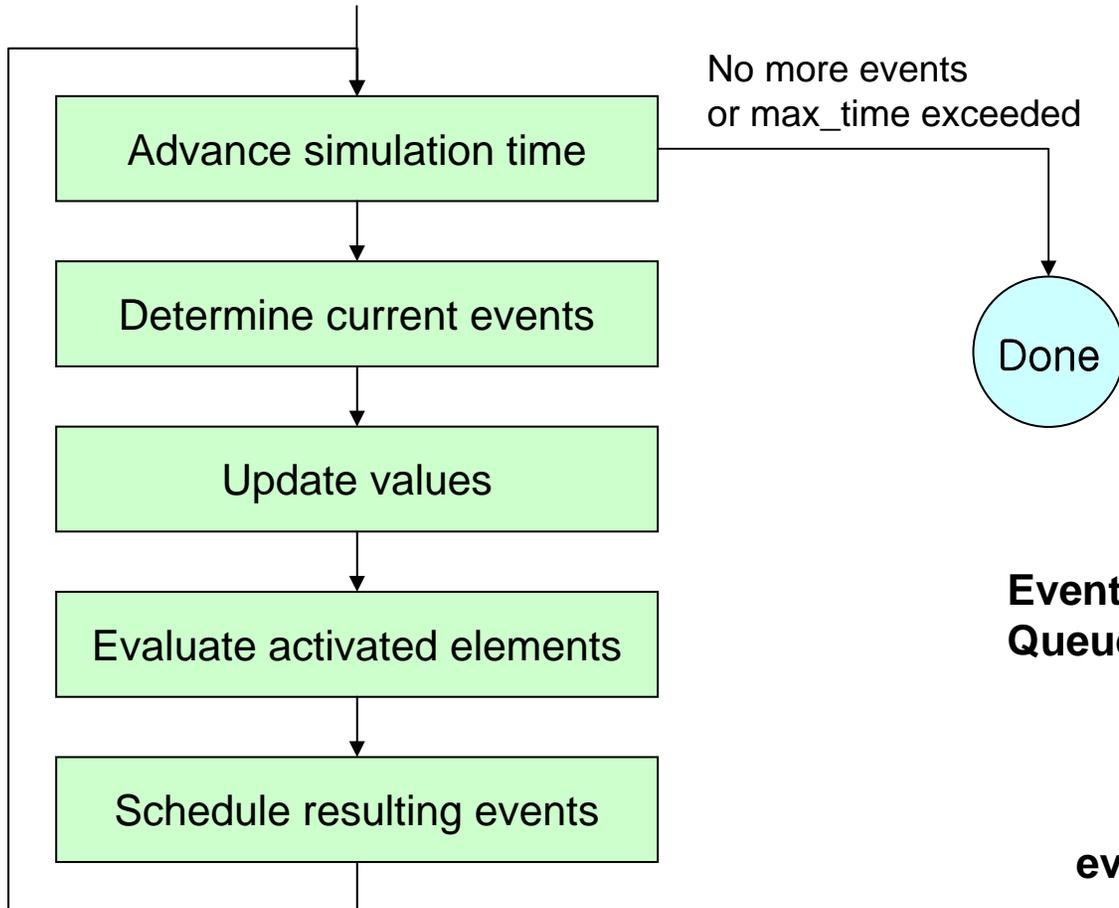
■ Event-driven Simulation

- Global event queue manages the outstanding events in the chronological order
 - *Event queue management overhead is huge: sorting complexity $O(N \log N)$.*
- Efficient if events are generated in an irregular fashion.
 - *(ex) network simulation*
- Should avoid causality error
- How to resolve simultaneous events?
 - *All events in a minimal time slot are regarded as simultaneous*
 - *VHDL uses the notion of delta time*
 - *Other options? (topological sort, fixed point computation)*

■ Time-driven Simulation

- Invoke every module every cycle
 - *Skip execution if there is no incoming event*
- Efficient if events are generated regularly
 - *(ex) hardware simulation, architecture simulation*

Event Driven Simulation



Event Queue

A-C, 5us, 1.0
B-C, 5us, 0.0
B-C, 2us, 3.0
A-C, 1us, 2.0

event path event time value

- **Started purely as a simulation environment**
- **SIMULINK Model**
 - An extension to MATLAB® that allows developers to rapidly build computer models of dynamic systems
 - de facto standard in many industrial application domains, particularly automotive control.
 - Has a multitude of semantics depending on user-configurable options, informally and sometime only partially documented
- **Simulink-based Tools**
 - The Mathworks: Real-Time Workshop
 - dSPACE: TargetLink
 - *Generate code only for blocks of the dSpace-provided Simulink library*

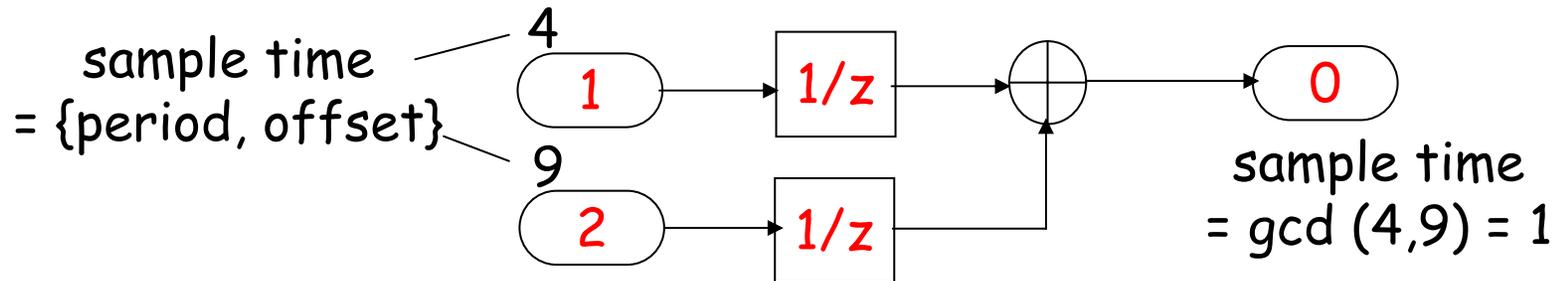
Simulink Model (1)

- **Simulink has a continuous-time semantics**

- A Simulink block in a “Discrete library” produce piece-wise constant continuous-time signals.

- **Notion of “sample time”**

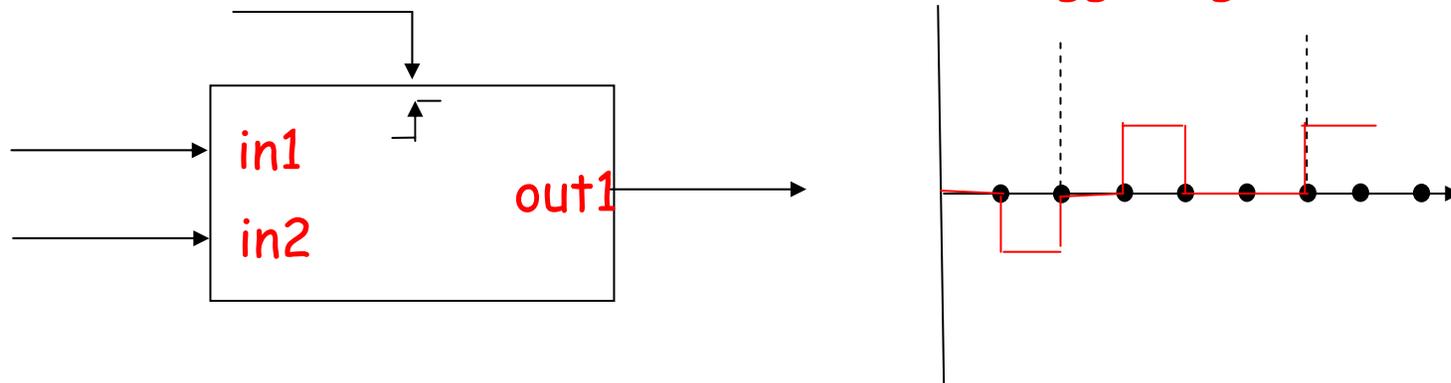
- “-1” means that it is inherited from the predecessor



Simulink Model (2)

■ Triggered subsystem

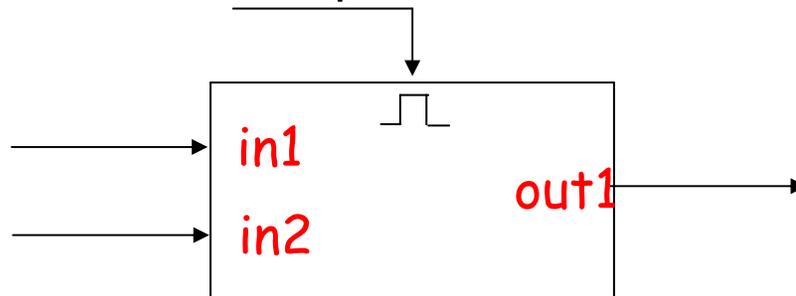
- The sample time of blocks inside a triggered subsystem cannot be set by the user.
- The sample times of the trigger, inputs and outputs of B must all be equal.



triggering event: the signal has to remain at zero for more than one time step before triggering

■ Enabled subsystem

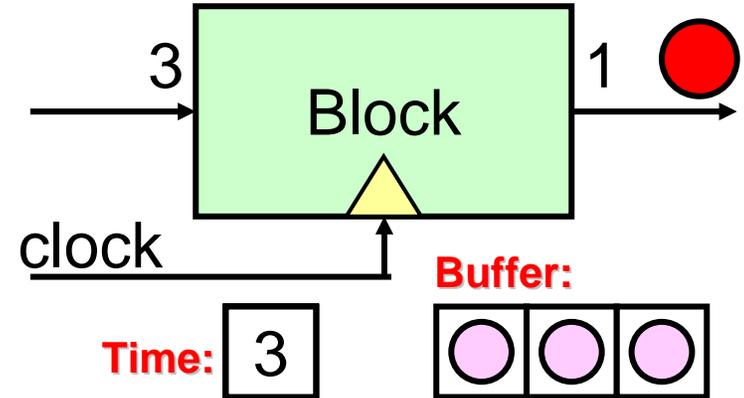
- The subsystem is enabled every time the enabling signal is non-zero
- No restriction on sample times to make it very complicated



- Code can be safely generated with the same restriction as the triggered subsystem: all sample times of its internal blocks are “-1”

H.264 Decoder Modeling in Simulink

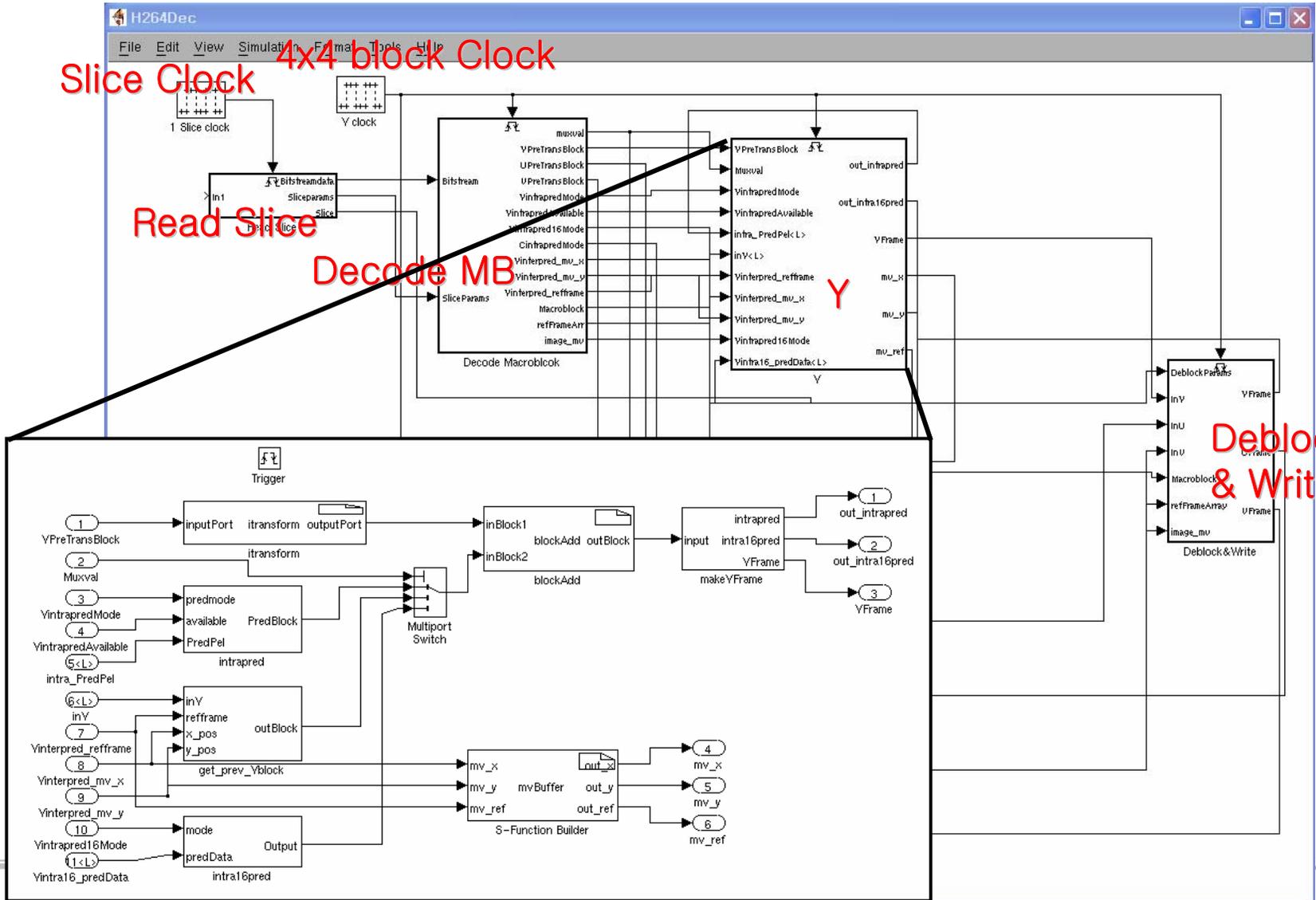
- **Use clocks, counter variables, and some buffers to model multirate property**
 - To know when to update the buffer and when to execute the block, counter variable is needed
 - All ports of block(clocks, counter variables and buffers) should be controlled separately



```

void block() {
    static int counter = 0;
    counter++;
    inputBuffer[counter] = read_data;
    if (counter==3) {
        counter=0;
        run(); //main code
        write_to_outputPort;
    }
}
    
```

