

HPC Fault tolerance



Two well known fault tolerance approaches

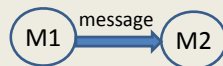
- **Replication**
 - Processes are replicated and executed in parallel
 - Replication requires twice hardware resources and energy
- **Re-execution** (optimized using Checkpoints & Rollback Recovery)
 - State is saved periodically
 - Rollback to saved state upon failure

1

Duplication



Different models for replicating communication



1) Independent execution



2) Full message duplication




3) Only main duplicate messages

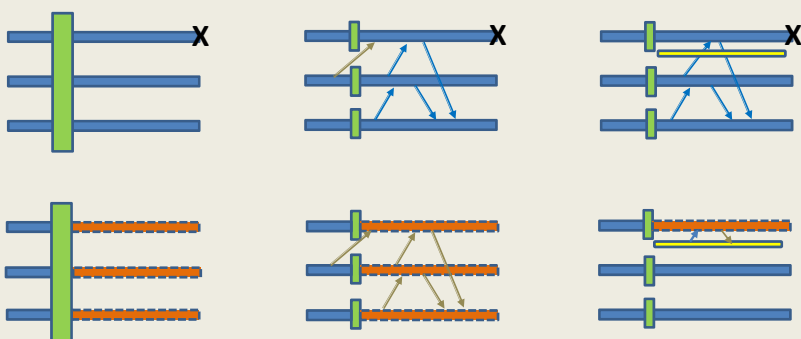


4) Main relays messages

Checkpointing



- Periodically pause execution and save state
- In event of failure restore from the last saved state




Global checkpoint

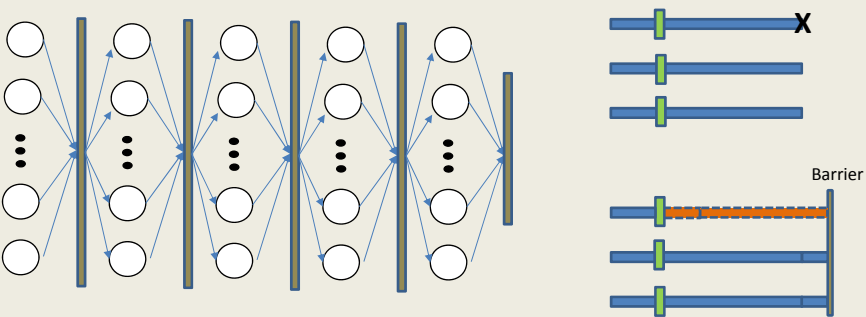
Local checkpoints
(coordinated @ barrier)
Global rollback

Local checkpoints
(coordinated @ barrier)
Local rollback

Effect of Barriers on Local Checkpointing




- The Bulk Synchronous Parallel model (BSP)
 - Dominant in High Performance Computing
 - Pregel: Google's extension to map/reduce (for analytics)

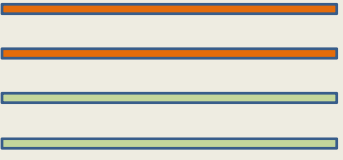


Replication

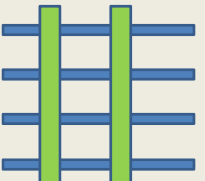
vs.

