Shadow Computing – An Adative Power-Aware Resiliency Framework for Exascale Computing

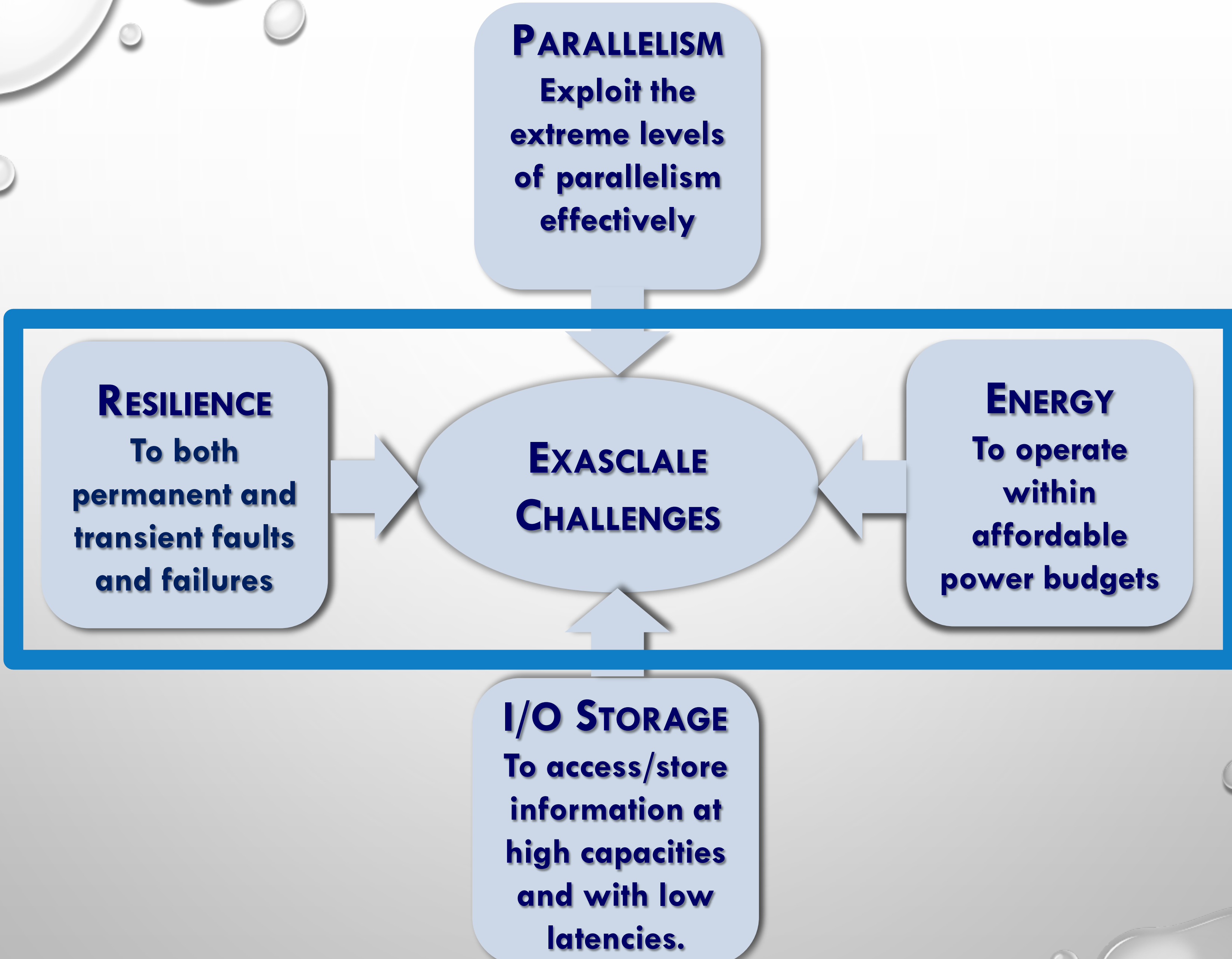
Taieb Znati‡, Rami Melhem‡ And Krishna Kant†

