PTIDES: A **P**rogramming Model for **Ti**me-Synchronized **D**istributed R**e**al-time **S**ystems

by Yang Zhao[1], Jie Liu[2], Edward A. Lee[1]

[1]EECS, UC Berkeley

[2]Microsoft Research

13th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS 07)

presented by Patricia Derler

Compositionality Seminar Winter 2007/2008, Department of Computer Sciences, University of Salzburg

Motivation

- Implement distributed real-time systems
  - Deterministic
  - Lightweight
- by extending a modeling technique that has analyzable deterministic behavior
  - Discrete Event Modeling
- to enable static analysis of distributed real-time systems

Distributed real-time systems

Definition:

- **Multiple** computers connected on a network
- Computers **interact** with physical world through
  - Sensors
  - Actuators
  - and human computer interfaces
- Interact **with**/interact **through** the physical world
  - Passage of time becomes a central feature

- Applications: manufacturing, instrumentation, surveillance, multi-vehicle control, avionics systems, automotive systems, scientific experiments, …
- Require high precision

Distributed real-time systems

- Modeling distributed/embedded systems
  - OO programming using frameworks such as CORBA, SOAP, DCOM, …
    - Heavyweight
    - Non-deterministic
  - Synchronous languages such as Esterel, Scade, Lustre, …
    - Tight coordination difficult for distributed systems
  - Time-triggered languages and the logical execution time
    - Not only periodic tasks

Discrete Event Modeling

Event:

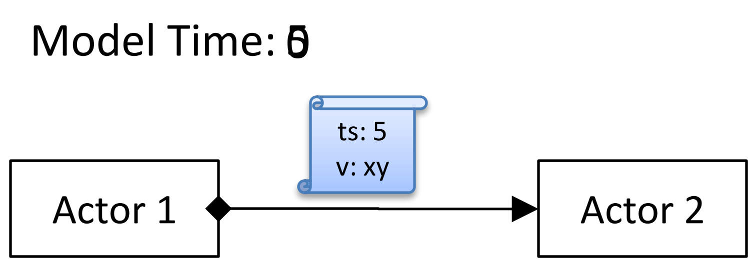
- Time stamp: when should an action be taken
- Value: kind of action

Signal:

Sequence of events in chronological order

Time:

Common notion of time in the whole system



Discrete Event Modeling

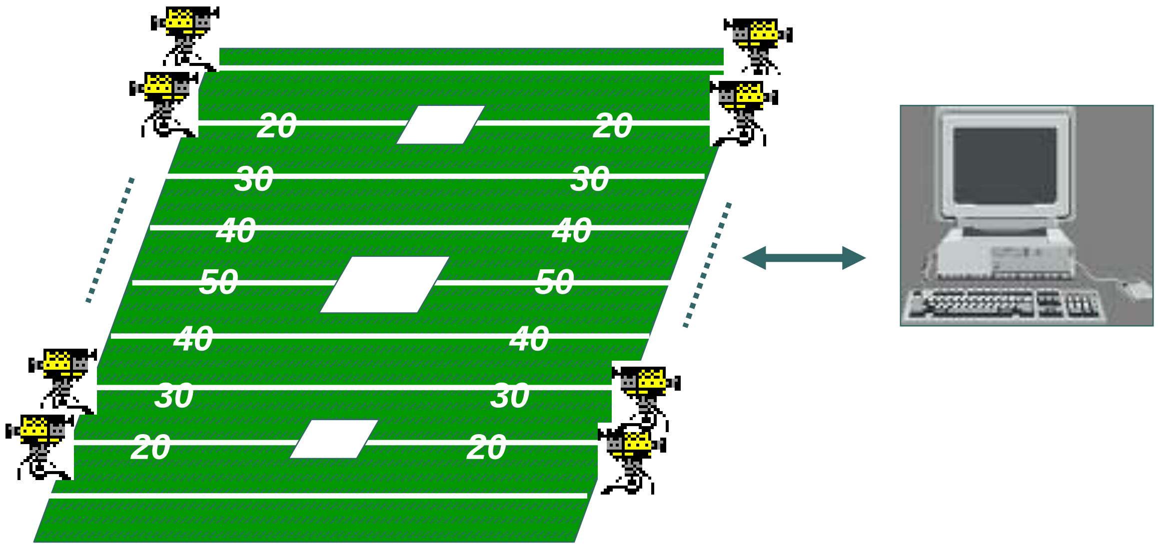
- Concurrent compositions of components that interact via *events*
  = time-stamped value, where time is "logical time" or "modeling time"
- Correct execution by ordering according to time stamps
- Analyzable deterministic behavior
- Simulation technology
  1) Conservative technique: total ordering of events
     waste of resources, does not meet realistic real-time constraints
  2) Optimistic technique: perform speculative execution and backtrack if necessary
     cannot backtrack physical interactions

PTIDES:

- Application specification language
- Global common notion of time through network time synchronization
- Execution strategy:
  – Global coordination layer: can event be processed immediately or wait
  – Local resource scheduling layer: e.g. EDF
- For certain events, modeling time is mapped to physical time
  conservative approach, but looser coupling than with (1)

- Partial order on events
  = relevant order
- Release events out of their time stamp order

Example: Camera application



N cameras (= sensor and actuator)

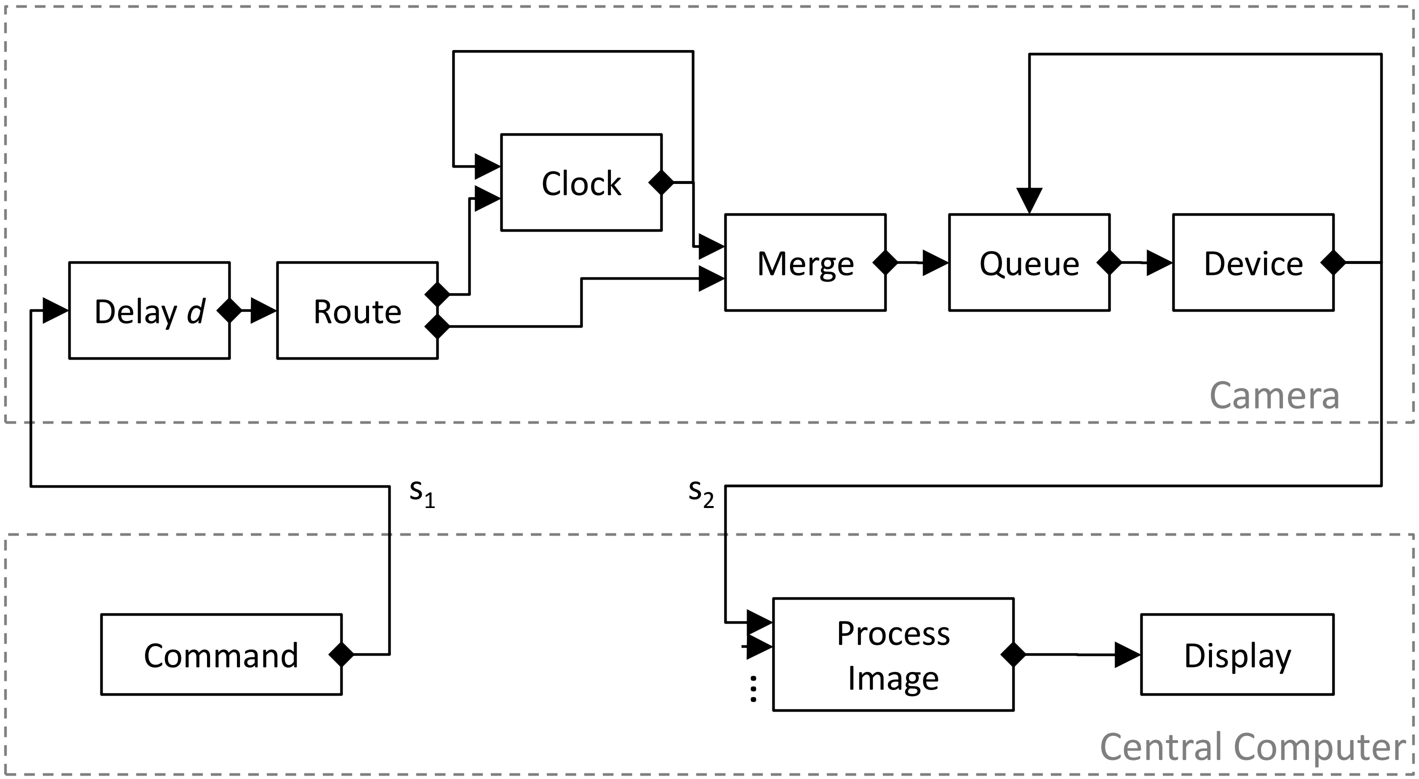
- synchronized clocks
- connected via Ethernet
- partial view of the field
- (simultaneous) take picture (time-stamped), zoom

