nuKSM: NUMA-aware Memory De-duplication on Multi-socket Servers

Akash Panda, Ashish Panwar, Arkaprava Basu

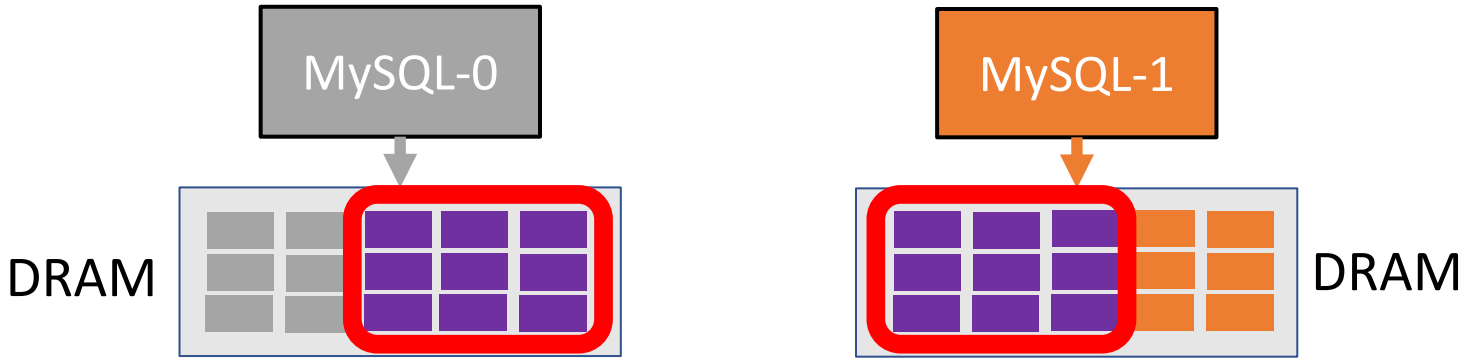
Department of Computer Science and Automation
Indian Institute of Science

https://csl.csa.iisc.ac.in

Executive summary

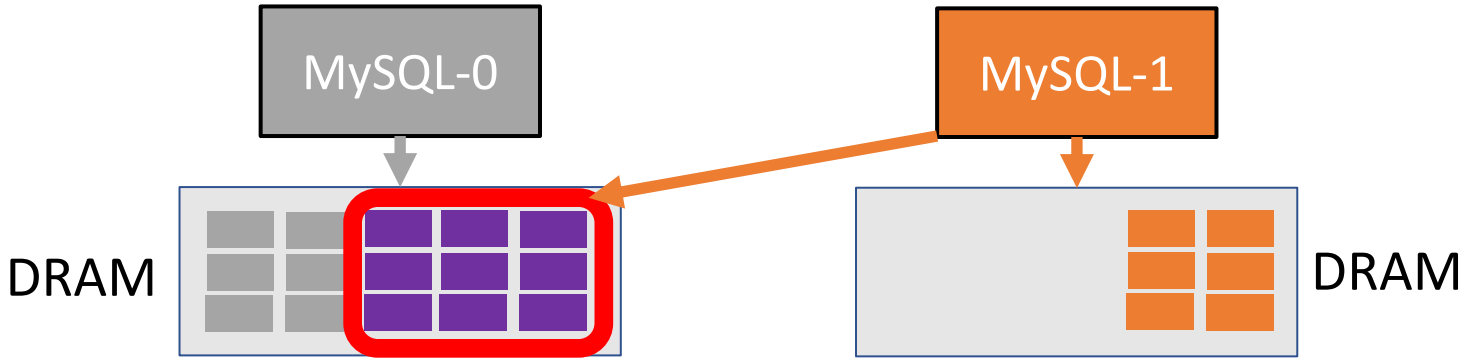
- Memory de-duplication (KSM in Linux) is NUMA-unaware
  - Uncontrolled performance variability
  - Subversion of process priority
- Our proposal: NUMA-aware KSM (nuKSM) in Linux
  - Judicious placement of de-duplicated pages to limit NUMA overheads
  - Priority-aware placement of de-duplicated pages
- Beyond NUMA: De-centralize KSM's data structures for scalability

