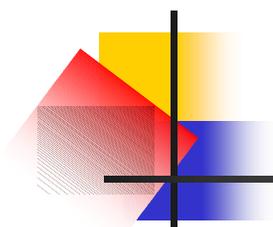


Compiler-Inserted Fault Tolerance for Message Passing Applications

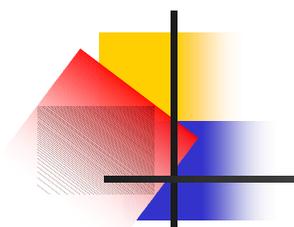
Keshav Pingali,
Dan Marques, Paul Stodghill, Greg Bronevetsky

Cornell University

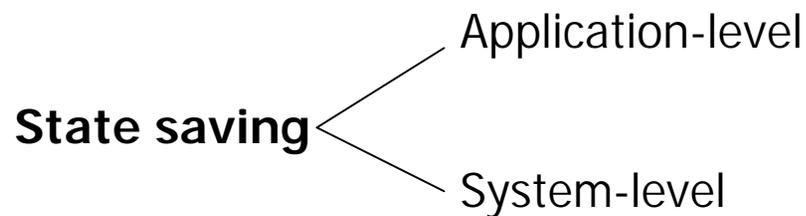
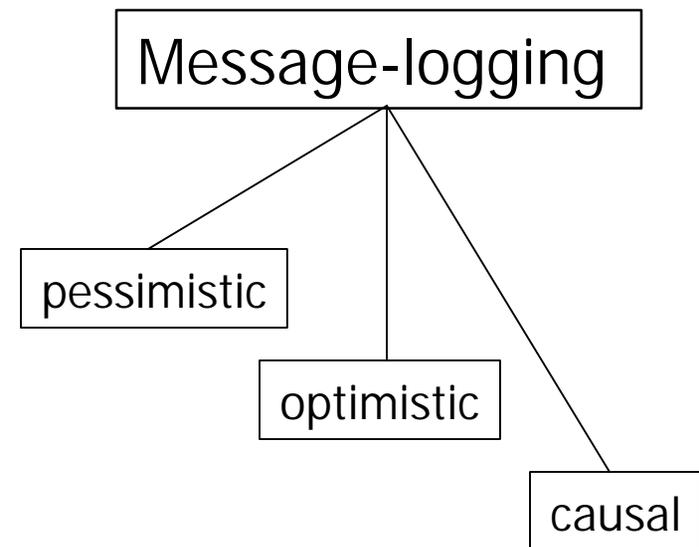
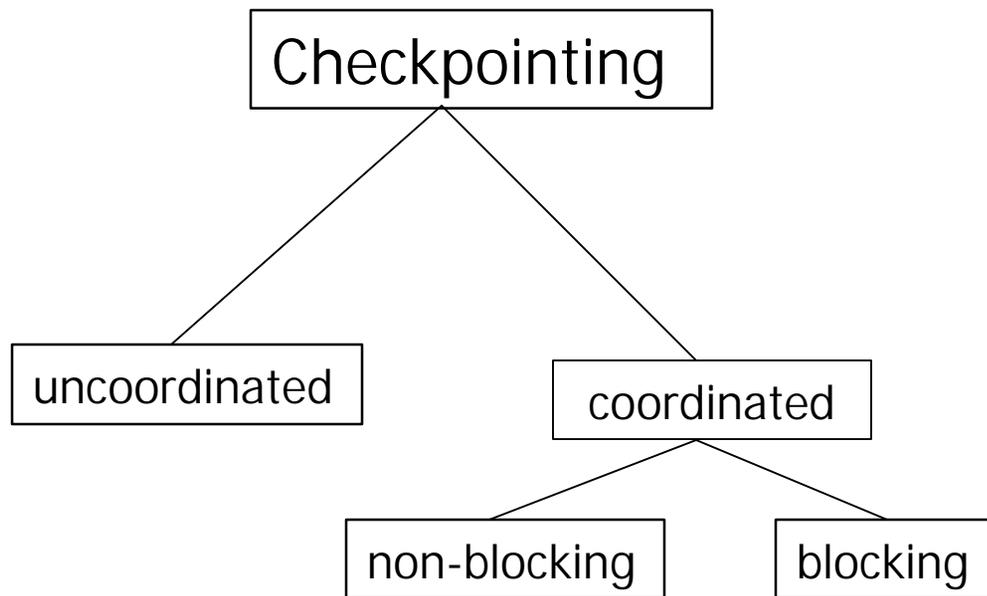