1 central computer

- controls taking picture, zooming
- process pictures

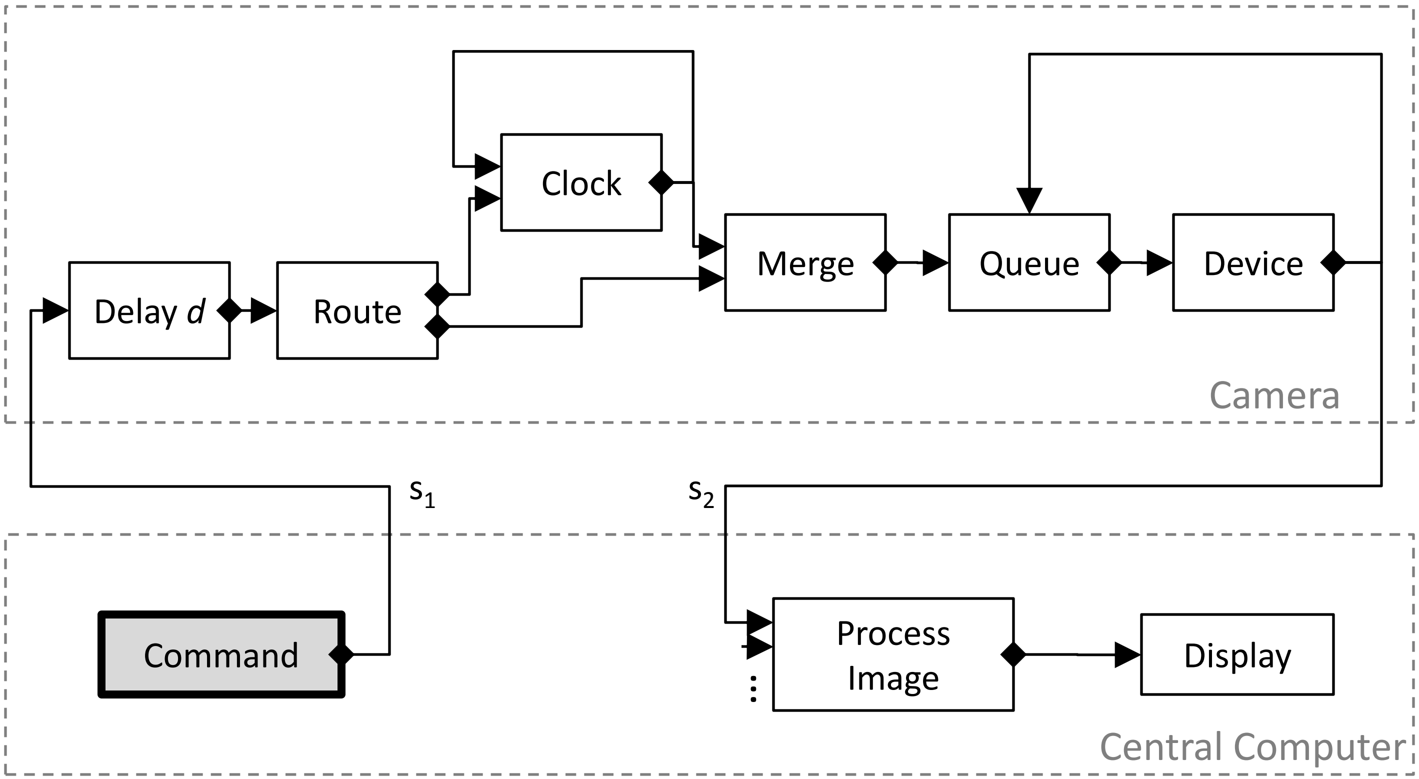
Challenge: coordinate cameras to get precisely synchronized images

Specification of the networked camera application



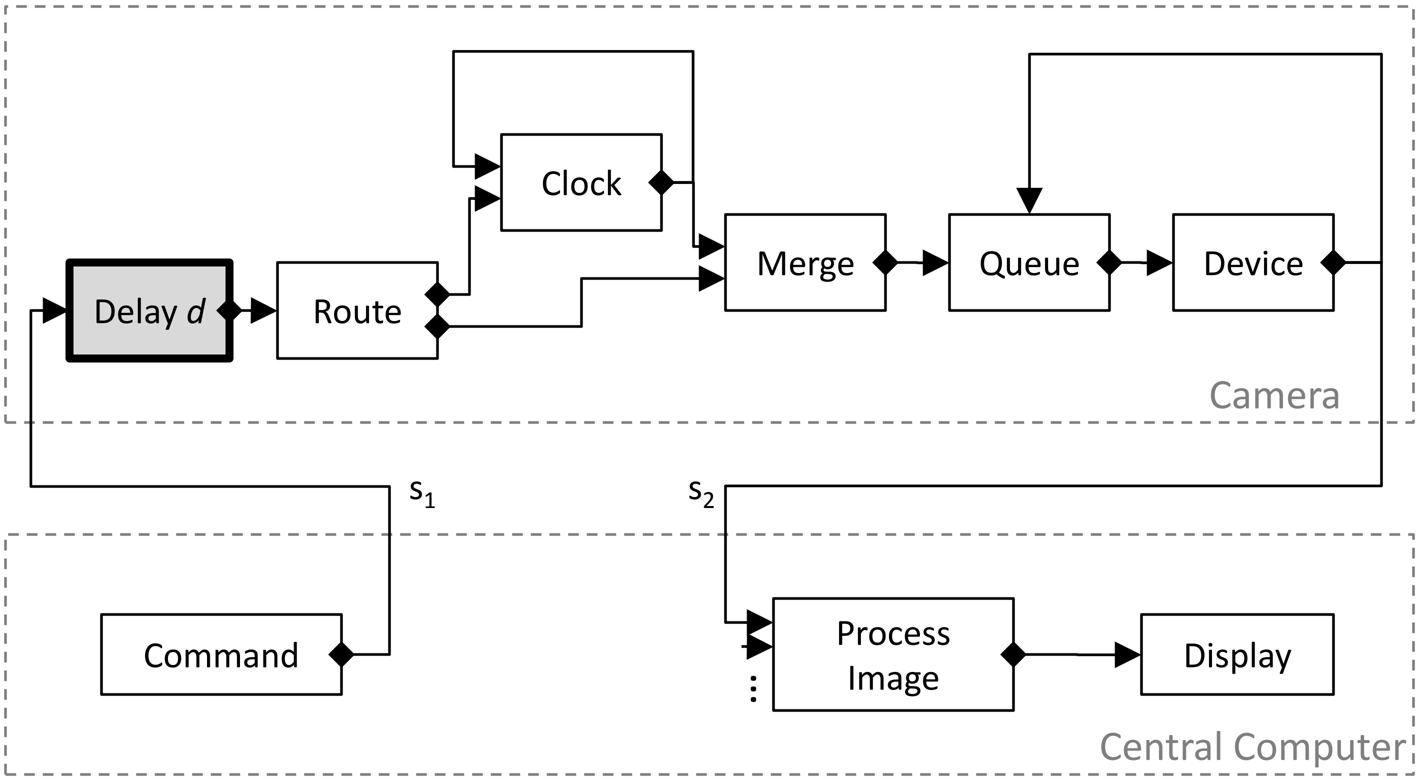
only communication via ports

Specification of the networked camera application



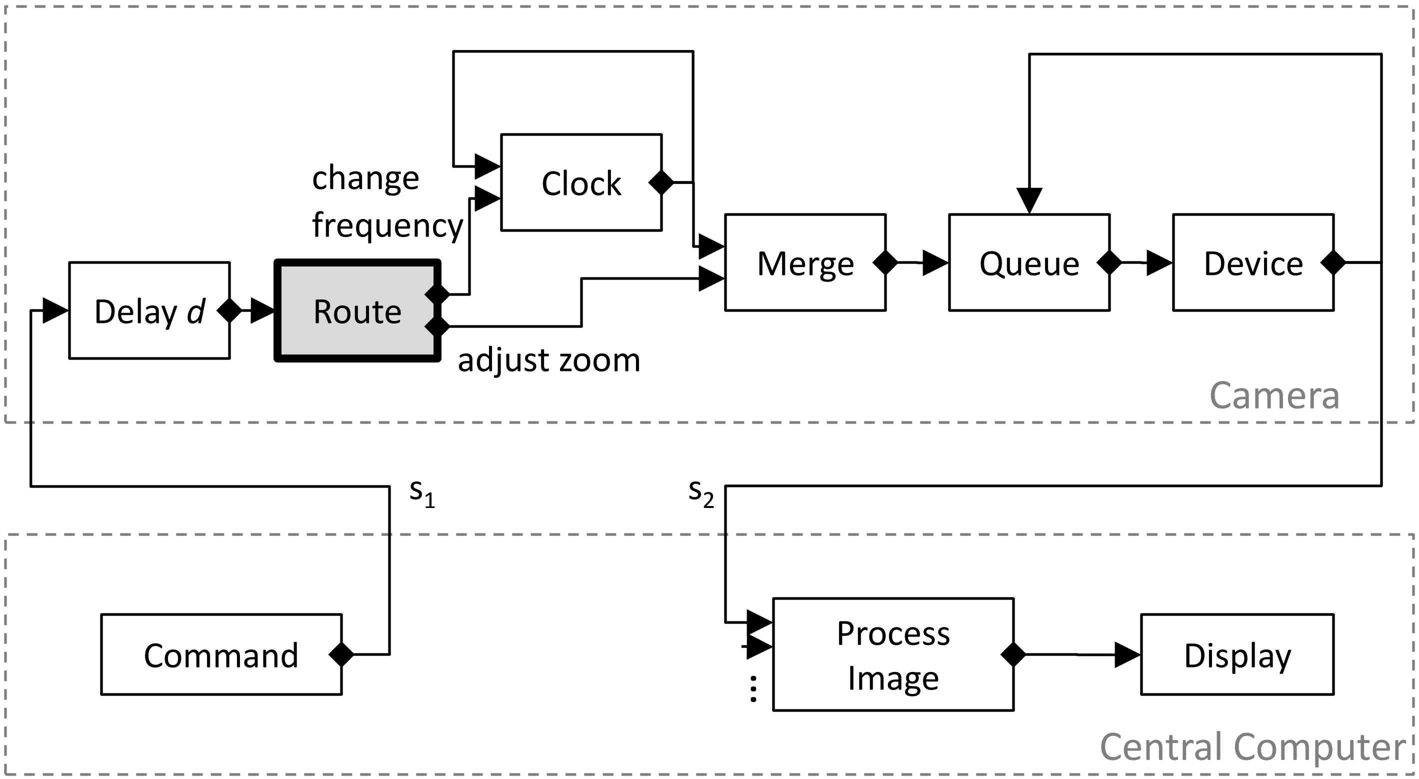
- Wraps interaction with UI device
- Sends event with current time stamp for each user input to all cameras

