MCC: A dynamic verification tool for MCAPI user applications

Subodh Sharma, UofU
Ganesh Gopalakrishnan, UofU
Eric Mercer, BYU
Jim Holt, Freescale

Supported by NSF CCF 0903408/0903491 and SRC 2009-TJ-1993/1994

Concurrency Space in Multicore Era

	S/W solution	H/W solution
Cloud/HPC Computing	MPI, PVM, Mapreduce, MSCS, Dryad	Clusters, Vector machines, Supercomputers
Desktop computing	OOSD, Scripting, Pthreads, TBB, CT, OpenMP	Desktop machines
Embedded Computing	Lightweight counterparts of the above.	FPGAs, DoC, SoC, etc.

Formalize Emerging Communications API in the Embedded Space

	S/W solution	H/W solution
Cloud/HPC Computing	MPI, PVM, Mapreduce, MSCS	Clusters, Vector machines, Supercomputers
Server/Desktop computing	OOSD, Scripting, Pthreads, TBB, CT, OpenMP	Desktop machines
Embedded Computing	**Lightweight counterparts of the above.**	**FPGAs, DoC, SoC, etc.**

Formalize Standards, build Query Oracle, Derive Tests

Build Dynamic Formal Verifier for Applications

Demonstrate and Evaluate Prototype Solutions

MCAPI (Multicore Communication API)
Introduction

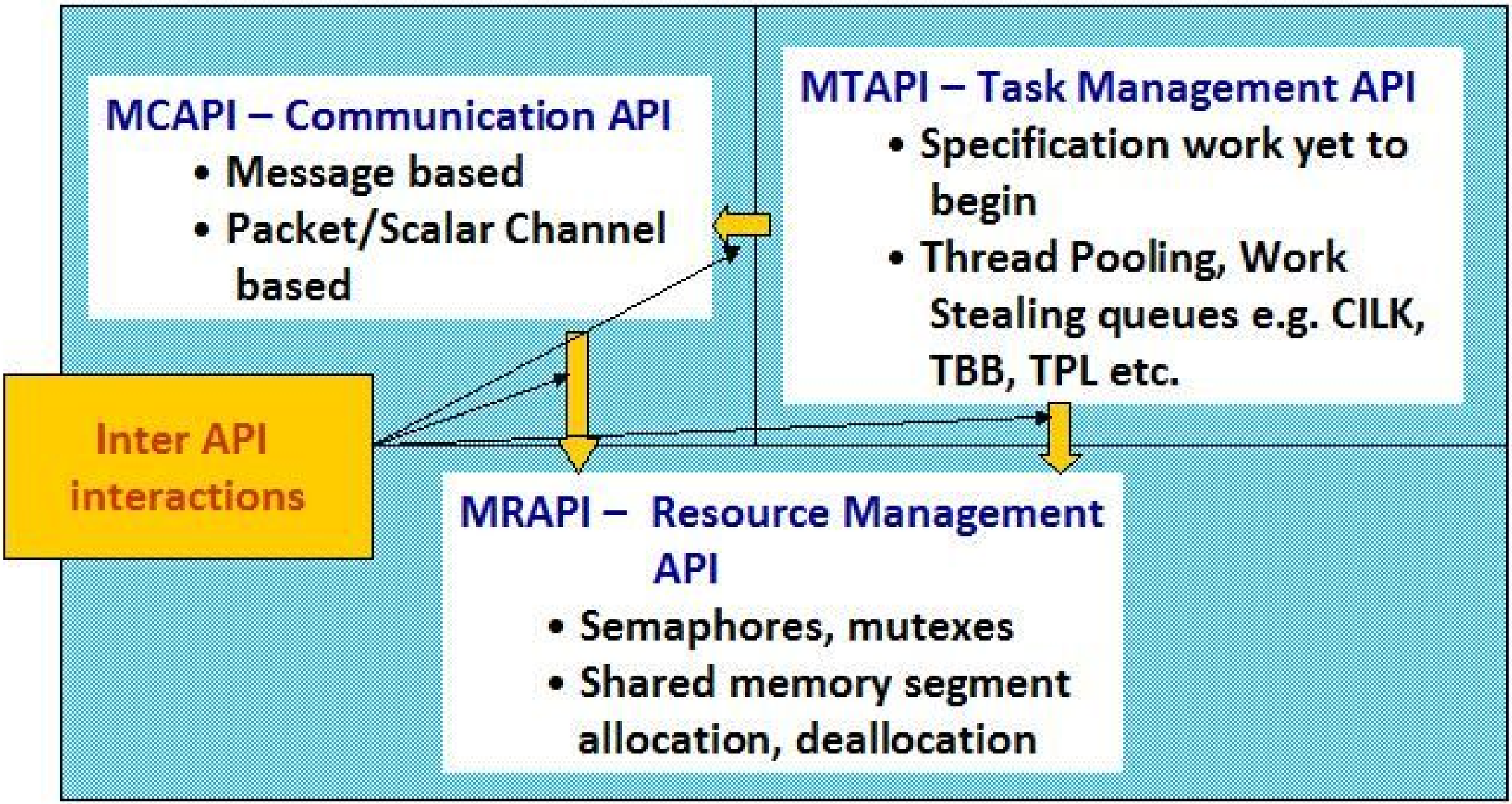
What is MCAPI (Multicore Communication API)?

- An API specification from MCA (Multicore Association)
  - Member companies – Freescale, Samsung, Intel, etc.

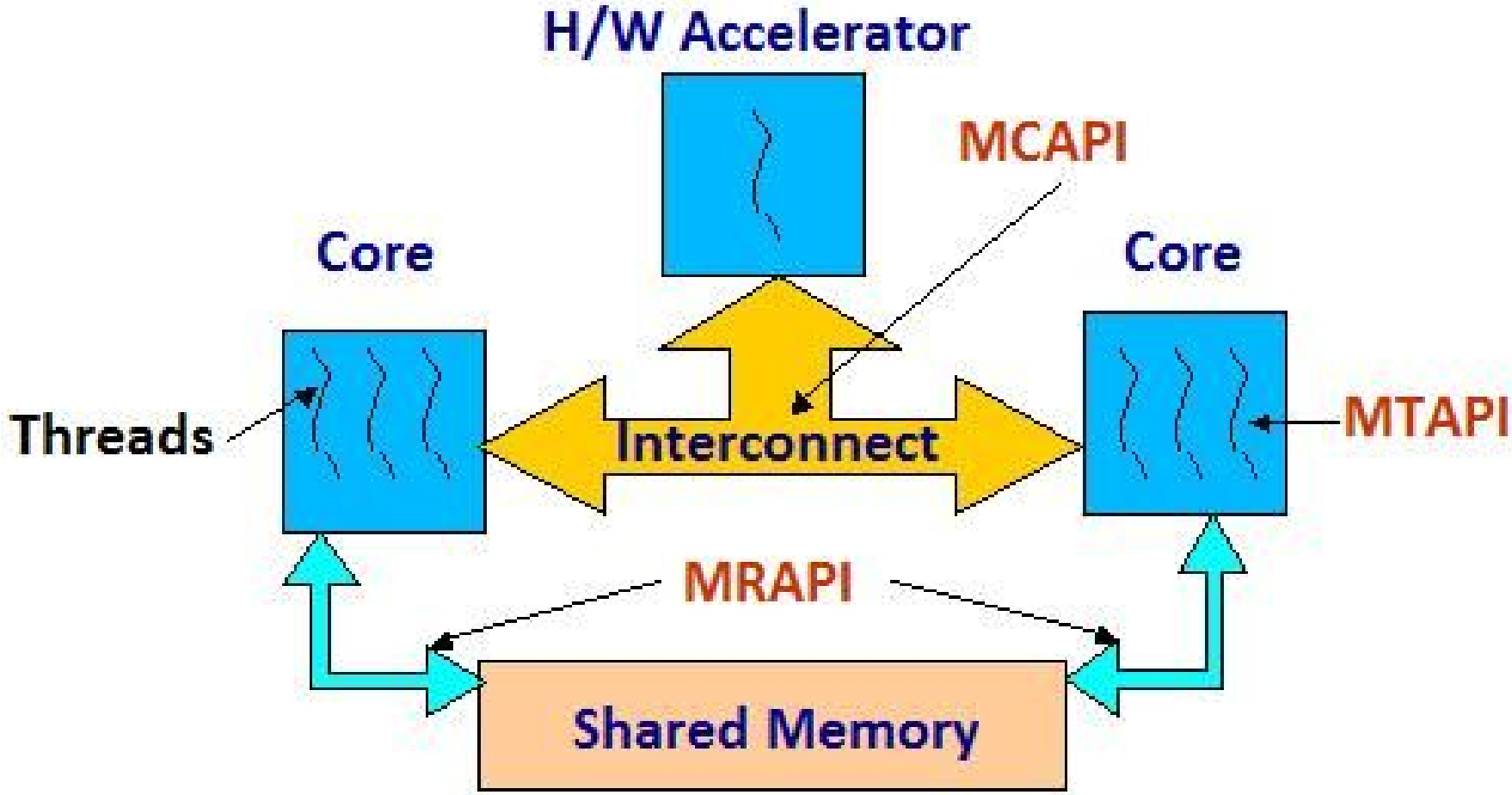
- To program embedded systems like mobile phones, PDAs, routers, servers, etc.

- Not restricted to SPMD (like MPI) or multi threaded style of programming.

