A QoS-driven Resource Allocation Framework based on the Risk Incursion Function and its Incorporation into a Middleware Structure & Mechanisms Supporting Distributed Fault Tolerant Real-time Computing Applications

For presentation at the dissertation defense
December 6th, 2001

Juqiang Liu

Department of Electrical and Computer Engineering
University Of California, Irvine
jqliu@ece.uci.edu
http://dream.eng.uci.edu/jqliu

UCI
DREAM Lab

Outline

- Motivation
- The Time-triggered Message-triggered Object (TMO) scheme
  - A real-time distributed software component structure
- The TMO Support Middleware (TMOSM) Architecture
  - A middleware architecture supporting distributed RT computing on COTS platforms
- The Risk Incursion Function (RIF) Scheme and Example Application
- The RIF-based Resource Allocation Framework
- Real-time Fault Tolerance Schemes Incorporated in the framework
  - The Supervisor-based Network Surveillance(SNS) Scheme
  - The Primary Shadow TMO Replication (PSTR) Scheme
  - The Primary Passive TMO Replication (PPTR) Scheme

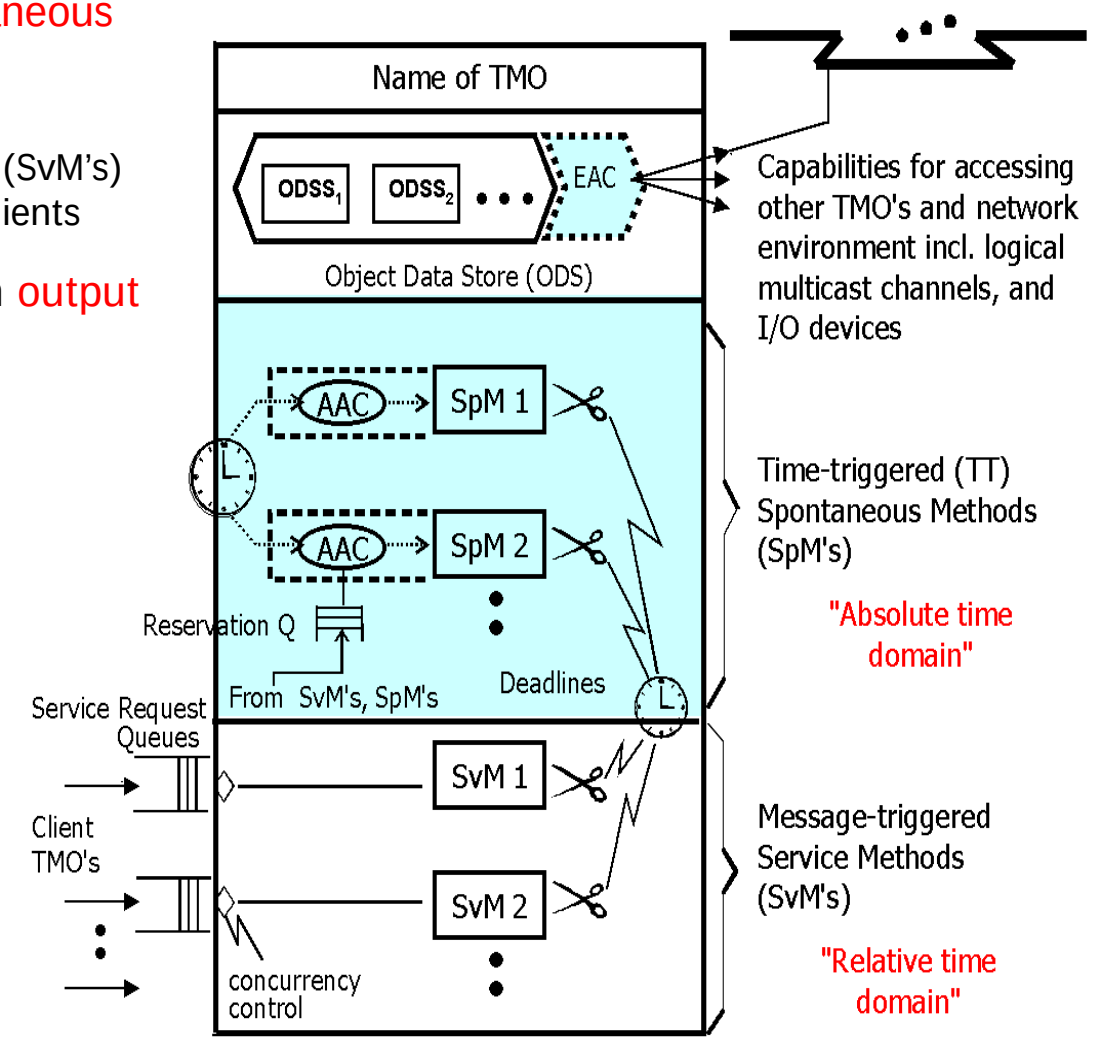
Motivation

- OO design approaches have become dominant in the development of non-real-time business data processing software, however, OO-structuring has had minimal impacts in real-time computer system (RTCS) engineering.
- In spite of the steady decline of computer hardware cost in the computer systems, allocation of computing resources is still a major issue, especially in complex distributed, real-time computer systems.
- Few schemes have been proposed to address the resource allocation problems in an integrated fashion, from the application requirements to the scheduling of various computation resources, such as processors, communication bandwidth and I/O devices.
- The analysis of the fault detection latency bound and recovery bound of a real-time fault tolerance scheme, which is a rare practice until recently, is of critical importance in safety-critical real-time computer systems.

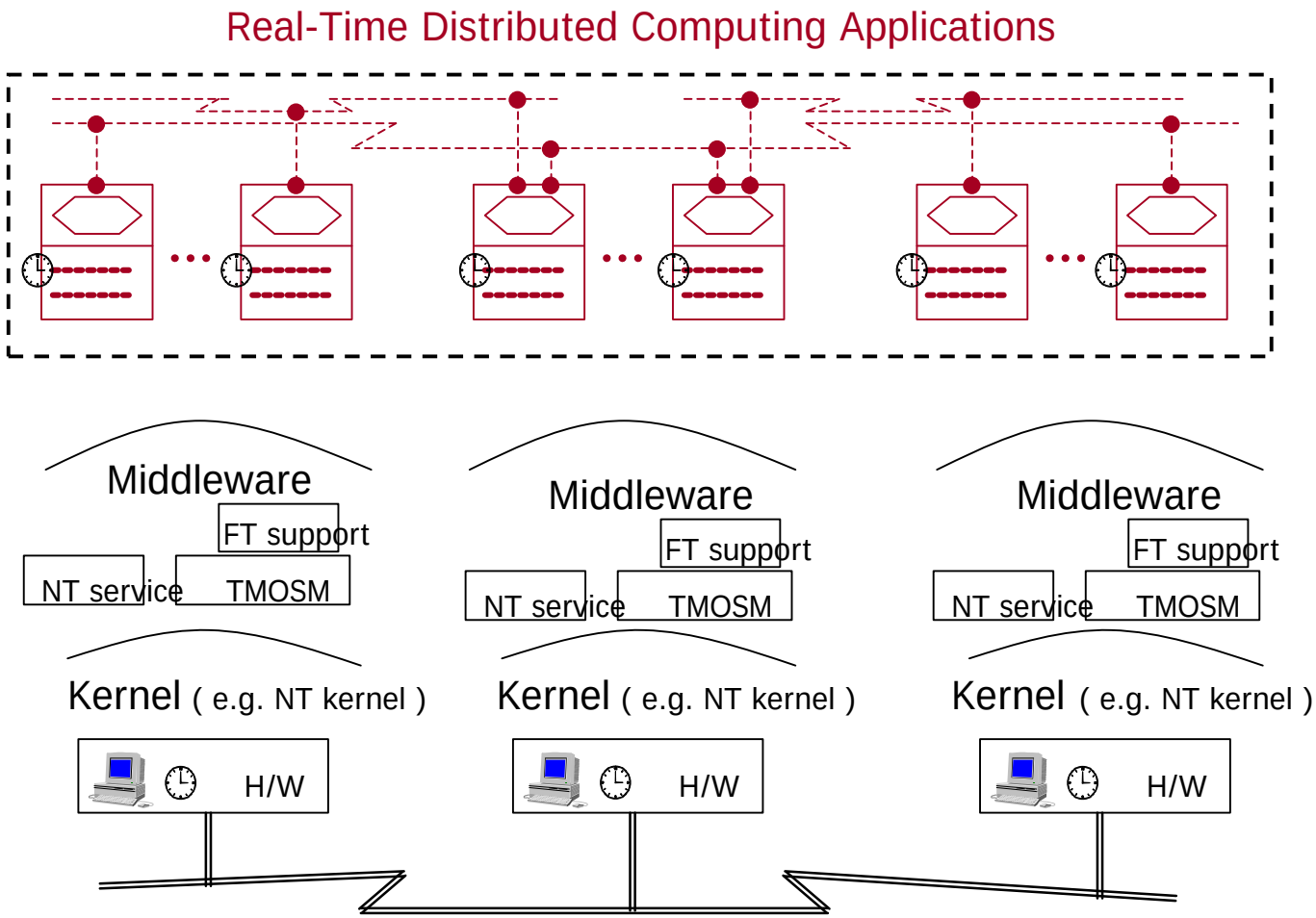
Background:
Time-triggered Message-triggered Object (TMO)
And TMO Support Middleware (TMOSM)

Time-triggered Message-triggered Objects (TMO) Structuring Scheme

- Time-triggered (TT-) or spontaneous methods (SpM's):
  - Clearly separated from the conventional service methods (SvM's) triggered by messages from clients
- Time-window imposed on each output action and method completion
- Connections to the network environment as possible data members:
  - Programmable data-field-channels
  - TMO access capabilities (possibly remote TMO's)
- Basic concurrency constraint (BCC):
  - SpM executions not disturbed by SvM executions.
  - Eases design-time guarantee of timely services of TMO's



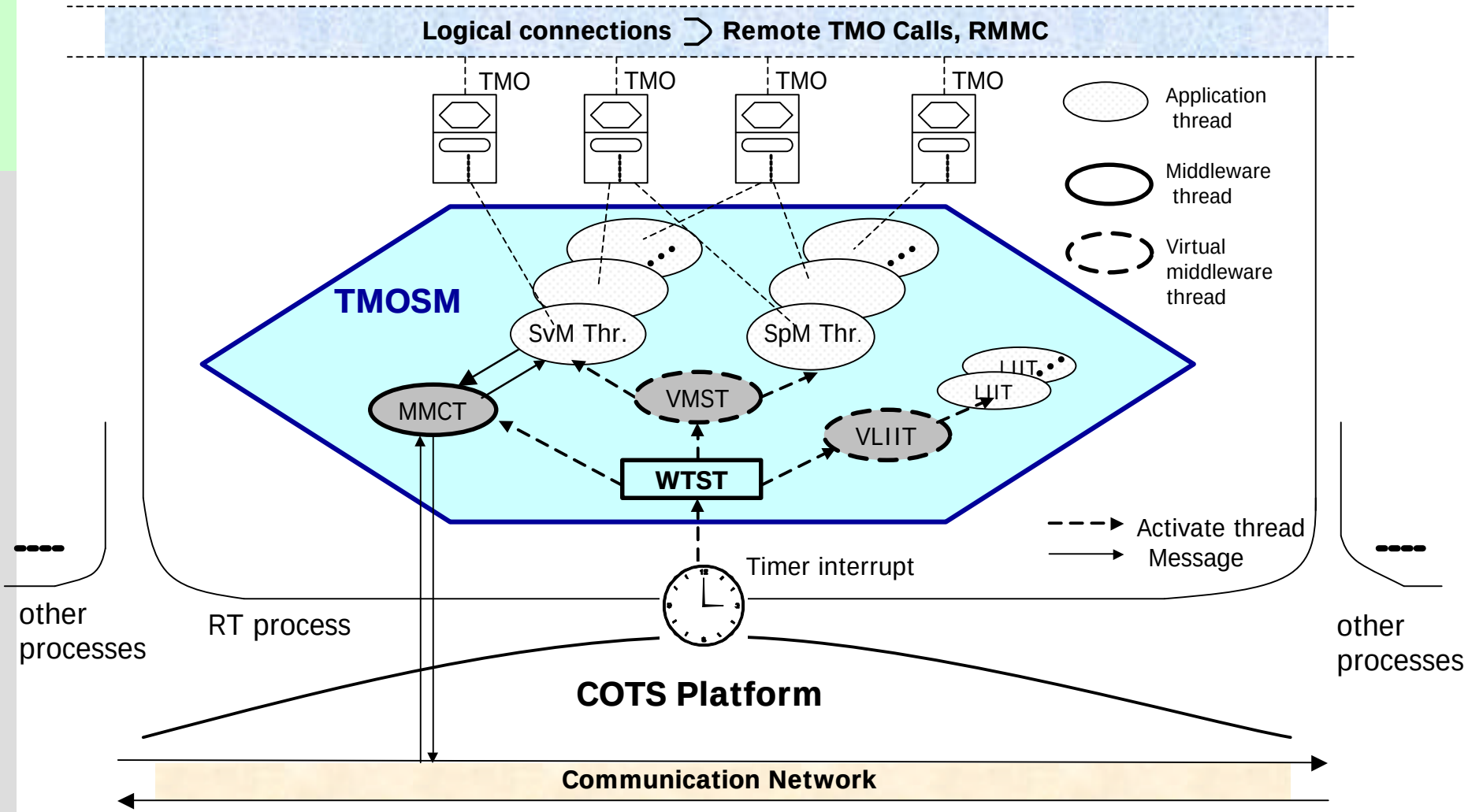
TMO Network Structured Application Execution Facilities



No concerns with

- Processes & Threads
- Object locations (except in avoiding overloaded nodes)

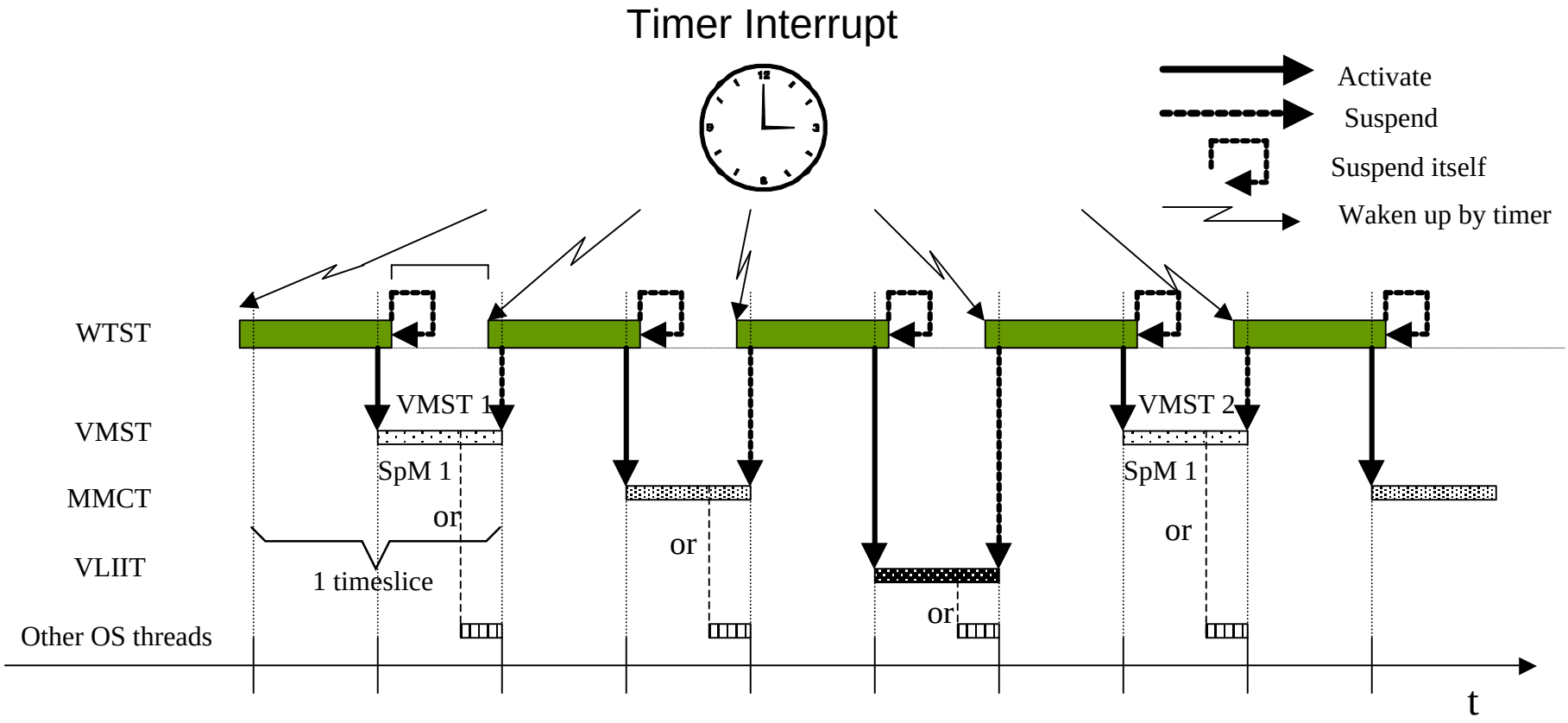
TMOSM Thread Structure



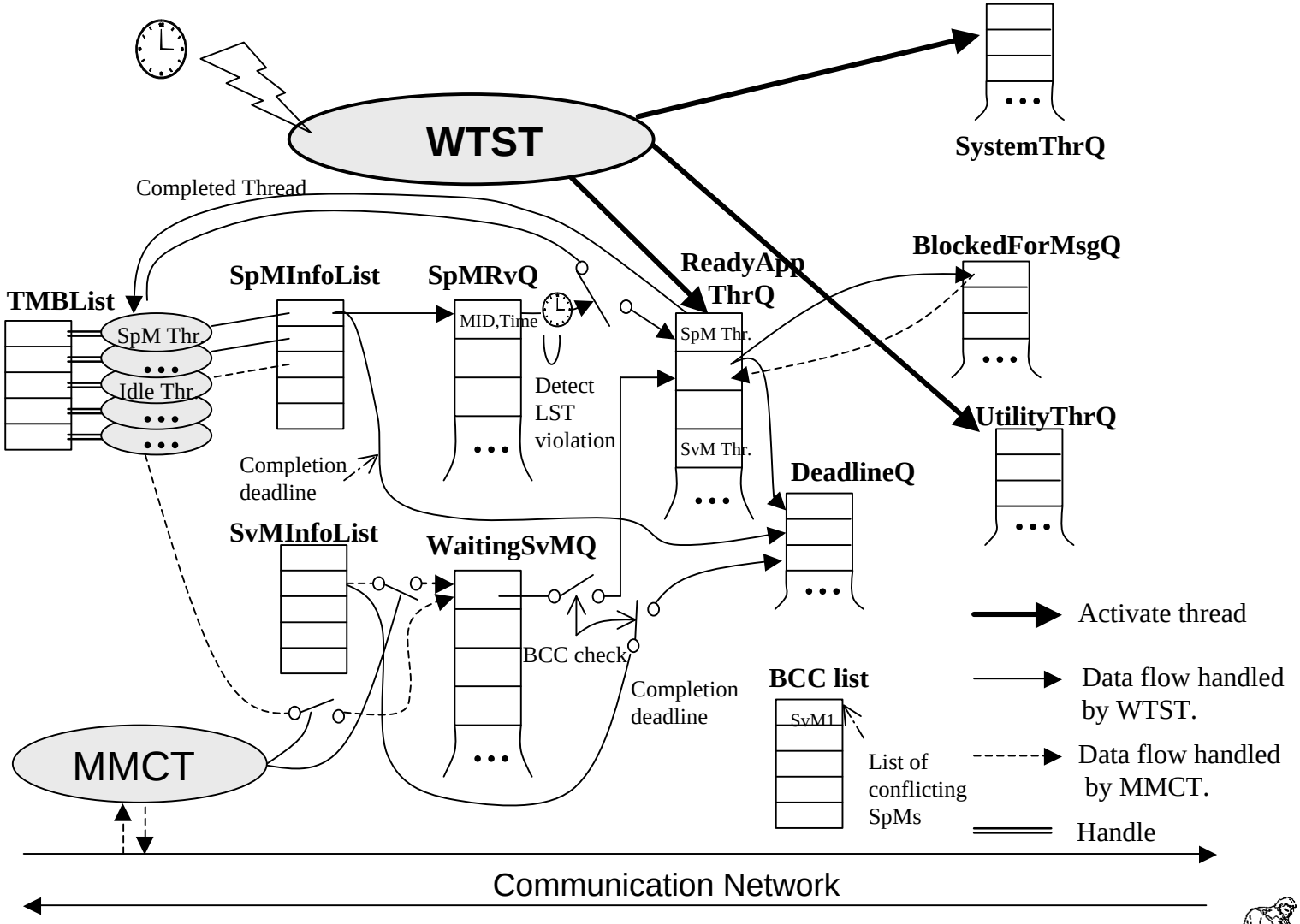
TMOSM -- Thread Structure (cont.)

- WTST (Watchdog Timer & Scheduler Thread): *Master Micro-Thread*
  - Manages the scheduling / activation of all other threads in TMOSM and checks if there are deadline violations
- MMCT (Middleware-to-Middleware Communication Thread)
  - Distributes messages coming through the communication network to their destination threads
- VLIIT (Virtual Local I/O Interface Thread)
  - A virtual thread Managing local I/O activities such as serial character I/O and disk I/O
- VMST (Virtual Main System Thread)
  - A virtual thread representing all application and utility threads including:
    - SpM threads
    - SvM threads
    - *Utility threads*

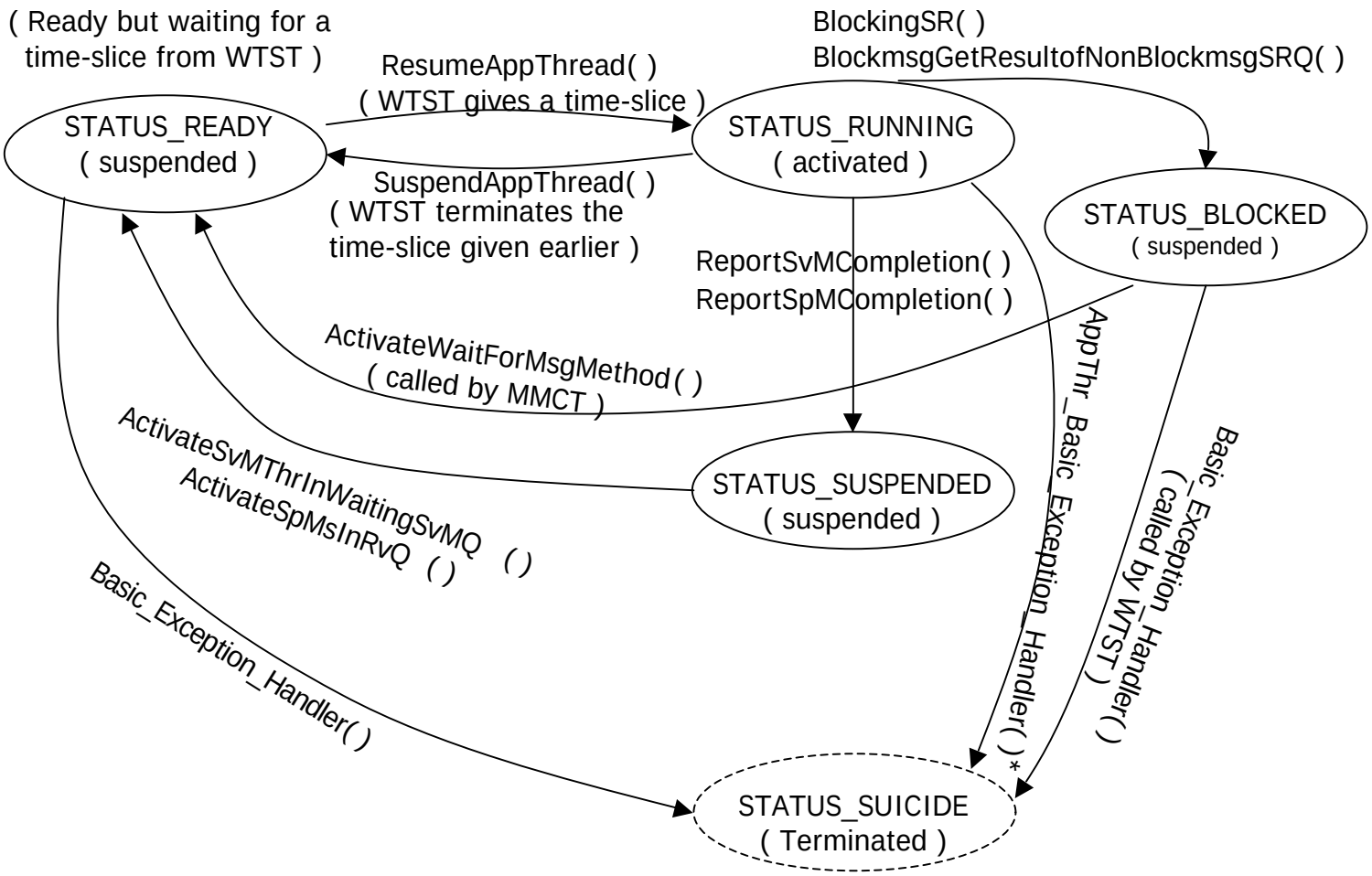
TMOSM – The Time-slicing Scheme

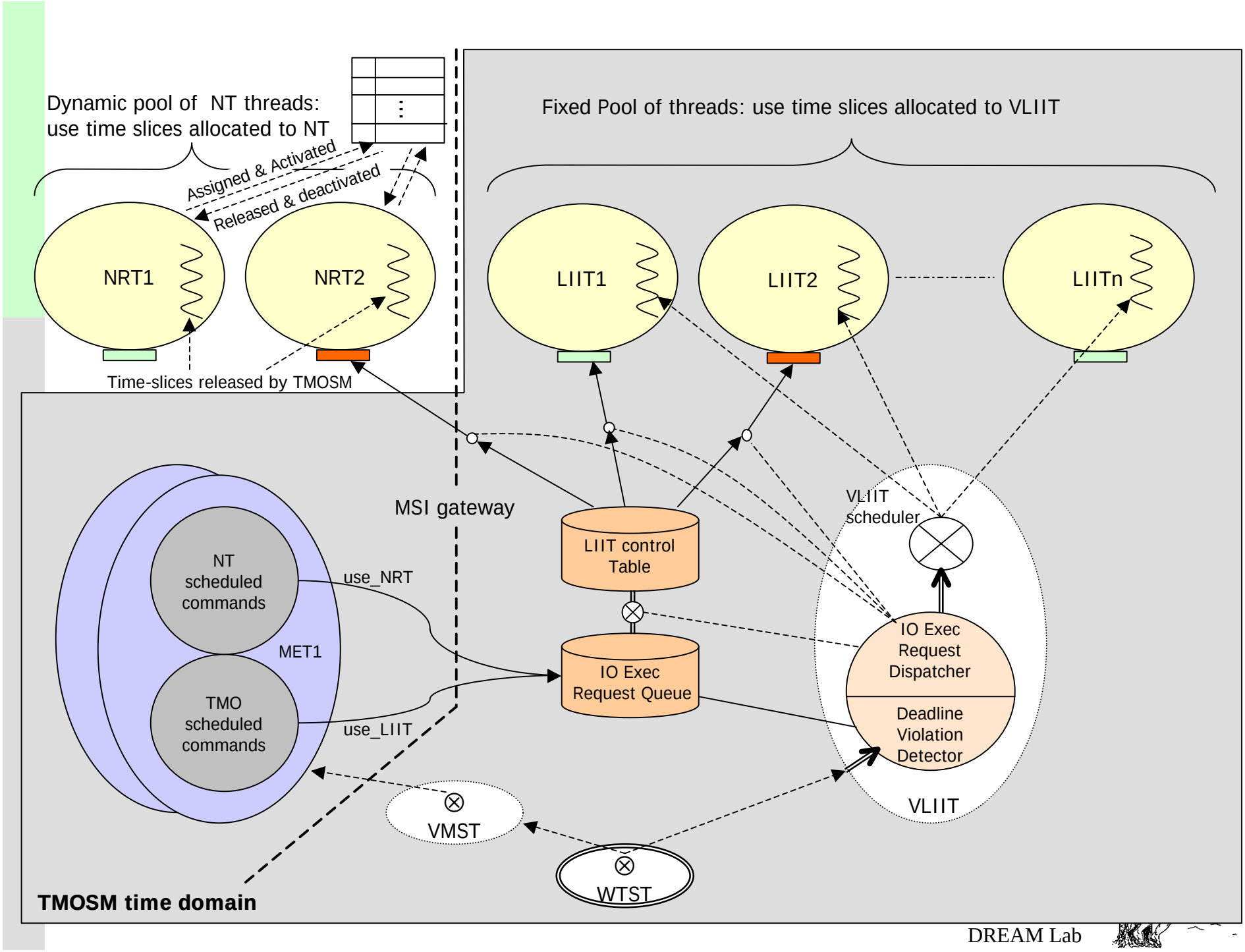


TMOSM – Internal Control Flow



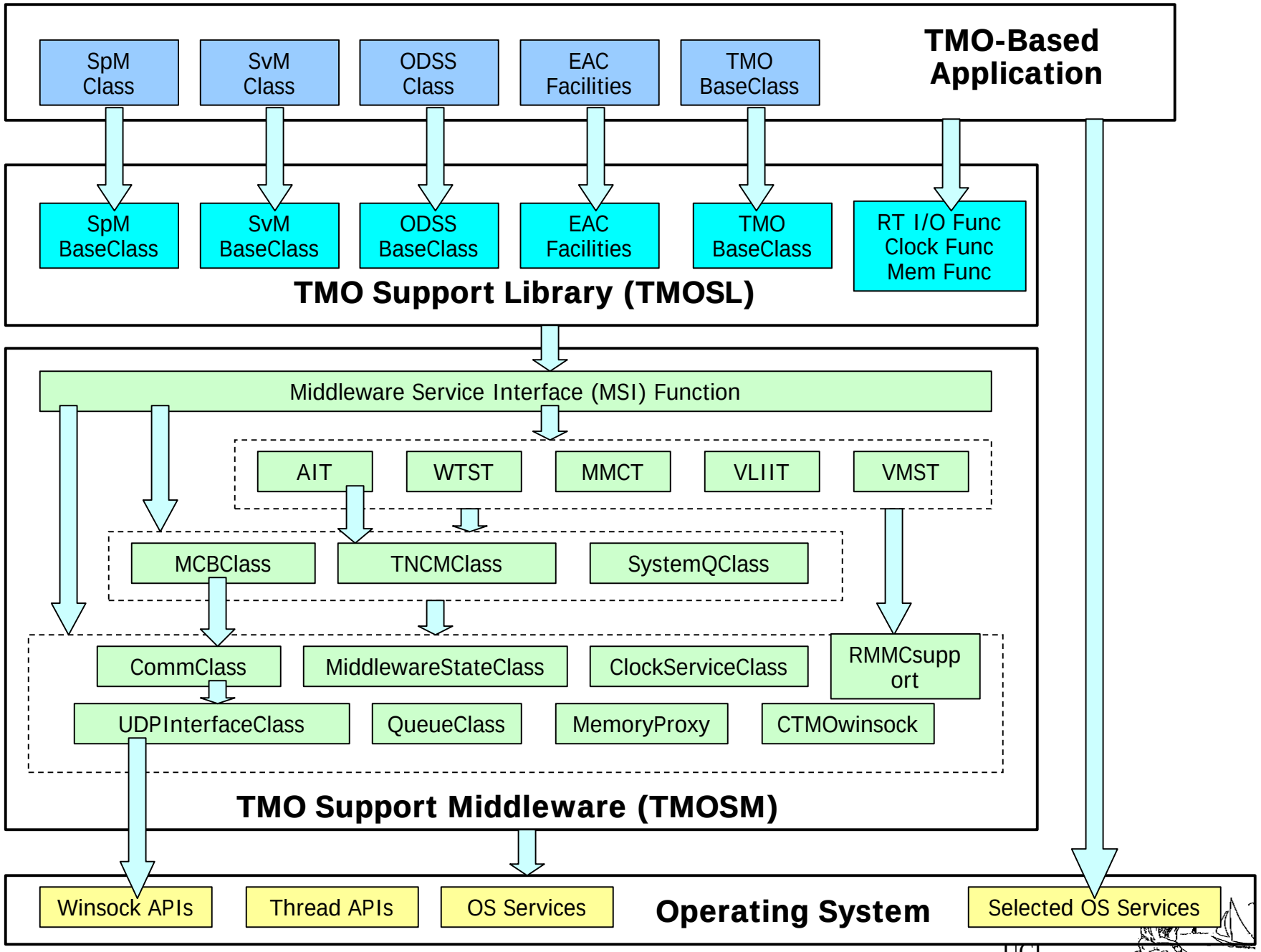
TMOSM – Thread State Transition Diagram



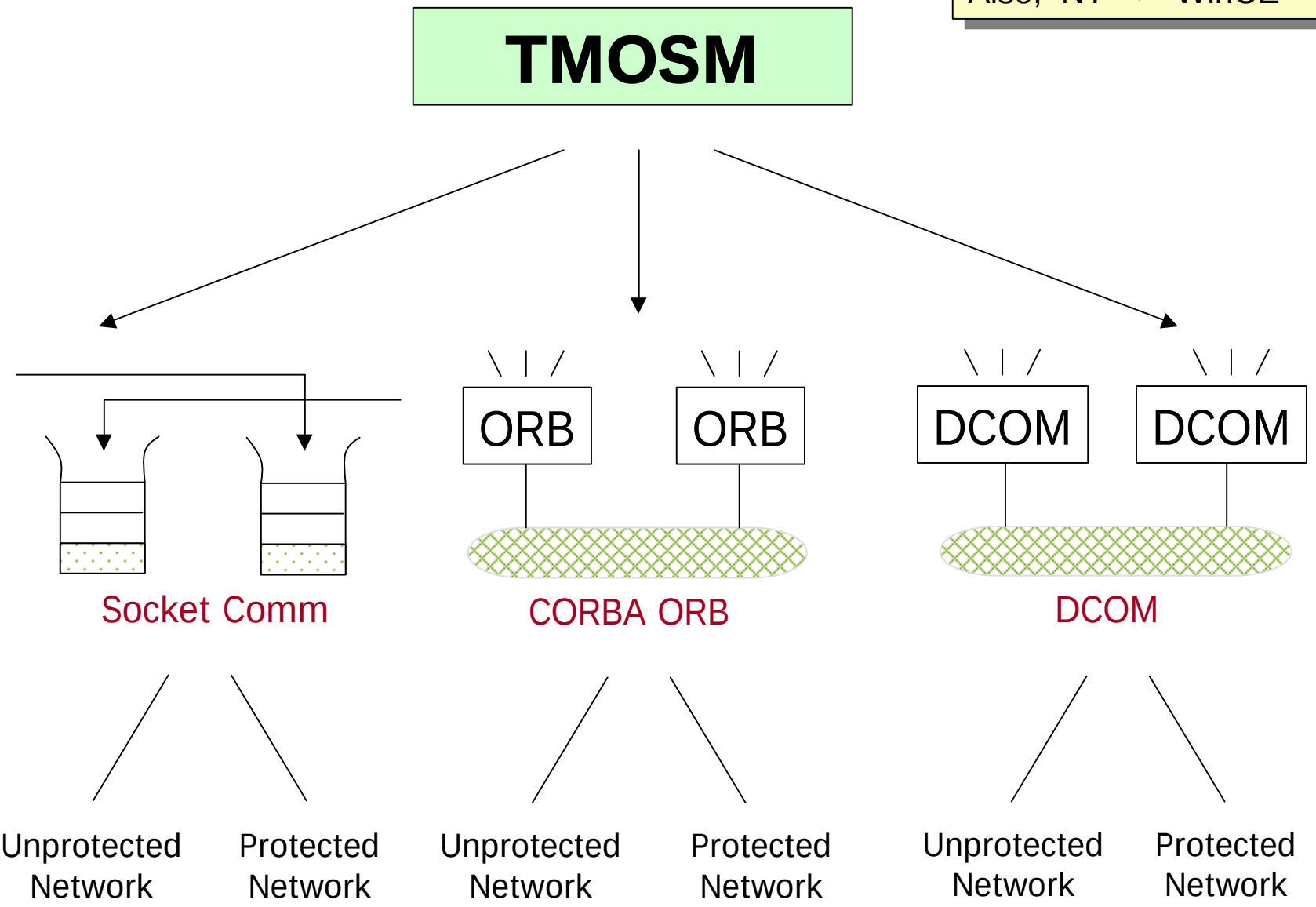

Dynamic pool of NT threads:
use time slices allocated to NT
Assigned & Activated
Released & deactivated
NRT1
NRT2
Time-slices released by TMOSM
Fixed Pool of threads: use time slices allocated to VLIIT
LIIT1
LIIT2
LIITn
MSI gateway
LIIT control Table
VLIIT scheduler
IO Exec Request Dispatcher
Deadline Violation Detector
IO Exec Request Queue
VLIIT
NT scheduled commands
TMO scheduled commands
MET1
use_NRT
use_LIIT
VMST
WTST
TMOSM time domain

TMOSM / NT:
A prototype implementation of TMOSM on Windows NT

- Windows NT's features needed by TMOSM
  - Multi-tasking support
  - High-resolution timer interrupt
    - Waitable Timer construct: Periodic interrupt signal at one millisecond intervals)
  - Top-priority real-time process/thread support
    - TMOSM process is the highest priority-level process (REALTIME_PRIORITY_CLASS)
    - WTST is the highest priority-level thread (THREAD_PRIORITY_TIME_CRITICAL)
    - All other threads in TMOSM are the second highest priority-level threads (THREAD_PRIORITY_HIGHEST)
- Performance of the prototype implementation
  - Supports the time-window for activating a method as small as 10ms
  - Supports the execution deadline as short as 20ms


TMO-Based Application
SpM Class
SvM Class
ODSS Class
EAC Facilities
TMO BaseClass
SpM BaseClass
SvM BaseClass
ODSS BaseClass
EAC Facilities
TMO BaseClass
RT I/O Func
Clock Func
Mem Func
TMO Support Library (TMOSL)
Middleware Service Interface (MSI) Function
AIT
WTST
MMCT
VLIIT
VMST
MCBClass
TNCMClass
SystemQClass
CommClass
MiddlewareStateClass
ClockServiceClass
RMMCsupport
UDPInterfaceClass
QueueClass
MemoryProxy
CTMOwinsock
TMO Support Middleware (TMOSM)
Winsock APIs
Thread APIs
OS Services
Operating System
Selected OS Services

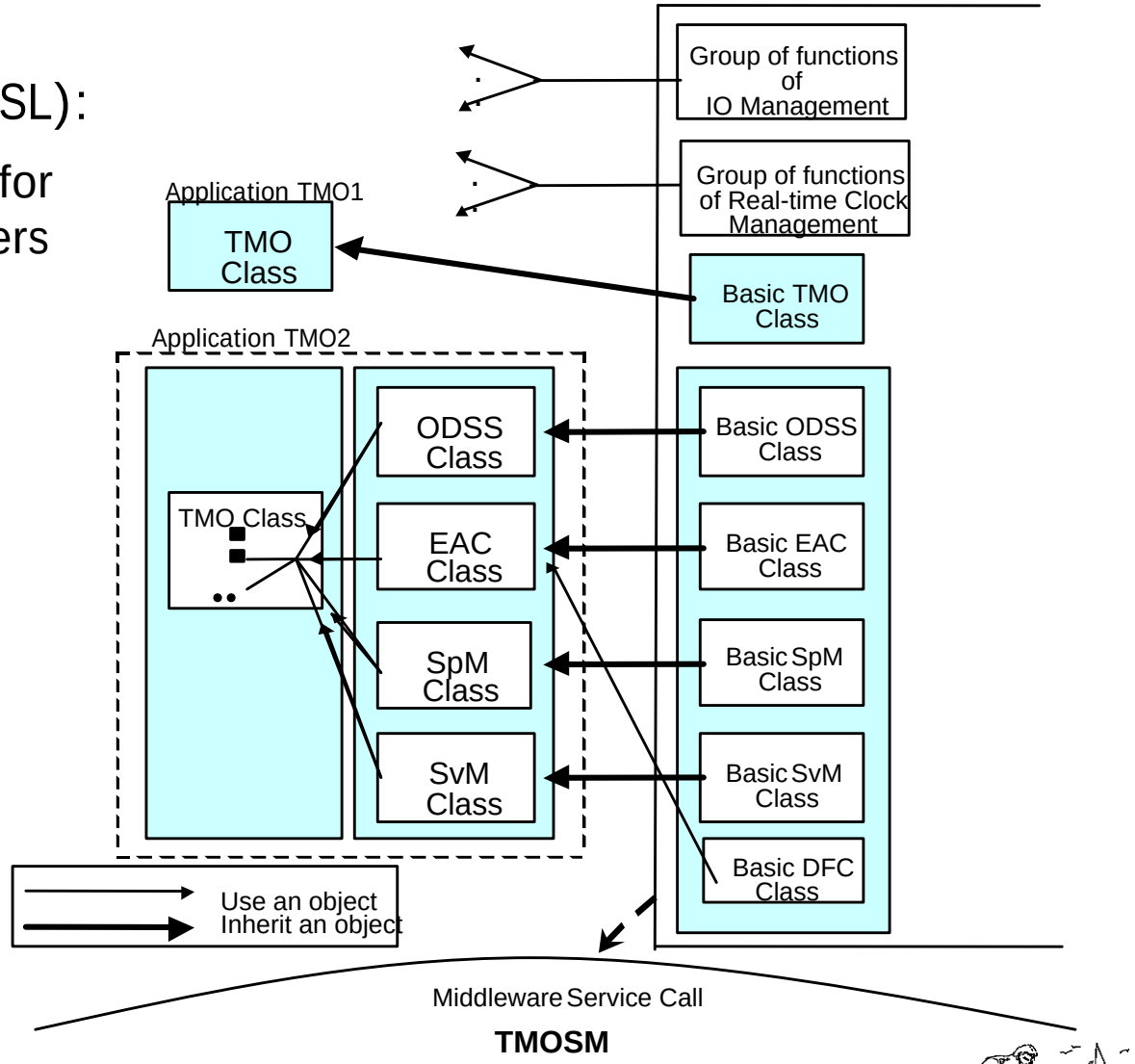
Also, NT --> WinCE



TMOSM Support Library

TMO support library (TMOSL):

User friendly API library for C++ TMO programmers



QoS-driven Resource Allocation Framework based on the RIF (Risk Incursion Function) Scheme

UCI
DREAM Lab

Introduction

- In spite of the continuing decline of computer hardware costs, allocation of computer resources is still a major issue in designing complex, real-time, computer systems.