Specification of the networked camera application



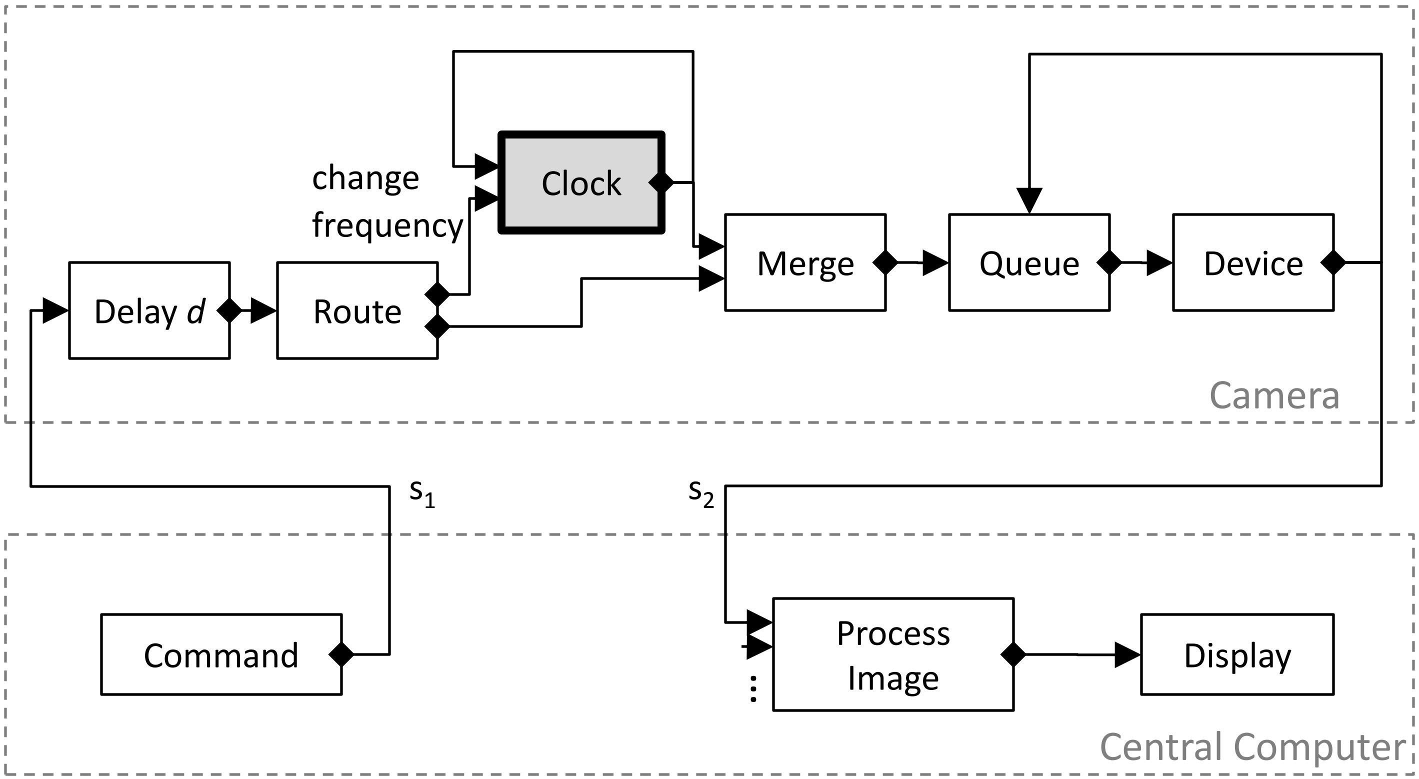
Produce event with timestamp $t + d$ for event with time stamp t

