A Framework for Proactive Fault Tolerance

Geoffroy Vallee (ORNL)

Kulathep Charoenpornwattana (LATech)

Christian Engelmann (ORNL)

Anand Tikotekar (ORNL)

Chokchai "Box" Leangsuksun (LATech)

Thomas Naughton (ORNL)

Stephen L. Scott (ORNL)

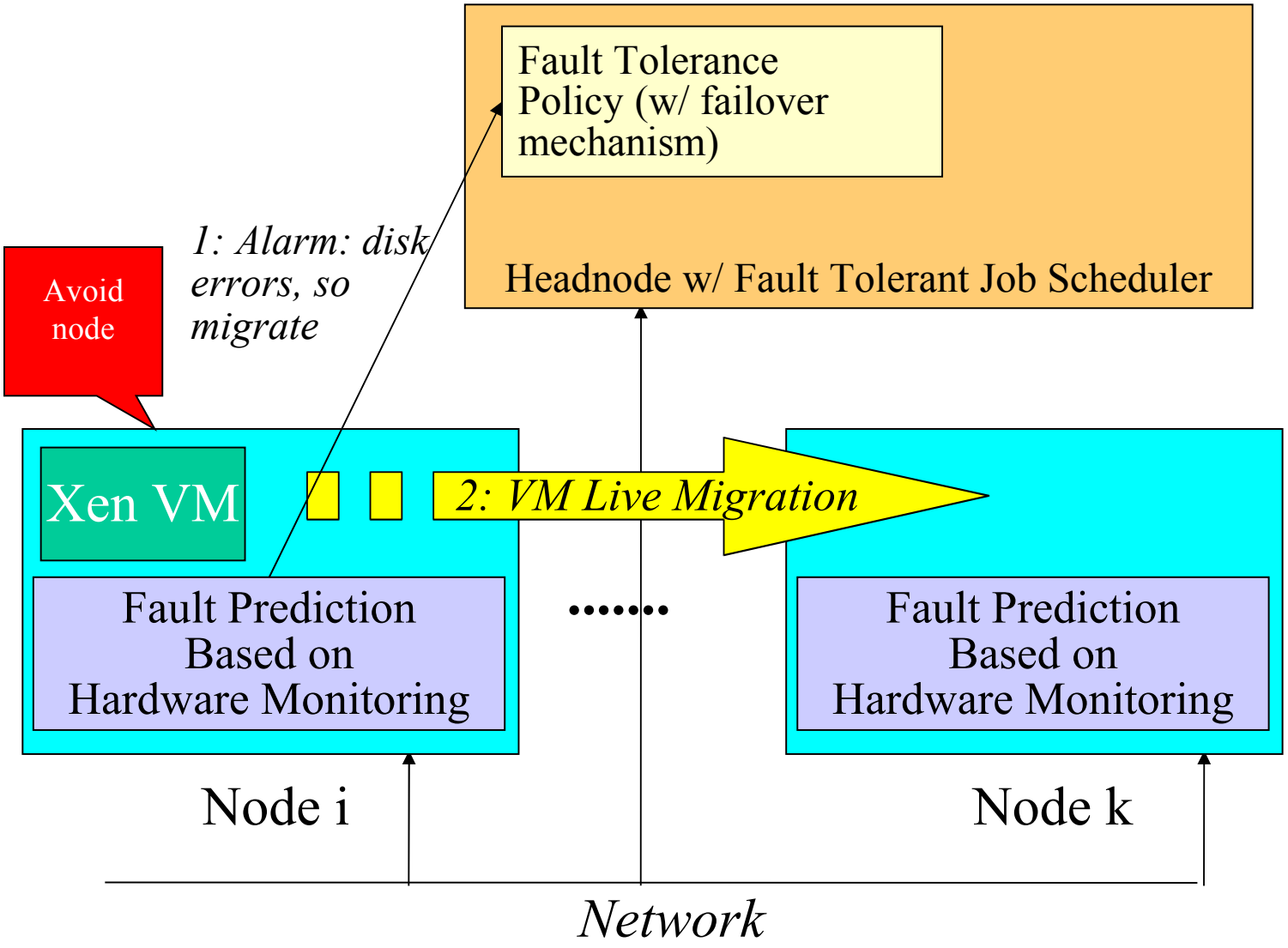
OAK RIDGE National Laboratory

Context & Background

- **Large-scale systems & long running applications**
  - hundred of thousands of nodes, individual components can fail
  - specialized nodes (compute nodes vs. I/O nodes vs. login nodes)
  - avoid any kind of overhead on compute nodes (priority to applications)
  - Standard parallel applications (MPI-like applications)
- **No Fault Tolerance (FT) intelligence in most parallel applications**
- **Basic fault tolerance solutions**
  - Production: *reactive policies*, *i.e.*, how to react to a failure?
  - Research: *pro-active policies*, *i.e.*, how to anticipate failures?
- **Different execution platform characteristics**
  - Failure distribution
  - Predictable vs. unpredictable failures
  - Platform types: disk-less or disk-full

