Real-Time and Dependability Concepts

Presented by:

David Wang

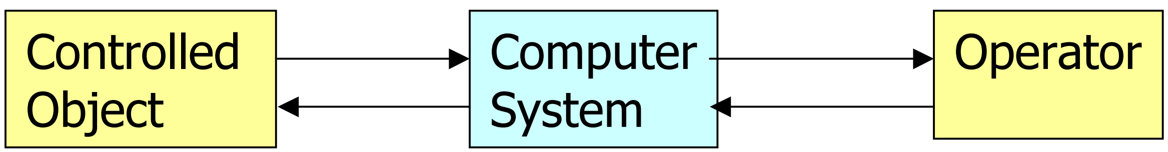
Pallavi Priyadarshini

Agenda

- Real-time System (RTS) concepts and classification
- Dependability concepts
- Models of distributed real-time computing
- Replica Determinism
- Input/Output
- Summary

Introduction (1)

- Real-time system – changes its state as a function of (real) time



- Controlled object, Operator = environment of computer system
- Computer system reacts to stimuli from environment within time intervals dictated by environment
- Instrumentation interface = interface between computer system and controlled object (sensors, actuators)

Introduction (2)

- Man machine or operator interface = interface between computer system and operator
- Duration between stimulus and response time constrained
- Real time transaction – sequence of communication and processing steps between a stimulus and response
- Distributed RTS – Consists of set of nodes connected by a RT communication system

Assumptions about environment

- Reason : to guarantee temporal requirements are satisfied given processing capacity RTS is limited
- Load hypothesis
  - Peak load generated by the environment
  - If RTS not designed to handle peak load, system will fail
- Fault hypothesis
  - Types and frequency of faults that RTS must handle
  - If more faults generated, RTS will fail
- Worst Scenario that RTS should handle: Simultaneous peak load and maximum faults

Classification of RTS (1)

- Based on timing failure
- Soft RTS
  - Consequences of timing failure not very serious
  - Eg., letter-sorting machine, telephone switching system
- Hard RTS
  - Consequences of timing failure catastrophic
  - Eg., bank online transaction processing system, railway signaling system

Classification of RTS (2)

- Based on safe states
- Fail-safe RTS
  - Safe states can be identified and accesses during system failure
  - Eg., railway signaling system (all trains can be stopped and lights turned red)
- Fail-operational RTS
  - No safe state can be identified.
  - Eg., flight control system (has to provide minimum level of service during failure)

Classification of RTS (3)

- Based on system implementation approach
- Guaranteed-response RTS
  - Deliver a design to meet specific fault and load hypotheses even in extreme scenario. No probability
  - Based on resource adequacy
- Best-effort RTS
  - No precise fault and load hypotheses. Probabilistic
  - Based on resource inadequacy. Dynamic resource allocation
  - Majority existing RTS today

Typical RT Applications

- Plant Automation – first application
- Telephone Switching – rigid availability and maintainability requirements
- Automotive Control – early stages
- Intelligent Products
  - Autonomous RTS providing a service to users
  - Eg., automatic scale with an integrated microcontroller for calibration, weighing and record keeping

Dependability

- Nonfunctional attributes that relate to the quality of the service over an extended period of time.
- Important measures of dependability attributes:
  - Reliability
  - Safety
  - Availability
  - Maintainability
  - Security

Failure and Faults

- Failure
  - Event that deviates the agreed specification of the system occurring at a particular point in real time
- Fault
  - The cause that bring system into an incorrect internal state (error) is called fault
- Fault-tolerance
  - Mask and repair errors
- Key of fault-tolerance systems
  - Redundancy

Redundancy (1)

- Types of Redundancy
  - Physical resource redundancy
    - Replication of physical resources
    - E.g. three computers instead of one, select the result that is in the majority
  - Time redundancy
    - Repetition of computation or communication
    - E.g. resend message if no ack. back
  - Information redundancy
    - Specific encoding technique
    - E.g. adding parity bit into original data