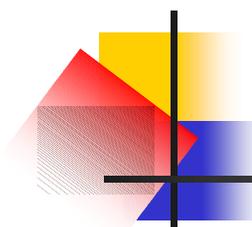
Fault tolerance

- Fault tolerance comes in different flavors
 - Mission-critical systems: (eg) air traffic control system
 - No down-time, fail-over, redundancy
 - Computational applications
 - Restart after failure
 - Minimize expected time to completion



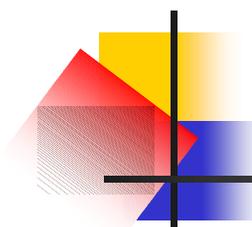
Fault tolerance strategies





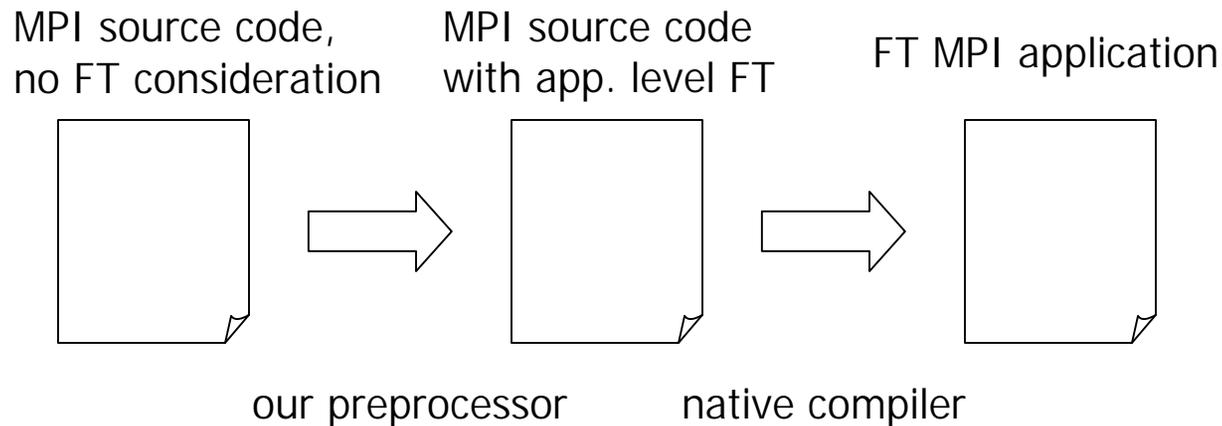
Our experience/beliefs:

- Message-logging does not work well for communication-intensive numerical applications
 - Many messages, much data
- System-level checkpoint is not as efficient as application-level
 - IBM's BlueGene protein folding
 - Sufficient to save positions and velocities of bases
 - Alegra talk
 - App. level restart file only 5% of core size