Top model of H.264 decoder in Simulink



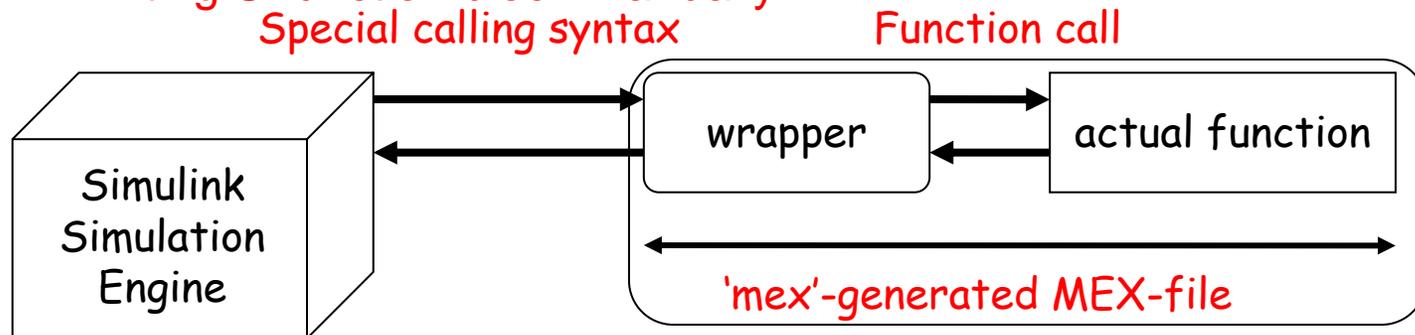
Block Definition in Simulink

■ S-function

- a computer language description of a Simulink block
- supports C, C++, Ada, and Fortran for modeling blocks

■ How to use S-function?

1. Use S-function builder → too restricted
2. **Writing S-function block manually**
Special calling syntax



■ Difficulties of debugging

- “printf” can only be used in “wrapper file”
- If some block has a critical bug, it kills Simulink itself

■ SystemC is modeling platform

- a set of C++ class library, plus a simulation kernel level
- Not a new language, but a C++ class library to provide hardware style communication, notion of time, concurrency, reactivity, and hardware data

■ SystemC can be an executable specification of the system.

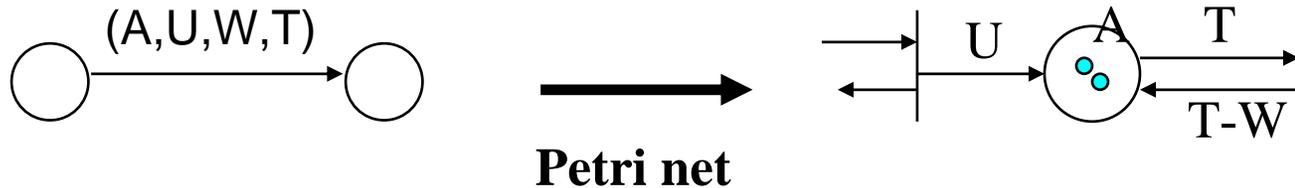
■ Remind SystemC specification

- A **system** consists of a set of **concurrent processes**
- Processes communicate with each other through **channels**.
- Processes can be combined into **modules** to create **hierarchy**.

Dataflow Coordination Languages

- **Karp and Miller's computation graph**
- **Synchronous dataflow**
- **Cyclostatic dataflow**
- **Processing graph method (PGM)**
- **Granular lucid**

Computation Graphs of Karp and Miller



■ Symbols

- A: # of tokens initially in the queue
- U: # of tokens to be added after the input node is executed
- W: # of tokens to be fetched for the output node to be fired.
- T: the minimum queue length for the output node to execute

■ Characteristics

- determinate: execution result is independent of execution order
- static analysis of termination conditions
- static analysis of storage requirements

Marked Graph

- **Marked Graph is a subset of Petri nets.**
 - Each place is an input for exactly one transition and an output for exactly one transition -> regard a place as an arc in homogeneous dataflow models.
 - Cycle: a closed sequence of transitions that form a loop
 - Cycle analysis
 - *live: the number of tokens on each cycle is at least one*
 - *safe: every place is in a cycle and every cycle has exactly one token.*

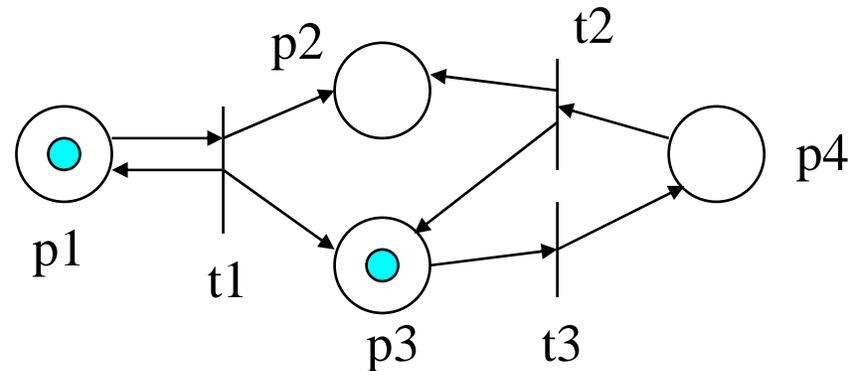
■ **PN = (P,T,A,M)**

- P: Place
- T: Transition
 - *voluntary,*
 - instantaneous,*
 - complete*

● A: Arc

● M: Initial marking

– $M = \{m_1, m_2, m_3, m_4\} = \{ 1 \ 0 \ 1 \ 0 \}$



- **Neither Turing Complete, nor finite state**
- **Uninterpreted model for useful analysis**
- **Events within signals need not be ordered,**

Analysis Example

■ Safeness

- No more than one token can ever be in any place of the net at the same time: place = condition
- There is a bound on the number of tokens in any place of the net

■ Boundedness

- The number of tokens is bounded by $k \rightarrow$ queue size

■ Conservative

- strictly conservative: The number of tokens is conserved.

■ Liveness

- Every transition is live: executable and reachable from initial marking

Reachability Tree

■ Finite reachability tree

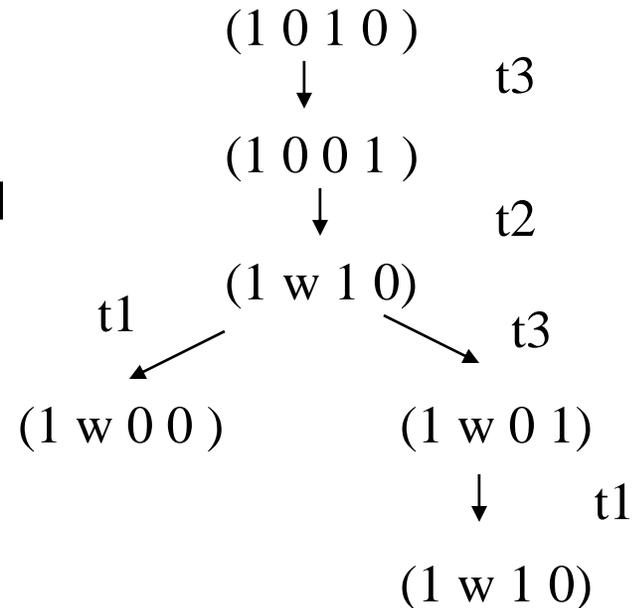
- w if $M' > M \Rightarrow$ the number of tokens grow arbitrarily.
- Explode easily.

■ Reachability problem

- solvable, but
- exponential time-hard, space-hard

■ Unsolvable problems

- given two marked Petri nets has the same reachability set.



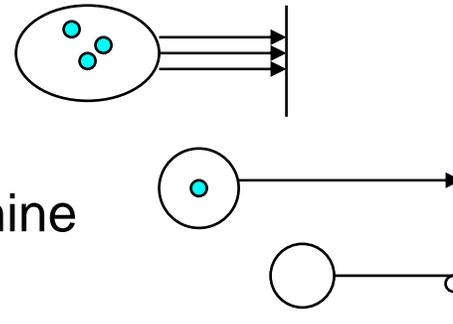
Extension to Petri nets

■ Generalized Petri nets

- equivalent to ordinary Petri nets

■ Zero-testing

- same modeling power as Turing machine



■ Stochastic Petri net (SPN)

- $SPN = \{ P, T, A, M, Q \}$ where $Q = \{q_1, q_2, \dots\}$ average transition rate for exponentially distributed firing times
- average performance analysis

■ Petri net + time

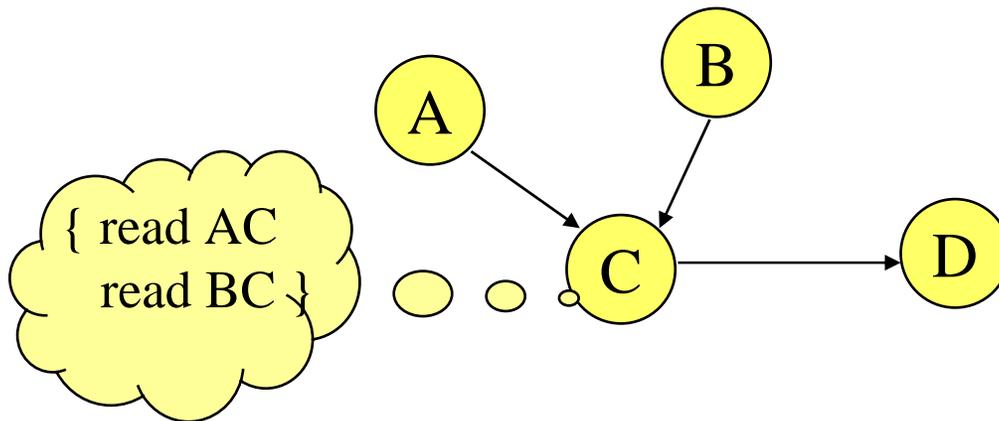
- timed Petri net: fixed delay
- Time Petri net: delay interval

Outline

- **Introduction**
- **State-oriented model**
- **Activity-oriented model**
- **Process network model**
 - Kahn Process Network
 - YAPI model and COSY
 - Dataflow Process
- **Heterogeneous approach**
- **System-level specification experiment**

Kahn Process Networks

- A process is a mapping from input sequences to output sequences.
- Blocking read, non-blocking write
- Concurrent processes communicate only through one-way FIFO channels with unbounded capacity.



