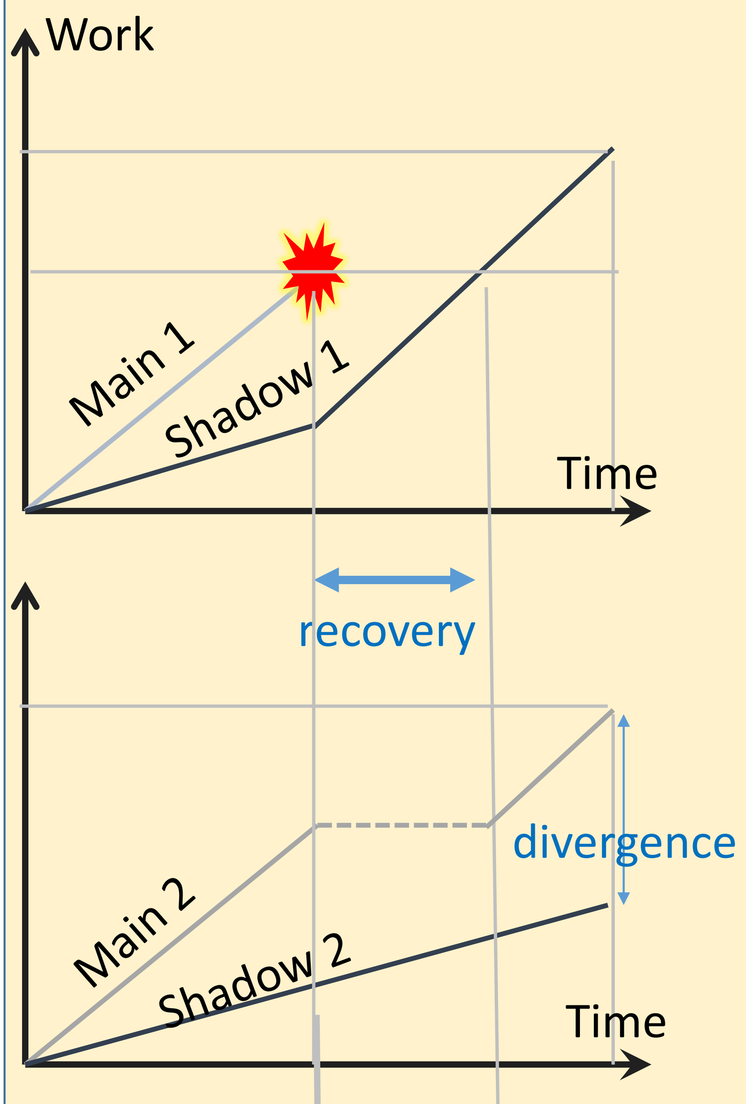
Lazy Shadowing

An Adaptive, Power-Aware, Resiliency Framework for Extreme Scale Computing

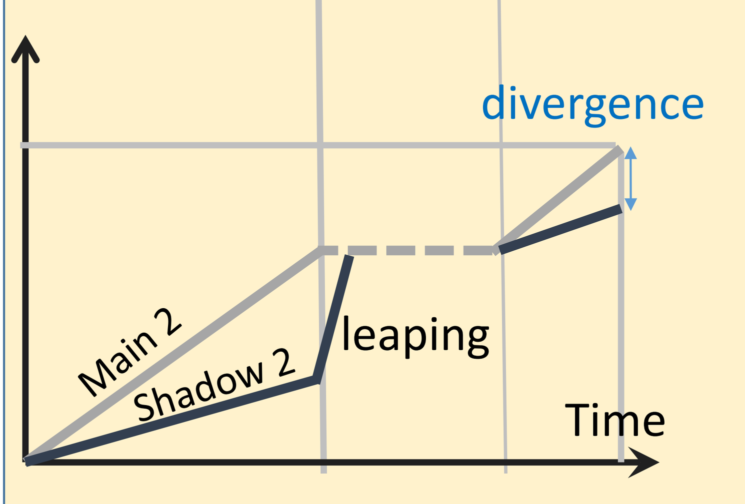
R. Melhem and T. Znati
U. Of Pittsburgh

K. Kant
Temple U.



- A shadow for each main
- Shadows run at reduced rate (energy/performance tradeoff)
- If main completes, shadow terminates (low overhead)
- If one main fails, shadow's rate increases (fast recovery)
- While a shadow recovers, other mains wait due to synchronization
- Recovery depends on gap between main and shadows (divergence)