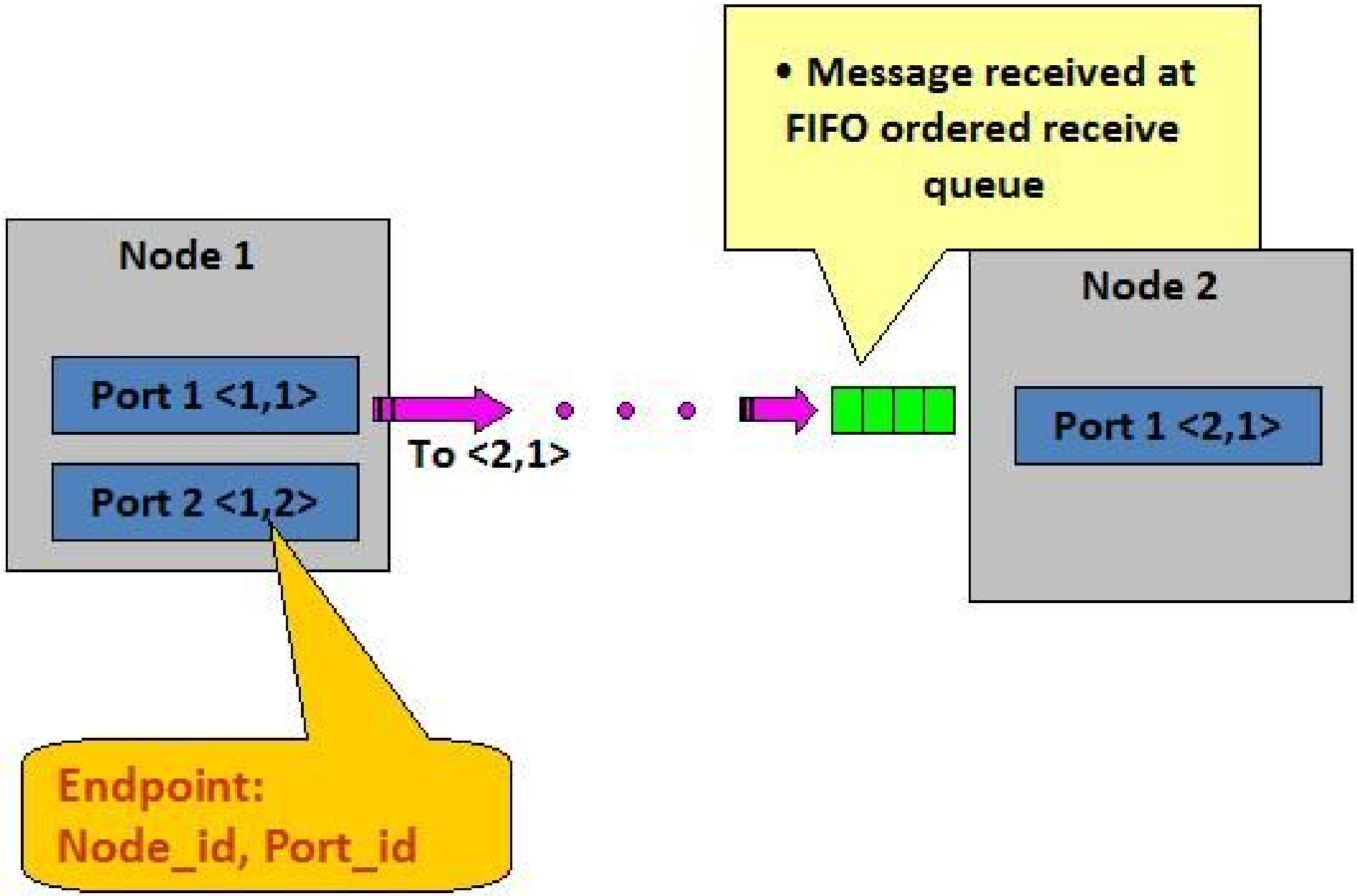
Taxonomy for MCA APIs



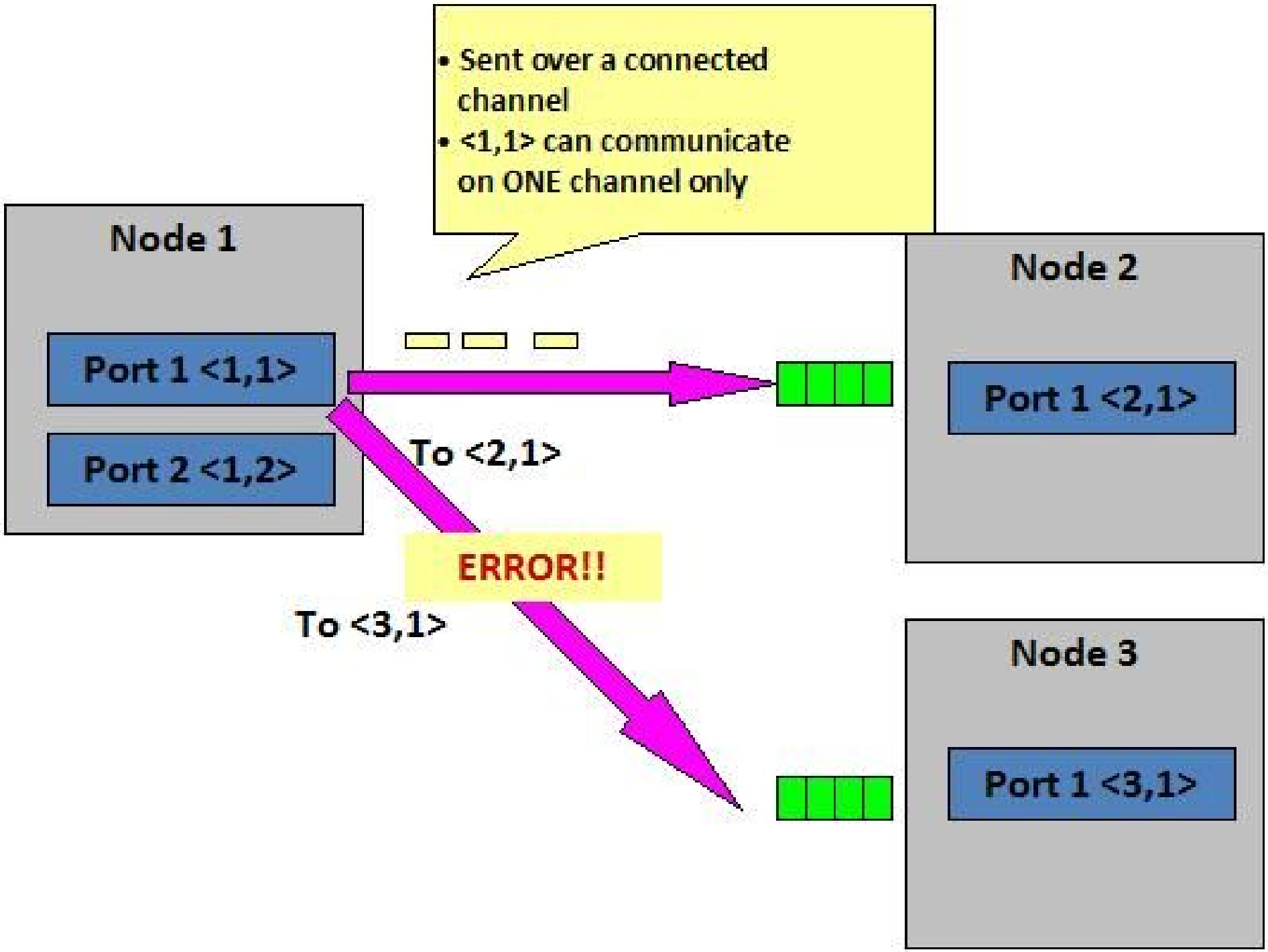
Example usage scenario of MCA APIs



MCAPI Messages



MCAPI Connection Oriented Communication

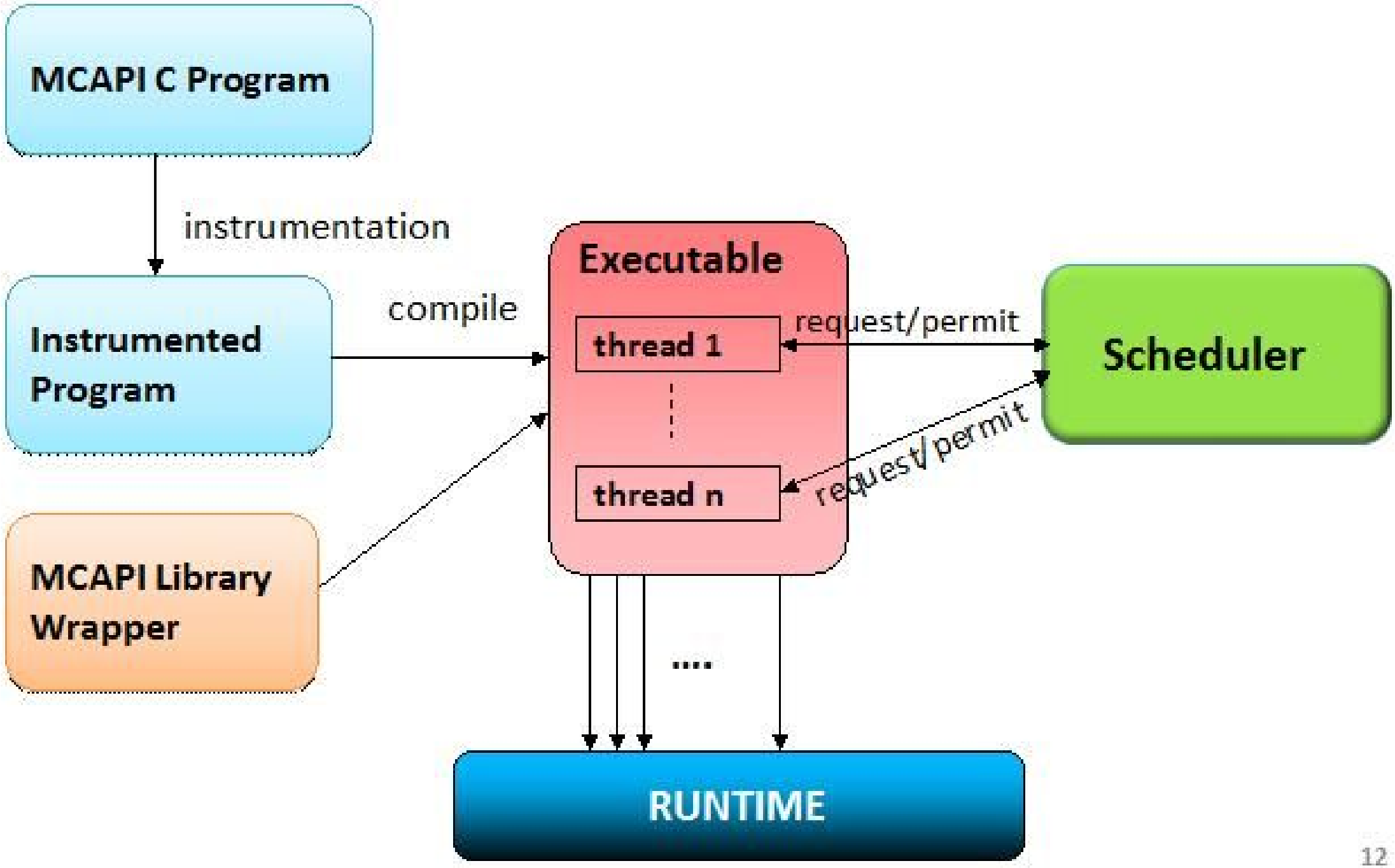


Our Contributions!

Early Engagement of FV in the MCAPI space

- To promote early adoption of the API
- To promote better programming practices
- To help avoid early pit-falls – e.g. unspecified behaviors

MCC - MCAPI Checker



Related Work

- Tools in this domain work on directly work on unmodified sources - inspired by Verisoft.

- Tools control the scheduling to achieve the goal of coverage guarantees

- Examples are CHESS from Microsoft, Inspect and ISP from Univ of Utah, etc.

Unique features of MCC

- External schedulers may not be able to exercise control over runtime.
    - Novel way of enforcing deterministic match of transitions at runtime
- Instruments Pthread calls (i.e. would support hybrid programs written in future using MCAPI)