Checkpointing





$T/2$



$T/4 + 2 \text{ Checkpoints}$


$T = \text{serial execution time}$

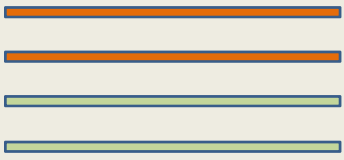
If each process is duplicated

- Can tolerate a fault in each pair of processes
- But execution time is doubled

But checkpointing is not as good as it looks


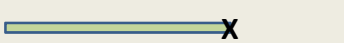
Effect of Synchronization/Communication






$T/2$



Duplication

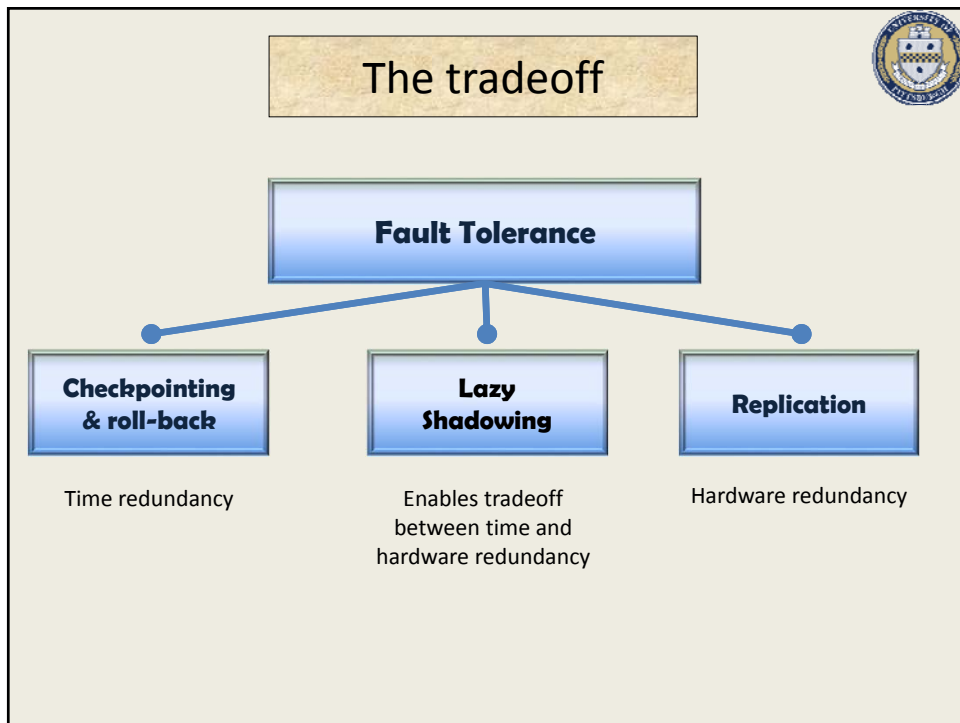
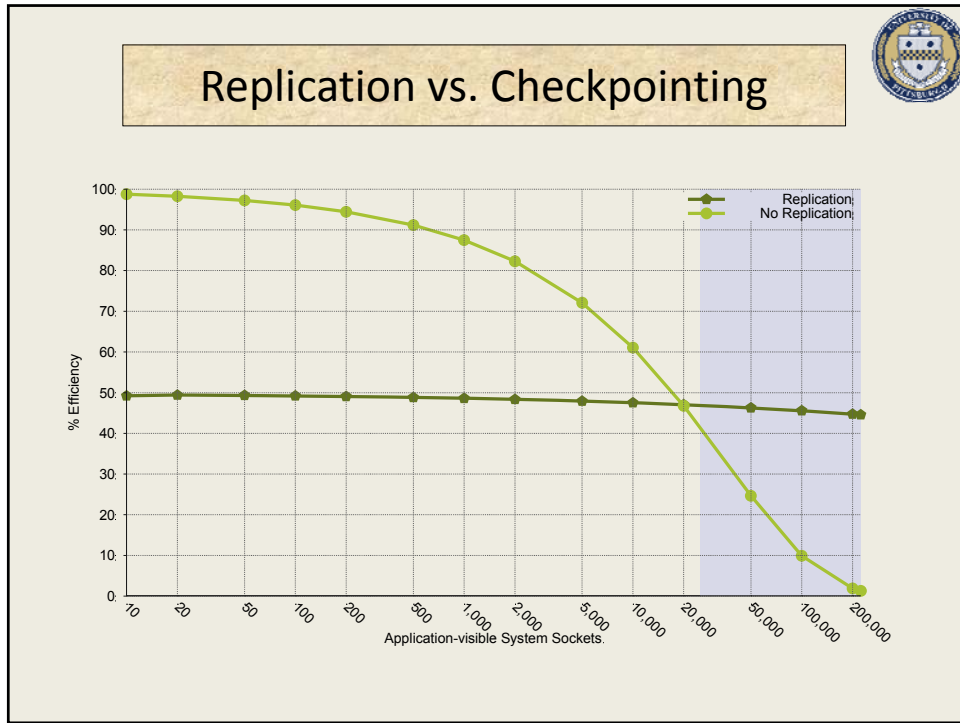