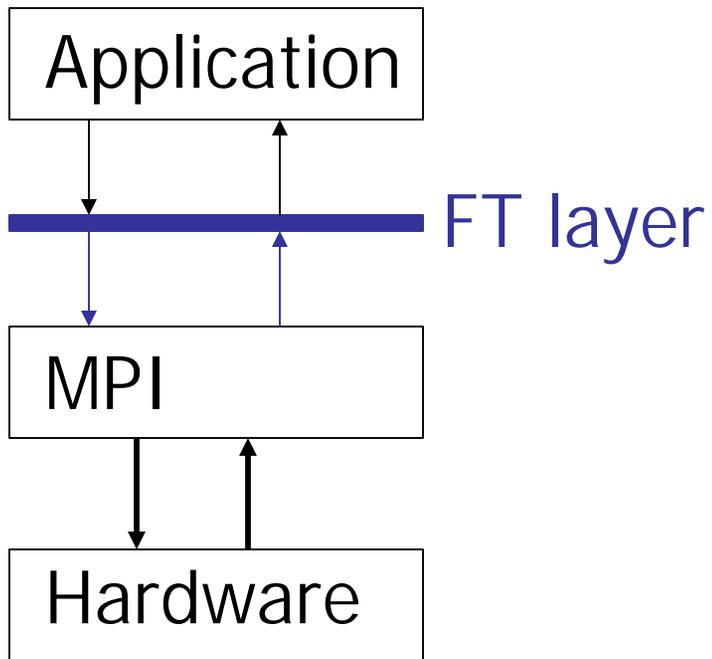


Our goal

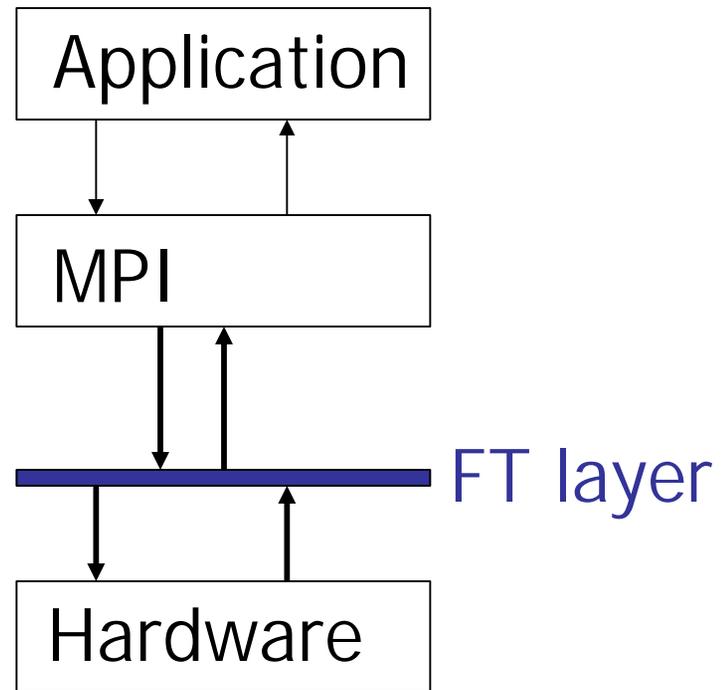
- Develop a preprocessor that will transparently add application-level checkpointing to MPI applications
 - As easy to use as system-level checkpointing
 - As efficient as user-specified application-level checkpointing

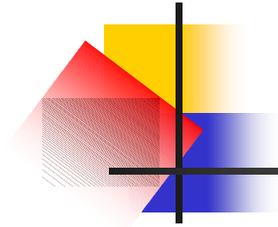


Choices for Runtime layer



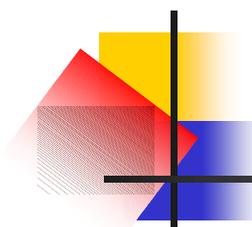
Our choice





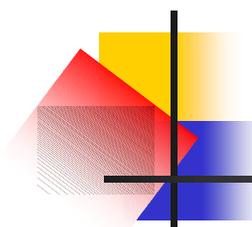
Outline

- Introduction
- Application-level FT for sequential applications
- Problems in supporting MPI applications
- Approaches to solving these problems
- Status and ongoing work



Sequential application state

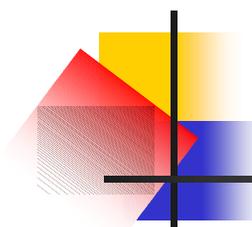
- An application's state consists of
 - Program counter
 - Call stack
 - Globals
 - Heap objects
- Similar in technique to PORCH
 - Ramkumar, Strumpfen (Iowa / MIT)



Example

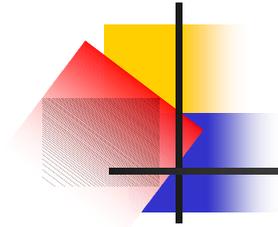
```
main()
{
    int a;
    VDS.push(&a, sizeof a);
    if(restart)
        load LS;
        copy LS to LS.old
        jump dequeue(LS.old)
    // ...
    LS.push(2);
label2:
    function();
    LS.pop();
    // ...
    VDS.pop();
}
```

```
function()
{
    int b;
    VDS.push(&b, sizeof b);
    if(restart)
        jump dequeue(LS.old)
    // ...
    LS.push(2);
    take_ckpt();
label2:
    if(restart)
        load VDS;
        restore variables;
    LS.pop();
    // ...
    VDS.pop();
}
```