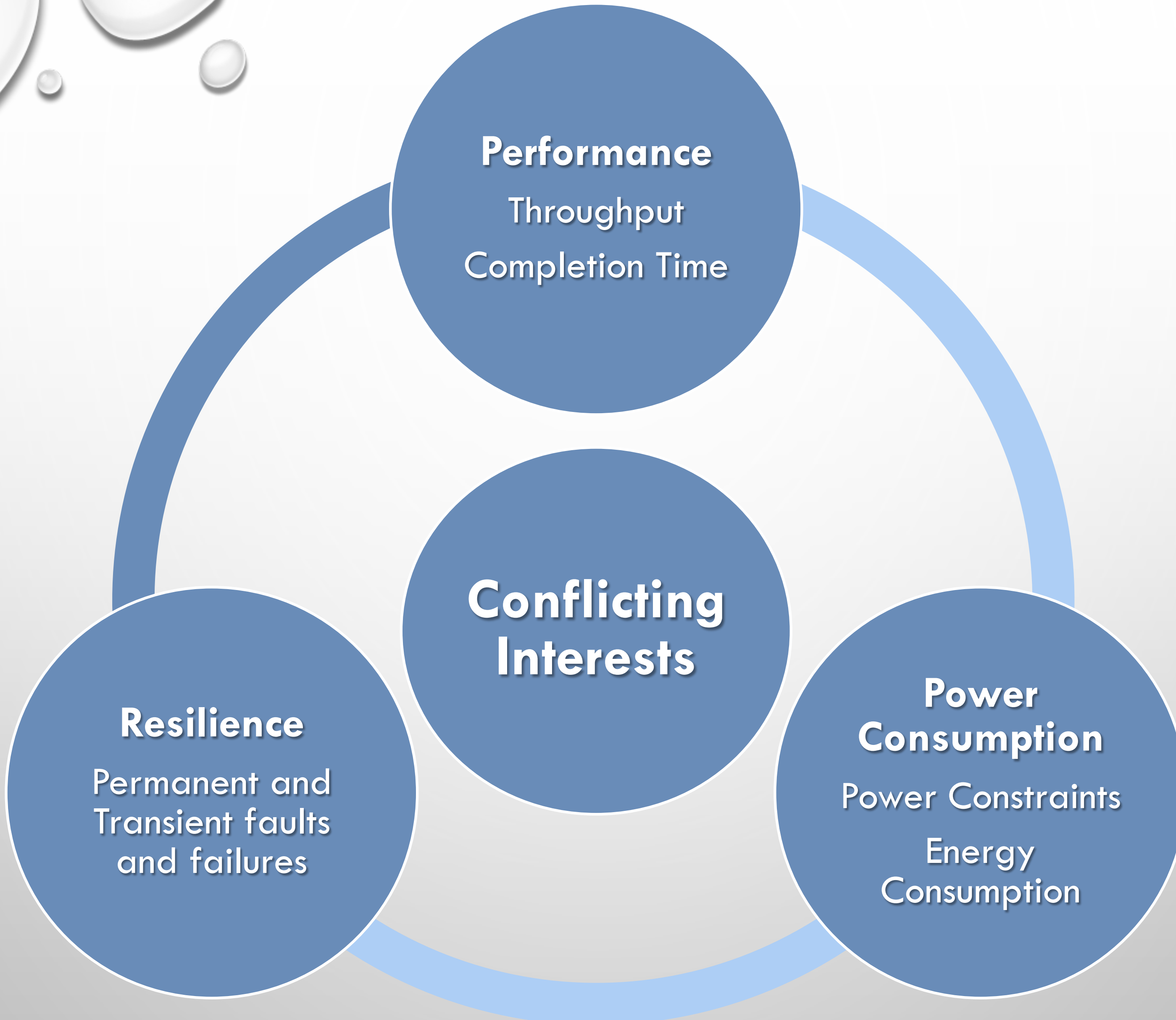
‡University of Pittsburgh

†Temple University

OUTLINE

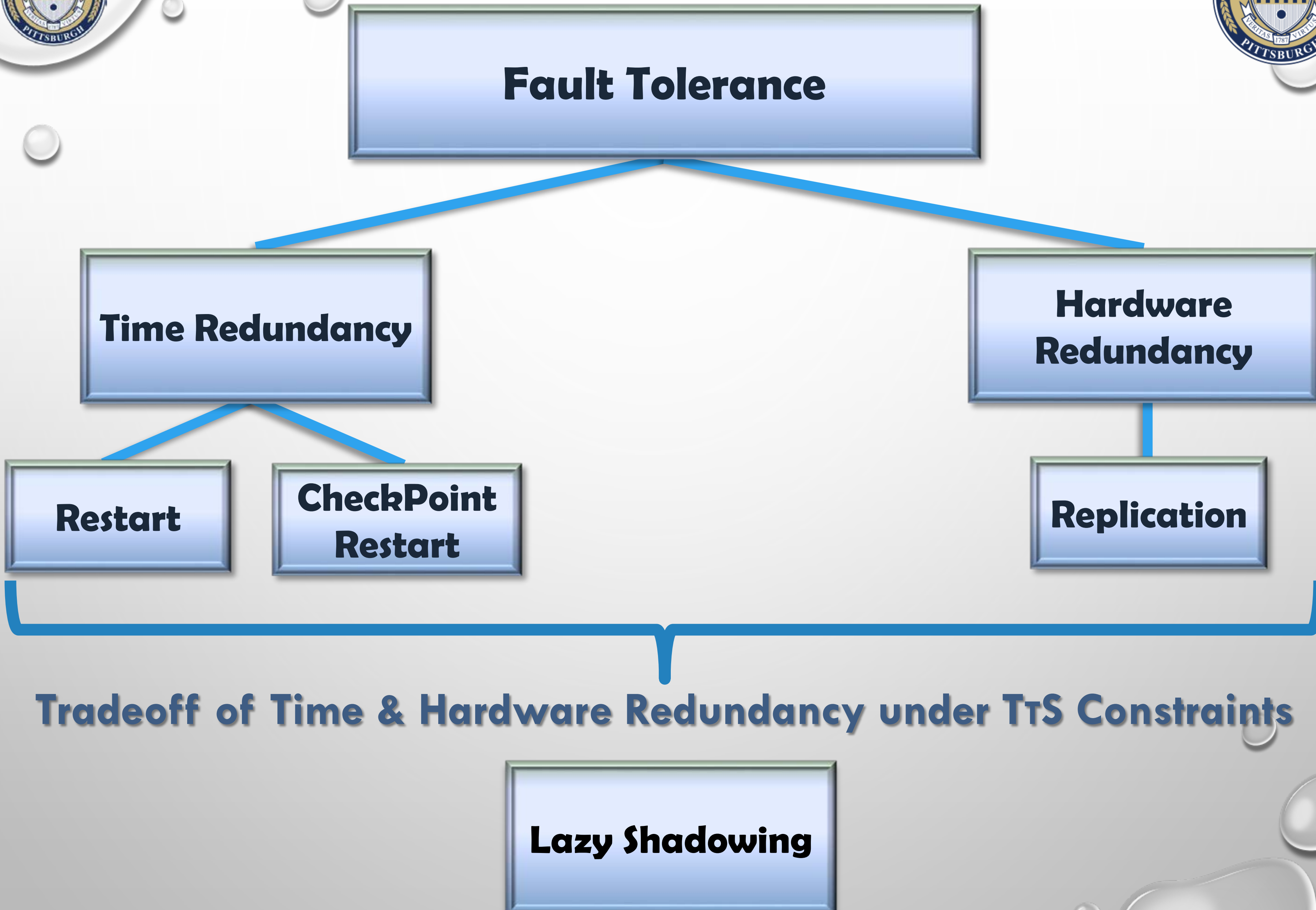
- Introduction and background
- Shadow Computing – Adaptive Framework for Fault Tolerance
  - Basic Model – Lazy Shadowing
  - Leaping for Forward Progress
    - Failure-induced and Forced Leaping
  - Rejuvenation to Mitigate Impact of Multiple Failures
- Conclusion and Future Work


PARALLELISM
Exploit the extreme levels of parallelism effectively
RESILIENCE
To both permanent and transient faults and failures
EXASCLALE CHALLENGES
ENERGY