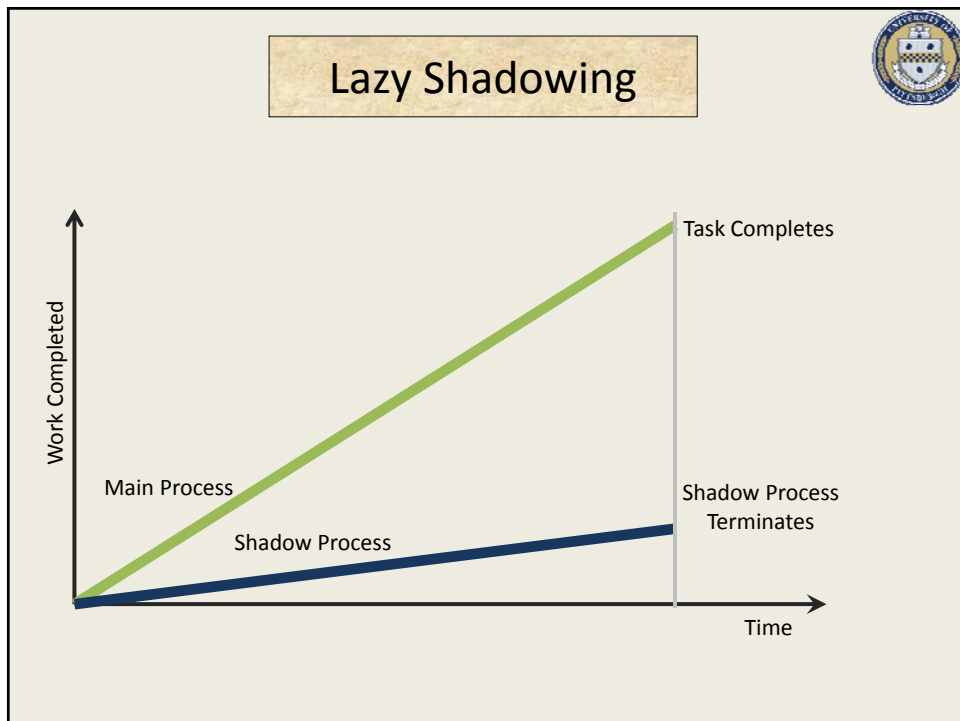
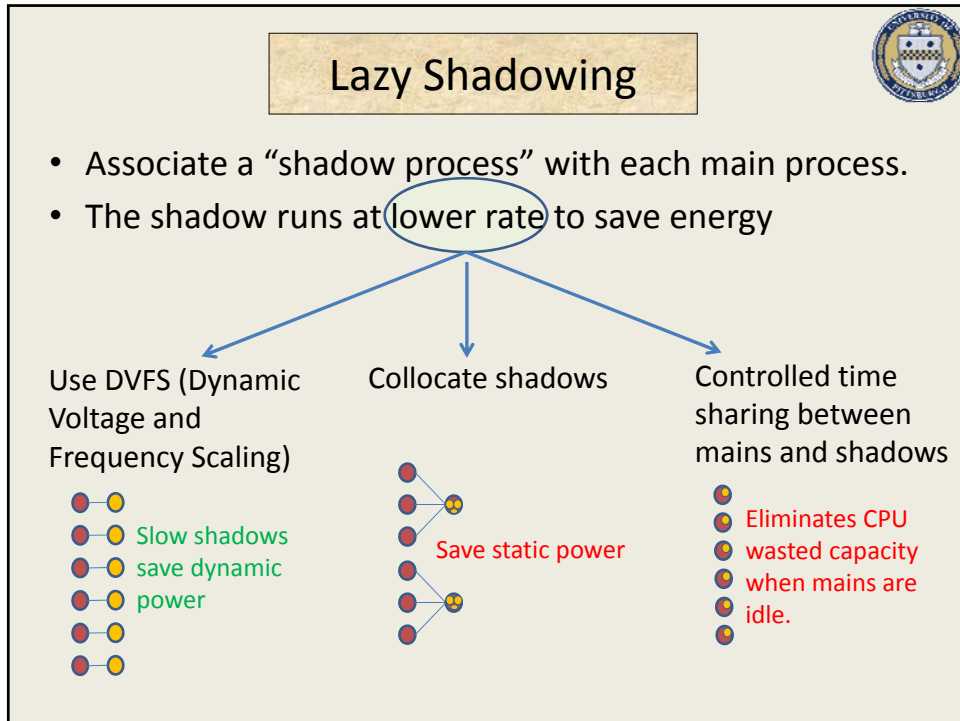
$T/4 + 2 \text{ Checkpoints}$

