Recovery Strategies

- Event Sequence
  - fault detection
  - fault location
  - system reconfiguration
  - system recovery
    - different for distributed and shared memory systems
  - continuation of operation
- Trivial Recovery
  - termination of program and re-execution
    - performance disadvantage, unrealistic

Recovery Strategies

- Desirable Recovery
  - Forward Recovery
    - Would like to continue execution right from where the error was detected and ensure alternative mechanism to ensure correctness
  - Rollback Recovery
    - Rollback program to a previous correct state
    - If rollback is inevitable, prefer not to roll back too far
    - Need to frequently store system state that can be “rolled back” to
  - Checkpointing
    - Storing system state information at discrete points in the program
    - Q: What constitutes a system state?
    - A: Depends on system…

Recovery Strategies

- Forward Recovery
  - e.g. **Recovery blocks**
    - approach based on software redundancy
    - introduced by Brian Randell (1975)
    - basic idea is that a language construct is used that supports software redundancy
      - assume acceptance test T and “try blocks” B_i
      - B_1 is the primary try block, B_k is the (k-1)th alternative

```
ensure T by $B_1$
          else by $B_2$
          …
          else by $B_n$
          else error
```

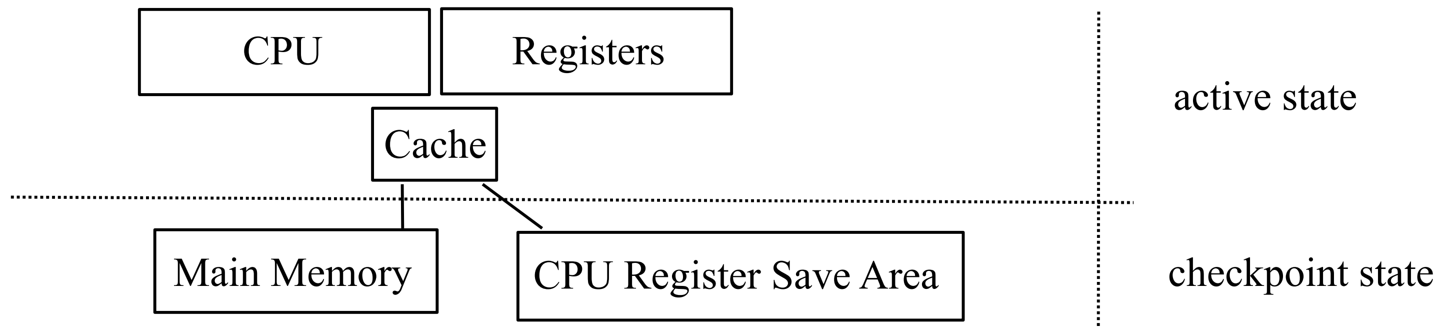
Recovery Strategies

- Rollback Recovery based on Checkpoints
  - We will first discuss this topic at the processor level
    - Processor Cache-Based Checkpoints
    - Virtual Checkpoints
- Processor Cache-Based Checkpoints
  - Powerful for building a machine that can tolerate transient faults
  - Checkpoint defined by storing registers in safe area of main memory and writing back cache lines to main memory.
    - safe main memory area (stable storage)
    - => e.g. battery back-up, redundant memory
  - Program now executes using active data in cache.
    - i.e. Write-through cache policy is not suitable
  - Cache miss forces a checkpoint