To operate within affordable power budgets
I/O STORAGE
To access/store information at high capacities and with low latencies.



Objective: Optimize any combination of the three

Constraints: Bound any of the three


Fault Tolerance
Time Redundancy
Hardware Redundancy
Restart
CheckPoint Restart
Replication
Tradeoff of Time & Hardware Redundancy under TTS Constraints
Lazy Shadowing

LAZY SHADOWING – BASIC MODEL

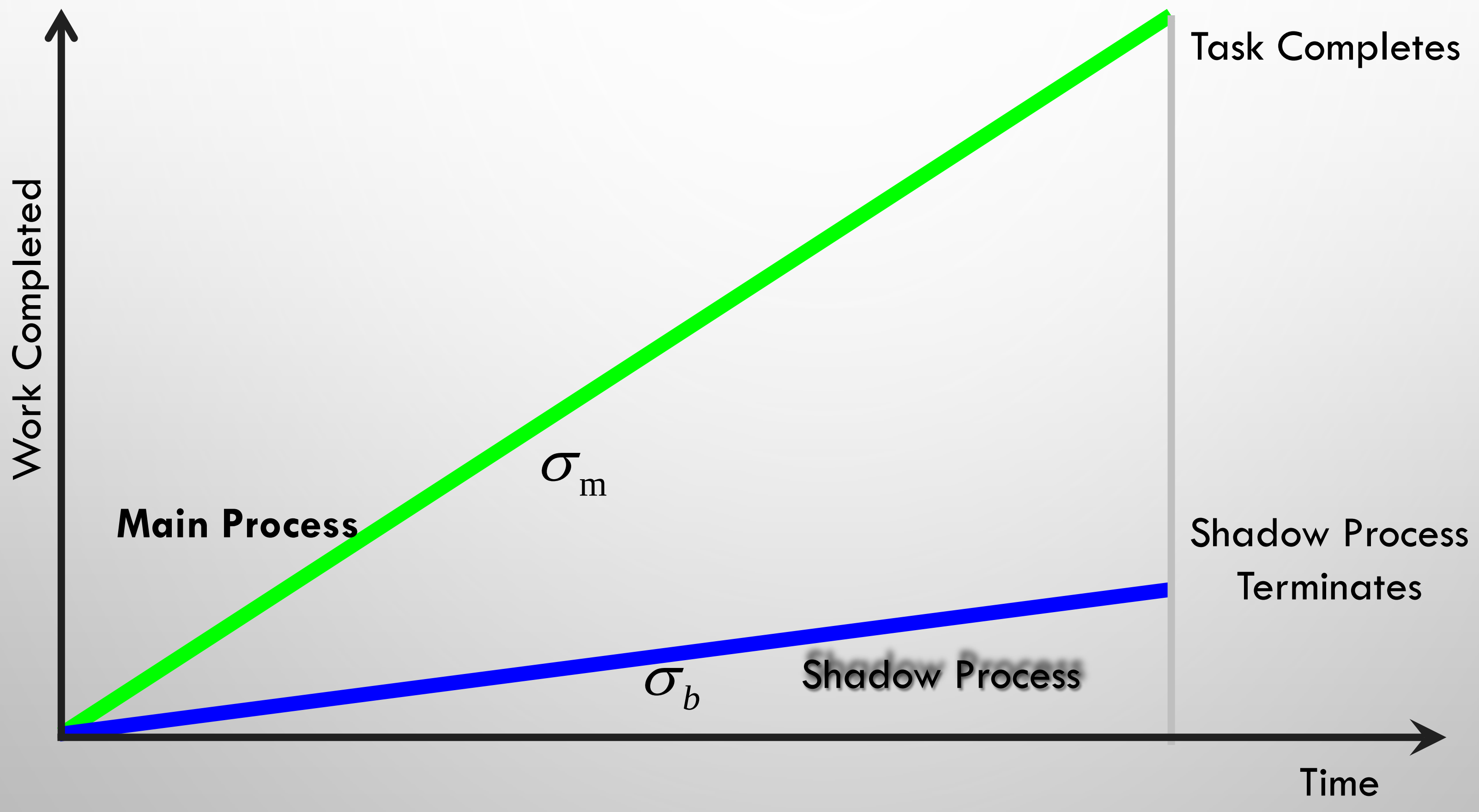
- Associate a “**shadow**” **process** with each **main** process
  - Shadows **run at lower execution rate** than associated mains to save energy