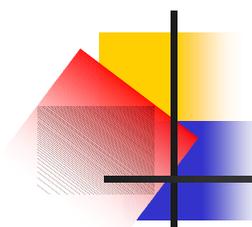
Optimizations

- Where should we checkpoint?
 - CATCH
 - Li, Fuchs (Illinois)
- Memory exclusion
 - Live/Clean/Dead variable analysis
 - Plank, Beck, Kingsly (Univ. Tennessee)
- Recomputation vs. restoring
 - Protein folding example



Outline

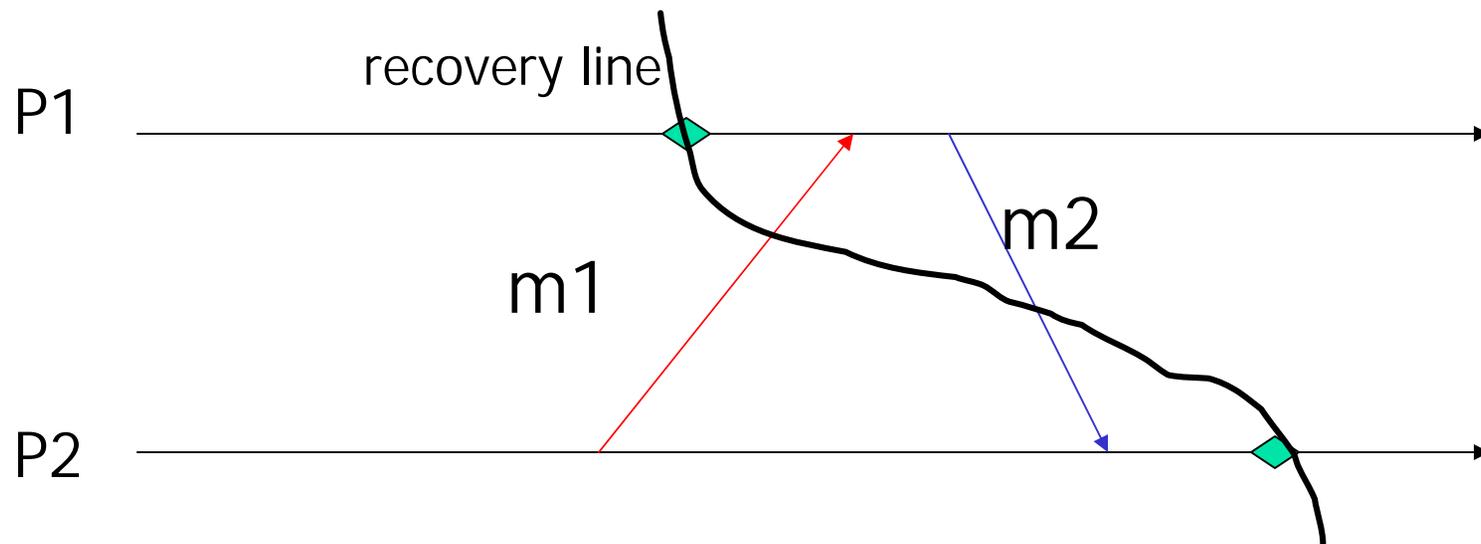
- Introduction
- Application-level FT for sequential applications
- Problems in supporting MPI applications
- Approaches to solving these problems
- Status and ongoing work



Supporting MPI applications

- It is not sufficient to take a checkpoint of each individual process
- We need to account for the following
 - In-flight messages
 - Inconsistent messages
 - Non-blocking communication
 - “Hidden” MPI state
 - At application level, message send/receive not necessarily FIFO
 - Process can use tags to receive messages out of order