  - In complex, real-time computer systems, the rate of component failures is not negligible.
  - In such systems, tight resource conditions can rise due to the failure of computing components.
  - Moreover, the real-time recovery of the computation disturbed by the faulty components also involves resource allocation actions.

- Many established resource allocation approaches have a fundamental limitation that they are based on the use of excessively simplistic characterizations of computation-segments competing for use of the execution resources.

  - Assigning fixed-priorities is still the most popular scheme in current practice.

Introduction

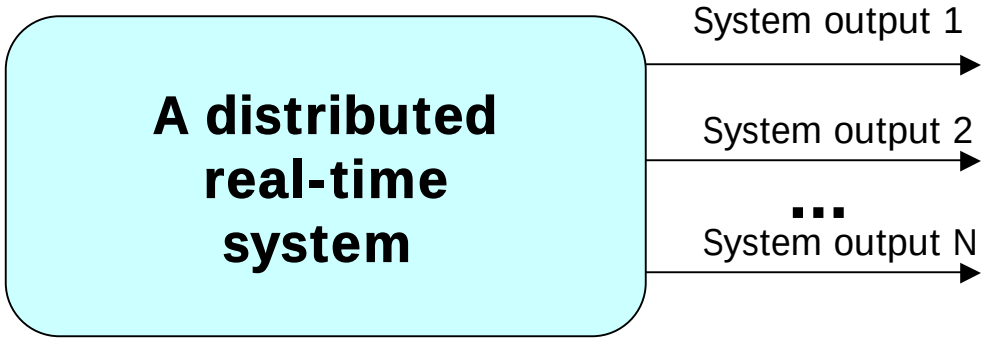
- assigning fixed priority is a very *primitive* and *crude* way of expressing the relative *importance* or *urgency* among different tasks or processes.

  - Fixed priority assignment introduces complexity for the distributed RT system design. The designer of distributed, RT systems should concentrate on high-level concepts such as computing objects, instead of considering details such as “process”, “thread”, “priority” or communication protocols.

  - Fixed priorities are the attributes that can be easily observed by the low-level node execution engine.

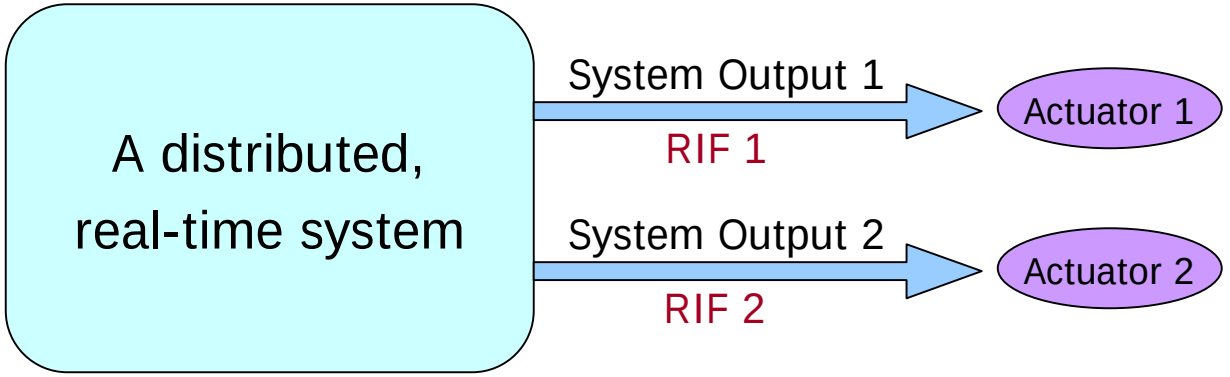
  - If there are timing requirements inherent in the target applications, it should be expressed in the simplest, easily analyzable form in the high-level system design.

Introduction



- Ultimately, execution resource requirements come from the needs of producing acceptable-quality outputs of application functions.
- The most meaningful purpose of any resource allocation is meeting the application requirements with the best quality of execution results and with minimal use of execution resources.
- An real-time computing system is required to take every service action accurately not only in “time dimension” but also in “logical dimension”.
- System design engineers must understand not only the QoS requirements (i.e., output accuracy, fault tolerance), but also the impacts of QoS losses, i.e., inaccurate outputs on the overall application success.

the Risk Incursion Function (RIF)

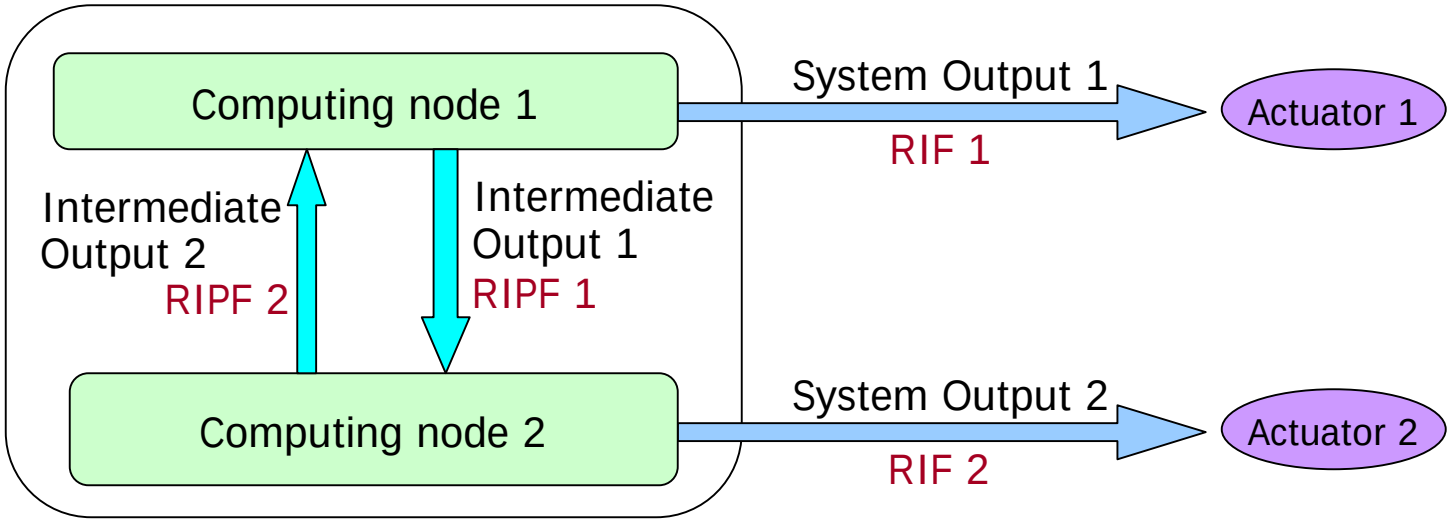


Risks - Damaging impacts of QoS losses to the application mission

RIF (a.k.a. Benefit Loss Function)

:= relation (Loss in timed value accuracy of each output action, Potential application damage)
:= relation (QoS loss, Risk)

Risk Incursion Potential Function (RIPF)



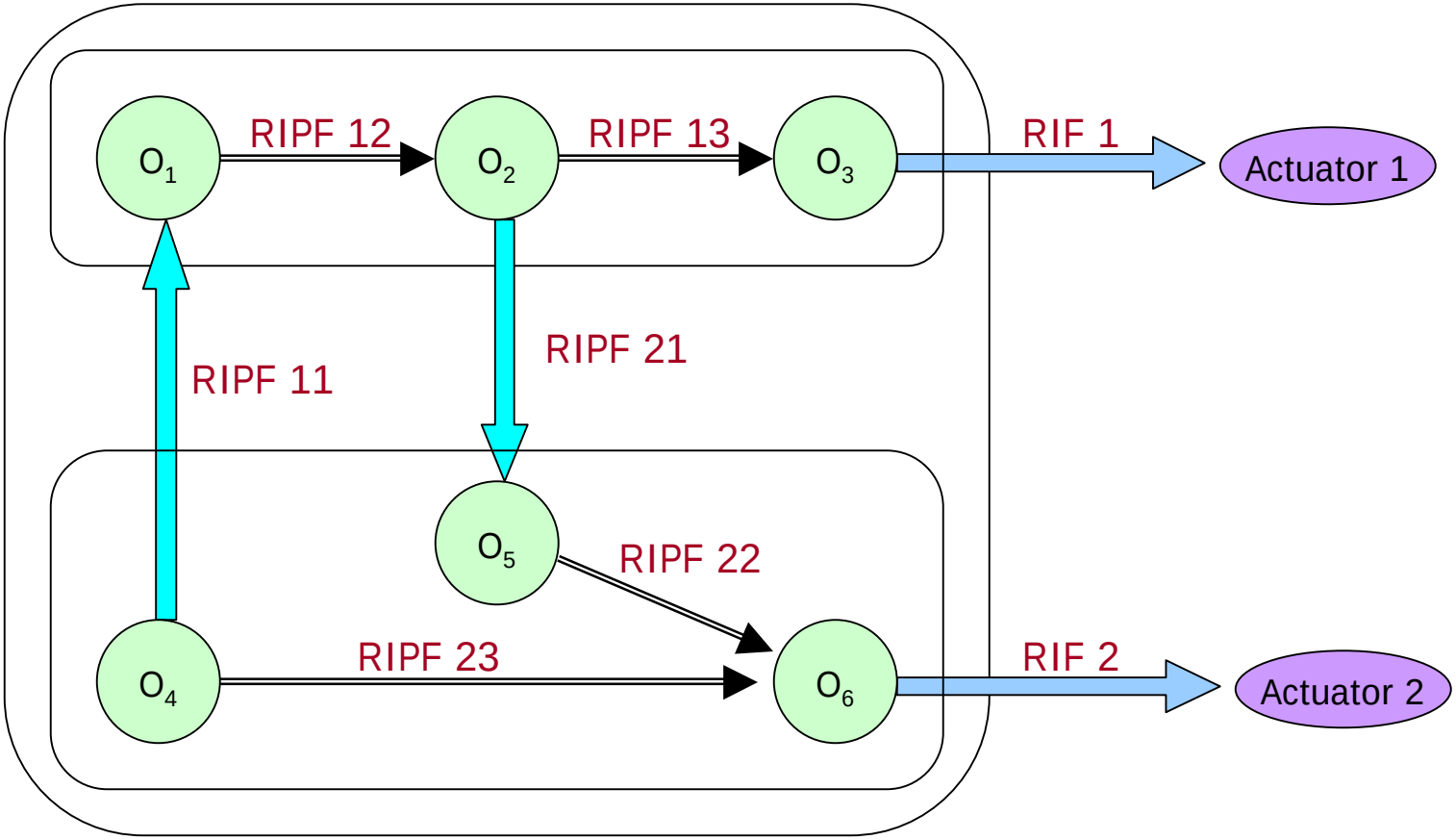
System-level RIF and derived RIF (= RIPF)

Derived RIF = RIPF (Risk Incursion Potential Function)

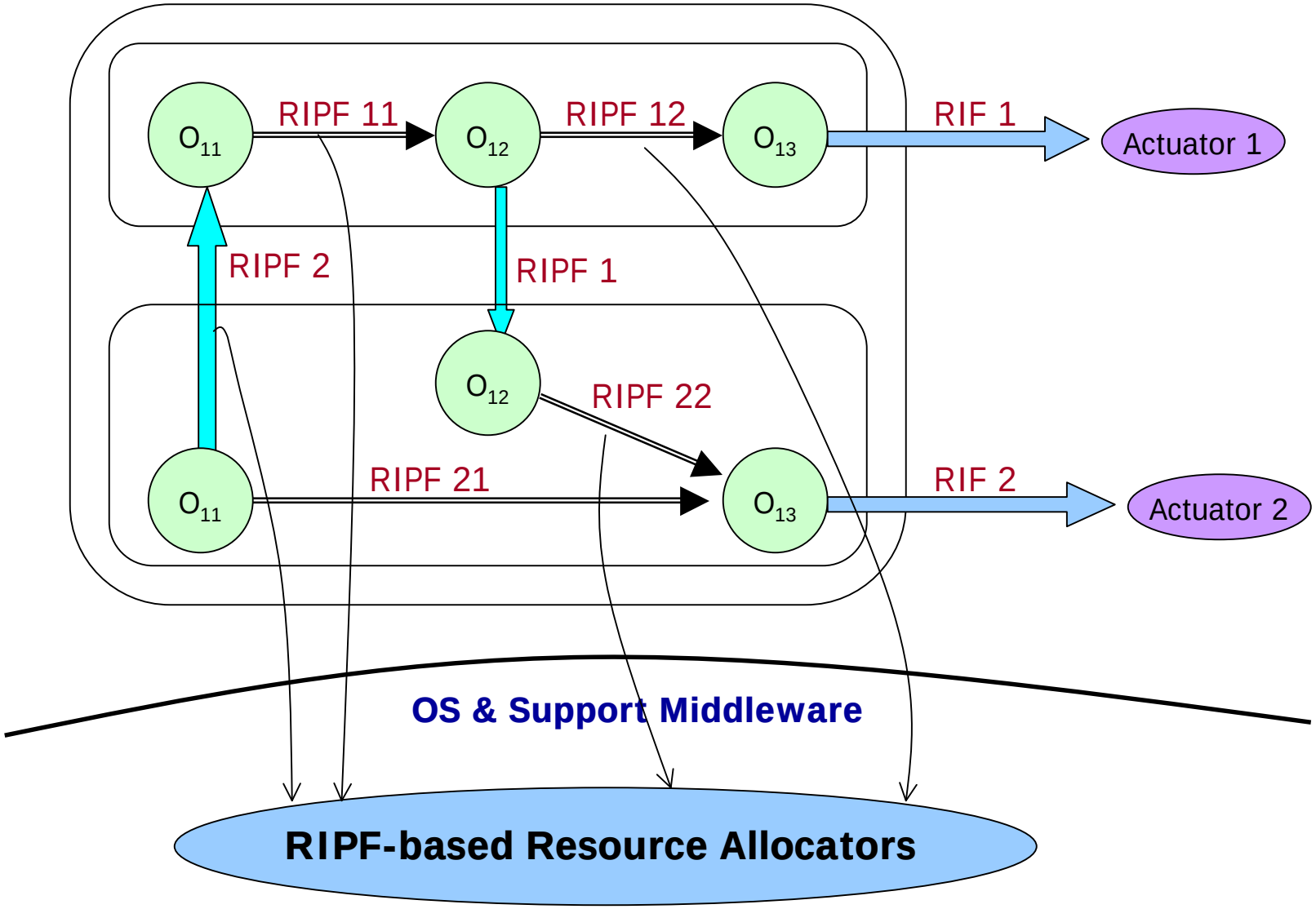
= relation (Accuracy loss in intermediate output,

Potential risk)

Risk Incursion Potential Function (RIPF)

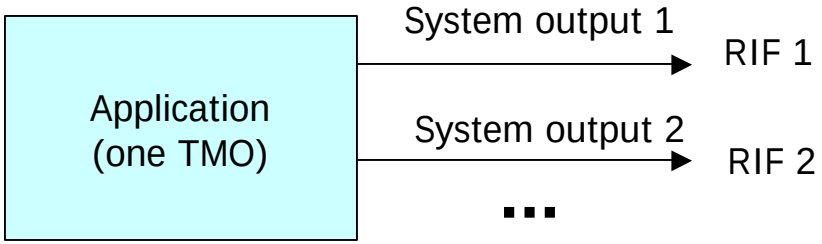


Risk Incursion Potential Function (RIPF)

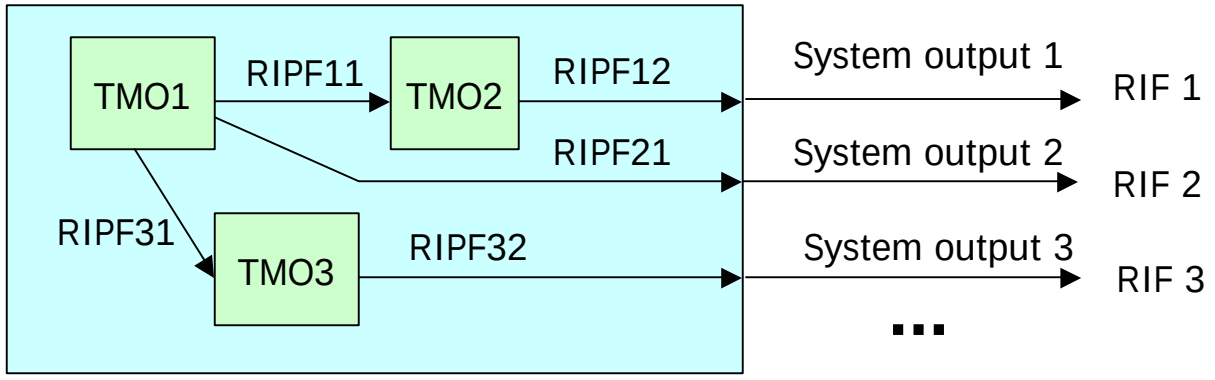


The procedure of TMO-based application development

The whole application started as one TMO

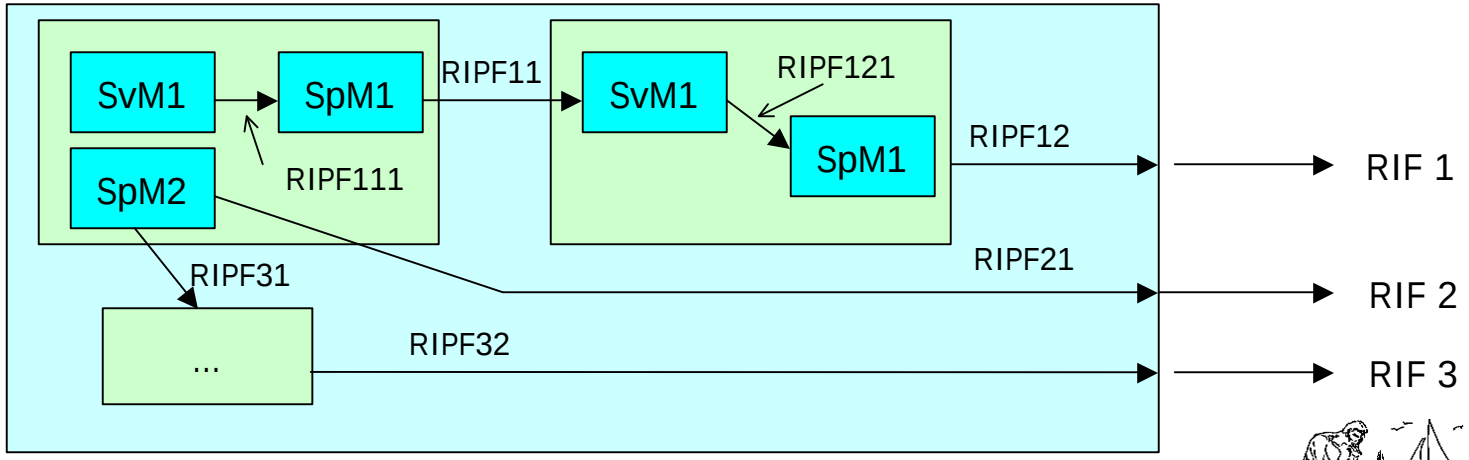


Then the TMO is divided as multiple TMO, At the same time, the RIPFs are derived from the RIFs



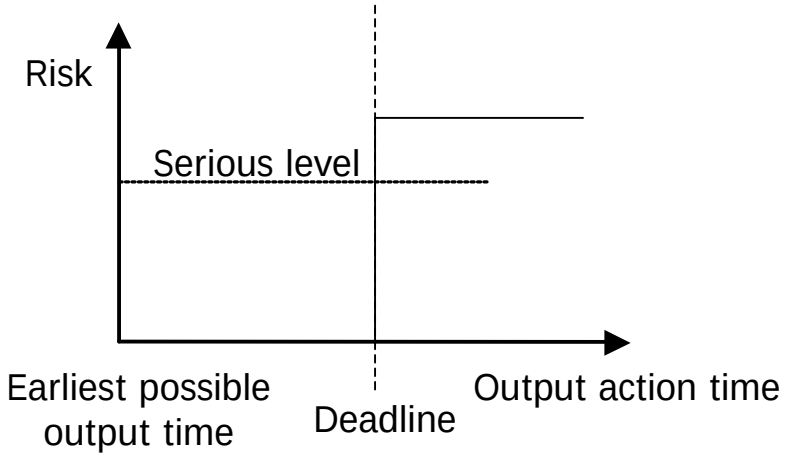
...

Final, the application is described as a TMO network (basic scheduling unit is SxM supported by a thread)

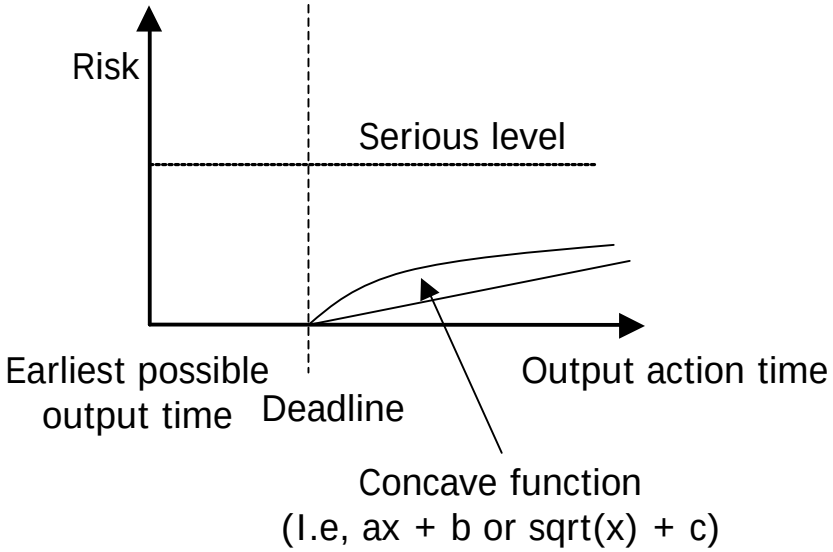


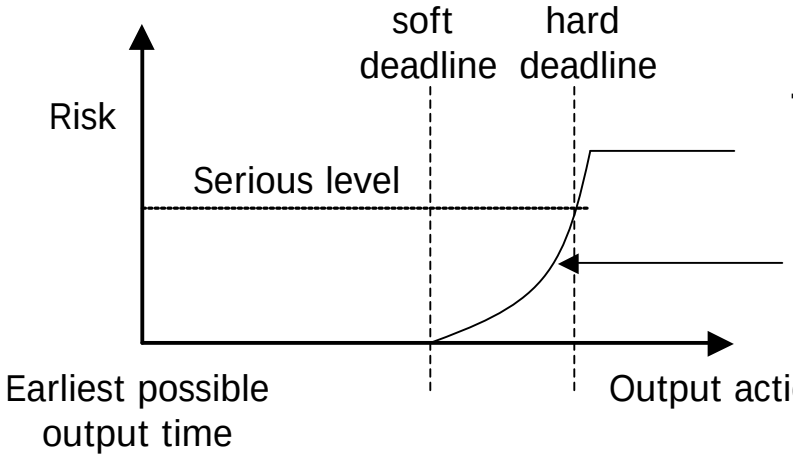
RIF (RIPF) examples

Type I: Hard Deadline



Type II: Soft Deadline





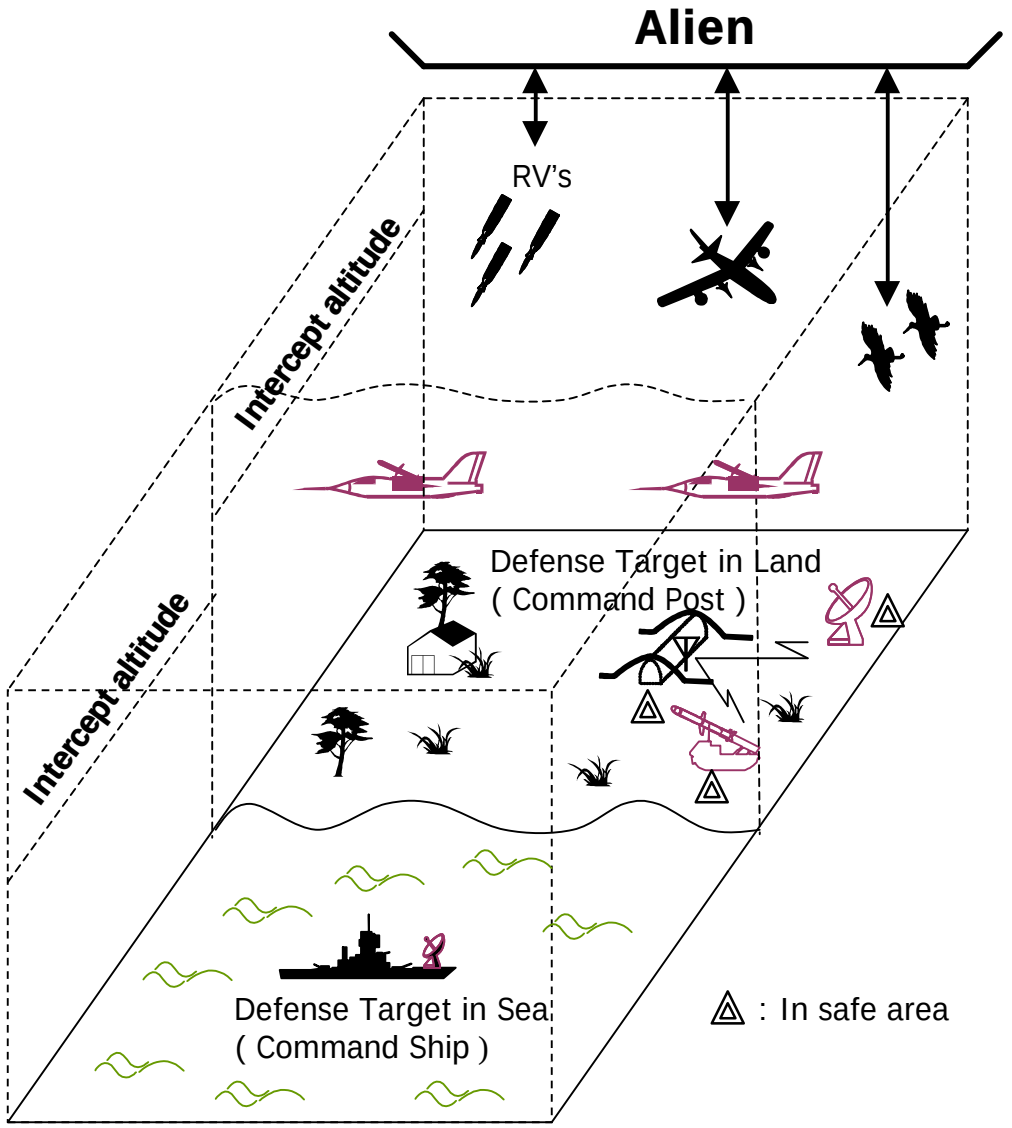
Type III: Soft deadline followed by a hard deadline

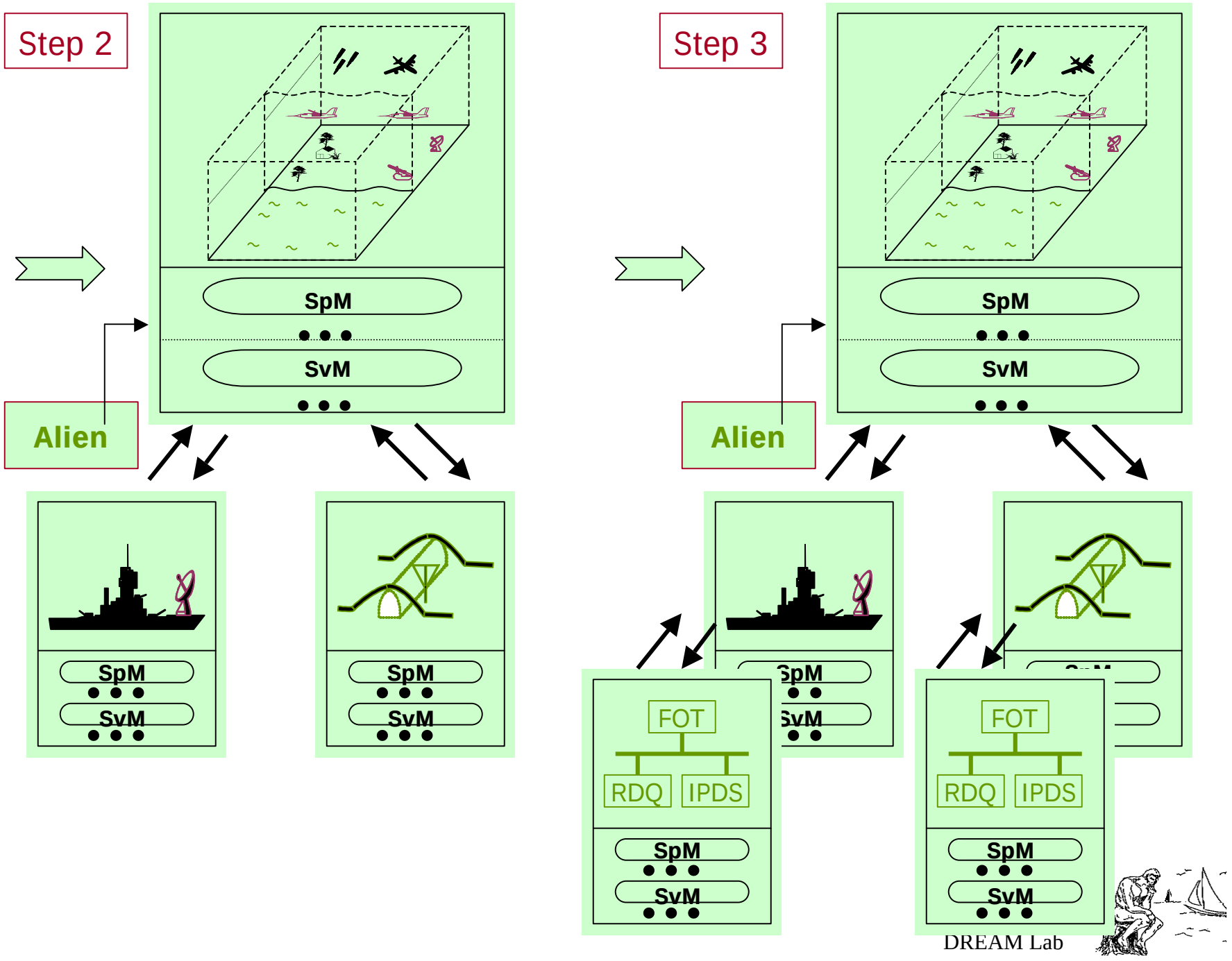
Convex function
(Polynomial function,
i.e., $ax^3 + bx^2 + cx + d$)

Case Study:

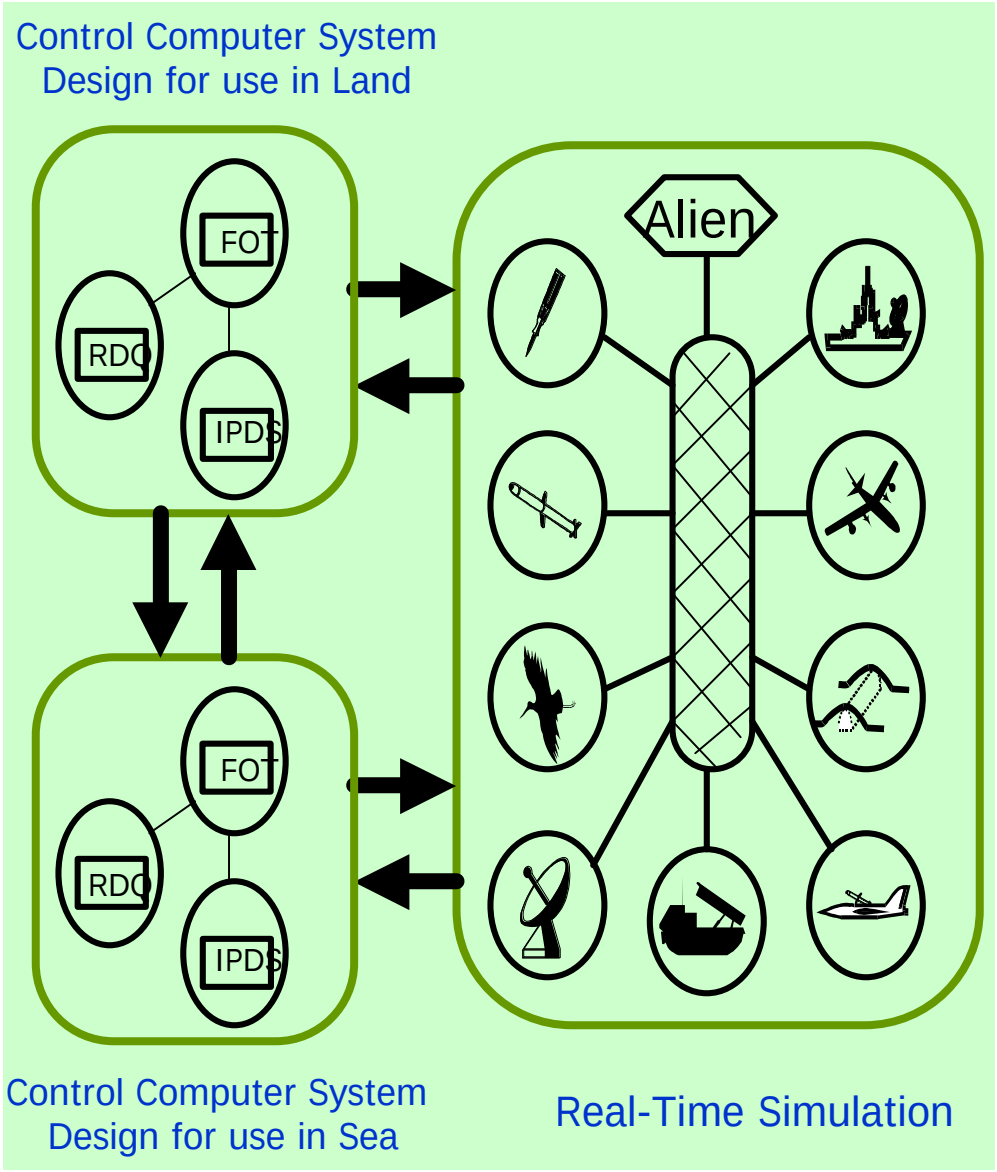
CAMIN

(Coordinated Anti-Missile Interceptor Network)
Theater




Step 2
SpM
SvM
Alien
SpM
SvM
SpM
SvM
Step 3
SpM
SvM
Alien
SpM
SvM
FOT
RDQ
IPDS
SpM
SvM
FOT
RDQ
IPDS
SpM
SvM