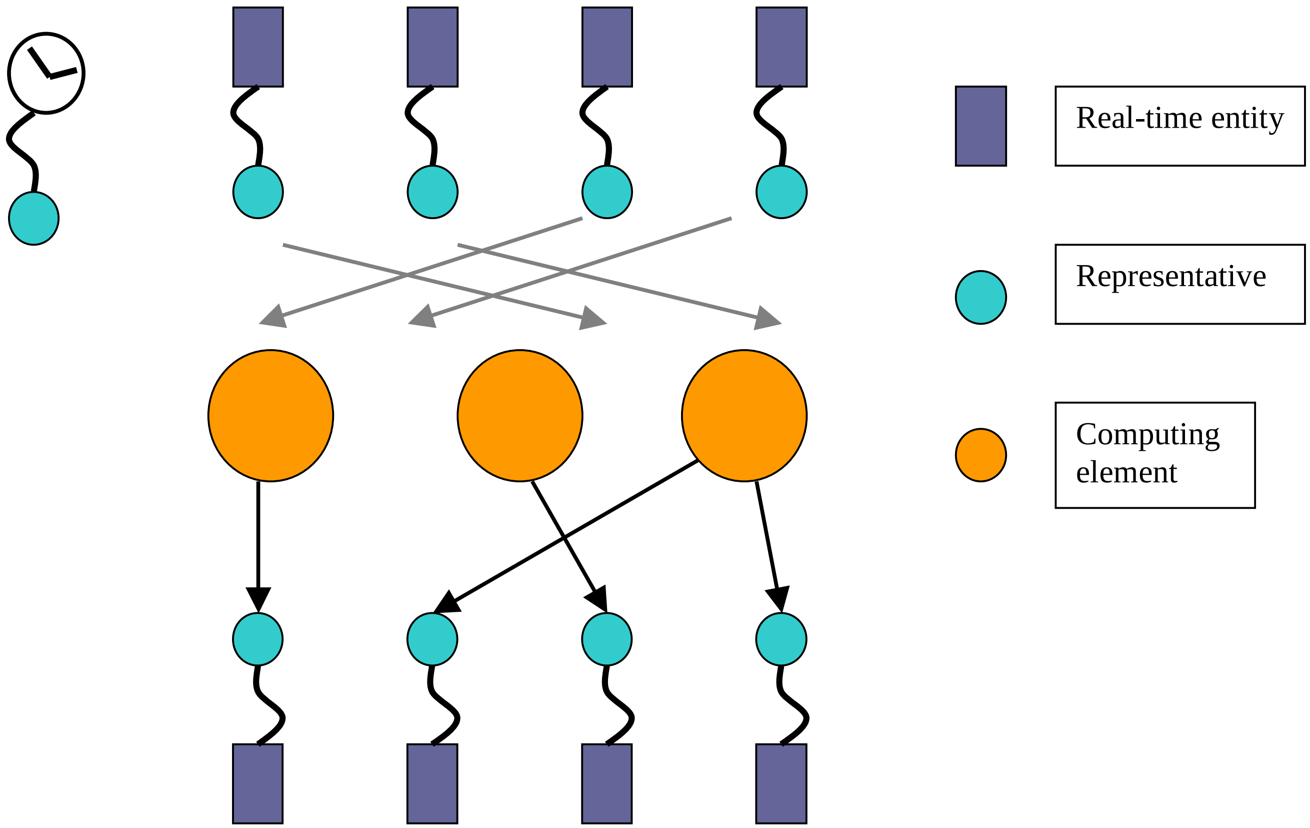
Redundancy (2)

- Schemes of redundancy
  - Passive redundancy (standby or cold redundancy)
    - Activates the redundant physical resources only after the primary resource has failed
    - Assumes primary system has error detection
    - Subsystems need to be informed
  - Active redundancy system (hot redundancy)
    - Activates all redundant physical resources simultaneously
    - Requires that the replicated subsystems visit that same states at about the same time

Models of Distributed Real-time Computing

- Map physical reality into computer world
  - Real-time entity (RT): element has a state that system is suppose to acquire or modify
    - E.g. fluid valve (level, flow); oven (temperature)
  - Representative : computational entity that observes or acts on RT
    - E.g. temperature sensor reader
  - Computing element: computational entity that processes data form RT and acts on other RT
    - E.g. control programs

Models (1)