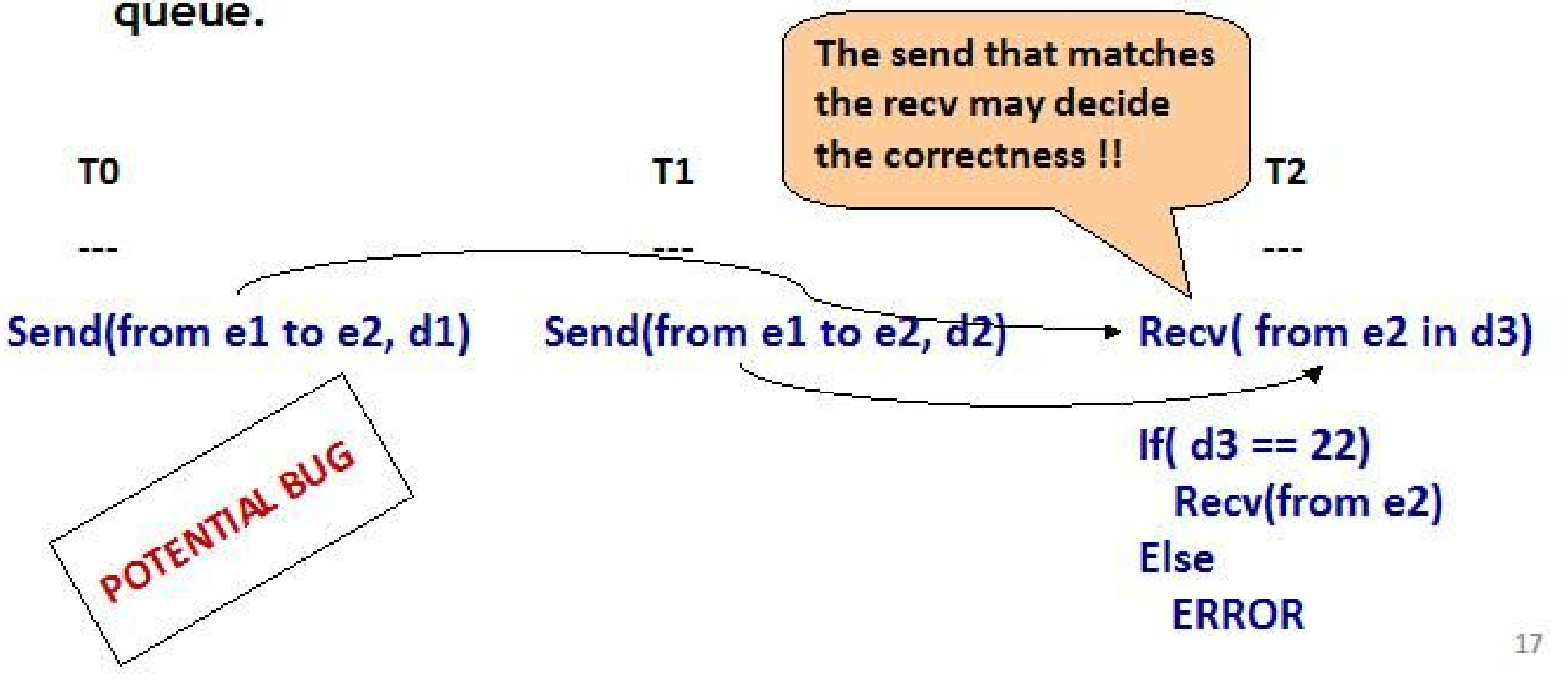
MCAPI API calls supported by MCC

Initialize/Finalize	Sets and deletes the environment. Must be called by each communicating node.
Create/Delete endpoint	API calls to manage creation/deletion of endpoints on the owner node
Get_endpoint	A blocking call to get the address of a remote endpoint.
Send (sndEp, rcvEp, ..)	Sends a data-payload from a sending endpoint to a receiving endpoint. API provides blocking and non blocking versions.
Recv (rcvEp, ..)	Receives a data payload from the receiving endpoint. API provides blocking non-blocking versions.
Wait (reqHndl, ..), Test(reqHndl,..)	Checks the completion of the request. Wait – Blocking; Test – Non blocking

Bugs in the application space?
Why Dynamic Analysis?
Why Deterministic Execution Control?

Non-deterministic MCAPI Receive Calls

- The recv() call is passed with only receiving endpoint as a parameter.
- The receiving endpoint extracts message from the FIFO receive queue.