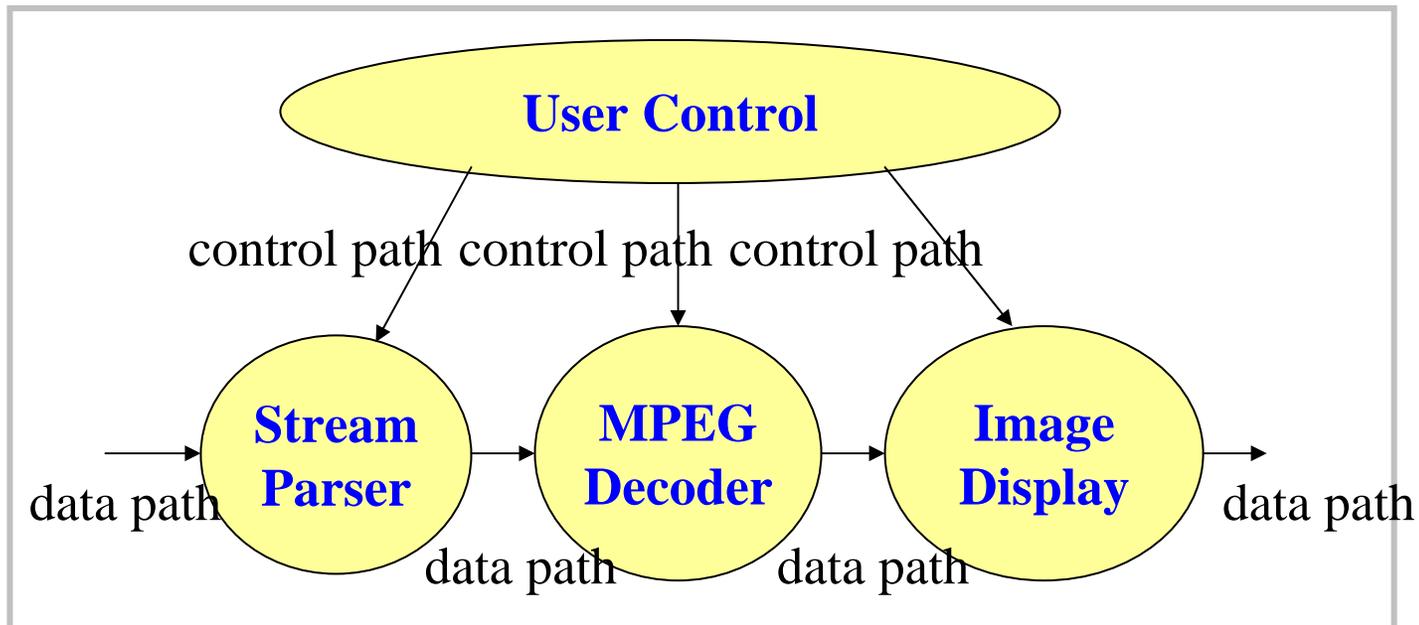
Determinate: independent of execution order of A and B

KPN Characteristics

- **Rather easy to translate legacy C code to KPN**
- **Deterministic execution**
- **Execution Policy**
 - Demand driven execution
 - *Minimize resource requirement*
 - *Run-time overhead*
 - Data driven execution
 - *It is not easy to determine the queue size: overflow or deadlock*
- **Asynchronous input is prohibited**

Design Framework : COSY

- **IP-based real-time design methodology**
- **YAPI: extension of the Kahn process network model**
 - Kahn process network + “select” operation
- **coarse grain mix-and-match of several IP blocks**



- **Unified approach**

- control module and computation module are inside a process: not distinguishable from the outside

- **YAPI model provides a coordination method of different components**

- YAPI: Y-chart Application Programmers Interface

- **COSY does not specify any formal model inside a component process - only coarse grain composition**

■ Processes

- read, write are blocked when data is not available or cannot be delivered
- “select” takes two input ports as input and returns a port ID, If neither input port has data available, it blocks. If both have, select one non-deterministically.

■ Directed FIFO

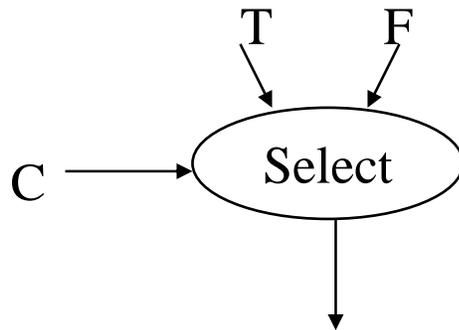
■ Process network

```
n = select (in1, in2)
if (n == 0) {
    read (in1, x);
    f1(x);
} else if (n == 1) {
    read (in2, y);
    f2(y);
}
```

Code fragment

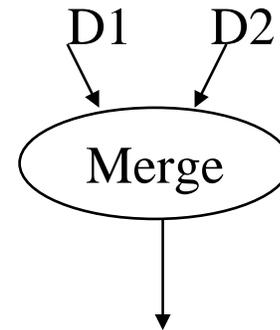
Dataflow Process

- Each process is decomposed into a sequence of *firings*.
- Firing rules: a set of input patterns to fire the actor



$$R_1 = \{*, \perp, T\}$$

$$R_2 = \{\perp, *, F\}$$



$$R_1 = \{*, \perp\}$$

$$R_2 = \{\perp, *\}$$

Sequential Firing Rules

- **Dataflow coordination language + host language**
- **Sequential Firing rules** $R = \{R_1, R_2, \dots, R_N\}$
 - Find an input j such that all firing rules require at least one token from that input. If no such input exists, **fail**.
 - For the choice of input j , divide the firing rules into subsets remove input j from all firing rules.
 - If all subsets have empty firing rules, then succeed. Otherwise, repeat these steps for any non-empty subset.



Select has sequential firing rules

- examine the control input first and choose R_1 or R_2 as the next firing rule

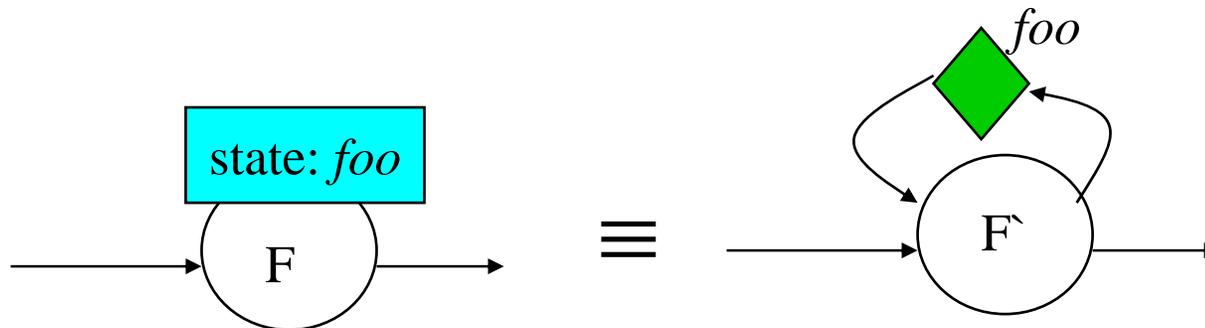
Merge does not have sequential firing rules

- nondeterminate!

Dataflow Process

- **Sufficient condition for a dataflow process to be continuous**
 - **actor firing is functional:** output tokens are purely a function of input tokens: stronger condition than Kahn condition that a process be functional (actors can have and manipulate states)
 - **the set of firing rules be sequential**

- **State:** syntactic sugar of a self-loop



Kahn Process vs. Dataflow Process

■ Kahn process

- context switch on blocking read: context switch overhead of processor suspension on blocking read and processor resumption.
- Large granularity.

■ Dataflow process

- processes can be freely interleaved by a scheduler in the unit of firings.
- Finer granularity is practical.

■ Concurrent processes

- demand-driven style multi-tasking (Kahn and MacQueen)
- data-driven multi-threading (Park)

■ Dynamic scheduling

- by hardware: dataflow architecture
- by software: the scheduler tracks the availability of tokens on the inputs to the actors and fires actors that are enabled.

■ (Quasi) static scheduling: SDF, BDF, and DDF

- static scheduling as much as possible

■ Tagged-token model - no need for FIFO discipline

■ Merits

- Loose synchronization (distributable)
- Determinate under simple conditions
- Maps easily to threads, but much easier to use
- Turing complete (expressive)

■ Limitation

- Control-intensive systems are hard to specify
- Resource requirement can not be predicted

- **Introduction**
- **State-oriented model**
- **Activity-oriented model**
- **Process network model**
- **Heterogeneous approach**
 - Ptolemy II: U.C.Berkeley
 - Cocentric System Studio: Synopsys
 - PeaCE: S.N.U.
- **System-level specification experiment**

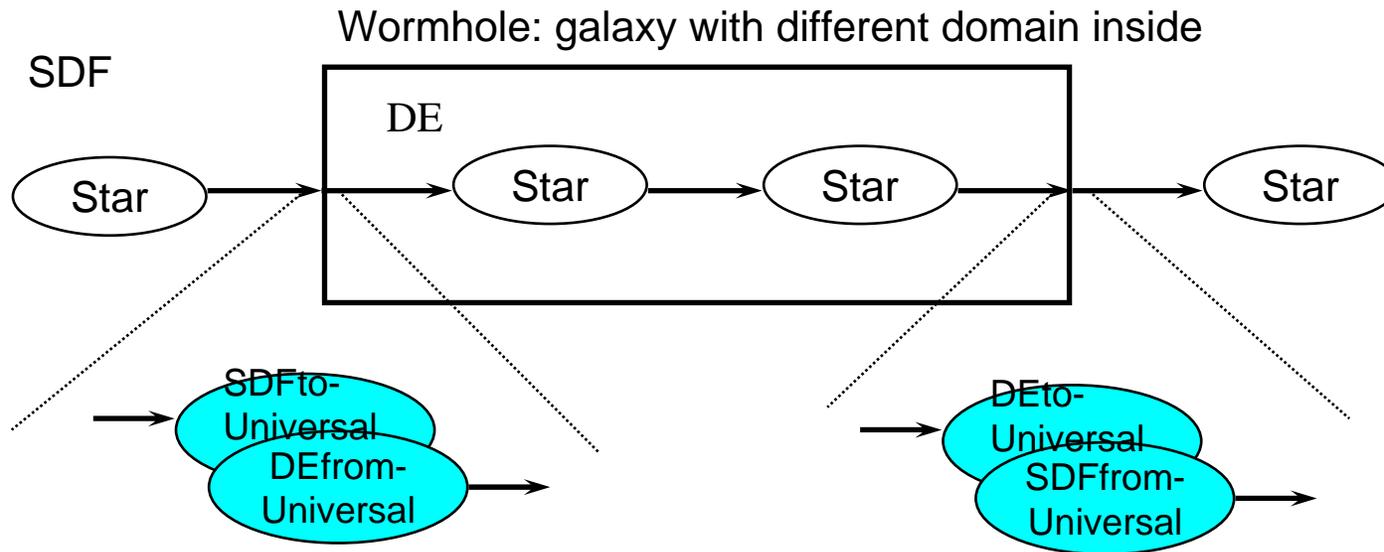
Ptolemy Classic

■ Pioneering work of heterogeneous modeling

- Started in 1990 at U.C.Berkeley
- C++ kernel

■ Hierarchical Domain interface

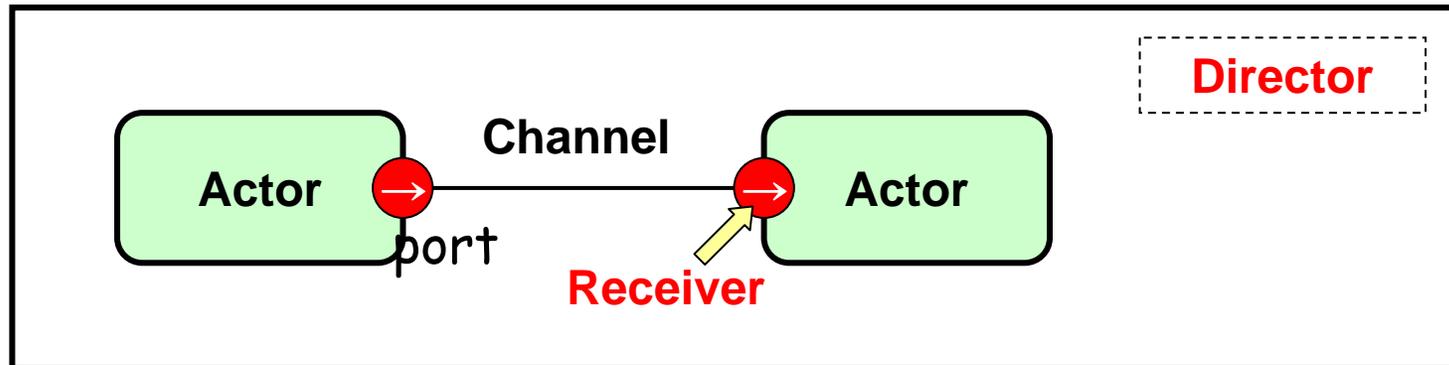
- Wormhole mechanism



“EventHorizon”

- derived from PortHole

- Start from the scratch with rigorous treatment of formal model
- Java-based framework



