WESTERN MICHIGAN UNIVERSITY

Deterministic Replay for MCAPI Programs

Mohamed Elwakil and **Zijiang James Yang**

Department of Computer Science

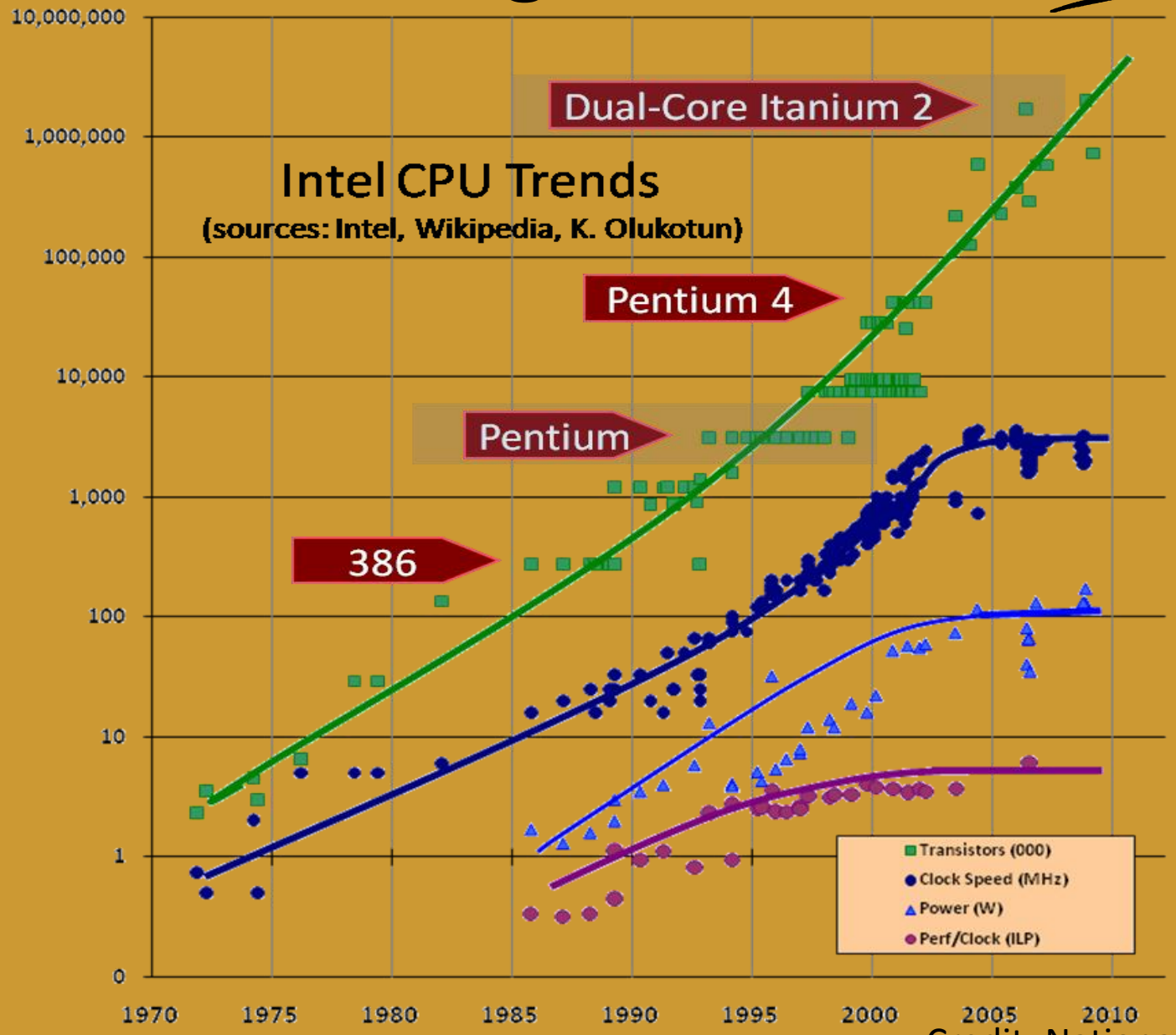
Western Michigan University

PADTAD'11

Toronto, ON, Canada

July 17, 2011

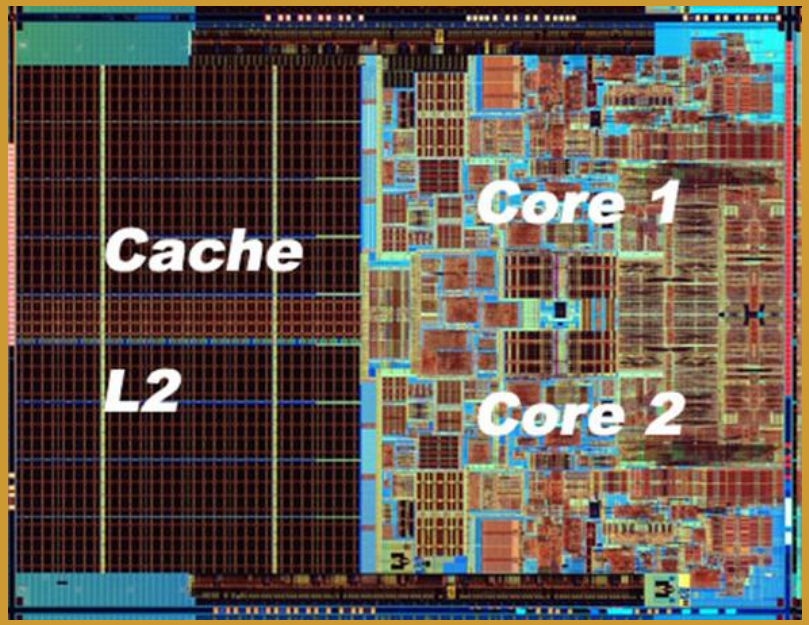
The Megahertz Race



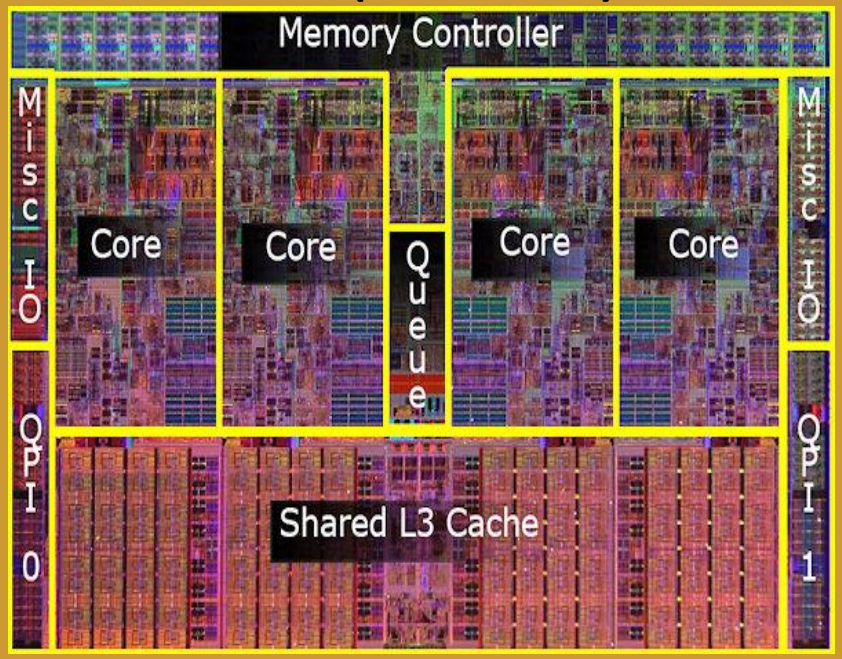
Credit: National Instruments

Multicore Processors

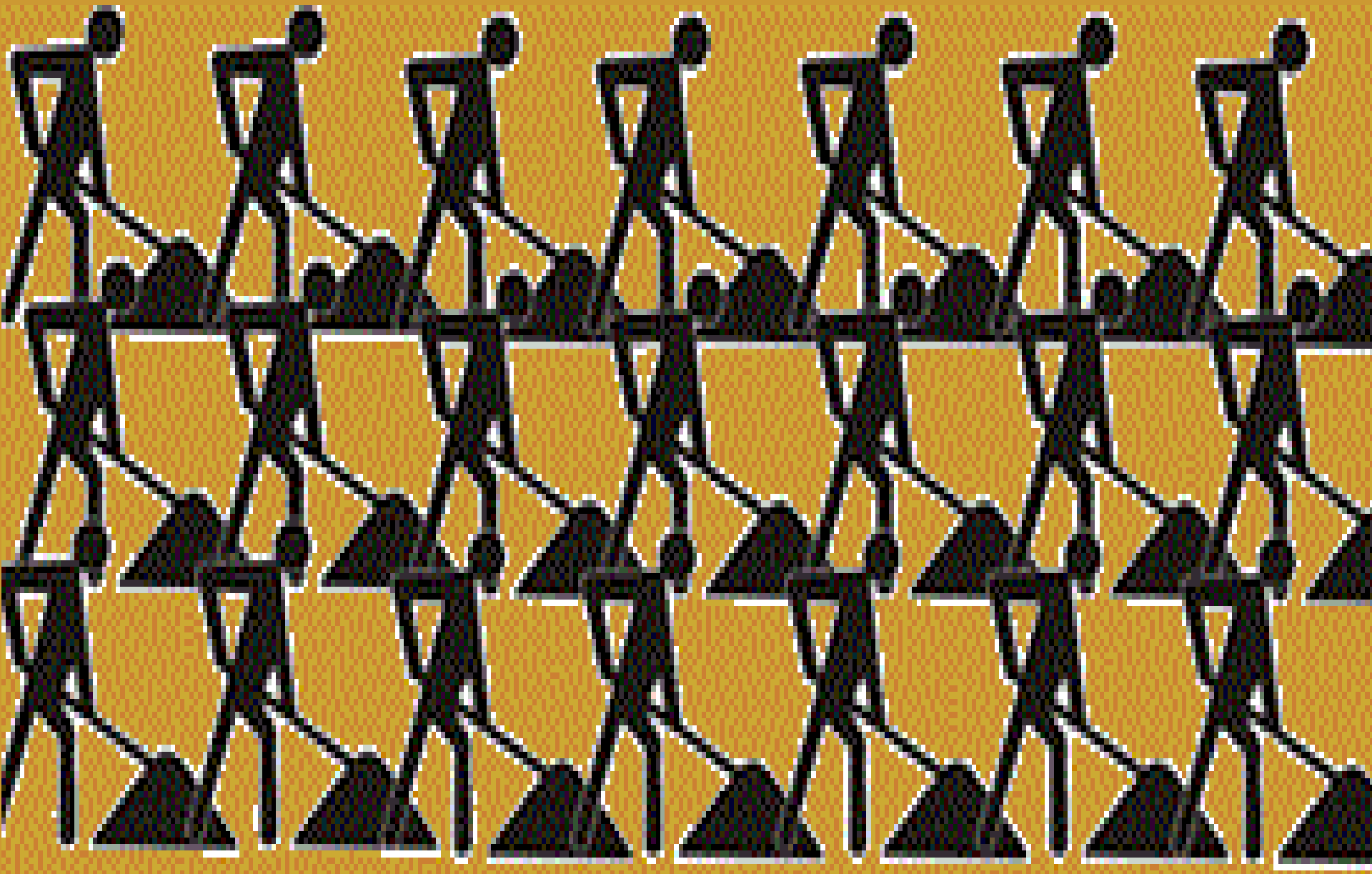
Intel Core 2 Duo (Dual Core) - 2006



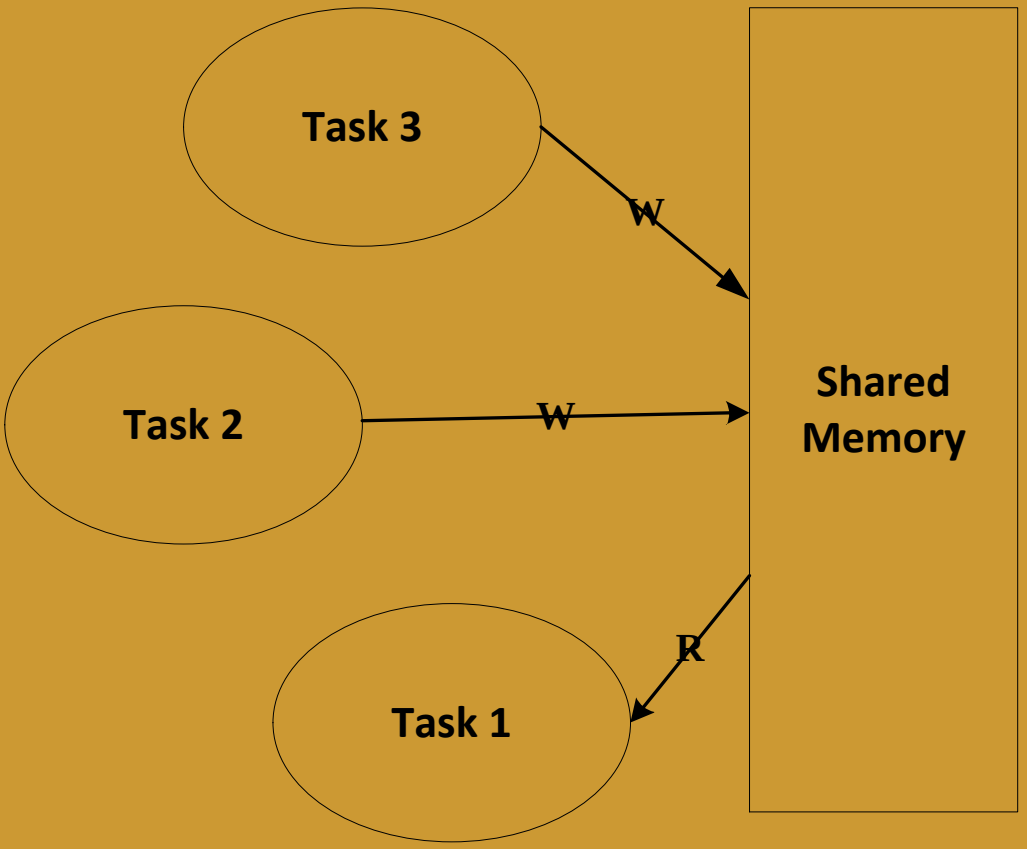
Intel Core i7 (Quad Core) - 2008

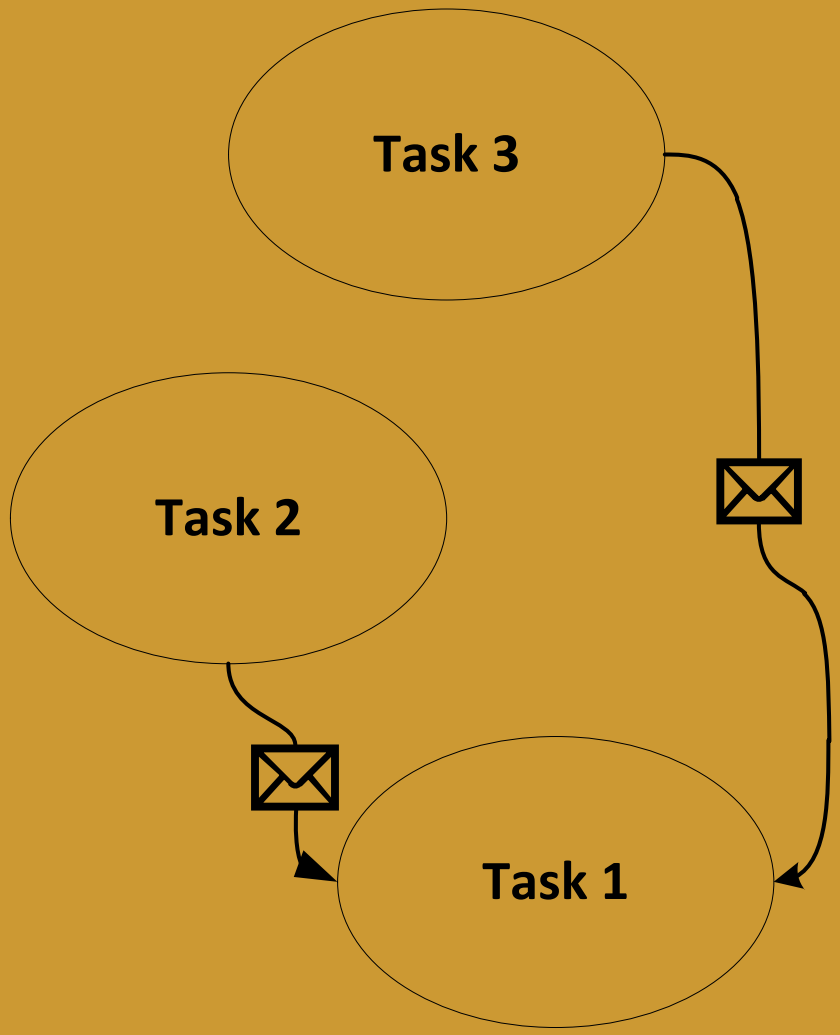


Concurrent Programs



Shared-memory and Message-passing