Only pro-active FT is in the scope of this presentation

Pro-active Fault Tolerance – Introduction



Pro-active Fault Tolerance Challenges

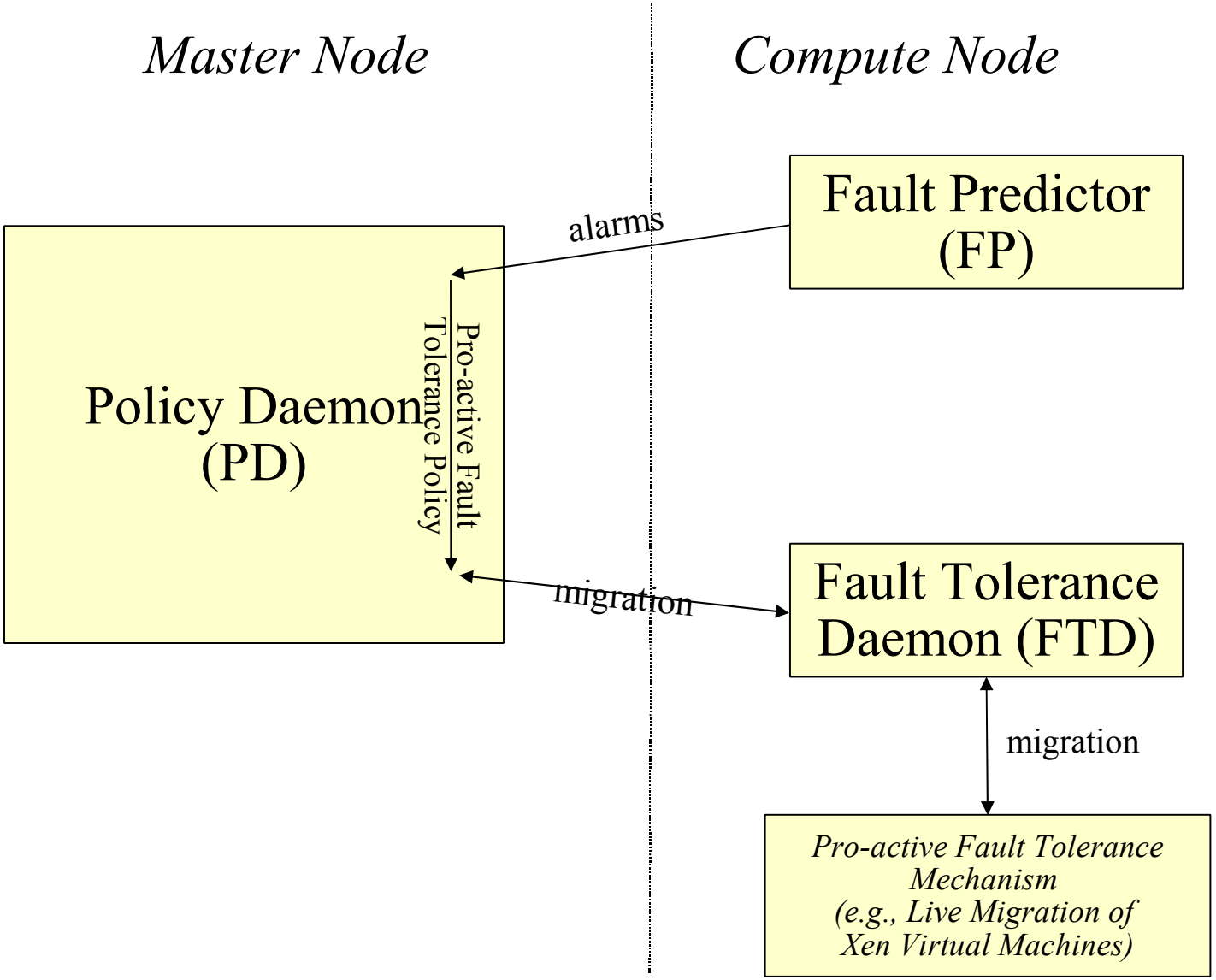
- Mechanisms challenges
  - fault prediction
  - prediction accuracy
  - application manipulation
    - migration
    - pause/unpause
- Policy challenges – adaptation to
  - platform characteristics
  - application characteristics

No one-fit all solution
=> proactive FT framework

Platform Architecture Overview

- **Specialized nodes**
  - **“master node”**
    - ***logical* centralized execution point for services**
    - **may NOT be a single node, it is a logical view of where the distributed services are hosted**
  - **compute nodes**
    - **where the application is running**
    - **should avoid interferences from the framework**
- **Communication sub-system**
    - **for scalability, we assume we reuse scalable communication sub-systems (*e.g.*, MRNet)**
    - **efficient way to “push” data to the master node**
    - **abstraction of the under-lying networking solutions**