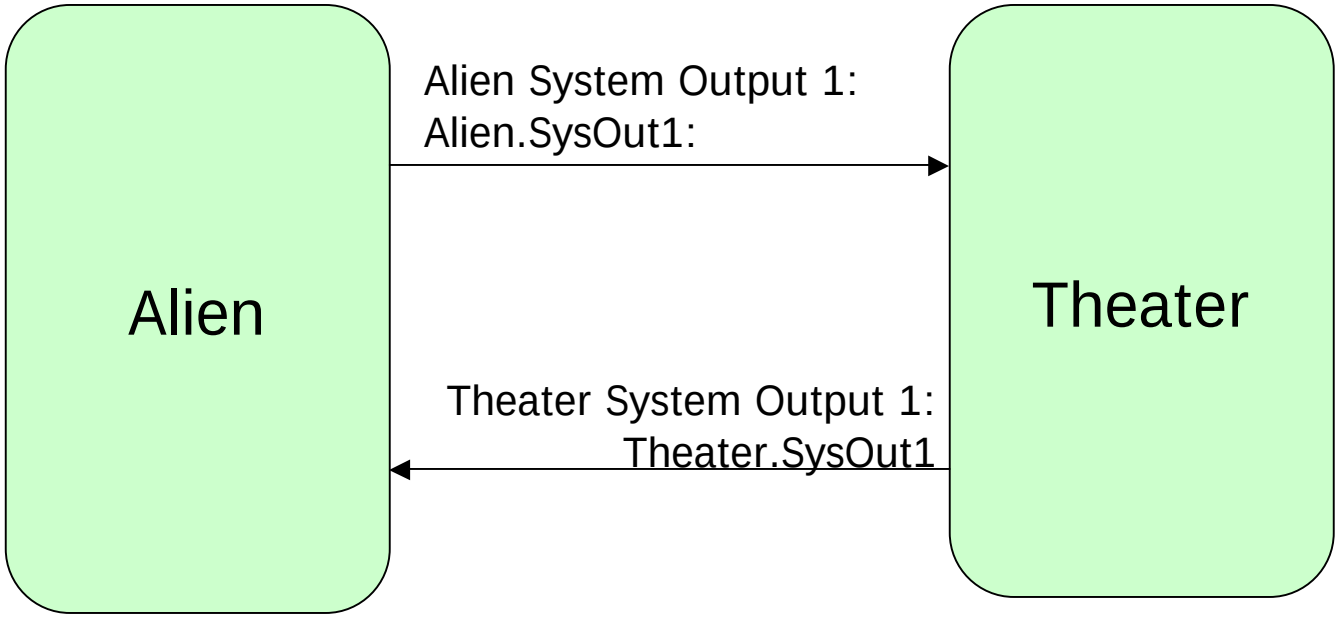
CAMIN as a network of TMO's



- Defense command-control system
- 9 TMO's; 2 TMO's made fault-tolerant
- Runs on LAN of 3+ PC's
- 25, 000 lines of C++ code

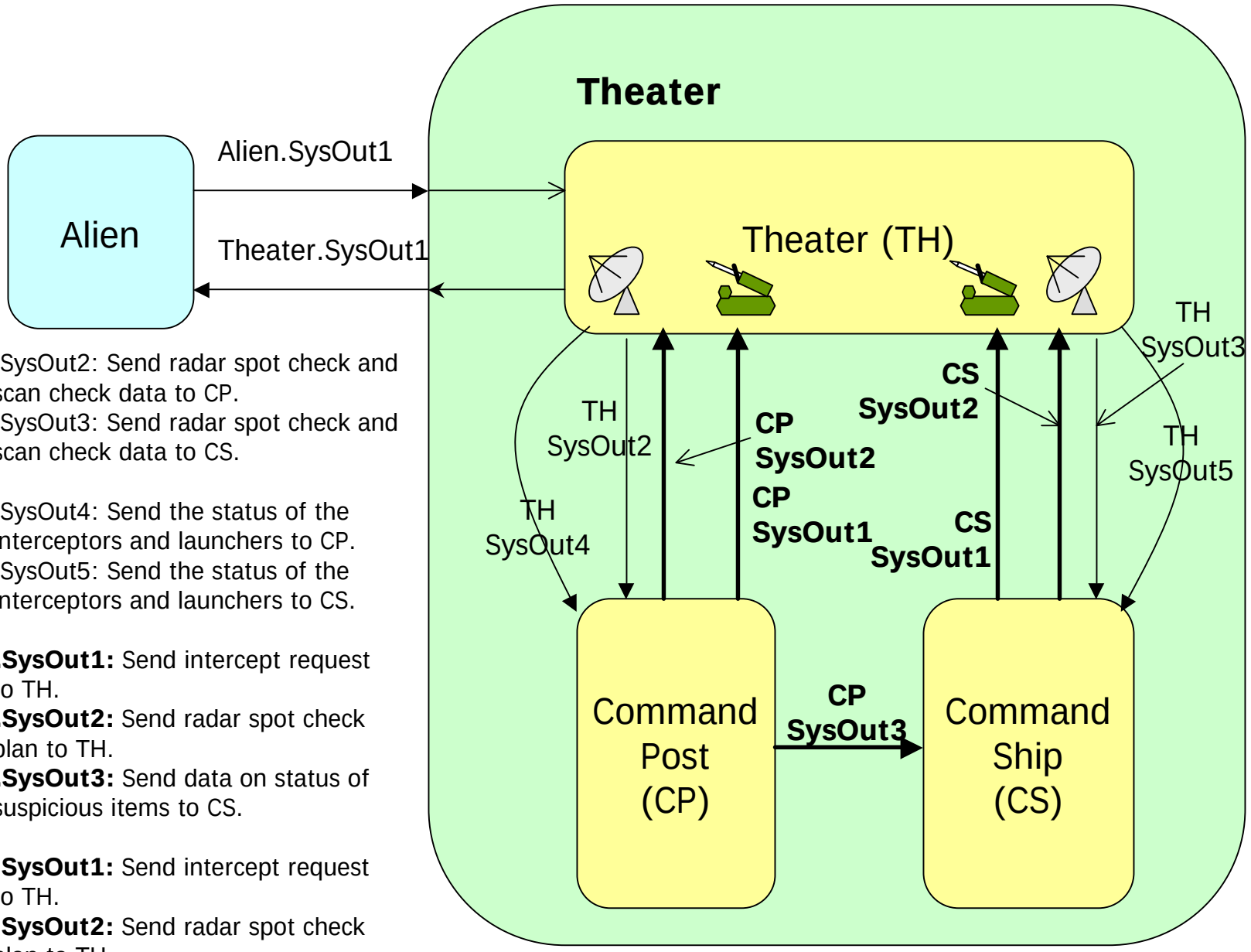
- Non-stop effective defense in the presence of
  - application software faults
  - processor faults
  - communication link (involves both software and hardware) faults
  - interconnection network (involves both software and hardware) faults

Case Study: CAMIN



Alien.SysOut1: Send reentry vehicle (missile) and NTFOs (non-threatening flying object) to the theater.

Theater.SysOut1: Send information about the defense targets to the alien;
Send current statuses of missiles and commercial airplanes leaving from the theater to the alien;



TH.SysOut2: Send radar spot check and scan check data to CP.
TH.SysOut3: Send radar spot check and scan check data to CS.

TH.SysOut4: Send the status of the interceptors and launchers to CP.
TH.SysOut5: Send the status of the interceptors and launchers to CS.

CP.SysOut1: Send intercept request to TH.
CP.SysOut2: Send radar spot check plan to TH.
CP.SysOut3: Send data on status of suspicious items to CS.

CS.SysOut1: Send intercept request to TH.
CS.SysOut2: Send radar spot check plan to TH.

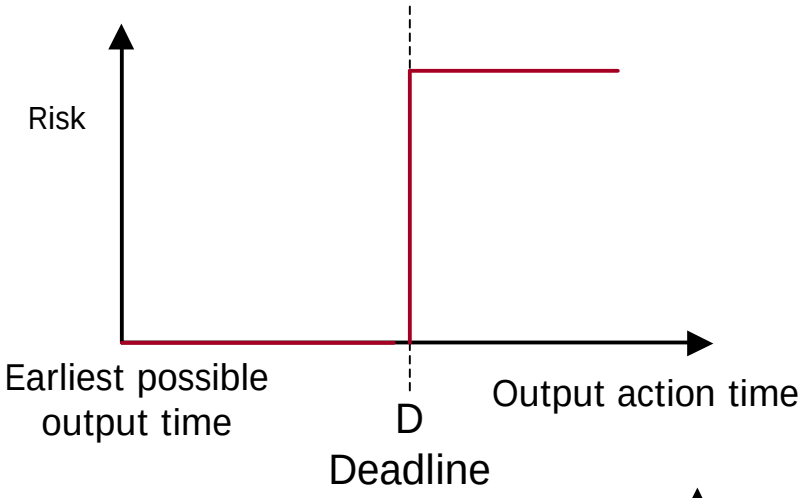
CP.SysOut1

RIF_CP1:

y = 0 if x ≤ D

or

400 if x > D



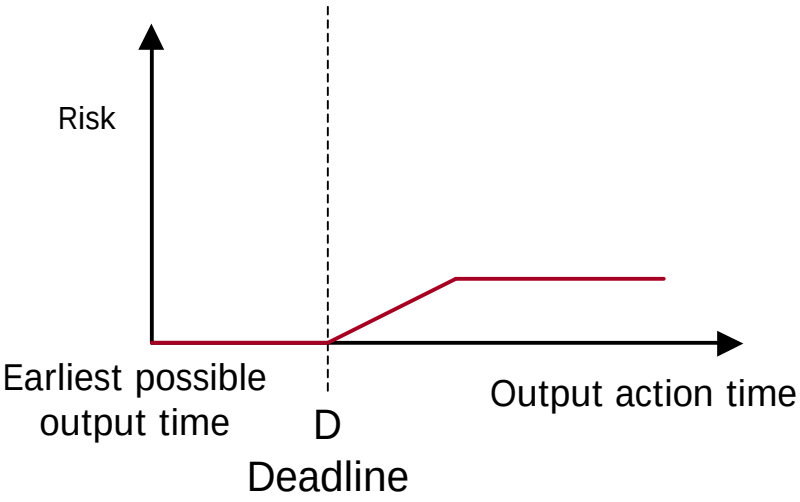
CP.SysOut2

RIF_CP2:

y = 0 if x ≤ D

x – D if D < x ≤ D + 50

50 if x > D + 50



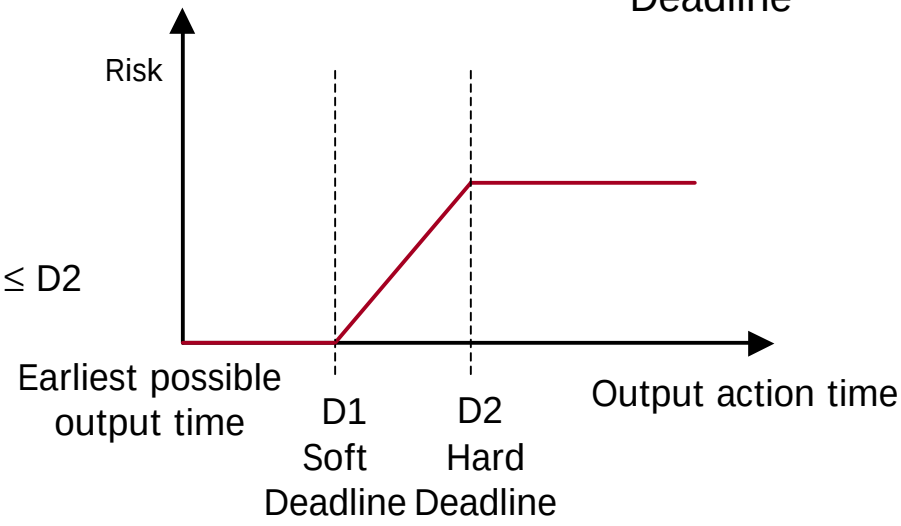
CP.SysOut3

RIF_CP3:

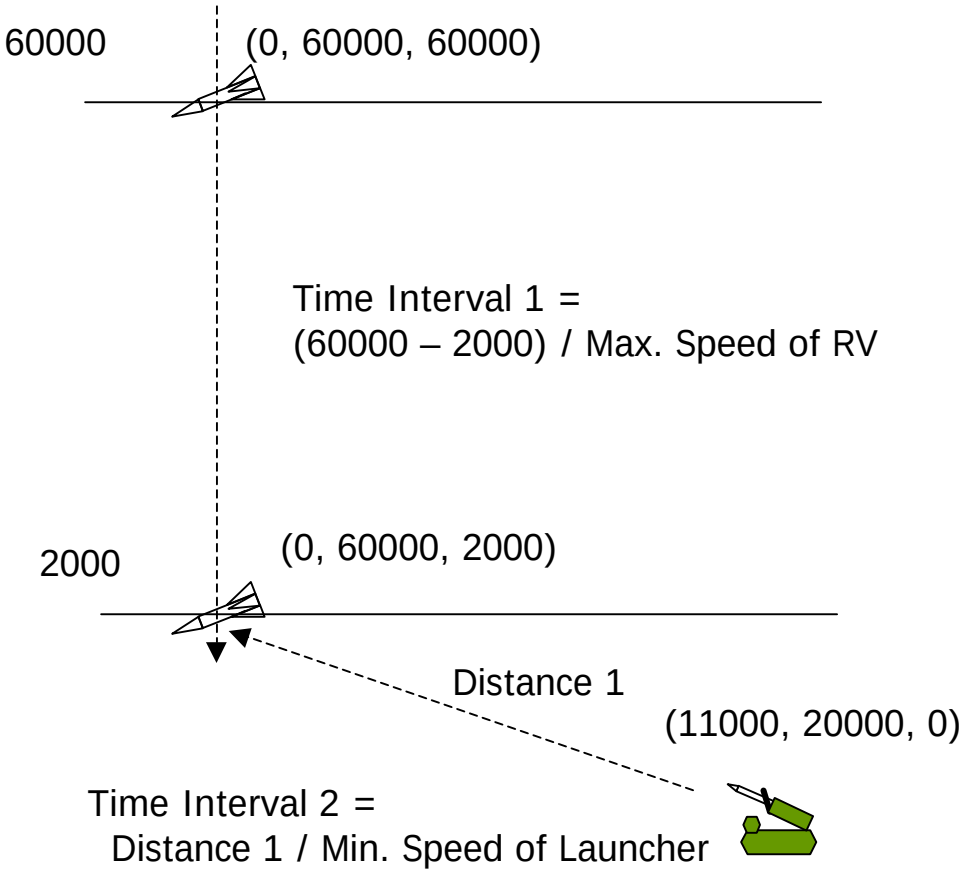
y = 0 if x ≤ D1

5(x – D1) if D1 < x ≤ D2

200 if x > D2



Constraints for the deadline of CP.SysOut1

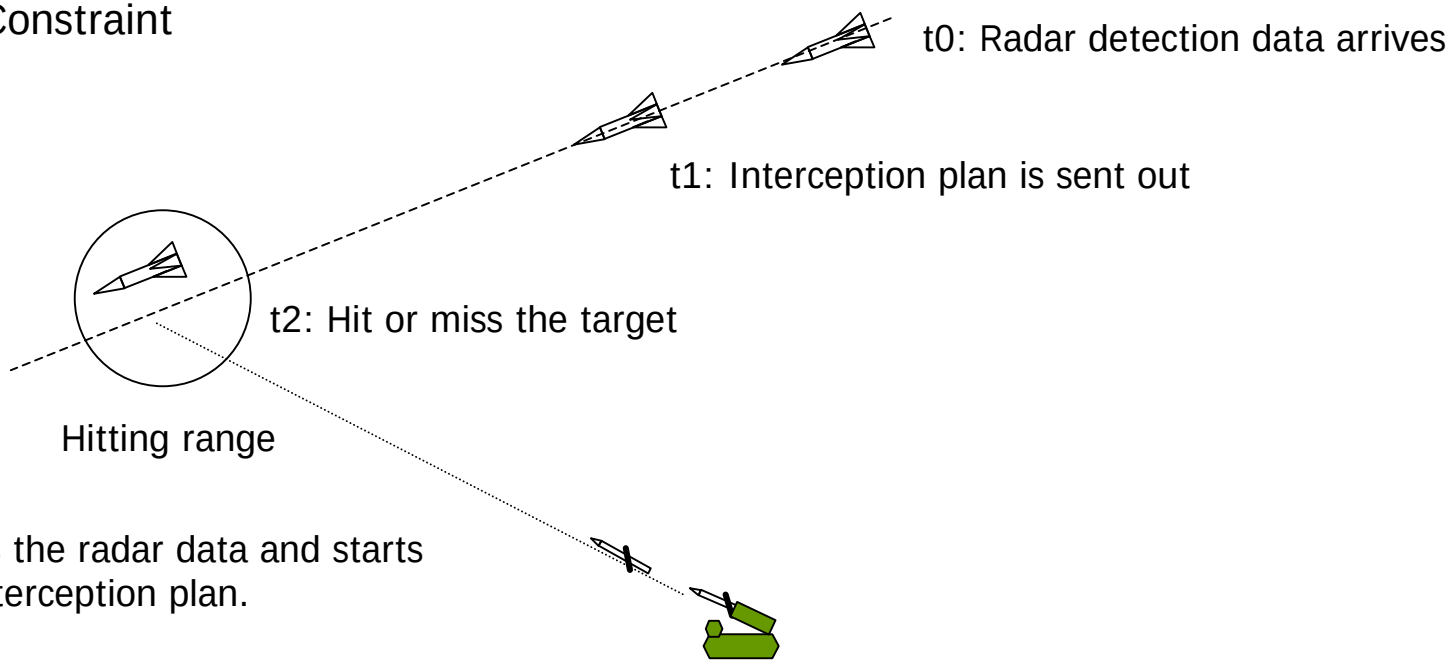


1. Spatial Constraint

Time Interval 1 – Time Interval 2

Constraints for the deadline of CP.SysOut1

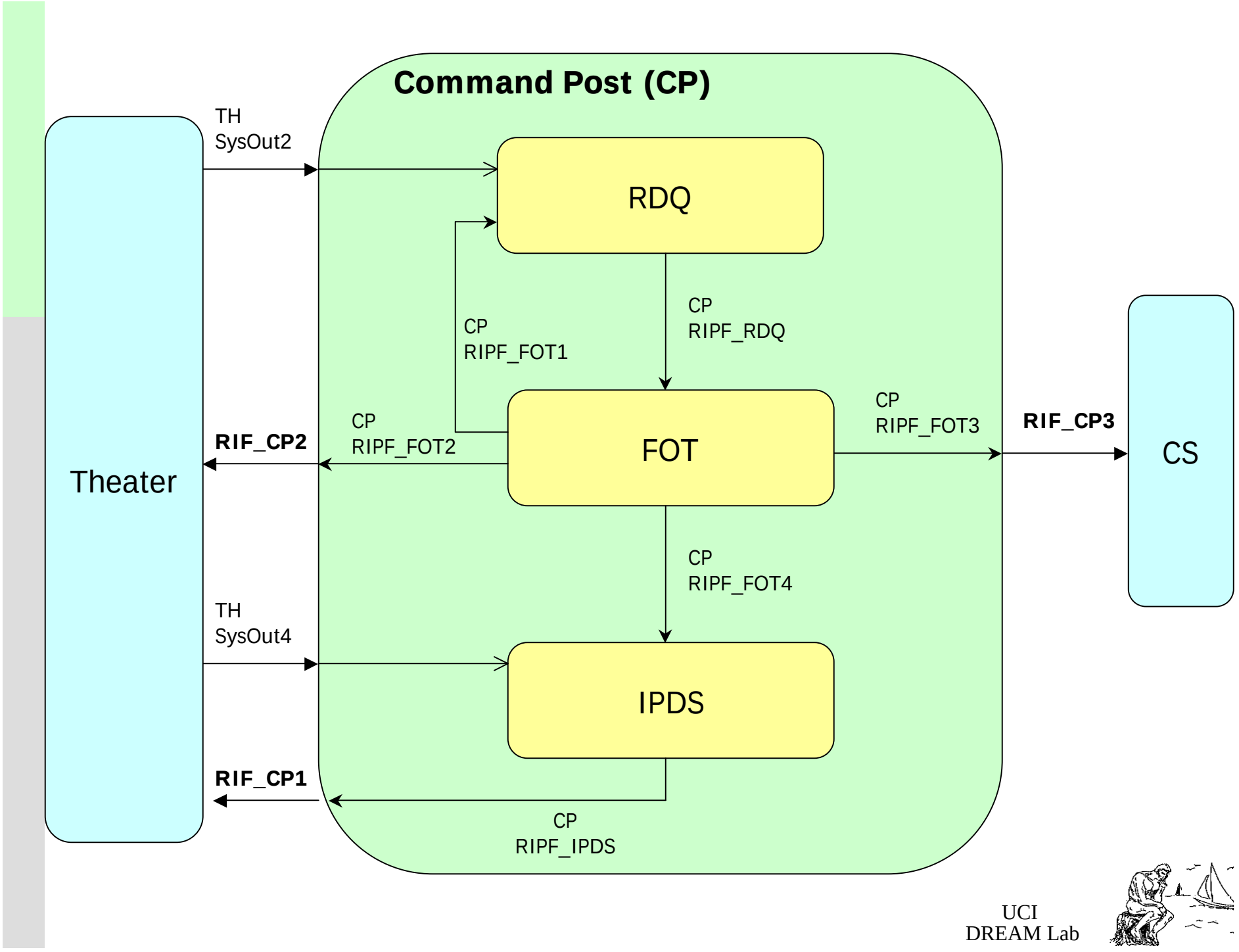
2. Temporal Constraint

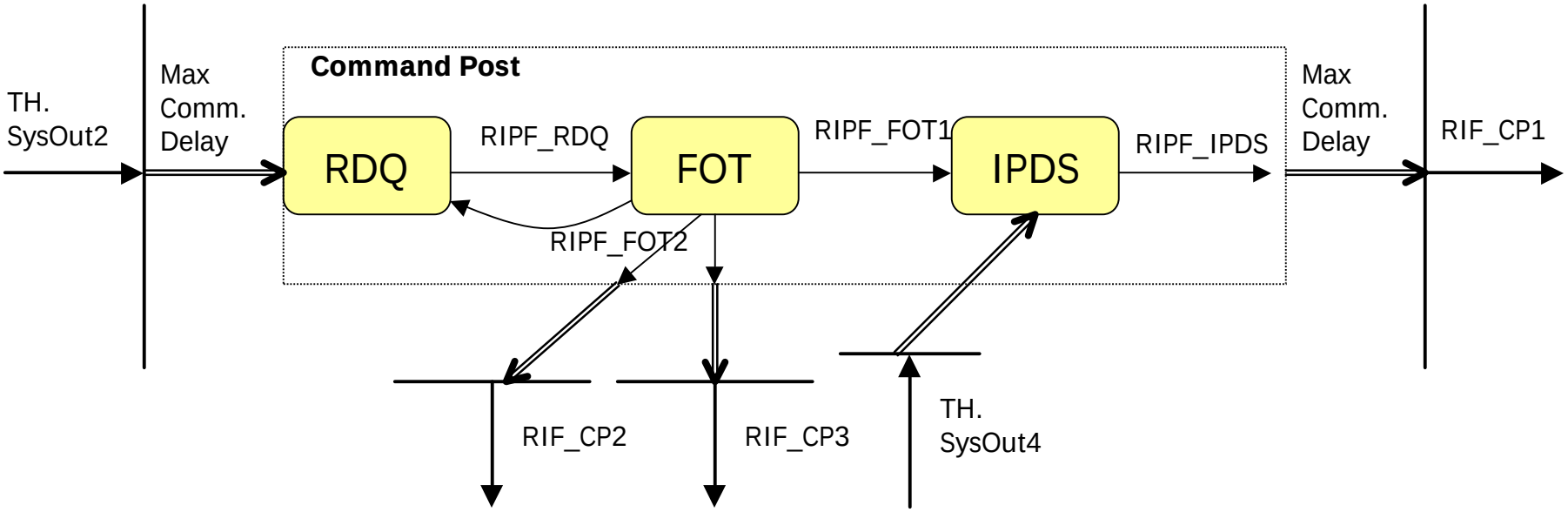


t0: CP receives the radar data and starts Building the interception plan.

t1: CP sends out the interception plan. The position Of the missile in t1 is extrapolated from the data of t0. While the t1 – t0 becomes bigger, the accuracy of the extrapolation becomes worse.

t2: If the missile is in the hitting range of the interceptor, the interception is successful. The success rate depends on the accuracy of the extrapolation at t1.


Command Post (CP)
Theater
TH
SysOut2
RDQ
CP
RIPF_RDQ
CP
RIPF_FOT1
CP
RIPF_FOT2
RIF_CP2
FOT
CP
RIPF_FOT3
RIF_CP3
CS
CP
RIPF_FOT4
TH
SysOut4
IPDS
RIF_CP1
CP
RIPF_IPDS

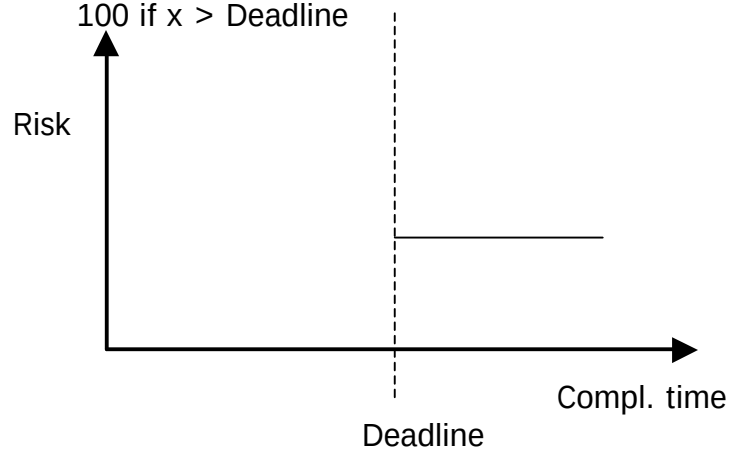


CP.SysOut1: RIF

y = 0 if x <= Deadline

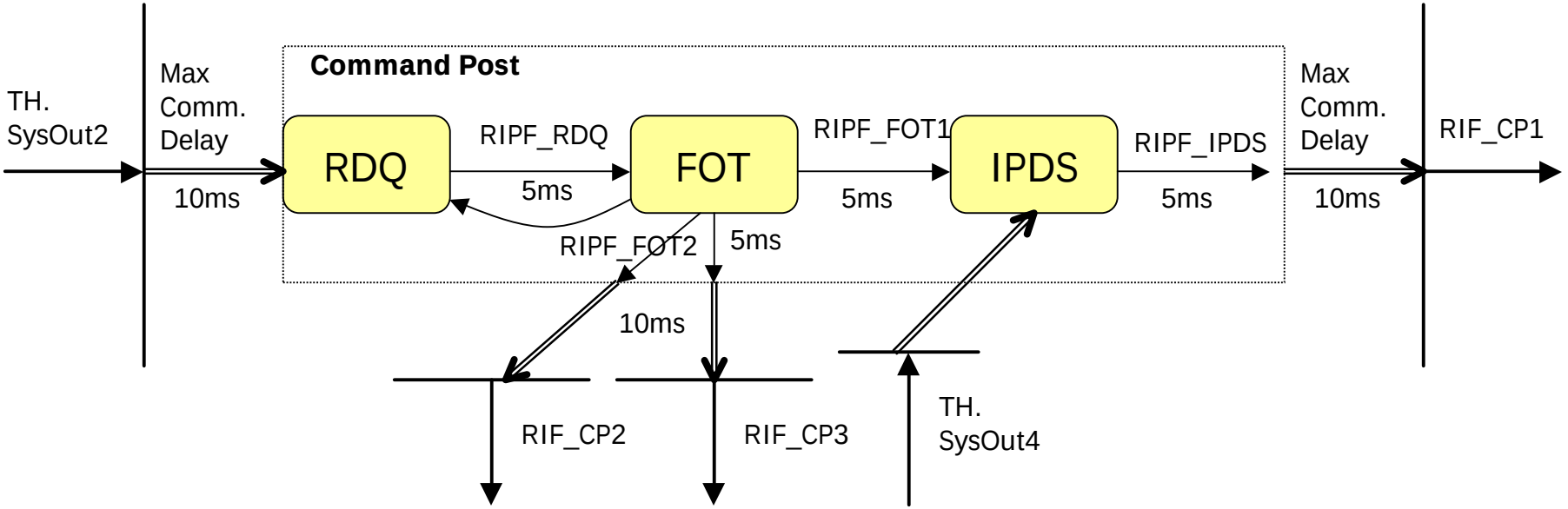
or

100 if x > Deadline



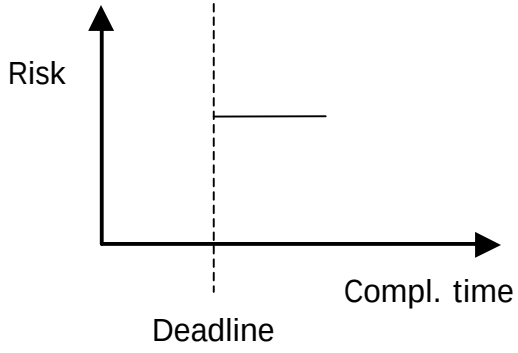
- The derivation of RIPF from RIF is based on the worst case execution time (WCET) analysis and the importance of each task.
- Let assume the maximum inter-TMO (intra-node) comm. delay is 5ms, inter-node comm. delay is 10ms. Let also assume RDQ, FOT and IPDS are running in the same node.
- In this design example, suppose we conclude that the deadline of CP.SysOut1 should be 200ms, and CP.SysOut2 and CP.SysOut3 should be 100ms. After analyzing the WCETs of RDQ, FOT and IPDS, we allocate this 200ms as follows:
  RDQ (25ms) FOT (50ms) IPDS (90ms)

- Since RDQ and FOT are related to all of the three system outputs, while IPDS is related with only system output 1, we set the threshold of deadline violation as follows:
  RDQ (80), FOT(80), IPDS (40)



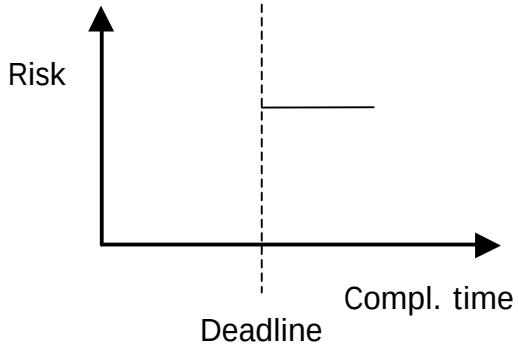
CP.RIPF_RDQ

y = 0 if x <= 25ms
or
80 if x > 25ms



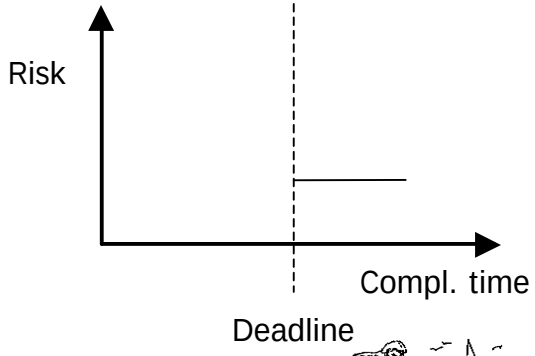
CP.RIPF_FOT1

y = 0 if x <= 50ms
or
80 if x > 50ms



CP.RIPF_RDQ

y = 0 if x <= 90ms
or
40 if x > 90ms

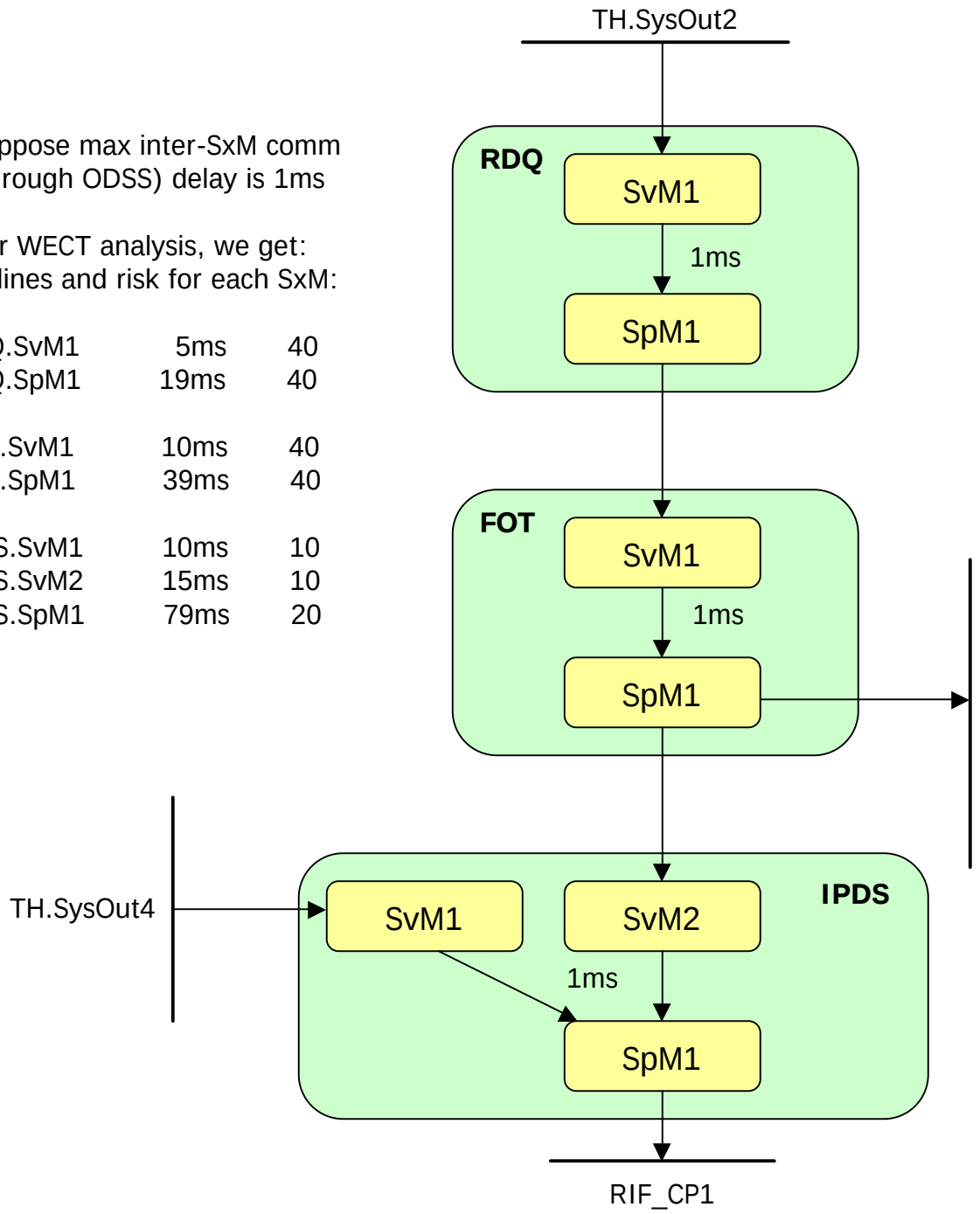


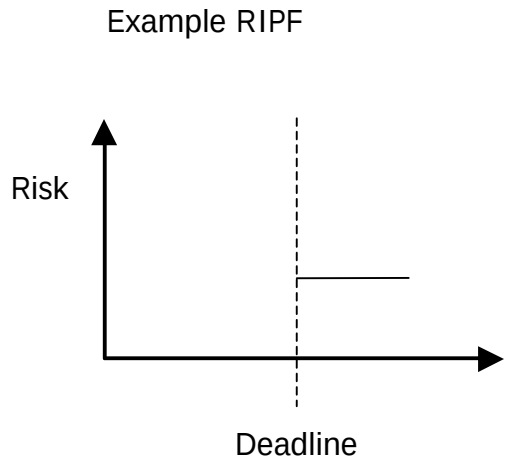
UCI
DREAM Lab

. Suppose max inter-SxM comm (through ODSS) delay is 1ms

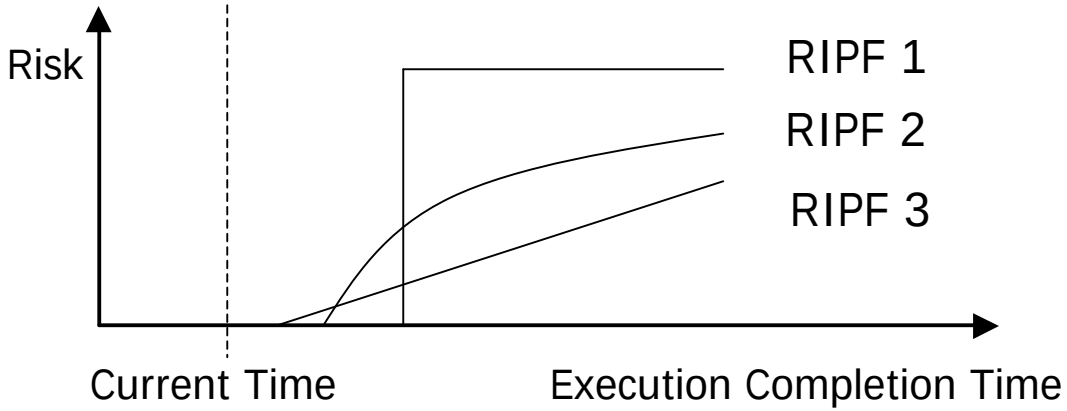
After WECT analysis, we get:
Dealines and risk for each SxM:

RDQ.SvM1	5ms	40
RDQ.SpM1	19ms	40
FOT.SvM1	10ms	40
FOT.SpM1	39ms	40
IPDS.SvM1	10ms	10
IPDS.SvM2	15ms	10
IPDS.SpM1	79ms	20





RIPF-driven CPU scheduling

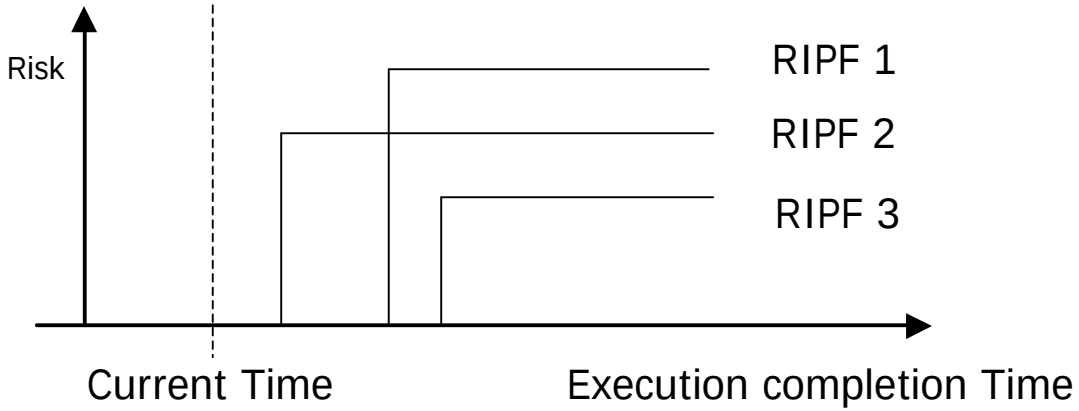


The optimal algorithm is NP-hard

Theorem 1: The optimal (lowest-total-risk) scheduling algorithm based on the proposed RIPF set is NP-hard

RIPF-driven CPU scheduling

Resource allocation problem 1



Theorem 1: Finding the optimal (lowest-total-risk) scheduling algorithm based on the proposed RIPF set is NP-hard

Proof: 1. The inexact 0-1 knapsack problem is known to be NP-hard;

Maximize $\sum_{i=1}^{n} v_i$

subject to: $\sum_{i=1}^{n} R_i < R$

Where there are n objects each with size R_i and value v_i, and R is the size of the knapsack. Both R_i and V_i are real number.