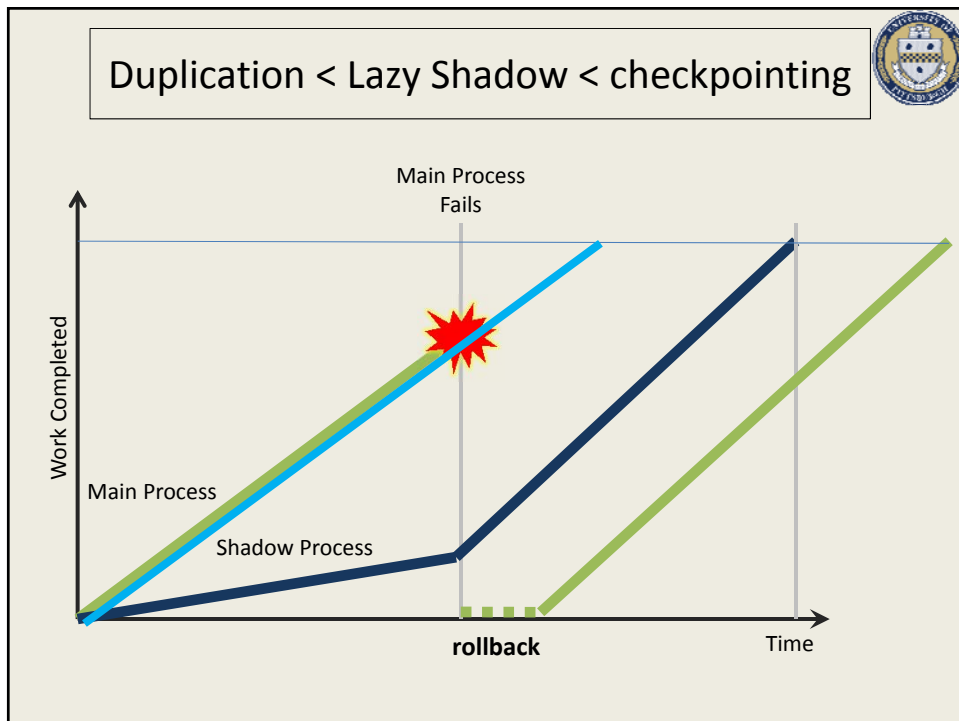
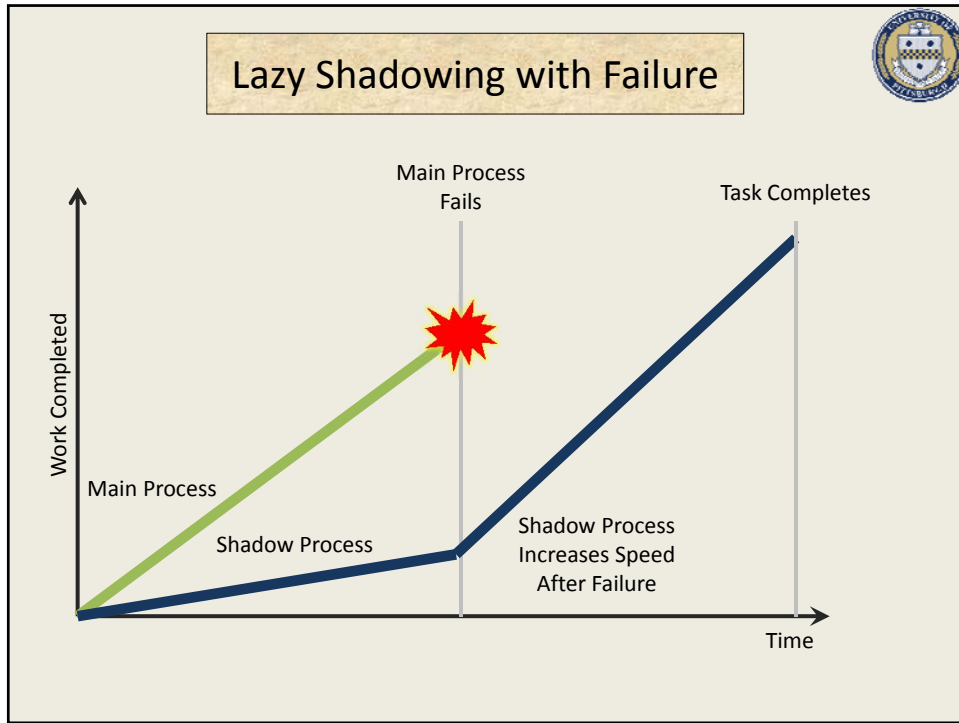
Checkpointing