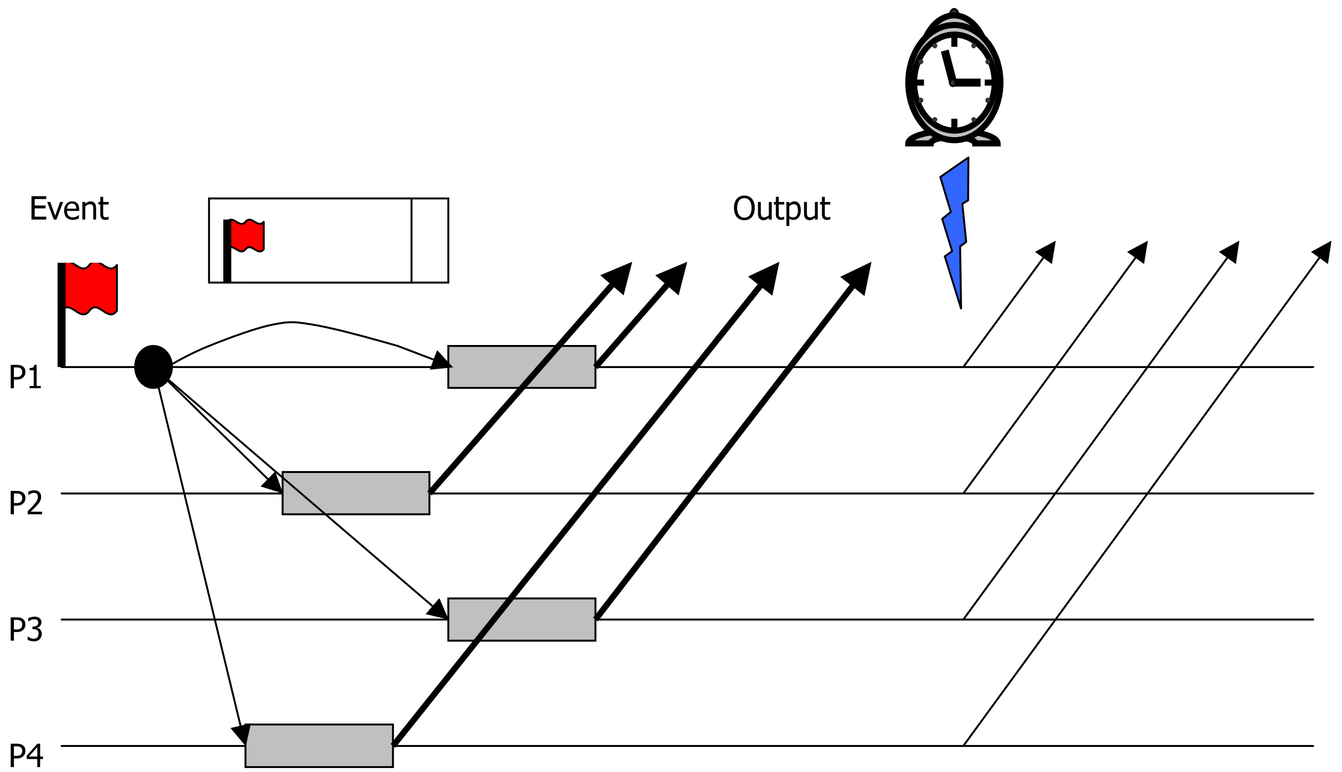
Models (2)

- Communication path in RT system
  - RT to Representative
    - Reliable and timely perception of, actuation on, the state of RT entities
  - Representative to Computing element
    - Reliable and timely processing of the information acquired and production of response

Design approach

- Event-Trigged (ET)
  - One that reacts to significant external events directly and immediately
- Time-trigged (TT)
  - One that reacts to significant external events at pre-specified instants in time

ET (1)