Shared-memory vs. Message-passing

- Message-passing:
  - "*that other model for concurrent programming*"
  - When shared-memory is not possible.

Shared-memory vs. Message-passing

- Message-passing:
  - "*that other model for concurrent programming*"
  - When shared-memory is not possible.
- A Case Study of Shared-memory and Message-passing [MIT 1995]:
  - Message-passing is **52%** faster when processes>4
  - When using fast communication channel.
  - Locks and barriers slows down shared-memory.

Shared-memory vs. Message-passing

- Message-passing:
  - "*that other model for concurrent programming*"
  - When shared-memory is not possible.
- A Case Study of Shared-memory and Message-passing [MIT 1995]:
  - Message-passing is **52%** faster when processes>4
  - When using fast communication channel.
  - Locks and barriers slows down shared-memory.
- The case for message passing on many-core chips [Intel & UIUC 2010]:
  - "*Message-passing is more suitable for the multicore era*".
  - Coherence Wall

New API's

- MCA MCAPI, 2008:
  - MCA: Intel, AMD, Samsung, Siemens, NI, TI, CMU, UoH.
  - General purpose message-passing API
  - Lightweight, assumes fast and reliable interconnects.
  - Two implementations: MCA's, Mentor Graphics

- Intel RCCE:
  - Message-passing API for Intel SCC (48 cores).