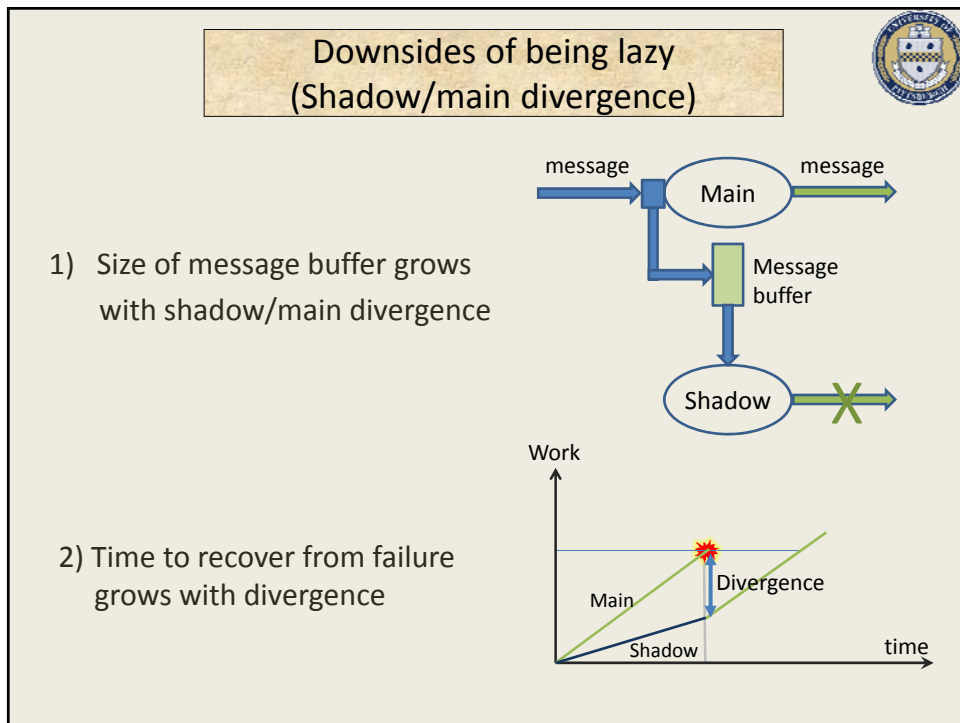
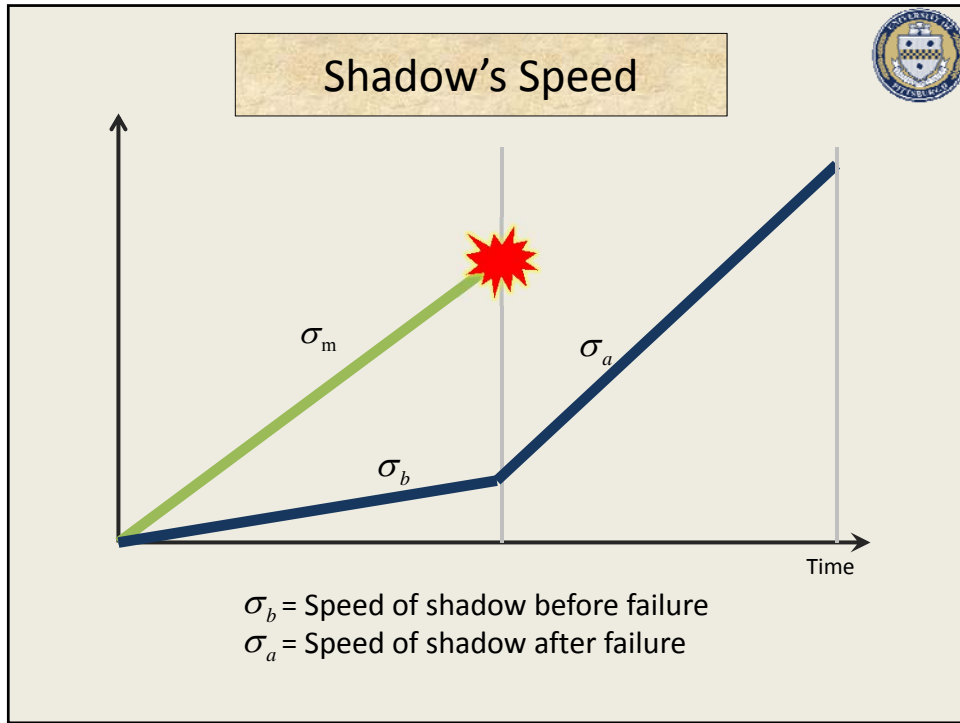