Specification of the networked camera application



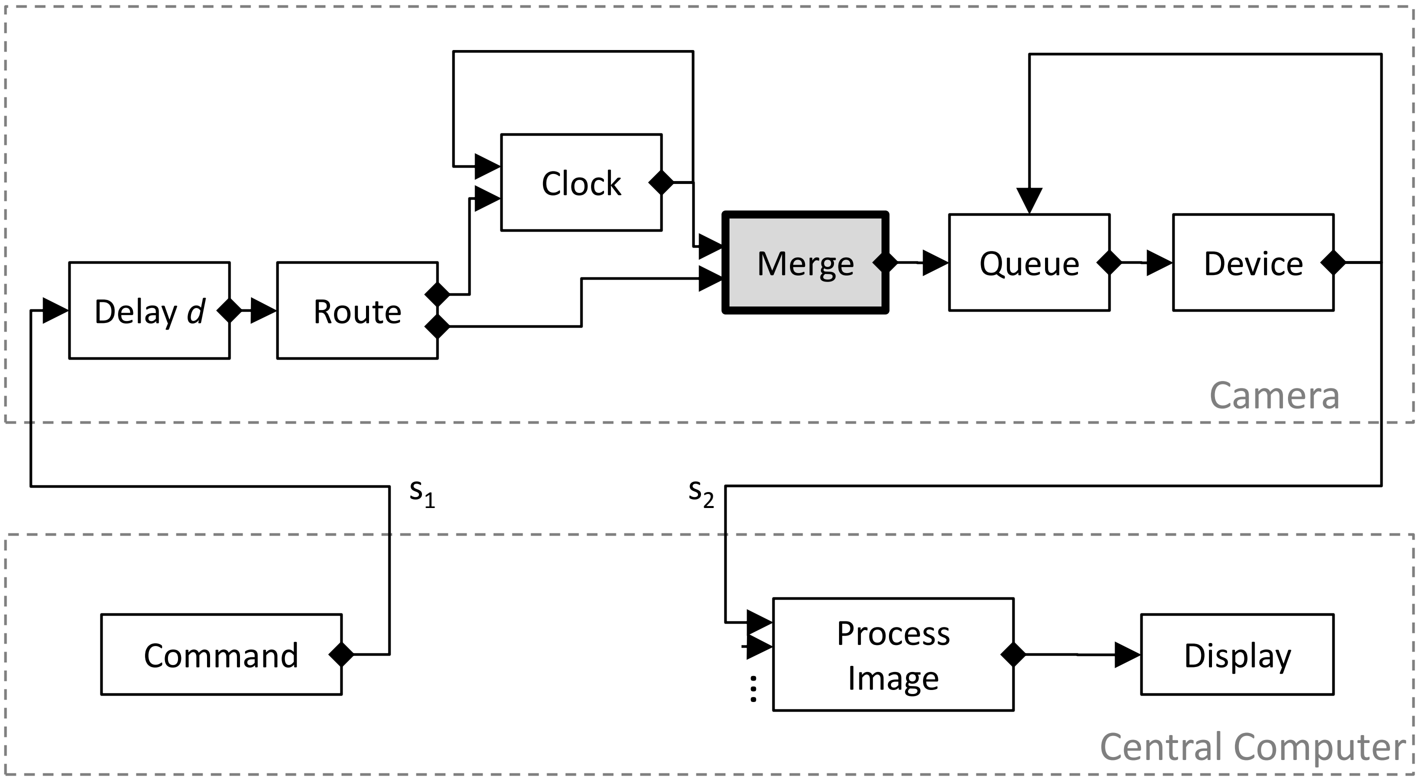
Separates commands

Specification of the networked camera application



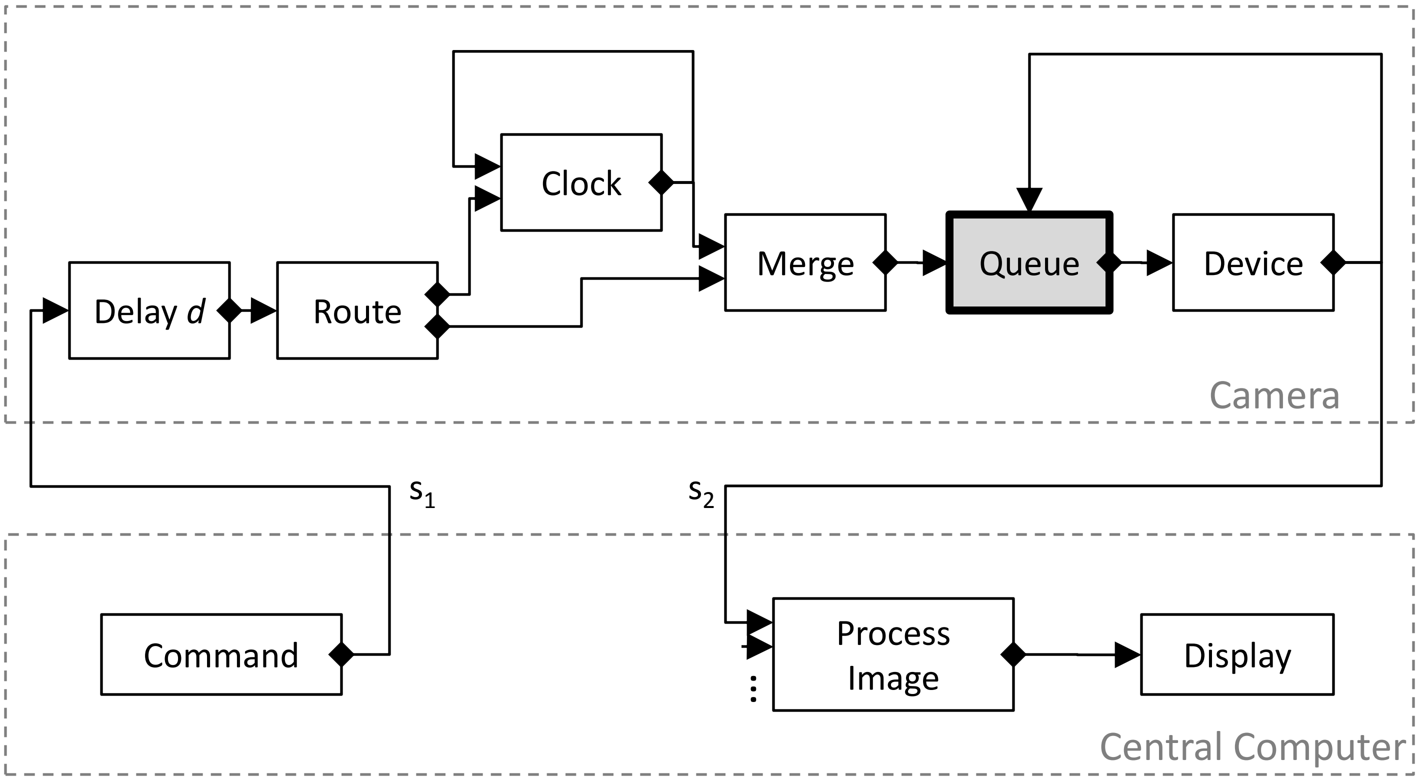
Control picture taking of camera by producing time-stamped outputs
Change frequency on user request

Specification of the networked camera application



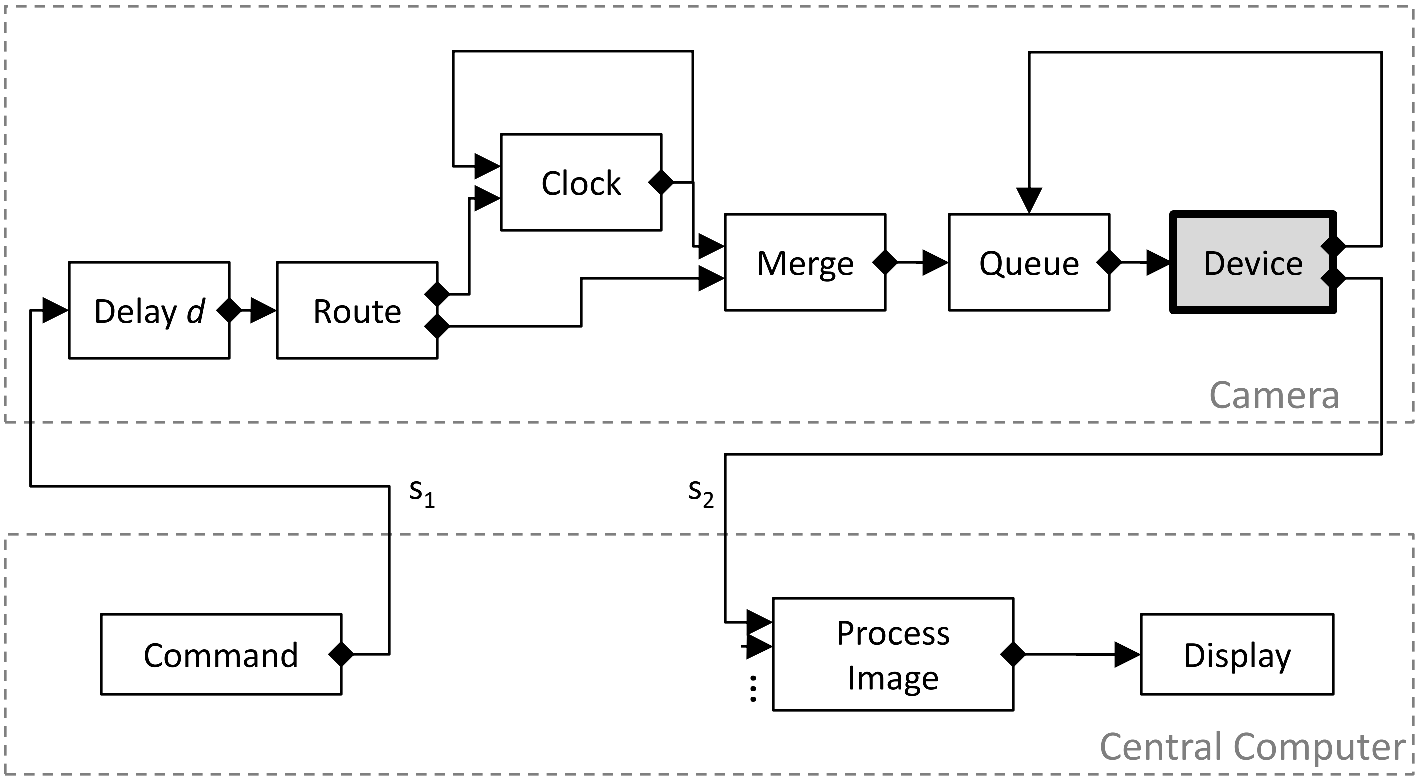
Merge events with priority for second input (to give user control)