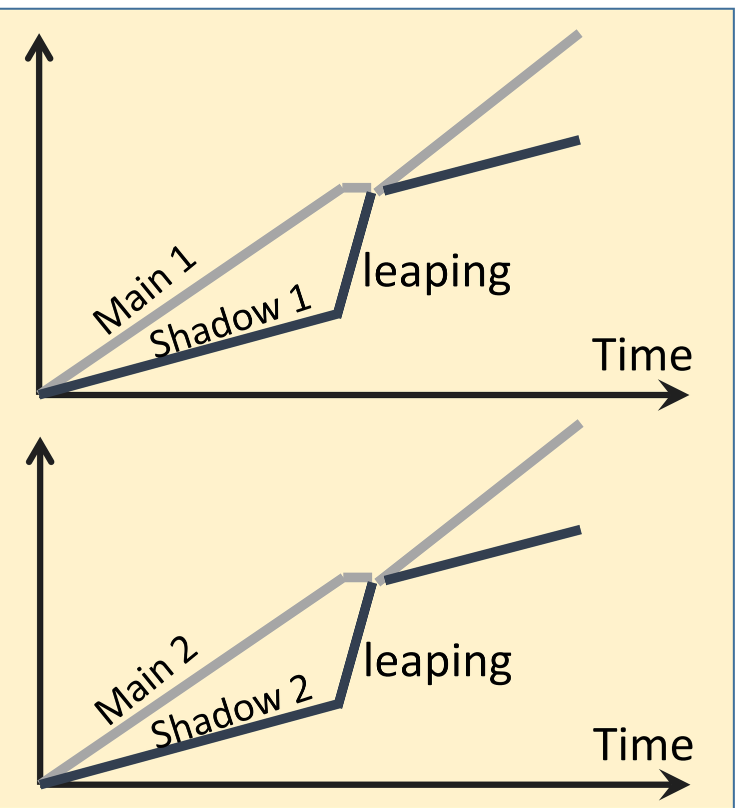
Fault-induced leaping



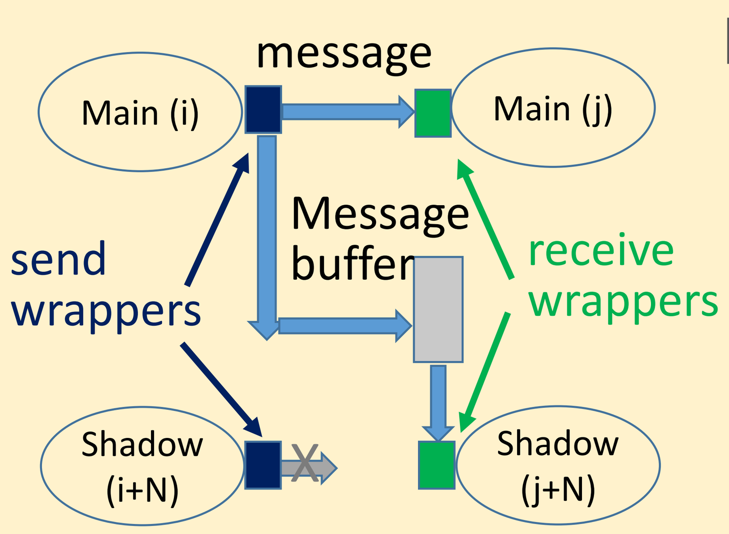
- Shadows roll-forward to the state of their mains
- Reduces recovery time for subsequent faults by reducing shadow/main divergence

Forced leaping

- Divergence grows in the absence of faults (causing long recovery from future faults)
- May periodically reduce divergence by leaping
- May leap when receive buffers at shadow exceeds a certain threshold.