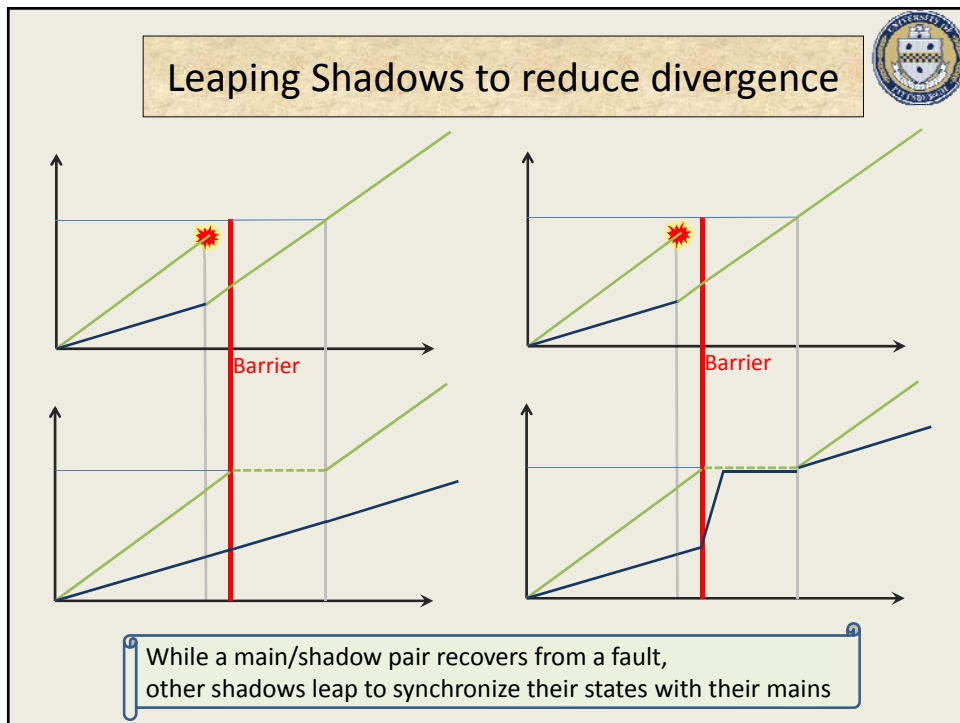
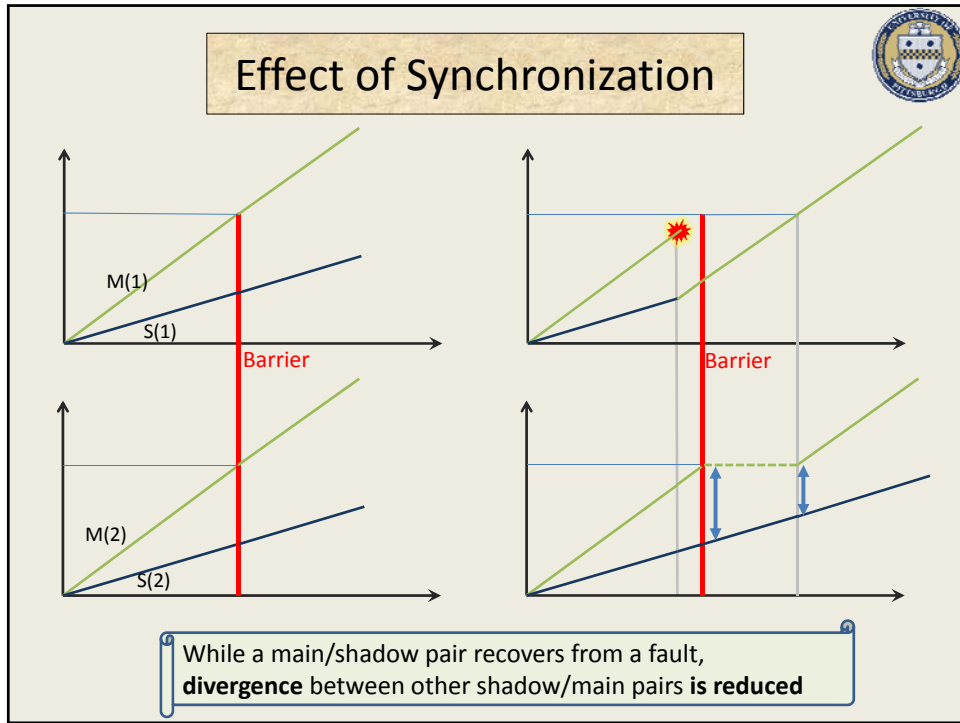
In large systems, checkpointing may take longer to execute and may consume more energy