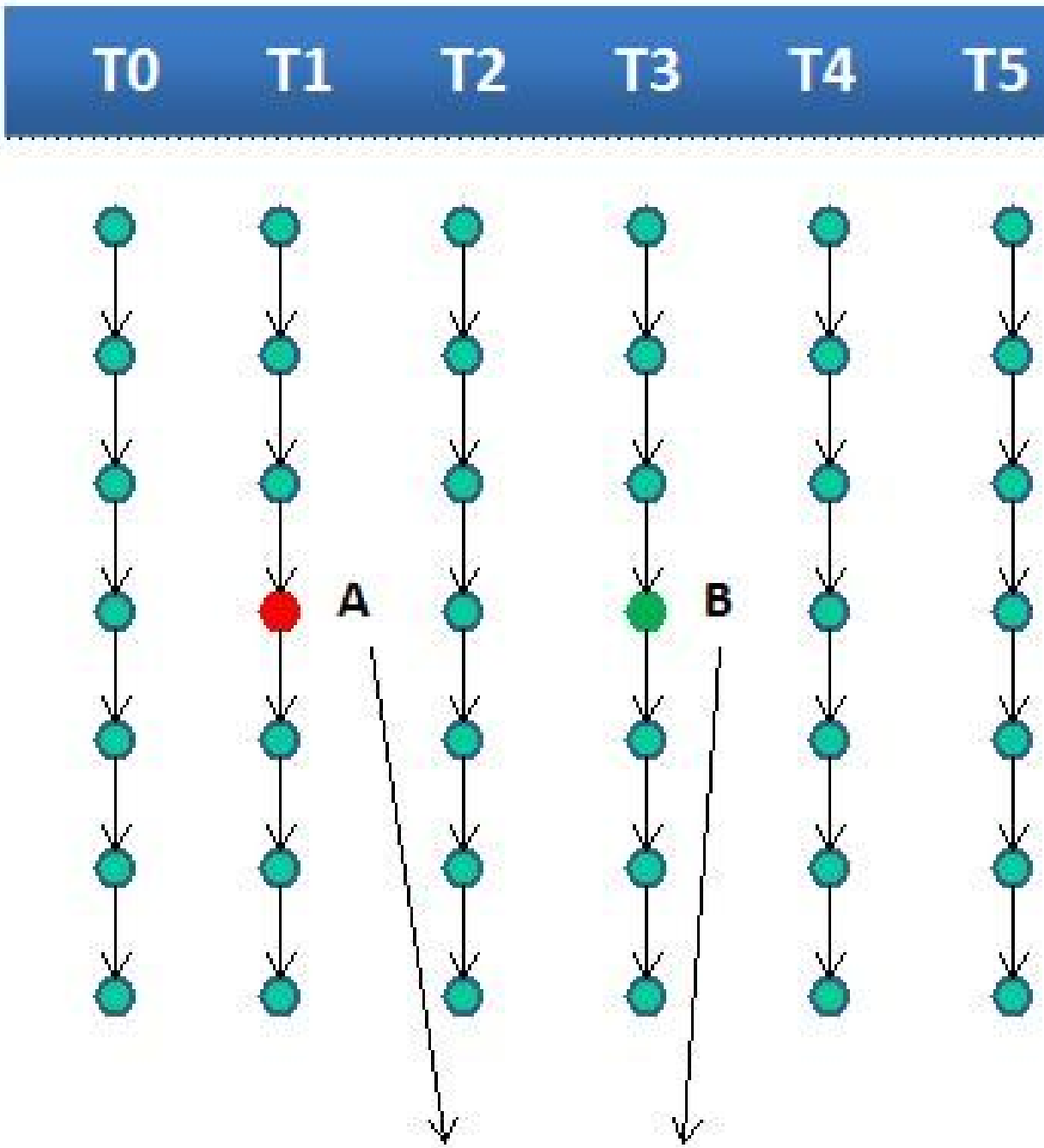
Mismatched Parameters

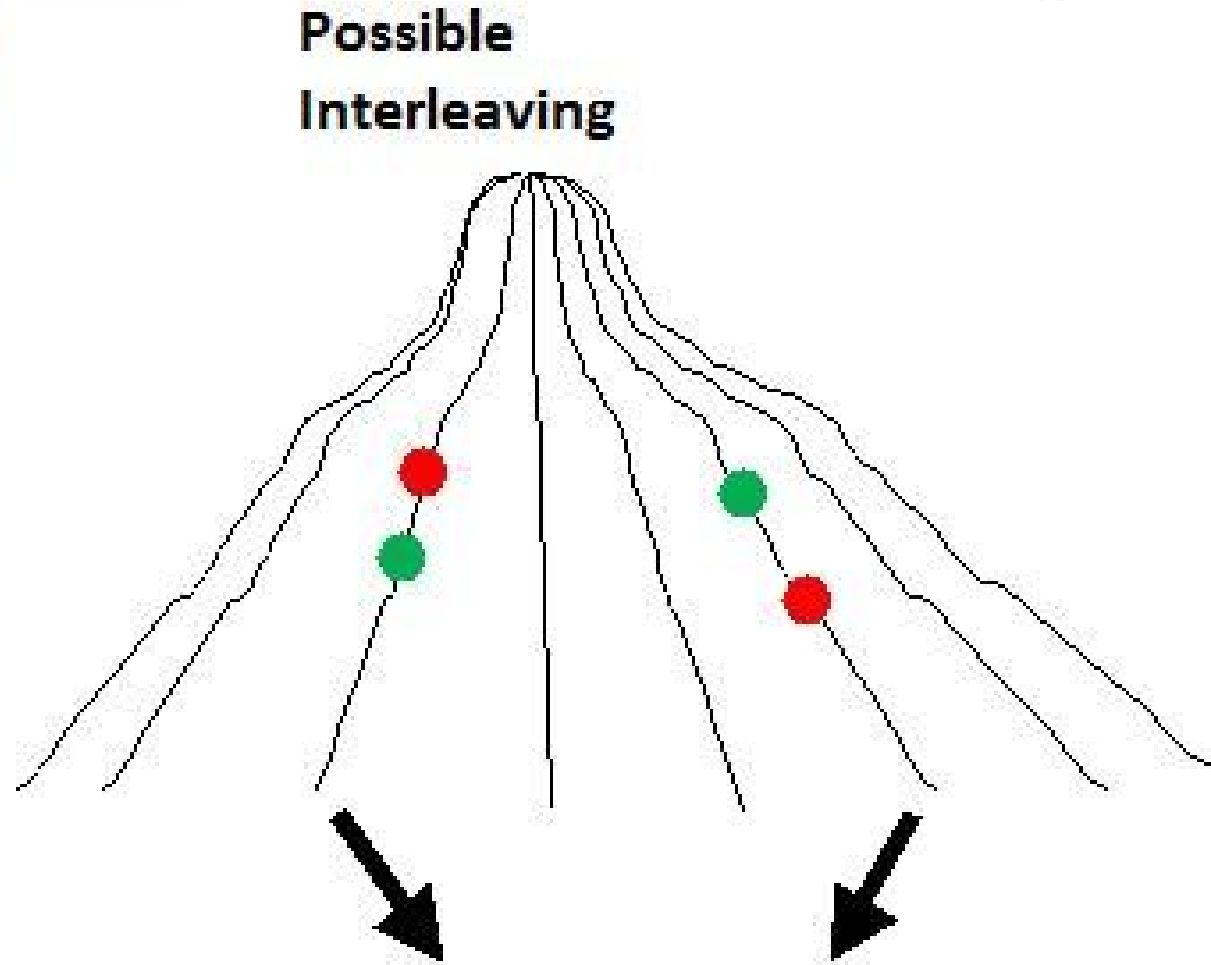
T0	T1
Create_Ep (0,1)	
R(<0,1>)	Get_Ep(<0,2>)
	S(1,<0,2>)

Does not exist!!

"get_endpoint" requested for a non-existent endpoint -- **ERROR**

Challenge: Exponential Interleavings

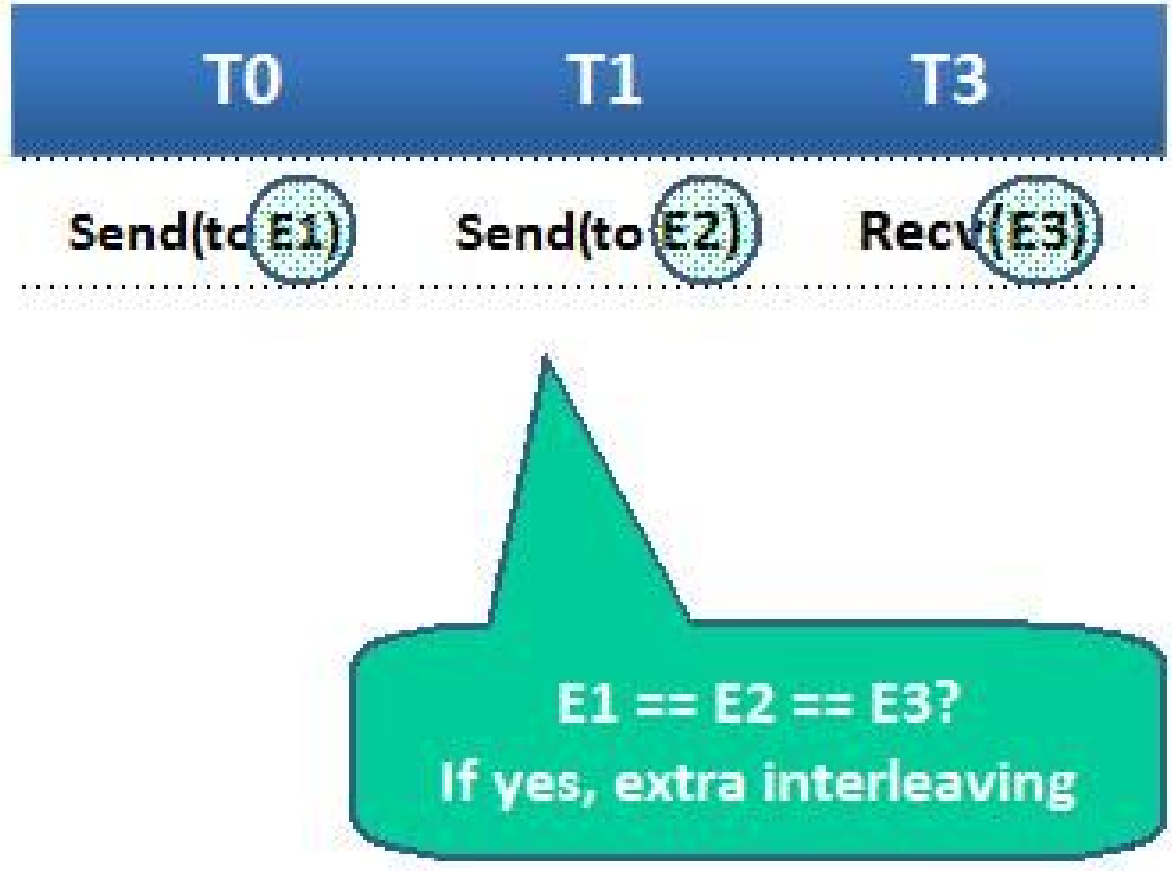




Only these 2 are RELEVANT!!!