Execution Rate Control

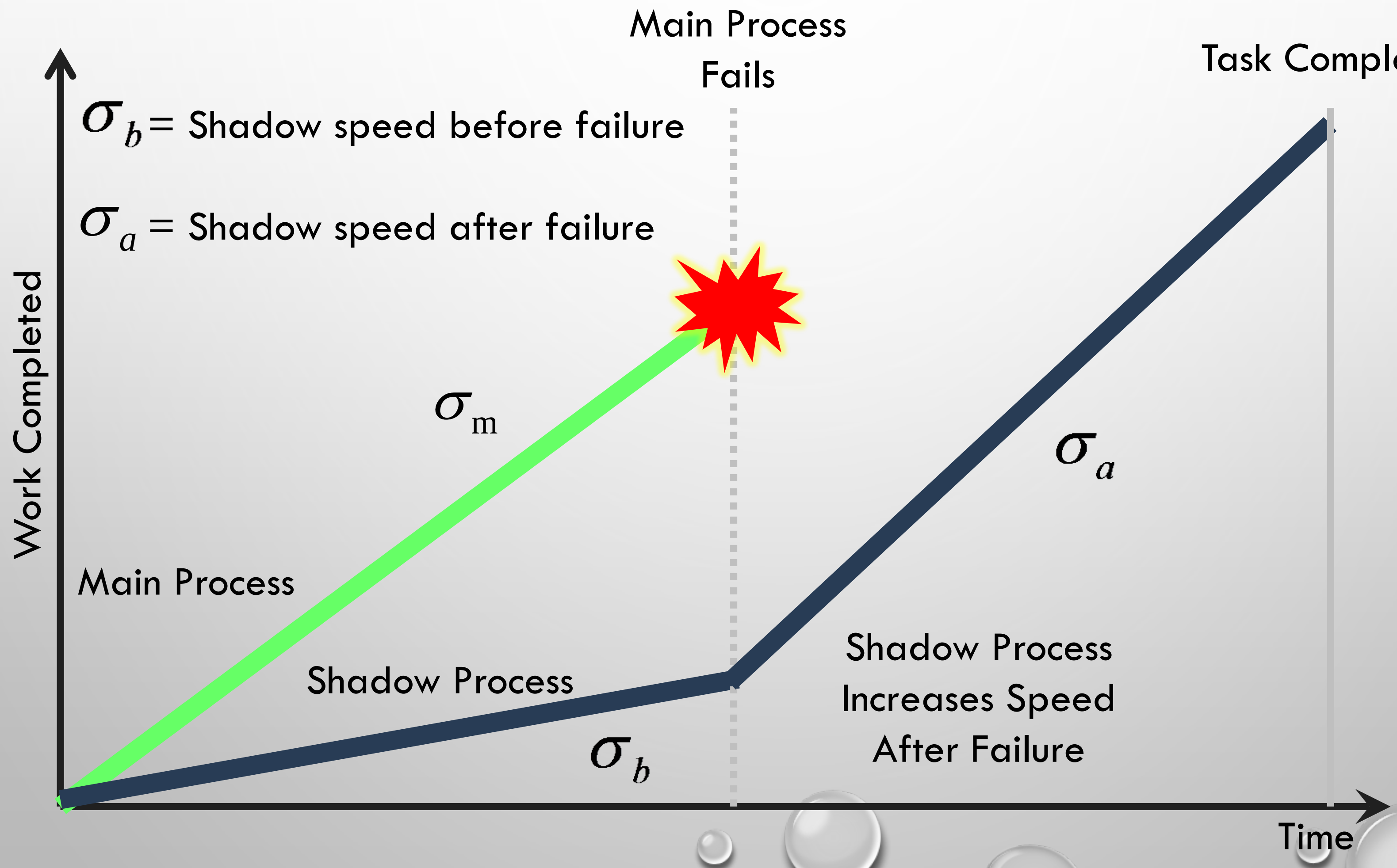
Dynamic Voltage and Frequency Scaling (DVFS)
To Save Dynamic Power