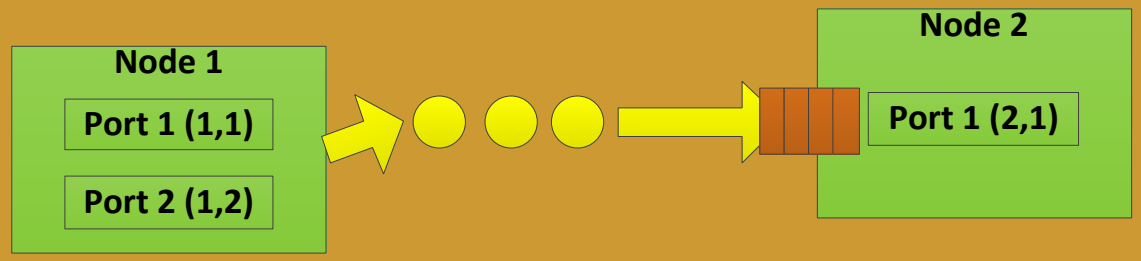
- API's and language-extensions are necessary, but not sufficient.

Debugging Message-passing Programs

- Challenging
  - Due to inherent non-determinism
- Neglected!
  - Most work on shared-memory.
- We can help!
  - Techniques and tools for supporting message-passing programs debugging.
- MCAPI

MCAPI

- MCAPI Node
  - Process, thread...
- MCAPI Endpoint
  - Communication termination point.
  - A tuple of <node id, port id>
  - Has a FIFO buffer



- API:
  - Node initialization and finalization.
  - Endpoint creation and deletion.
  - Sending and receiving messages

MCAPI API

Send (Data, Src_EP, Dest_EP)	Sends data from a source endpoint to a destination endpoint.
Recv (&Data, Recv_EP)	Retrieves a message from an endpoint buffer.
Send_i (Data,Src_EP, Dest_EP,&Request) Recv_i (&Data, Recv_EP,&Request)	Non-blocking calls.
Wait (Request) Test(Request) Waitany(Requests)	Checks the completion of the request.

MCAPI sources of non-determinism

- Promiscuous $Recv$
- Non-blocking $Test$
- Blocking $Waitany$

N0:
Send(Data,N0_EP1,N2_EP1);

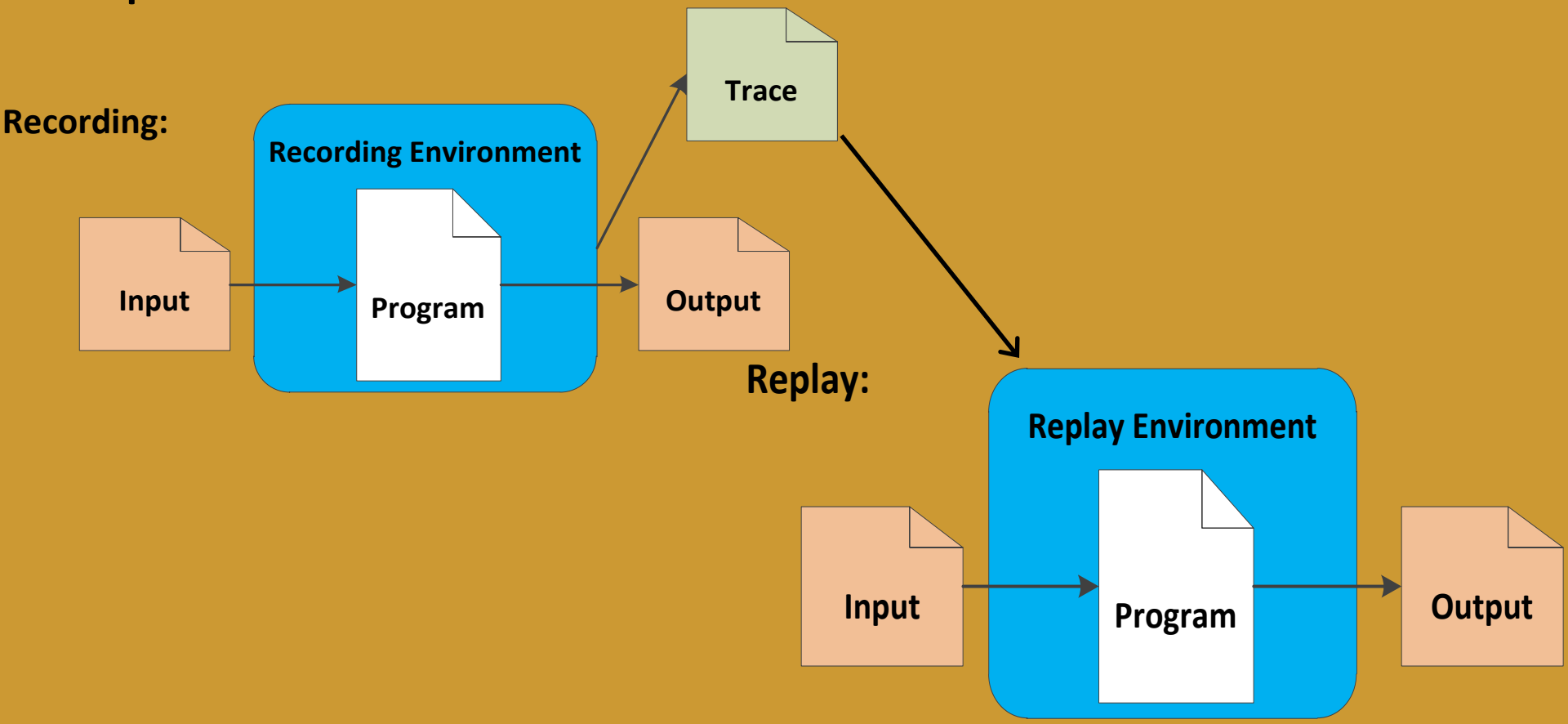
N1:
Send(Data,N1_EP1,N2_EP2);
while (!**Test**(Request))
{A++;}

N2:
Recv(&Data1,N2_EP1,&Reqs[0]);
Recv(&Data2,N2_EP2,&Reqs[1]);
Index=**Waitany**(Reqs);
Func(Index);

?