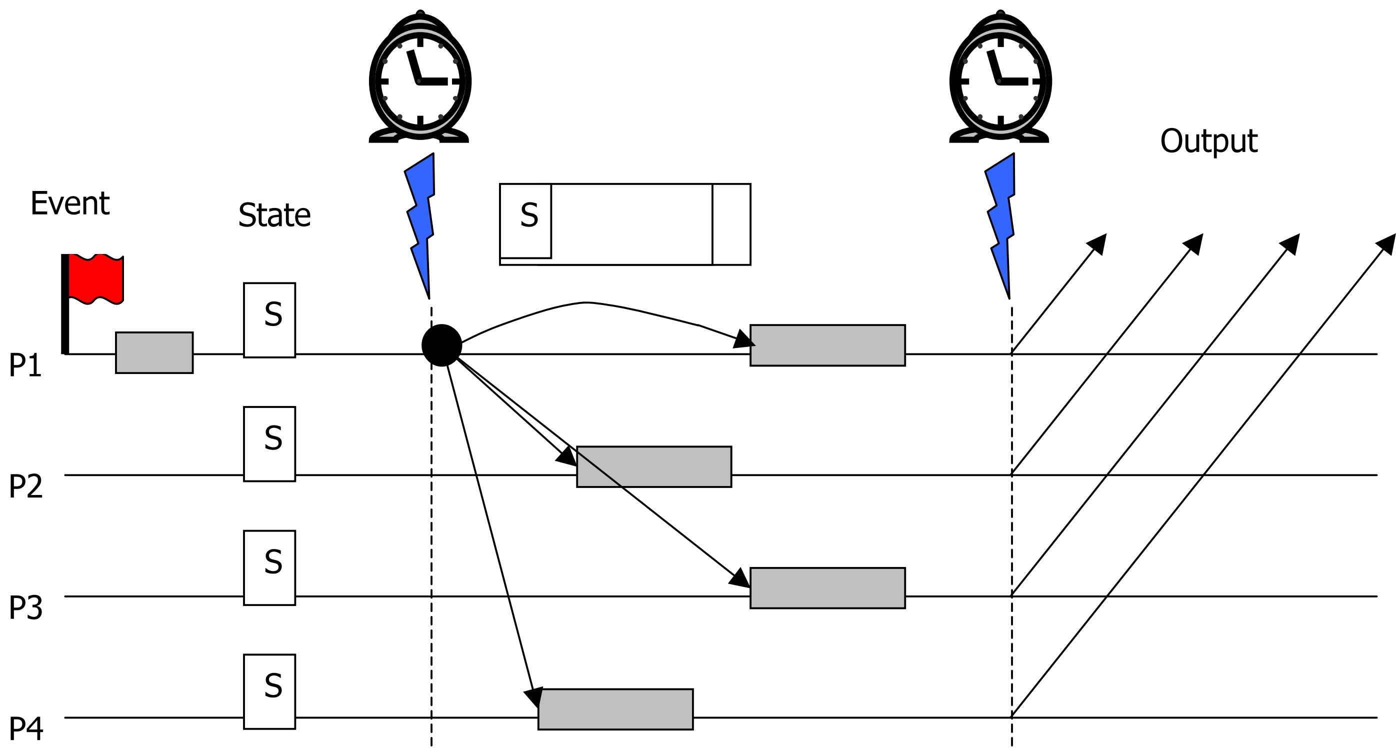
ET (2)

- Behavior
  - Idle, wait for something happen
  - When event occurs, a massage is sent to interior of system
  - Computing elements process it and produce outputs
- Design concern
  - The set of assumptions about the environment are by nature not rigid
  - What if the workload beyond designed worst-case

ET (3)

- Design concern (cont.)
  - System are prone to event shower - Exception that causes lots of system nodes providing alarm information, some of it redundant or repeated by propagation
  - Subsequent events cause by alarms can be predicted, so compact successive instantiations of the same alarm
  - Discard redundant events
  - Prepare computing and communication resource of forth coming event shower
  - Place load and flow control close to RT.

TT (1)



TT (2)

- Behavior
  - Events are collect and pre-process between period in the periphery of system
  - State information sent to computing elements at a giving moment
  - Produce outputs at pre-defined instant
- Design concern
  - The evolution of environment is well defined, and the worst-case workload is perfectly determined

TT (3)

- Design concern (cont.)
  - Environment must be thoroughly described
    - E.g. weapons-control TT system only designed for maximum 50 enemies, then it will be blind when the 51st enemy arrives
  - Reaction time is also important
  - Period should be short enough to match the rate of evolution of the environment and long enough for duration of processing

Compare ET and TT (1)

- ET
  - Good for sporadic event
  - When overloaded, important event may be lost
  - Faster average response time
  - Can use clock-less communication protocol (chapter 17)
- TT
  - Good for Periodic event
  - No concept of overload
  - Simpler to test

Compare ET and TT (2)

- TT (cont.)
  - Good for small closed system, but not easy to extend
  - Must use TT communication protocol
- Combining both produces good result

Replication and Replica Determinism

- Why replicate – Performance, Fault -tolerance
- 2 approaches:
  - Active replication – All replicas process requests at same time
  - Passive replication – One replica is primary, others are backup
- RTS need active replication. Passive schemes have glitch when primary replica fails.
- Active replication needs replica determinism:
  - Ability of replicas to produce same results upon receiving same inputs

Sources of non-determinism

- System or environmental Interaction
  - System calls that return host-specific information
    - gettimeofday(), gethostname(), …….
    - Random number generators
  - Environmental (third-party) interaction
    - Interaction with human through graphical interface
    - Interaction with shared memory, I/O, etc.
- Scheduling/Control Flow
  - Multithreading
  - Asynchronous Event – Interrupts, exceptions, signals

Example

- Simple example to motivate need for replica determinism

A triplicated flight control system

Channel 1	Speed<limit	Abort	Decelerate
Channel 2	Speed>limit	Takeoff	Accelerate
Channel 3	Speed>limit	Takeoff	Decelerate (erroneous)
Majority		Takeoff	Decelerate (erroneous)

Determinism and Scheduling

- Preemptive scheduling
  - Newly arrived higher priority request causes suspension of current lower priority request
- For Active replication in TTS and ETS without preemptive scheduling, need:
  - Deterministic programs
  - Reliable and ordered multicast
- ETS with preemptive scheduling : How to guarantee preemption does not hinder replica determinism?