Recovery Strategies

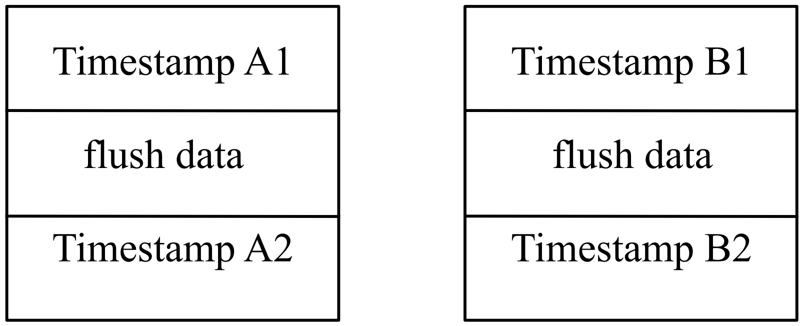
- Example
  - Active State: CPU, Registers, Cache
  - Checkpoint State: Main Memory, CPU Register Save Area
  - Cache employs copy-back policy, i.e. changes are made in cache only.
  - Now assume a parity error
  - Rollback by
    - invalidating dirty cache line or page
    - reload register state from CPU Register Save Area

Pradhan96 Fig 3.7



Recovery Strategies

- What happens if failure occurs during copying the state?
  - » then the checkpoint state has been partially updated
  - » neither old nor new state are valid
- Sequoia fault tolerant multiprocessor
  - » duplex processors operating in lockstep,
  - » duplexed main memory banks provide atomic checkpoint process
  - » uses 2 banks and time stamps (Pradhan96 Fig 3.8)
  - » sequence: TA1 - flush - TA2 - TB1 - flush - TB2
  - » use partial order of time stamps to determine which bank is corrupt



Recovery Strategies

Pradhan96 table 3.4

Condition	Failure	Action
TA1 = TA2 = TB1 = TB2	None	None
TA1 > TA2 = TB1 = TB2	Flush A	Copy Bank-B to A
TA1 = TA2 > TB1 = TB2	Between	Copy Bank-A to B
TA1 = TA2 = TB1 > TB2	Flush B	Copy Bank-A to B

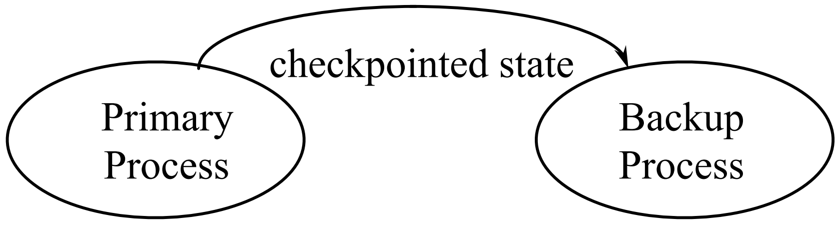
Note that only three time stamps are needed since TA2=TB1
Interpretation of "=": E.g. if both TA1 and TA2 are written (TA1 = TA2) then checkpoint has been successfully written.

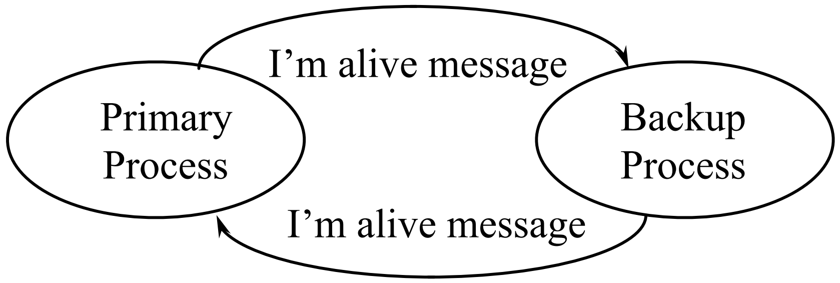
Recovery Strategies

- Virtual Checkpoints
  - Drawbacks of cache based checkpointing
    - checkpoint frequency depends on cache size
    - checkpoint frequency can be very high
    - resulting in high performance overhead
  - When checkpoint frequency is high => move strategy from processor cache into virtual memory.
    - thus include the state of memory into the checkpoint
    - use disk to store checkpoint
    - derivation of scheme where global and local checkpoint numbers V and v are used.
    - now, the active page will become the checkpoint when V is updated.