Process Collocation
To Save Static Power

LAZY SHADOWING
SUCCESSFUL COMPLETION

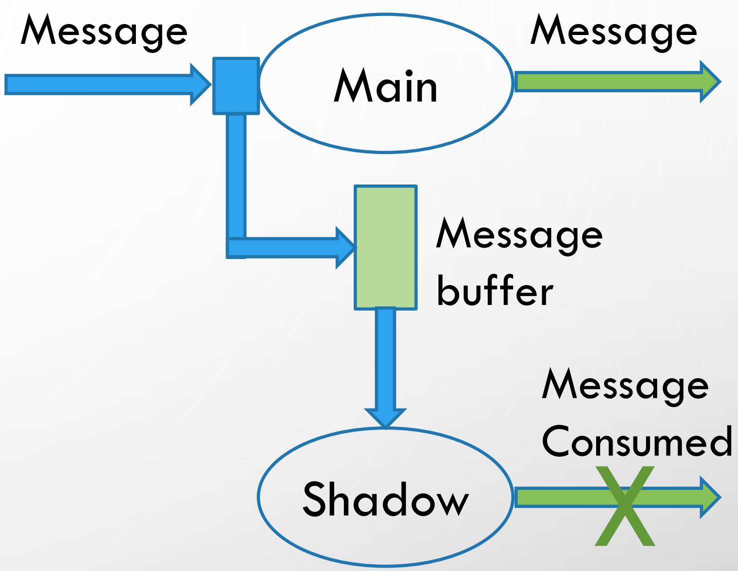


LAZY SHADOWING WITH FAILURE

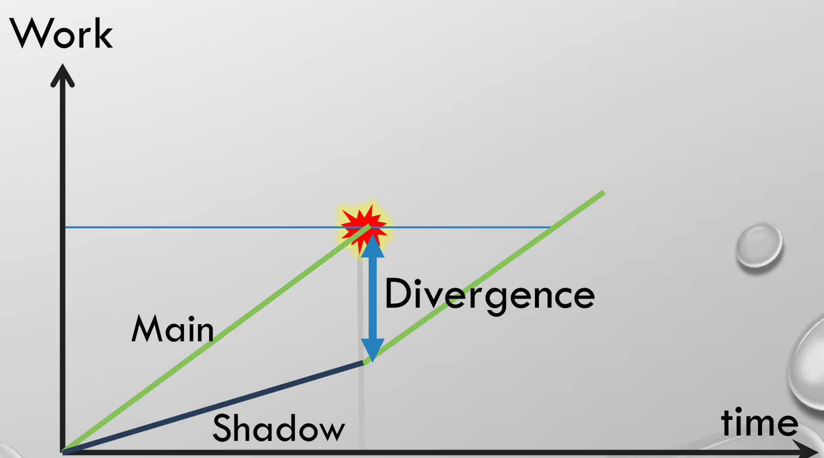


LAZY SHADOWING ANALYSIS

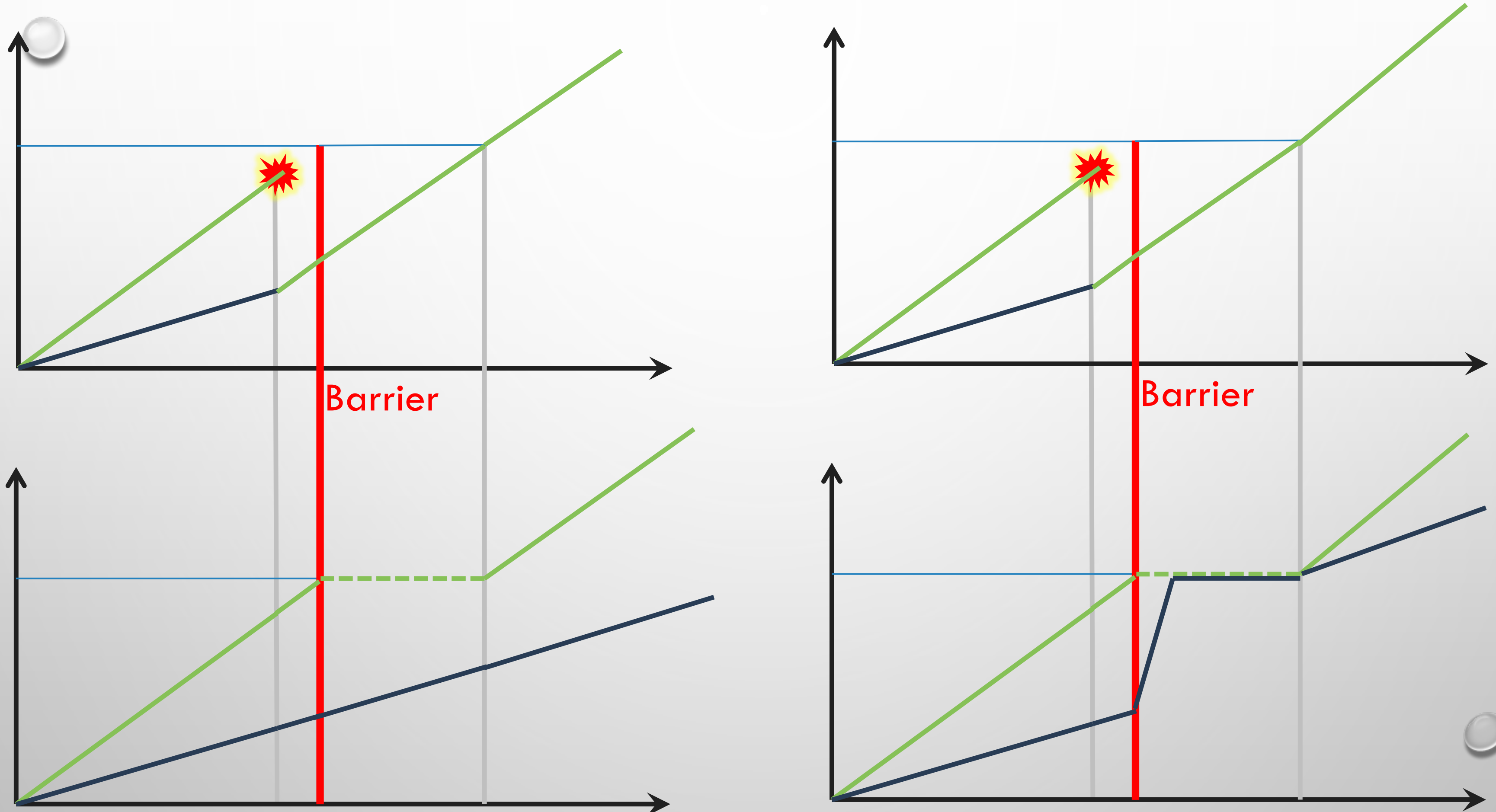
1. Size of message buffer grows with shadow/main divergence.



2. Time to recover from failure grows with divergence