Determinism with Preemptive Scheduling (1)

- Process and message preemption complement each other
- Preemption should be deterministic and occur in bounded time
- Preemption point – points in process execution where preemption can occur in its context without damaging execution correctness

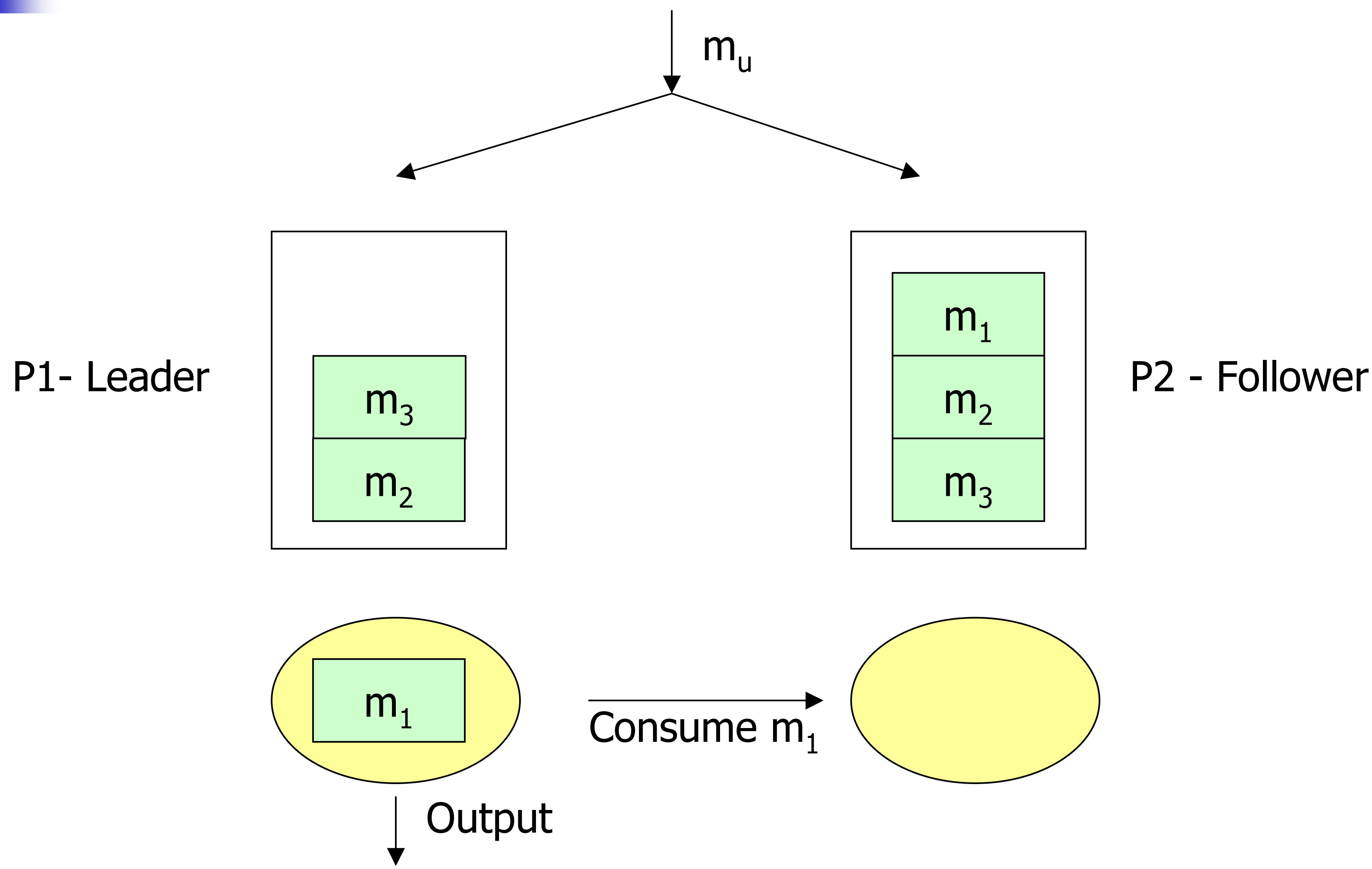
Determinism with Preemptive Scheduling (2)