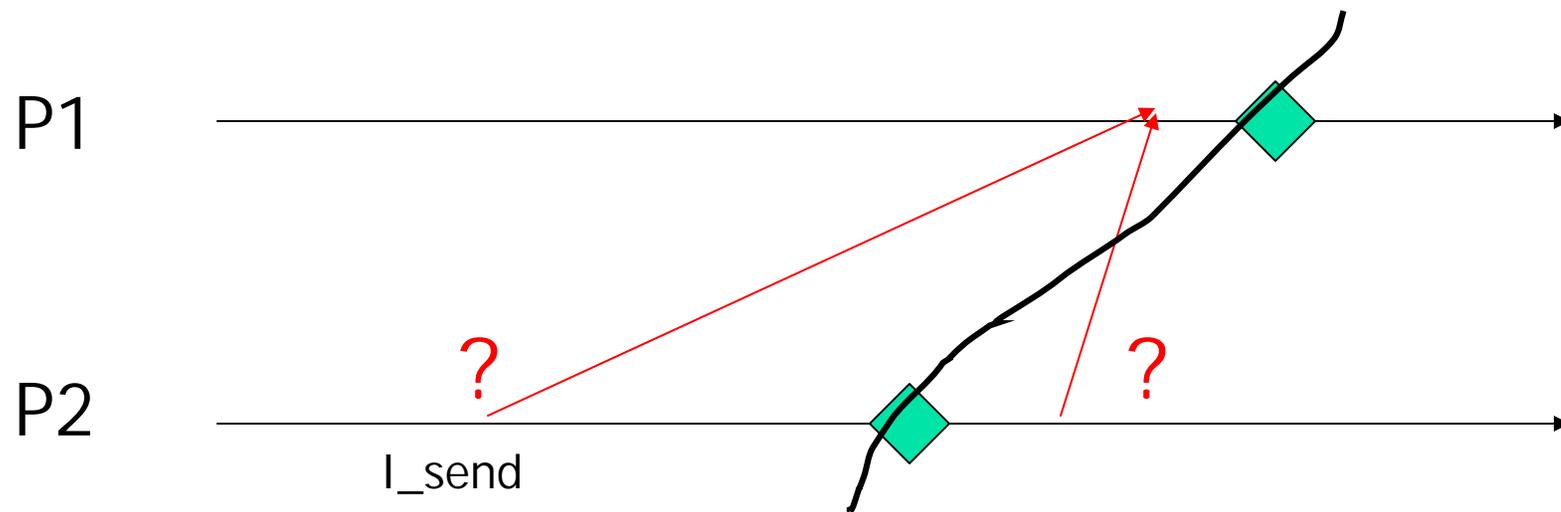
In-flight and inconsistent messages



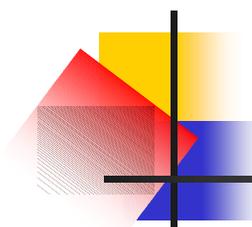
- m1 is in-flight (sent but not recvd)
- m2 is inconsistent (recvd but not sent)

Non-blocking communication

- MPI allows for non-blocking communication



- Did the send happen before or after P2's checkpoint was taken?
- If it happened before, it is consistent. If it happened after, it is inconsistent.