Implementation through MPI call wrappers

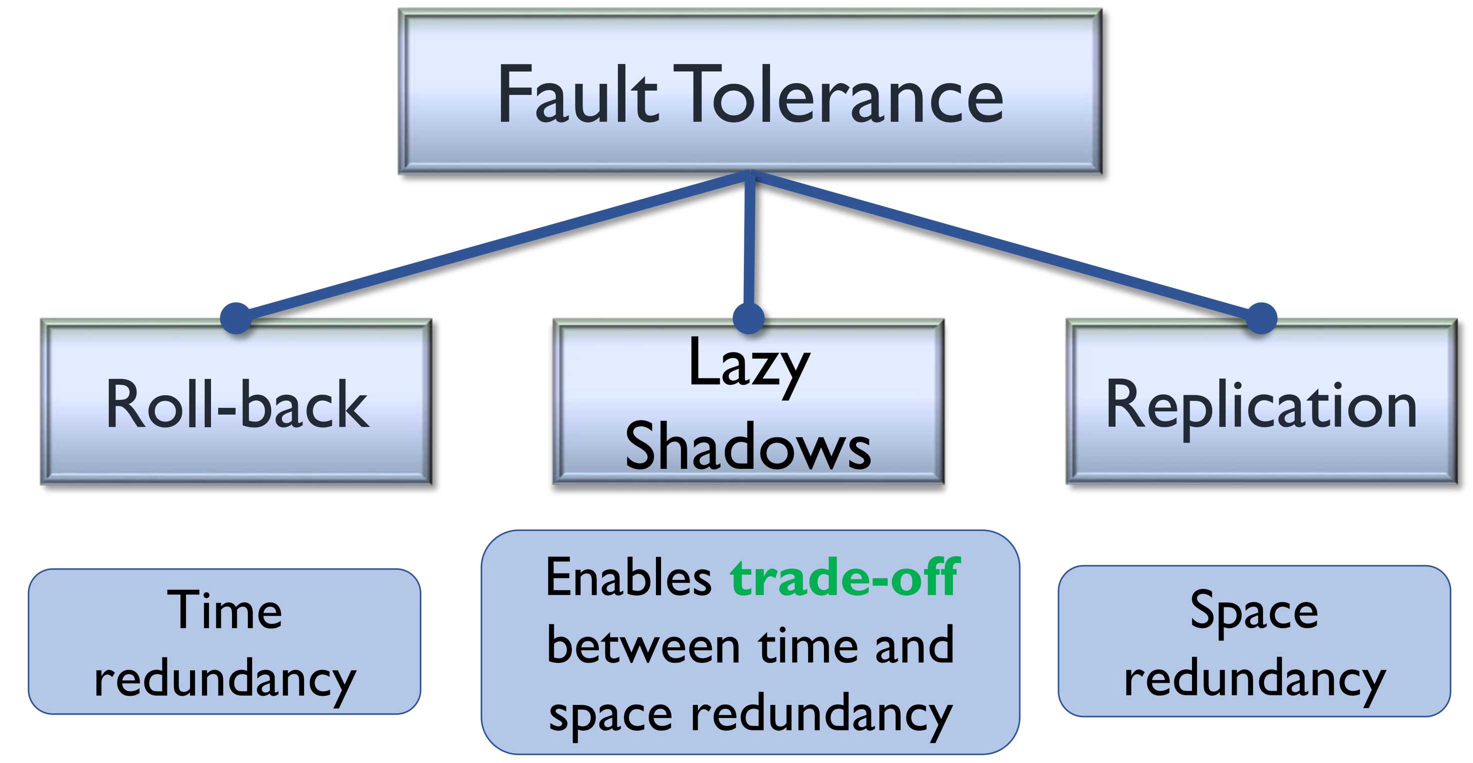


Example: MPI send/receive

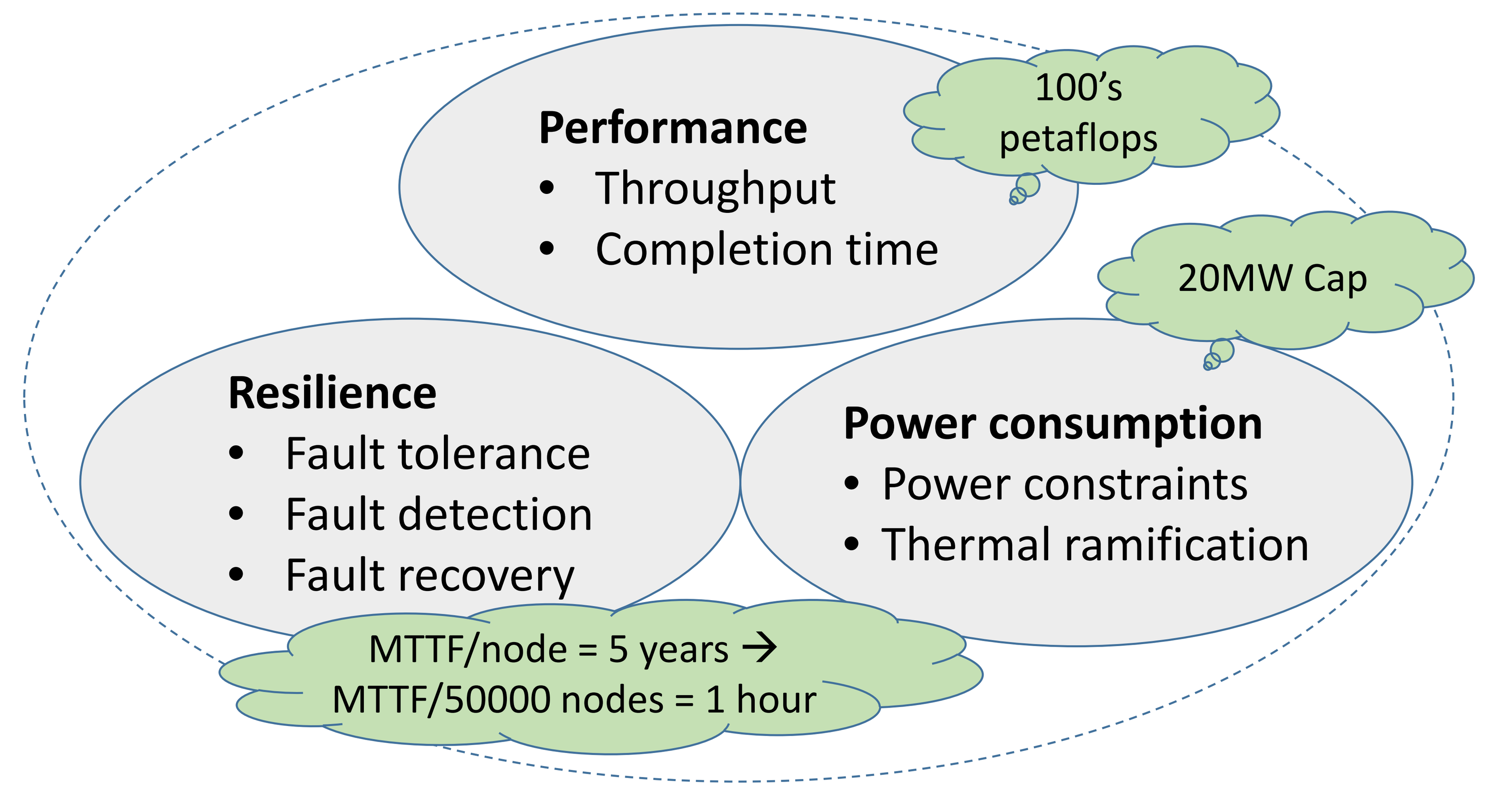
- Send at main: replicate msg.
- Send at shadow: suppress msg
- Receives at main: unchanged
- Receive at shadow: modify sender's rank

- Needs control messages to ensure deterministic execution in some MPI call (ex. any-source)