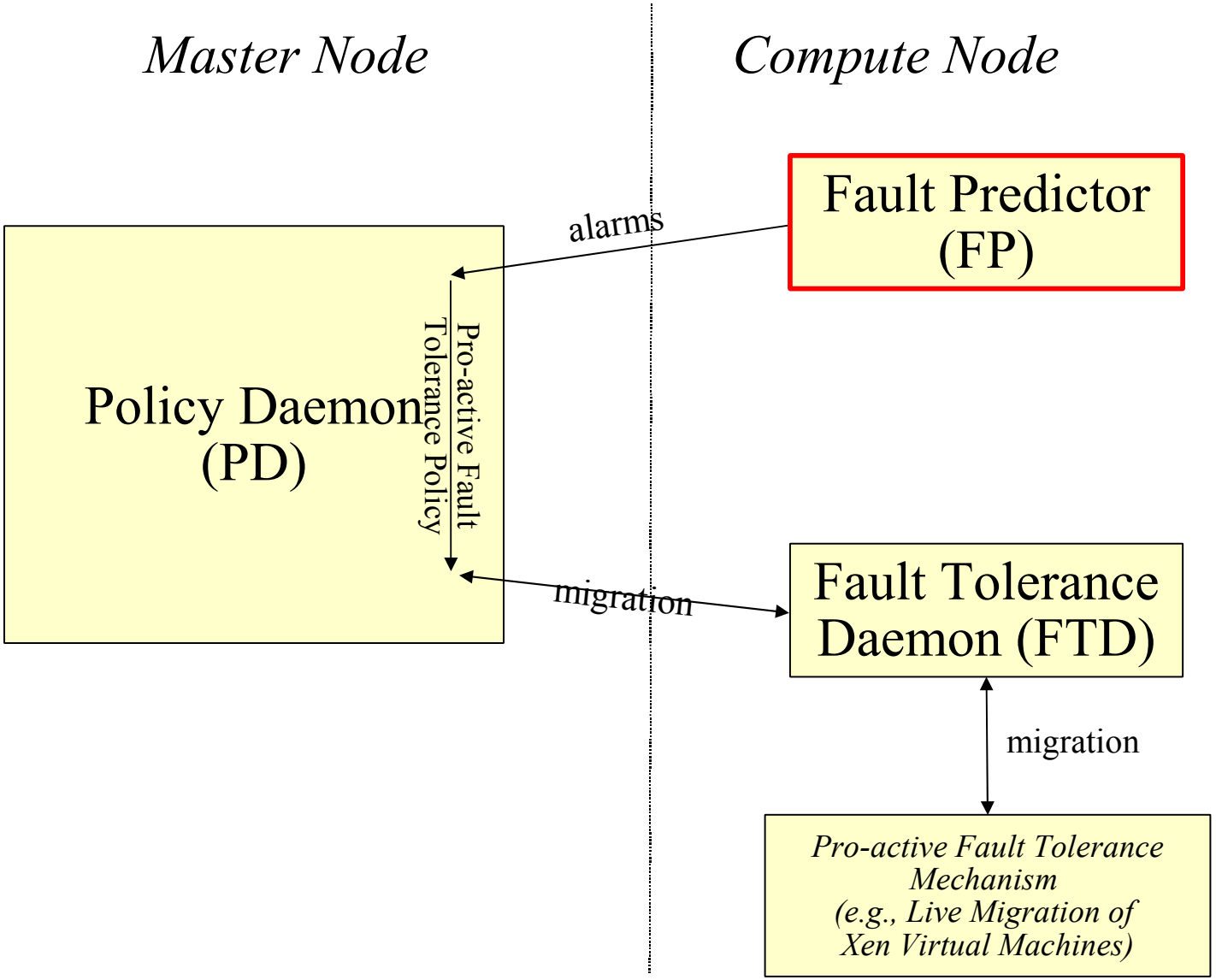
Pro-active FT Framework – Architecture



Framework Components – Event System

- Core of the framework: abstract all communications between framework components
- Abstract the underlying communication sub-system
  - abstraction of scalable sub-systems such as MRNet
  - abstraction of the physical network solution
- Based on the concepts of *mailbox*, *mailbox managers*, *subscribers*, and *publishers*
- Asynchronous, “tolerate failures” (*i.e.*, missing readers)
- Very low overhead when the system is healthy
- No interference with applications running on compute nodes

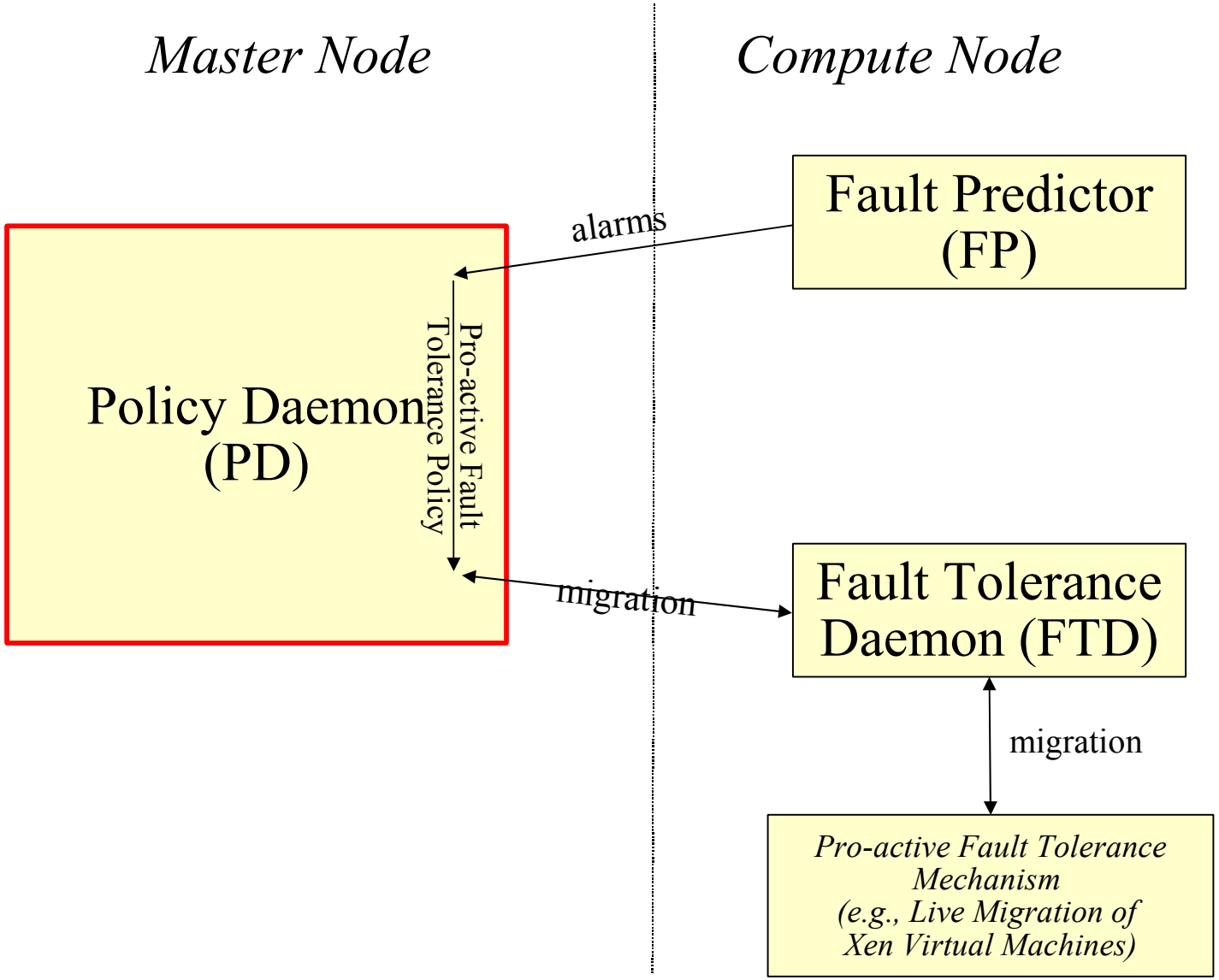
Pro-active FT Framework – Architecture



Framework Components – Fault Predictor

- Runs on each compute nodes
- Abstraction of the underlying mechanism for hardware monitoring and fault prediction (typically hardware probes)
- Filter data extracted from probes
- Prevent a global polling, creates an alarm only if probes report abnormal behavior (alarm sent to the policy daemon on the master node)
- Currently uses: lm-sensor, syslogs + experimental support of IPMI