- Problems posed by preemption:
  - Urgent messages put at head of message queue in all replicas, but queues might not be in same states
  - Preemption point should be same in all replicas
  - Preemption must occur in bounded time
- Solution to non-determinism?
  - Semi-active replication: between active and passive. Example, Leader-follower replication

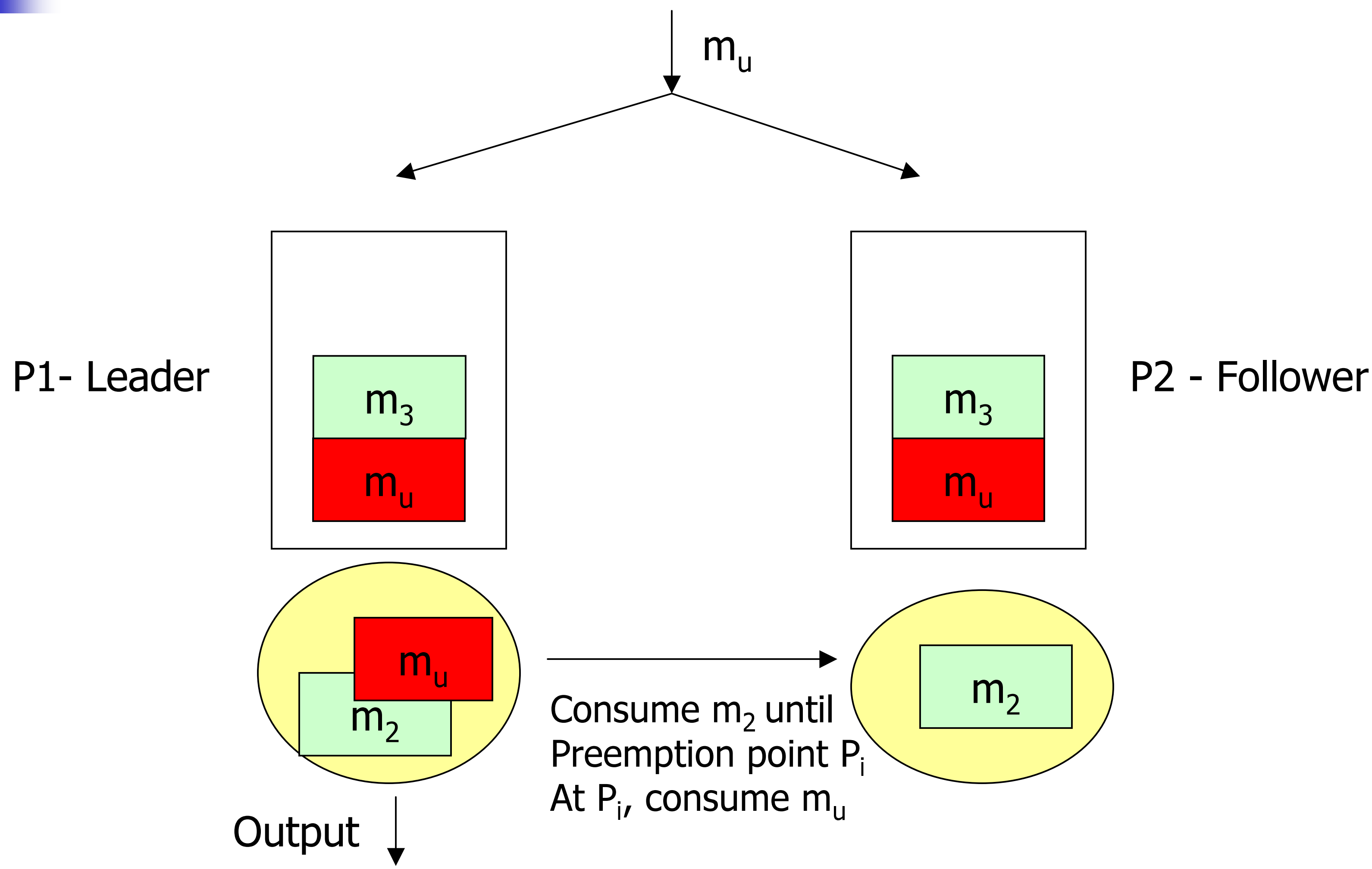
Leader-follower Replication (1)