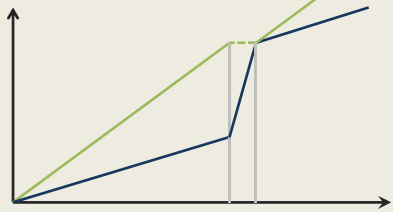




Forced leaping




- If shadow/main divergence reaches a threshold,
- Then force the shadow to leap



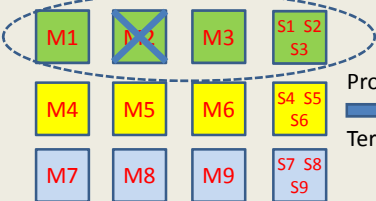
- Shadow leaping (both Fault-induced and Forced) requires rolling-forward the state of a shadow to the state of its main.
- Shadow leaping applies if execution speed is controlled by either DVFS or collocation.

Resilience implications of shadow collocation



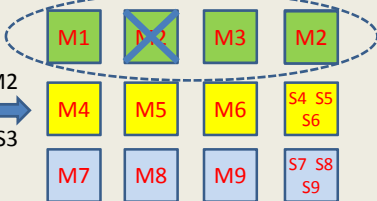
If shadows S_1, \dots, S_k are collocated on the same core, then the system can tolerate only one fault in the mains M_1, \dots, M_k