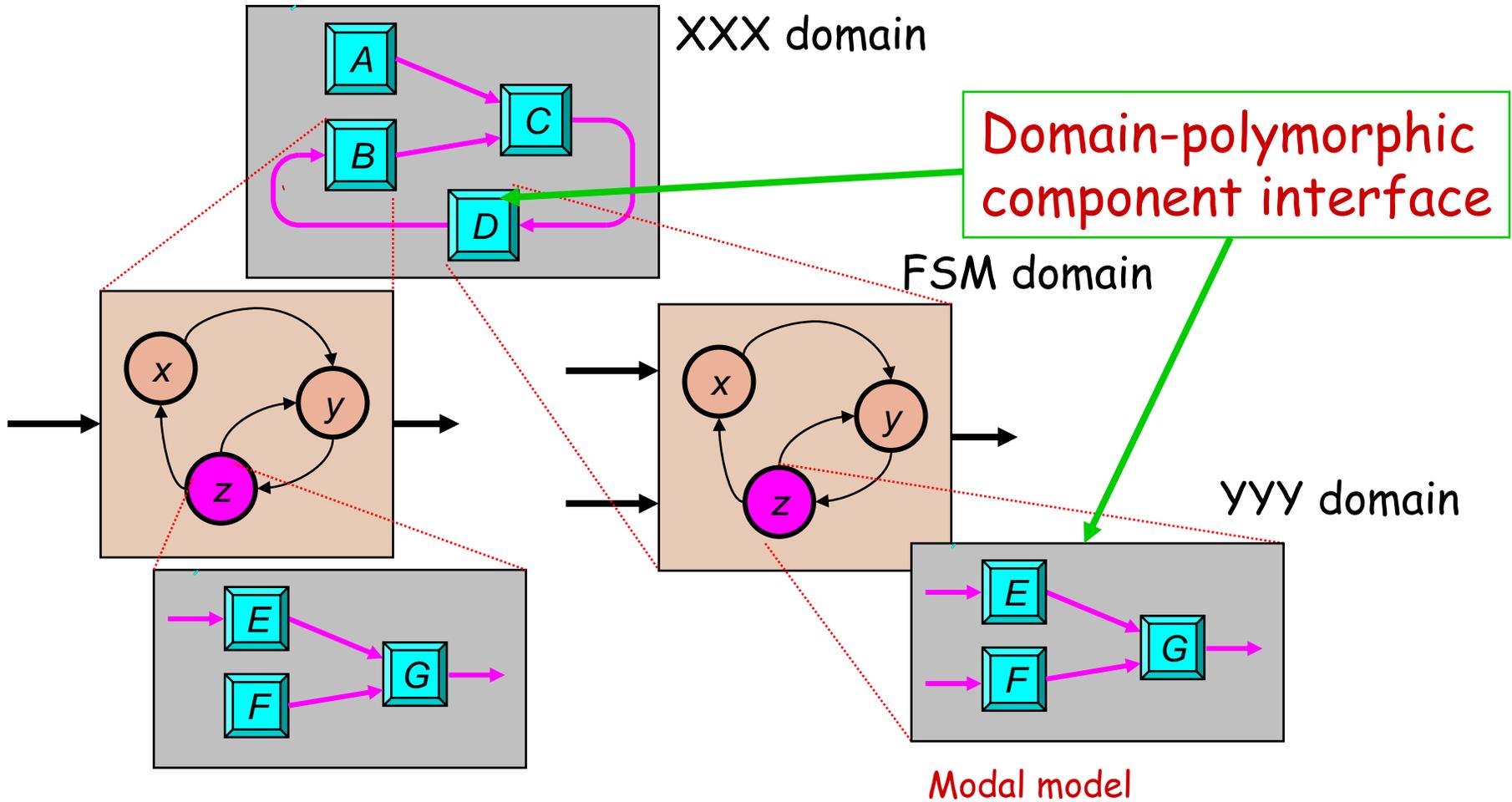
Director : determine the execution semantics of the model

ex) is actor active? passive? reactive?

Receiver : determine the communication protocol

ex) timed, synchronized, or buffered communication

*Charts: Exploiting Domain Polymorphism



* chart: Ptolemy II approach

■ Hierarchical FSM with multiple concurrency models

■ Motivation

- Decoupled FSM semantics from concurrency semantics unlike Statechart
- Combining FSMs with concurrent models of computation is attractive: existing ones tightly intertwine the concurrency model with the automata semantics
 - *SDL: FSM + process networks*
 - *CFSM: FSM + discrete-event concurrency*
 - *FSM + SDF*



Decouple the concurrency model from the hierarchical FSM model
Do not enforce a specific concurrent model

■ **Compositionality**

- composite modules can be treated as primitive modules

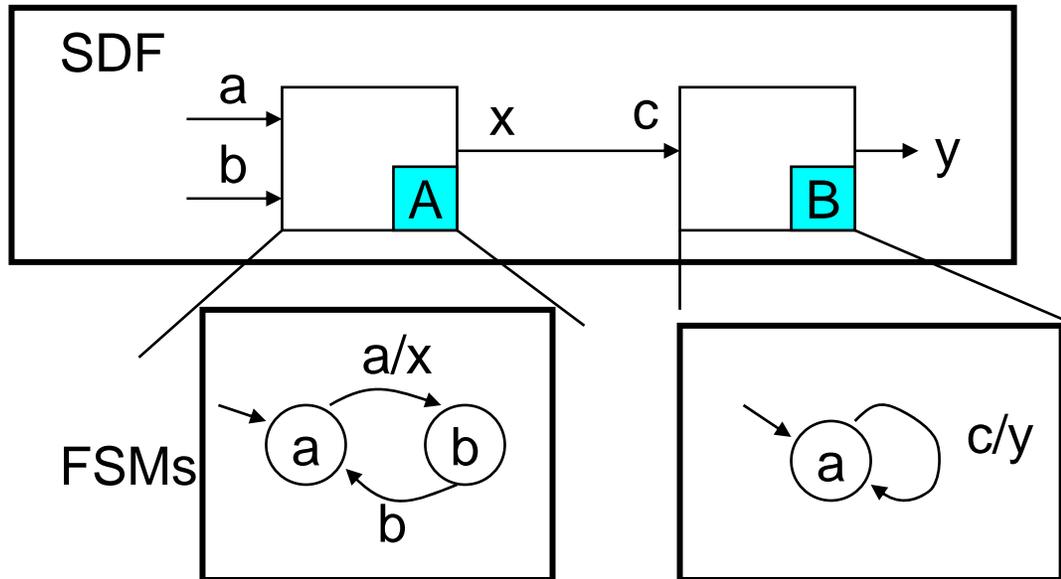
■ **Heterogeneity**

- composite modules can be embedded within a foreign model of computation

■ **Termination**

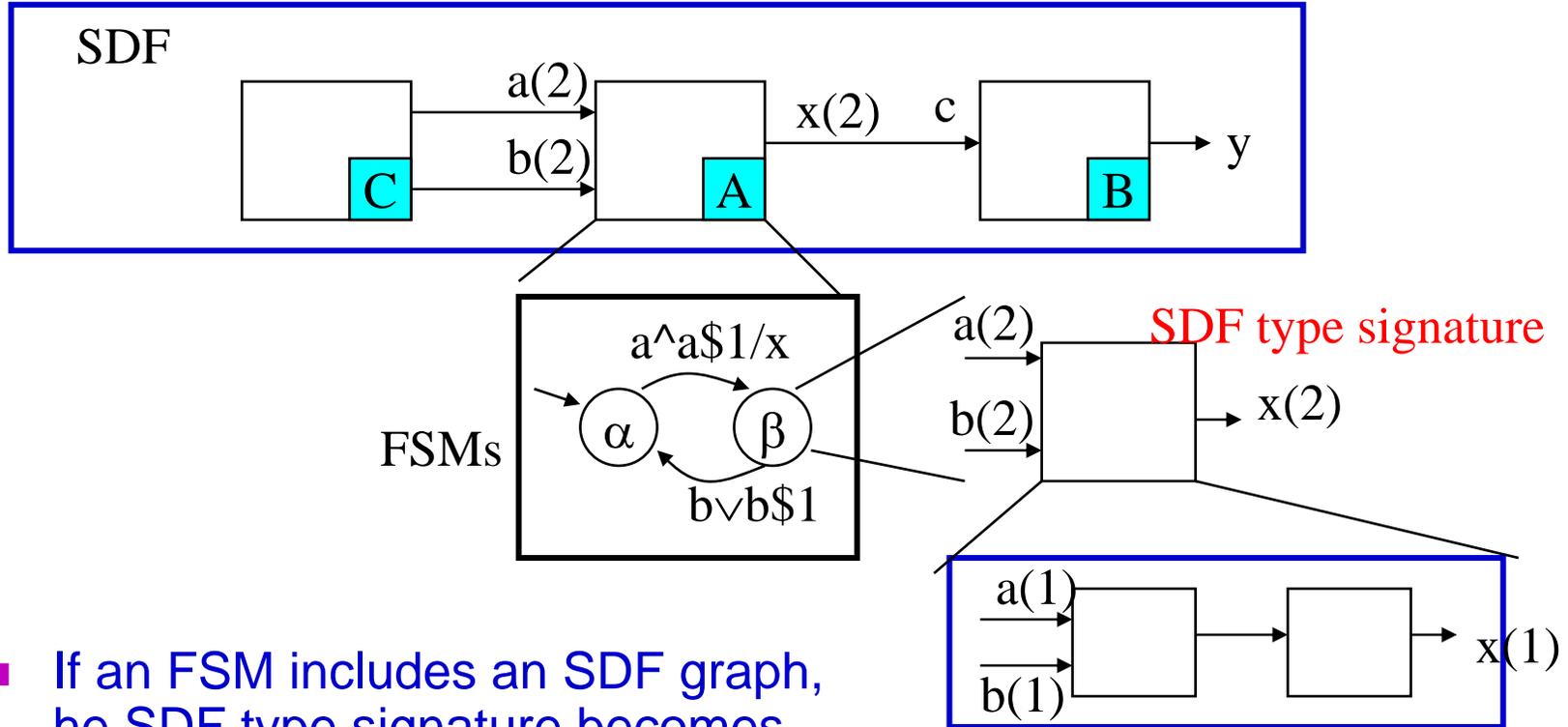
- Any concurrent model of computation that can refine a state of an FSM have a well-defined finite iteration.
 - *Non-terminating system can often be divided into a set of iterations (ex, SDF model)*

Dataflow with FSM



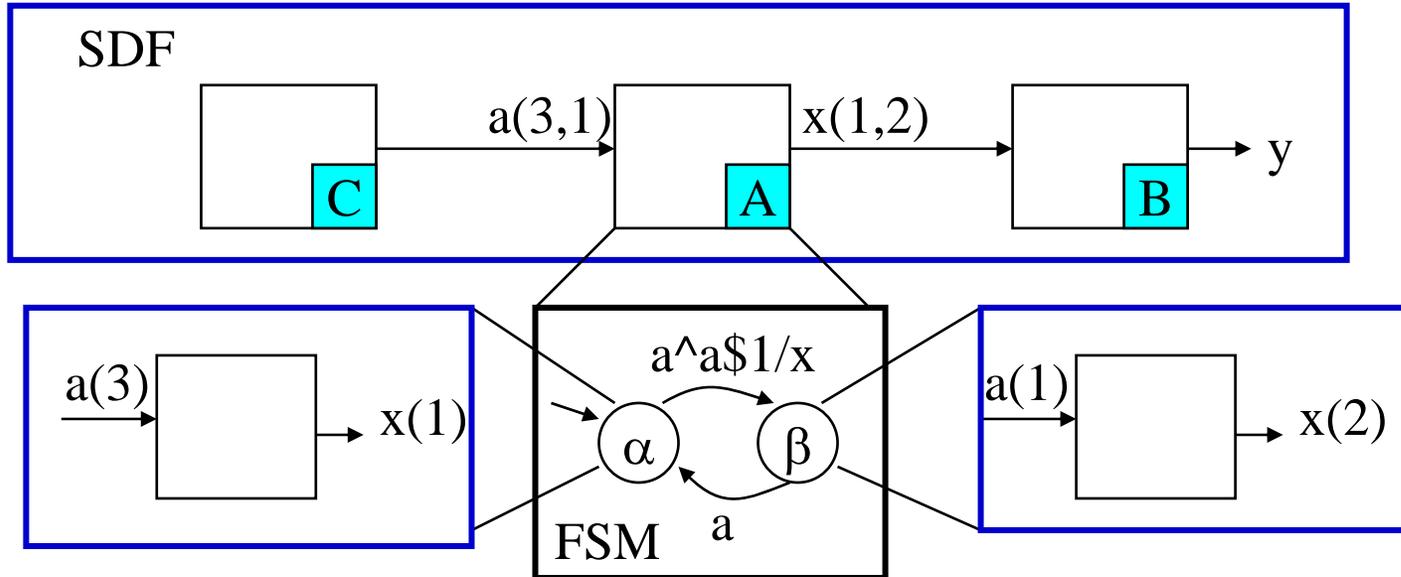
- Only subtlety: “absent” event appears explicitly as a token in the SDF graph
 - encode presence and absence using boolean-valued tokens