Reproducing Bugs-Deterministic Replay

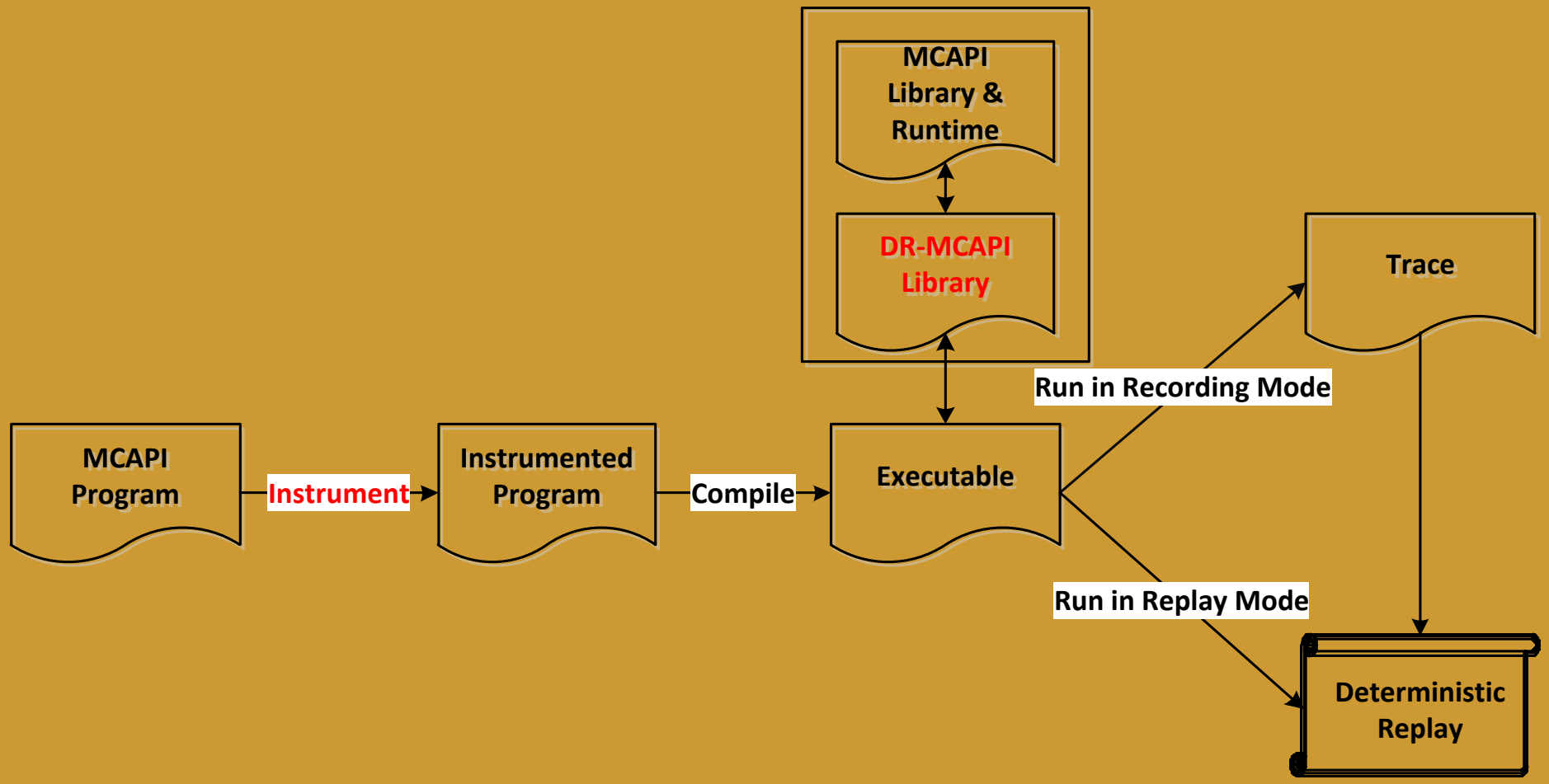
- A controlled execution that is logically equivalent to a previous execution of interest.