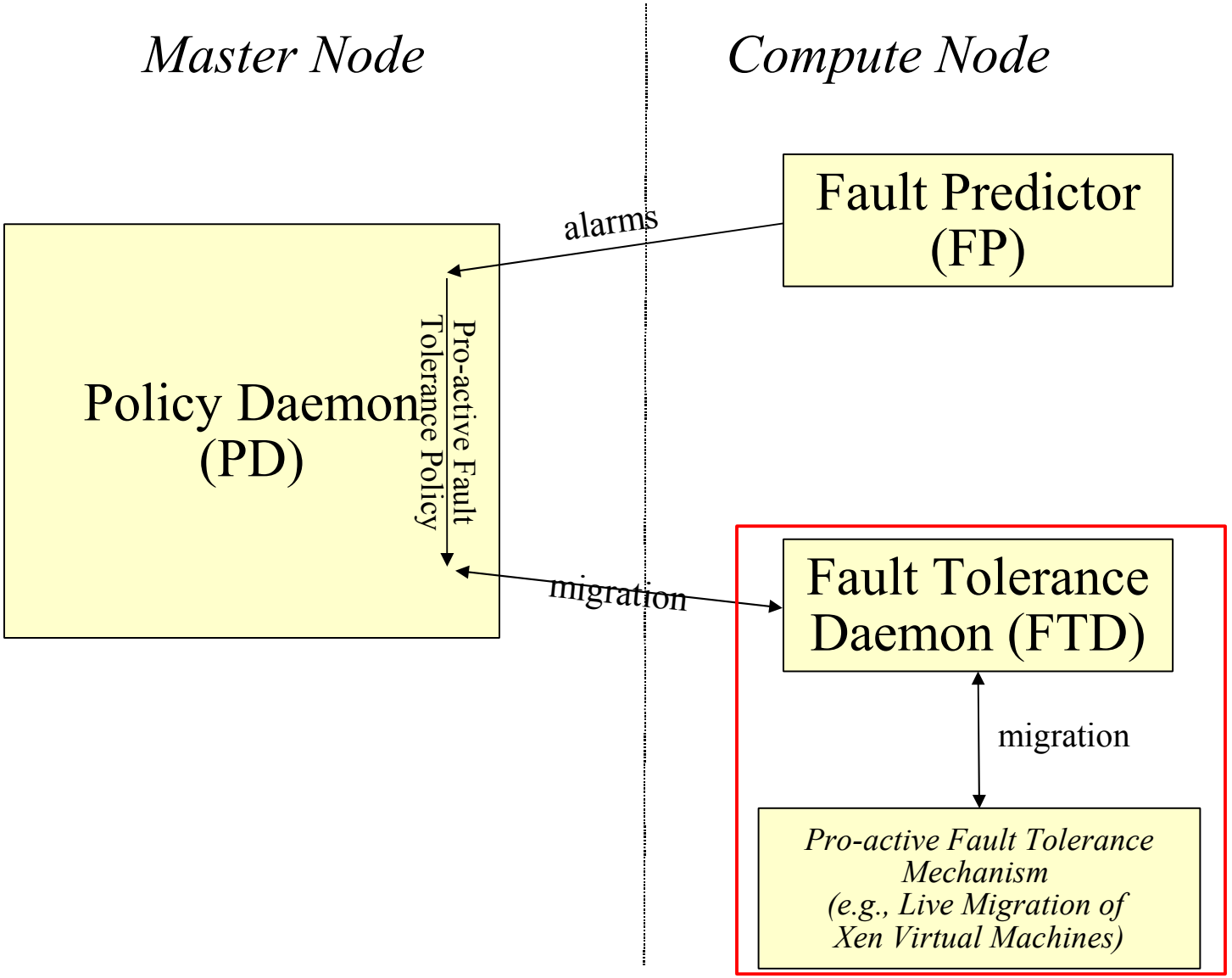
Pro-active FT Framework – Architecture



Framework Components – Policy Daemon

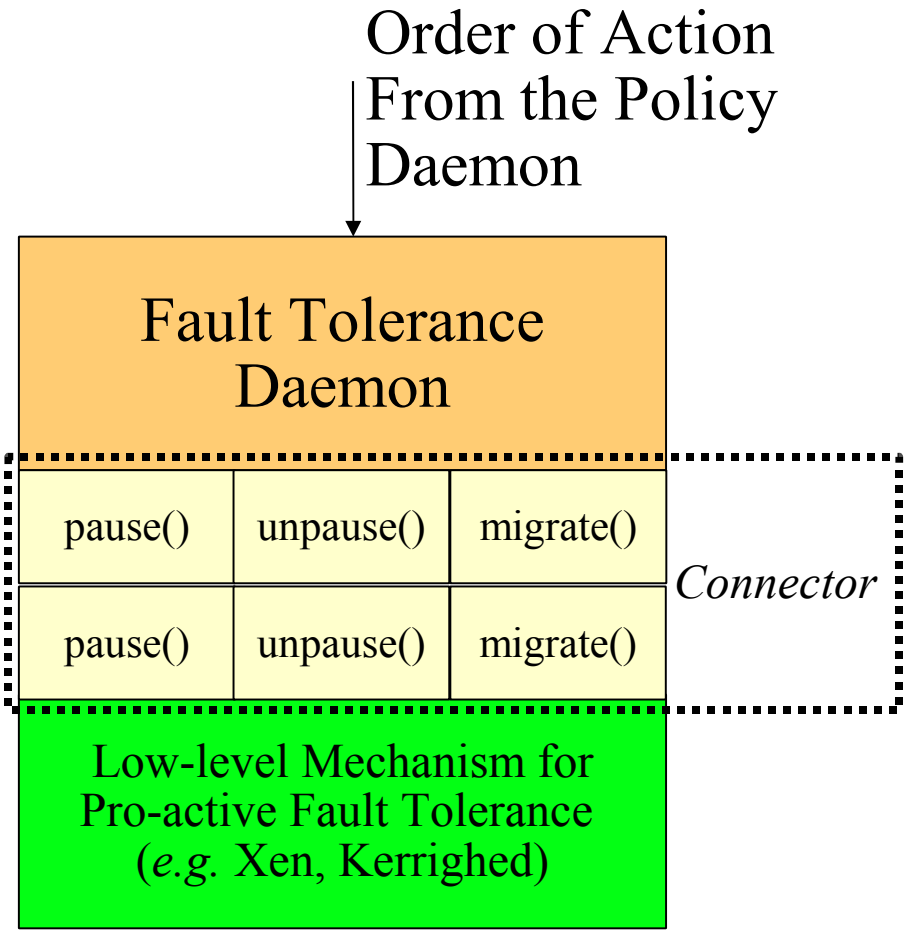
- Implement the proactive FT policy
- Running on the master node
- Receive and analyze alarms sent from *fault predictors*
- If needed, sends an alarm for migration or pause to the compute node