Multirate Dataflow with FSM



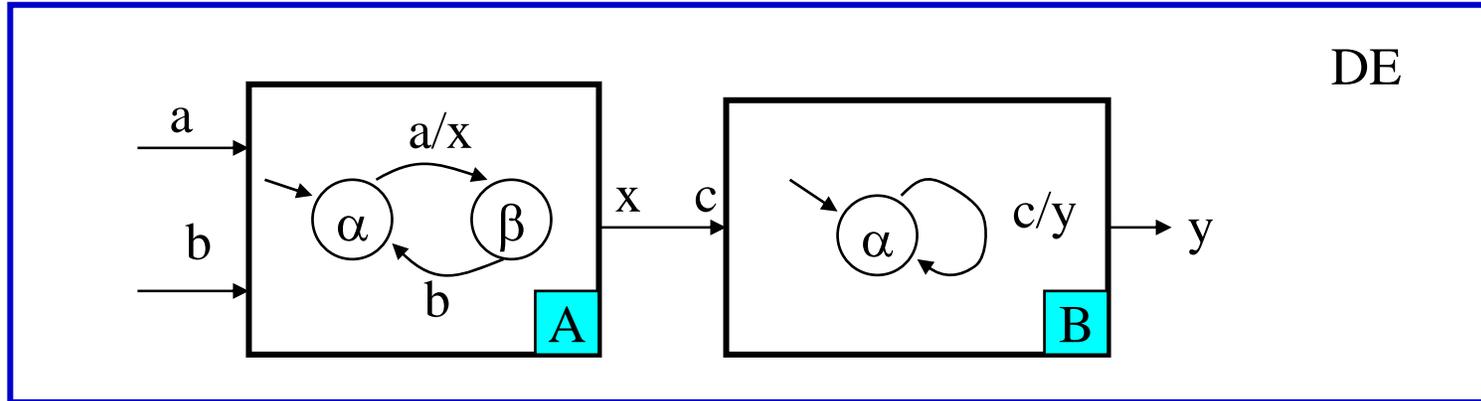
- If an FSM includes an SDF graph, the SDF type signature becomes that of the FSM subsystem.
- When the SDF schedule is CAAB, the first firing of A does not make state transition (type A firing), while the second firing does state transition (type B firing) - single transition in an iteration

Heterochronous Dataflow with FSM

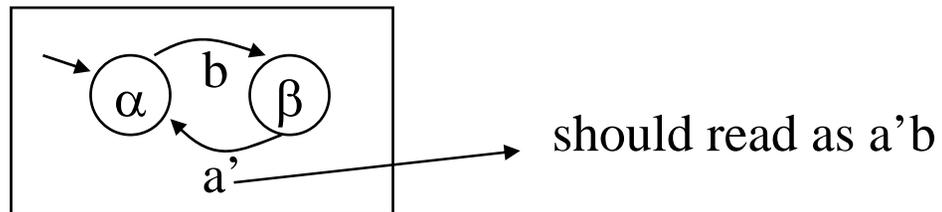


- In HDF, an actor has a finite number of type signature.
- Unlike CSDF, the order of which type signatures are used is not predictable.
- Still key analytical properties can be statically determined.
 - **Deadlock and bounded memory**

Discrete Events with FSM



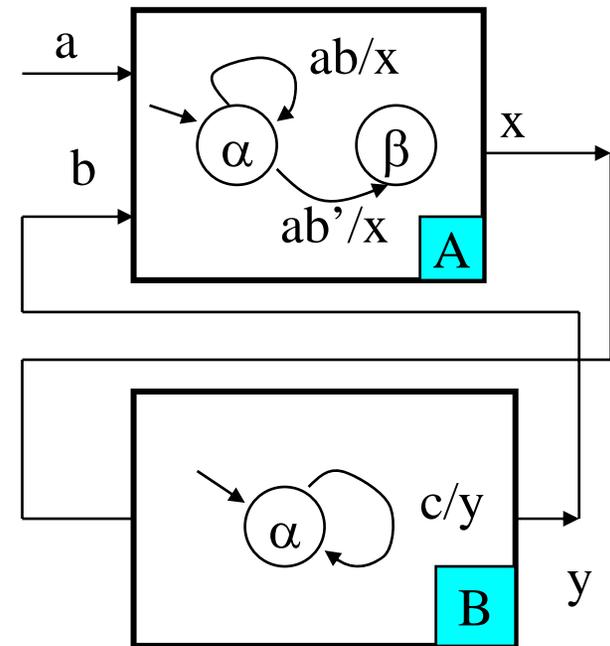
- Most straightforward combination from a control perspective.
- The FSM system appears to the DE system as a zero-delay actor: alert on zero-delay loop!
- Absence of token implies null event. - no transition should be made by null event.



Synchronous/Reactive System with FSM

- Two phases of execution of an FSM within SR
 - **Produce phase (type C):** produce outputs without transition any number of times in a single tick until a fixed point is reached
 - **Transition phase (type D):** state transition ignoring actions

- **Refined FSM inside SR**
 - more difficult



- **Synopsys's SystemC Design and Verification tool suite**
 - SystemC simulator and specification environment at **multiple** levels of abstraction
 - Joint verification and analysis of algorithms, architectures, hardware, and software

- **System Specification**
 - Hierarchical, graphical and language abstractions – unlimited hierarchical composition of dataflow graphs (DFGs) and finite state machines (FSMs)
 - *Dynamic data flow + hierarchical and concurrent FSM (cf) *-chart*
 - Block specification: C, C++, or SystemC

PeaCE from SNU CAP Lab.

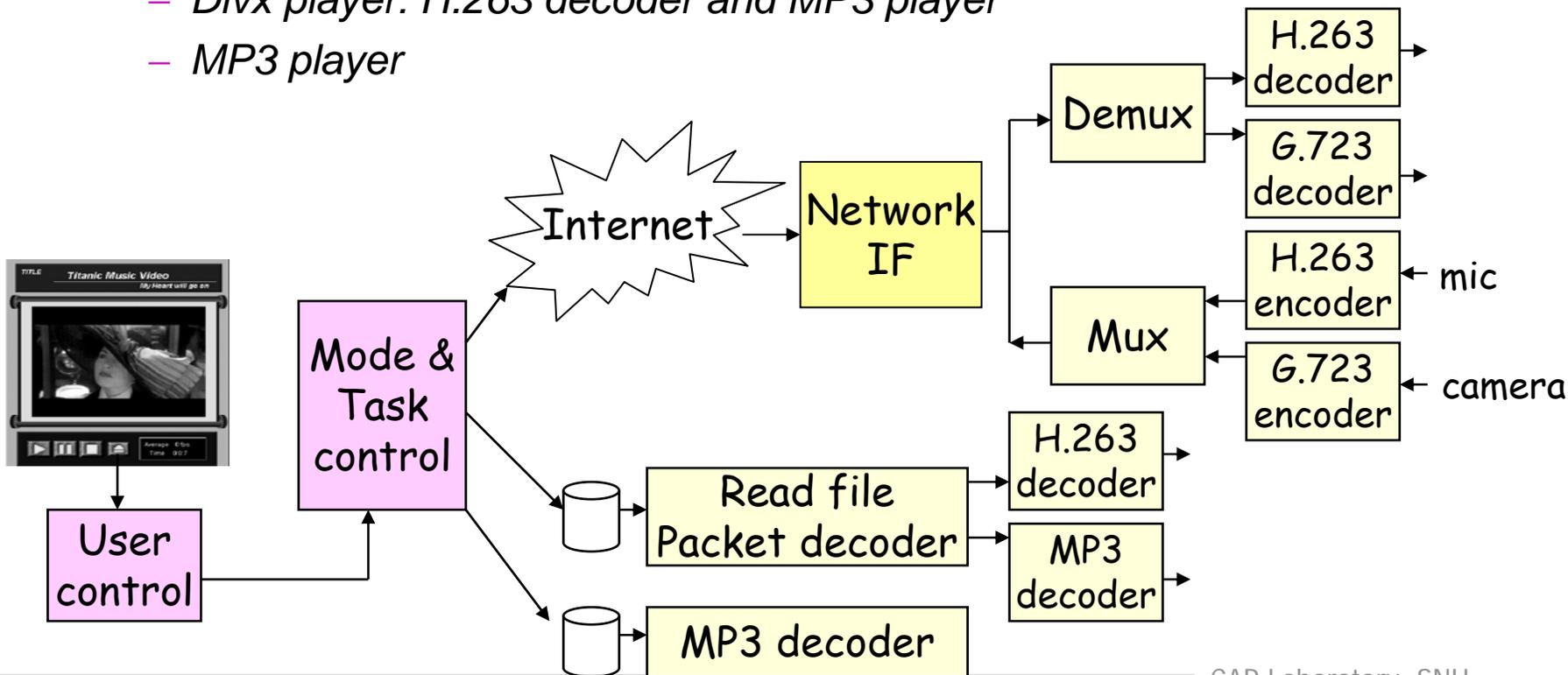
- **Hybrid approach of STATEMATE and Ptolemy**
- **Restricted composition of heterogeneous models of computations**
 - fFSM: control logic specification
 - SPDF : computation (signal processing) specification
 - A novel Task-model is used for top-level task-level specification
- **Motivation**
 - Use models consistently from specification to synthesis
 - *synthesis path from fFSM and SPDF is well established.*
 - Use natural combinations of heterogeneous models.

Motivational Example

Multi-mode Multi-media Terminal Example

- Three modes of operation:

- Video phone: H.263 encoder/decoder and G.723 encoder/decoder
- Divx player: H.263 decoder and MP3 player
- MP3 player



Challenges of Target Application

■ Task-level specification:

- (P3) How to support multiple modes of operations?
 - *Various interactions between tasks: data and control flow*
- (P4) Diverse execution semantics of tasks
 - *Data-driven, event-driven, or time-driven*
- (P5) Diverse port semantics
 - *FIFO queue type, buffer type*

■ Signal processing task specification

- (P1) Function-level HW/SW partitioning is desired
- Data dependent execution of task behavior

■ Control task specification

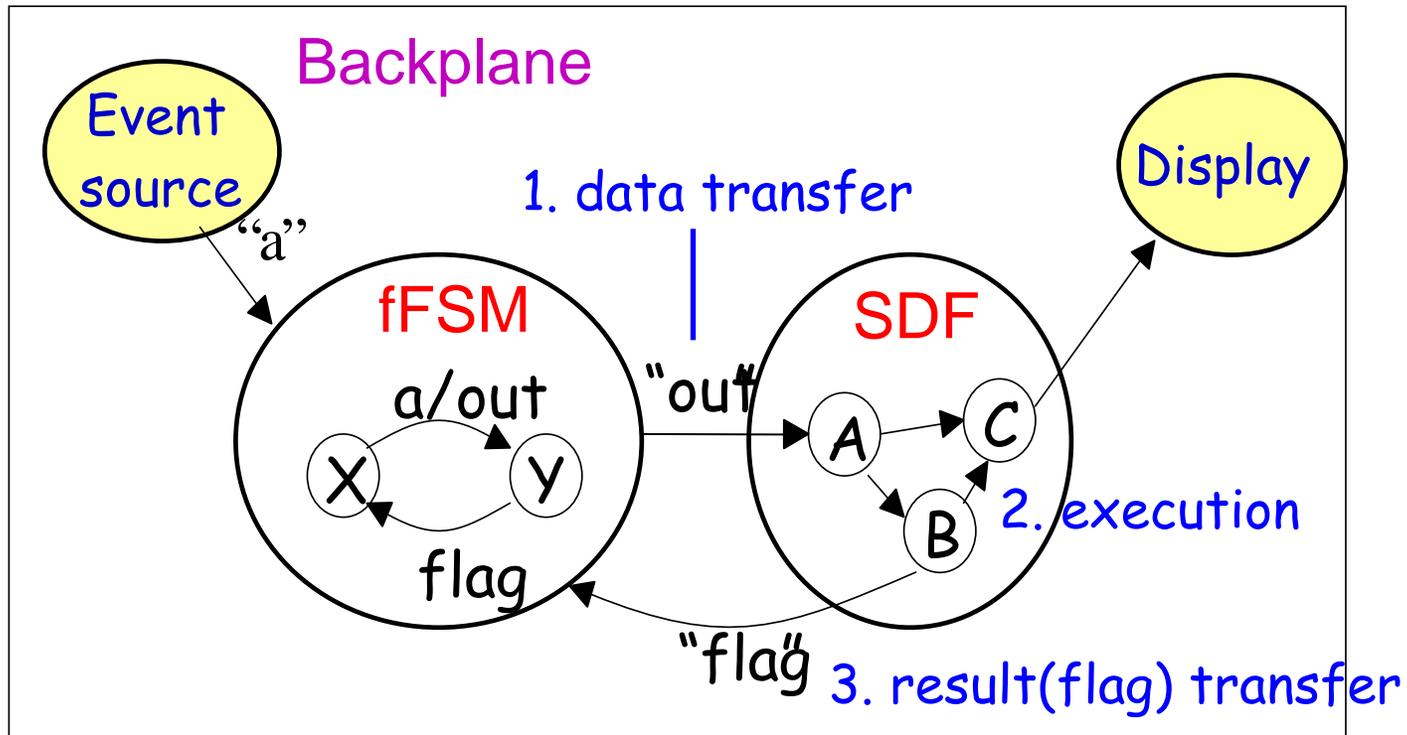
- (P2) Hierarchical and concurrent specification is required