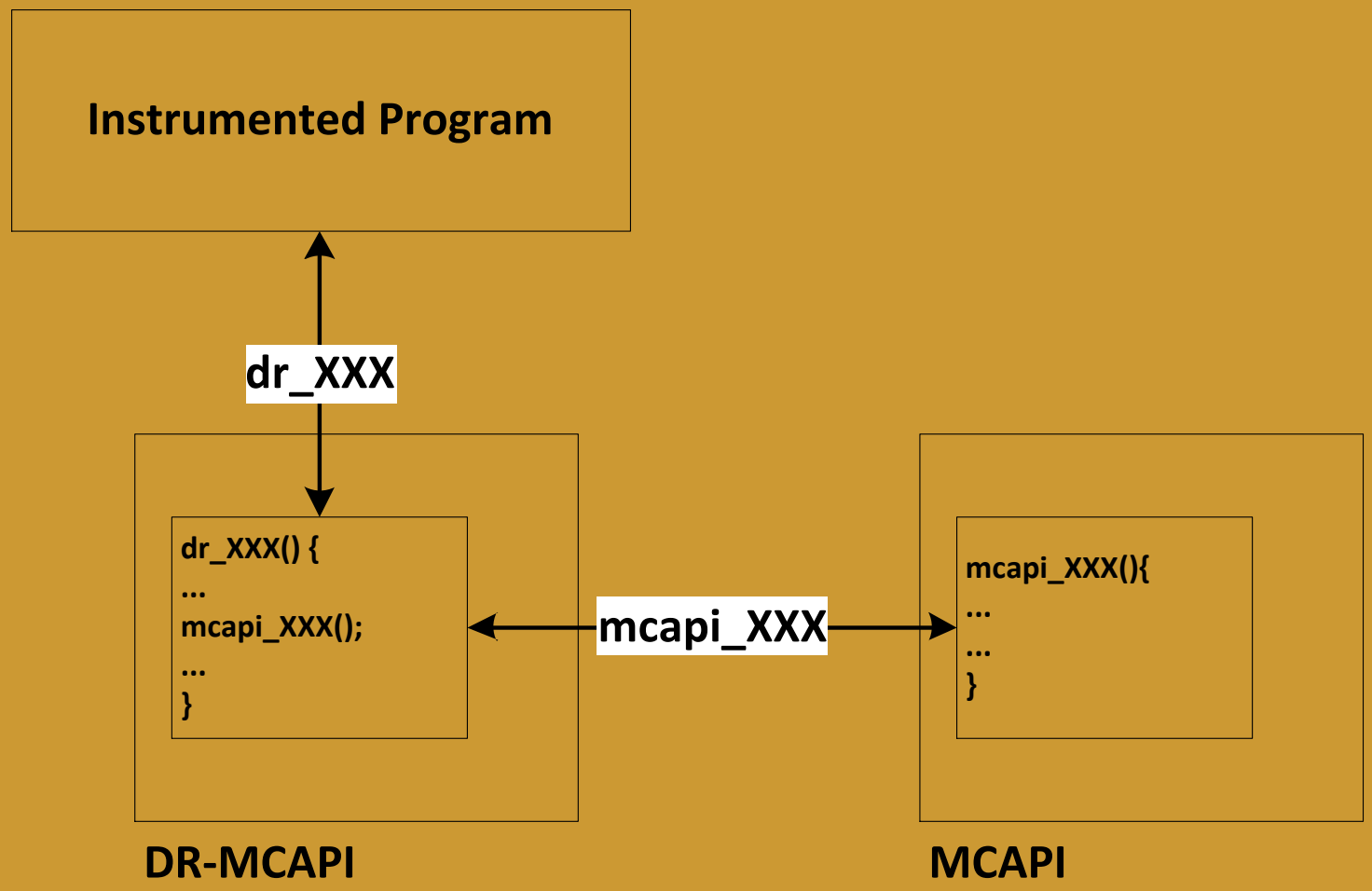
Replay Approaches

- Data Replay
  - Record message contents
  - Simulate receiving a message
- Order Replay
  - Record non-deterministic operations outcome.
  - Enforce these outcomes during replay.

DR-MCAPI


MCAPI Library & Runtime
DR-MCAPI Library
Trace
Run in Recording Mode
MCAPI Program
Instrument
Instrumented Program
Compile
Executable
Run in Replay Mode
Deterministic Replay

During Runtime



Receiver-based Order Replay

- Recording:
  - When a function call F, retrieves message M:
    - H=Hash(M)
    - Msg(F)=H
- Replay:
  - When a function call F is invoked
    - Find H=Msg(F)
    - Only message M with Hash(M)=H is returned to F.

Recording in Receiver-based Order Replay

N1	N2	N3	N4
A=10;	X=Recv();	B=1;	U=Recv();
Send(A,N2); // M0	Y=Recv();	Send(B,N2);//M3	W=Recv();
Send(A,N4);// M1	Z=X-Y;	Send(B,N4);//M4	O=Recv();
	Send(Z,N4);//M2		Assert(U>0);

Trace

Node 2	Node 4
R:2/1/H0	R:4/1/H1
R:2/2/H3	R:4/2/H2
	R:4/3/H4

n Receiver-based Order F

Retrieved M0

Retrieved M1

N1	N2	N3	N4
A=10;	X=Recv();	B=1;	U=Recv();
Send(A,N2); // M0	Y=Recv();	Send(B,N2);//M3	W=Recv();
Send(A,N4);// M1	Z=X-Y	Send(B,N4);//M4	O=Recv(
	nd(Z,N4);//M2		Ass t( );

M3 is already in *RecievedMessages*

M4 is already in *RecievedMessages*

M2 is already in *RecievedMessages*

RecievedMessages

Node 2	Node 4

Trace

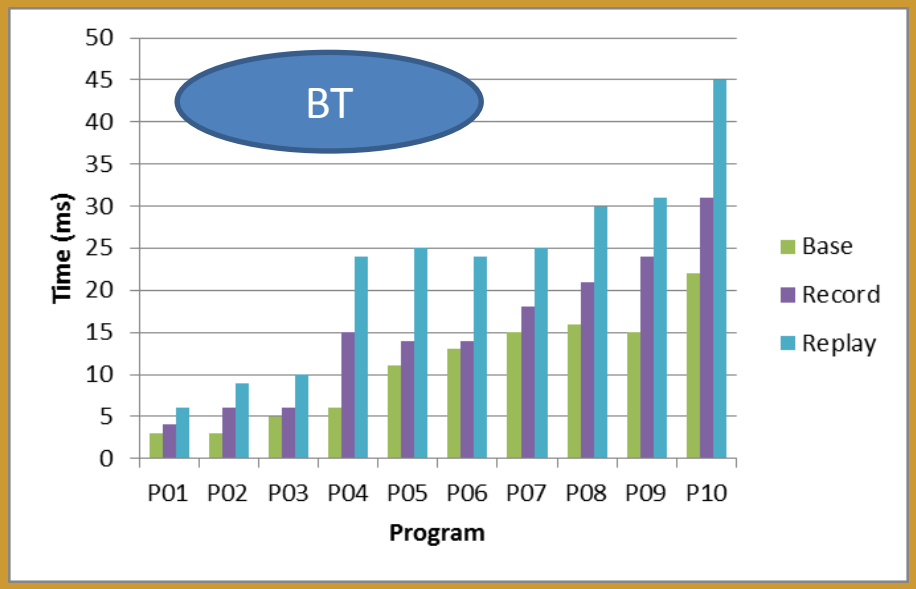
Node 2	Node 4
R:2/1/H0	R:4/1/H1
R:2/2/H3	R:4/2/H2
	R:4/3/H4