Dynamic Interleaving Reduction

- Dynamic reduction
  - Transition dependency at runtime
  - precise information → effective reduction



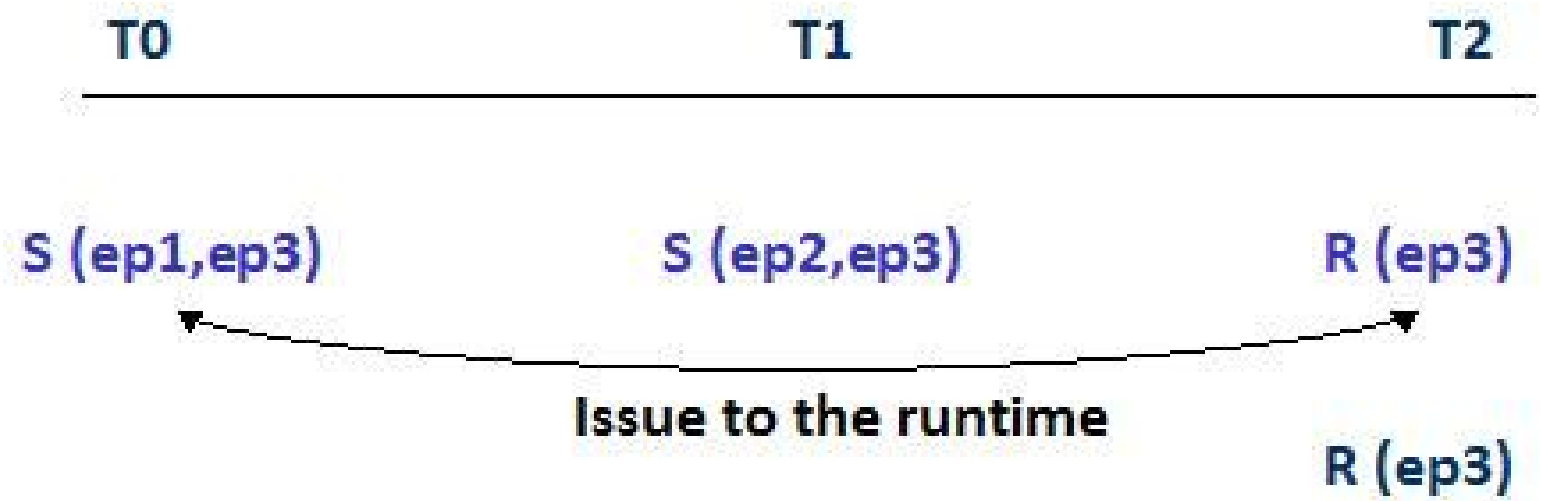
DYNAMICALLY:

- Discover all potential senders
- Match Recv with each sender
- Recurse through all such configurations

Relevant interleaving exploration by MCC scheduler

Relevant interleaving exploration by MCC scheduler

Interleaving 1



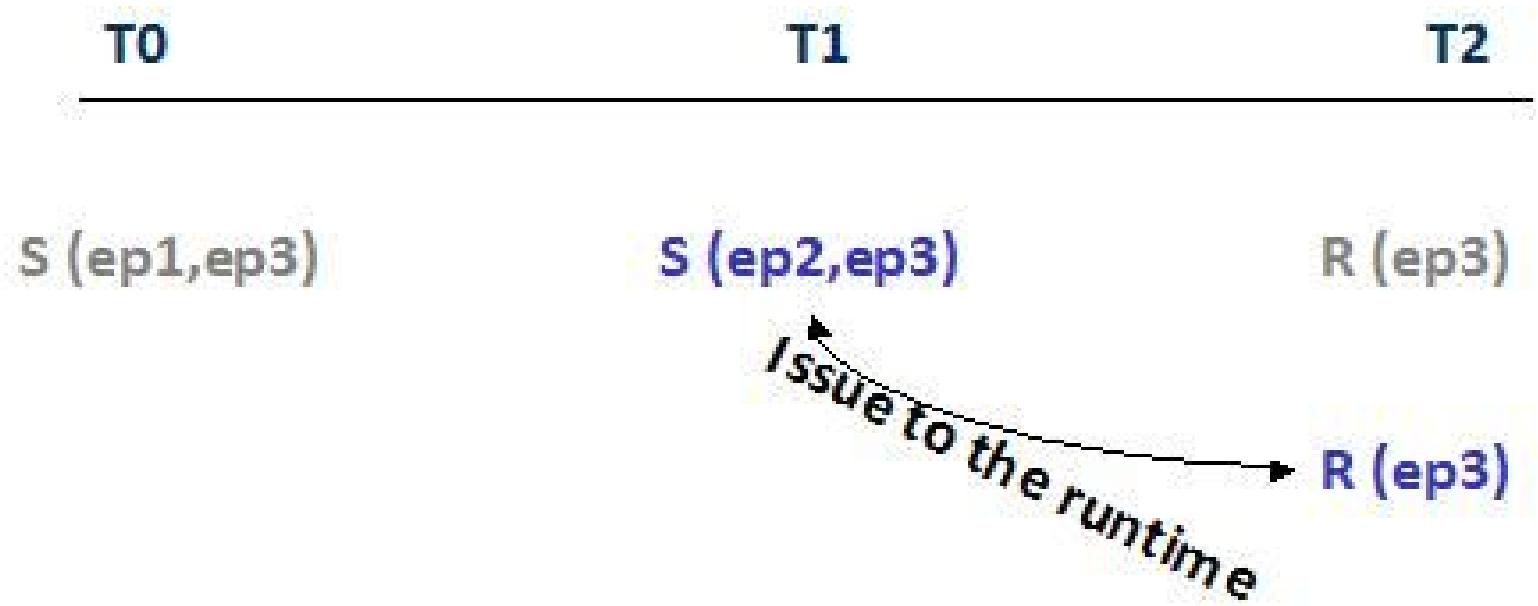
Match sets are computed at "decision points"

Set of matching transitions

All threads are blocked

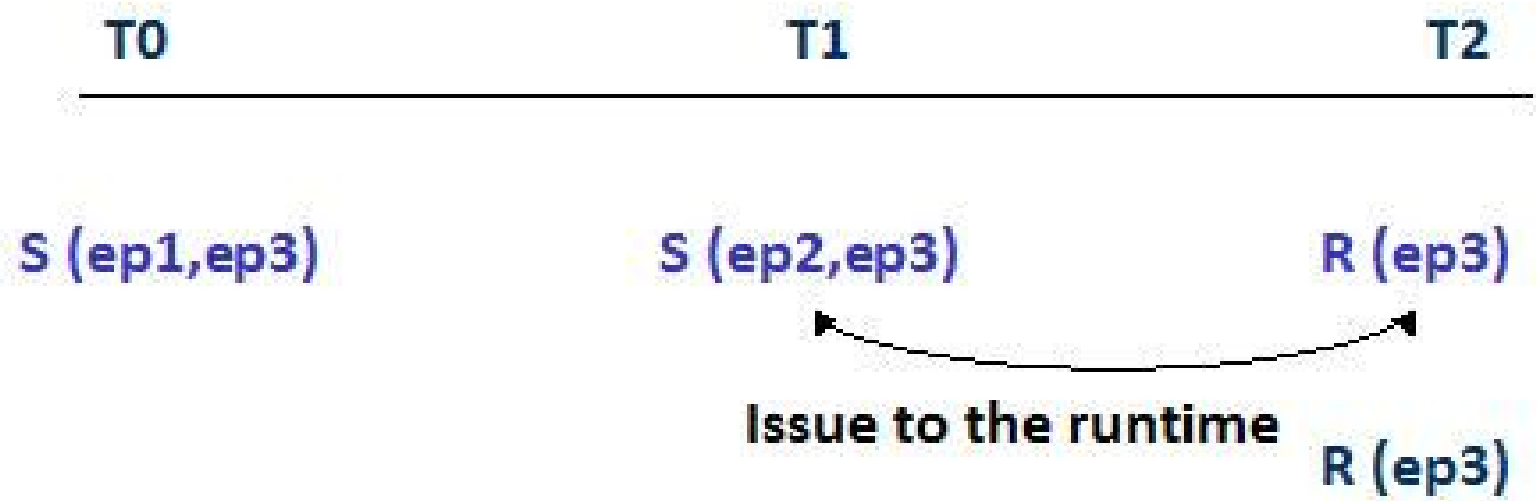
Relevant interleaving exploration by MCC scheduler