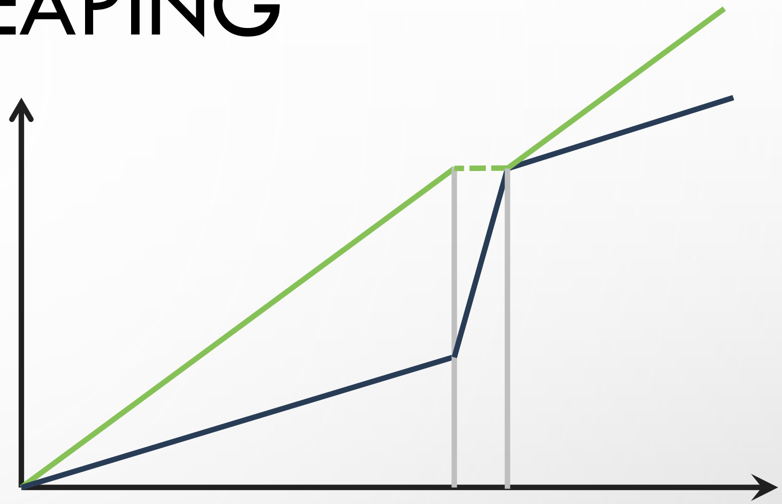
LEAPING – TO REDUCE DIVERGENCE



Turning a Foe into a Friend – While shadow of failed main recovers, all other shadows leap forward to synchronize their states with their mains

FORCED LEAPING

- When divergence between the main and its associated shadow reaches a threshold, "forced leaping" is invoked