- For Leaping, user should register the state to be transferred (similar to user-level checkpointing)

The Fault Tolerance Spectrum



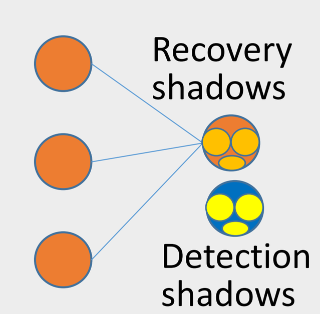
Conflicting requirements of extreme scale computing

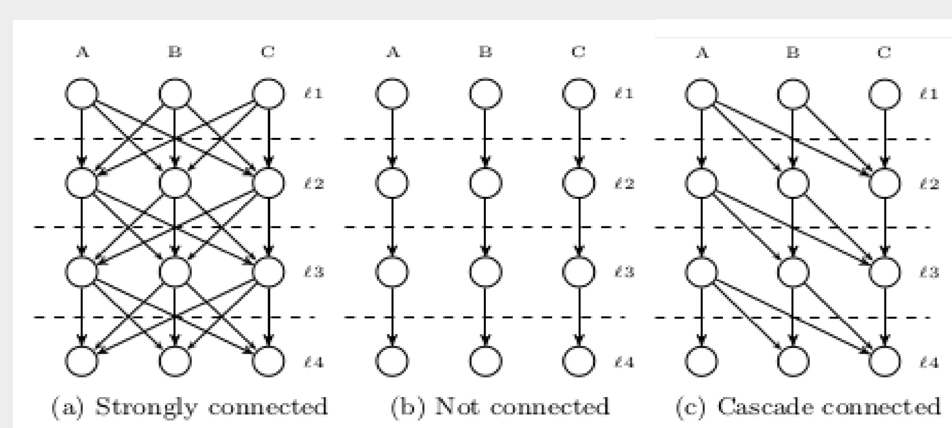


Objective = optimize any (or a combination) of the three
Constraints = bound any of the three