De-duplication reduces memory consumption

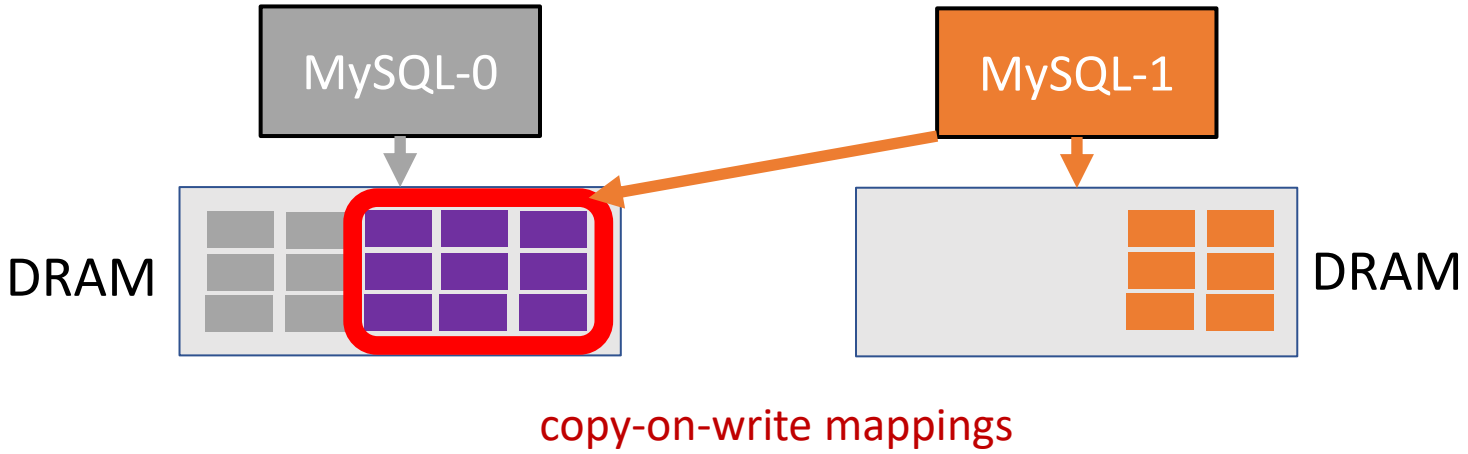


- Identical content across multiple processes/VMs → Opportunity for memory consolidation
  - Similar applications, OS images, libraries, etc.

De-duplication reduces memory consumption

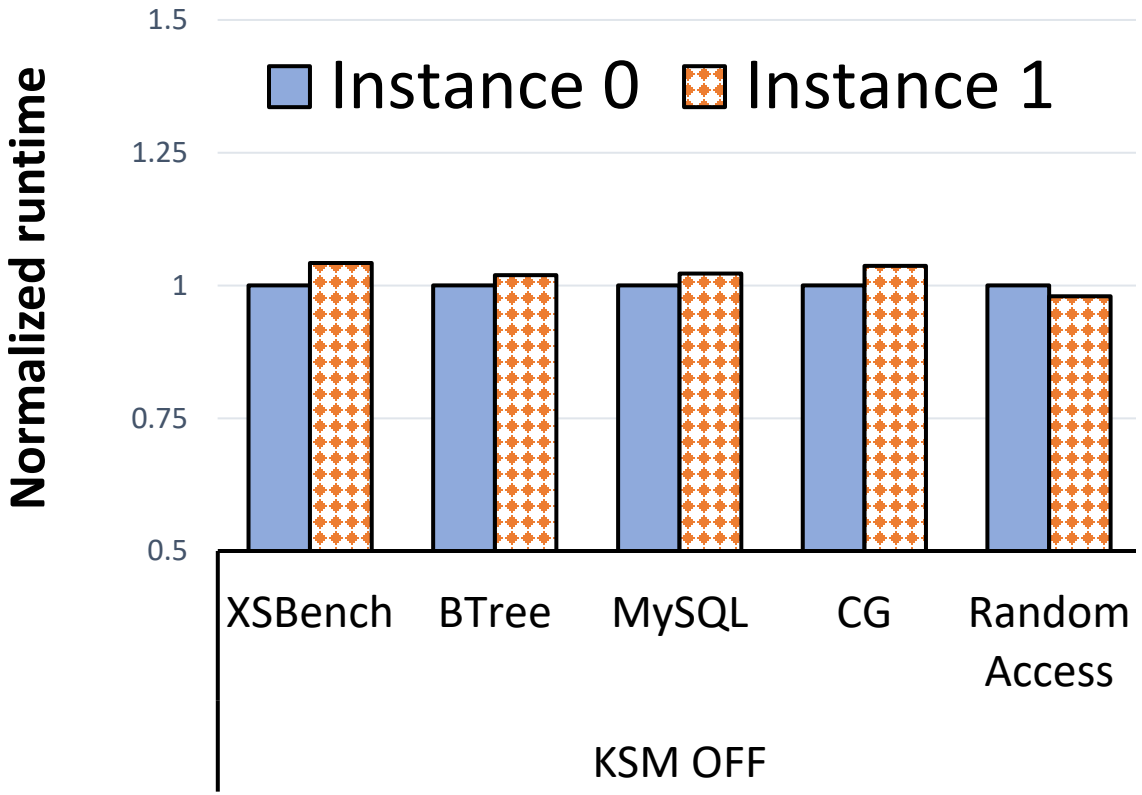


De-duplication reduces memory consumption



- Scan pages in physical memory
- Remove duplicate content using copy-on-write
  - Examples: Kernel same page merging (KSM) in Linux, VMware's Transparent Page Sharing (TPS)