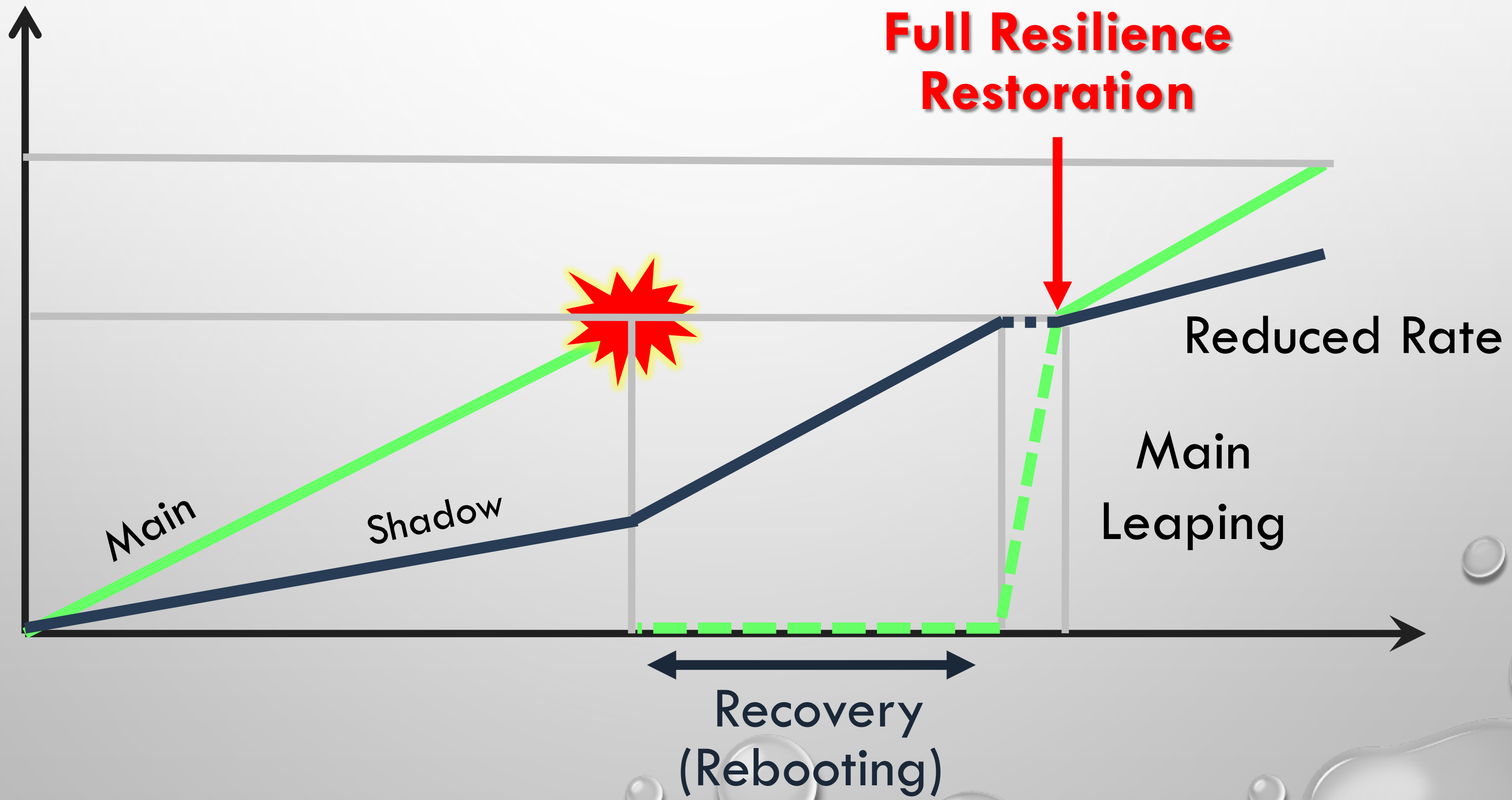


- Leaping, forced or failure-induced, entails forward rolling to synchronize the shadow's and main's state
  - **Process migration techniques are applicable, with reduced overhead**
- Shadow leaping is applicable, regardless of the mechanism used to control execution rate
  - **DVFS or Collocation**

DEALING WITH MULTIPLE FAILURES

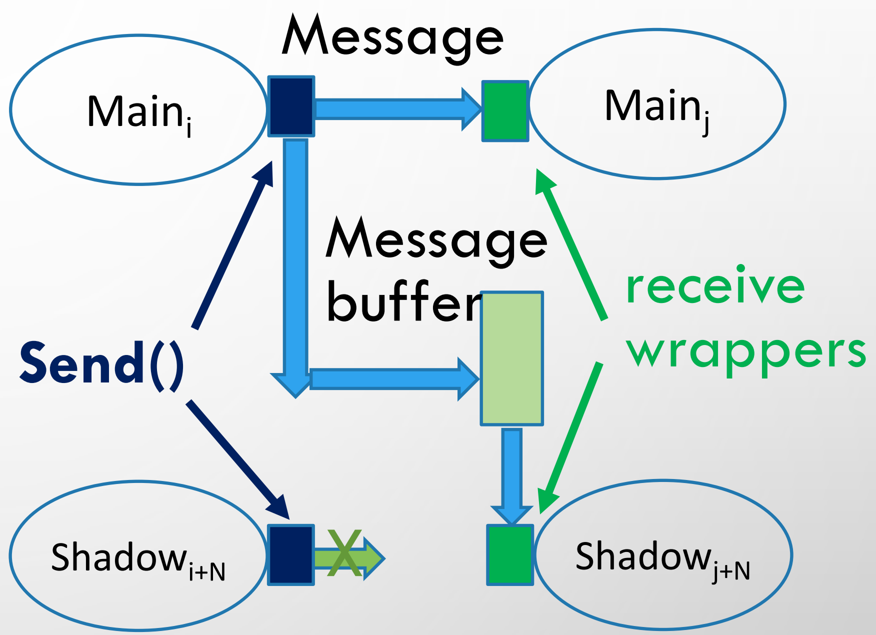
- **Shortcoming** – Vulnerability of Lazy Shadowing increases with the number of failures
  - Failures of a main and its shadow, for example, causes failure of the entire system
- How to mitigate impact of **multiple failures**?
  - **Shadow Suite** – Associate more than one shadow, **executing at decreasing rates**, with each main, based on application **criticality**
    - Too expensive
    - Leads to higher divergence between main and its shadows
  - **Rejuvenation** upon failure
    - Restoring main full state, upon shadow leaping