Lazy shadows for Slice-based fault detection

- Construct **sliced shadows** that computes only subsets of the state variables
- Acceptance tests on computed variables to check for errors

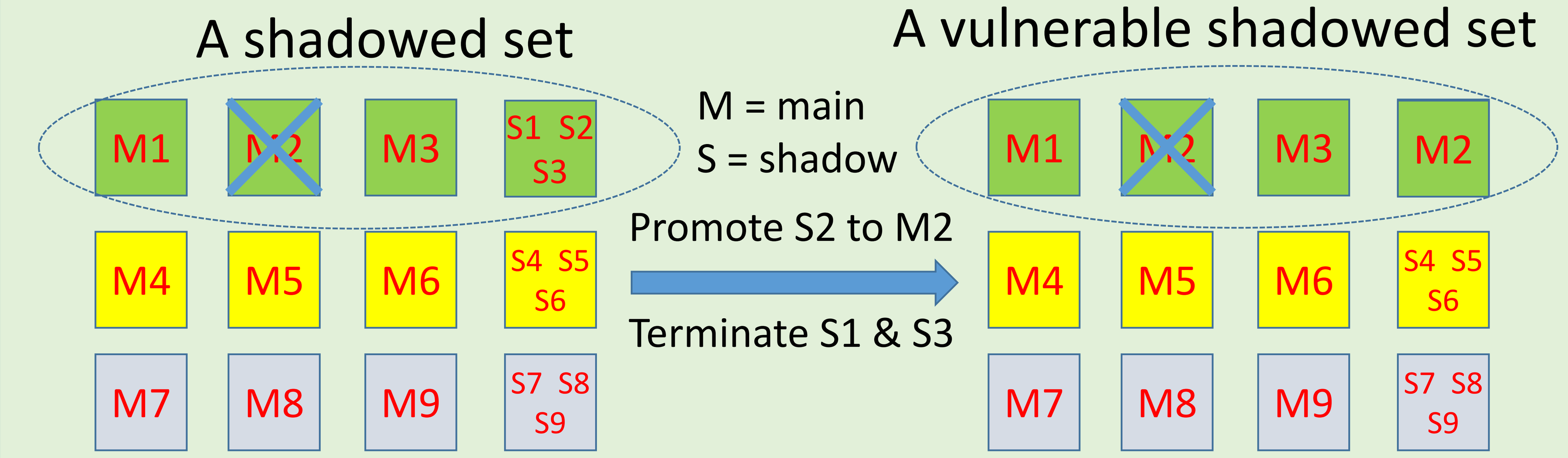




Sizes of slices depend on the control and data flows in the program

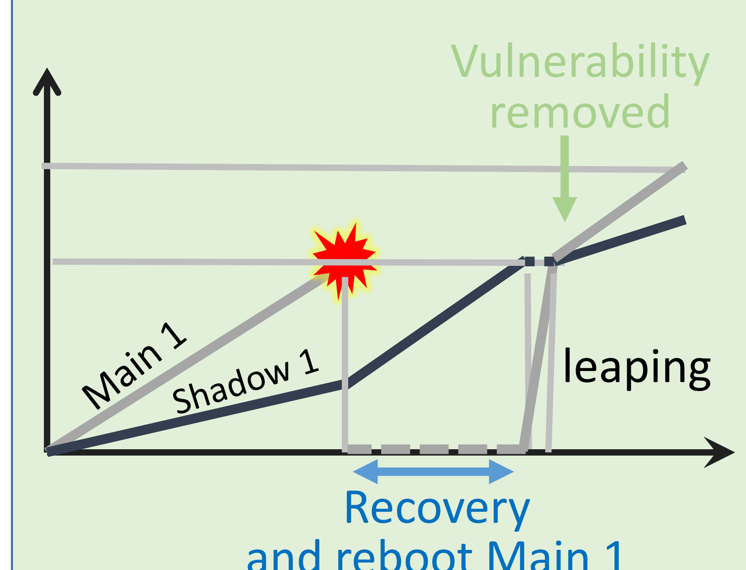
Laziness through shadow Co-location

- Reduce shadow's execution rate by overloading multiple shadows on the same processor
- May also reduce frequency/voltage
- Reduces **hardware** and **power** requirement
- Co-located shadows + their mains form a shadowed set