- **Signal Processing Tasks: answer to (P1)**
 - SDF Extension for Memory Optimization: **FRDF** (Fractional Rate Data Flow)
 - SDF Extension to enhance expression capability: **SPDF** (Synchronous Piggybacked Data Flow)

- **Control Tasks**
 - **(P2) fFSM**: Hierarchical and Concurrent FSM Model: a variant of statechart
 - **(P3) fFSM** Model supervises the behavior of the signal processing tasks

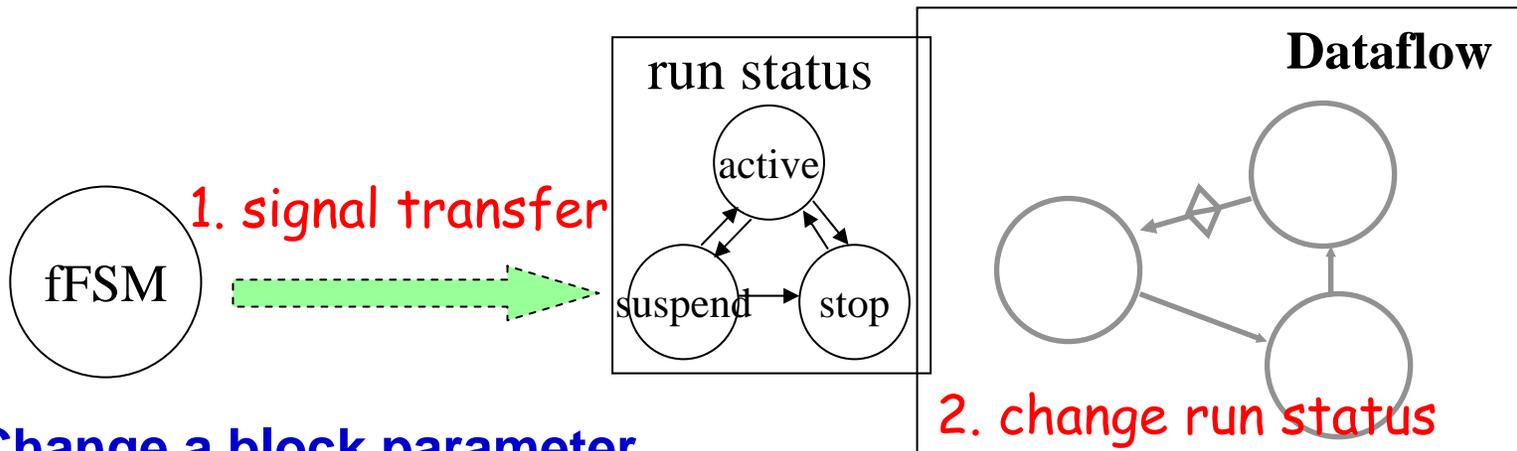
- **BP: Task Model for Codesign backplane: answers to (P4)(P5)**

■ Synchronous interaction

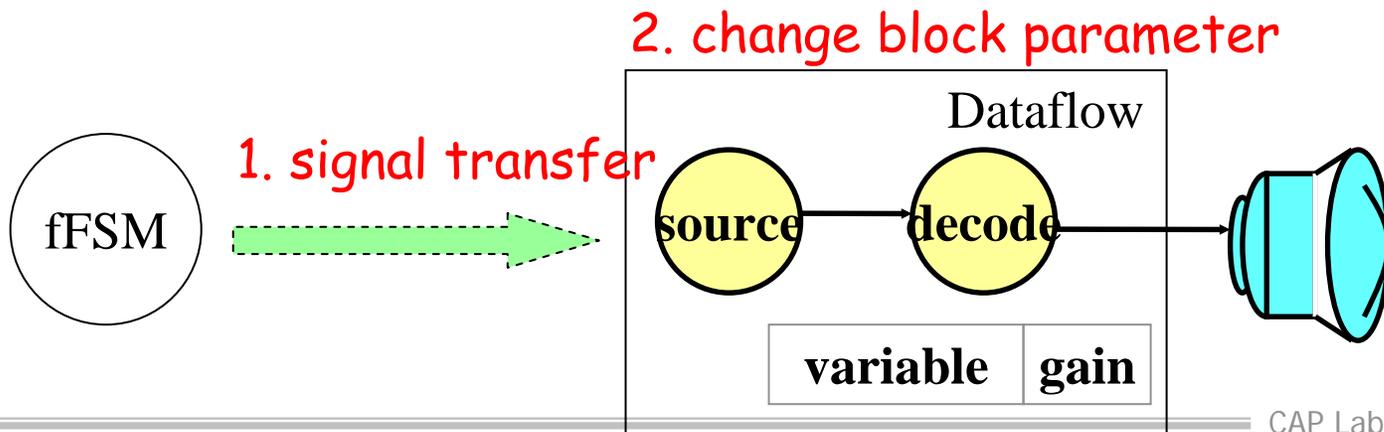


Asynchronous Interaction

- Change the scheduling state of a dataflow module



- Change a block parameter

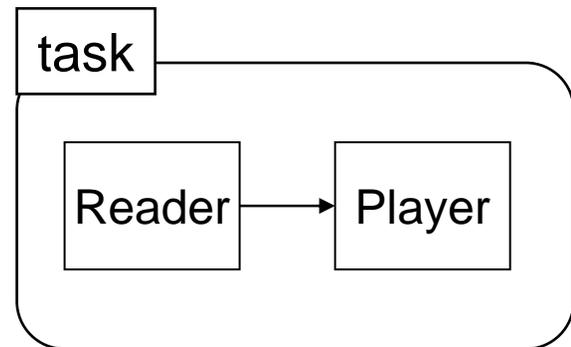
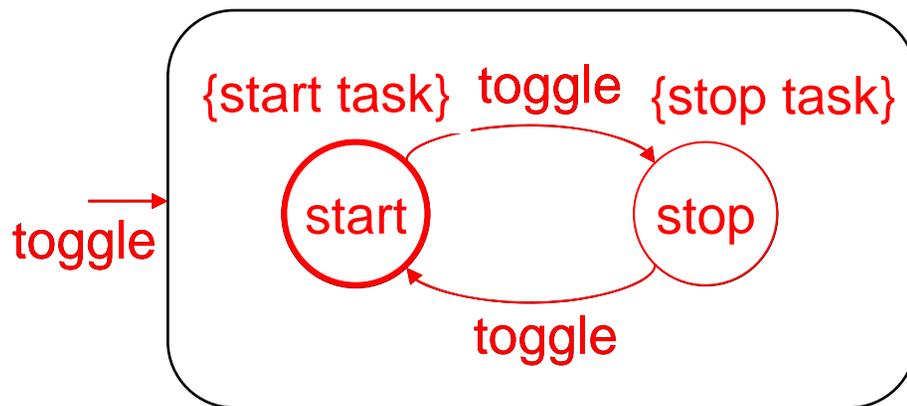


Problem 1: Multi-mode Support

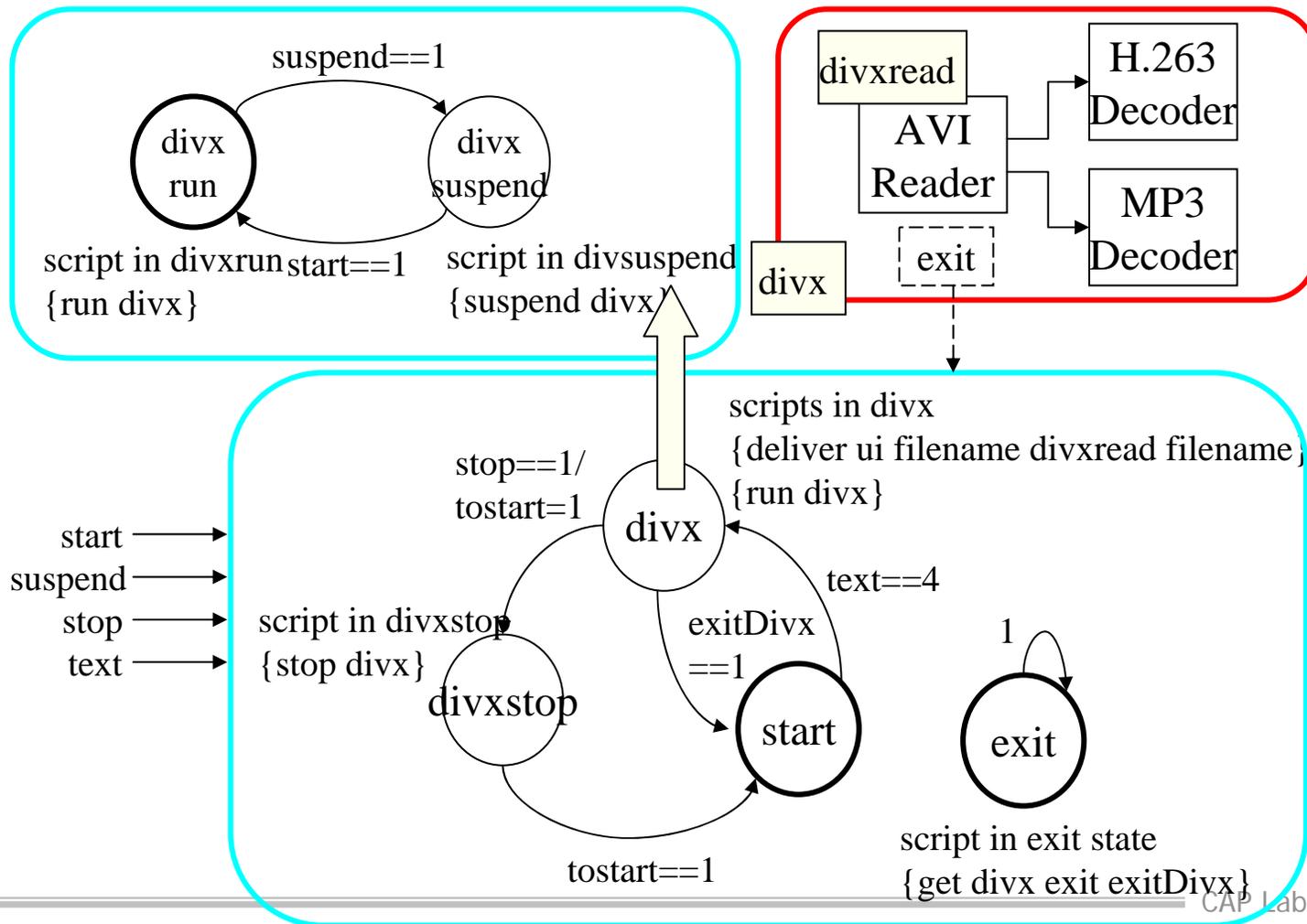
- **Write action scripts in a state of FSM model**

- Change a mode of operation to another
- Deliver initial parameters or control parameters
- Handling exceptions

- **Similar to Statechart**



DIVX Player System Specification

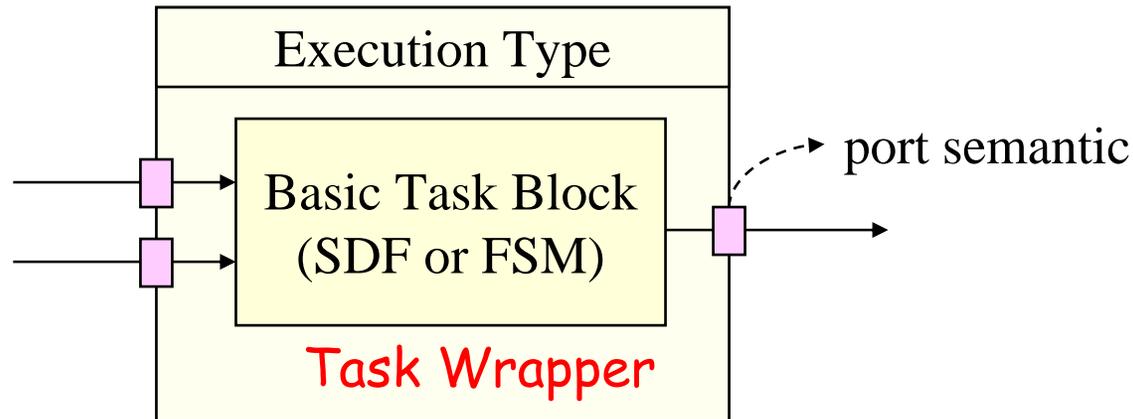


Supported Action Scripts in FSM model

Scripts	Actions
run <i>n_name</i>	Resume the <i>n_name</i> block
suspend <i>n_name</i>	Suspend the <i>n_name</i> block
stop <i>n_name</i>	Stop the <i>n_name</i> block
set <i>n_name parameter value</i>	Update the <i>parameter</i> with <i>value</i> in the <i>n_name</i> block
get <i>n_name parameter n_event</i>	Acquire the <i>parameter</i> in the <i>n_name</i> block to <i>n_event</i> of FSM
deliver <i>n_src src_param n_dst dst_param</i>	Deliver the value of the <i>src_param</i> state in the <i>n_src</i> block to the <i>dst_param</i> state in the the <i>n_dst</i> block