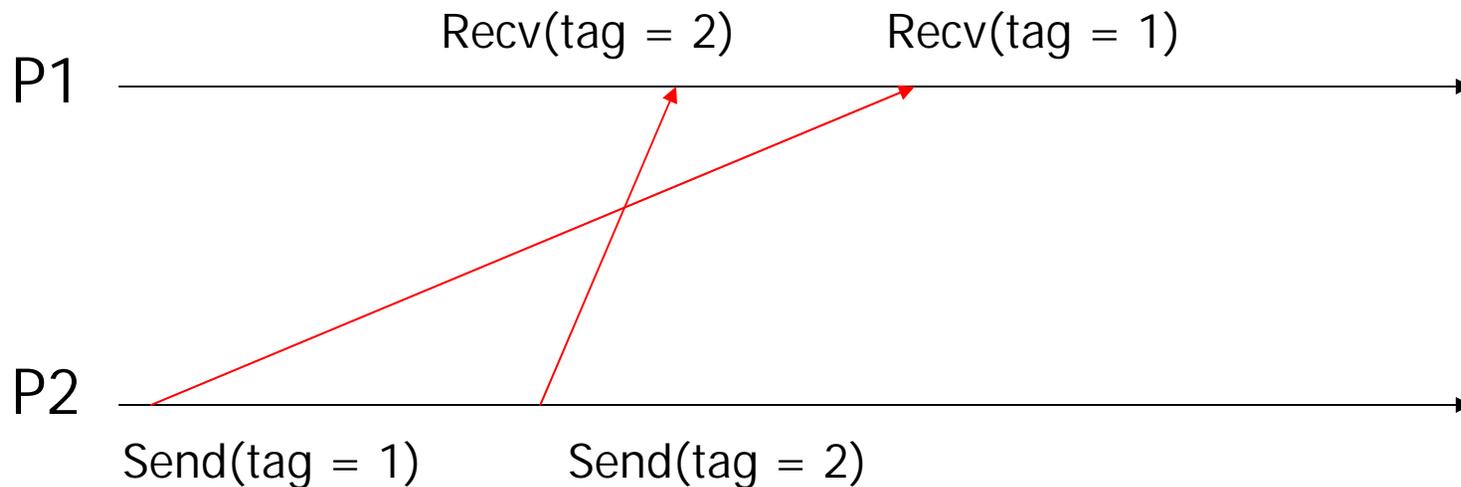


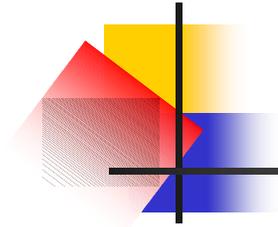
“Hidden” MPI state

- Need to save and restore the state of the MPI library
- This state is hidden from our preprocessor
- Two kinds of hidden state
 - **Persistent** - communicators, groups, etc.
 - Not correct to take system-level ckpt
 - **Volatile** - request objects (not handles)

Non-FIFO receive order

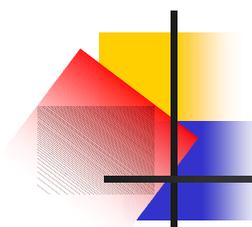
- Applications may receive messages in non-FIFO order
 - Two messages from P2 to P1 will be received in send order only if they have the same tag and communicator
- Most protocols assume FIFO





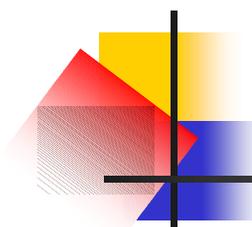
Outline

- Introduction
- Application-level FT for sequential applications
- Problems in supporting MPI applications
- Approaches to solving these problems
- Status and ongoing work



Beliefs

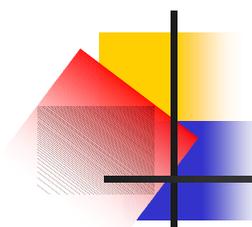
- Complexity of making program FT may vary from program to program
 - Not all programs will exhibit all the problems described earlier
- FT protocol should be customized to complexity of program
 - Minimize the overhead of fault tolerance



Degrees of complexity

Non-FIFO MIMD
MIMD(eg. Task parallelism)
Iterative Synchronous
Bulk Synchronous
Parametric computing

↑
Increasing
complexity
of protocol



Parametric computing

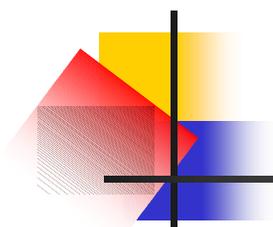
- Parametric computing, i.e. embarrassingly parallel

Distribute work

Do work

Collect Results

- No communication in “Do work” area
- Can take uncoordinated checkpoints within that area
 - Each takes its own checkpoints

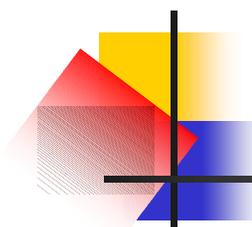


Bulk synchronous

- “Phase-step” model of computation

```
do work 1  
barrier  
do work 2  
barrier  
do work 3
```

- Communication and computation in “do work” areas
- Use blocking coordinated checkpointing, provided
 - no messages cross the barrier
 - no transient hidden state that crosses the barrier
 - →requires compiler analysis



Analysis problems

```
If(rank = 0)
    send(1)
Else
    send(0)

Barrier

If(rank = 0)
    recv(1)
Else
    recv(0)
```

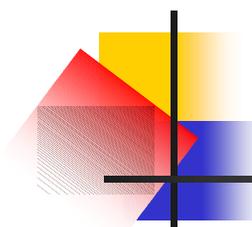


```
If(rank = 0)
    I_send(&r)
Else
    I_recv(&r)

Barrier

Wait(&r)
```