Pro-active FT Framework – Architecture



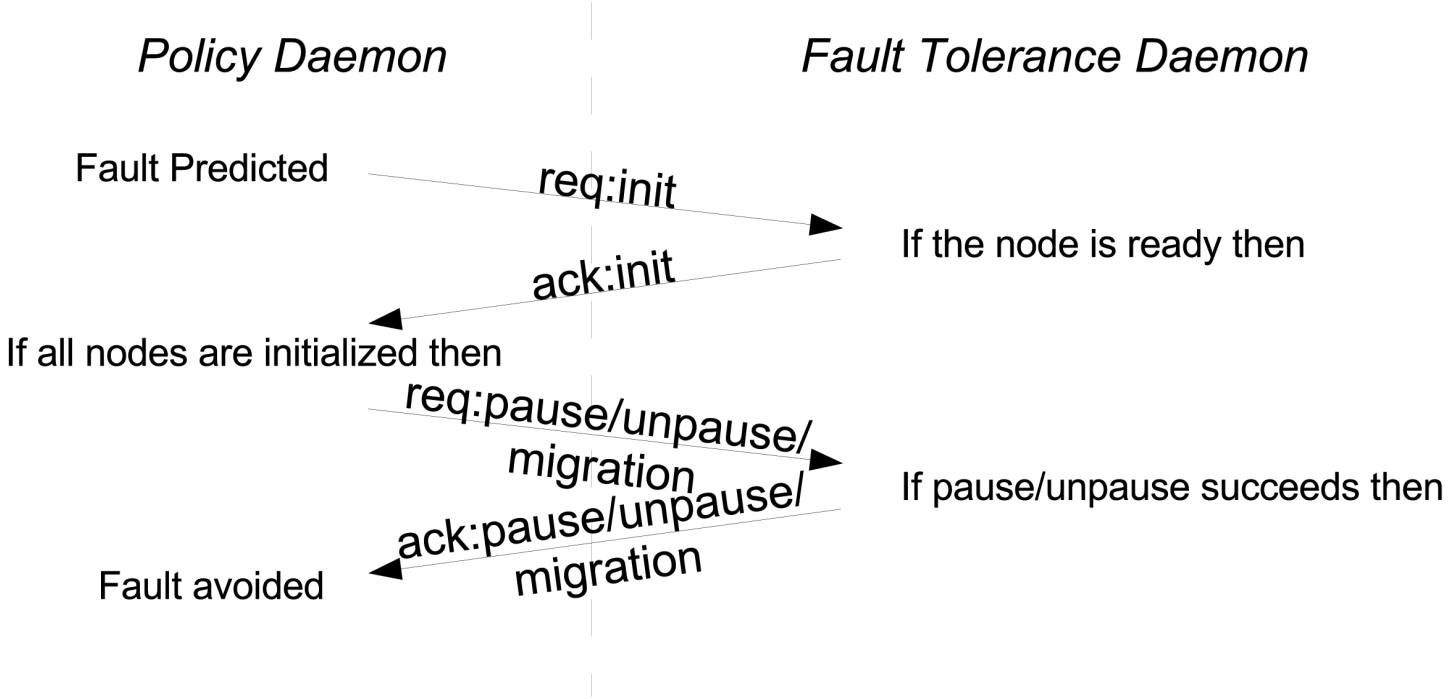
Framework Components – Fault Tolerance Daemon

- Running on the compute nodes
- Abstract the underlying mechanism for migration & pause/unpause (concept of connector)
  - similar to plug-ins
- Receive alarms from policy daemon for migration or pause