Recovery Strategies

- So far only uni-processor has been considered
- "Processor Pair" approach
  - used in Tandem
  - primary process executes actively
  - backup process is inactive
  - primary periodically sends up-to-date messages containing its state to the backup process
  - same principle as traditional roll-back scheme, however instead of checkpoint, the state is saved in backup process
  - assumes Fail-Stop behavior of processors





Recovery Strategies

- Rollback Recovery in Multiprocessors
  - communication needs to be considered
    - effect of communication since last checkpoint
    - communication might have inherent delays
  - different issues in different systems
  - tightly coupled multiprocessors (shared memory)
  - loosely coupled multiprocessors (no shared memory)

Recovery Strategies

- Tightly Coupled Multiprocessors
  - use write-back strategy
    - possibility for multiple copies of the same data in different caches
    - need cache coherence protocol
      - if different processors update a certain cached page, inconsistency may/will arise
    - write-through strategy would result in extremely large checkpoint frequency since every write would result in new checkpoint

Recovery Strategies

- Cache Coherence Protocols
  - single-bus-based
    - write-invalidate:
      - whenever write is to be performed, send out signal to invalidate all preexisting copies of the same piece of data in other caches before updating the local copy in the cache
    - write-update:
      - better if writes are less frequent
      - deliver write-data to other users simultaneously whenever a write access to a shared cache block is generated
  - directory-based
    - use table (called directory) to keep tabs on the state of each memory block
    - various schemes differ in their table management

Recovery Strategies

- The Problem of Rollback Propagation
  - with multiple processes data dependency might cause one rollback to trigger a domino effect,
  - i.e. the rollback of a process requires the rollback of another process ...
- Ahmed’s Solution
  - additional bus lines
    - <u>shared</u>: set by processor to indicate sharing of a block on the bus
    - <u>establish rollback point</u>: set by processor to indicate that a rollback point is being established
      - this causes all other processes to also establish checkpoints
      - causes problems as the number of processes increases
    - <u>rollback</u>: set by processor to indicate that it is backing up to the prior rollback point
      - causes all other processes to also rollback

Recovery Strategies

- Wu, Fuchs, Patel approach
  - again, checkpoint is taken whenever dirty cache line must be written back into memory
  - this checkpoint is however local
    - other processors are not required to also checkpoint
  - to prevent rollback propagation a checkpoint is also initiated whenever another processor reads a cache line modified since the last checkpoint
  - checkpoint identifiers are used to identify with which checkpoint a particular modified line is associated

Recovery Strategies

- Advantage of tightly coupled systems
  - easy to move failed process from a faulty to a good processor
  - where does one move it too?
    - stay on same processor or migrate to other processor?
    - typically: stay on same processor and keep track of transient faults
  - if processor fails rollback is needed
  - however, this rollback means that the checkpoint state must be activated by a different processor.
  - but, since the system is tightly coupled
    - no problem, “just” load from shared memory