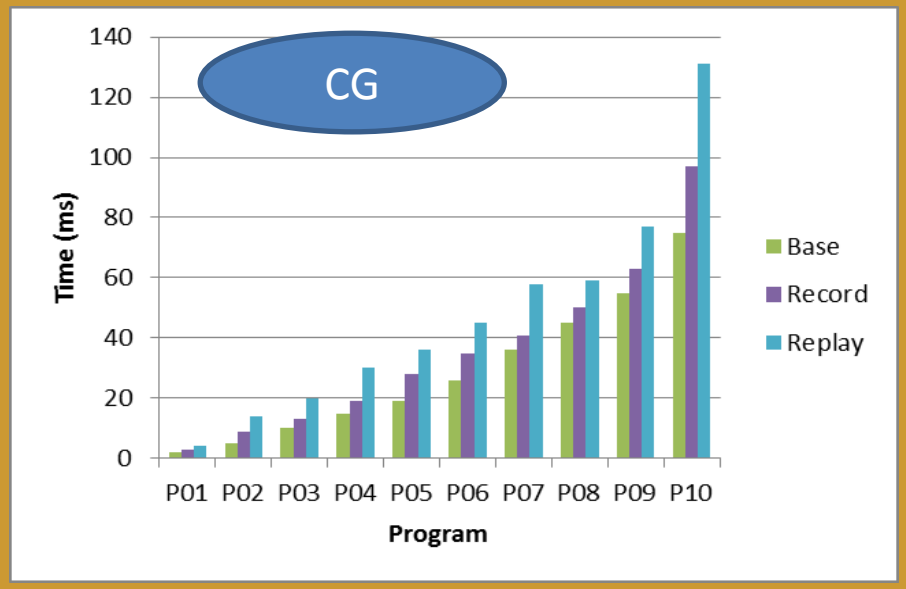
Other sources of non-determinism

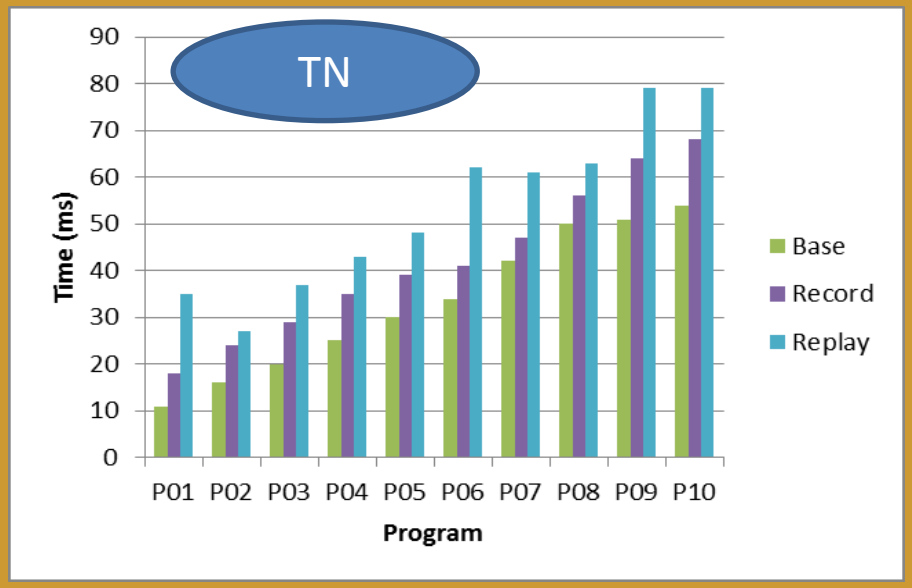
- Waitany (Request *Reqs)
  - Store the index returned during the recording phase.
  - Force the request associated with that index to complete and ignore other requests, during the replay phase.

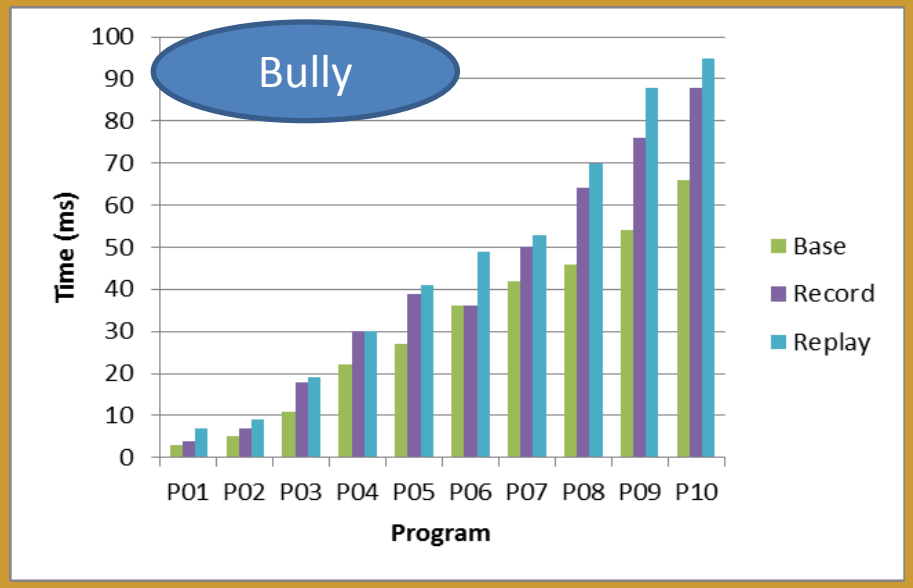
- Test(Request Req)
  - Let x be the number of failed invocations during the recording phase.
  - Automatically return false x then force the request to complete.