- All replicas process input messages
- Leader always ahead by one message - not in phase like active replication
- Only leader outputs results
- Reliable multicast but not ordered
- Leader selects order – picks message from queue, executes and orders followers to execute to process same
- Leader reaches preemption point (P_i), checks queue; if urgent message (m_u), preempts current processing (m_2)
- Instructs followers to execute m_2 till P_i, then m_2

Leader-follower replication (2)



Leader-follower replication (3)



Leader-follower replication (4)

- How L/F solves determinism problem:
  - Ordering of messages – Leader selects execution order
  - Preemption point – Same in all replicas
  - Bounded preemption latency
  - Bounded takeover glitch – When the leader fails, bounded election time, bounded lag between L/F

Determinism of Pseudo-replicas

- Certain entities made redundant cannot generate bit-for-bit identical values
- Examples:
  - analogue reading of 2 replicated temperature sensors of same object when digitized
  - 2 replicated random number generators provide 2 different numbers for replicated request
  - *get_time* instruction for time dependant computation
- How to guarantee outputs from distributed pseudo-replicas used consistently?

Approach 1 – Pre-processing

- Running a distributed consensus protocol to harmonise readings, and give single value
  - Source-based pre-processing – single value given by one or all pseudo-replicas
  - Destination-based pre-processing – requester performs consensus function upon receiving pseudo replica values
- Approach used in sensor readings

Approach 2 – Single server (1)

- Information already in digital form and local, each component is reliable/significant
- If each component uses local value, results diverge
- Hinders replica determinism
- Approach used in random number generators and clock

Approach 2 – Single server (2)

- To get mutually consistent result
  - Pseudo-replicas not accessed directly (e.g.., get_time system call); Encapsulated in a server
  - First replica arriving at request point addresses server and gets reply
  - Reply gets to all replicas (from first replica or server)
  - Other replicas at request point used the reply received earlier (consistent)

Interesting projects

- Examples of distributed fault-tolerant real-time systems
  - Delta-4 [2]
  - MARS : Maintainable Real-Time system [3]
- Employ several concepts discussed
- Any thoughts about real-time databases?

Input / Output

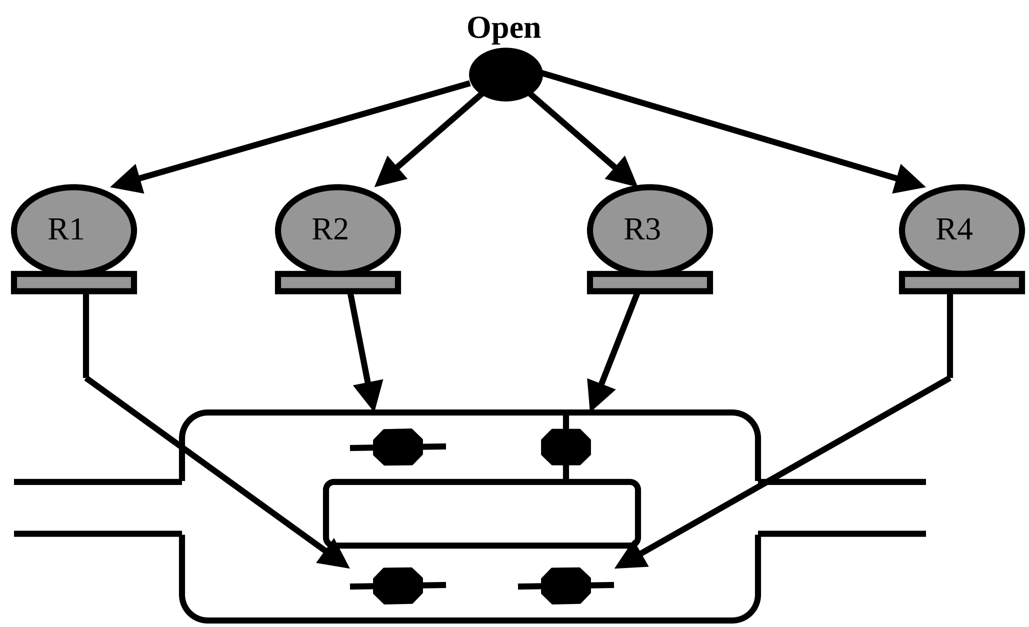
- Hardware I/O
  - Sensors and actuators
- Software I/O
  - Application gateways that pass information between system A and B
- Generally, can imagine each gateway as a sensor/actuator pair
- How to design I/O for real-time systems (next slide)

Actuators (1)

- Simplex single drive
  - One actuator / one representative (computer part)
  - Simplest for used when actuators are ‘trustworthy’ and computer part are considered not to fail
- Simplex multiple drive
  - One actuator / multiple representatives
  - Used when actuators are trusted but representatives are not reliable