Interleaving 1



Relevant interleaving exploration by MCC scheduler

Interleaving 2

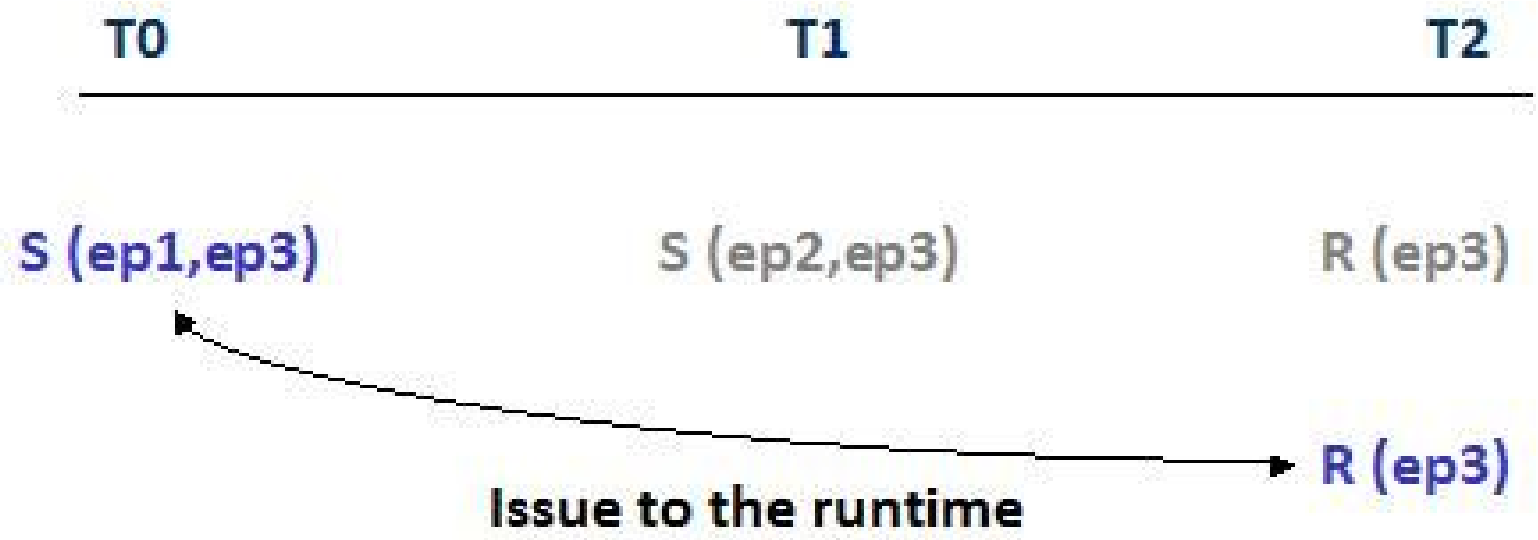


Enabled Transitions: S(ep1, ep3), S(ep2, ep3), R (ep3)

Match Set: **<S(ep2, ep3), R (ep3)>**

Relevant interleaving exploration by MCC scheduler

Interleaving 2

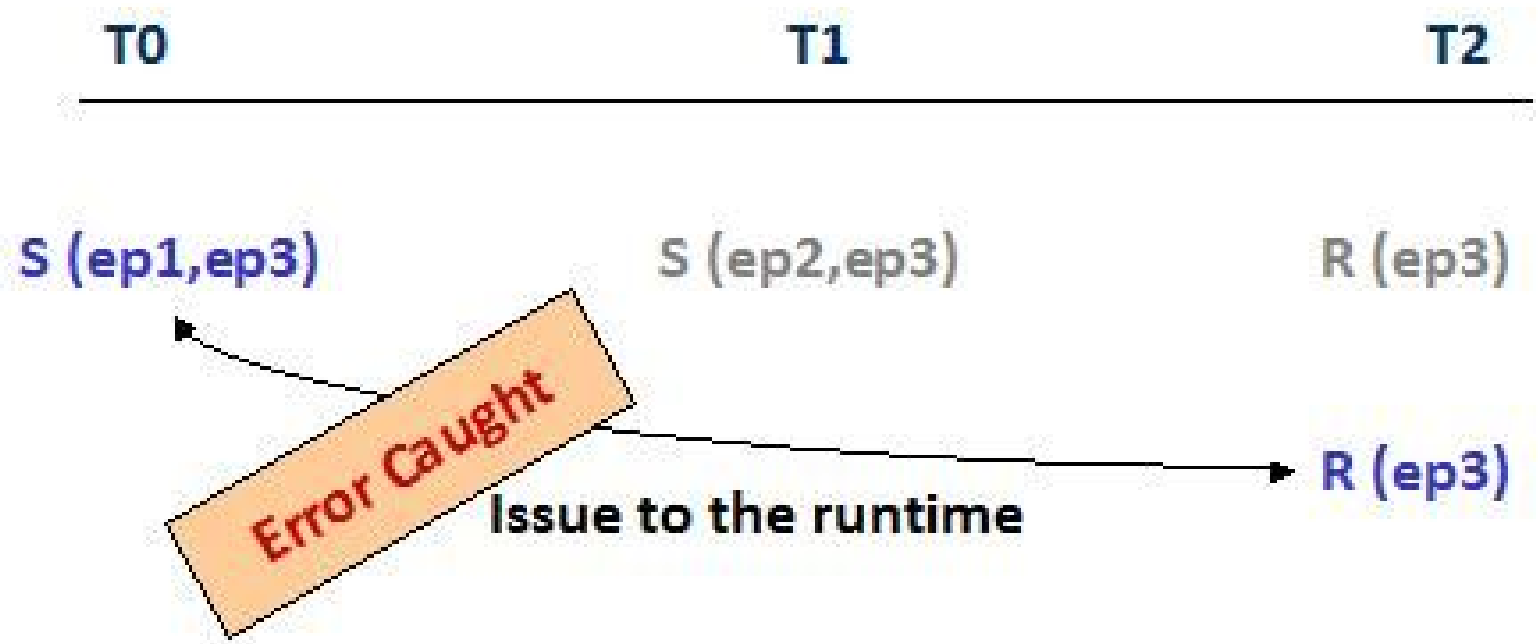


Enabled Transitions: S(ep2, ep3), R (ep3)

Match Set: <S(ep1, ep3), R (ep3)>

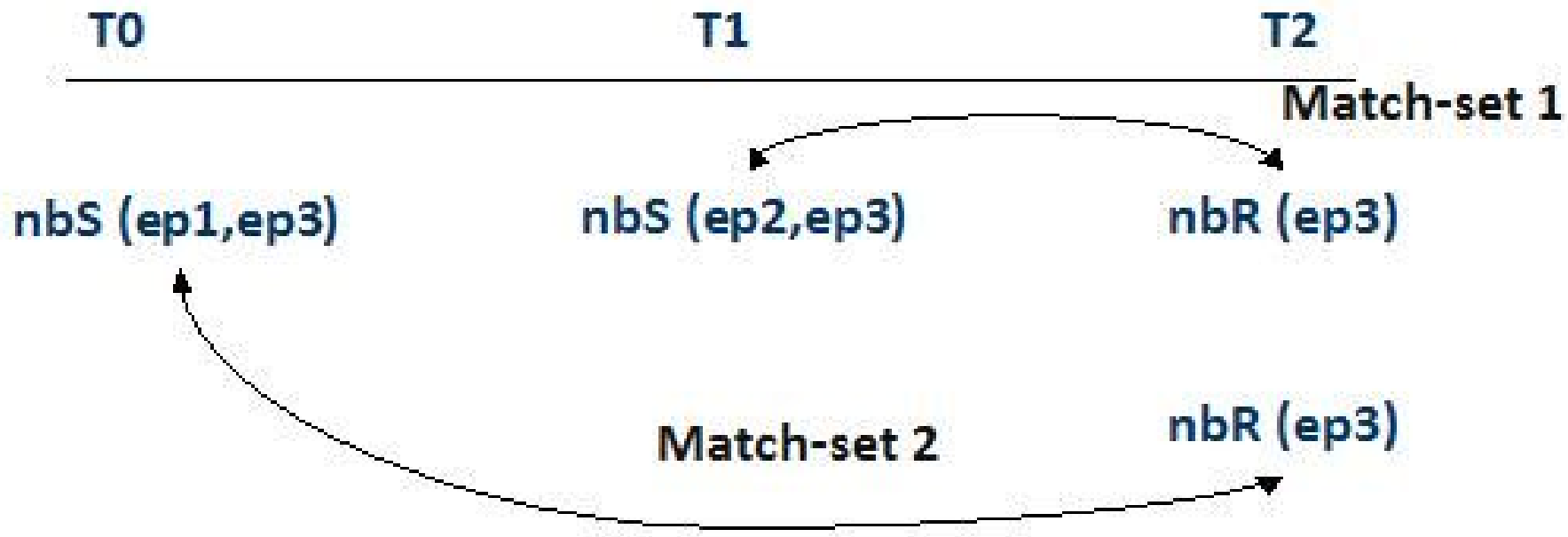
Relevant interleaving exploration by MCC scheduler

Interleaving 2



Enforcing a deterministic runtime match

- For non blocking requests, there is an additional problem
  - Runtime communication race even after a scheduler decides deterministically.



Enforcing a deterministic runtime match

Enforcing a deterministic runtime match

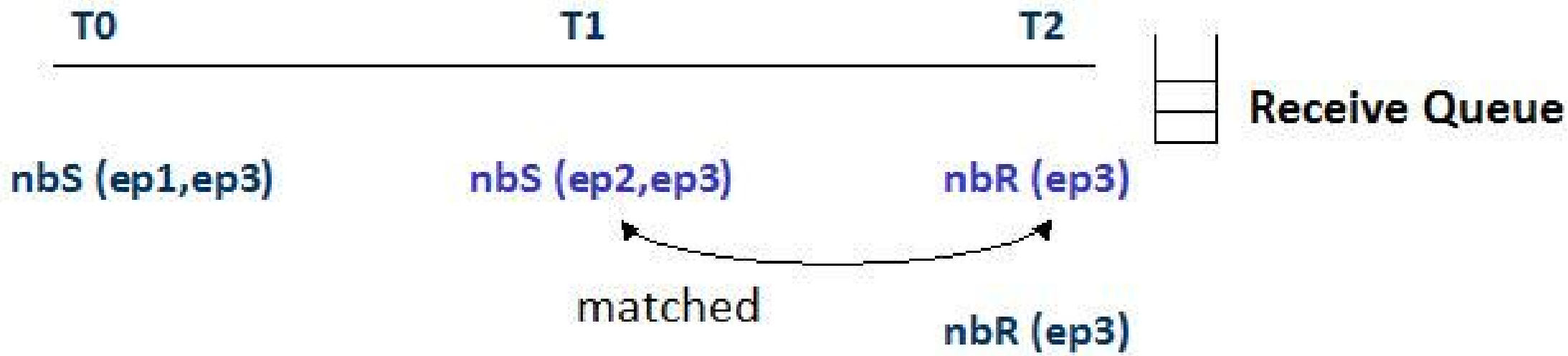
Solution 1 to enforce a deterministic match.