Specification of the networked camera application



Buffers events until device is ready

Specification of the networked camera application

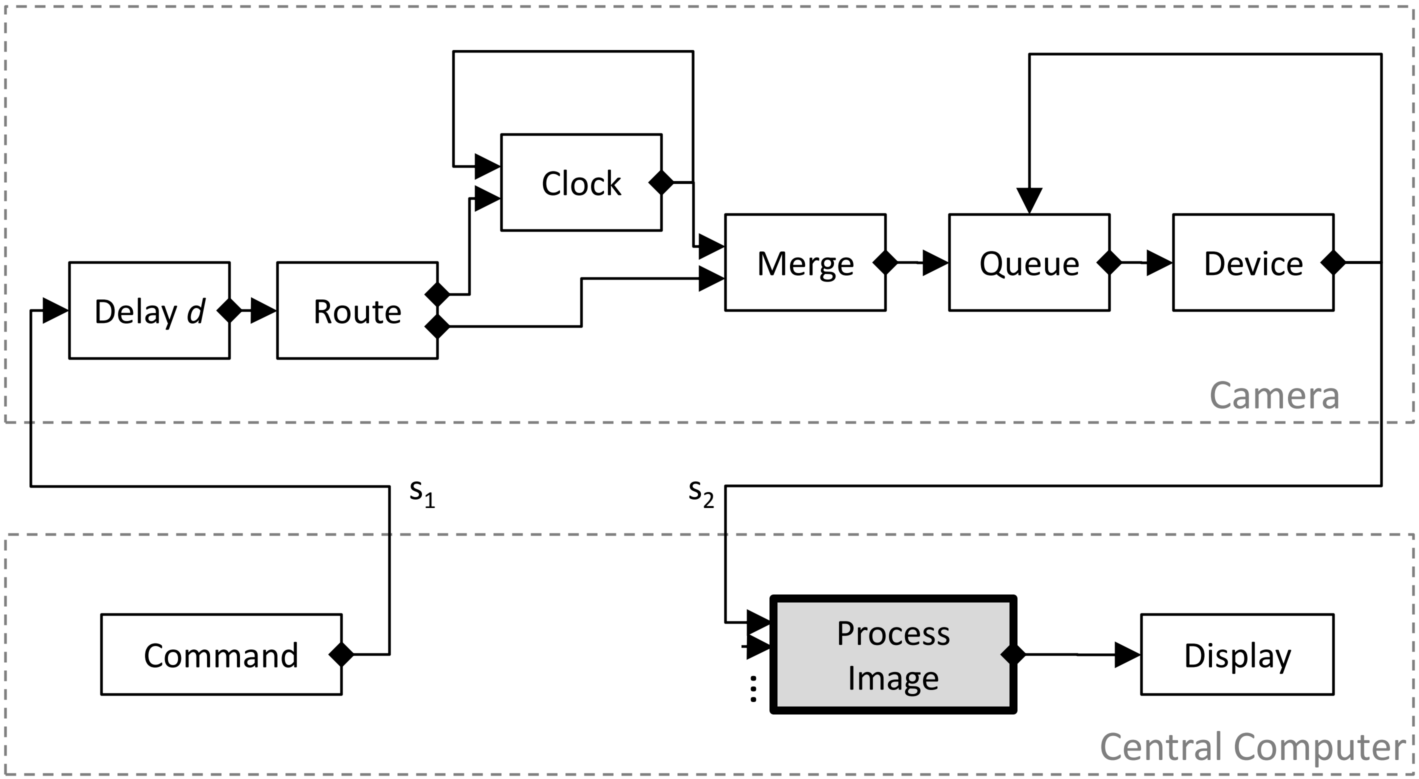


Wraps interaction with camera driver

1st output: value for each input event, time stamp > t (indicate physical action complete)

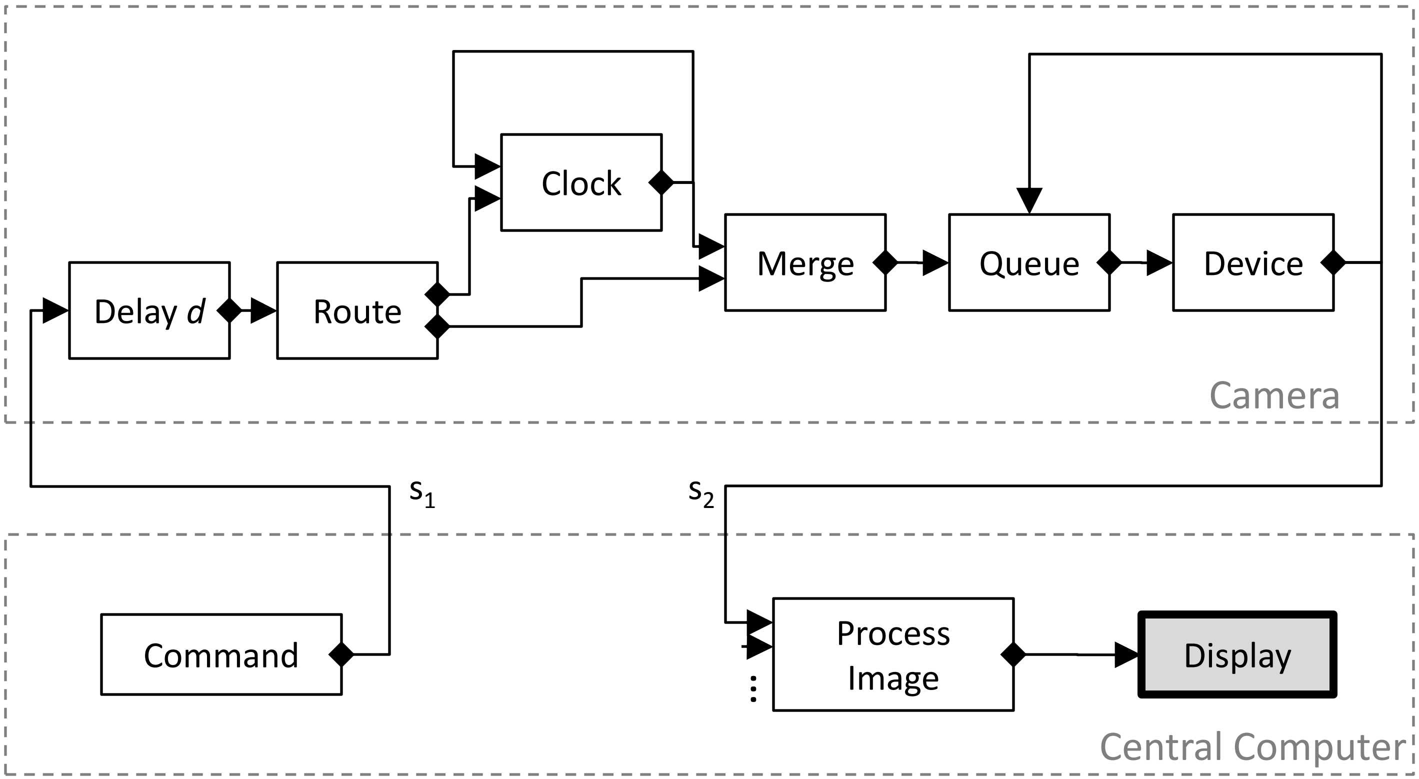
2nd output: time-stamped image

Specification of the networked camera application



.

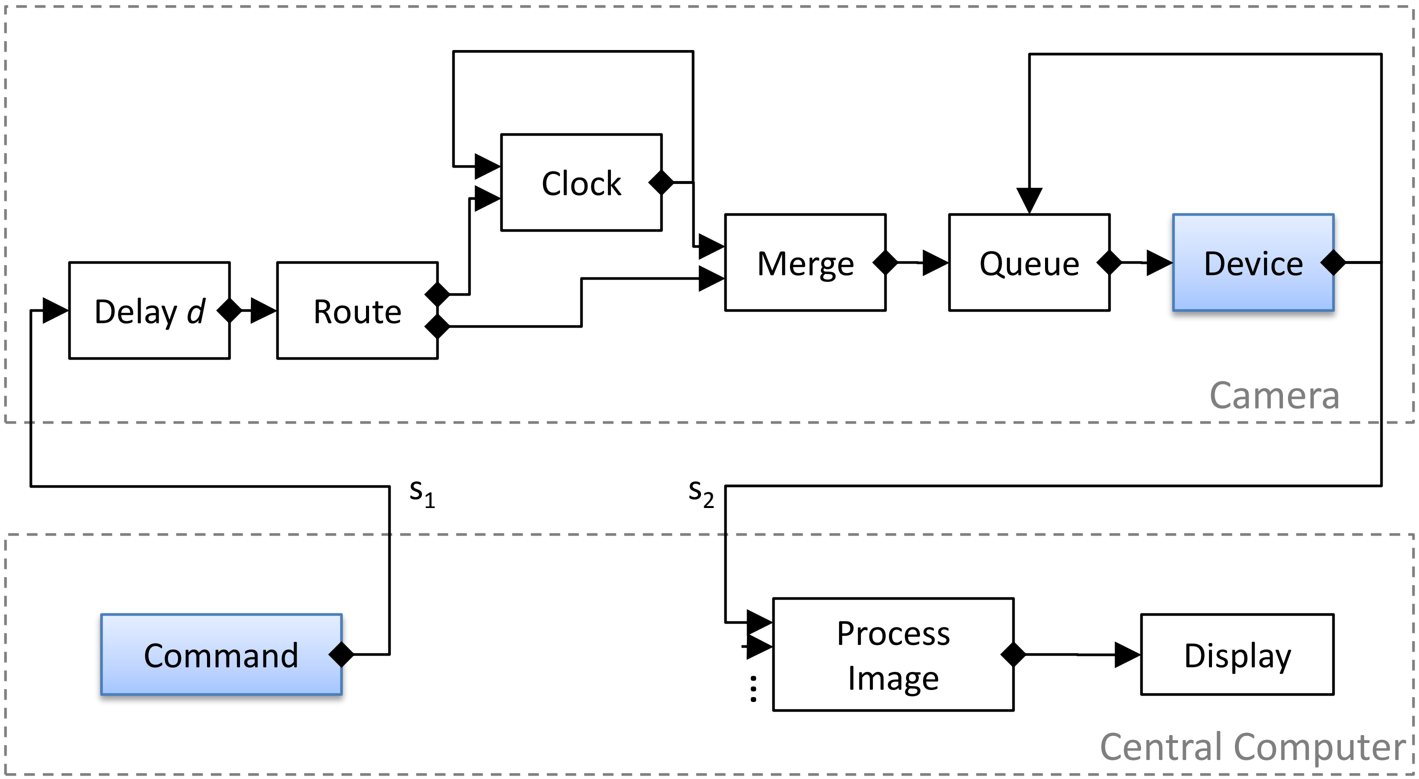
Specification of the networked camera application