Task-Level Specification Model (P4, P5)

- **Task-level specification model bridges gaps between formality of basic models and flexibility of expression**
- **Definition of task-level specification model**
 - Tasks which are specified by SDF or FSM models
 - Specification for execution types and port semantics at task wrapper



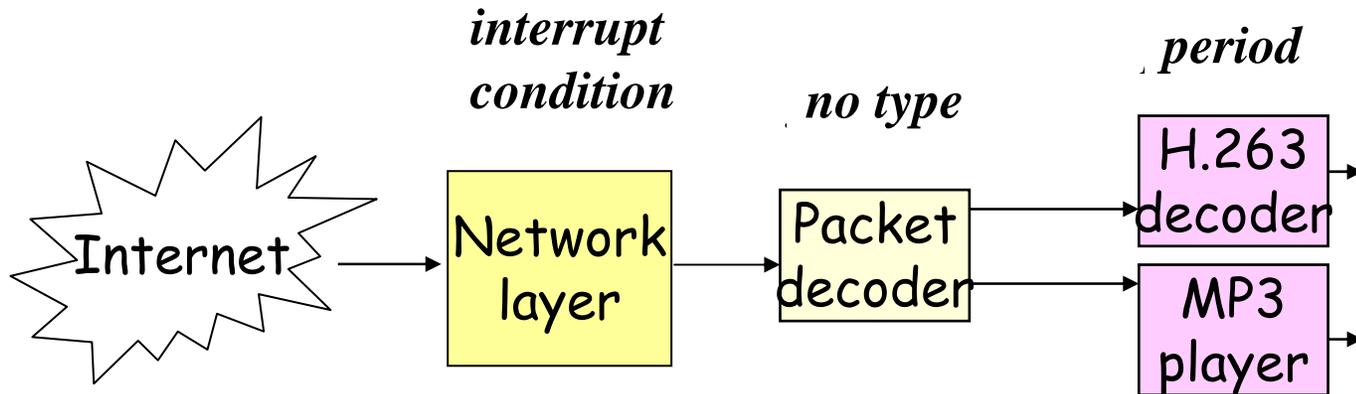
■ Task execution type

- periodic : triggered by time
- sporadic : triggered by external IO
- function : triggered by data

■ Port properties

- port type : queue or buffer
- data size : static or variable
- data rate : static or dynamic

Diverse Task Execution Types



■ Function type

- Basic execution type of SDF and FSM model
- When data is available, it starts an iteration and sends output data

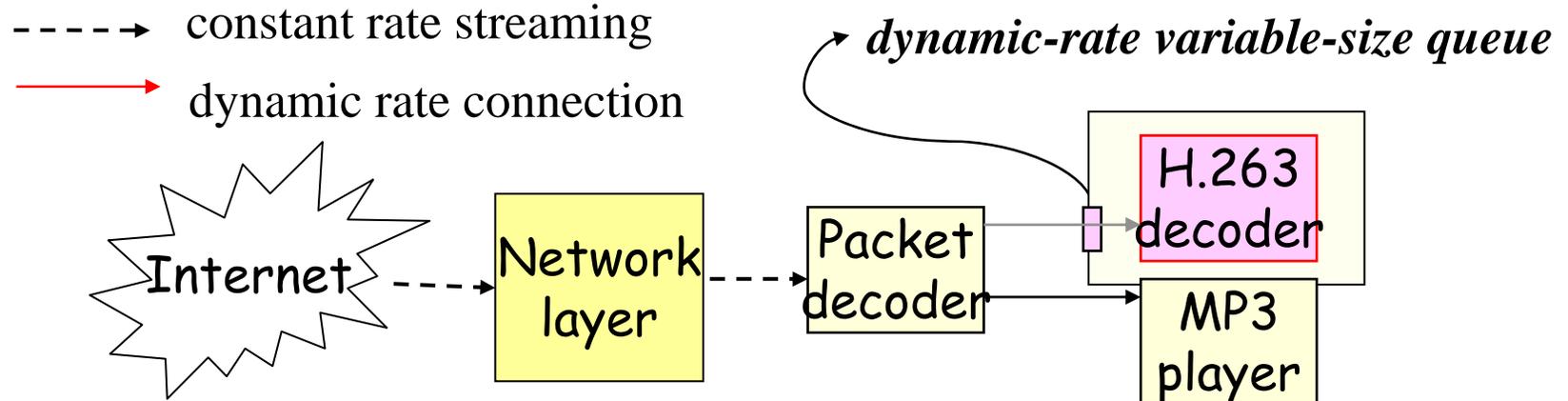
■ Periodic task

- Specify “period” of the task at task wrapper
- After triggered by time period, it becomes active

■ Sporadic task

- Specify “interrupt conditions” of the task at task wrapper
- When the condition is met, it is activated

Dynamic Execution Rate



■ Port properties

- data rate : static or dynamic
- data size : static or variable
- port type : queue or buffer

■ Automatic translation for basic ports in SDF and FSM models

■ Explicit specification for dynamic rate port at task wrapper

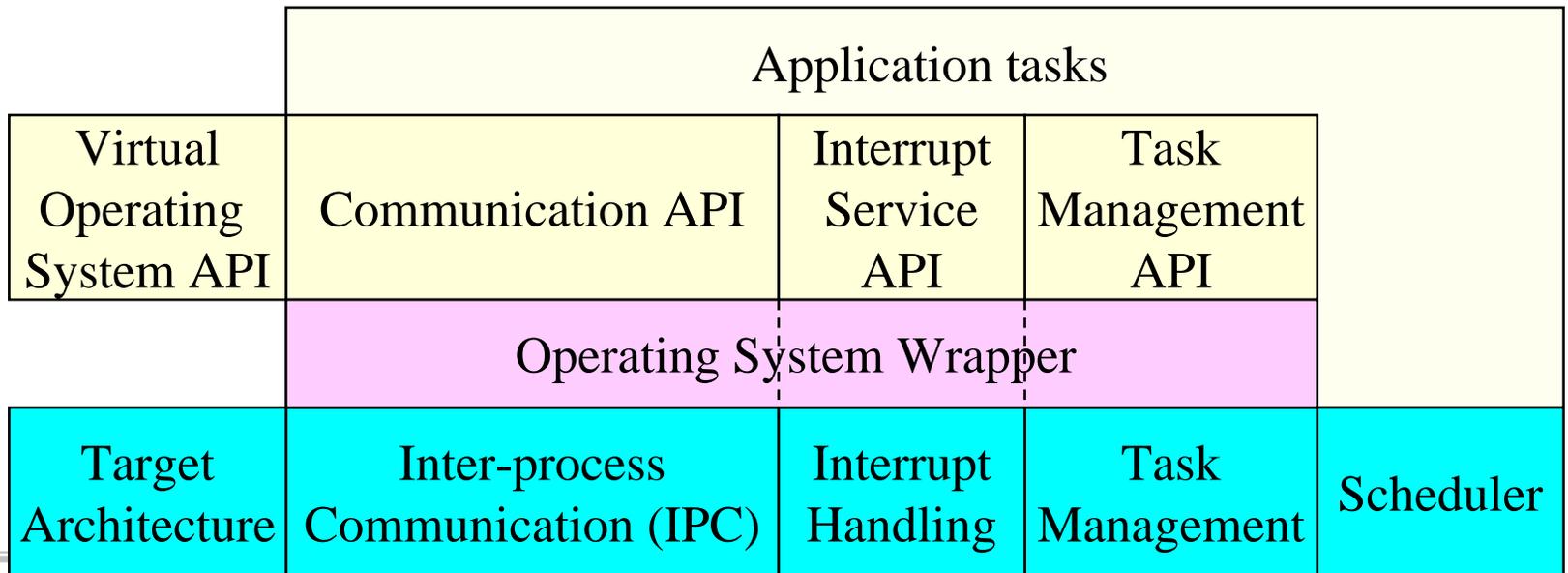
- Creates additional information to handle blocking operations

Automatic Translation for Basic Ports

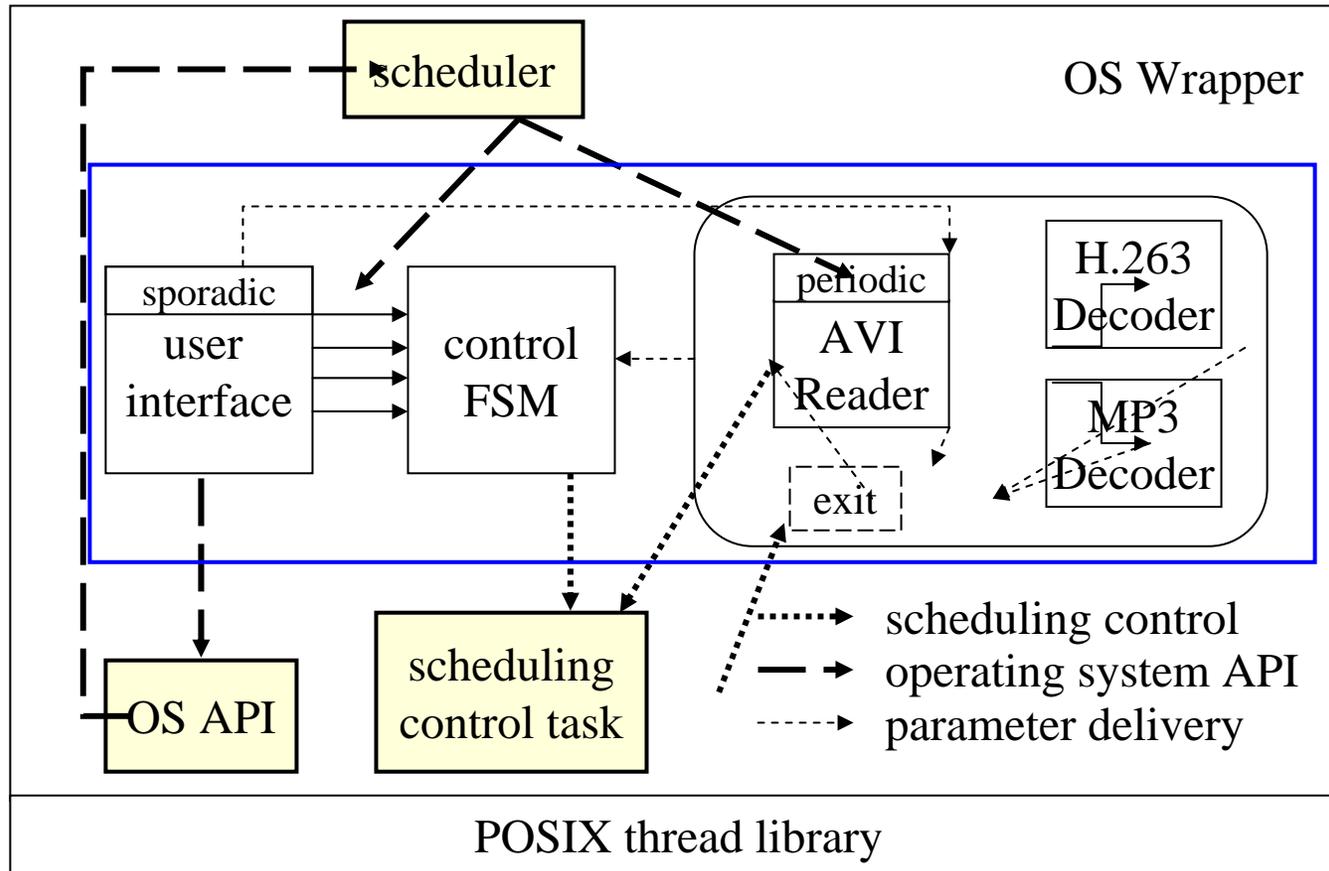
Port type	Port property
Data port of SDF model	Static-rate static-size queue
Data port of FSM model	Dynamic-rate static-size buffer
Scheduling control port	Static-rate static-size queue
Parameter delivery control port	Static-rate static-size buffer
Exception handling control port	Dynamic-rate static-size queue
Control port for periodic task	Static-rate static-size queue
Control port for sporadic task	Static-rate static-size queue