- A subsequent fault in a vulnerable shadow set = failure

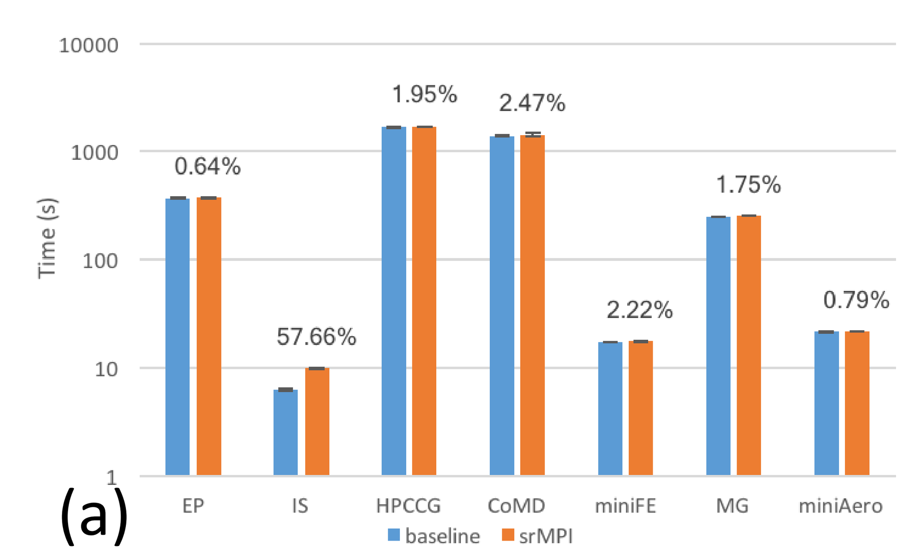
Rejuvenation to avoid vulnerability

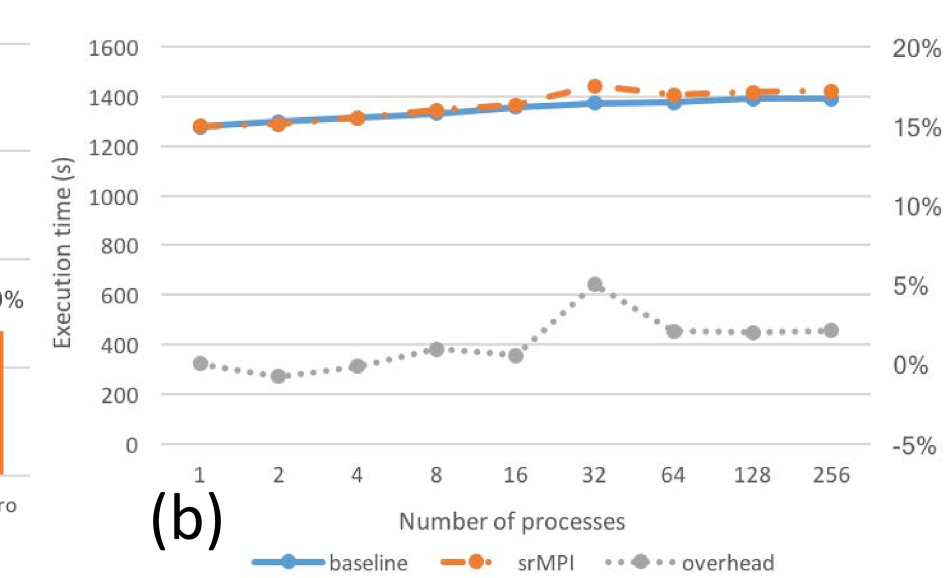


When a main fails

- Main reboots as shadow recovers
- Shadow recovers at full speed
- Rebooted main leaps to the recovered state and continues at full speed
- Shadow continues (at reduced rate) → system not vulnerable

Results from a prototype MPI implementation





a) Overhead for different benchmarks using 256 ranks
b) Scalability of overhead for HPCCG (fault-free execution)
c) Comparison with checkpointing for different number of faults