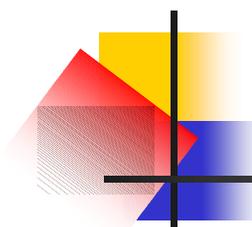


Iterative synchronous

- Each process runs the same number of iterations of a loop

```
for(i...)  
{  
    Communicate  
    Compute  
}
```

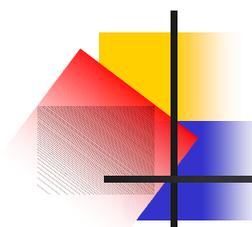
- Are there places where barriers can be (safely) inserted?
 - If so, treat as bulk synchronous



Analysis problem

```
For()  
{  
    if(rank = 0)  
        x = 1  
    else  
        x = 2  
    if(x = 1)  
        Barrier?  
}
```

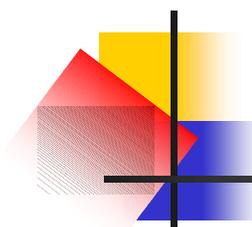
```
For()  
{  
    if(rank = 1)  
        recv  
  
        Barrier?  
  
    if(rank = 0)  
        send  
}
```



Task parallel (e.g. producer / consumer)

```
If(rank = 0)
{
    while(not done)
        send(DATA)
    send(DONE)
}
Else
{
    int x;
    while(1)
        recv(ANY_TAG)
        if(tag = DATA)
            x += f(DATA)
        else
            break
}
```

- There are no interesting (useful) places to insert barriers
 - Can't use blocking protocol
 - Must use non-blocking protocol

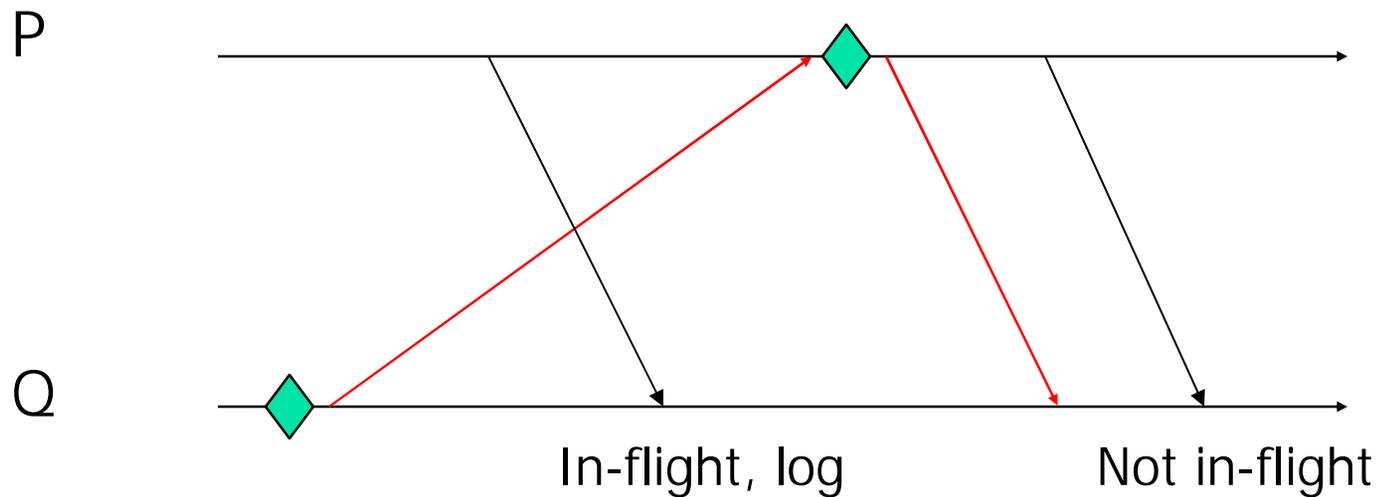


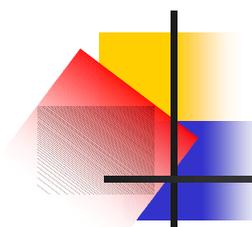
Non-blocking protocol

- Chandy-Lamport is a simple, well-known, coordinated non-blocking protocol
 - Assumes FIFO channels
 - Initiator takes local checkpoint, and sends marker to neighbors
 - On receiving marker, process takes checkpoint and sends its marker to neighbors
 - After taking checkpoint, process P logs all messages from process R, until R's marker arrives
 - These are in-flight messages