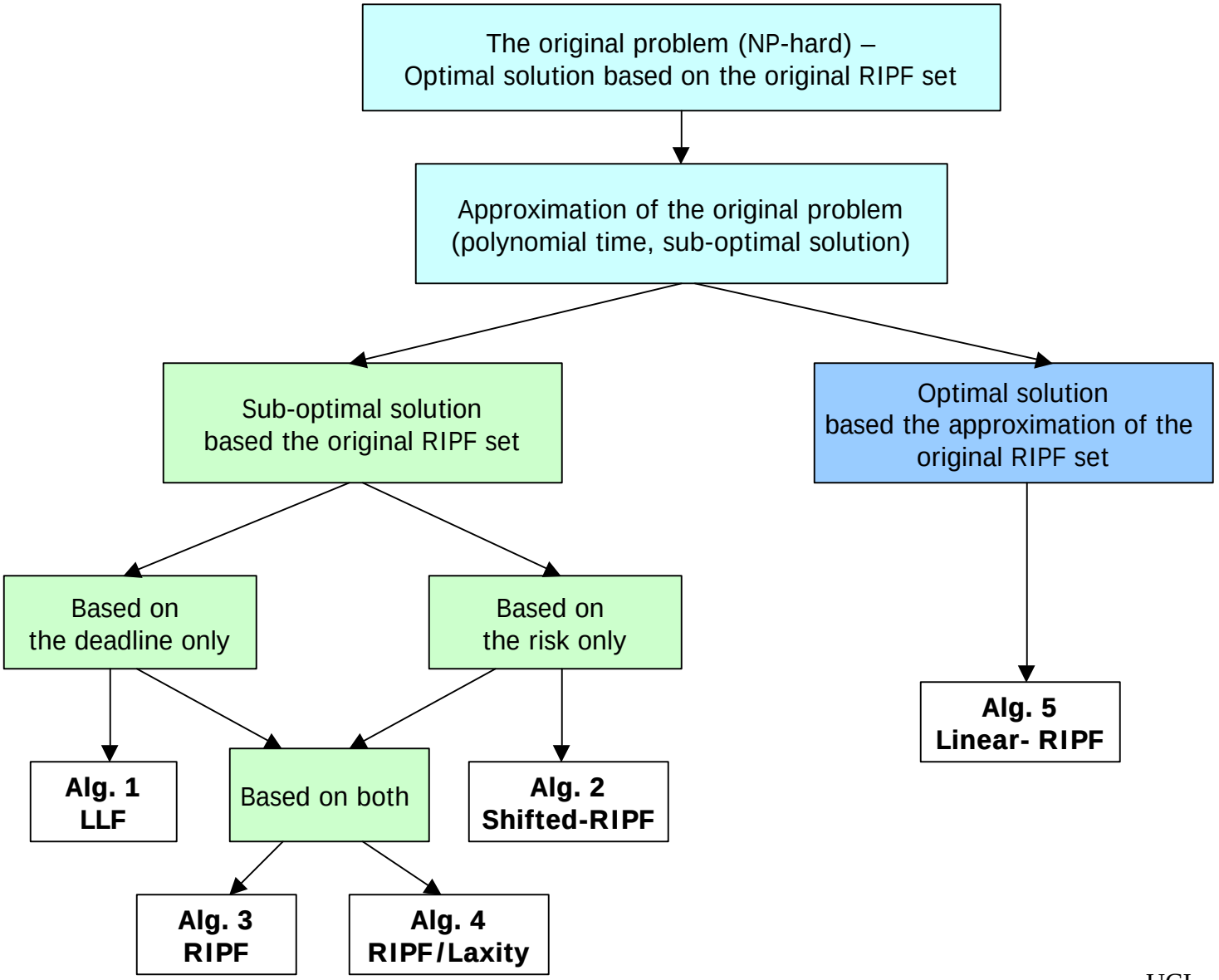
2. The above problem is equal to a special case of the problem 1, which is:

   F(x) = 0, if $x < R_i$ or

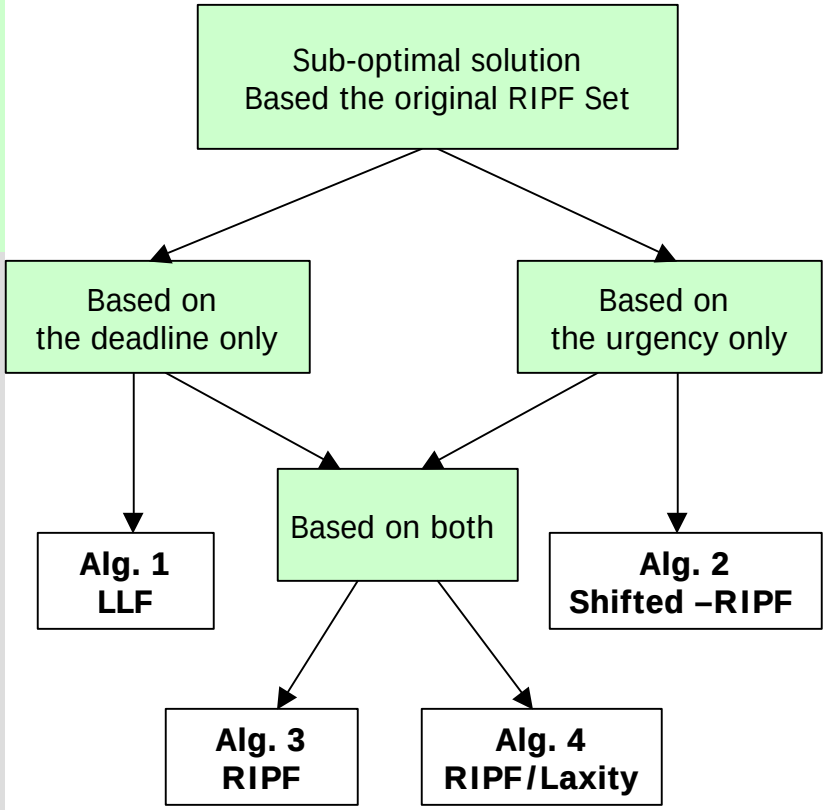
   V_i, if $x > R_i$

3. Therefore, the complexity of the problem 1 is NP-hard.

RIPF-driven CPU scheduling



RIPF-driven CPU scheduling

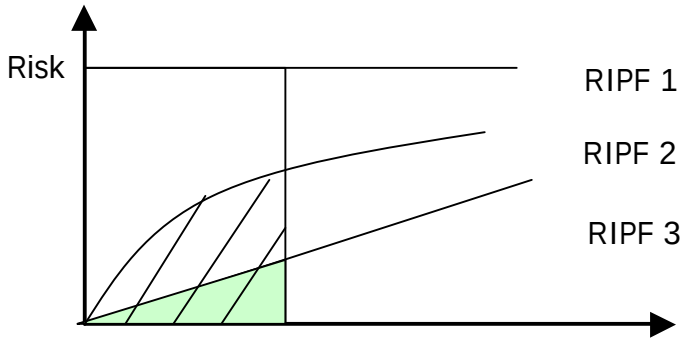


Alg. 1- LLF
O(nlgn)

- Least laxity First

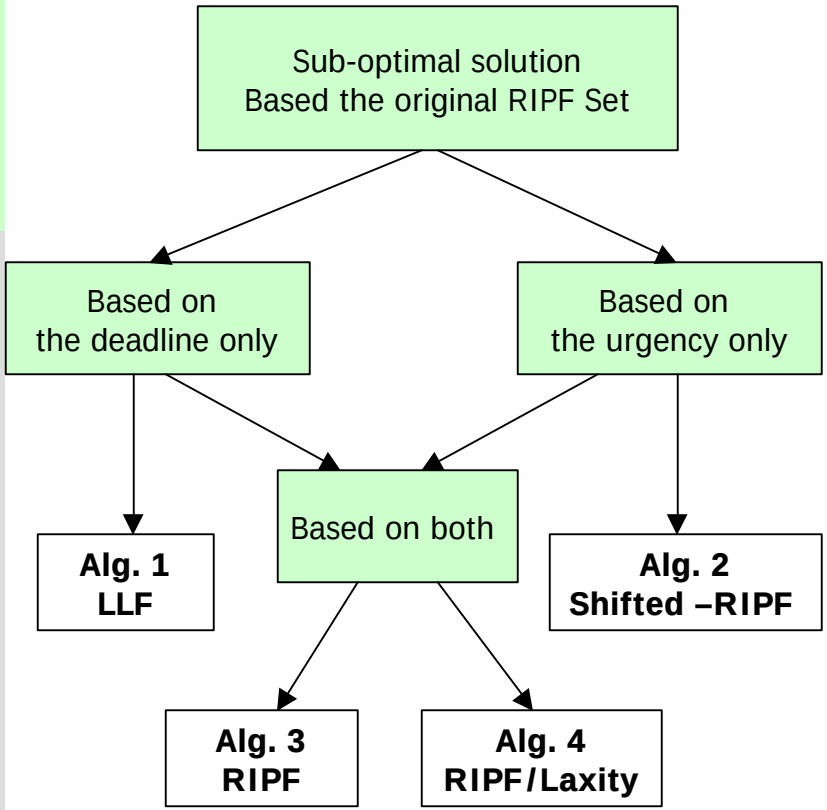
Alg. 2 –
Shifted-RIPF
O(nlgn)

- Move all RIPF 's deadline to 0.
- Compare the integration of the RIPF within current timeslice, schedule the highest one. If there are more than one highest, pick one randomly.



The integration
Of an RIPF within
One timeslice

RIPF-driven CPU scheduling

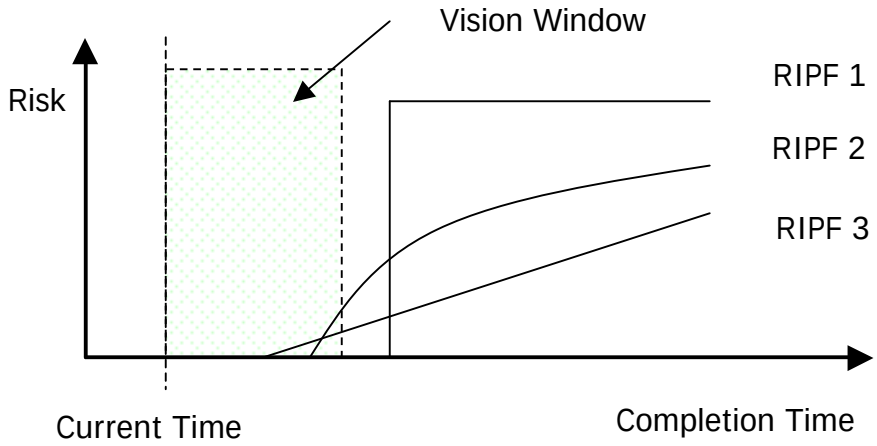


Alg. 3 – RIPF
O(nlgn)

- Run alg. 1 first, if zero risk arrangement is found, use it and return; Otherwise go to next step;
- Calculate the integrations of RIPFs within the next N timeslice (vision window). Schedule the one with the highest value.

Alg. 4 -RIPF/Laxity
O(nlgn)

- Run alg. 1 first, if zero risk arrangement is found, use it and return; Otherwise go to next step;
- Calculate the integrations of RIPFs within the next N timeslice (vision window), then divide it by Laxity. Schedule the one with the highest value.



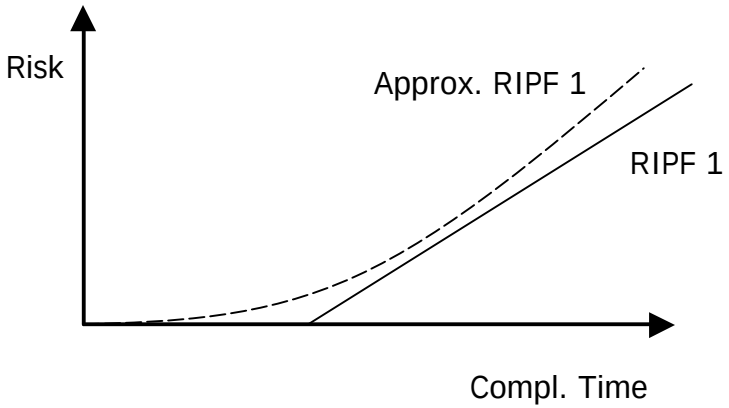
RIPF-driven CPU scheduling

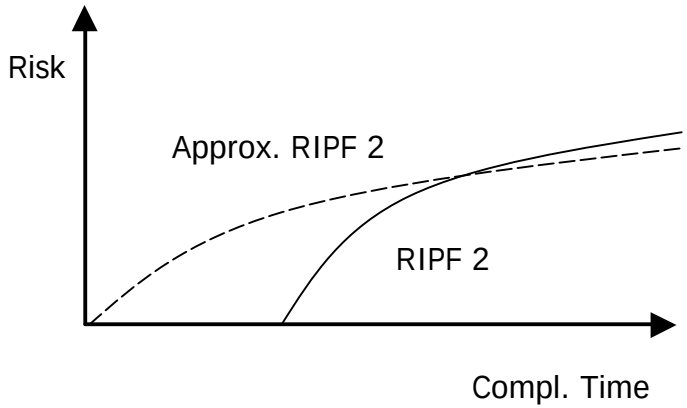
Optimal solution
Based the approximation of the original RIPF Set

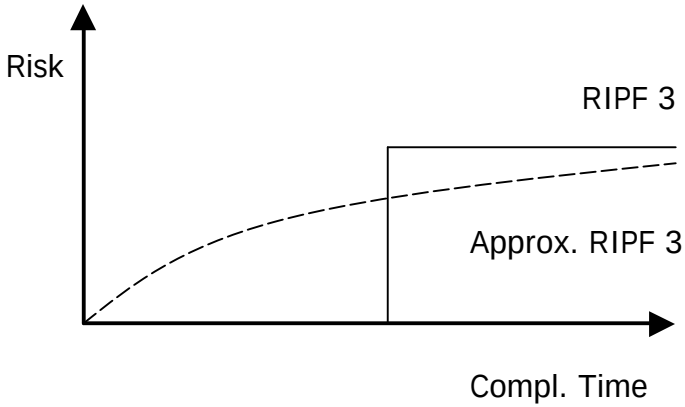
Mathematical Approximation of the original RIPF with a function that:

- monotonically increasing (f'(x) >0);
- continuous.

Alg. 4
Linear- RIPF







RIPF-driven CPU scheduling

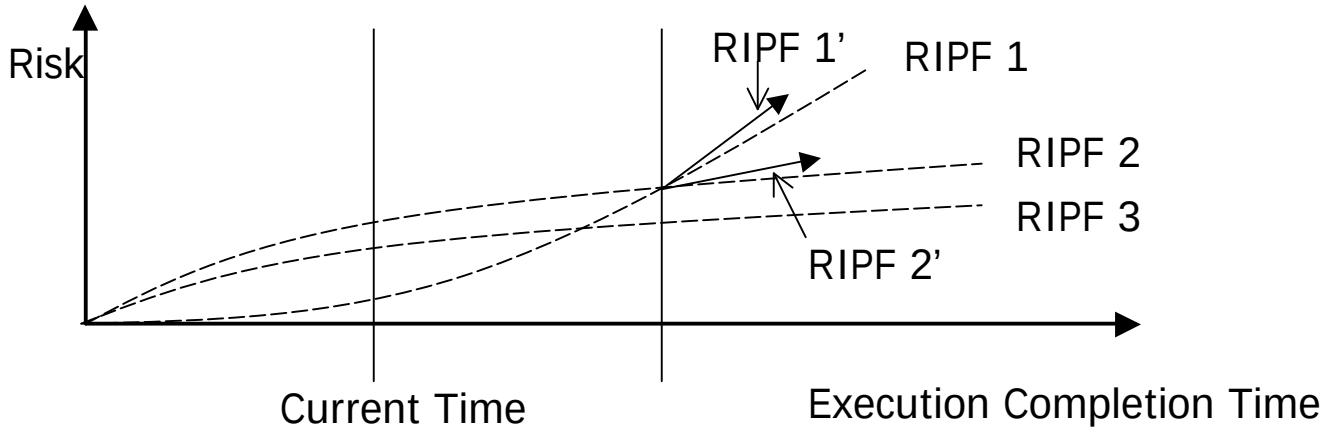
Optimal solution
Based the approximation of the original RIPF Set

Mathematical Approximation of the orignal RIPF with a function that:

- monotonically increasing (f'(x) >0);
- continuous.

Algorithm

- Compare the current value of the RIPF, pick the highest one to schedule;
- If more than one RIPF's have the highest value, compare the first derivative RIPF', and pick the highest one.



RIPF-driven CPU scheduling

Optimal solution
Based the approximation of the original RIPF Set

Mathematical Approximation of the original RIPF with a function that:

- monotonically increasing (f'(x) >0);
- continuous.

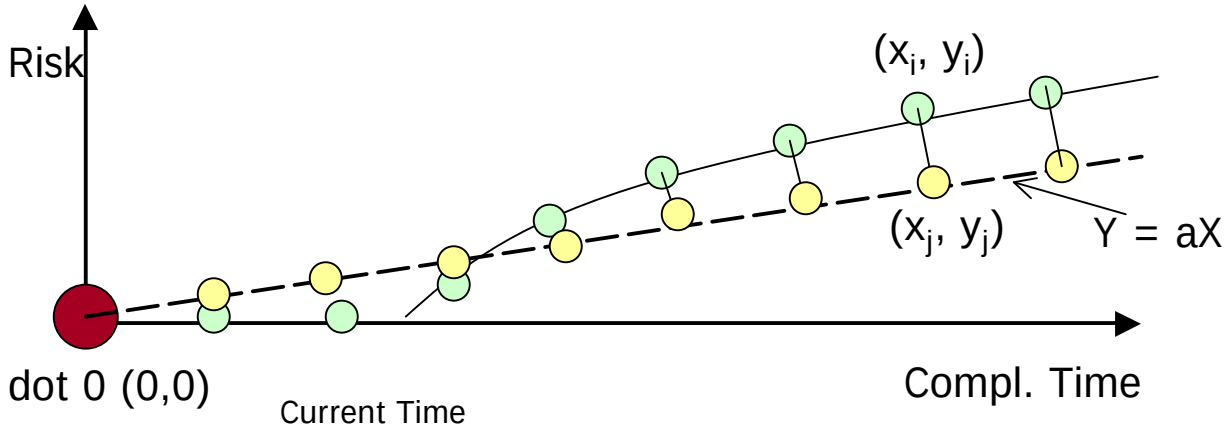
Alg. 4
Linear RIPF
O(nlgn) - online
O(n^2) - offline

Use linear approximation for the original RIPF's

- Pick a set of equally-distanced dots from the RIPF functions
- Find a linear function which go through dot0 (0,0) and the sum of the distances from all the dots to this linear function are the minimum.

$$MIN\left(\sum_{i,j=1}^{n}\sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}\right)$$

Subject to: $y_j = a\,x_j$ and $(y_j - y_i)/(x_j - x_i) = -1/a$



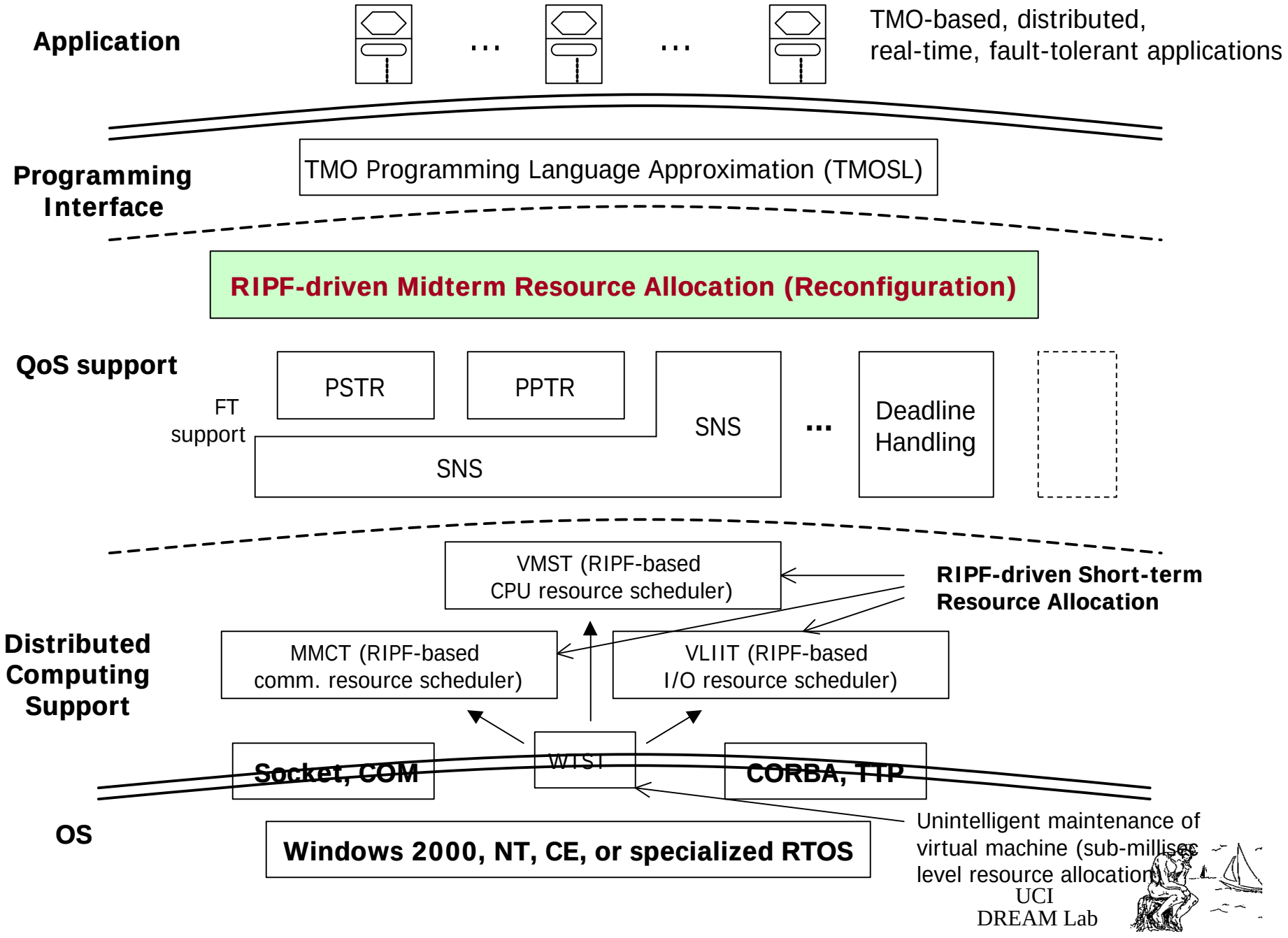
The implementation of the RIPF-driven CPU scheduling

- Since the derived RIPF set also incorporates deadline information for each SpM and SvM, the RIPF-driven resource schedulers can schedule various resources at least as efficiently as the deadline-driven resource schedulers do.

- Algorithm 3 mentioned previously has been implemented and incorporated into the current version of TMOSM. The performance of the EDF and the RIPF schedulers have been compared using the CAMIN application.

- Our analysis and experiments show that:
  - If the deadlines of all tasks can be met, the EDF and RIPF schedulers perform as efficiently;
  - In the case where not all deadlines can be met under EDF, RIPF scheduler can do a better job by considering the potential risk values together with the deadline information in the RIPFs, which means less important tasks are sacrificed first.

A QoS-driven Resource Allocation Framework based on RIF



RIPF-driven Midterm Resource Allocation (Reconfiguration)

- Two considerations about reconfiguration decision
  - Current maximum risk values returned by the RIPF-driven resource allocators
  - Current node work-load

- Maximum risk value
  - If the maximum risk value returned by one RIPF-driven resource allocator is more than zero, it means that some QoS guarantees might not be met; e.g., if the maximum risk value returned by the CPU scheduler is more than zero, some deadlines might be violated; If it is from the communication bandwidth scheduler, some communication bandwidth requirements might not be able to satisfied.

- Node work-load

  TMO work-load = $\sum$(SpM-GCT/ SpM-Interval) + $\sum$(SvM-GCT / SvM-MIR)

  GCT = Guaranteed Completion Time

  MIR = Maximum Invocation Rate

  Similarly, a node's work-load:

  Node work-load = $\sum$(TMO work-load)

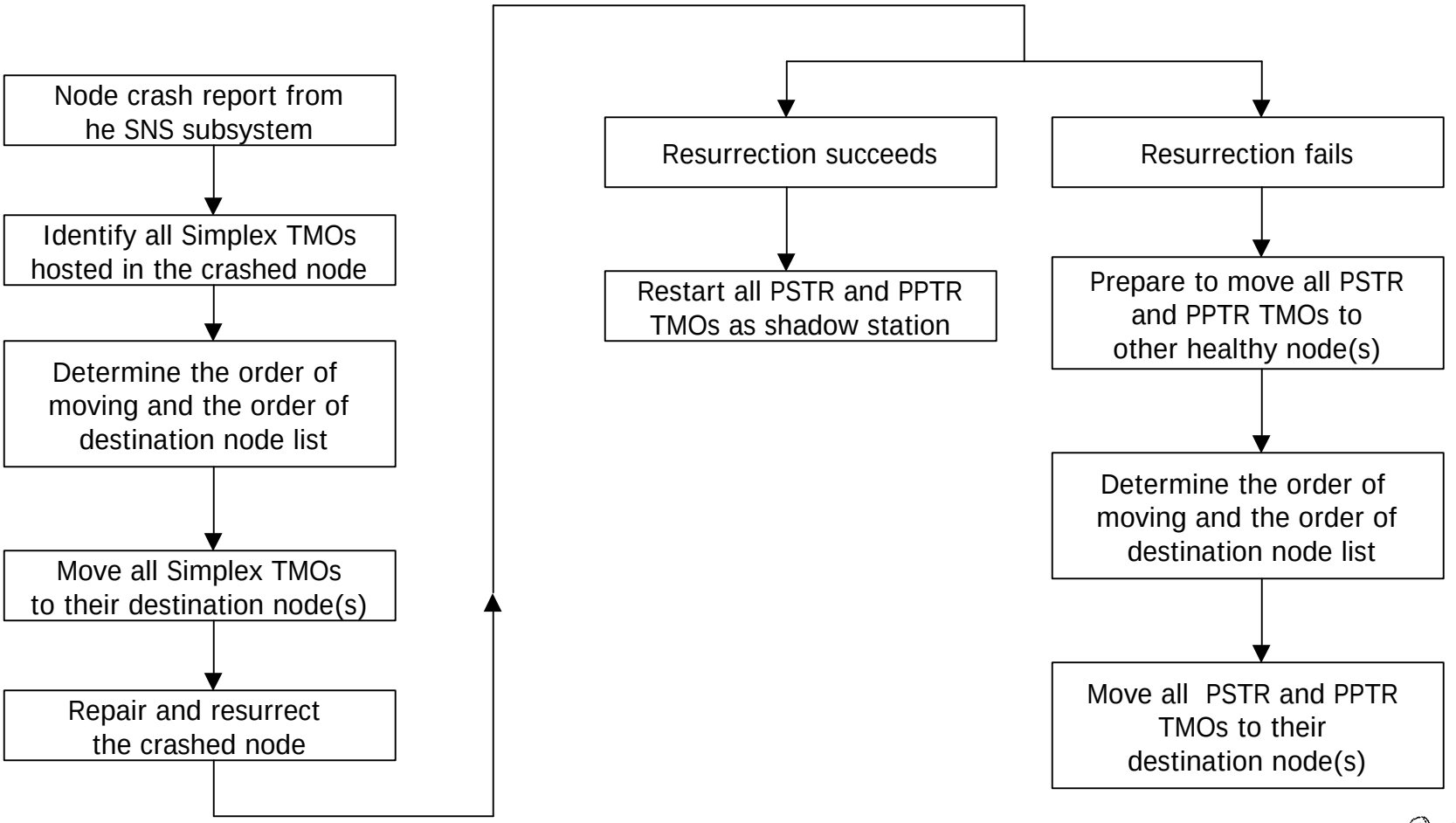
RIPF-driven Midterm Resource Allocation (Reconfiguration)

- The reasons for system reconfiguration
  - **Case 1:** Node crash occurs
  - **Case 2:** TMO crash occurs
  - **Case 3:** In a certain computing node, if the number of times that the maximum risk value appears to be positive is bigger than a threshold with a certain period, The TNCM might consider move some tasks from this node to another node.

- **Case 1: Node crash occurs**
  - TNCM examines the types of all TMOs hosted in the crashed node. The type of a TMO may be PSTR station, PPTR station, or Simplex.
  - Simplex TMOs should be moved immediately to other healthy computing node(s). Then the crashed node should be repaired and resurrected. All PSTR and PPTR TMOs hosted in this node may be restarted as the shadow station after the node is resurrected. If the resurrection fails, The PSTR and PPTR TMOs hosted in this node may be moved to other healthy node(s).
  - The order of moving TMO and the selecting of destination node(s) are based on the risk value incurred from the TMO movement.
    - The order of moving TMOs
      Examine the risk value incurred after the completion of the moving, based on the estimated moving time. The TMO with the highest risk value incursion may be moved first.
    - The selection of a destination node
      The maximum risk value of the node should be zero within a certain period;
      The node's work-load should be lower than a threshold.

RIPF-driven Midterm Resource Allocation (Reconfiguration)

Case 1: Node crash occurs: TNCM Flowchart



The Real-time Fault Tolerance Schemes Incorporated into the RIF-based Resource Allocation Framework

UCI
DREAM Lab

The Supervisor-based Network Surveillance (SNS) Scheme

The SNS scheme

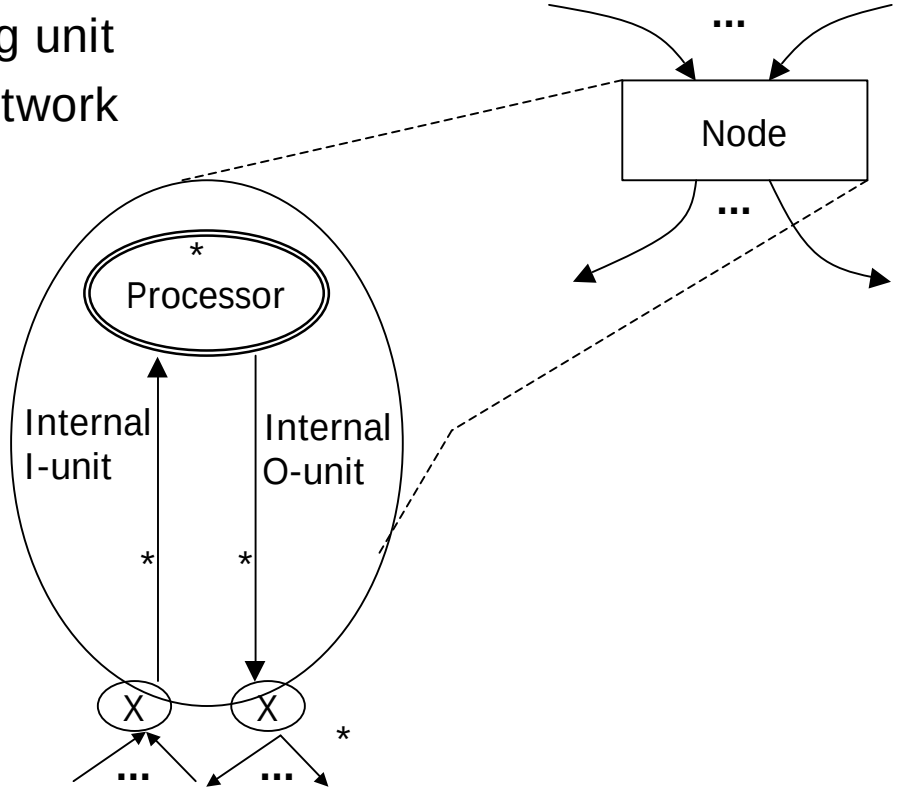
The Supervisor-base Network Surveillance (SNS) Scheme

- Network Surveillance (NS), which is basically a (partially or fully) decentralized mode of detecting faulty and repaired status of distributed computing components, is a major part of real-time fault-tolerant distributed computing.
- There are only small number of NS schemes which yield to rigorous quantitative analyses fault coverage, and the SNS scheme is one of them.
- The SNS scheme is semi-centralized real-time NS scheme effective in a variety of point-to-point networks and can also be adapted to broadcast networks.

The SNS scheme – Fault sources

Fault sources

- Processor
- incoming communication handling unit
- outgoing communication handling unit
- point-to-point interconnection network



The SNS scheme – Fault Frequencies

Fault frequencies assumptions:

(A1) The fault-source components in each node do not generate messages containing erroneous values or untimely messages.

(A2) Each of the nodes performing store-and-forward functions (as well as the source node) transmits each stored message twice continuously. It is assumed that this makes the probability of transient faults in the components of the two neighbor nodes and transient faults in the link between the two neighbor nodes causing message losses to be negligible.

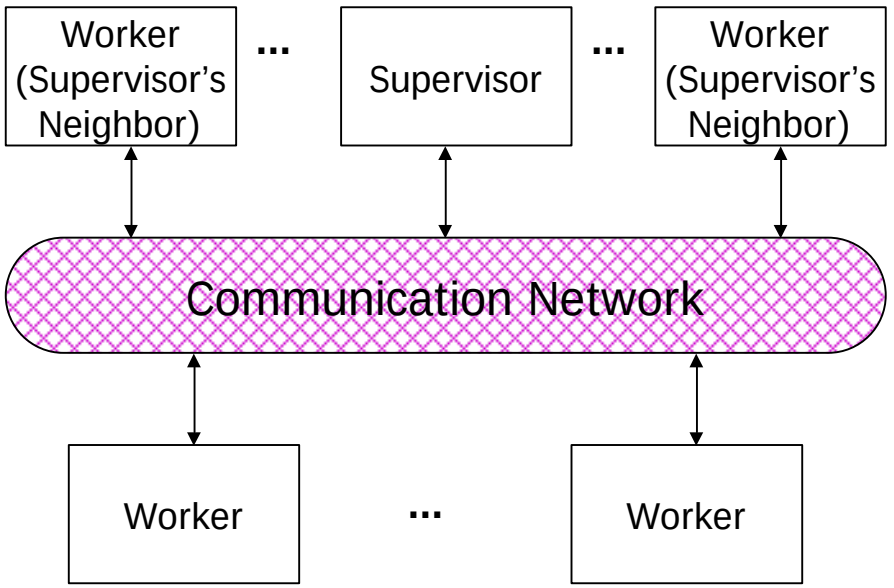
(A3) It is assumed that no second permanent hardware fault occurs in the system until either the detection of the first permanent hardware fault F or a fast re-election of the supervisor (which involves one message multicast) is done. Also network partitioning doesn't occur during the lifetime of the application.

(A4) The clocks in the nodes are kept synchronized sufficiently closely for practical purposes, i.e., for the given applications. GPS (global positioning system) based approaches and other cheaper high-precision approaches which have become available in recent years may be utilized.

The SNS scheme architecture

Basic duties of work nodes

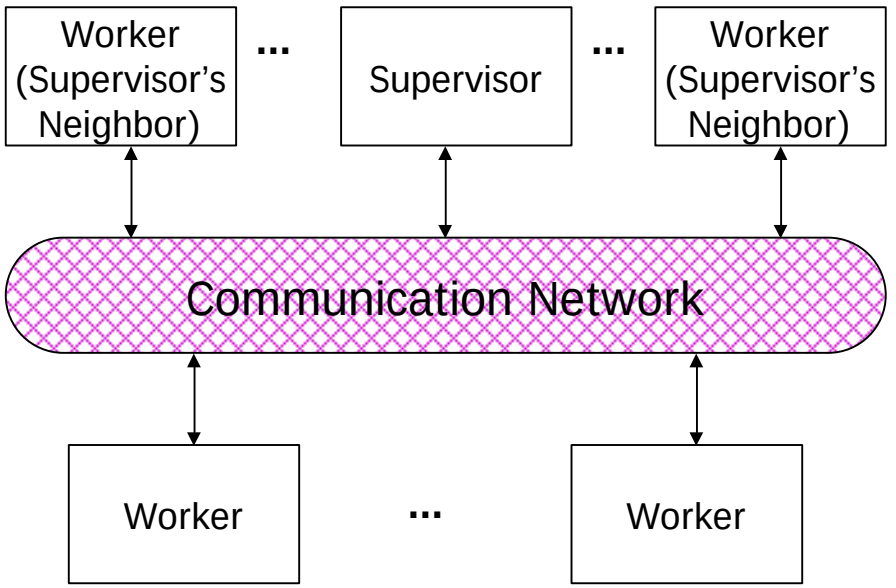
- Exchange heartbeat messages with its neighbors;
- Monitor its neighbors' health status;
- Generate fault suspicion report if necessary.