Recovery Strategies

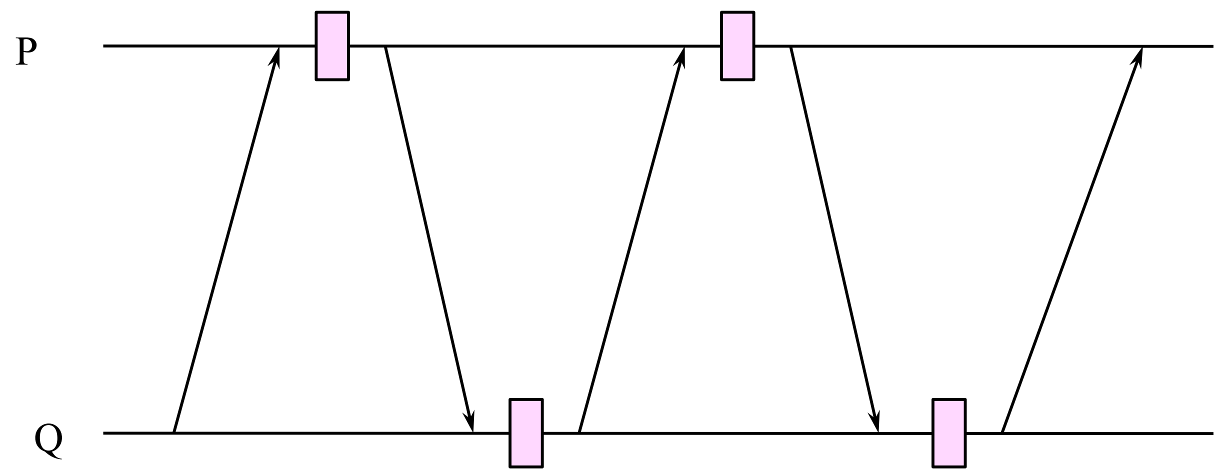
- Loosely coupled Multiprocessors
  - just local memory
  - assumes that when processor fails, other processors are informed in finite time.
  - message delays cause problems
  - assume that no message delay exists then the following cannot be:
    - » state of process *P* indicates it has sent message *m* to process *Q*, but *Q*'s state reflects that message *m* has not been received.
    - » converse situation: *Q*'s state indicates received message, but *P*'s state indicates that it has not send message yet.

Recovery Strategies

- A set of recovery points form a consistent system state if:
  - the set contains exactly one recovery point for each process
  - there is no event for sending a message in a process P <u>succeeding</u> its recovery point, whose corresponding receive event in another process Q occurs <u>before</u> the recovery point of Q that is contained in the set.
  - there is no event for sending a message in a process P <u>preceding</u> its recovery point, whose corresponding receive event in another process Q occurs <u>after</u> the recovery point of Q that is contained in the set.
- Recovery line
  - set of checkpoints across <u>all</u> processors to which the programs executing on various processors can be rolled back.

Recovery Strategies

- *Domino Effect*:
  - in the event of an error, multiple processors might have to roll back to the beginning of the program in order to obtain a consistent set of checkpoints across all processors.



Now assume that Q aborts: both P and Q roll back to the beginning

Recovery Strategies

- Message Logging
  - discussion based on
    - L. Alvisi and K. Marzullo, *Message Logging: Pessimistic, Optimistic, Causal, and Optimal,* IEEE TSE, Vol. 24, No 2, Feb 1998.
  - addresses crash faults
  - each process
    - 1) periodically records its local state
    - 2) logs the messages it received after having recorded that state
  - when process crashes
    - 1) new process is created in its place
    - 2) process is initiated with appropriate recorded local state
    - 3) process is replayed the logged messages in the order they were originally received.

Recovery Strategies

- Message Logging
  - requirement: state of the recovered process must be consistent with the states of the other processes

    => consistency requirement

  - consistency requirement
    - expressed in terms of *orphan processes*
      - which are surviving processes whose state are inconsistent with the recovered state of the crashed process
    - message-logging protocols guarantee that upon recovery no process is an orphan.

Recovery Strategies

- Message Logging
  - Pessimistic protocols
    - avoid creation of orphans during an execution
  - Optimistic protocols
    - take appropriate actions during recovery to eliminate all orphans
  - Causal message-logging protocols
    - neither create orphans when there are failures, nor do they ever block a process when there are no failures

Recovering a Consistent State

More on recovery

Material based on “Fault Tolerance in Distributed Systems”, by Pankaj Jalote.

- Recovery with multiple processes
  - problematic if processes are communicating
  - checkpointing and rollback is not as simple as in single-proc. system
  - state of the system now consists of the state of its processes
- Problem
  - there is no direct method for establishing a checkpoint at the same time at all the different nodes to capture the state of the entire system at a particular time instance

Recovering a Consistent State

- Naïve solution
  - let each node establish a checkpoint independently
  - a collection of these checkpoints is interpreted as system-wide checkpoint
  - what would be the problem of this approach?