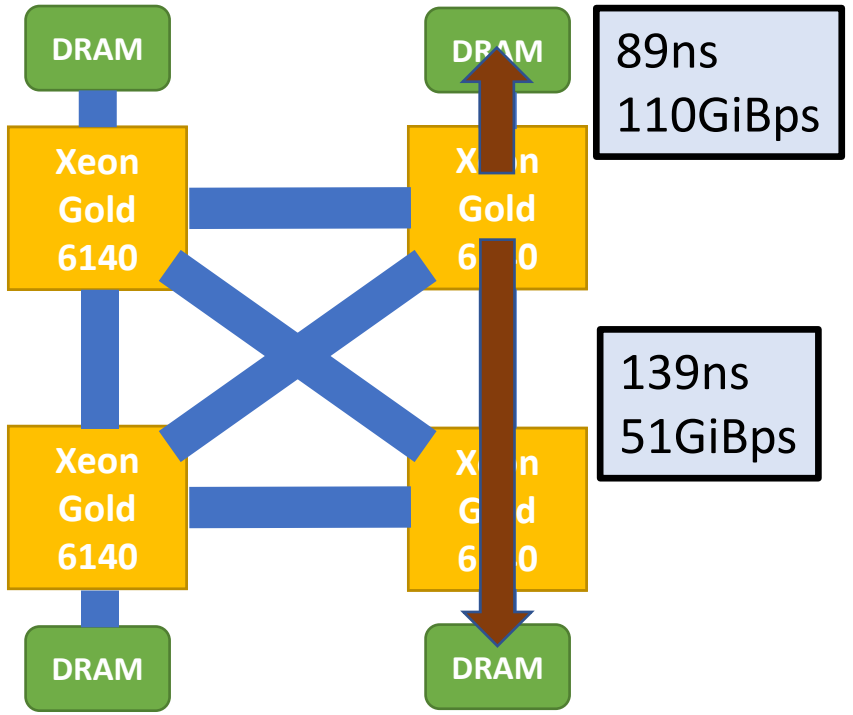
Observation 1: De-duplication introduces perf. variations



20-46% performance variation across instances of the same application!

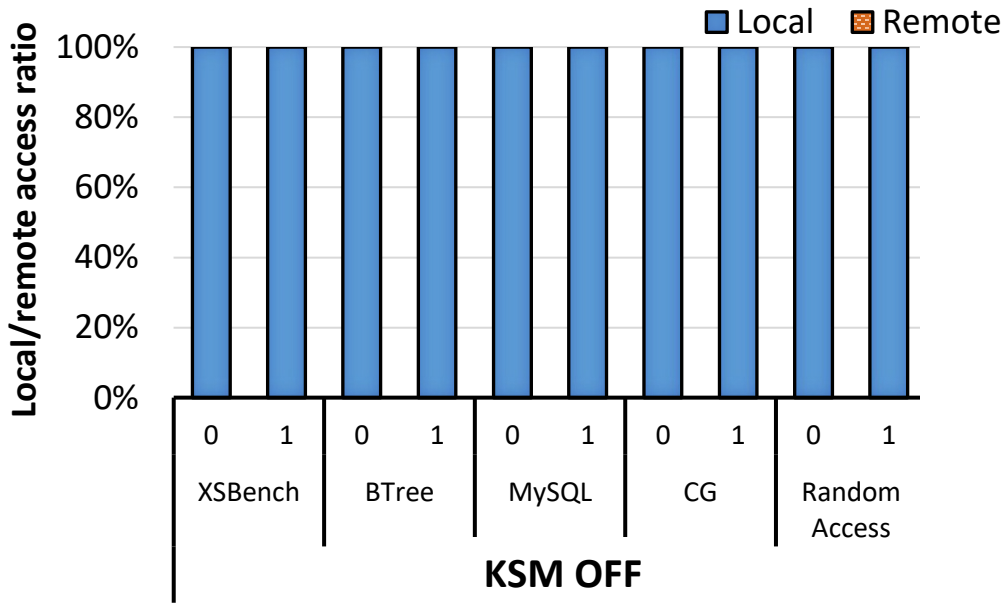
NUMA-ness in multi-socket servers

- Non-uniform memory access (NUMA) systems
  - Important for scaling memory capacity and bandwidth
  - Performance determined by access latency