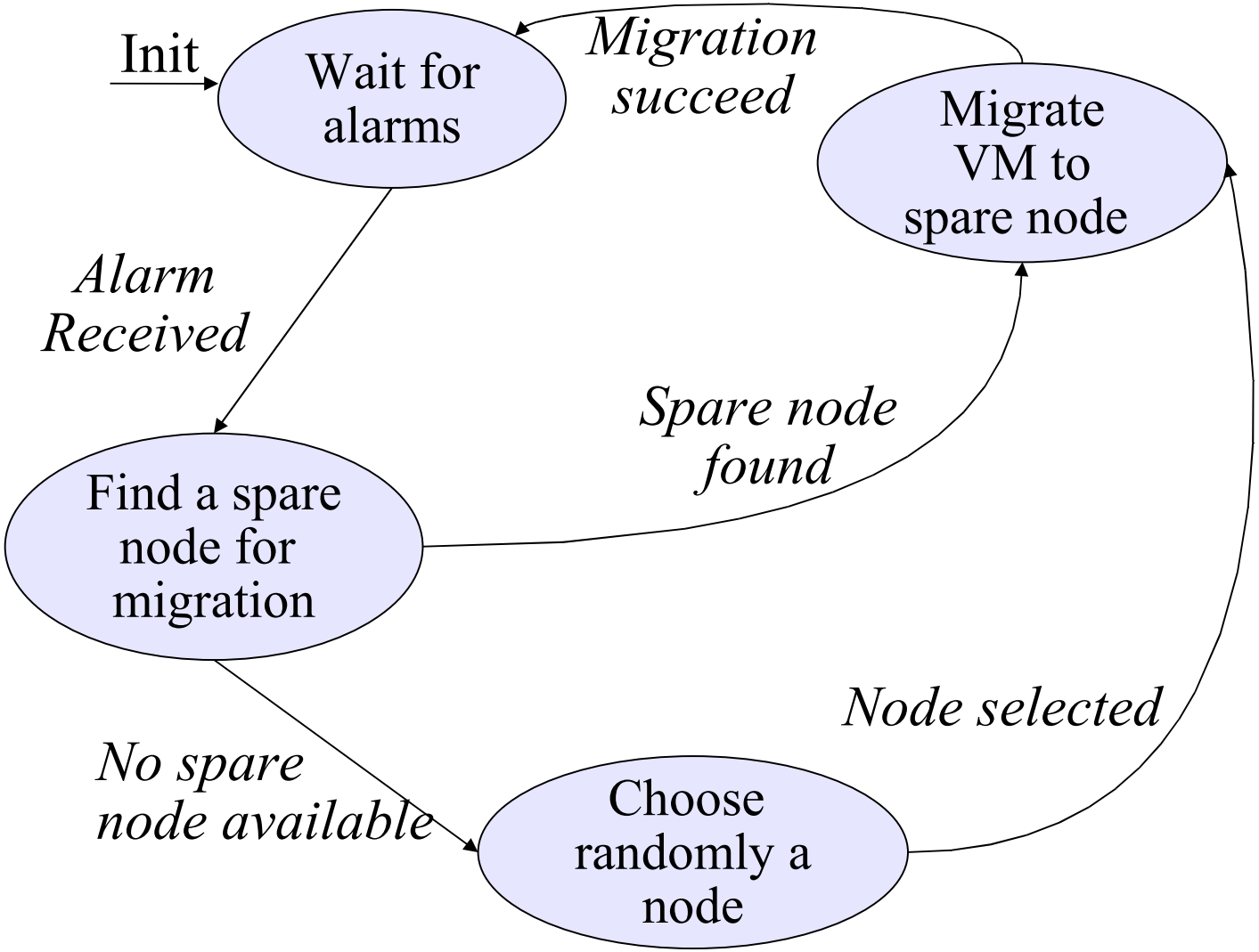


Pro-active FT Framework – Protocol

- **Goal**
  - **guarantee pro-active FT**
  - **detect failures: avoid conflicts between reactive/proactive FT**



Pro-active FT Policy – Example



- ***PS: policy used for evaluation***

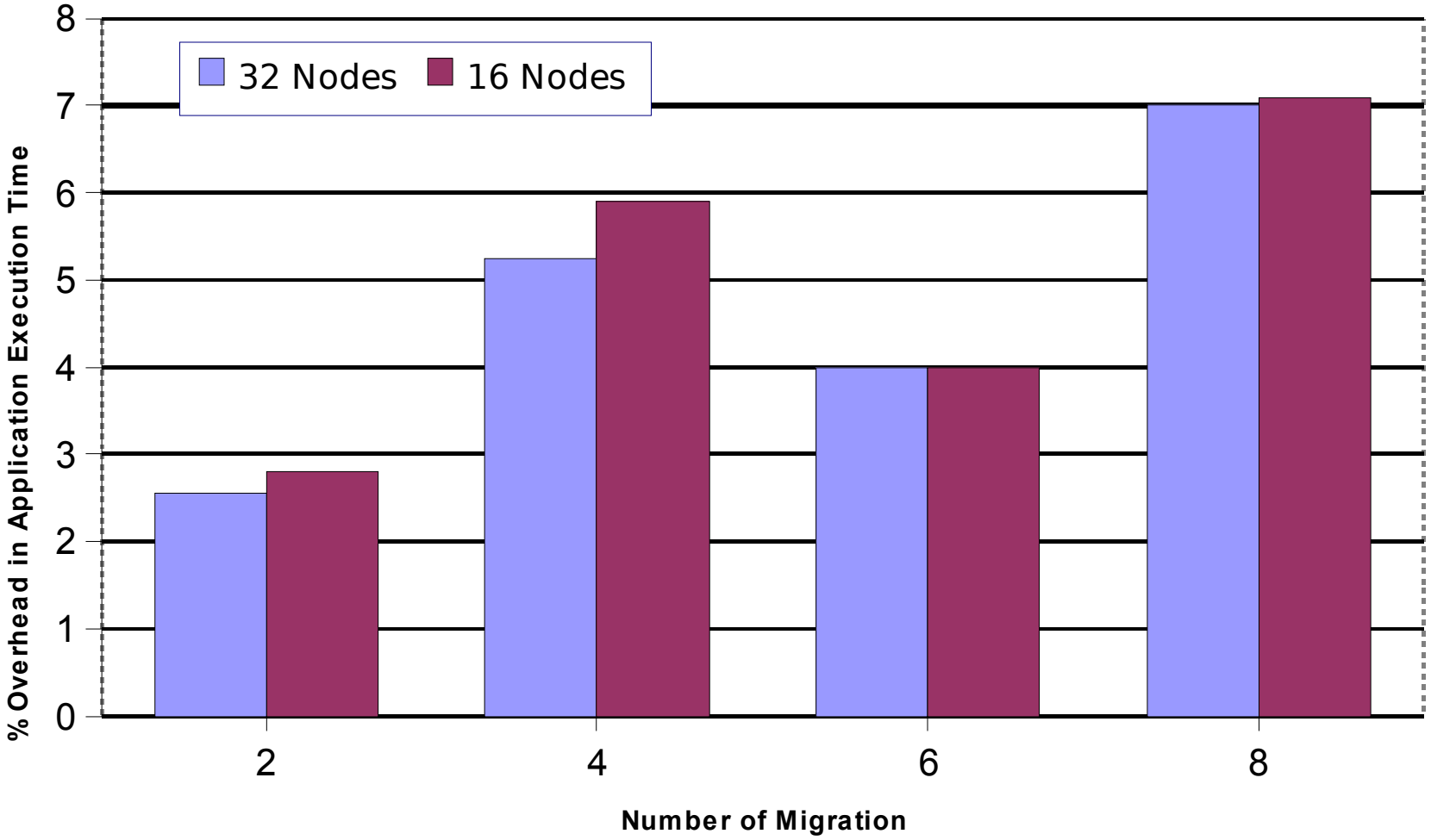
Experimentation Protocol

- 2 sets of experimentations: 16 & 32 nodes
- HPCC benchmark
- We argue that
  - the implementation of multiple policies *cannot* validate the framework (no reference)
  - we can use our simulator as reference
- Policy presented in slide 15
  - users can take benefit of a pool of spare nodes
  - if a alarm is received, we migrate the VM away from the faulty node
    - using a spare node if any available
    - stacking VMs on a random node if no spare node available