Example

- Process Q initiated the checkpoint.
- It logs all messages from P until P's marker arrives
- On restart, Q "receives" from log until empty



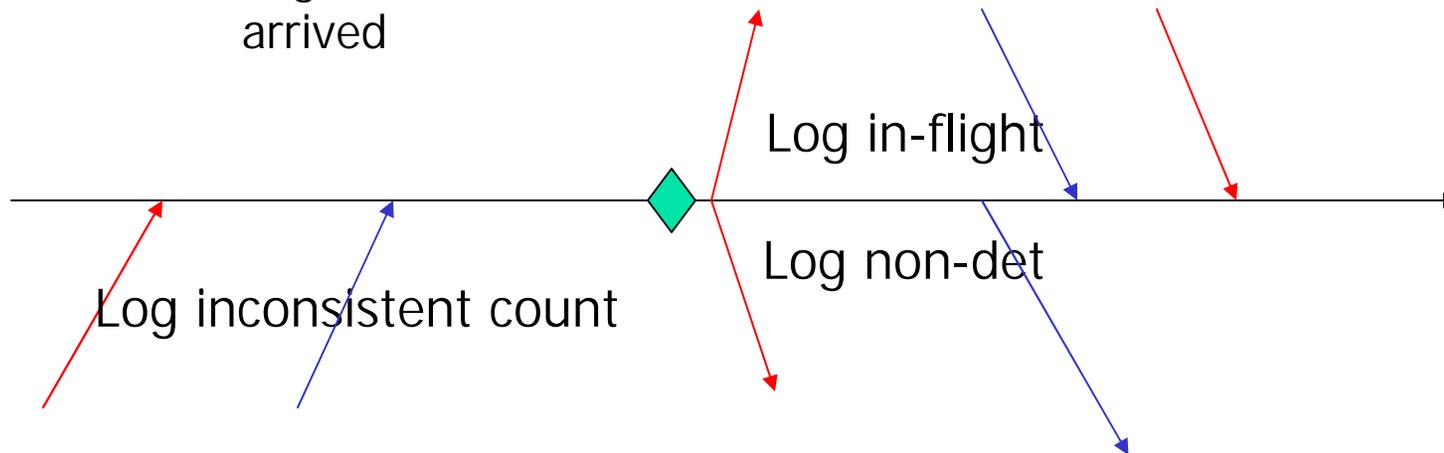


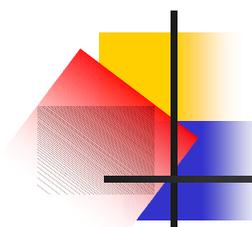
Drawbacks of C-L protocol

- Does not work for application-level checkpointing
 - In C-L, process must checkpoint as soon as it receives a marker from a neighbor
- Assumes fixed communication graph
- Assumes FIFO communication
- No notion of collective communication

CL with delayed checkpointing

- Before checkpoint
 - log count of all messages from R that arrive after R's marker arrived
- After checkpoint
 - Log all messages that arrive from S until S's marker arrives
 - Log all non-deterministic choices made until all markers have arrived

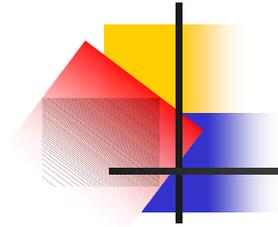




Degrees of complexity

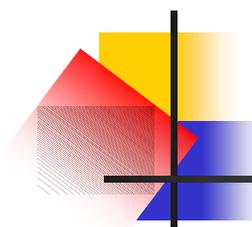
Non-FIFO MIMD
MIMD(eg. Task parallelism)
Iterative Synchronous
Bulk Synchronous
Parametric computing

↑
Increasing
complexity
of protocol



Outline

- Introduction
- Application-level FT for sequential applications
- Problems in supporting MPI applications
- Approaches to solving these problems
- Status and ongoing work



Status

- Completed
 - Preprocessor for saving/restoring sequential state
 - No optimizations
- In progress
 - Application API
 - Determining checkpoint locations
 - Support for in-flight/non-FIFO msgs/.....
 - Implementing modified CL protocol
 - Support for saving volatile hidden MPI state
 - Analysis problems