REJUVENATION PROCESS



SHADOW INTEGRATION INTO MPI – CALL WRAPPING

- SEND()@MAIN – REPLICATES MESSAGE TO ASSOCIATED SHADOWS
- SEND@SHADOW: SUPPRESSES MESSAGE
- RECEIVE()@MAIN – UNCHANGED
- RECEIVE()@SHADOW – MODIFIES SENDER'S RANK

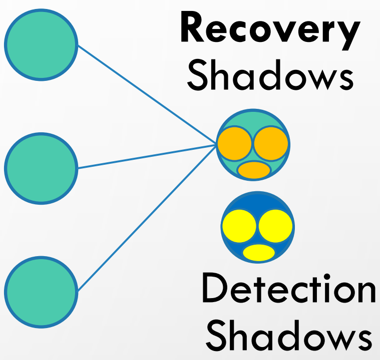


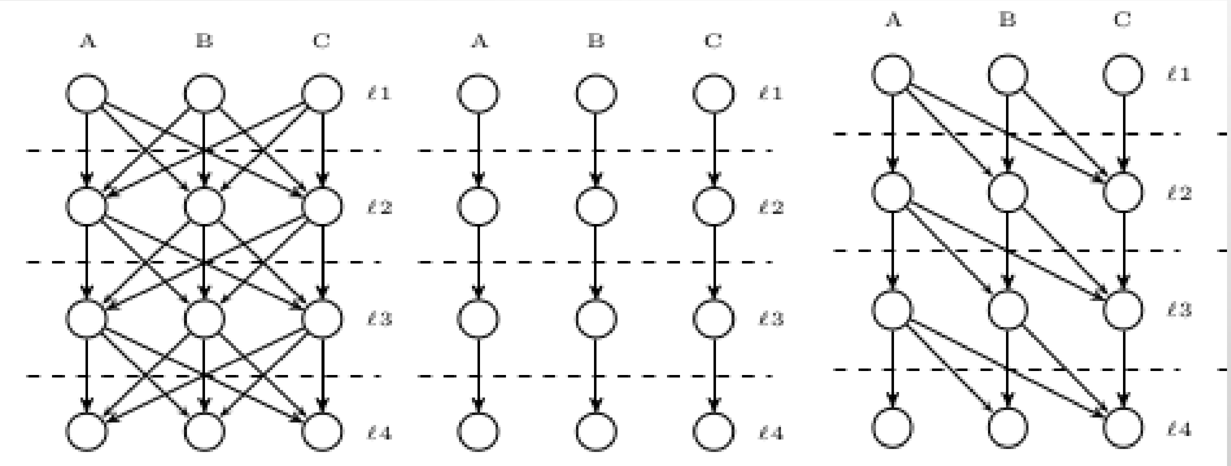
- Control messages may be required to ensure deterministic execution
  - MPI any-source call
- Leaping – User registers state to be transferred
  - Similar to user-level checkpointing

LAZY SHADOWING – SLICING FOR FAULT DETECTION

A **workload slice** is a set of programs statements that may affect the values at some point of interest

- Construct **sliced shadows** that computes only subsets of the state variables
- Compare the results of slices with mains for correctness





(a) Strongly connected (b) Not connected (c) Cascade connected

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

Conclusions

- Lazy Shadowing is a hybrid model that harnesses hardware and time redundancy, to optimize TTC under transient and permanent failures of space and time redundancies
  - Trades off resilience, performance and power/energy
  - Converges to space or time redundancy, to closely meet the workload TTS and resiliency requirements
  - Can be implemented using DVFS or Colocation
- Leaping allows shadow to roll-forward and elimination of main/shadow computational divergence
- Rejuvenation mitigates impact of multiple failures
  - Restore system full system resilience upon multiple failures
- Early results demonstrate efficiency at extreme scale
- Future Work
  - "Harden" implementation of Rejuvenation and Slicing in MPI
  - Testing and Analysis
  - Application of Shadow Computing to Data-intensive HPC Applications **– "Burning the candle from both sides"**