The SNS scheme architecture

Additional duties of the supervisor node

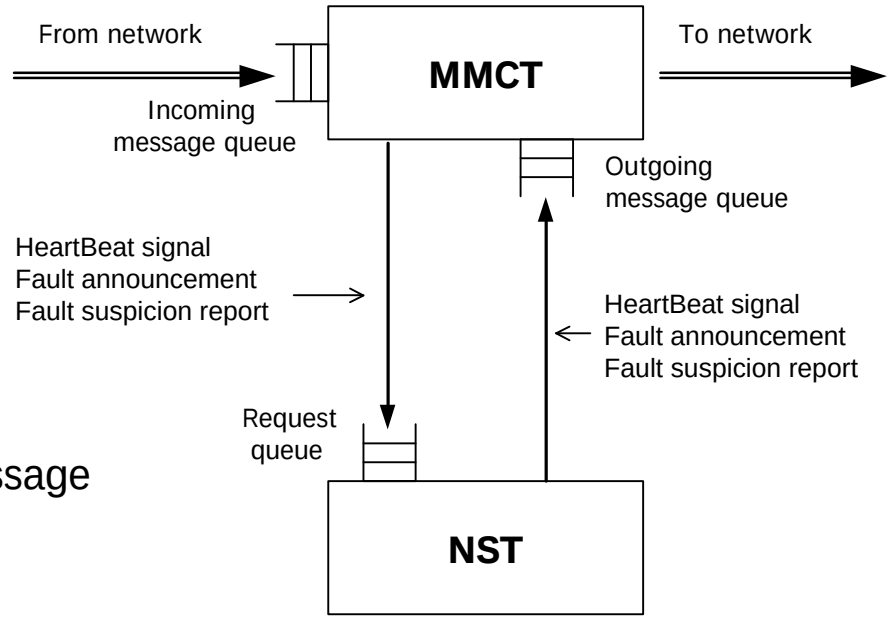
- Determine other nodes' health status based on the received suspicion reports;
- After confirming a fault, inform all the related nodes.



The SNS Implementation on the TMOSM

SNS message types:

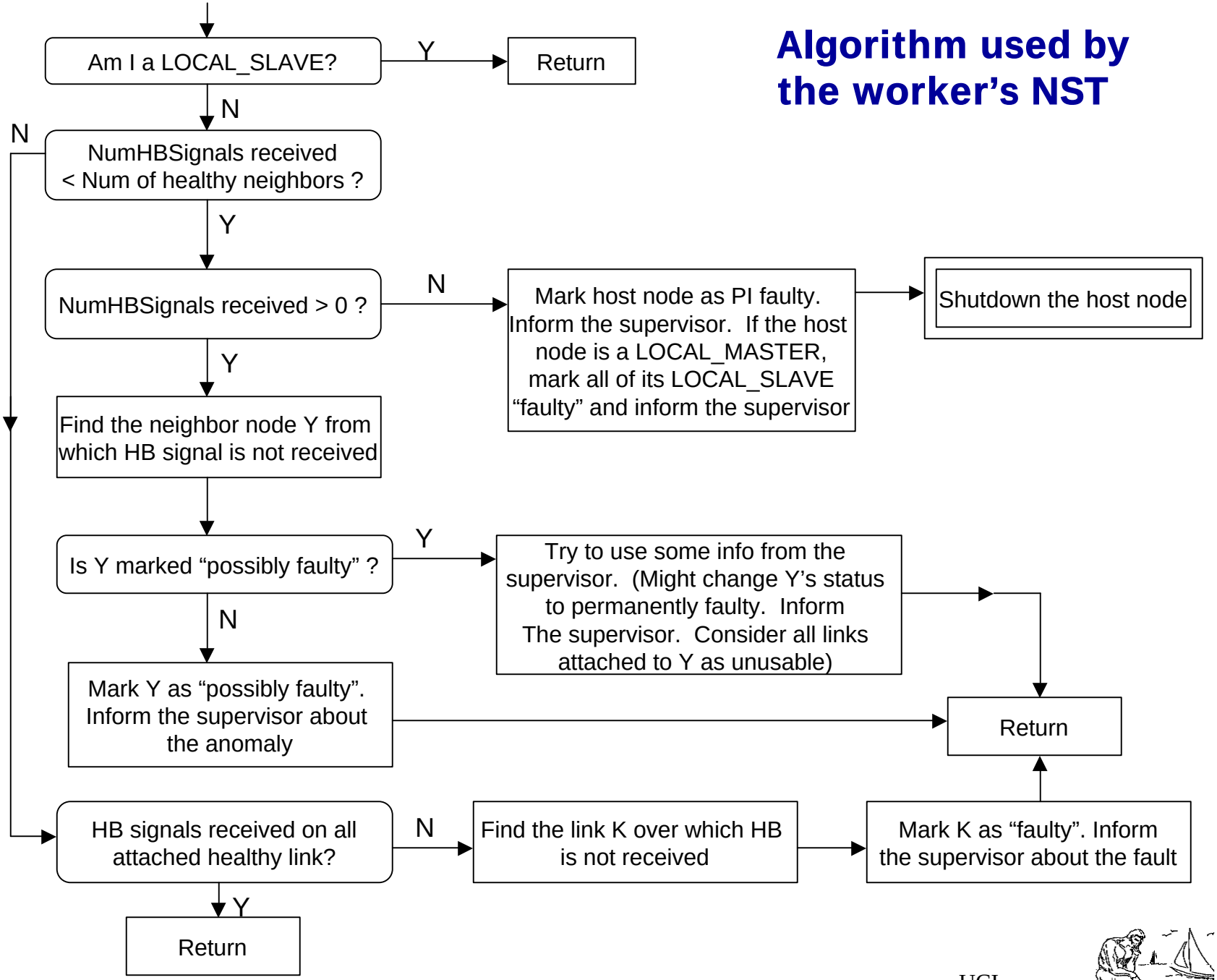
- Heart Beat Message
- Fault Suspicion Message
- Fault Announcement Message
- Supervisor-Fault Suspicion Message
- New Supervisor Announcement Message



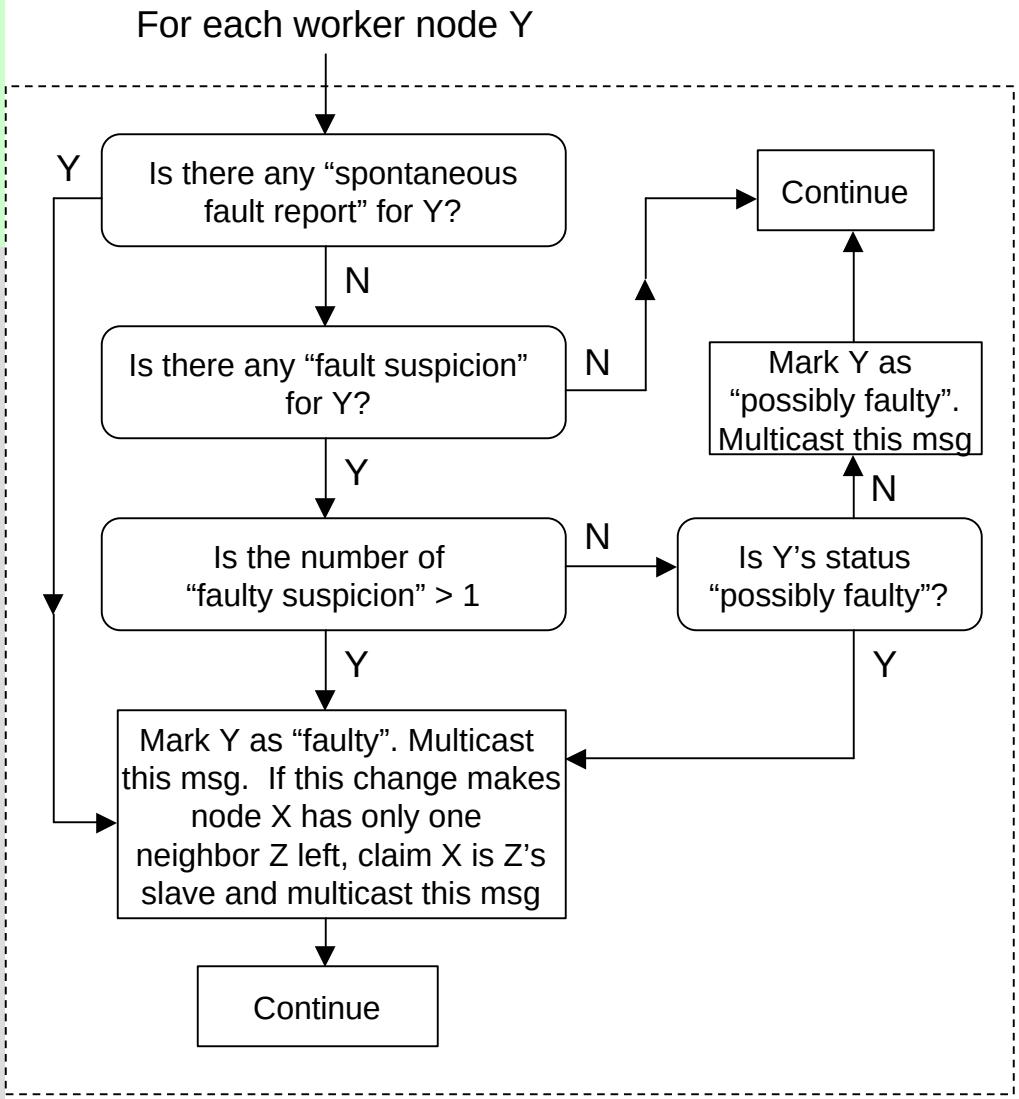
Note:

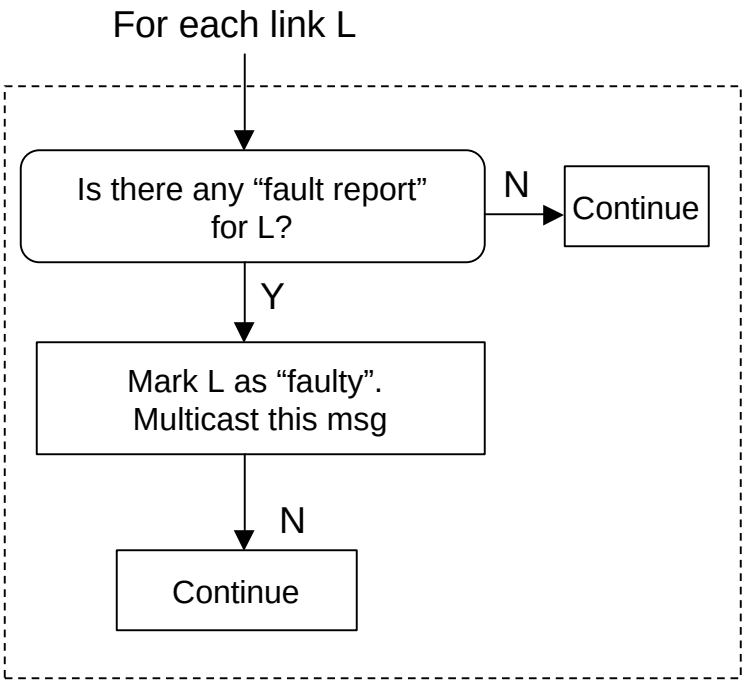
- Message sending and receiving are done in MMCT;
- Generation and analysis of messages are done in NST (Network Surveillance Thread), which is a special SpM.

Algorithm used by the worker's NST

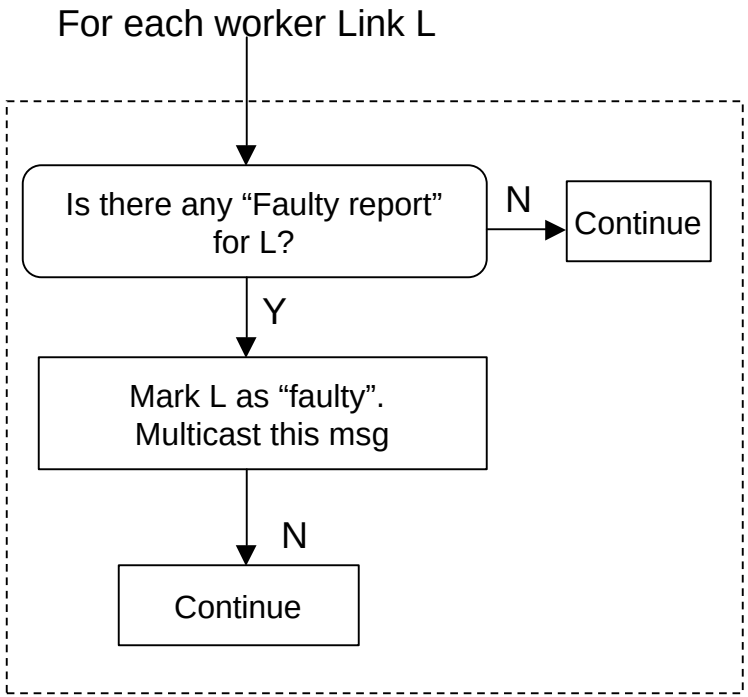


Algorithm used by the supervisor NST





Algorithm used by the supervisor NST



Fault Detection time Bound Analysis

Definition:

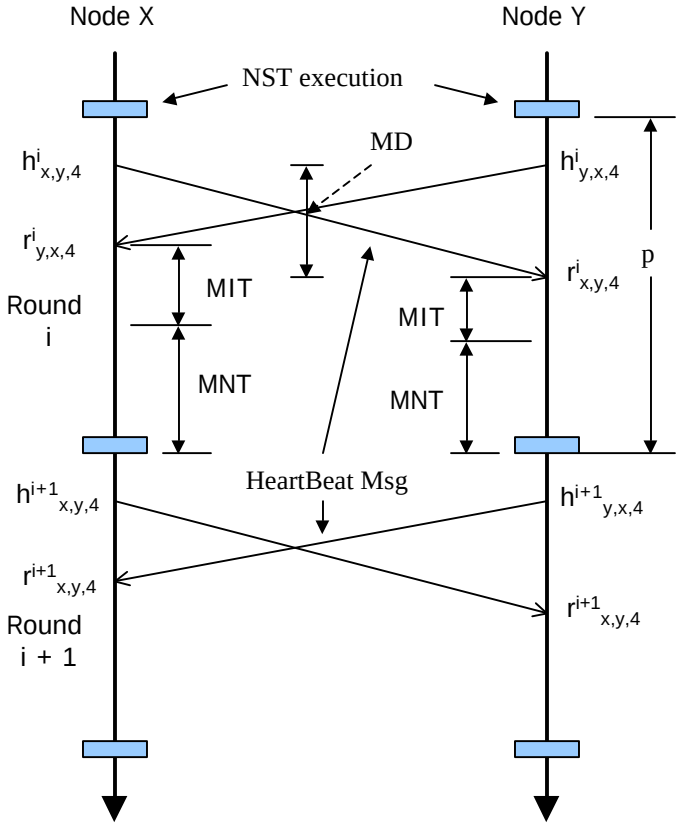
1) MIT: Maximum incoming message turnaround time of MMCT. i.e., Maximum amount of time that elapses from the arrival of a message in the input queue of MMCT in a node to the time at which MMCT completes the forwarding of the item to its destination thread.

2) MOT: Maximum outgoing message turnaround time of MMCT. i.e., Maximum amount of time that elapses from the time of arrival of an item at the input queue of MMCT to the time at which MMCT sends out the item.

3) MNT: Maximum NST turnaround time. i.e., Maximum amount of time that elapses from the time of arrival of an item at the input queue of NST to the time at which NST completes the processing of the item.

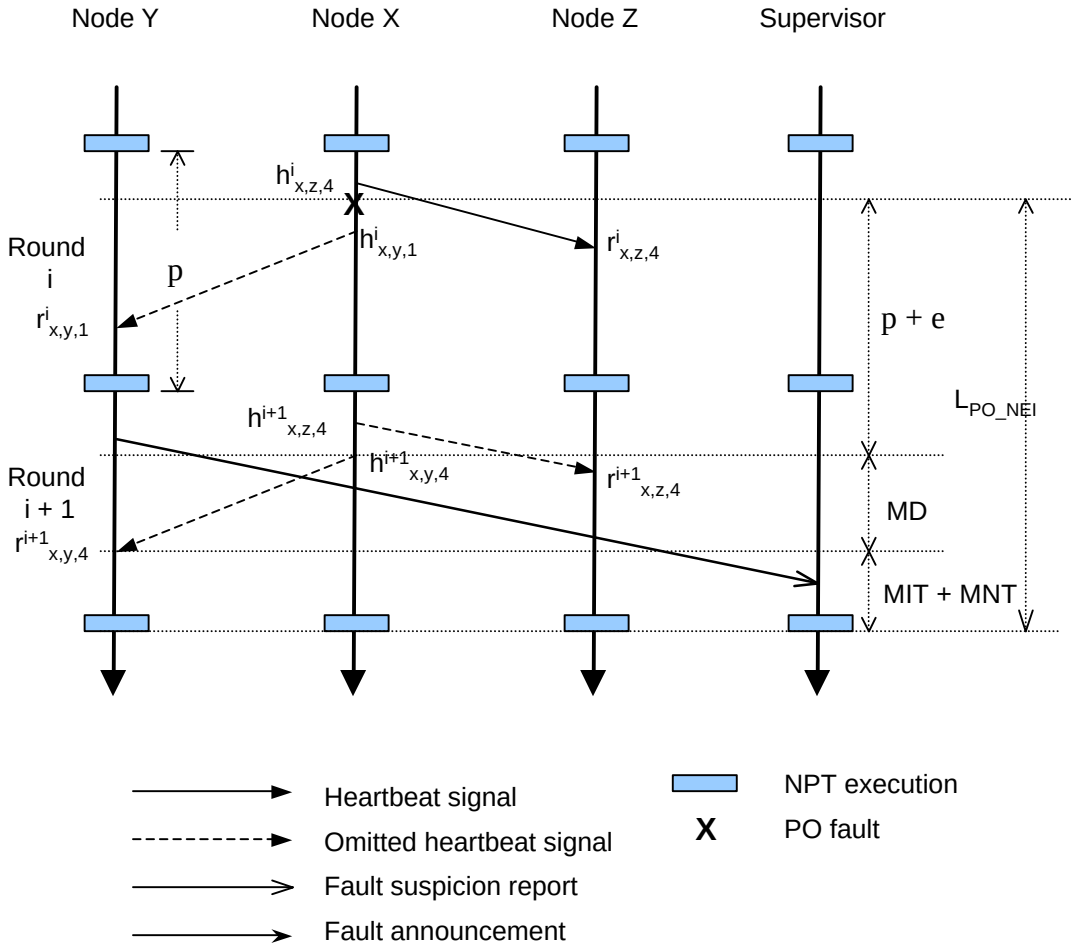
...

The SNS scheme -
Fault detection time bound analysis

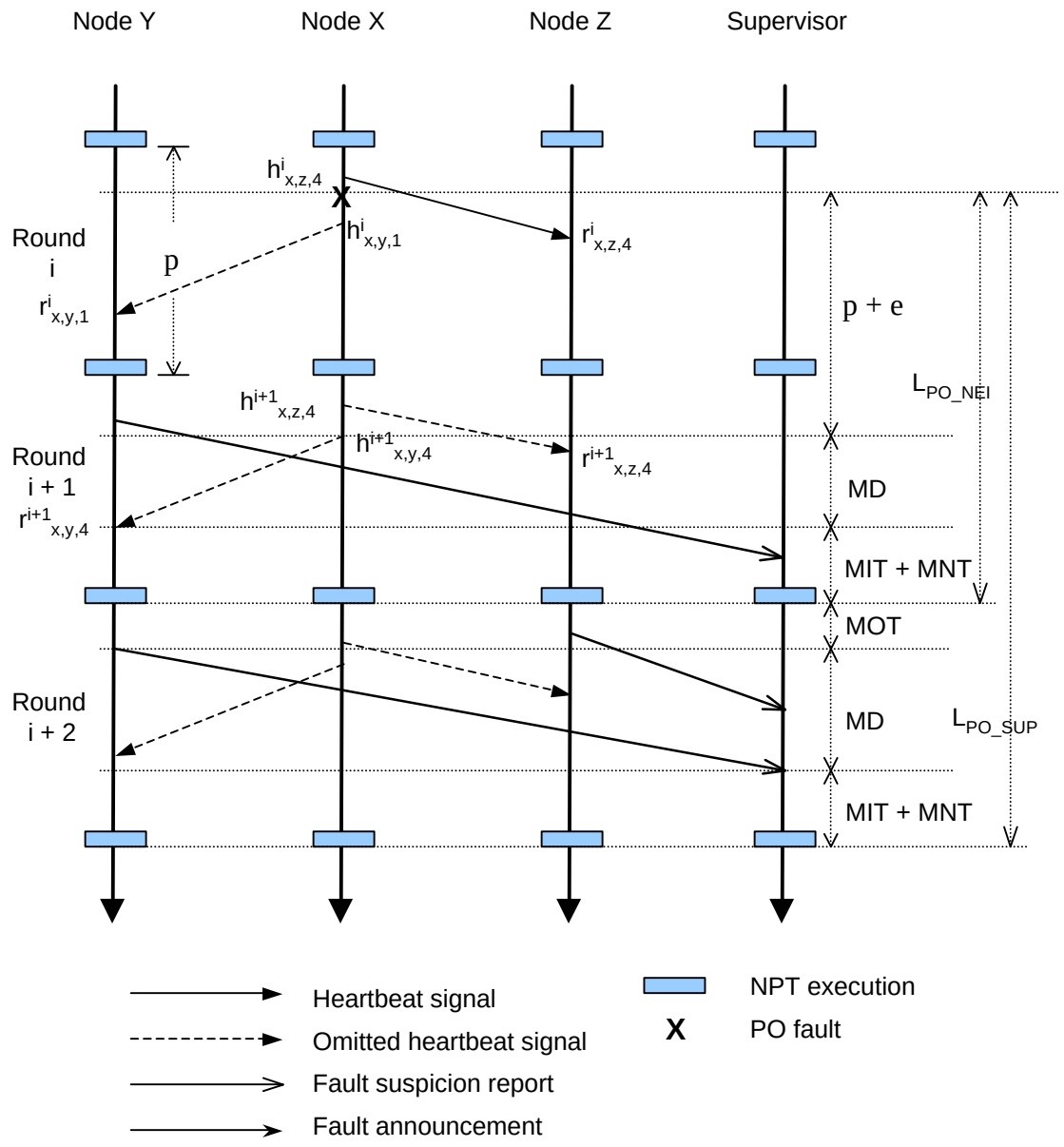


- All messages initiate in round i will be received in the the same round.
- When NST starts to execute, all messages initiate the previous round have been delivered to its input queue. All of the messages in the input queue will be processed before the completion of the NST execution.

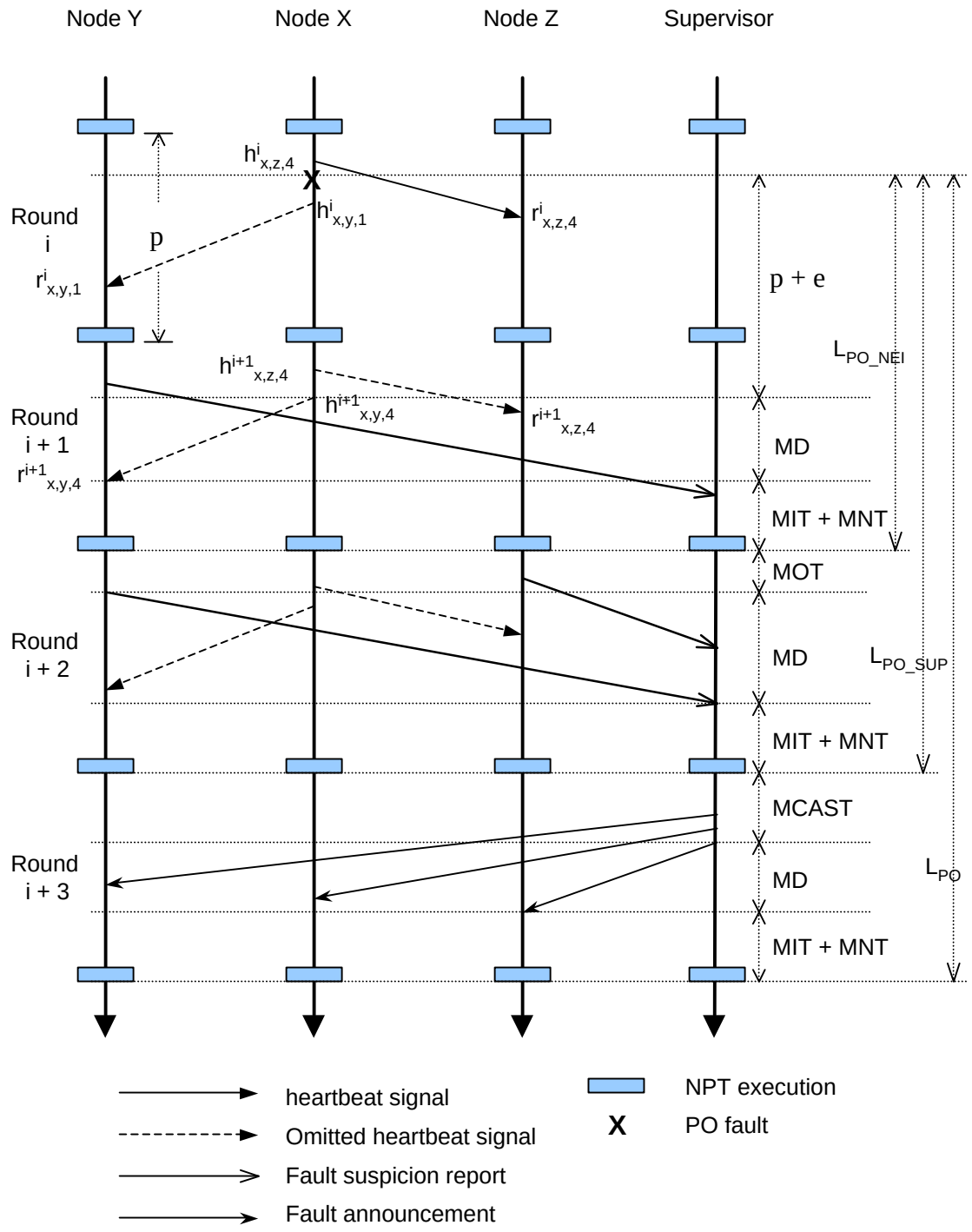
The SNS scheme -
Fault detection time bound analysis



The SNS scheme -
Fault detection time bound analysis

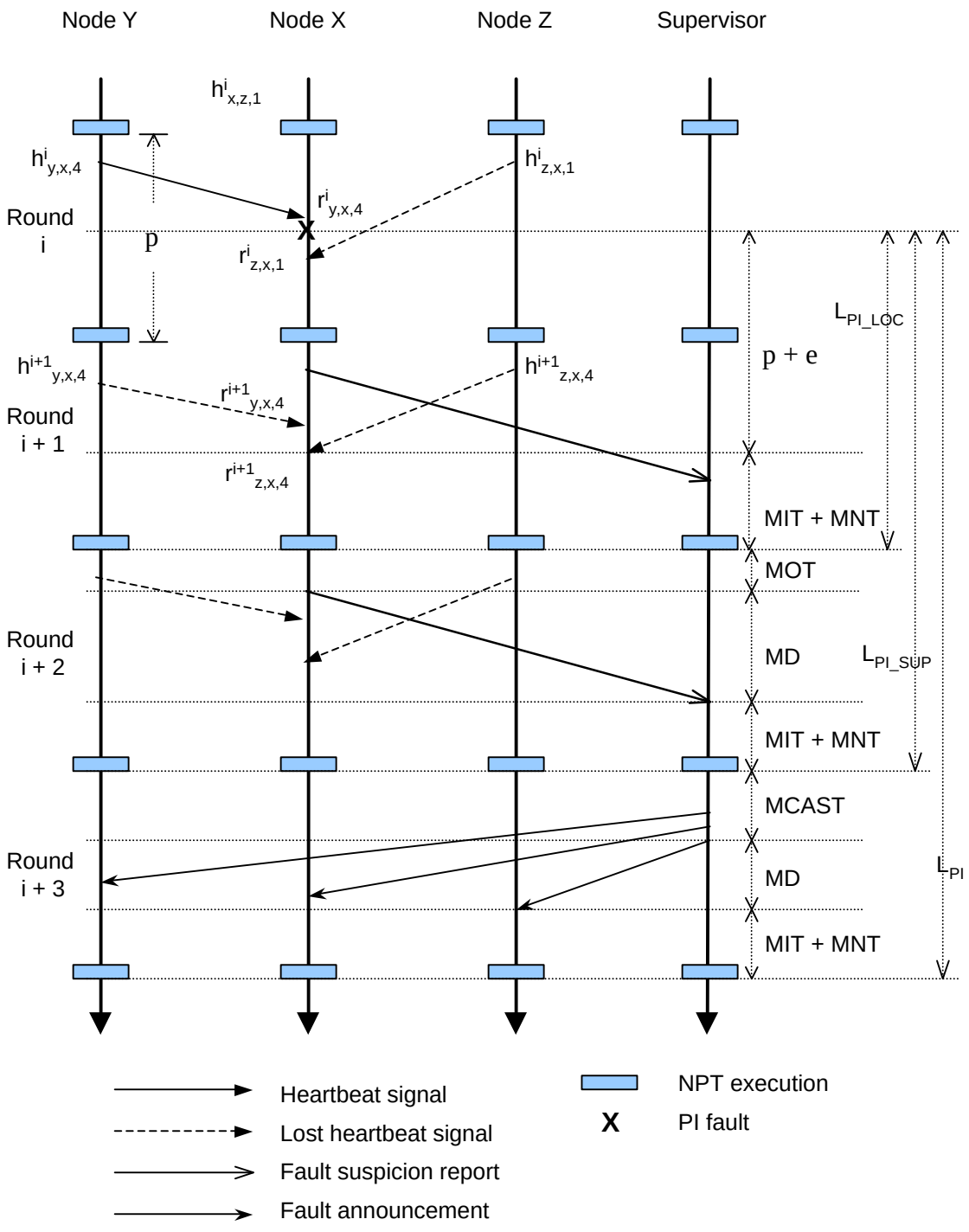


The SNS scheme -
Fault detection time bound analysis



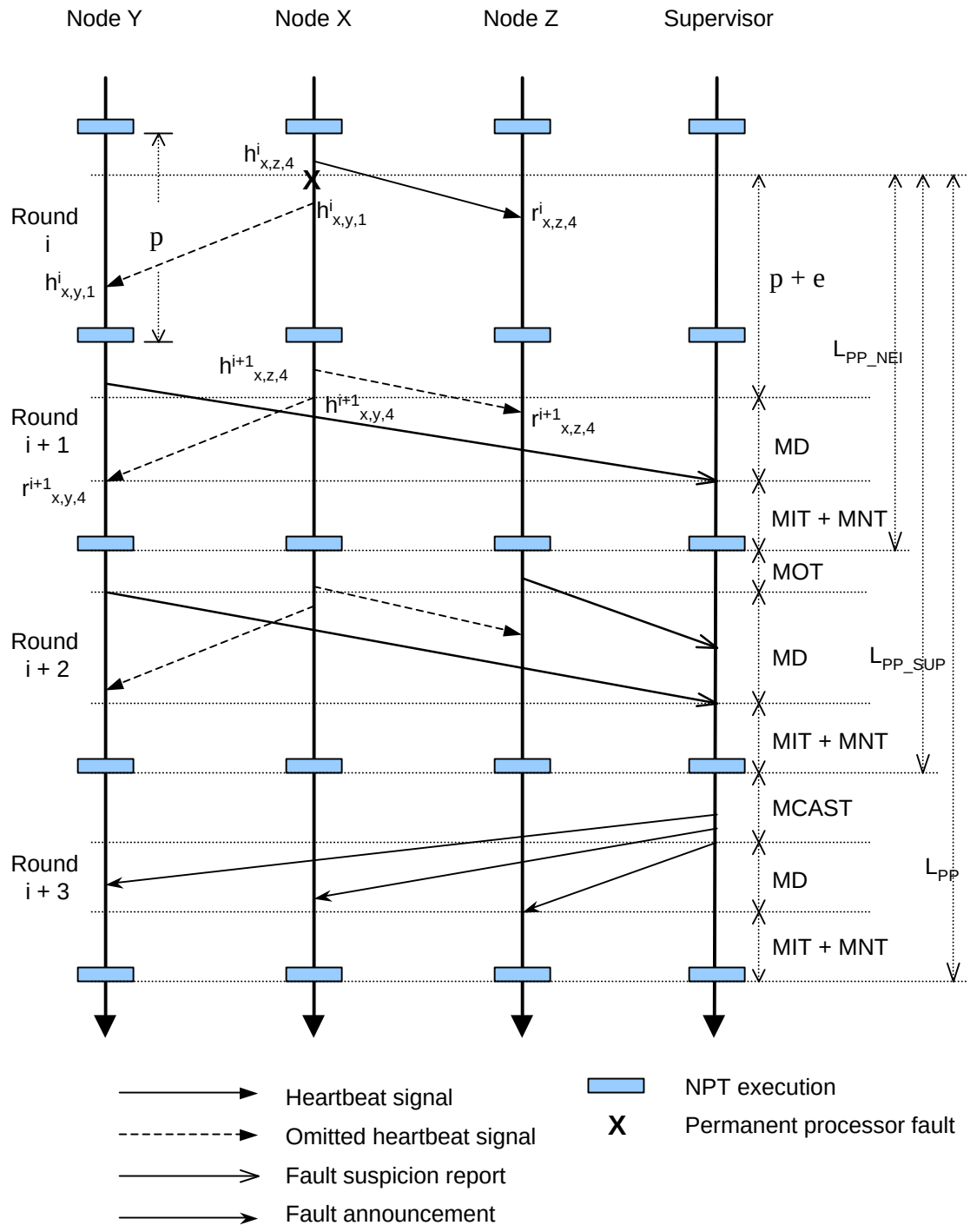
The detection procedure of a PO fault
in a worker node – node X

The SNS scheme -
Fault detection time bound analysis



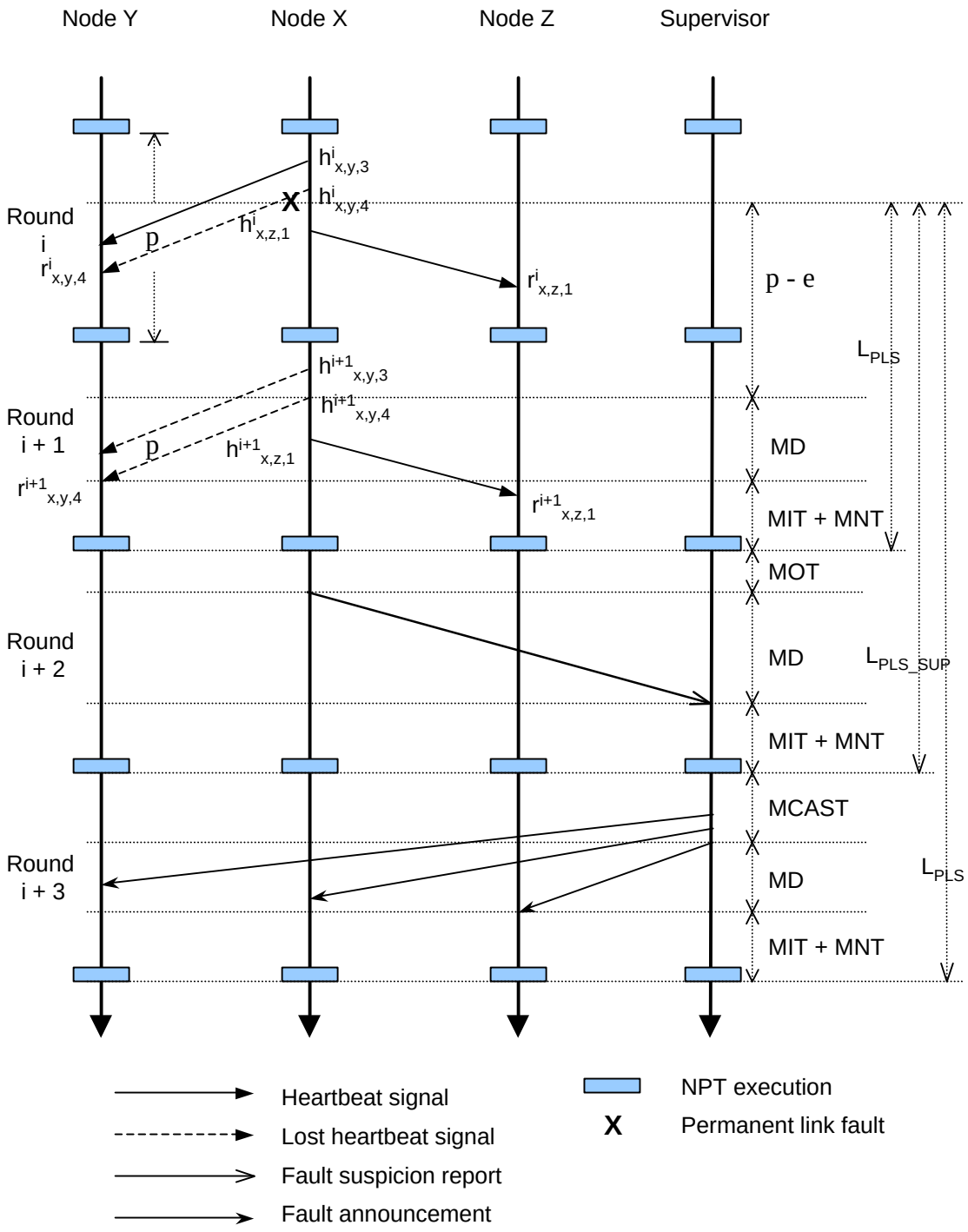
The detection procedure of a PI fault
in a worker node – node X