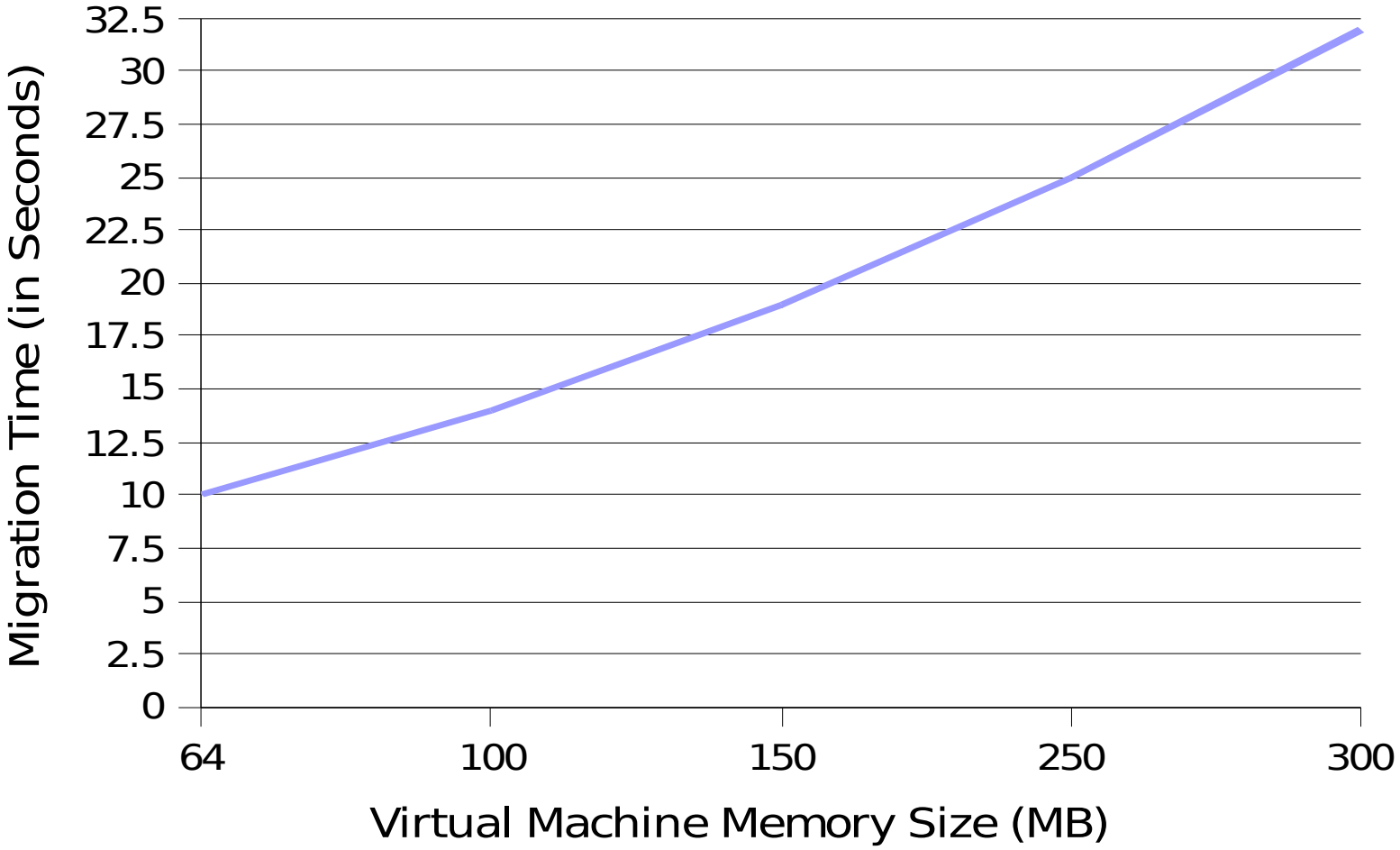
Preliminary Experimentation & Validation

- Comparison w/ our FT simulator
- Experimentation platform
  - based on Xen 3.0.2
  - 40 PIII nodes: HostOS has 200MB of memory; VMs 250 MB
- Simulator characteristics
  - Cluster'07 paper [tiketekar]
  - based on LLNL ASCI White System logs
  - specification of many platform parameters: migration overhead, platform characteristics and so on
  - specify our physical platform characteristics