Actuators (2)

- Fully redundant (most reliable)
  - Multiple actuators / multiple representatives

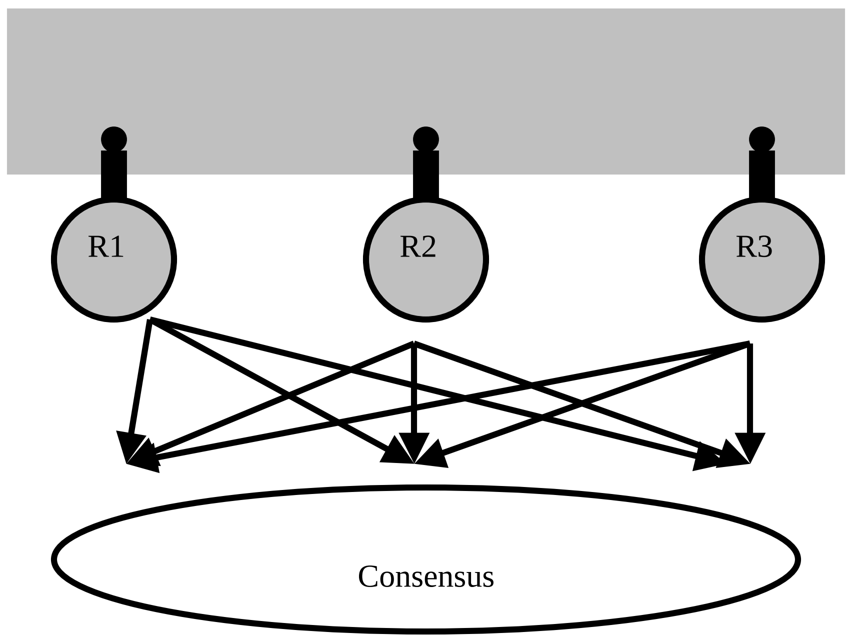


Sensors (1)

- Simplex single access
  - One sensor / one representative
  - Simplest for used when sensors are ‘trustworthy’ and computer part are considered not to fail
- Simplex multiple access
  - One sensor / multiple representatives
  - Used when sensors are trusted but representatives are not reliable

Sensors (2)

- Fully redundant
  - Multiple sensors / multiple representatives



Distribution and Reliability (1)

- Time of distributed perception and actuation
  - The role of time is paramount
  - Goes far beyond supplying a time base for delayed or periodic actions
  - Clock-driven perception can determine the order of external events from different nodes
  - Clock-driven actuation can trigger outputs in relation with event occurring in other nodes
  - When reliability is of concern, use redundancy, and use global clock to synchronize the individual replicas of actuators and sensors

D & R (2)

- Semantic of distributed actuation
  - At-least-once (weakest and simplest)
    - Should be used whenever possible. Namely, when actuation is idempotent. (E.g. open valve)
    - Achievable with any replication scheme
  - Exactly-once (Strongest and complex)
    - Fulfills any needs
    - Achieved by active replication with synchronized output
    - Tricky to obtain with semi-active replication

D & R (3)

- Semantic of distributed actuation (cont.)
  - At-most-once (for completeness)
    - Normally not a design approach for reliable actuation
    - Used when the requirement is such that the semantics is not at-least-once, but cannot be enforced to be exactly-once.
    - Admit certain ‘omission’ failures (falls into ‘one or none’ behavior)

Summary

- Main issues relevant to distributed, fault-tolerant RTS
- Distinguishing characteristics of RTS
- Classifications
- Failure and redundancy
- Event and time triggered system
- Predictable redundancy management
- I/O of real-time system

References

- [1] *Distributed Systems*, 2nd edition, 1993, Mullender, Chap 16.
- [2] P.A. Barret, A.M. Hilborne, P.G. Bond, D.T. Seaton, P. Verissimo, L. Rodrigues, and N.A. Speirs. The delta-4 extra performanc architecture (xpa). In *Proc. of the 20th International Symposium on Fault-Tolerant Computing*, pages 481 – 488, June 1990.
- [3] Kopetz, H.; Damm, A.; Koza, C.; Mulazzani, M.; Schwabl, W.; Senft, C.; Zainlinger, R. Distributed fault-tolerant real-time systems: the Mars approach. Micro, IEEE Volume 9, Issue 1, Feb. 1989 Page(s):25 - 40
- [4] http://www.vmars.tuwien.ac.at/projects/xbywire/projects/new-esrel97.html

Q & A