The SNS scheme -
Fault detection time bound analysis



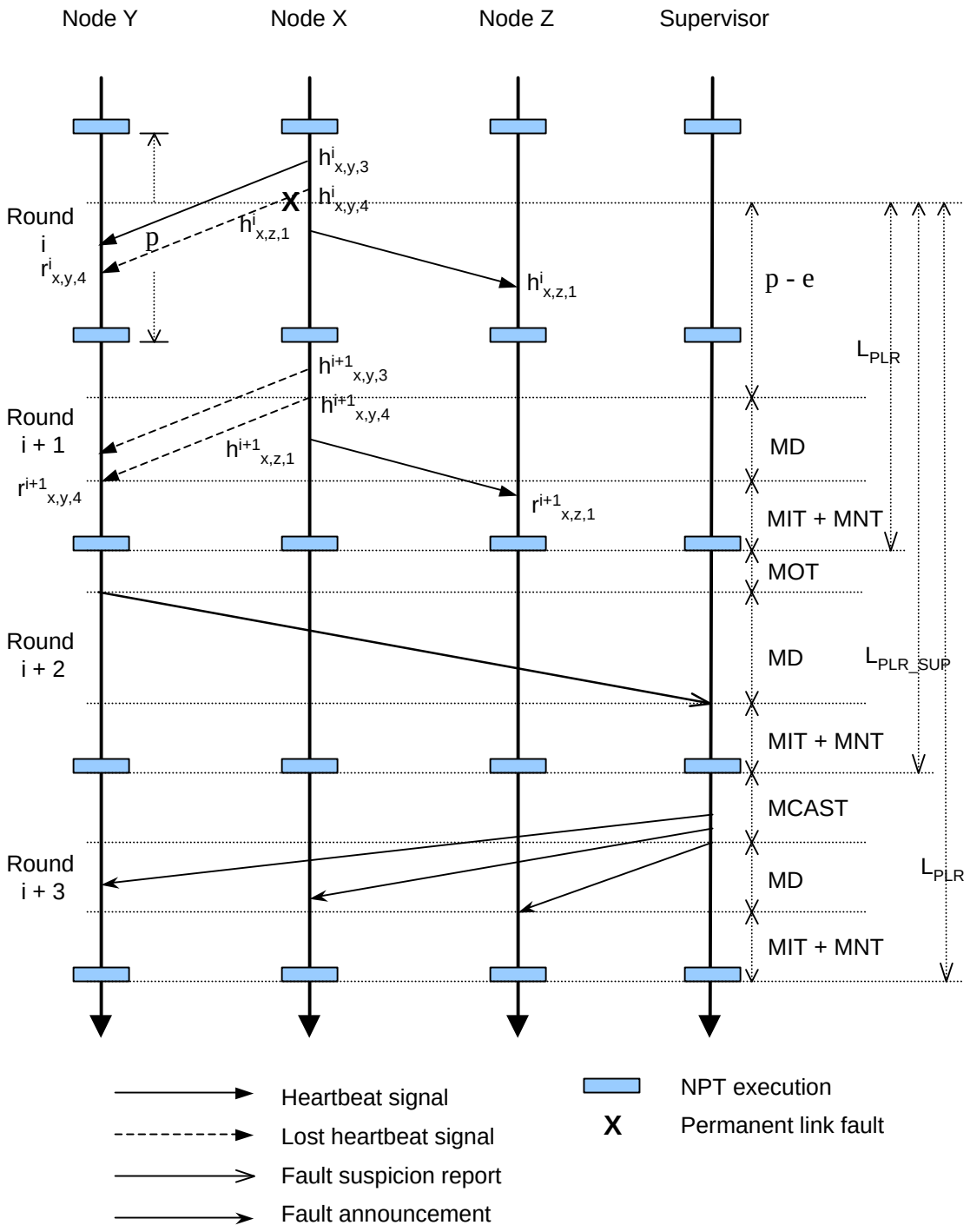
The detection procedure of a permanent processor fault in a worker node – node X

The SNS scheme -
Fault detection time bound analysis



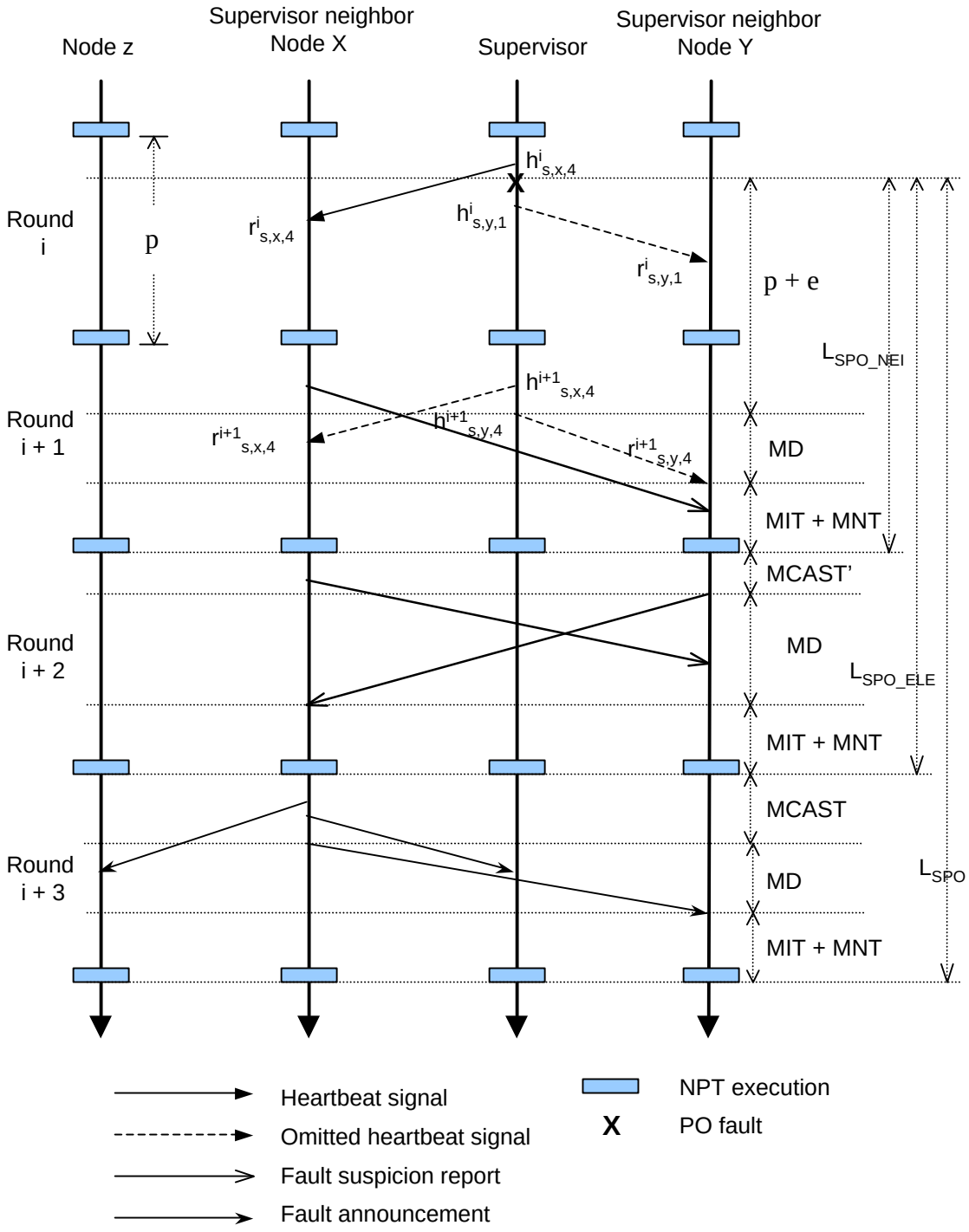
The detection procedure of a permanent Link fault
by the sender node – node X

The SNS scheme -
Fault detection time bound analysis



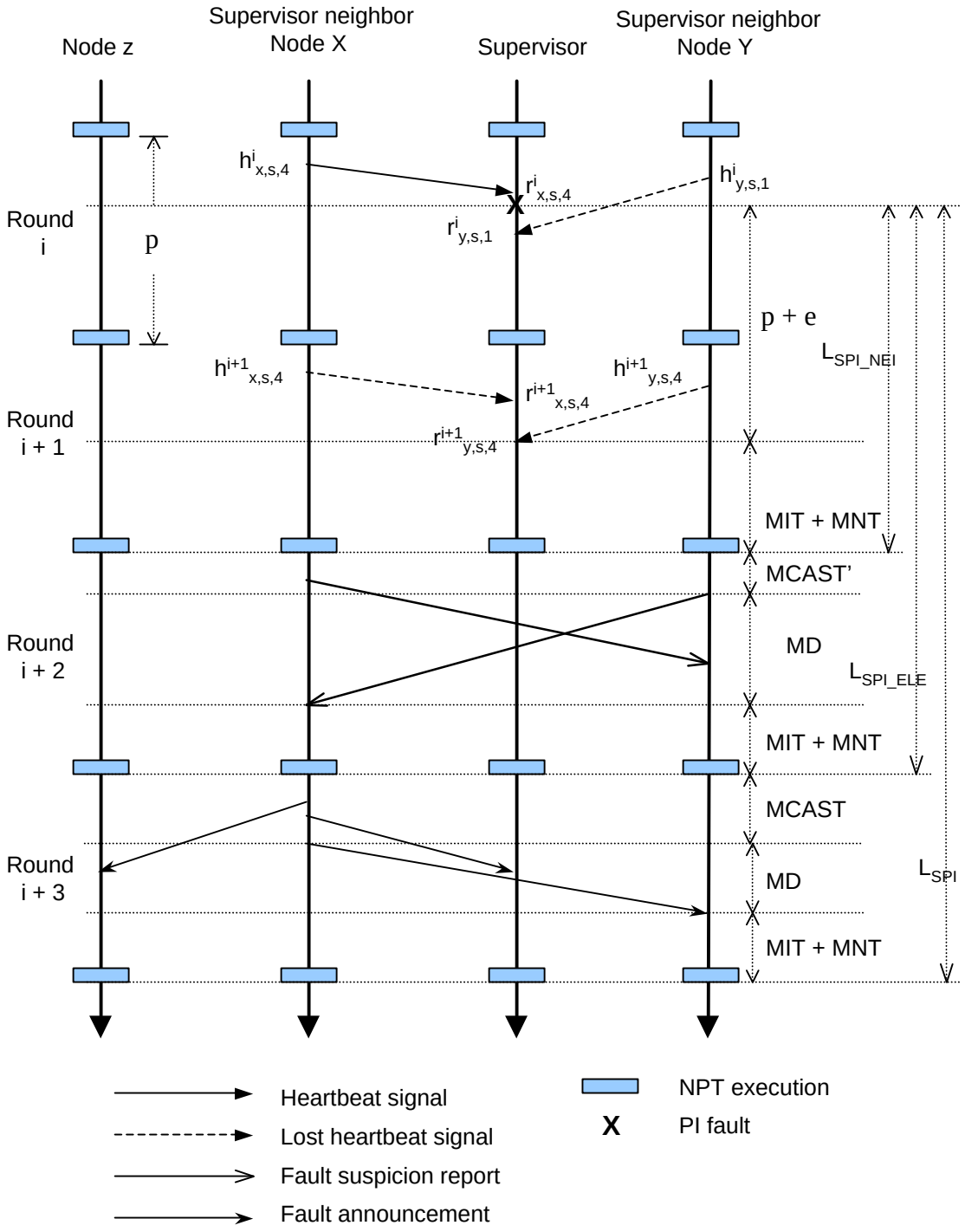
The detection procedure of a permanent Link fault
by the receiver node – node Y

The SNS scheme -
Fault detection time bound analysis



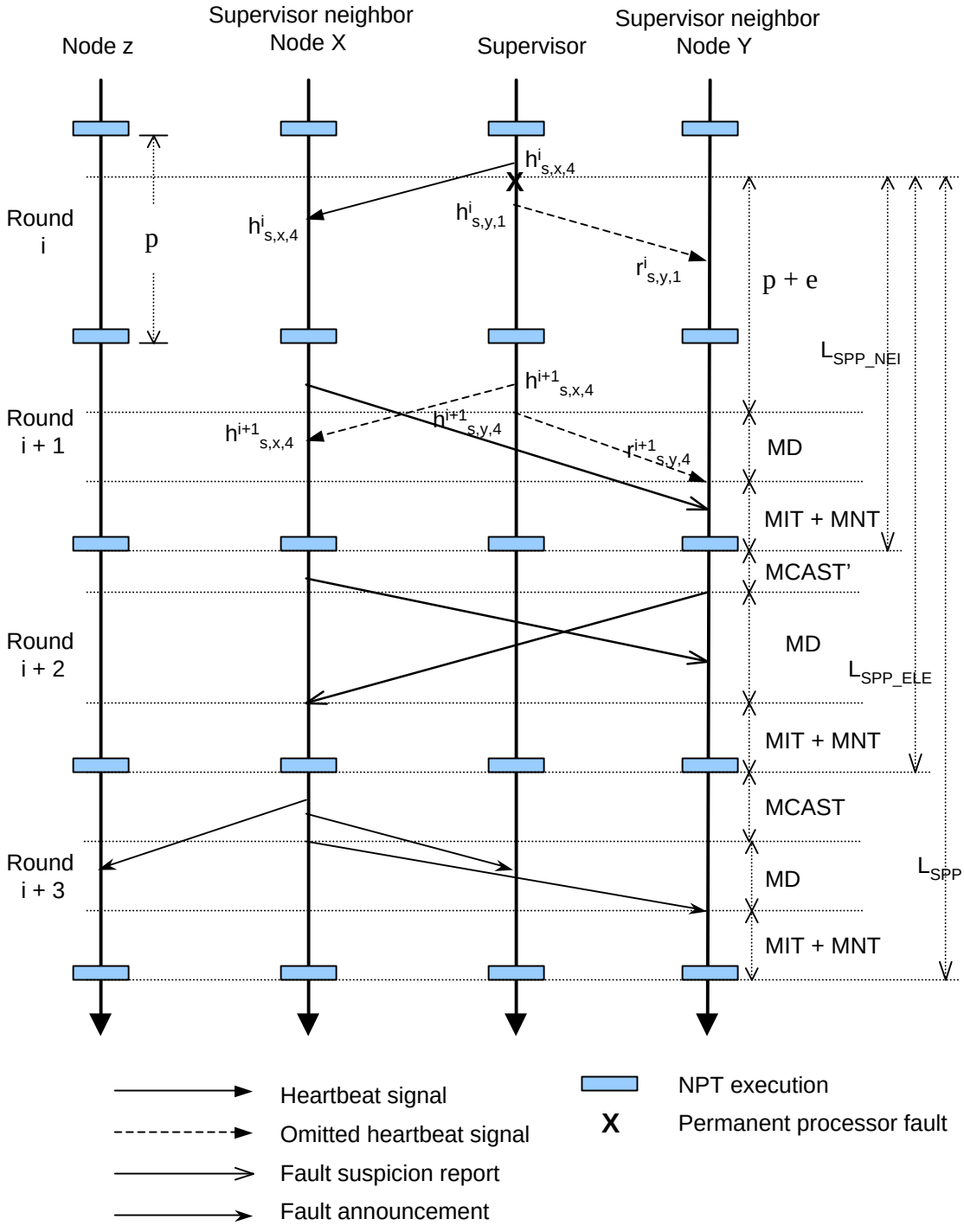
The detection procedure of a PO fault in the supervisor node

The SNS scheme -
Fault detection time bound analysis



The detection procedure of a PI fault in the supervisor node

The SNS scheme -
Fault detection time bound analysis



The detection procedure of a permanent processor fault in the supervisor node

Algorithm used by the supervisor NST

Experimental data

Message delay

1) 400 byte package
    1. In isolated network: 189us;
    2. In Internet environment: 192us;

2) 600 byte package
    1. In isolated network: 212 us;
    2. In Internet environment: 236us.

Maximum MMCT turnaround time: 82us.

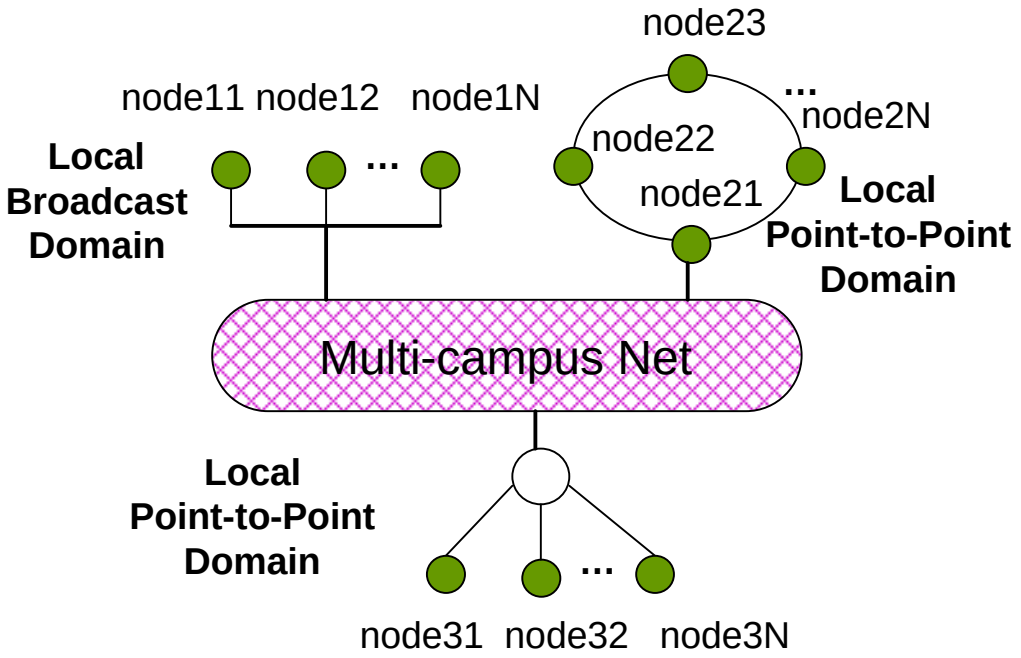
Maximum NST turnaround time: 28us.

Selecting NST execution period p = 12ms,
both the fault detection and the new supervisor election take about 3.5 p, 42ms

Algorithm used by the supervisor NST

The main issues of adaptation are:

1) selecting appropriate neighboring scheme, and
2) establishing two independent communication paths between any two nodes in the system.



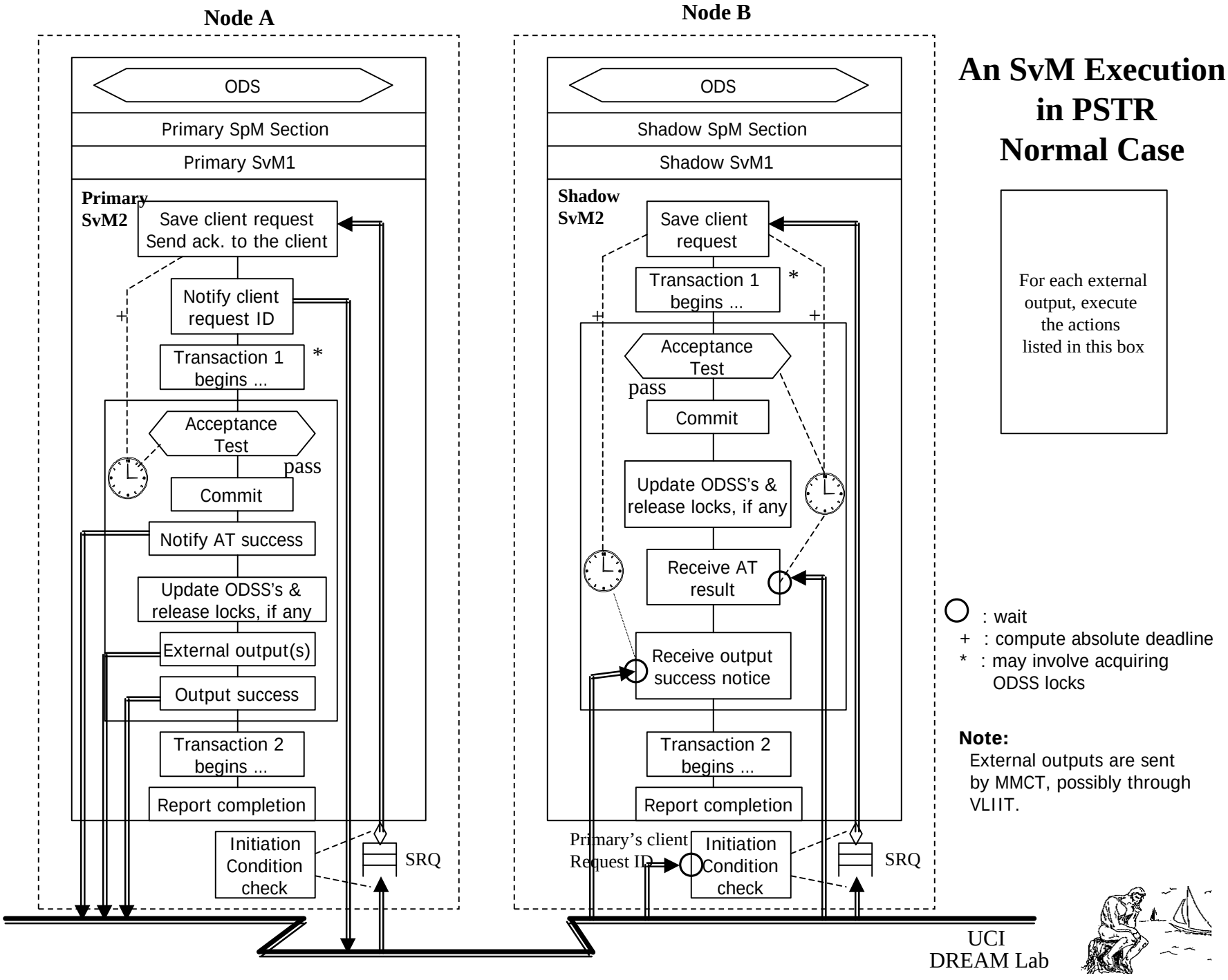
PSTR -
The Primary Shadow TMO Replication Scheme

UCI
DREAM Lab

The PSTR scheme

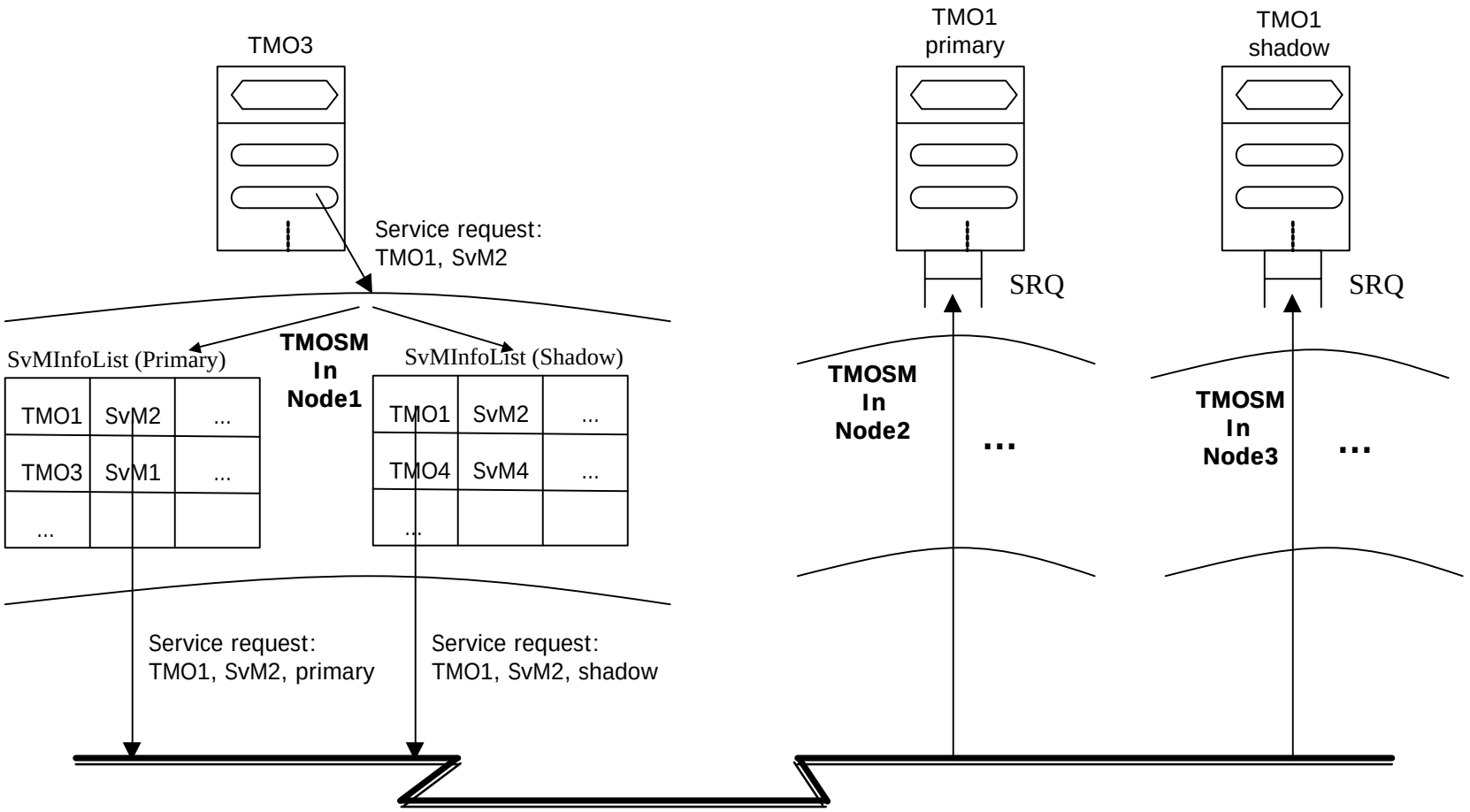
- The PSTR scheme is a result of incorporating the primary-shadow active replication principle, into the TMO object structuring scheme.
- A natural way to incorporate the active replication principle into the TMO structuring scheme is to replicate each TMO to form a pair of partner objects and host the partners in two different nodes.
- The methods of the primary object along will produce all external outputs under normal circumstance.
- Since each partner has the same external inputs and its own object data store (ODS), the methods of both objects perform the same execution and ODS updates.

An SvM Execution in PSTR Normal Case

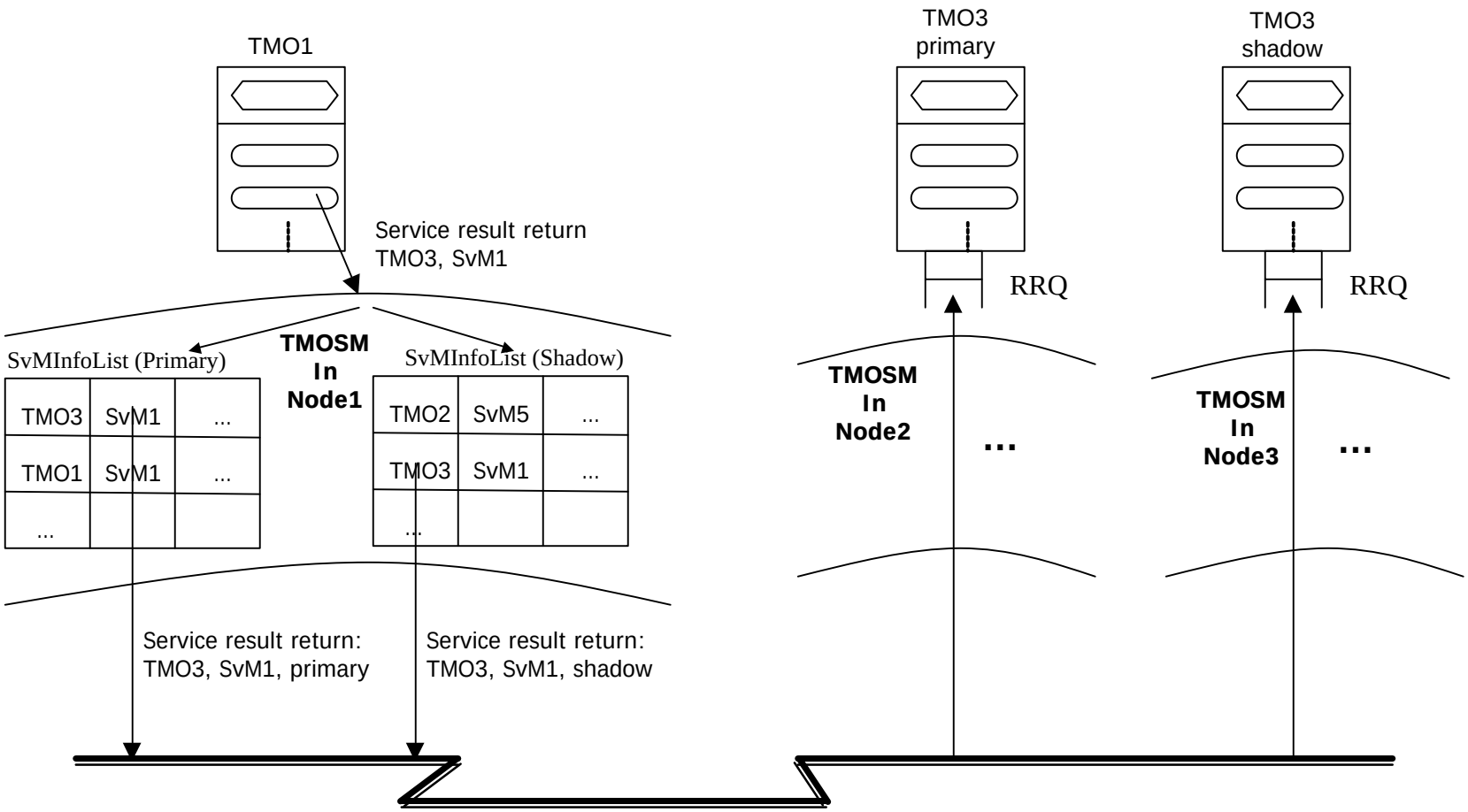


UCI
DREAM Lab

Handling inputs to TMO replicas – Service request



Handling inputs to TMO replicas – Result return



Types of faults & their symptoms

- **Hardware faults**
  - *Symptoms 1.1* Node crash
  - *Symptoms 1.2* Process/thread gets corrupted – no progress
  - *Symptoms 1.3* Process/thread gets corrupted – progress but with contaminated state (Low probability)
  - *Symptoms 2.1* Resource shortage -> Process/thread lockup/stall

- **OS faults**
  - *Symptoms 1.1* Node crash
  - *Symptoms 1.2* Process/thread gets corrupted – no progress
  - *Symptoms 1.3* Process/thread gets corrupted – progress but with contaminated state (Low probability)
  - *Symptoms 2.1* Resource shortage -> Process/thread lockup/stall

- **Communication failures**
  - *Symptoms 3.1* Message loss
  - *Symptoms 3.2* Duplicated messages

- **Application design faults**
  - *Symptoms 1.2* Process/thread gets corrupted – no progress
  - *Symptoms 1.3* Process/thread gets corrupted – progress but with contaminated state (high probability)

PSTR fault detection mechanism

- Primary's AT - logic test (Detection mechanism(DM) 1.1)
- Primary's AT – timeout (DM 1.2)
- Primary's sending of clientRequestID – timeout (DM 1.3)
- Shadow's wait for clientRequestID – timeout (DM 2.1)
- Shadow's AT - logic test (DM 2.2)
- Shadow's AT – timeout (DM 2.3)
- Shadow's wait for primary's AT result – timeout (DM 2.4)
- Shadow's wait for primary's notice of external output success – timeout (DM 2.5)
- SNS's node failure notice (DM 3.1)
- Message-sequence check(Double transmission over redundant links are done) (DM 4.1)
- Absence of ack. (DM 4.2)
  - Server's ack of an SvM request (DM 4.2.1)
  - Server's return of the expected result (DM 4.2.2)
- Unacceptable request to kernel/middleware (DM 5.1)

Note:

1. When a TMO changes its role between primary & shadow, it reports the change to TMOSM which in turn notifies the TNCM. The TNCM can detect primary-primary situations
2. Every external output should be done in an independent manner.

PSTR fault detection mechanism

• Primary's AT - logic test (Detection mechanism(DM) 1.1)	- Given by application programmers
• Primary's AT – timeout (DM 1.2)	- Given by application programmers or by the tools
• Primary's sending of clientRequestID – timeout (DM 1.3)	- Given by application programmers or by the tools
• Shadow's wait for clientRequestID – timeout (DM 2.1)	- Given by application programmers or by the tools
• Shadow's AT - logic test (DM 2.2)	- Given by application programmers
• Shadow's AT – timeout (DM 2.3)	- Given by application programmers or by the tools
• Shadow's wait for primary's AT result – timeout (DM 2.4)	- Derived
• Shadow's wait for primary's notice of external output success – timeout (DM 2.5)	- Derived

PSTR fault detection mechanism

- SNS's node failure notice (DM 3.1)
- Message-sequence check(Double transmission over redundant links are done) (DM 4.1)
- Absence of ack. (DM 4.2)
  - Server's ack of an SvM request (DM 4.2.1)
  - Server's return of the expected result (DM 4.2.2)
- Unacceptable request to kernel/middleware (DM 5.1)

} - Provided by TMOSM

Typical cases of fault detection under PSTR + SNS

Faults in the primary

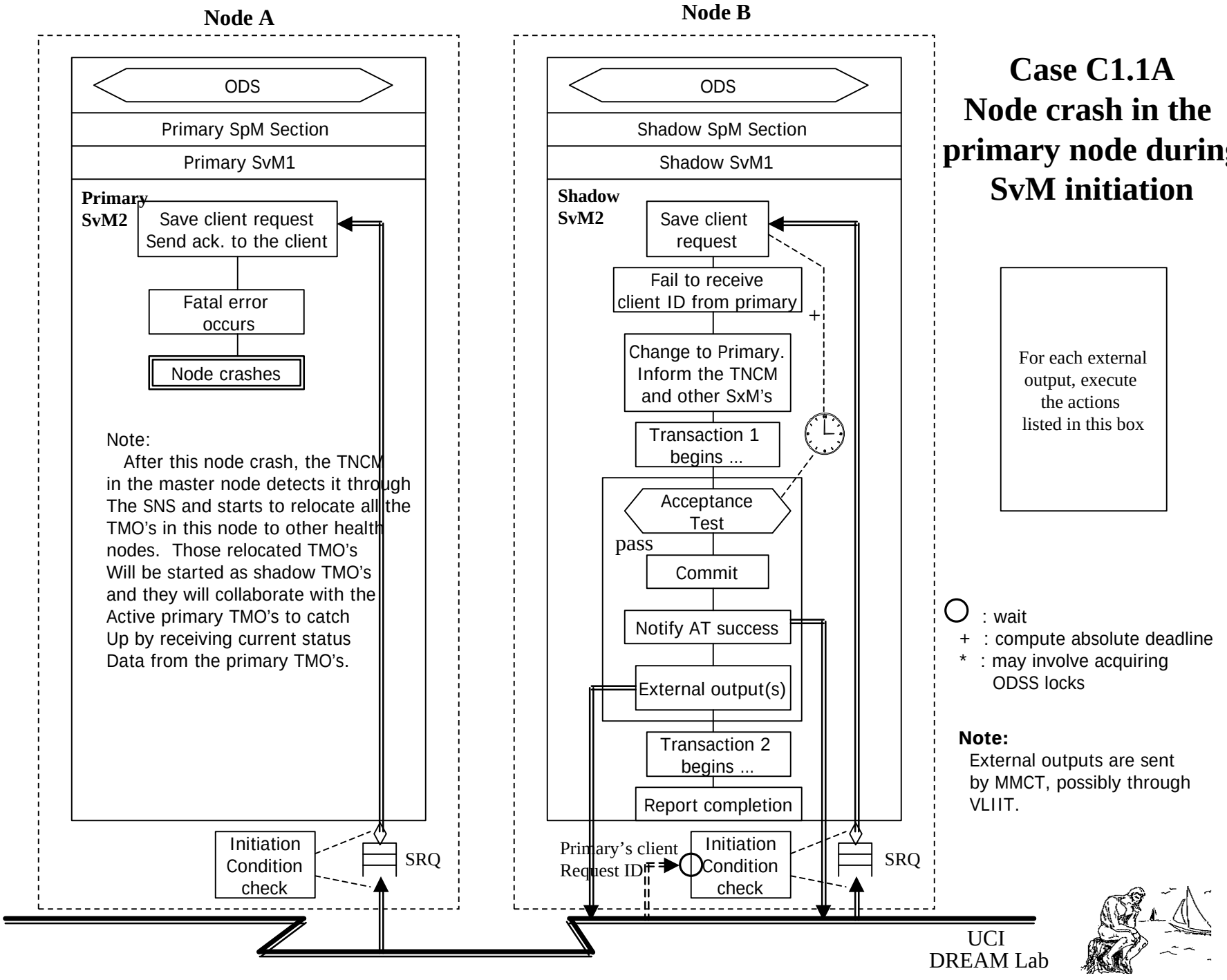
Fault types \ Detection mechanisms		Primary			Shadow					SNS	Messaging		Kernel/ Middleware
		DM 1.1	DM 1.2	DM 1.3	DM 2.1	DM 2.2	DM 2.3	DM 2.4	DM 2.5	DM 3.1	DM 4.1	DM 4.2	DM5.1
Hard ware	Sym1.1				C1.1			C1.2	C1.3	C1.4		C1.5	
	Sym1.2	C1.6	C1.7	C1.8	C1.1			C1.2	C1.3			C1.5	
	Sym1.3	C1.6	C1.7	C1.8	C1.1			C1.2	C1.3			C1.5	C1.9
	Sym2.1	C1.6	C1.7	C1.8	C1.1			C1.2	C1.3	C1.4		C1.5	
OS	Sym1.1				C1.1			C1.2	C1.3	C1.4		C1.5	
	Sym1.2	C1.6	C1.7	C1.8	C1.1			C1.2	C1.3			C1.5	
	Sym1.3	C1.6	C1.7	C1.8	C1.1			C1.2	C1.3			C1.5	C1.9
	Sym2.1	C1.6	C1.7	C1.8	C1.1			C1.2	C1.3	C1.4		C1.5	
Comm	Sym3.1				C1.1			C1.2	C1.3			C1.5	
	Sym3.2											C1.5	
App	Sym1.2	C1.6	C1.7	C1.8	C1.1			C1.2	C1.3			C1.5	
	Sym1.3	C1.6	C1.7	C1.8	C1.1			C1.2	C1.3			C1.5	C1.9