•

Specification of the networked camera application

React to an **input** event with time stamp t if physical time $\tau \leq t$
Produce an **output** event with time stamp t' at physical time $\tau' \geq t'$



setup time σ for real-time ports, react if $\tau \leq t - \sigma$

Challenge

- First-come-first-serve strategy cannot be used
  - no deterministic DE semantics since network may alter order of events
- Brute-force implementation of a conservative technique might stall execution in the camera

- Out of order release of events
  - without loosing determinism
  - without requiring backtracking
- Only process events in time-stamp order when they are causally related

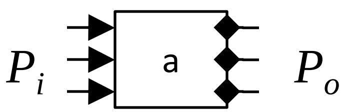
- PTIDES (Programming Time-Integrated Distributed Embedded Systems):
  - Discrete-event semantics
  - Carefully chosen relations between model time and real time
  - Determinacy is preserved at runtime.
  - Does not depend on domain specific network architectures

Causality Interface

Causality relations among events:

- dependency that output events have on input events
- statically analyzed

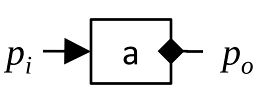
Causality Interface:



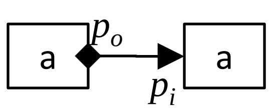
$$\delta_a : P_i \times P_o \longrightarrow D$$

- D is an ordered set (algebra) with two binary operations for composition
  - $\oplus$(=parallel; minimum)
  - $\otimes$(= serial; addition)
- Elements of D are called **dependencies** $\delta_a(p_1, \ p_2)$ ... dependency that port p_2 has on p_1
- minimum model-time delay between input events at p_1 and resulting output events at p_2
- Create dependency graph

$\delta(p_i , p_o) < \infty$



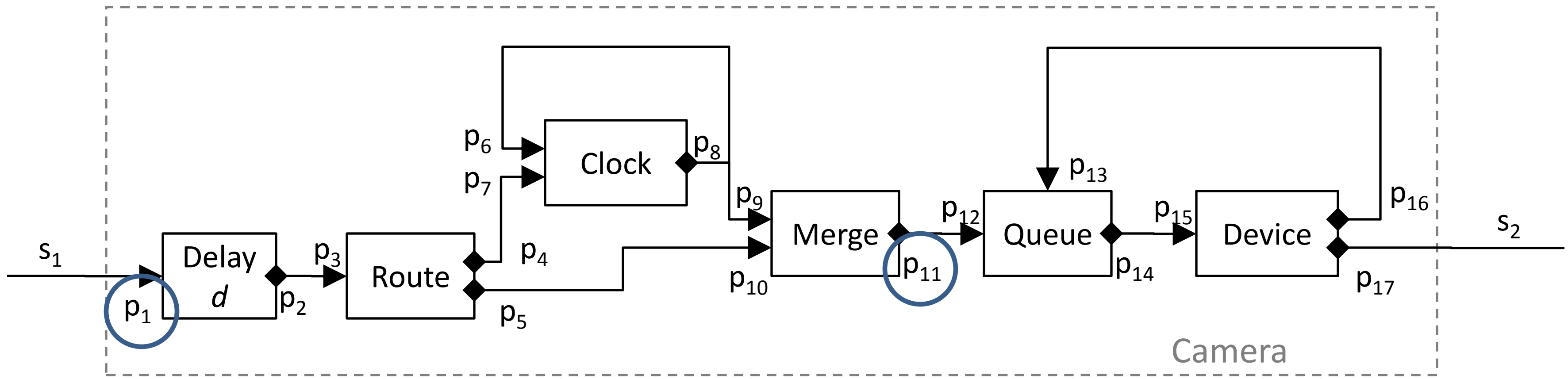
$\delta(p_o , p_i) = 0$



$else: \delta \ (p_1, p_2)= \infty$

p_1 ??? p_2

Dependencies



Composition via ⊗ and ⊕

⊗ … serial, addition

⊕ … parallel, min

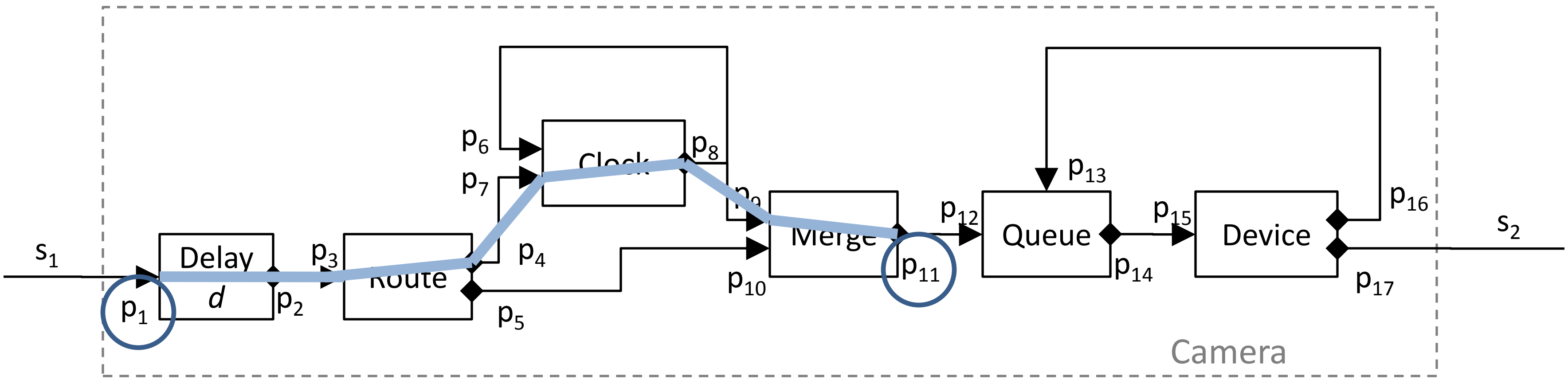
$$\delta(p_1, p_{11}) = \min(ph_1, ph_2, ph_3) \text{ where}$$

$$ph_1 = \delta_{Delay}(p_1,p_2) + 0 + \delta_{Router}(p_3,p_4) + 0 + \delta_{Clock}(p_7,p_8) + 0 + \delta_{Merge}(p_9,p_{11}),$$

$$ph_2 = \delta_{Delay}(p_1,p_2) + 0 + \delta_{Router}(p_3,p_5) + 0 + \delta_{Merge}(p_{10},p_{11}),$$

$$ph_3 = \delta_{Delay}(p_1,p_2) + 0 + \delta_{Router}(p_3,p_4) + 0 + \delta_{Clock}(p_7,p_8) + 0 + \delta_{Clock}(p_6,p_8) + 0 + \delta_{Merge}(p_9,p_{11})$$

Dependencies



Composition via $\otimes$ and $\oplus$

- $\otimes$ … serial, addition
- $\oplus$ … parallel, min

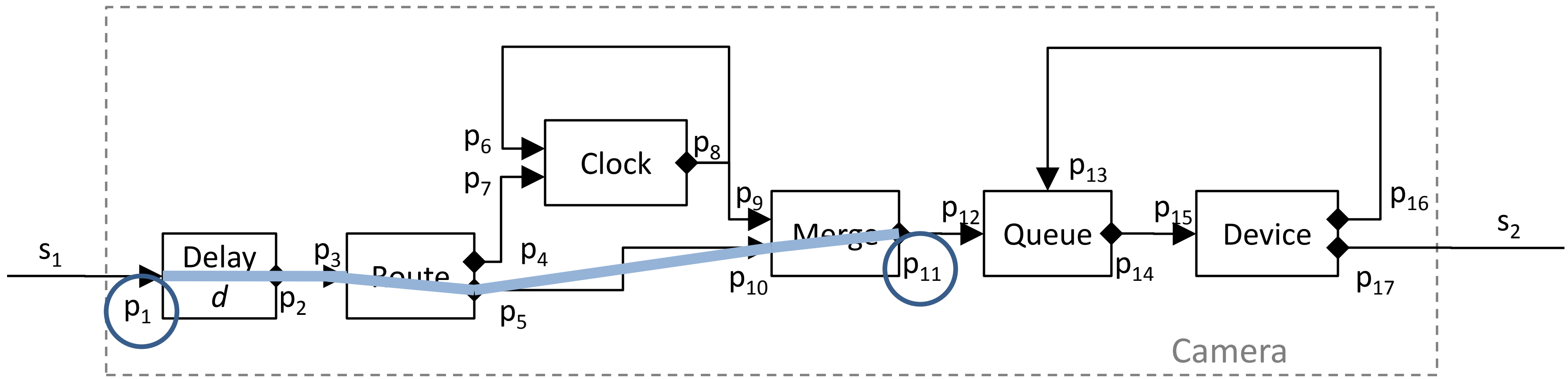
$\delta(p_1, p_{11}) = min(ph_1, ph_2, ph_3)$ *where*

$$ph_1 = \delta_{Delay}(p_1,p_2) + 0 + \delta_{Router}(p_3,p_4) + 0 + \delta_{Clock}(p_7,p_8) + 0 + \delta_{Merge}(p_9,p_{11}),$$

$$ph_2 = \delta_{Delay}(p_1,p_2) + 0 + \delta_{Router}(p_3,p_5) + 0 + \delta_{Merge}(p_{10},p_{11}),$$

$$ph_3 = \delta_{Delay}(p_1,p_2) + 0 + \delta_{Router}(p_3,p_4) + 0 + \delta_{Clock}(p_7,p_8) + 0 + \delta_{Clock}(p_6,p_8) + 0 + \delta_{Merge}(p_9,p_{11})$$