- Addition of Probing function to the MCAPI library
  - Probes an endpoint's receive queue for a message
  - Returns TRUE if the message is found

- Scheduler issues the next match-set only when the probe function returns TRUE.

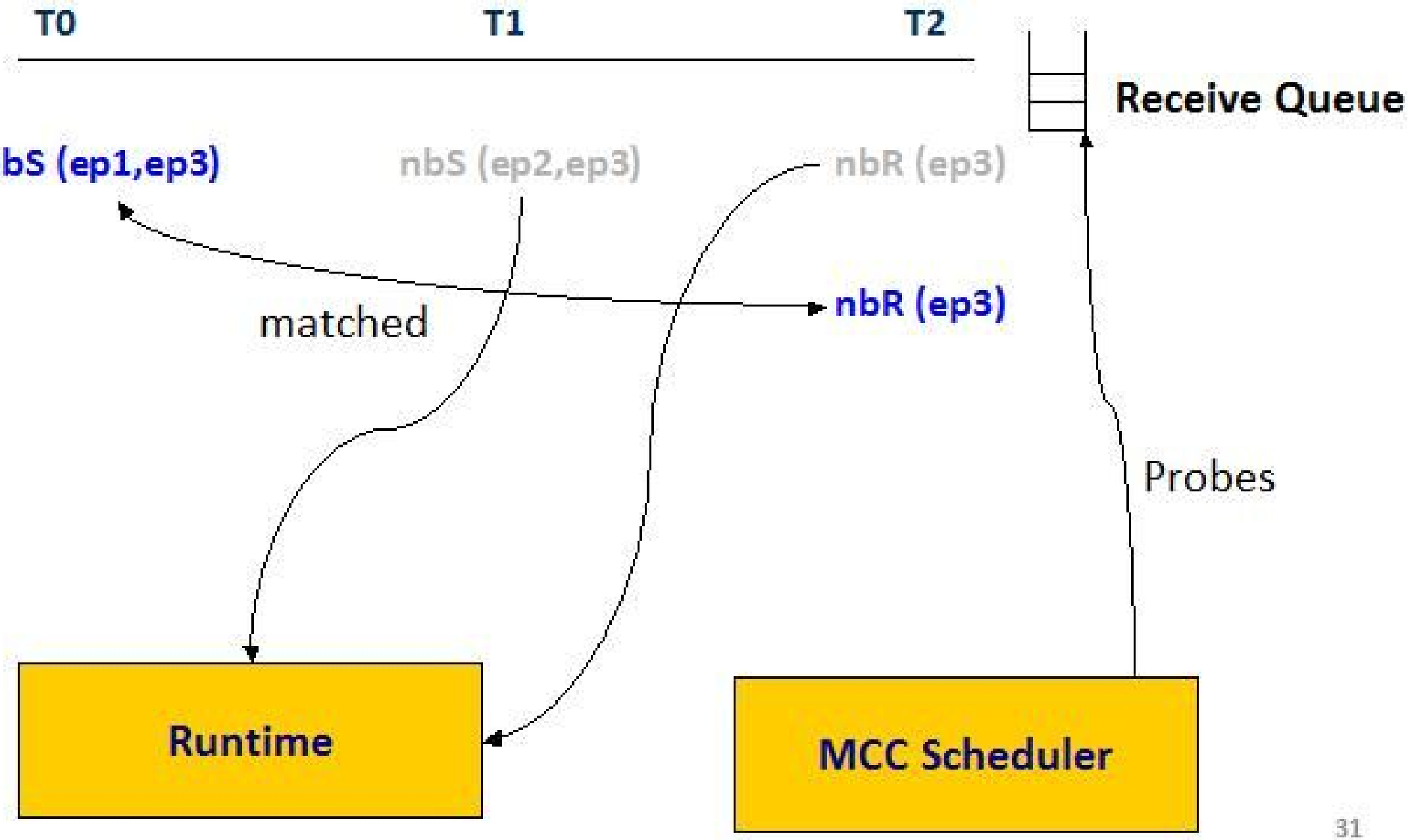
Enforcing a deterministic runtime match

Probing Solution



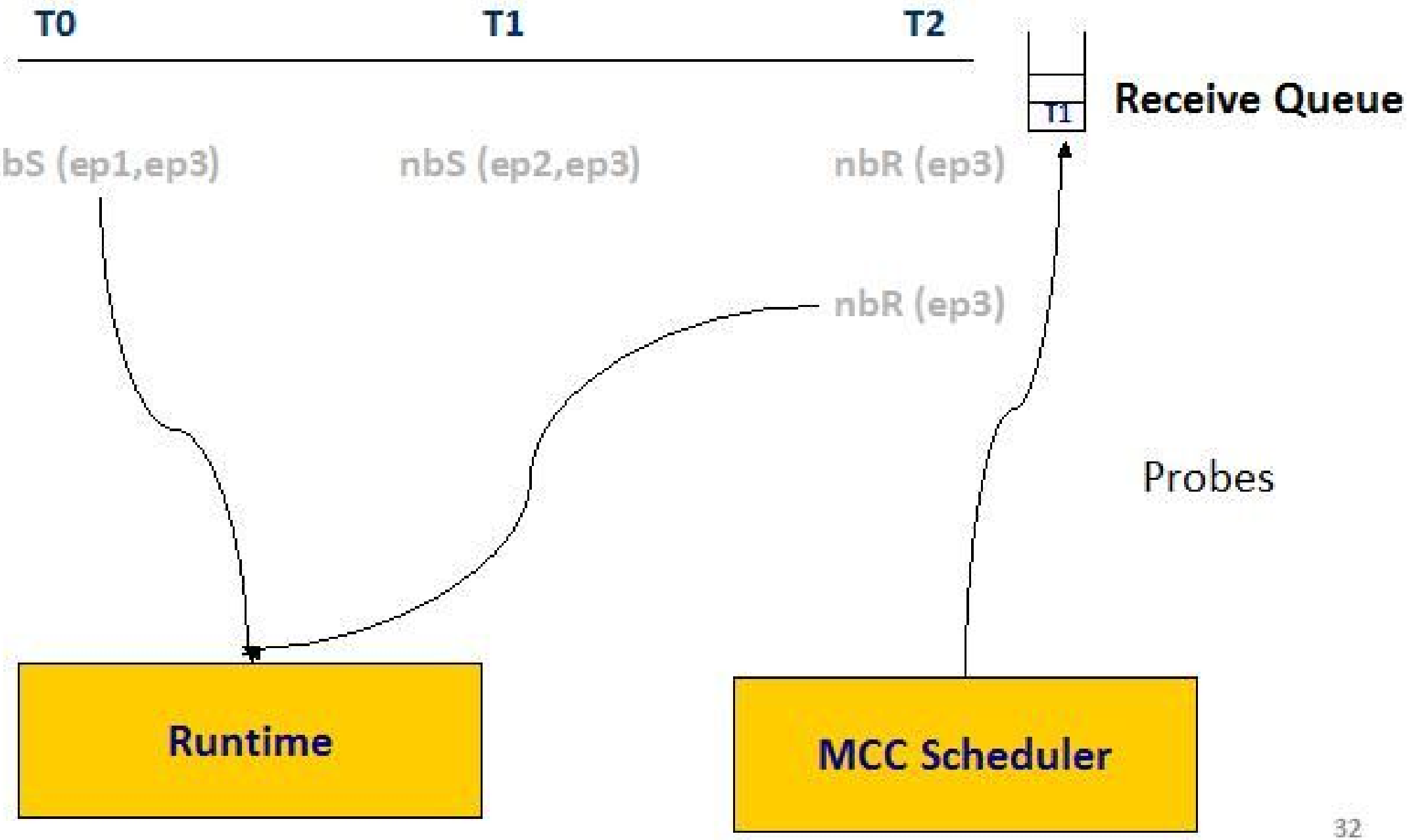
Enforcing a deterministic runtime match

Probing Solution



Enforcing a deterministic runtime match

Probing Solution



Enforcing a deterministic runtime match

Probing Solution

Enforcing a deterministic runtime match

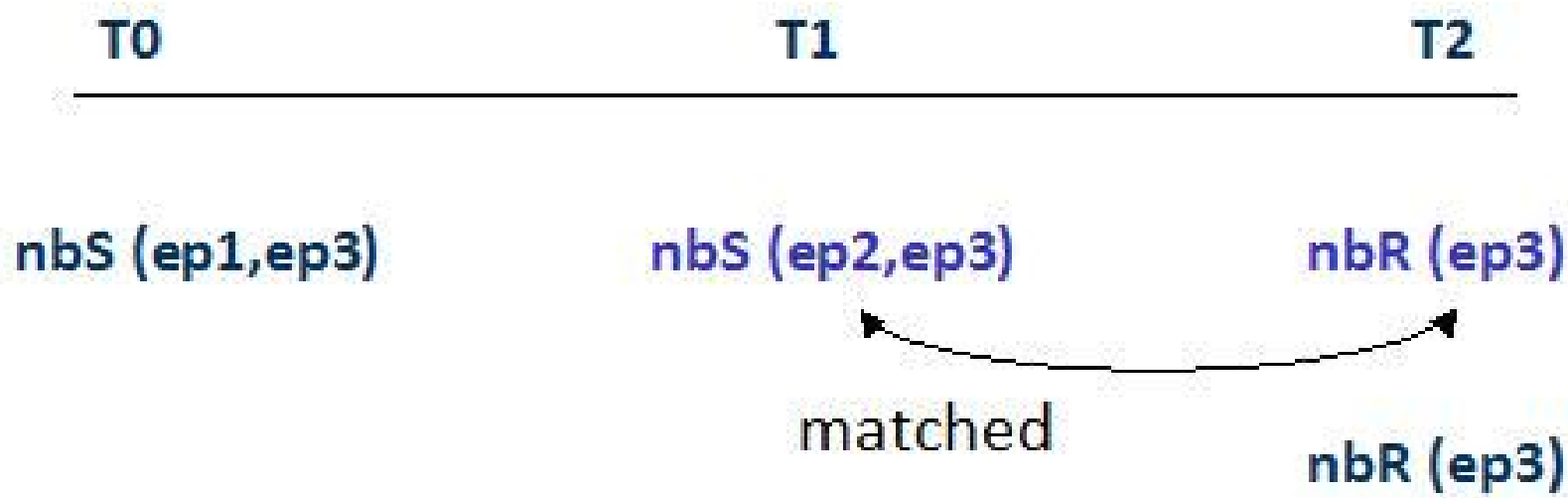
Solution 2 to enforce a deterministic match.

- Scheduler induces a “test” call
- Scheduler spin-loops on the request handle until the successful completion of the “test” call.

We opt Solution 2 in our work as it is non-intrusive.

Enforcing a deterministic runtime match

Dummy "wait" Solution