Runtime Overhead

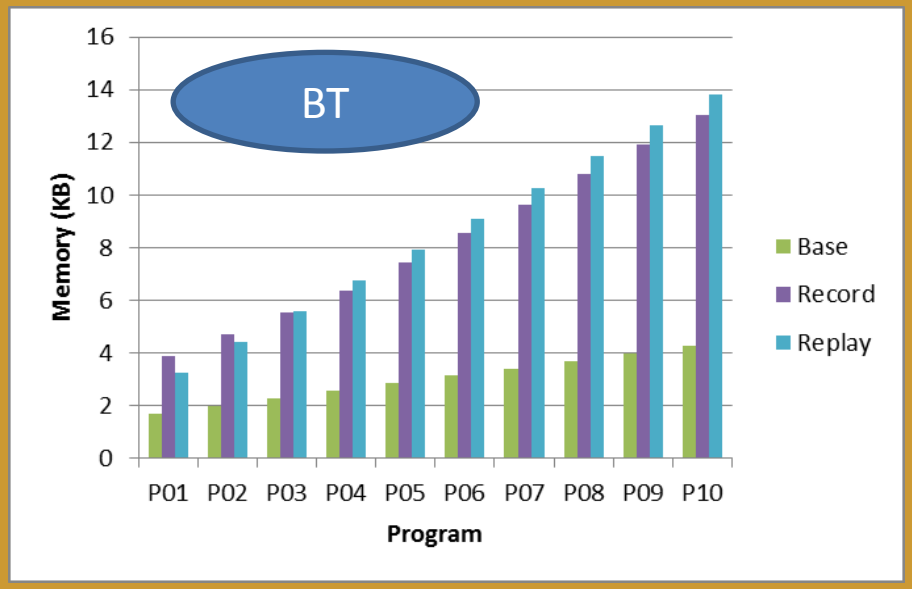


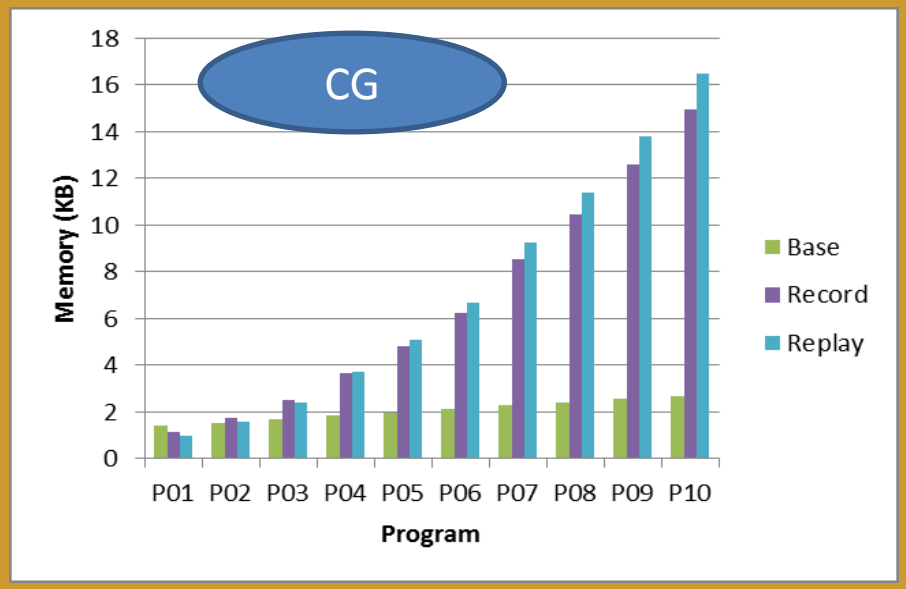


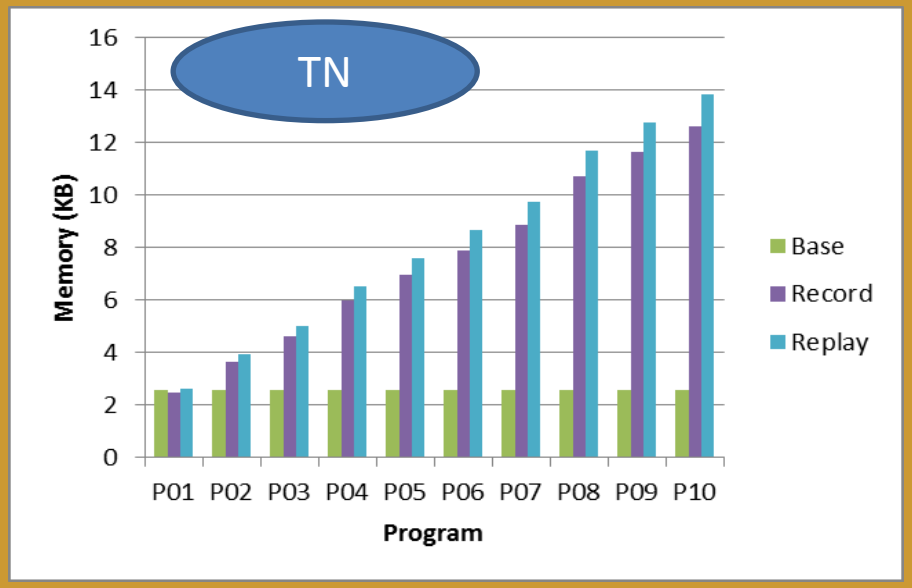


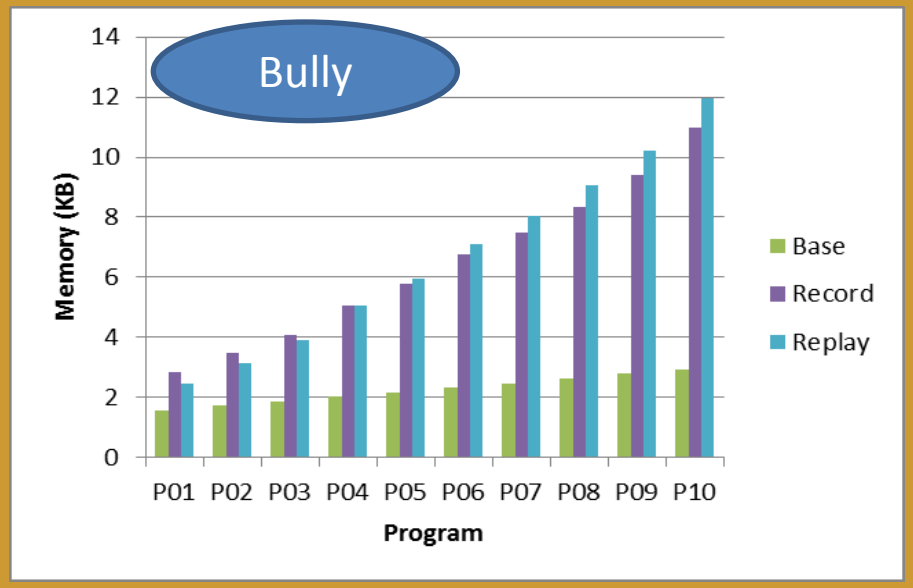


Memory Overhead









Related Work

- An Integrated Record & Replay Mechanism for Nondeterministic Message Passing Programs [2001]
  - 3x runtime overhead
- Parallel Program Debugging based on Data-Replay [2005].
- MPIWiz [2009]
  - Doesn't handle MPI_Test
  - MPI-specific

Conclusion

- Message-passing software development will be as important as shared-memory software
  - Debugging support will encourage programmers to make the switch
- Future Work:
  - Trace compression
  - Check-pointing

Acknowledgements

- This research is supported in part by:
  - NSF Grant CCF-0811287
  - ONR Grant N000140910740

Questions?