Typical cases of fault detection under PSTR + SNS

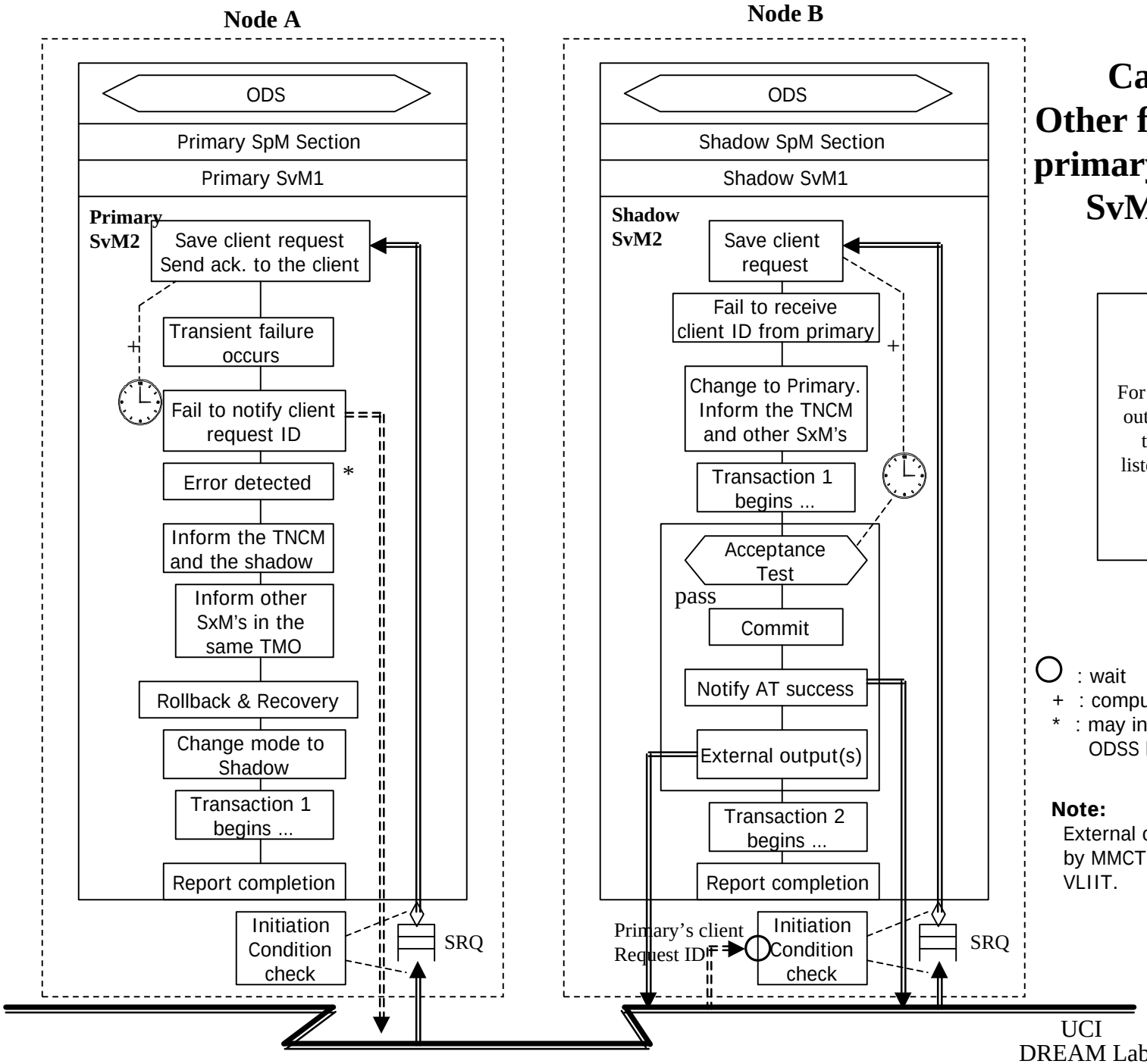
Faults in the shadow

Fault types \ Detection mechanisms		Primary			Shadow					SNS	Messaging		Kernel/ Middleware
		DM 1.1	DM 1.2	DM 1.3	DM 2.1	DM 2.2	DM 2.3	DM 2.4	DM 2.5	DM 3.1	DM 4.1	DM 4.2	DM5.1
Hard ware	Sym1.1									C2.1		C2.4	
	Sym1.2					C2.2	C2.3					C2.4	
	Sym1.3					C2.2	C2.3					C2.4	C2.5
	Sym2.1					C2.2	C2.3			C2.1		C2.4	
OS	Sym1.1									C2.1		C2.4	
	Sym1.2					C2.2	C2.3					C2.4	
	Sym1.3					C2.2	C2.3					C2.4	C2.5
	Sym2.1					C2.2	C2.3			C2.1		C2.4	
Comm	Sym3.1					C2.2	C2.3					C2.4	
	Sym3.2											C2.4	
App	Sym1.2					C2.2	C2.3					C2.4	
	Sym1.3					C2.2	C2.3					C2.4	C2.5

Case C1.1A
Node crash in the primary node during SvM initiation



Case C1.1B
Other failures in the primary node during SvM initiation

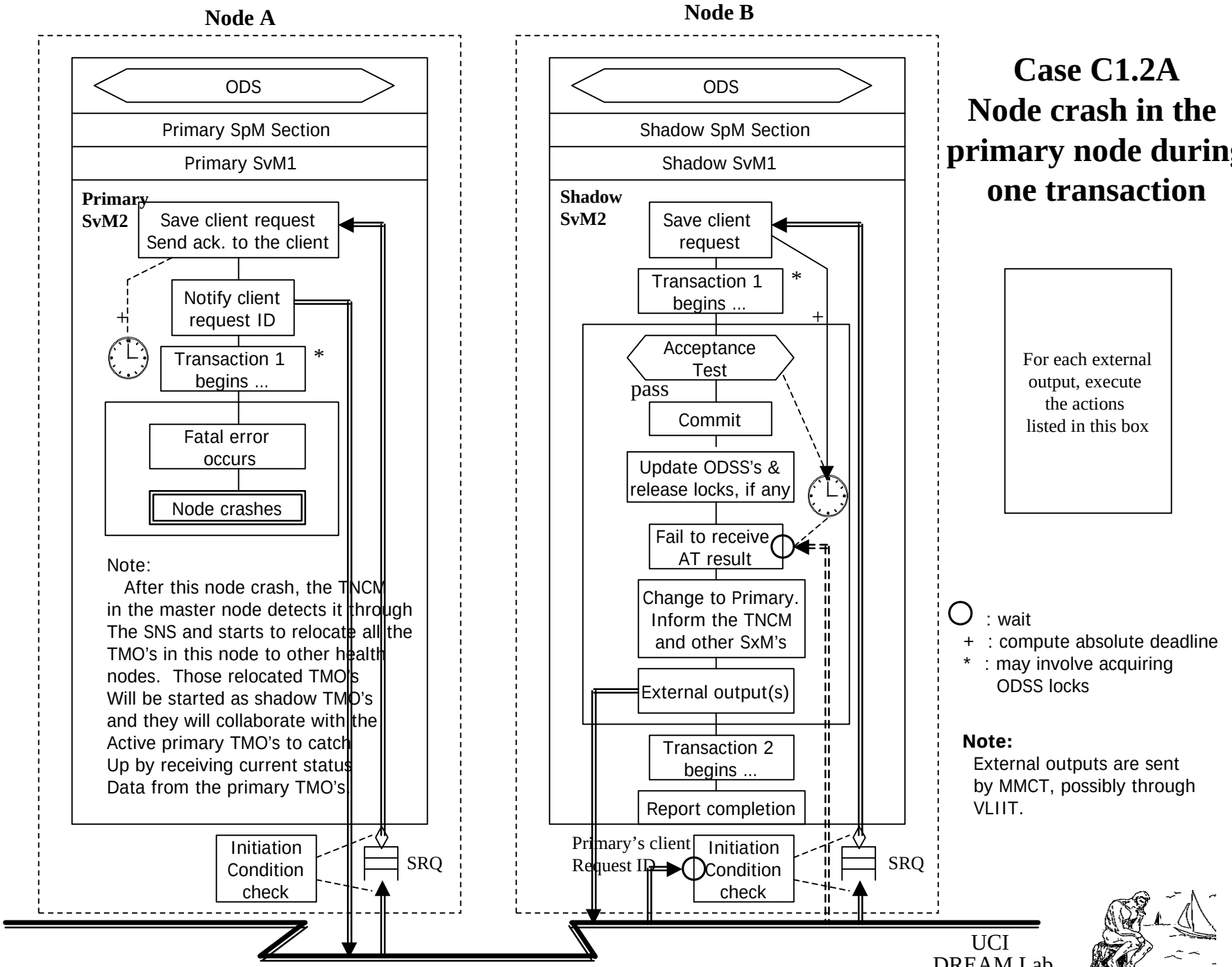


For each external output, execute the actions listed in this box

○ : wait
\+ : compute absolute deadline
* : may involve acquiring ODSS locks

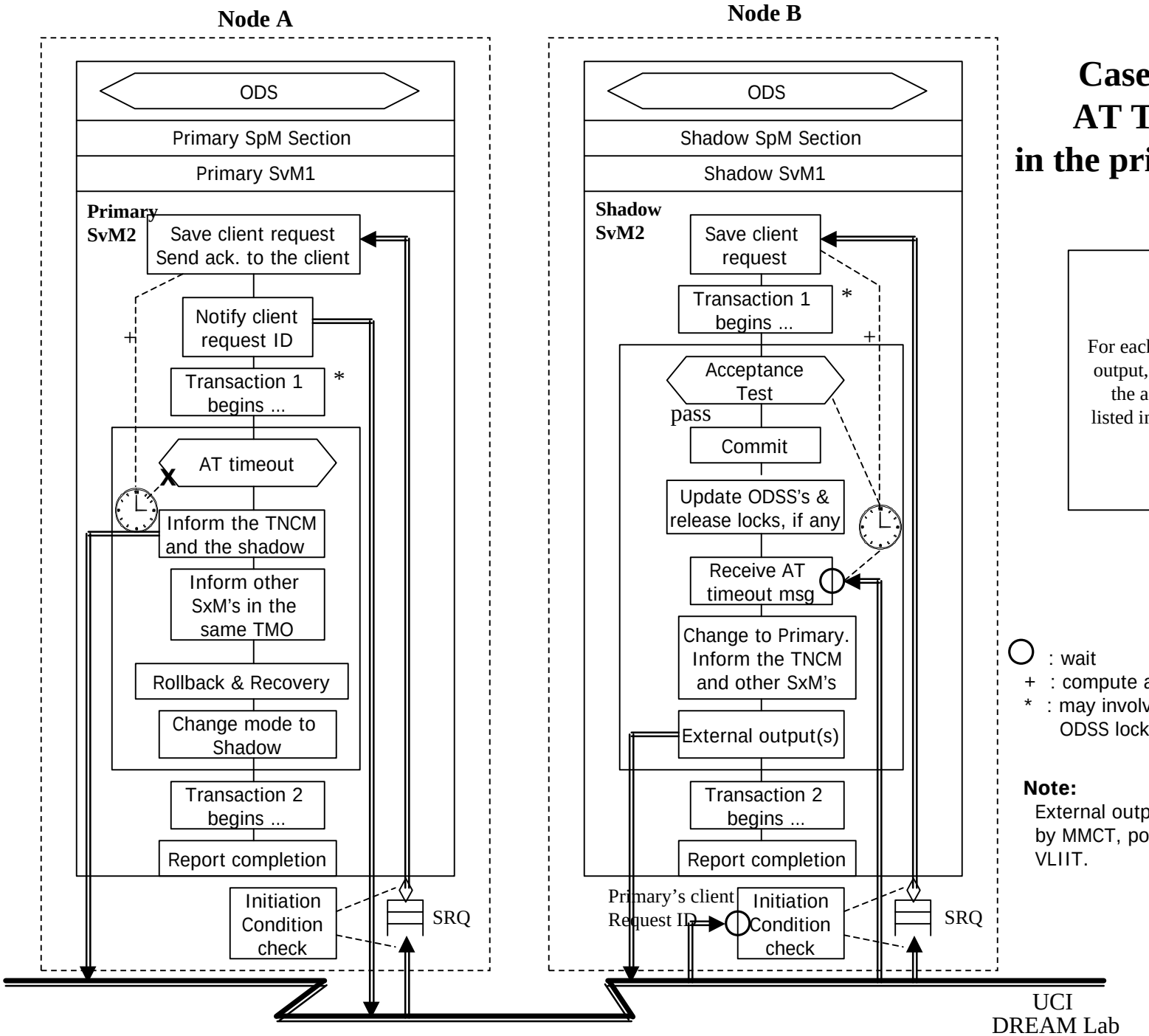
Note:
External outputs are sent by MMCT, possibly through VLIIT.

Case C1.2A
Node crash in the primary node during one transaction



UCI
DREAM Lab

Case C1.2B
AT Timeout
in the primary node

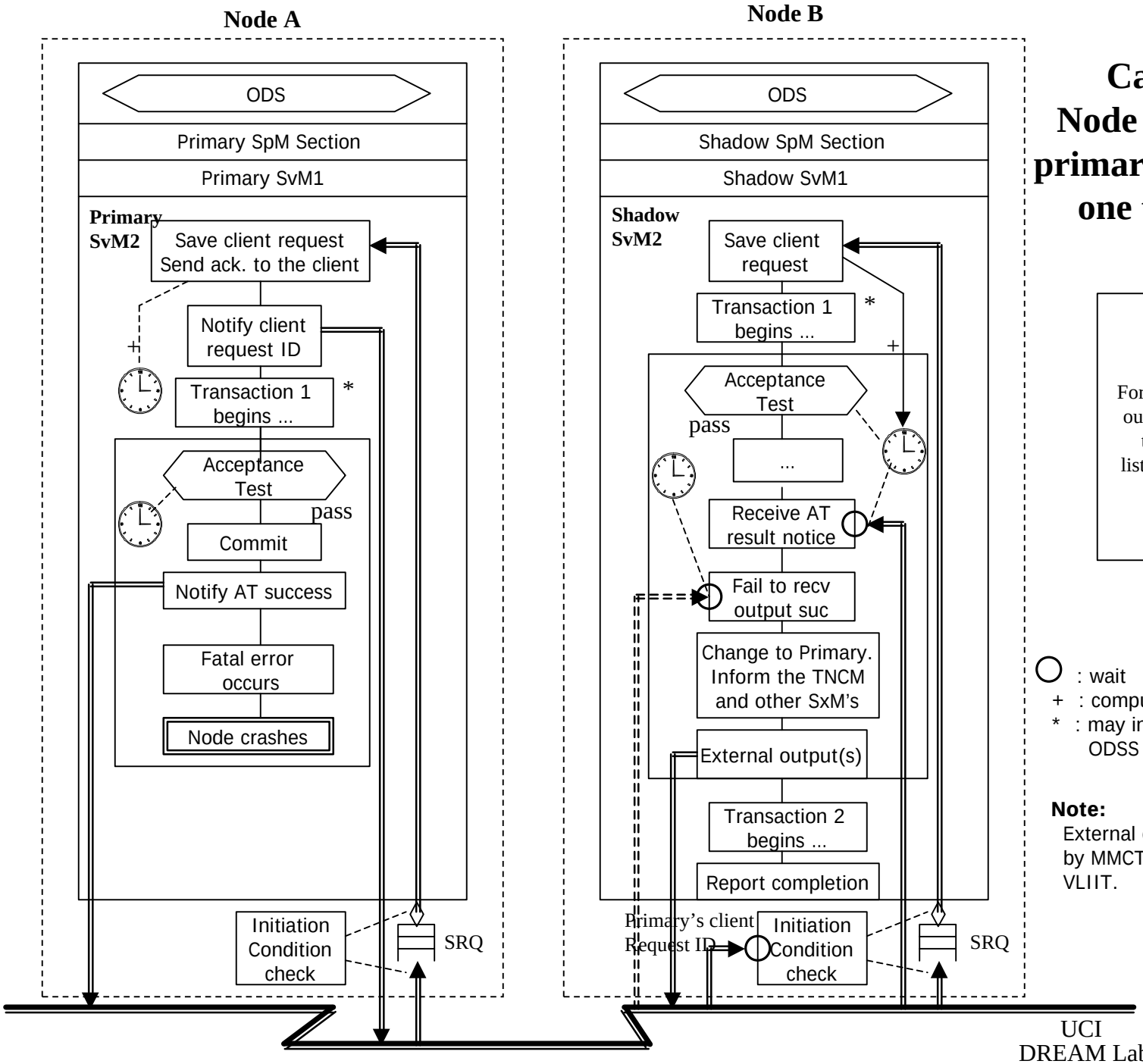


For each external output, execute the actions listed in this box

○ : wait
\+ : compute absolute deadline
* : may involve acquiring ODSS locks

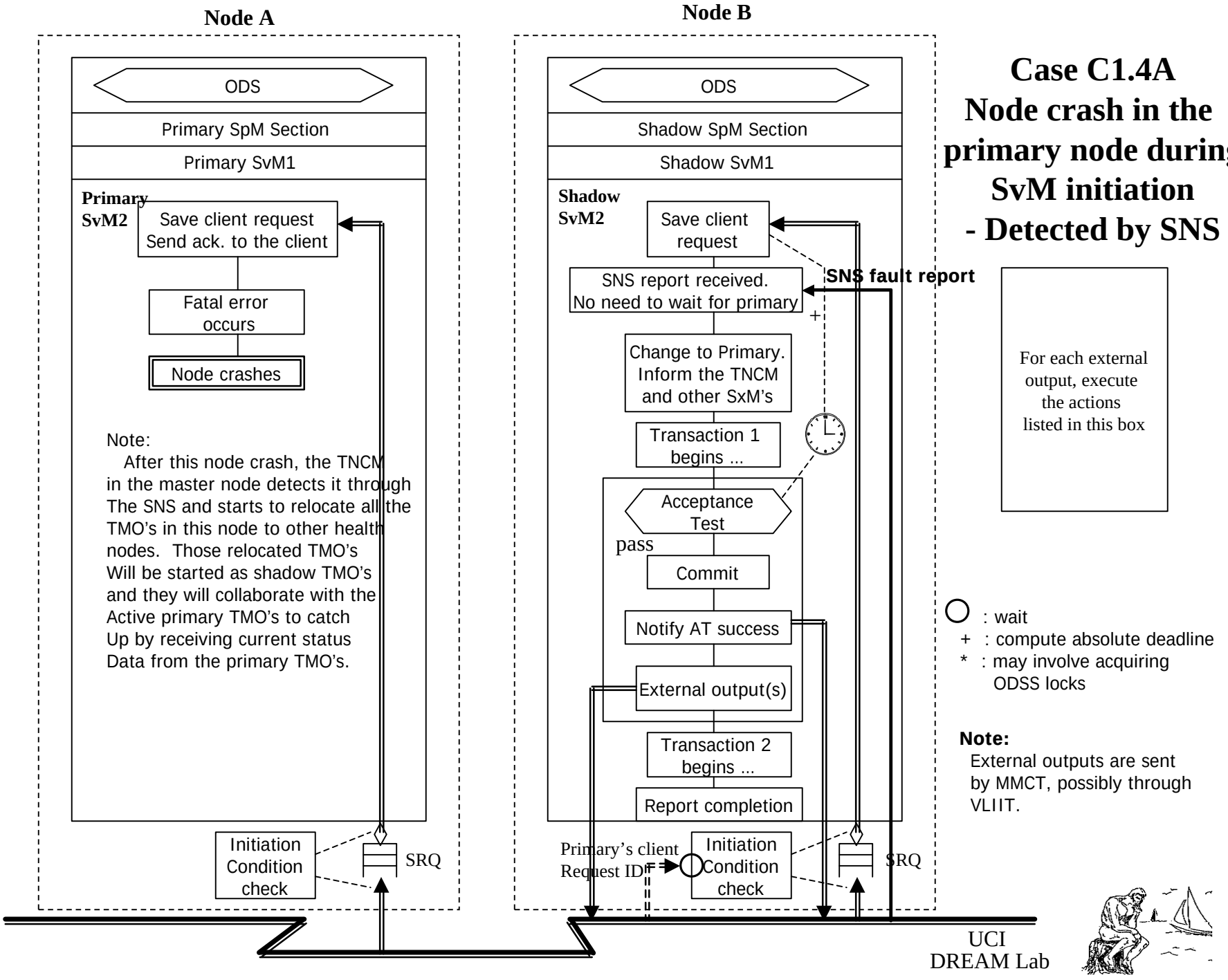
Note:
External outputs are sent by MMCT, possibly through VLIIT.

Case C1.3A
Node crash in the primary node during one transaction

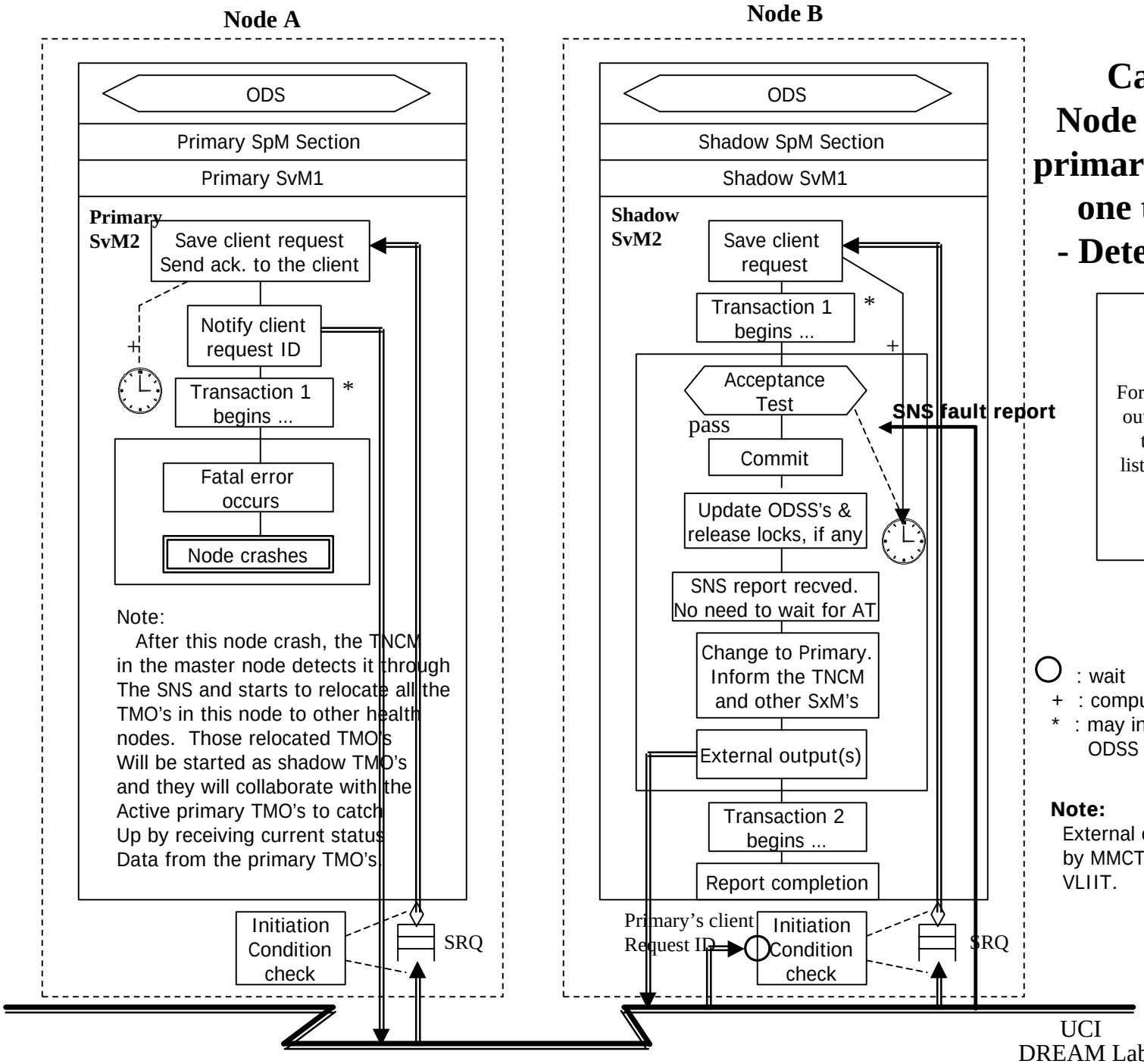


UCI
DREAM Lab

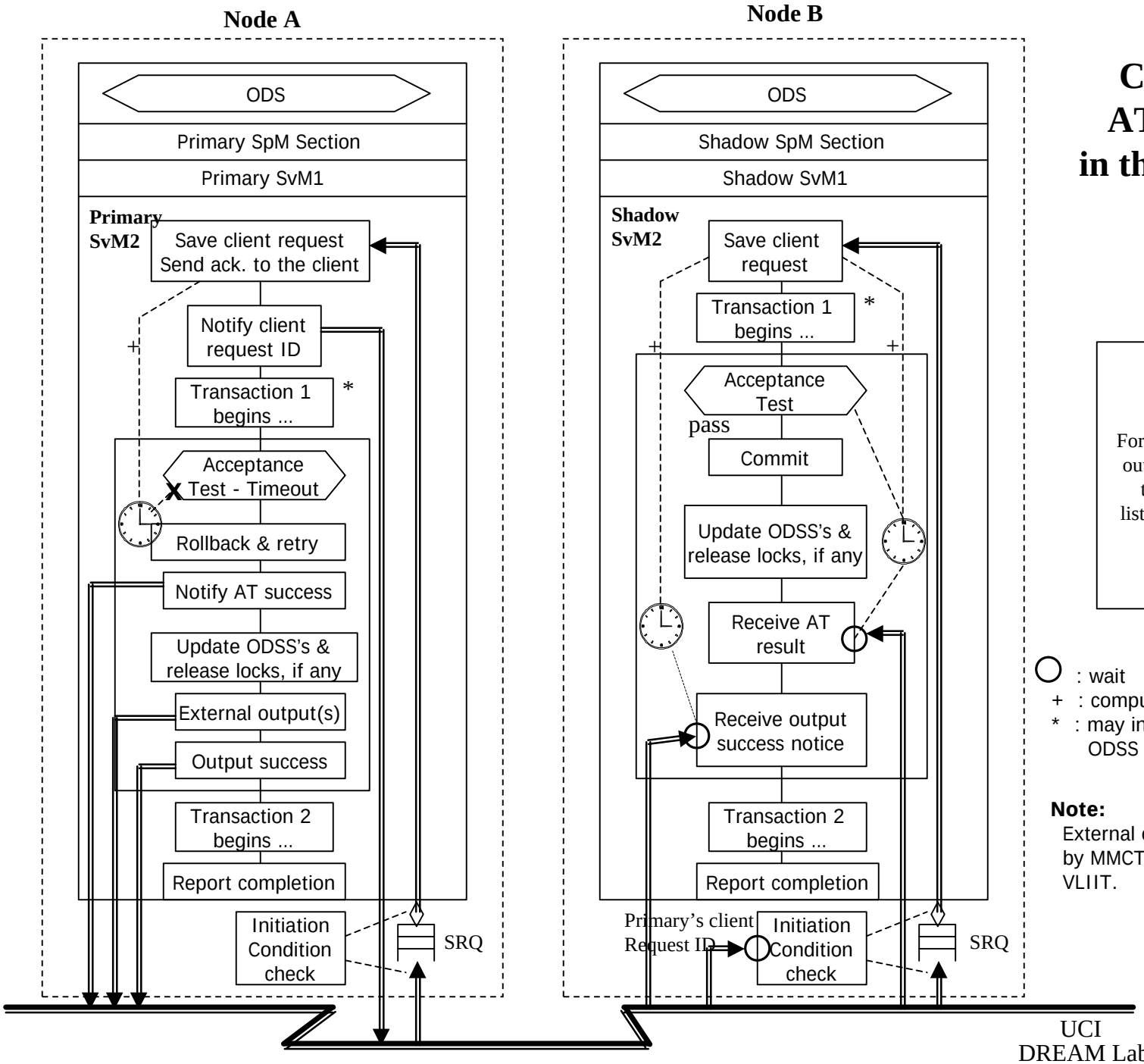
Case C1.4A
Node crash in the primary node during SvM initiation
- Detected by SNS



Case C1.4B
Node crash in the primary node during one transaction
- Detected by SNS



Case C1.7
AT timeout
in the primary

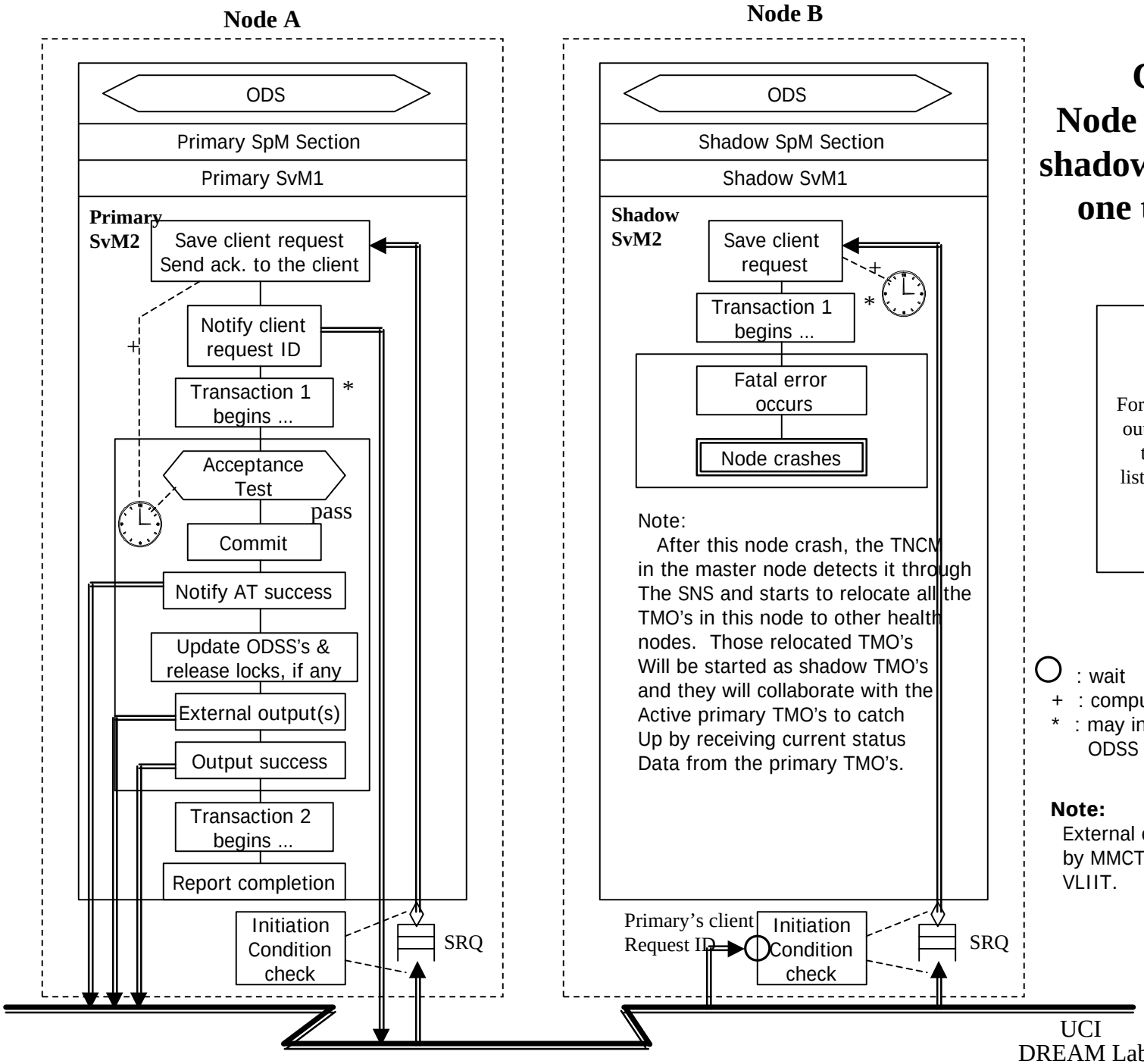


For each external output, execute the actions listed in this box

○ : wait
\+ : compute absolute deadline
* : may involve acquiring ODSS locks

Note:
External outputs are sent by MMCT, possibly through VLIIT.

Case 2.1
Node crash in the shadow node during one transaction

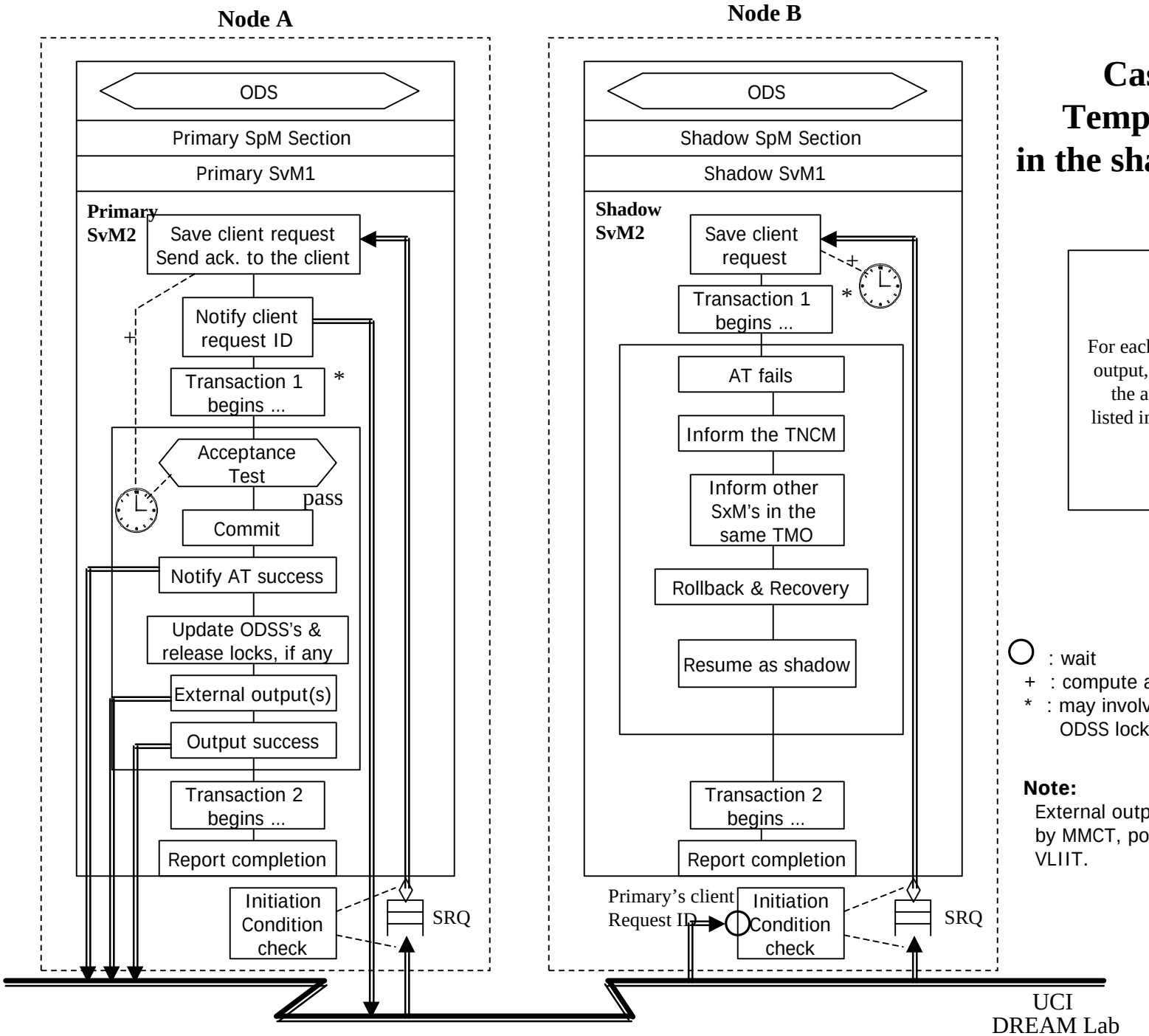


For each external output, execute the actions listed in this box

○ : wait
\+ : compute absolute deadline
* : may involve acquiring ODSS locks

Note:
External outputs are sent by MMCT, possibly through VLIIT.

Case 2.2
Temp failure in the shadow node

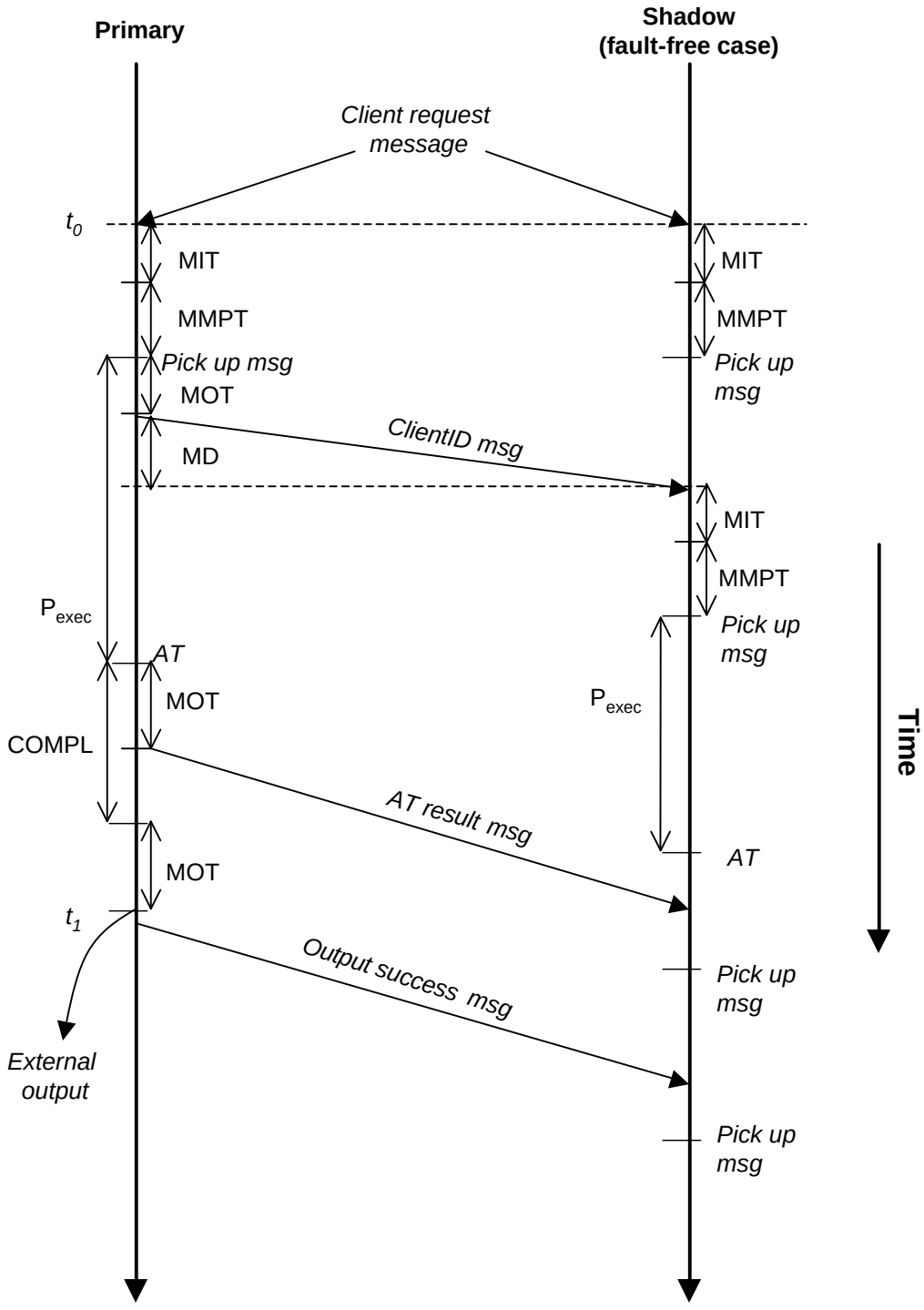


For each external output, execute the actions listed in this box

○ : wait
\+ : compute absolute deadline
* : may involve acquiring ODSS locks

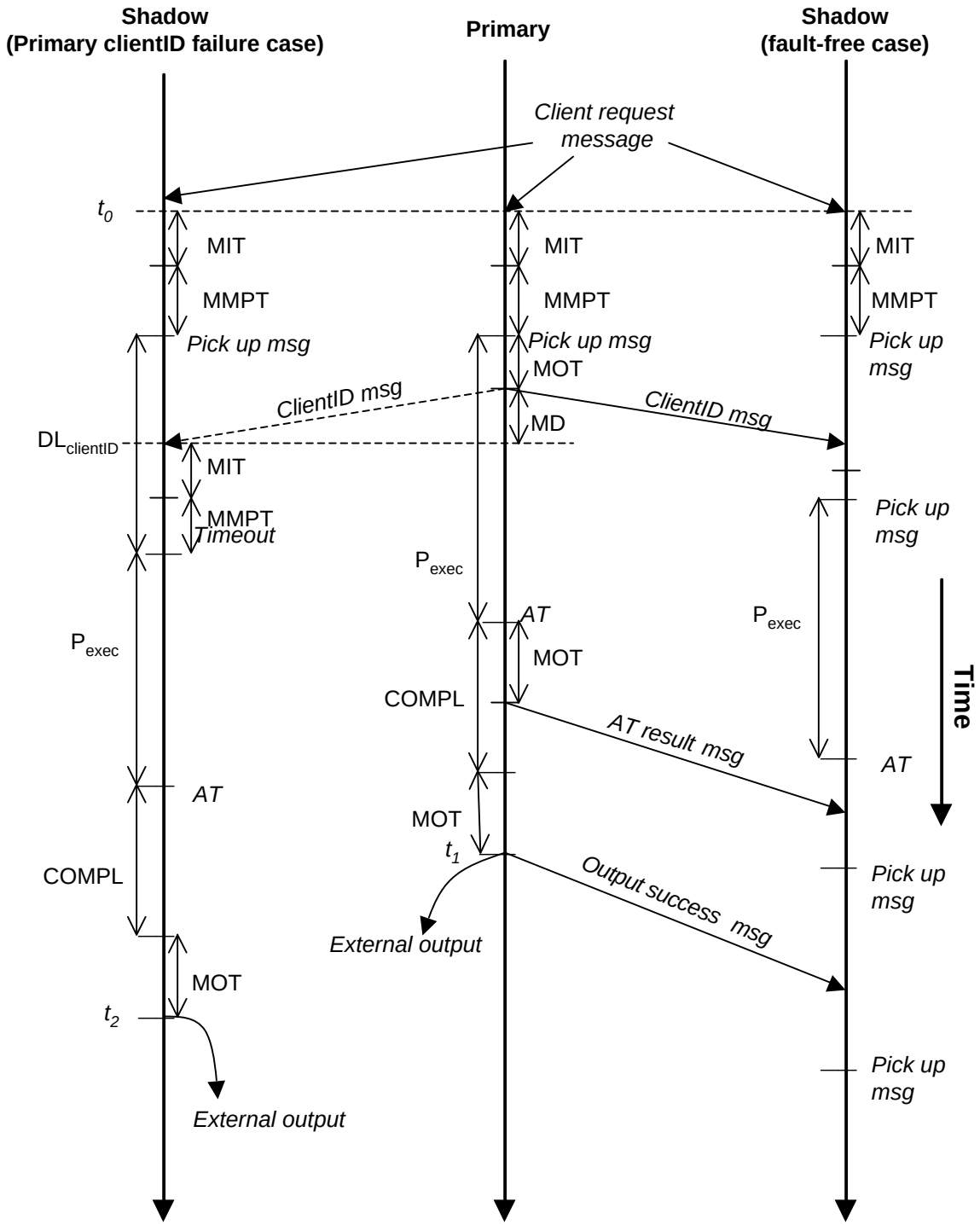
Note:
External outputs are sent by MMCT, possibly through VLIIT.