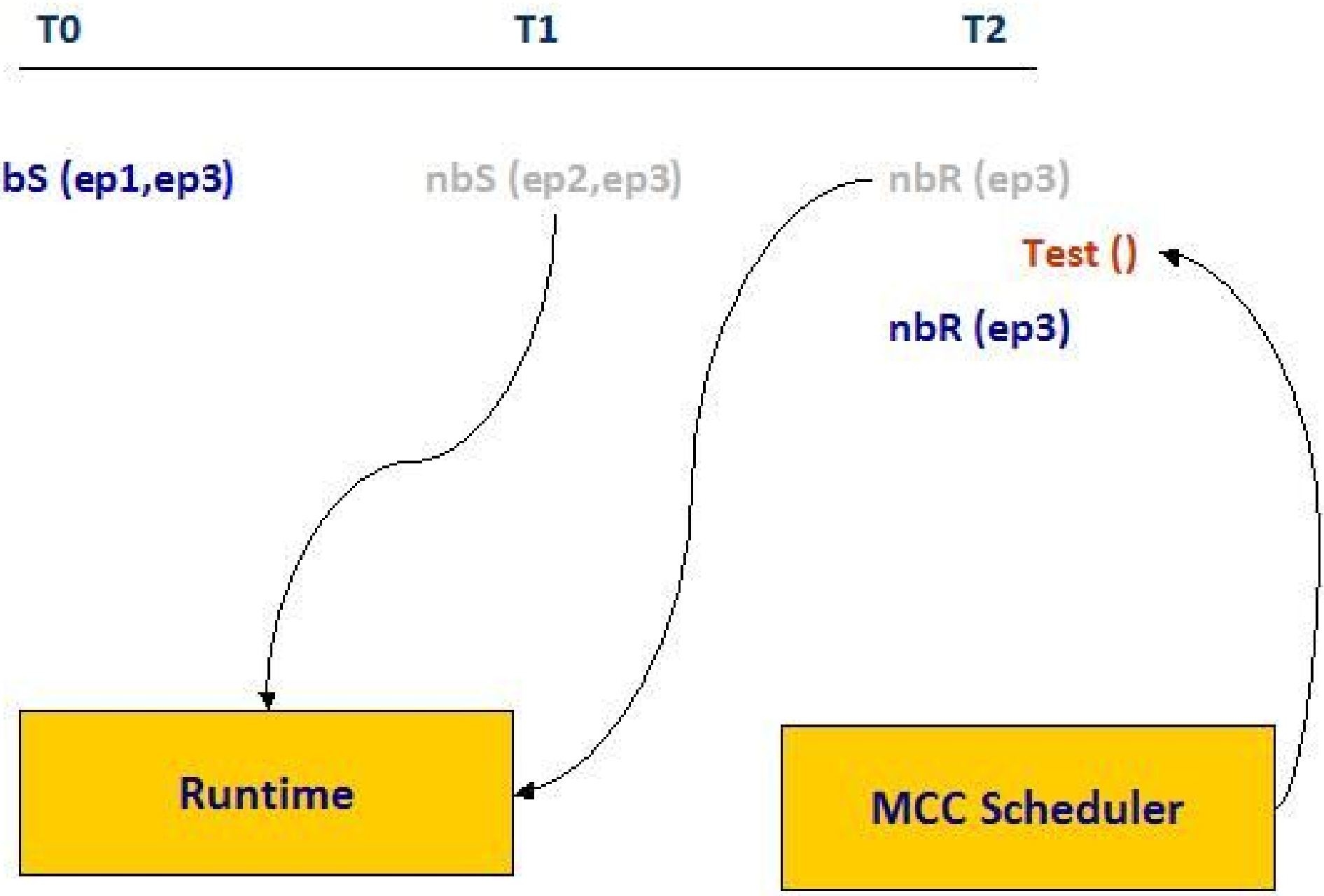


Enabled Transitions: Send(ep1, ep3), Send(ep2, ep3), Recv (ep3)

Match Set: **<Send(ep2, ep3), Recv (ep3)>**
<Send(ep1, ep3), Recv (ep3)>

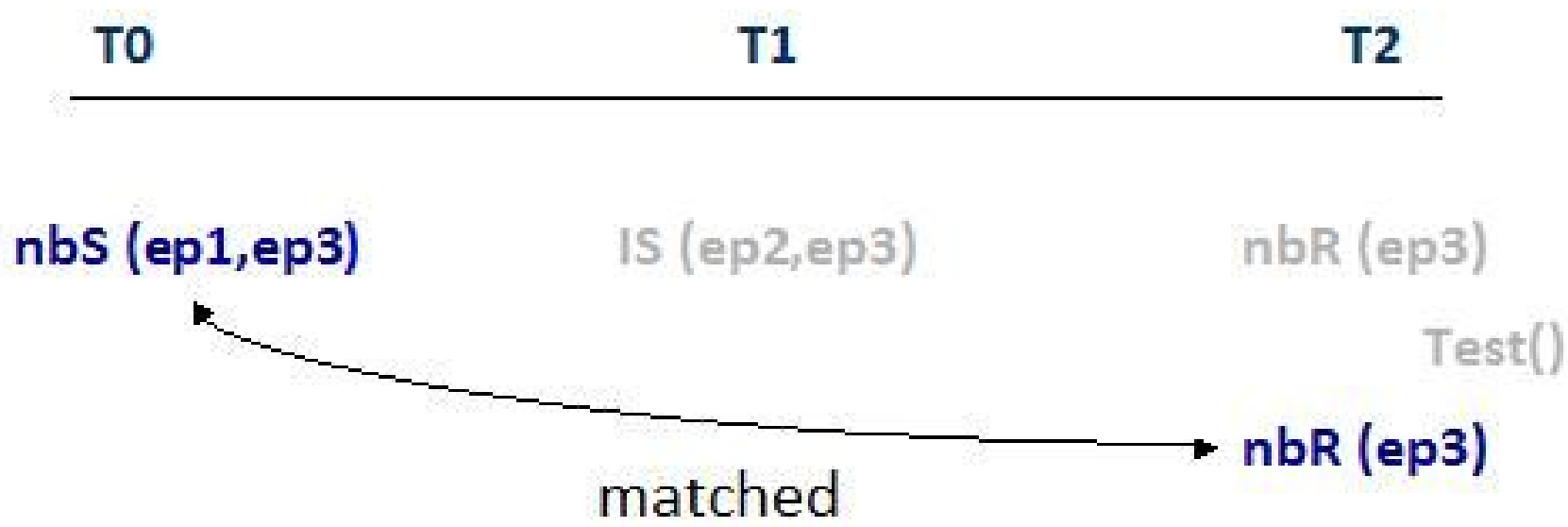
Enforcing a deterministic runtime match

Dummy "wait" Solution



Enforcing a deterministic runtime match

Dummy "wait" Solution



Runtime

MCC Scheduler

Concluding remarks and future work

- Code is currently being developed
  - MCC currently supports blocking and non-blocking connectionless API constructs
  - Checks for safety assertion violations and Deadlocks
  - Porting concurrency benchmarks into MCAPI – Eg. Rodinia Benchmarks, BSS use case
  - MCC will be tested on these benchmarks.

- Steadily improve MCC
  - Support for connection-oriented API calls, sanity checks etc.
  - Accommodate "non-determinism" in the shared memory space.

Thank You!

www.cs.utah.edu/formal_verification