Recovering a Consistent State

- Backward-recovery
  - asynchronous checkpointing
    - checkpoint at different nodes is not coordinated
    - sufficient information is kept in the system to allow system to be rolled back to consistent state
  - coordinated checkpointing
    - coordinate the establishment of checkpoints at different nodes
    - this set of checkpoints forms consistent system state
    - also called *synchronous checkpointing*
  - Goal of rollback scheme
    - take the system back to a state that has either existed at some time in the execution, or could have existed in this execution.

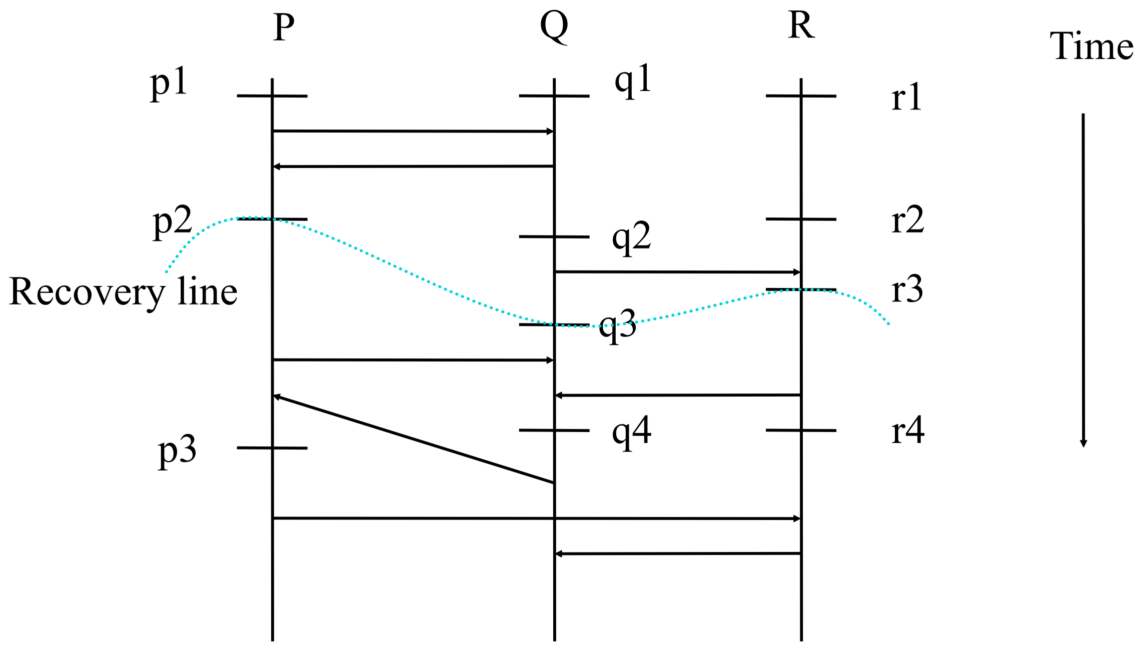
Recovery Strategies

- Recall this slide :-)

- A set of recovery points form a consistent system state if:
  - the set contains exactly one recovery point for each process
  - there is no event for sending a message in a process P succeeding its recovery point, whose corresponding receive event in another process Q occurs before the recovery point of Q that is contained in the set.
  - there is no event for sending a message in a process P preceding its recovery point, whose corresponding receive event in another process Q occurs after the recovery point of Q that is contained in the set.
- Recovery line
  - set of checkpoints across all processors to which the programs executing on various processors can be rolled back.

Recovering a Consistent State

- example



Assume P is rolled back to p3
-R has to be rolled back to r4 tp “unreceive” the message from P
- this rollback “nullifies” the message it sent after r4 to Q
- now Q has to roll back to q4 which “nullifies” its message to P
- so P rolls back to p2, causing Q to roll back to q3 which
- rolls R back to r3
- all processes stop at the recovery line