A shadowed set

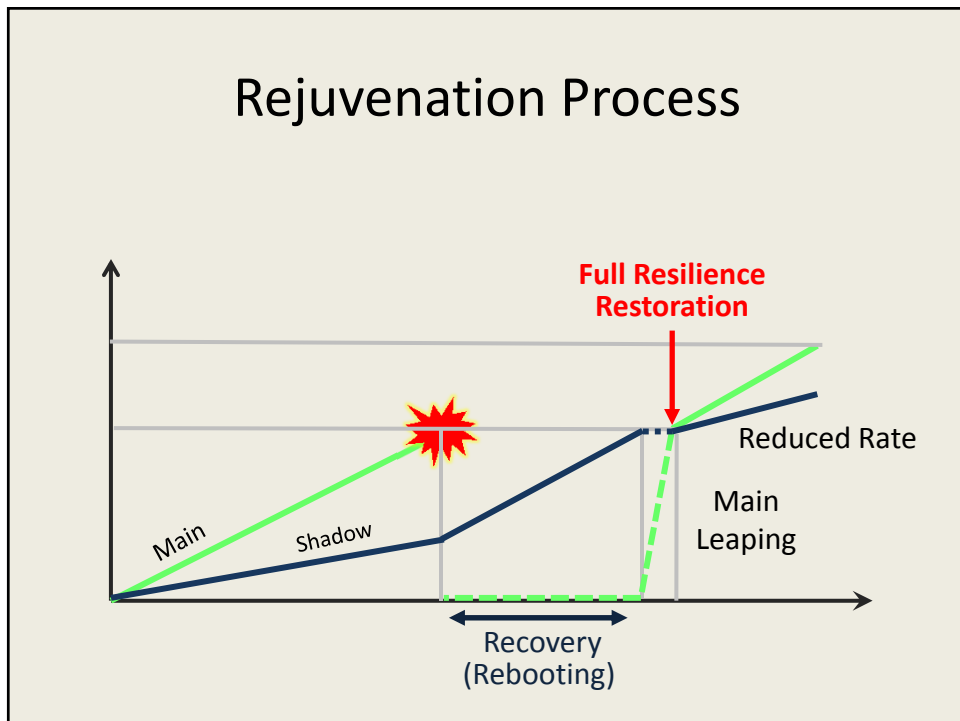
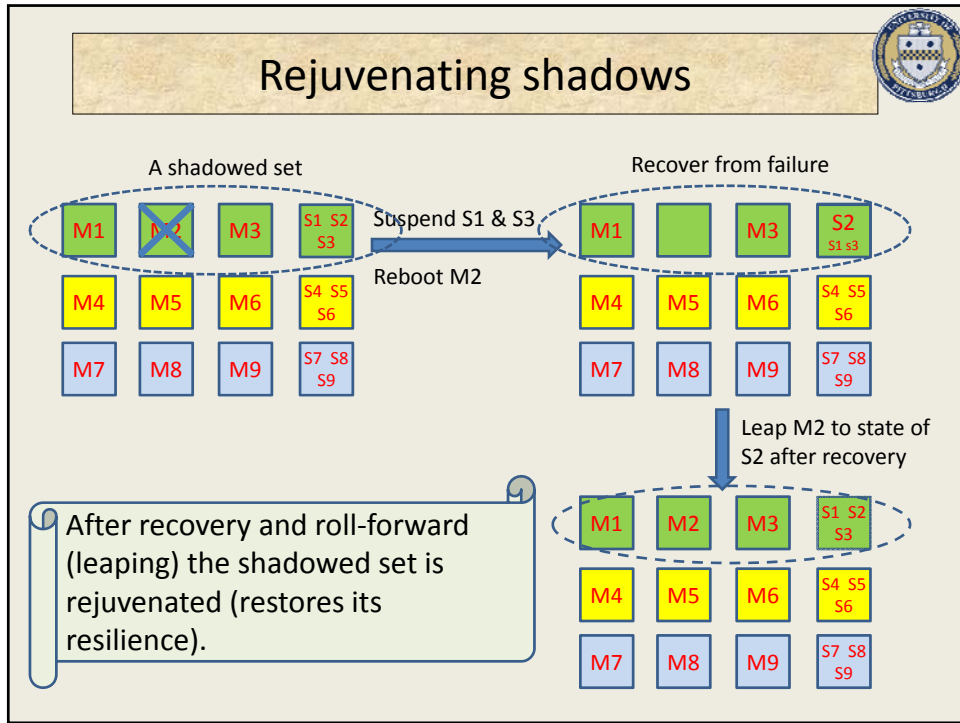


Promote S2 to M2
Terminate S1 & S3


A vulnerable shadowed set



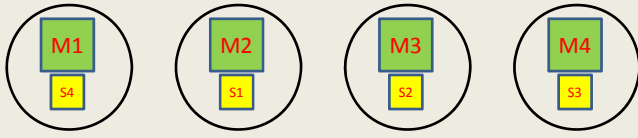
When many shadowed sets become vulnerable, the system needs to be rejuvenated.



Stealing Shadows




Instead of collocating shadows, collocated a main and a shadow

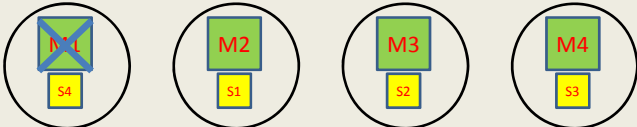


- Shadowed set size = 2
- Can control the speeds of mains and shadows through scheduling (priority – nice)
- Shadows steal cycles from mains when they are blocked (ex. waiting for communication or synchronization due to load imbalance).
- If put shadows at very low priority, they only advance when main are blocked (do not impede the speed of the mains)

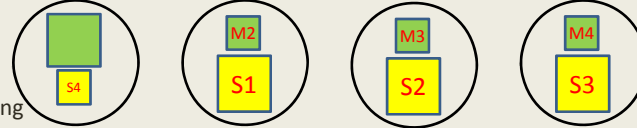
Stealing Shadows



When a main fails:
Synchronization causes other mains to block



Shadows speed up automatically
No need for shadow leaping



Rejuvenation:
After shadow recovers and main reboots, shadow transfers its state to the rebooted main