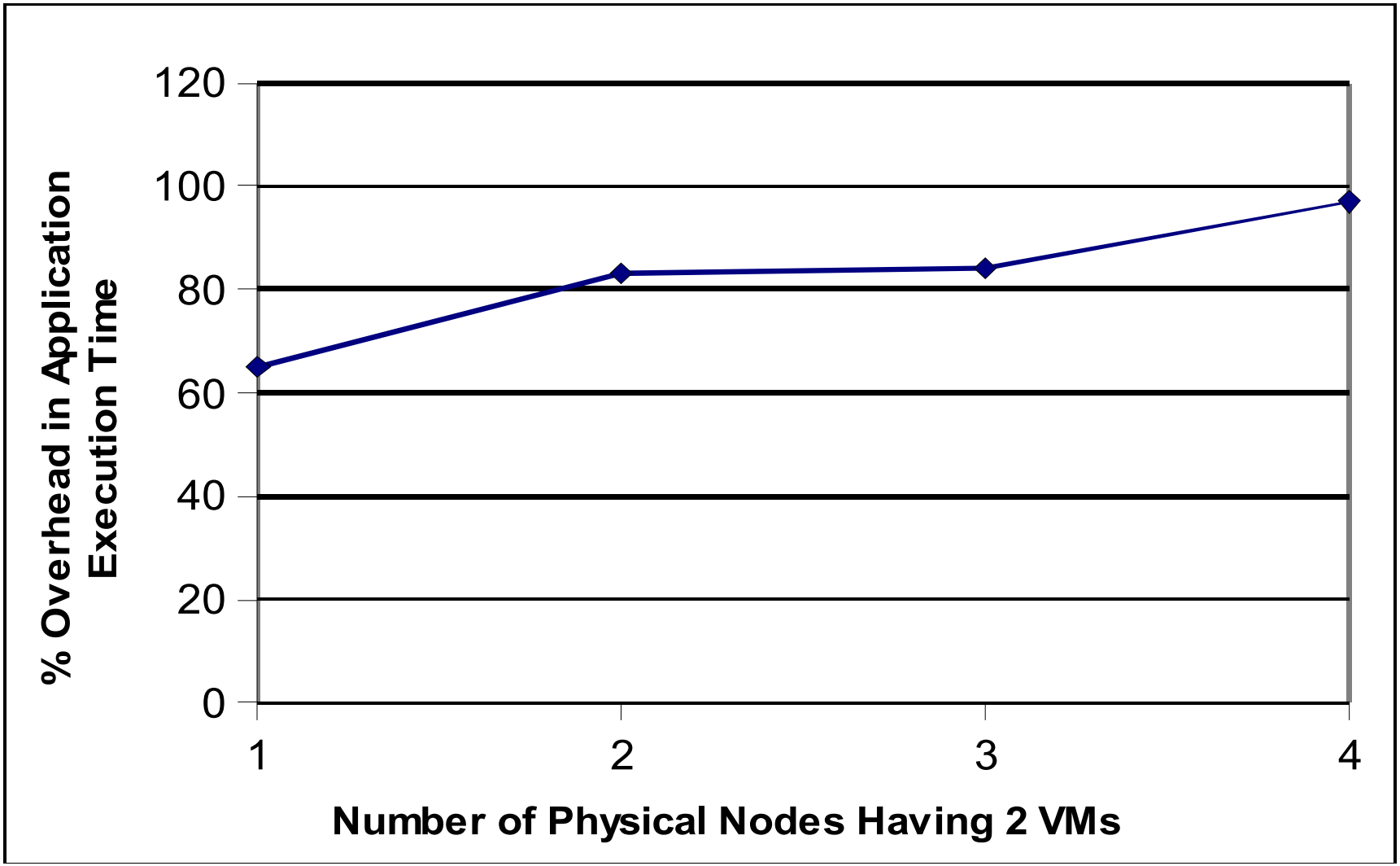
Migration Overhead Evaluation



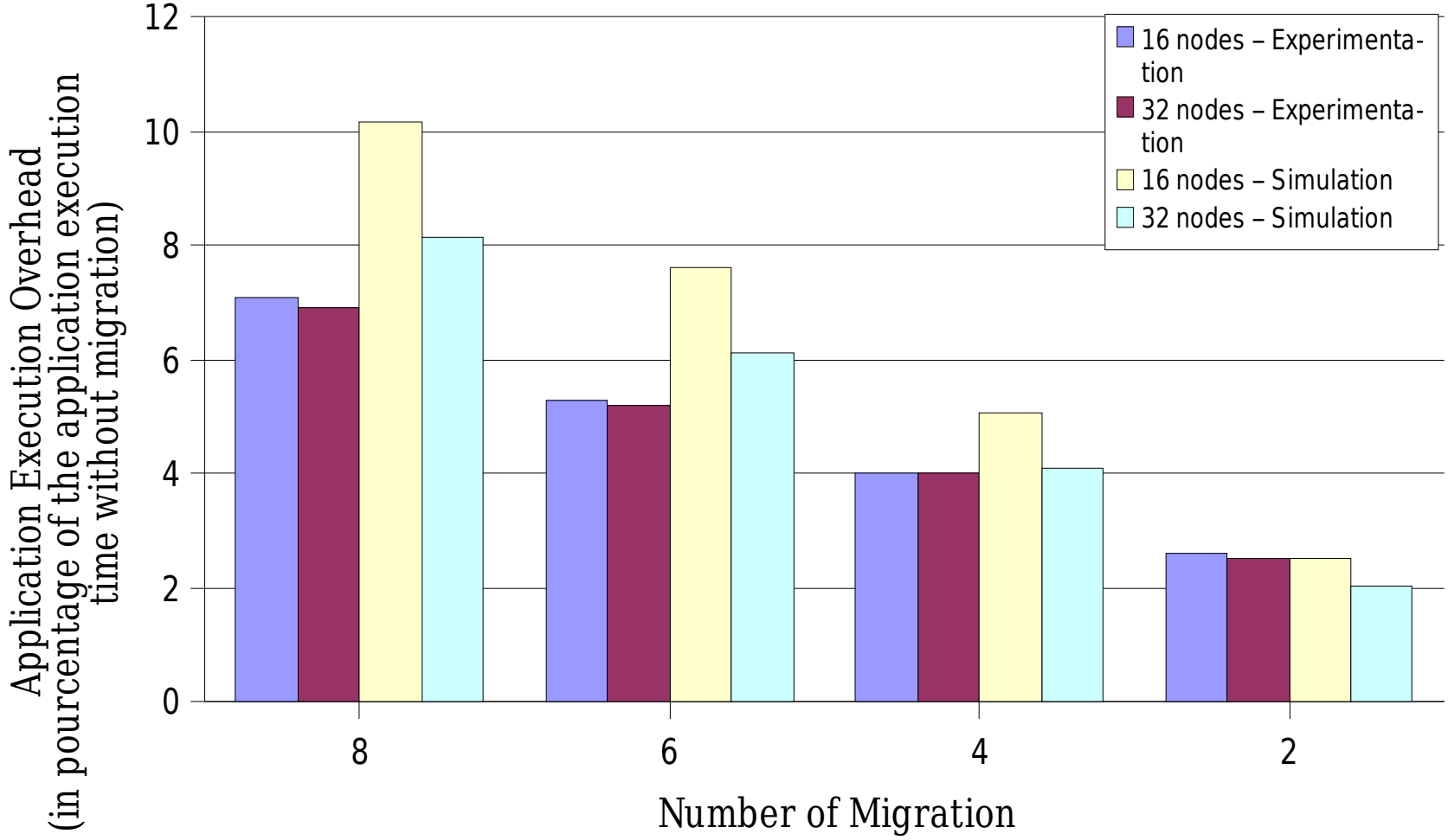
Impact of VM Memory Footprint on VM Migration



VM Stacking Effect



Simulation vs. Experimentation



Conclusion & Future Work

- Proactive FT framework
  - ease the implementation of new pro-active FT policies
  - capable of supporting many different low-level mechanisms
    - virtual machine migration & pause/unpause
    - process-level migration & pause/unpause
  - easily extensible
- Future work
  - reactive FT support
  - integration with scalable communication sub-system
    - Scalable Tool Communication Infrastructure (STCI)

Questions?