The PSTR scheme -
Fault detection time bound analysis



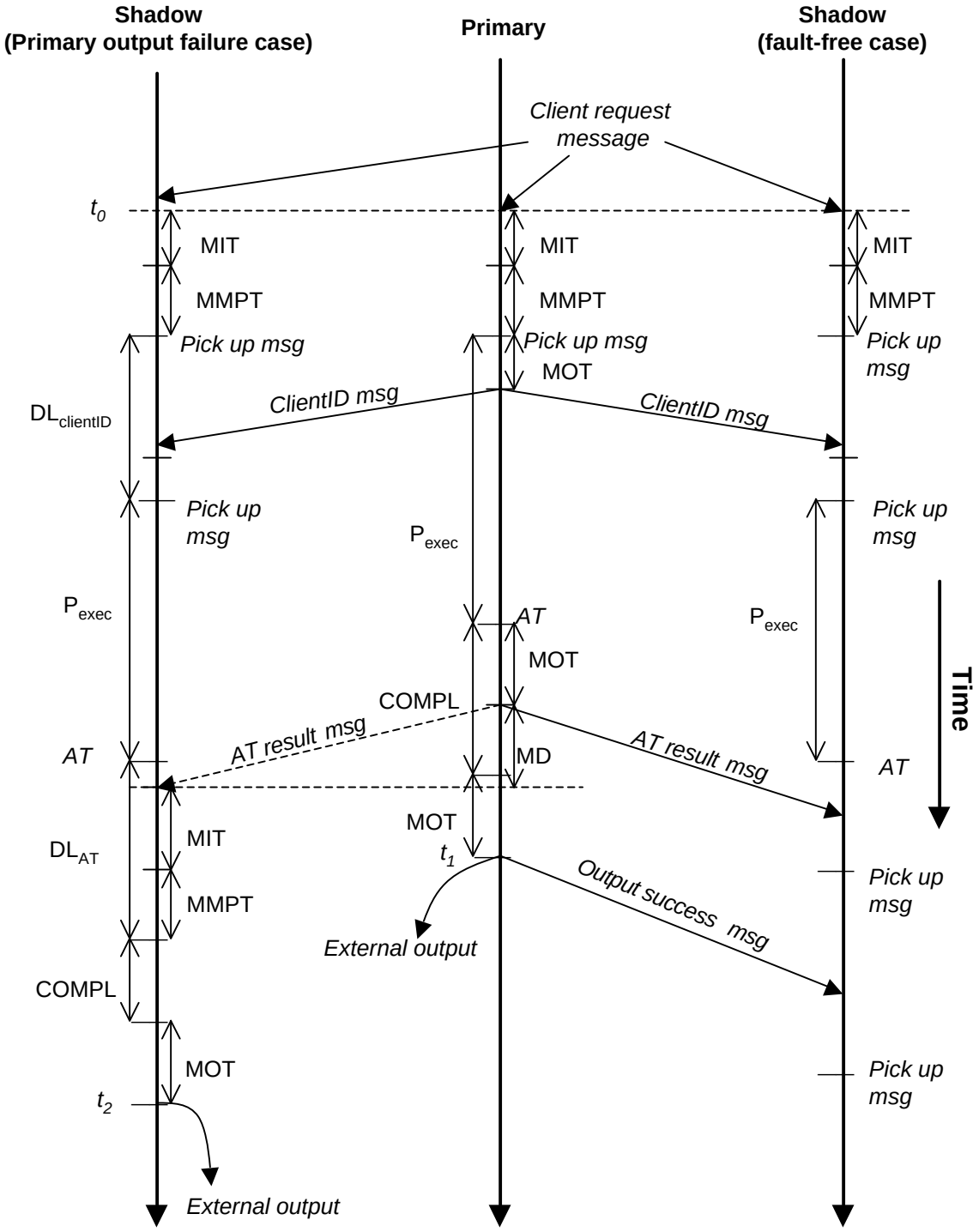
PSTR timing chart
– normal case

The PSTR scheme -
Fault detection time bound analysis



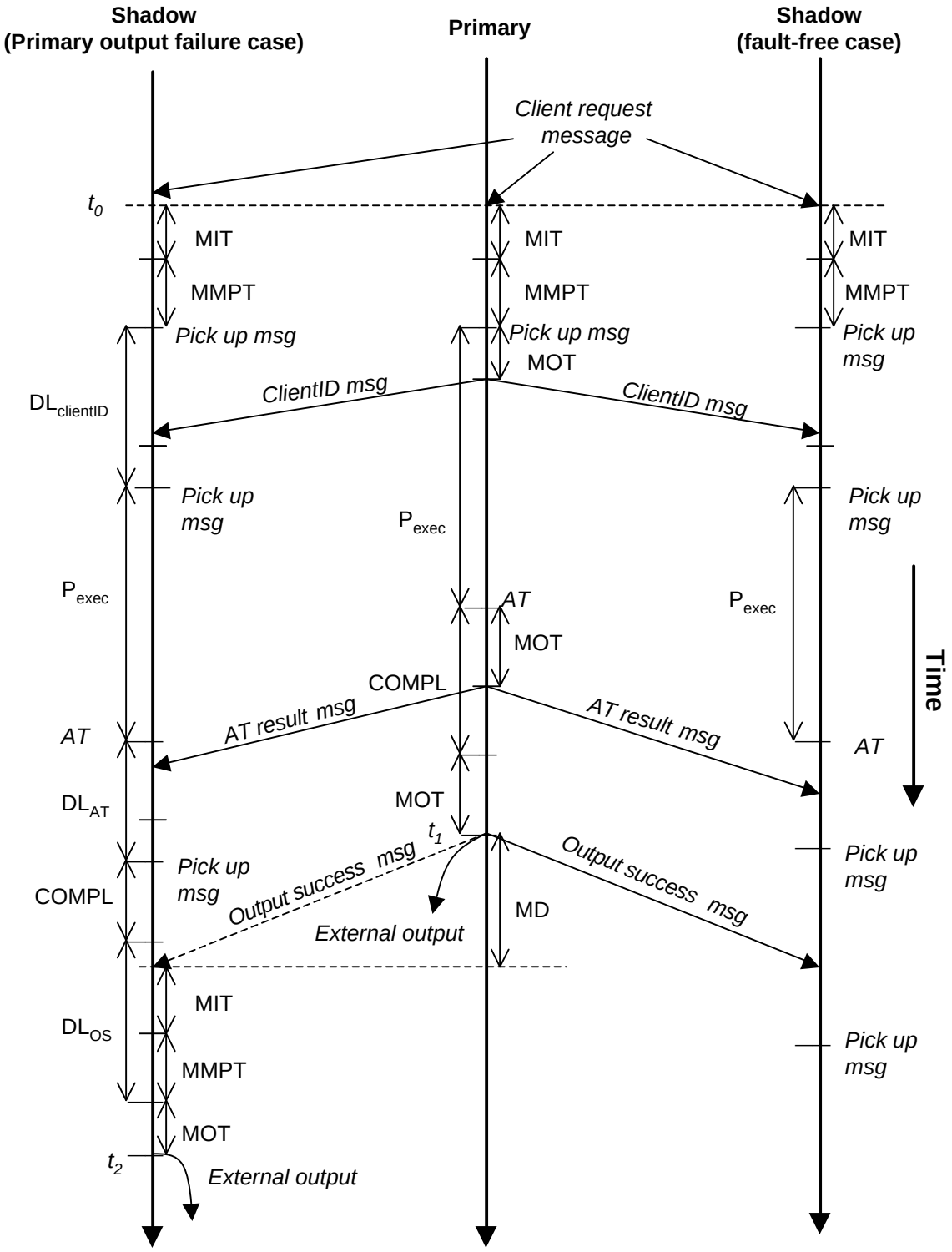
PSTR timing chart
– Primary clientID failure case

The PSTR scheme -
Fault detection time bound analysis



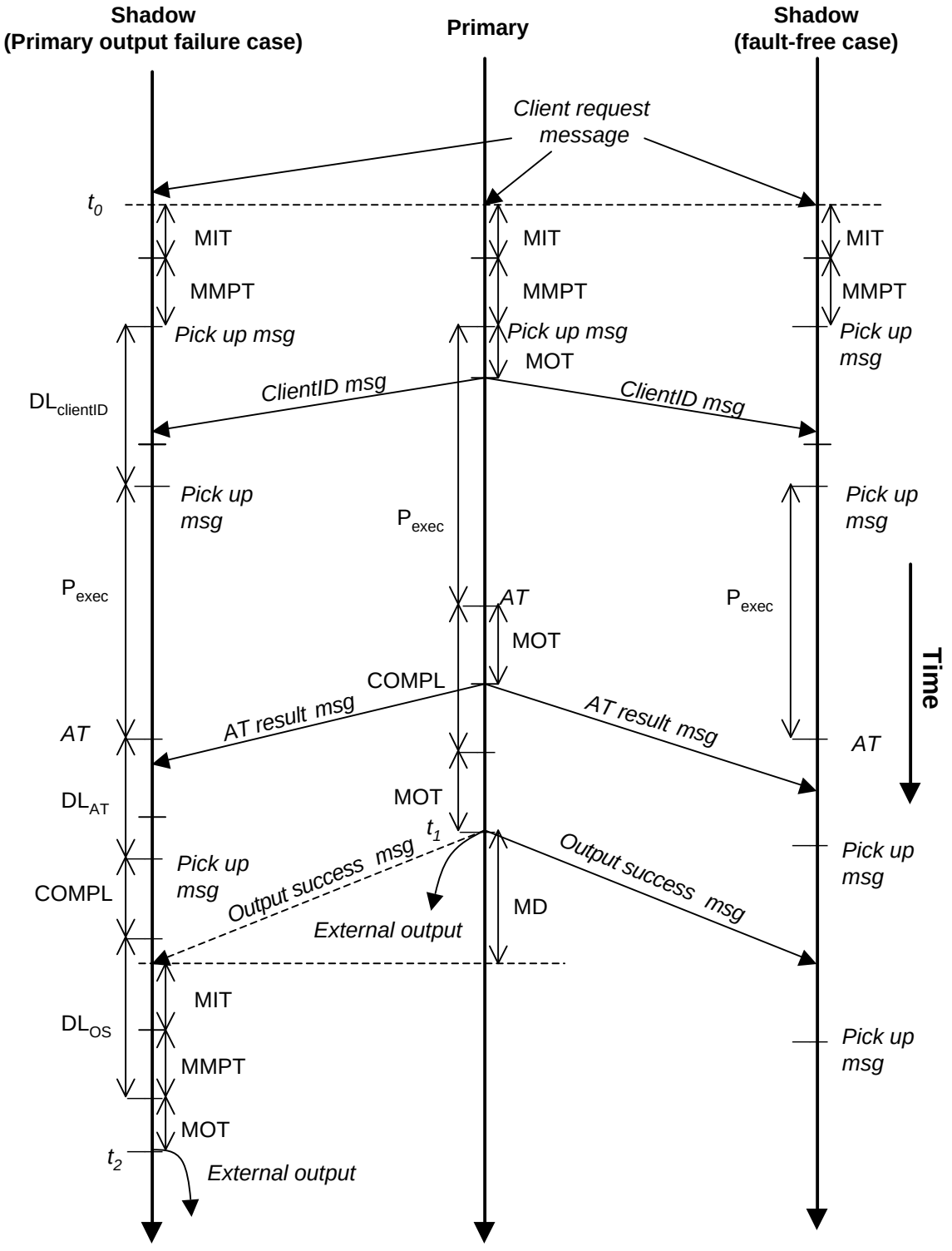
PSTR timing chart
– primary AT failure case

The PSTR scheme -
Fault detection time bound analysis



PSTR timing chart
– external output failure case

The PSTR scheme -
Fault detection time bound analysis



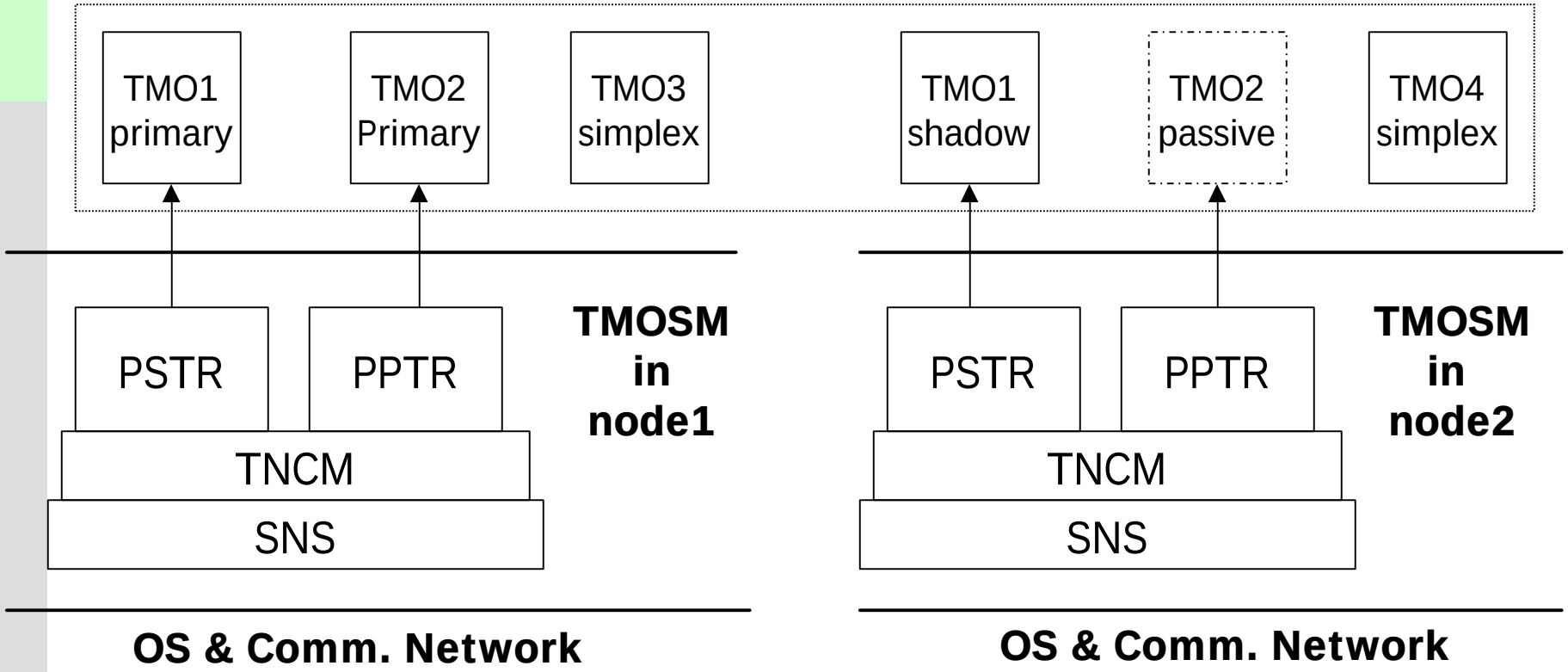
PSTR timing chart
– external output failure case

PPTR -
The Primary Passive TMO Replication Scheme

UCI
DREAM Lab

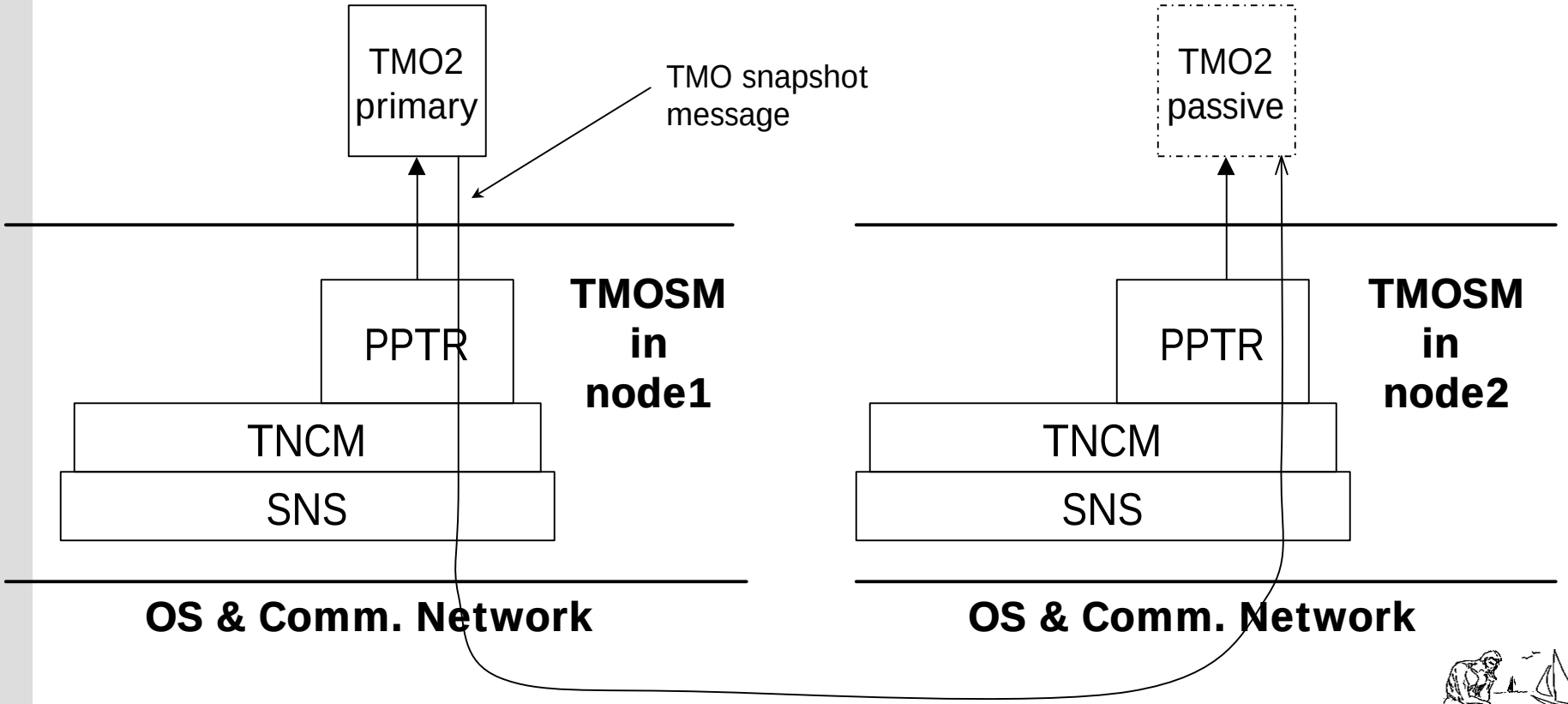
Fault Tolerance Support in TMOSM

- Simplex TMO's
  - no FT support
- Redundant TMO's
  - Active redundant (PSTR)
  - Semi-active redundant (PPTR)



Co-operations between the primary and passive replicas

- The TMOSM supporting the primary replica periodically records the TMO image (Snapshot), and sends it to the TMOSM supporting the passive replica;
- Upon receiving the snapshot of the primary replica, the TMOSM supporting the passive replica updates the passive replica's status;
- In case the node supporting the active replica crashes, the TMOSM supporting the passive replica, which is informed by the SNS subsystem, will convert the passive replica to the primary and start scheduling it.



Fault Tolerance Support in TMOSM

- **Active redundant (PSTR)**
  - More synchronization between redundant replicas;
  - Use more resource (CPU, memory, network bandwidth);
  - Both primary and shadow are active in the normal time;
  - Shadow becomes primary when fault happens in the primary;
  - fault recovery time is short;
  - After switch, the new primary continue its execution;

- **Semi-active redundant (PPTR)**
  - Less synchronization between redundant replicas;
  - Use less resource (CPU, memory, network bandwidth);
  - Only one replica, primary is active in the normal time;
  - passive replica becomes active when fault happens in the primary;
  - Fault recovery time is long;
  - After switch, the new primary replica starts from the last checkpoint;

The contents of a TMO snapshot

Ideally, a snapshot of TMO should consists of the following data:

1. **Global data**
   - ODSS
   - Heap data (Should not be used in a TMO program)

2. **Local data**
   - Local variables in the stack

3. **Current Thread context and CPU register value for each SxM**

4. **Un-processed service request from the client**
   - SRQ
   - MMCT inputQ
   - BlockedForMsgQ

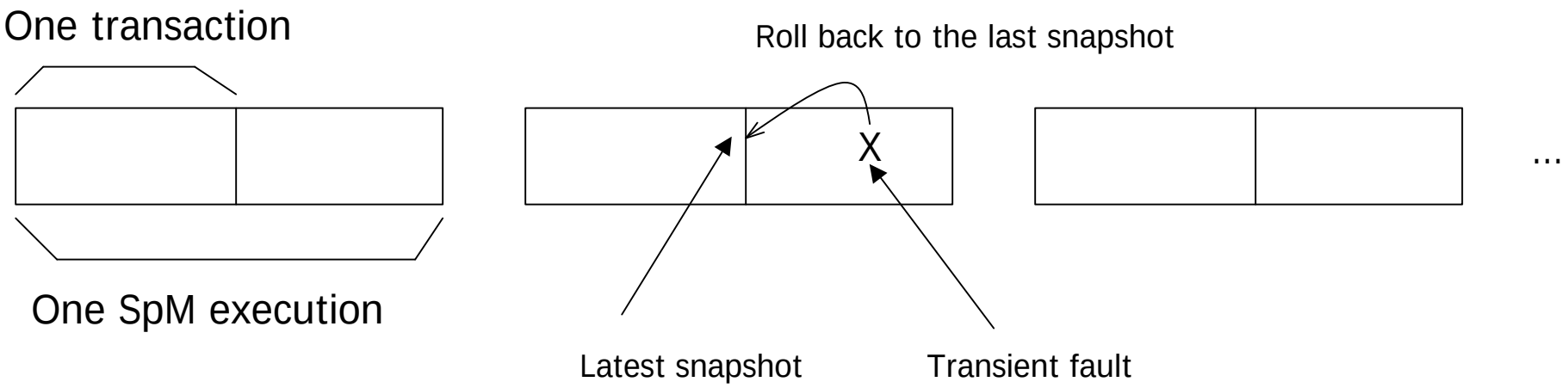
Note:

Saving & recovering 1 and 4 are easy, but saving & recovering 2 and 3 are difficult. The reason is that 2 and 3's data are only meaningful within a process, but we may need to migrate some TMO's to another node or process.

Fault recovery (PPTR)

Case 1.1 Transient fault

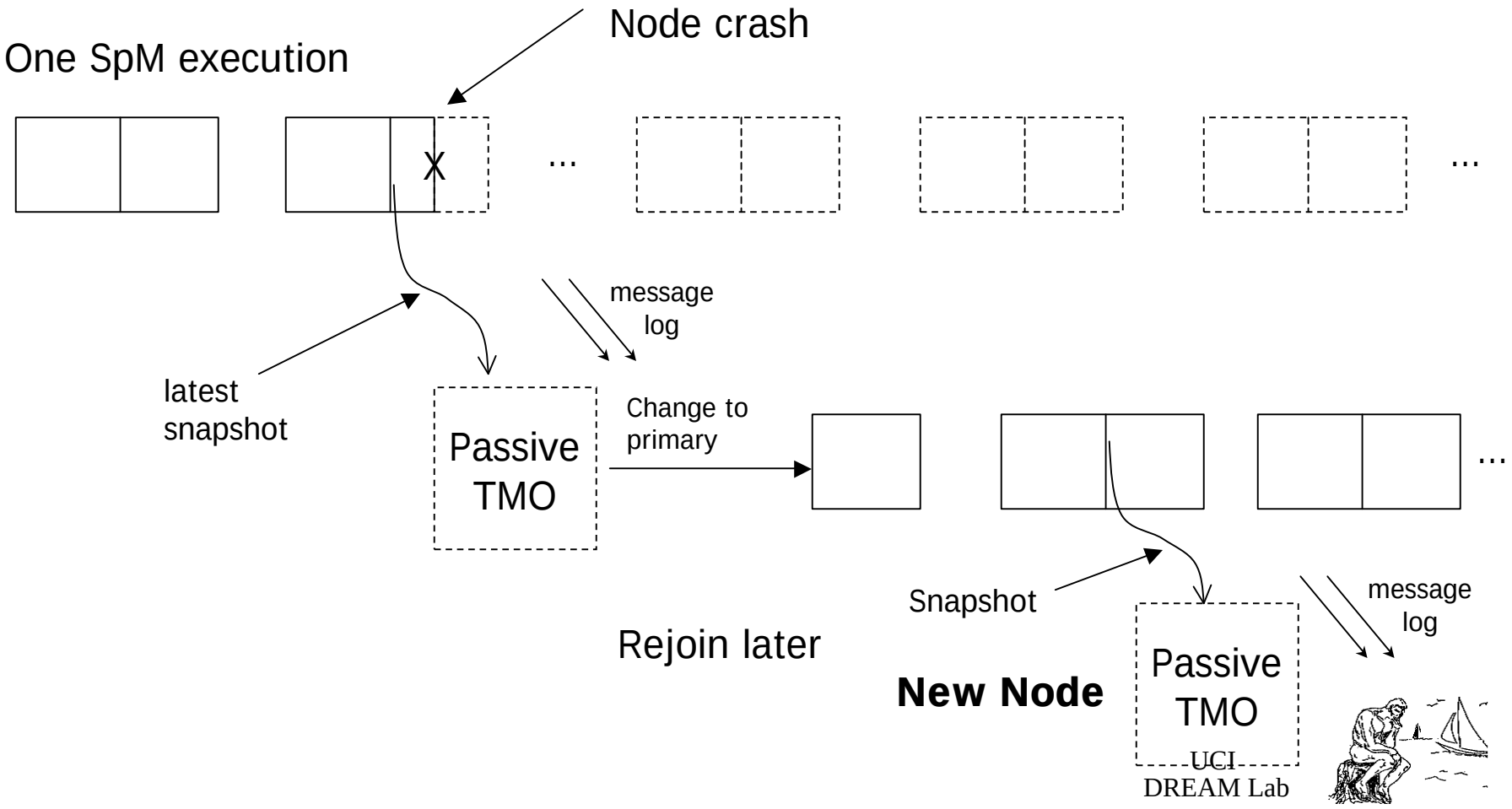
- recovered by a local rollback to the last snapshot



Fault recovery (PPTR)

Case 1.2 Node crash (and node rejoin)

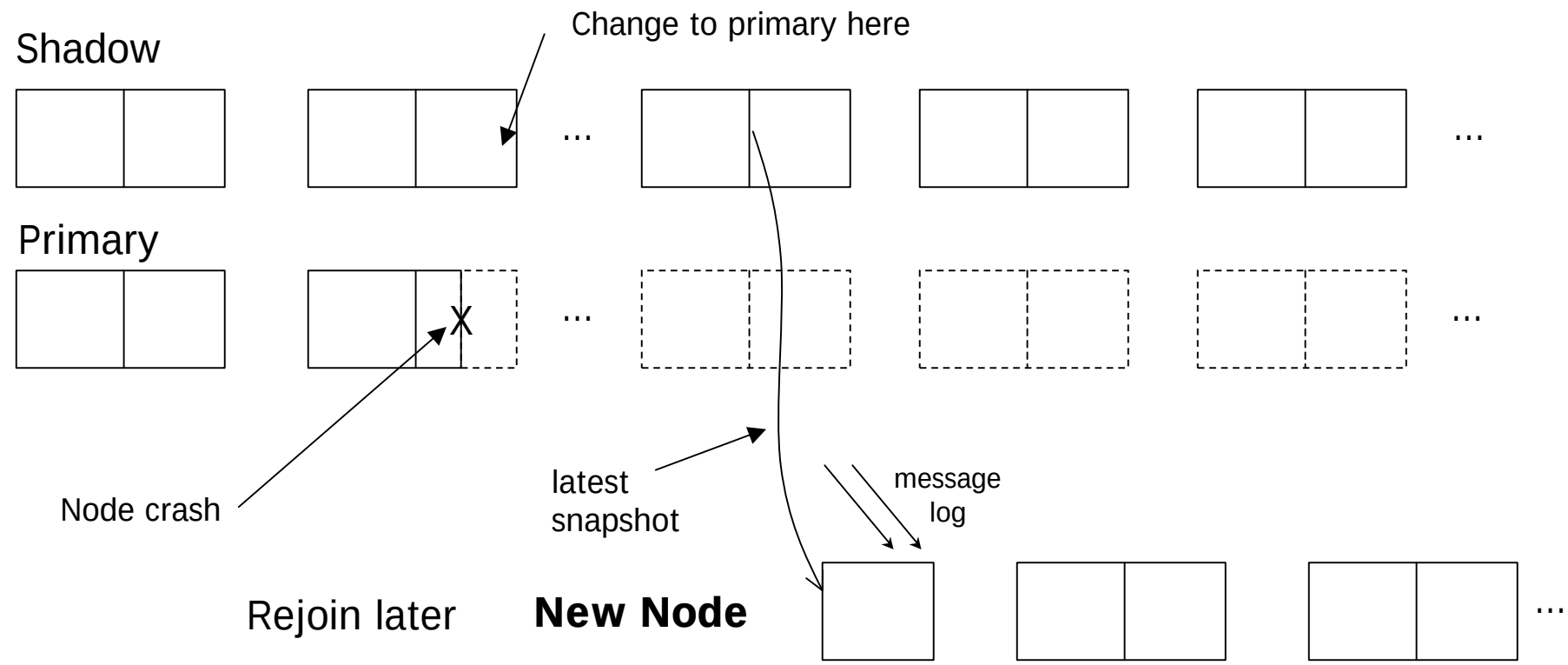
- recovered by convert the passive replica to the primary



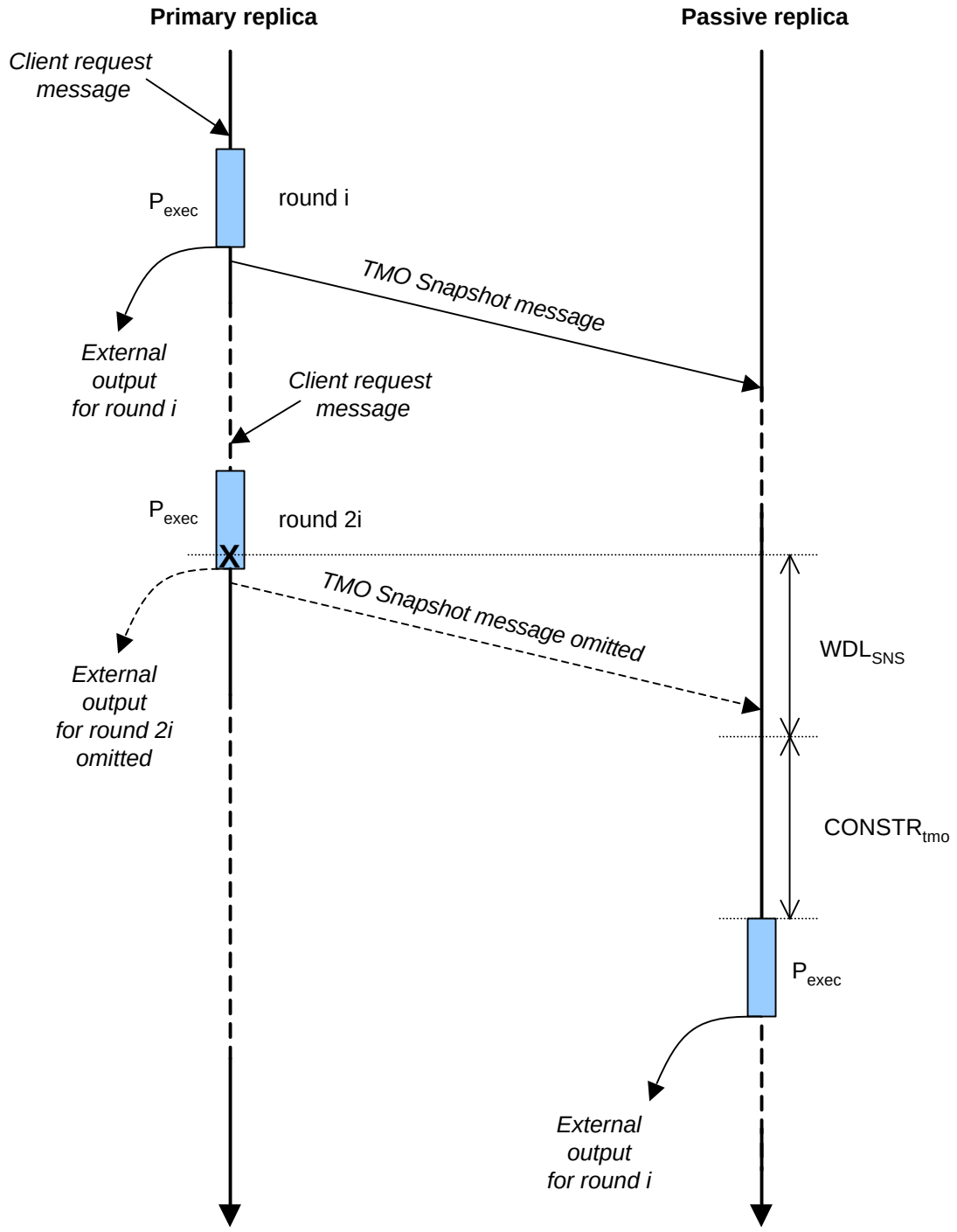
Fault recovery (PSTR)

Case 1.3 Node crash (and node rejoin)

- recovered by converting the shadow to the primary



The PPTR scheme -
Fault detection time bound analysis



PPTR timing chart

Conclusion

- A middleware architecture, named TMOSM (time-triggered message-triggered object support middleware), has been established to support the development and execution of the distributed real-time safety-critical applications, and the RIF-based resource allocation framework and the real-time fault tolerance schemes have been incorporated into it.

- RIF framework is a multi-level framework that covers from the application QoS requirement specifications to the scheduling algorithms of various computation resources, supporting multiple QoS dimensions, such as timeliness, fault tolerance and deadline handling. RIF-based resource allocation scheme is a major improvement from the current practice.

- The RIF-based resource allocation framework incorporates two real-time fault tolerance schemes, PSTR/SNS (Primary Shadow TMO Replication / Supervisor-based Network Surveillance) and PPTR/SNS (Primary Passive TMO Replication / SNS) schemes. The main strength of the SNS scheme and the implementation are in that they enable relatively easy determination of tight bounds on the fault detection latency.

Future Research Directions

- Implementations of TMOSM on other COTS platforms, such as WinCE, UNIX, Linux, and other distributed computing support environments, such as DCOM, .Net, and real-time Java Virtual Machine.

- For RIF-based resource allocation framework, more works on tools which help the application developer to derive the RIPF set from the RIF set are needed. Some other QoS dimensions not covered currently, such as dynamic reconfiguration and security, can be pursued as a future research direction.

- Searching for better scheduling algorithms based on RIPF and the integration of the scheduling decisions of processor, communication network bandwidth and I/O devices are very promising research issues also.

- For real-time fault tolerance scheme, a passive replication in which case the passive replica does not interact with the primary replica and consume any resources during normal operation time, can be considered to be incorporated into the current framework.