Layered Structure of SW Implementation

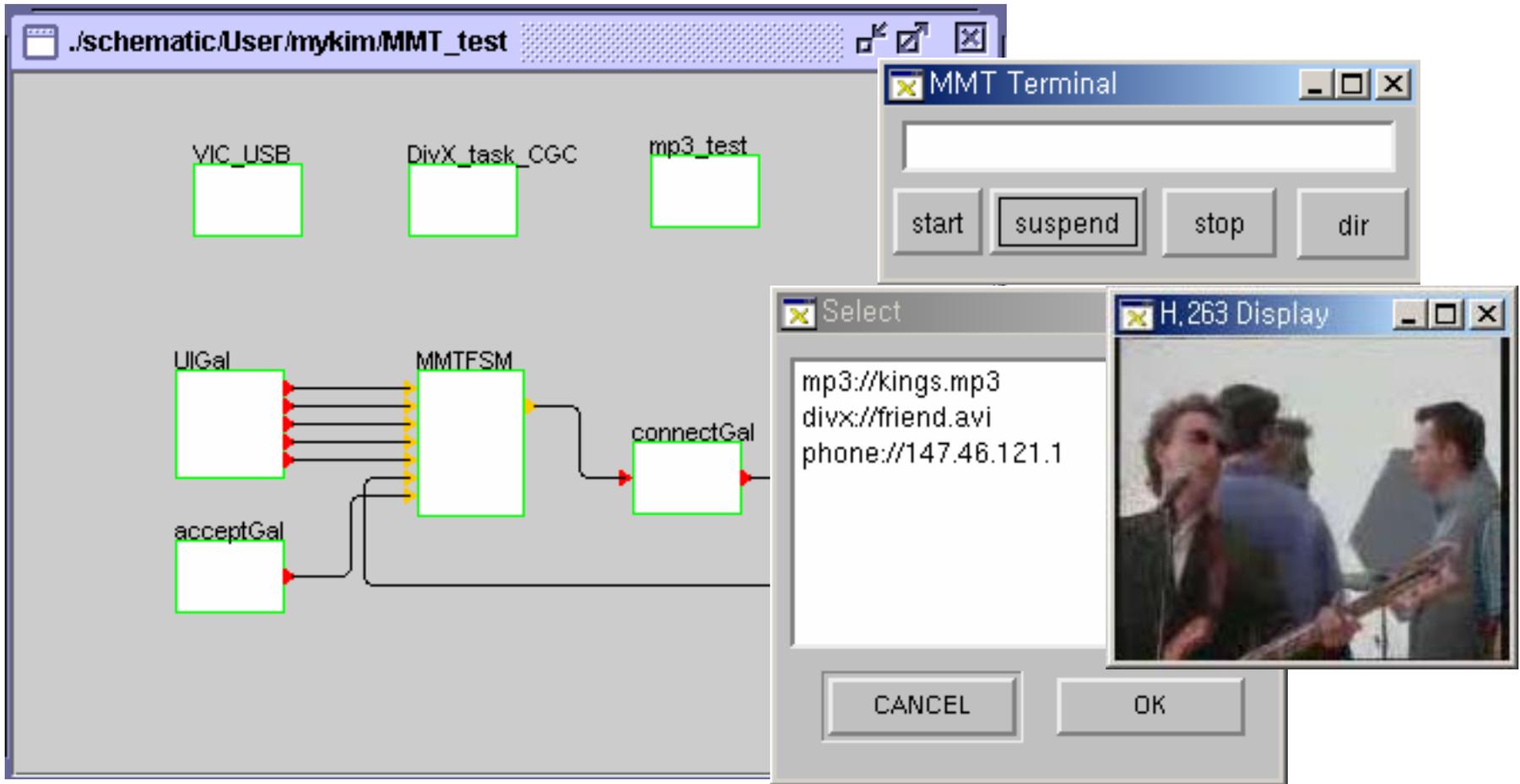
- **Application tasks**
 - Task in task-level specification model → task in operating system
- **Virtual operating system (OS) API**
 - Architecture and design step independent API to application tasks
- **Operating system wrapper**
 - Implementation of virtual OS API on the target architecture



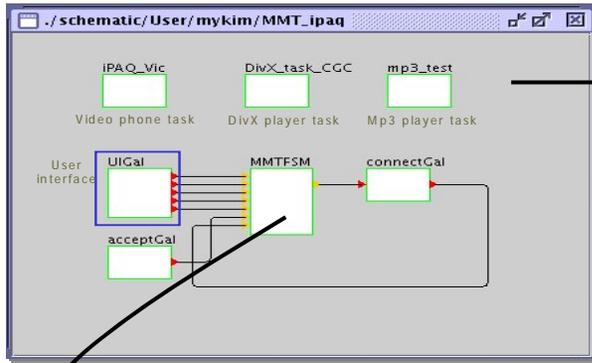
OS Wrapper based on POSIX Thread



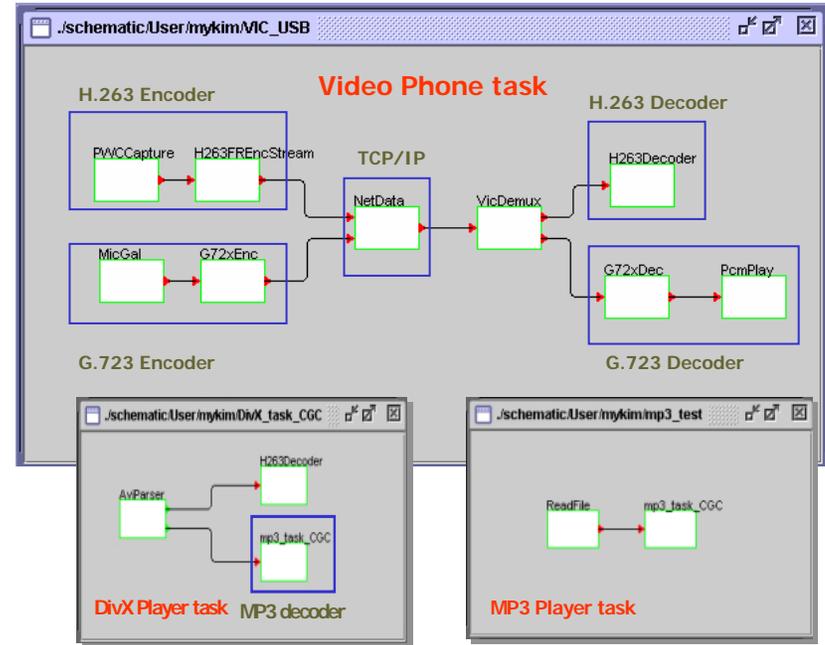
Multi-mode Multimedia Terminal



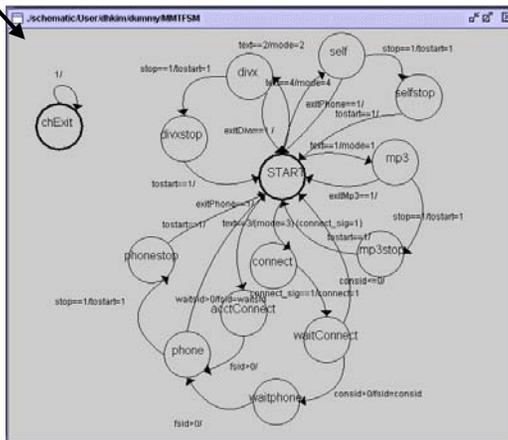
System Level Specification



Top-level schematic



Three mode schematics:
Each mode is a multi-task application.



Mode control FSM

Performance Profile

```

1. MP3
=====
task 3. SELF PHONE
=====
task      execTime (%)          RunCnt  GoCnt  taskName
[00]      0.004170 (0.027024)   418     417    VicDemuxII0
[02]      0.003333 (0.021599)   258     64     PcmPlayII8
[03]      7.054401 (45.715765)  160     160    PWCCaptureII10
[04]      5.051045 (32.733096)  161     160    H263FREncStreamII12
[05]      1.103295 (7.149859)   161     160    H263DecoderII15
[06]      1.171357 (7.590932)   258     0      G72xEncII17
[07]      1.027135 (6.656307)   258     0      G72xDecII20
[08]      0.006772 (0.043886)   319     319    MicGalII23
[09]      0.002135 (0.013836)   161     160    VicPackII25
[10]      0.002912 (0.018871)   258     257    VicPackII28
[11]      0.004402 (0.028527)   418     417    NetDataII31
[17]      0.000000 (0.000000)   0       1      UIGalI3
[18]      0.000000 (0.000000)   0       0      acceptGalI9
[19]      0.000000 (0.000000)   0       0      connectGalI11
[20]      0.000046 (0.000298)   1       1      MMTFSM_TopI14
=====
execTime      15.431003
schedTime     0.361578 (2.289543)
totalTime     15.792581
  
```

Summary

■ Different models are used for different application areas

- State-oriented model: control-intensive reactive system
 - *FSM, statechart, Esterel*
- Activity-oriented model
 - *DSP application: dataflow model*
 - *Timed simulation: Discrete model*
 - *SIMULINK: system simulation*
- Concurrent processes model
 - *concurrent SW development*

■ Heterogeneous Modeling

- Use a mixture of heterogeneous models to specify a complicated system behavior
- Inter-model interface should be well defined.
- Ptolemy II, PeaCE

Questions

- 1. Assume a system that has the following behavior:

input: user control button B. data input Y

output: Z

Y input gets a new event every clock while B input gets irregular input. If B gets an input, control signal X is toggled. In case $X = 0$, Z becomes 1 at every third appearance of ($Y=1$). In case $X = 1$, Z becomes 1 at every second appearance of ($Y=1$). The bottom figure illustrates an example scenario of the system.

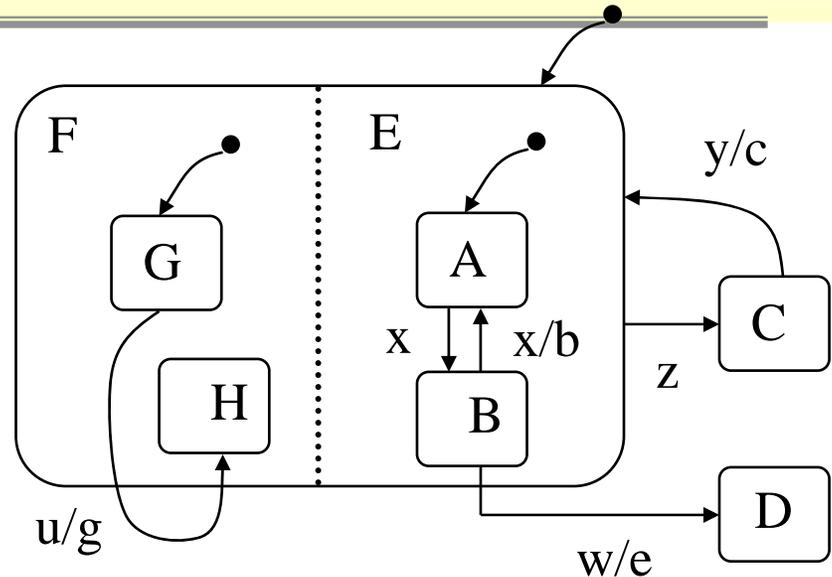
(a) Specify the system behavior using a hierarchical and concurrent FSM.

(b) How many states are required if the FSM obtained in (a) is translated to a flat FSM?

X	0							1						0
Y	0	1	1	0	0	1	1	1	0	1	1	0	1	1
Z	0	0	0	0	0	1	0	0	0	1	0	0	1	0

Continue...

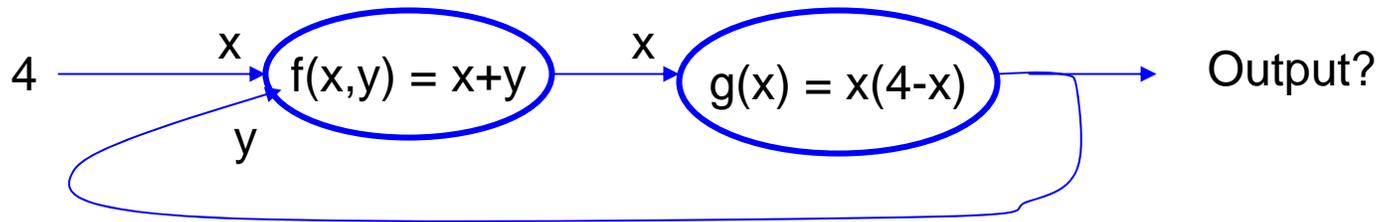
- 2. The right figure shows a hierarchical & concurrent FSM (Statechart). The initial state is designated by an arc from the starting point. That is, the initial states of this FSM are {F(G), E(A)}. F and E are hierarchical states. To avoid ambiguity, we do not allow instantaneous transition except for internal events. High-level transition has priority.



- The system receives the events starting from the initial state. What will be final state of the system. And show the output events during the transition.
Input events: x,u,x,w
- Assume that it is translated to a pure & flat FSM. How many states and arcs will exist in the translated FSM?
- Two arcs between G and H and between B and D violate the compositionality. Translate the FSM to make it compositable by using internal events.

Continue...

- 3. What is the meaning of “synchronous” in Synchronous/Reactive Model? What will be the output of the following SR graph?



- 4. Explain the limitation of discrete event model.
- 5. Simulink has “continuous-time semantics“. Explain its meaning with a simple example.
- 6. Discuss the similarity and difference between PeaCE model and STATEMATE model.
- 7. Kahn process network can not accept asynchronous input. What does the YAPI solve this limitation?
- 8 Discuss two interaction mechanism between fFSM and SDF in PeaCE