Dependencies



Composition via ⊗ and ⊕

- ⊗ … serial, addition
- ⊕ … parallel, min

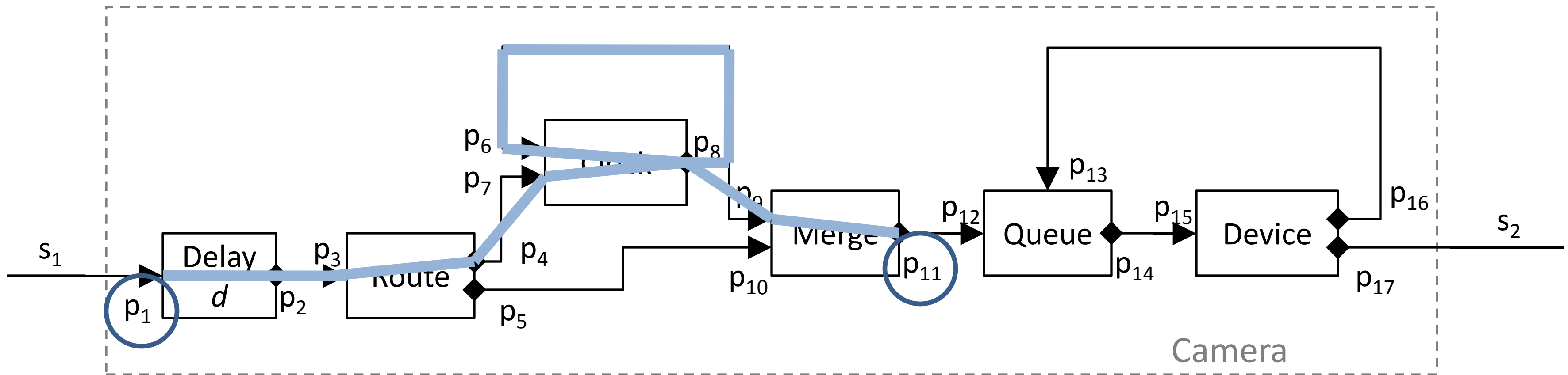
$\delta(p_1, p_{11}) = min(ph_1, ph_2, ph_3)$ *where*

$$ph_1 = \delta_{Delay}(p_1,p_2) + 0 + \delta_{Router}(p_3,p_4) + 0 + \delta_{Clock}(p_7,p_8) + 0 + \delta_{Merge}(p_9,p_{11}),$$

$$ph_2 = \delta_{Delay}(p_1,p_2) + 0 + \delta_{Router}(p_3,p_5) + 0 + \delta_{Merge}(p_{10},p_{11}),$$

$$ph_3 = \delta_{Delay}(p_1,p_2) + 0 + \delta_{Router}(p_3,p_4) + 0 + \delta_{Clock}(p_7,p_8) + 0 + \delta_{Clock}(p_6,p_8) + 0 + \delta_{Merge}(p_9,p_{11})$$

Dependencies



Composition via ⊗ and ⊕

⊗ … serial, addition

⊕ … parallel, min

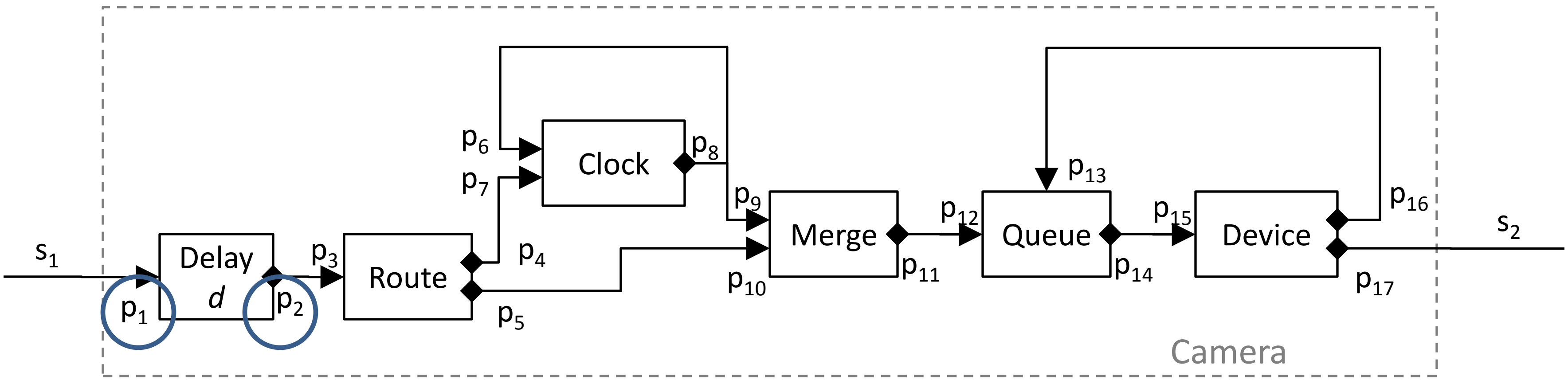
$\delta(p_1, p_{11}) = min(ph_1, ph_2, ph_3)$ *where*

$$ph_1 = \delta_{Delay}(p_1,p_2) + 0 + \delta_{Router}(p_3,p_4) + 0 + \delta_{Clock}(p_7,p_8) + 0 + \delta_{Merge}(p_9,p_{11}),$$

$$ph_2 = \delta_{Delay}(p_1,p_2) + 0 + \delta_{Router}(p_3,p_5) + 0 + \delta_{Merge}(p_{10},p_{11}),$$

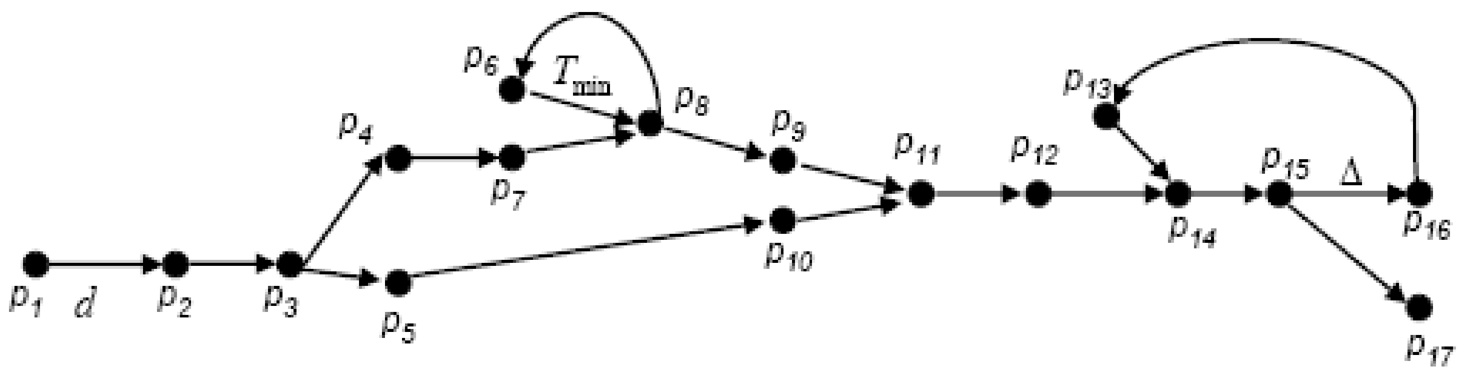
$$ph_3 = \delta_{Delay}(p_1,p_2) + 0 + \delta_{Router}(p_3,p_4) + 0 + \delta_{Clock}(p_7,p_8) + 0 + \delta_{Clock}(p_6,p_8) + 0 + \delta_{Merge}(p_9,p_{11})$$

Dependencies



Dependency graph:

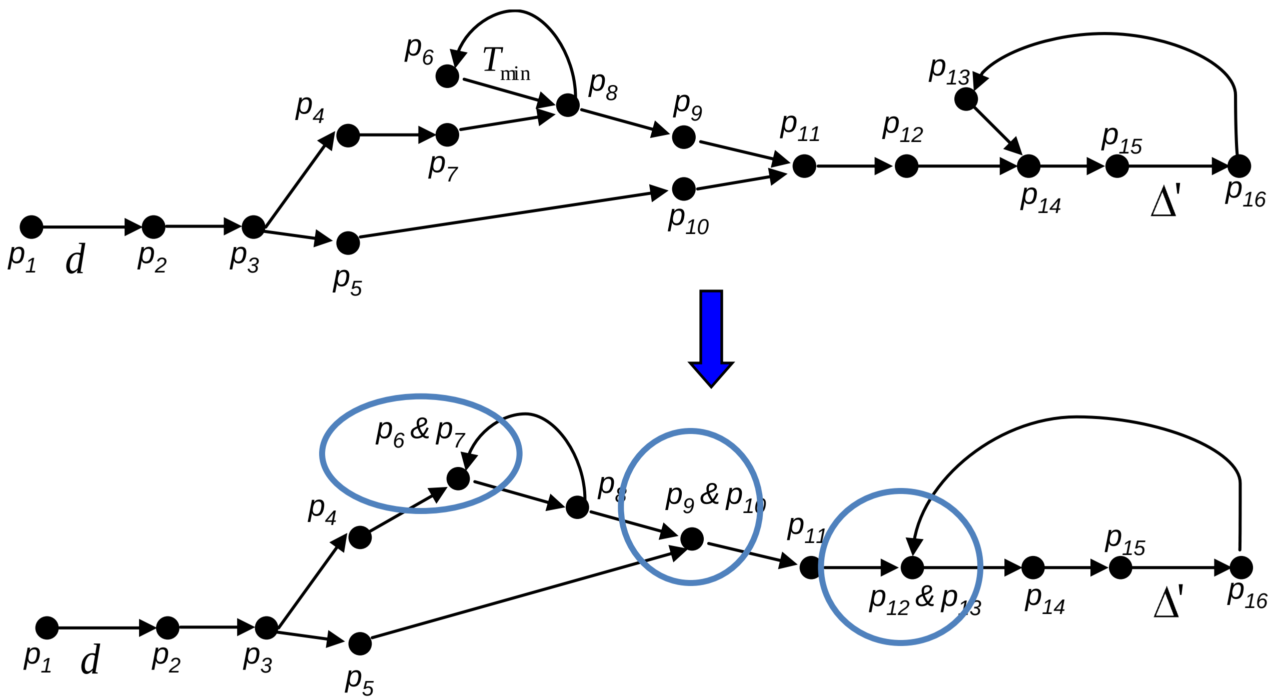
$\delta(p_1, p_2) = d$ means any event with time stamp t at p_2 can be processed when all events at p_1 are known up to time stamp $t - d$



Dependencies

- If an event at p_1 will affect an output signal that may also depend on an event at p_2, p_1 and p_2 are equivalent.

- $\exists_p \epsilon P_o$, *such that* $\delta(p_1, p) < \infty$ *and* $\delta(p_2, p) < \infty$

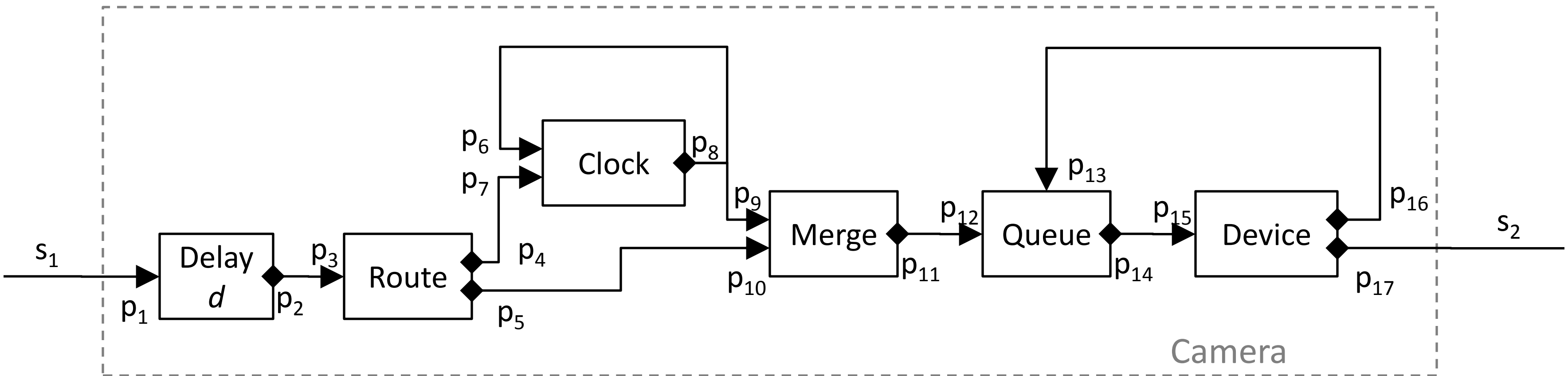


Relevant Dependency

$$d: Q \times Q \longrightarrow D$$

- Q … set of equivalence classes of input ports in a composition
- Examine weights of relevant dependency graph
- Indicates how much we can advance time at a port without knowing all events on the other ports in a composition

Relevant Dependency

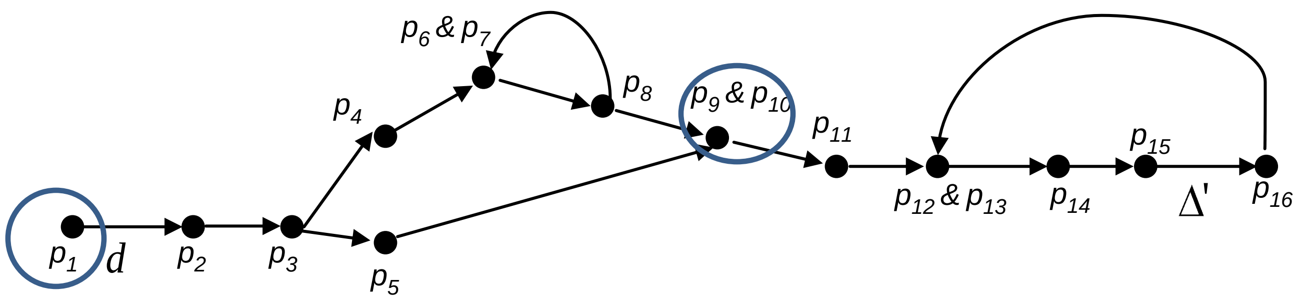


Causality Interface of each actor:

$$\delta_{Delay}(p_1, p_2) = d,$$
$$\delta_{Router}(p_3, p_4) = 0, \quad \delta_{Router}(p_3, p_5) = 0,$$
$$\delta_{Clock}(p_6, p_8) = T_{min}, \quad \delta_{Clock}(p_7, p_8) = 0,$$
$$\delta_{Merge}(p_9, p_{11}) = 0, \quad \delta_{Merge}(p_{10}, p_{11}) = 0,$$
$$\delta_{Queue}(p_{12}, p_{14}) = 0, \quad \delta_{Queue}(p_{13}, p_{14}) = 0,$$
$$\delta_{Device}(p_{15}, p_{16}) = \Delta, \quad \delta_{Device}(p_{15}, p_{17}) = 0$$

Relevant Dependency

- E.g. $d(p_1, p_{9,10})$



any event with time stamp t at port p_9 can be processed when all events at port p_1 are known up to time stamp $t - d$

Relevant Order

Partial order on events

e_1 event with time stamp t_1 at port in q_1

e_2 event with time stamp t_2 at port in q_2

$$e_1 <_r e_2 \Leftrightarrow t_1 + d(q_1, q_2) < t_2$$

$<_r$ ... relevant order

If neither $e_1 <_r e_2$ nor $e_2 <_r e_1$ then $e_1 \parallel_r e_2$

Execution Strategy

… based on relevant order

1. Start with E, a set of events in the event queue.
2. Choose $r \subset E$ so that each event in r is *minimal* in E.
   (e is minimal when $\forall e' \epsilon E$, $e <_r e'$ or $e \,||_r\, e'$)
3. Process events in r, which may produce a set of new events E'.
4. Update E to $(E \setminus r) \cup E'$.
5. Go to 2.

Execution Strategy

… based on relevant order

1. Start with E, a set of events in the event queue.
2. Choose $r \subset E$ so that each event in r is *minimal* in E **and we have seen all events that are less than events in r in the relevant order.**
3. Process events in r, which may produce a set of new events E'.
4. Update E to $(E \setminus r) \cup E'$.
5. Go to 2.

Consider:

- network delay: network ports receive events from other computers or external I/O

Execution Strategy

… based on relevant order

1. Start with E, a set of events in the event queue.
2. Choose $r \subset E$ so that each event in r is *minimal* in E and we have seen all events that are less than events in r in the relevant order.
3. Process events in r, which may produce a set of new events E'.
4. Update E to $(E \setminus r) \cup E'$.
5. Go to 2.

Consider:

- network delay: network ports receive events from other computers or external I/O
- synchronization errors on the clocks
  - Wait for the physical time to be the estimated phys time + error

Conclusion

- Key requirement in a PTIDES program for preserving runtime determinism:

  each event e with model time t at a real-time port must be received before the physical time exceeds $t-\sigma$*, where* σ *is the setup time of the real-time port.*

- A PTIDES program is deployable if this requirement can be guaranteed.