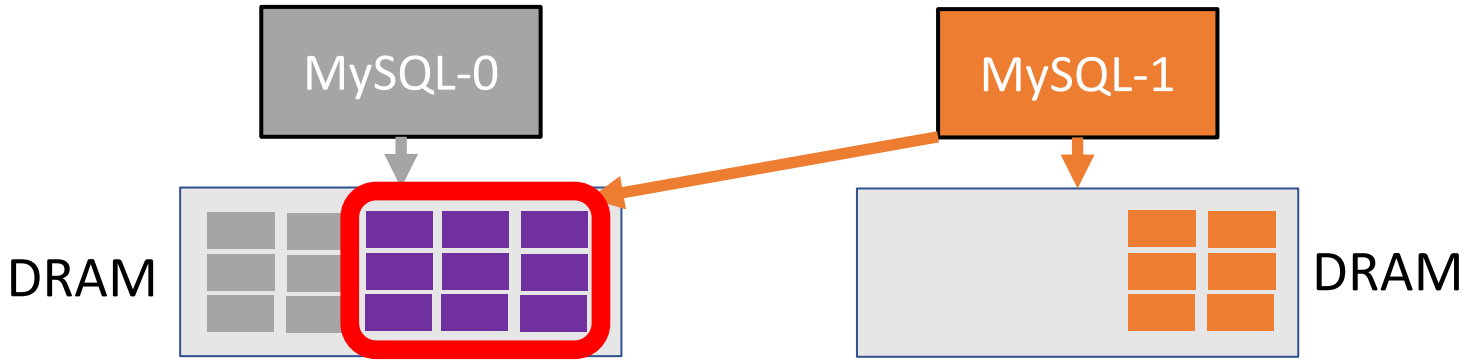


Insight: De-duplication collides with NUMA

- Unbalanced local/remote memory access ratio across instances

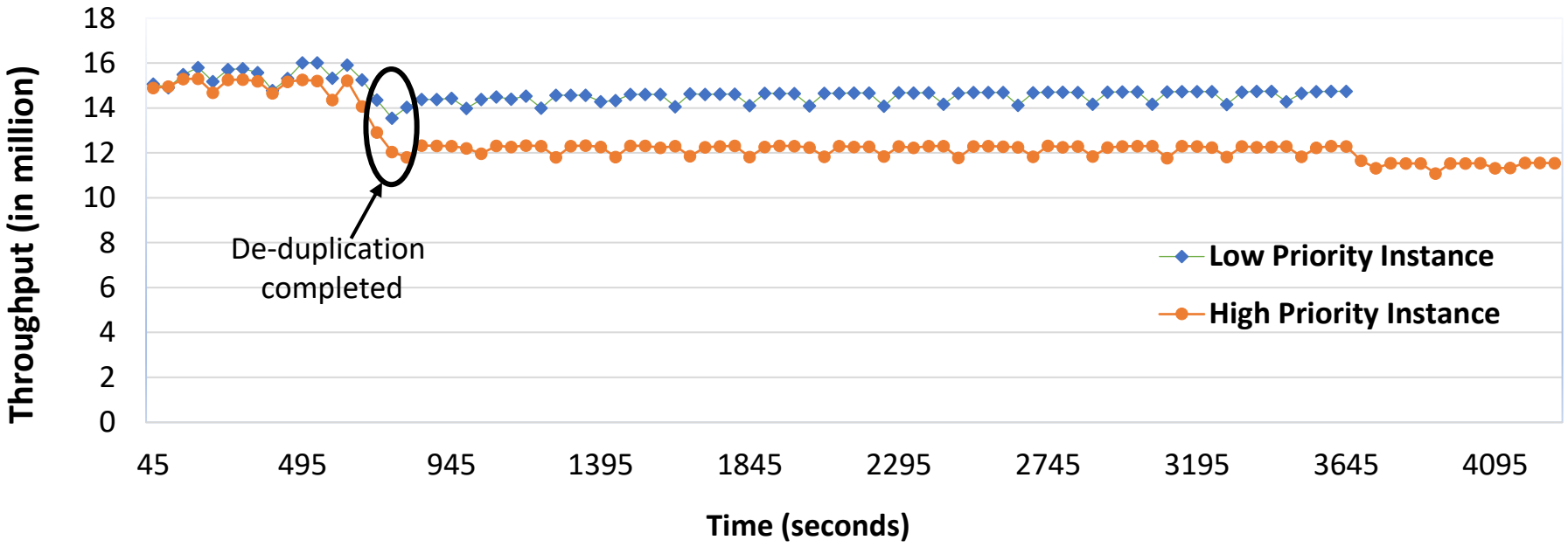


Insight: De-duplication collides with NUMA



- Could have placed de-duplicated pages on Instance “1”’s local memory
- How does KSM decide where to place a merged page?
  - Rather arbitrary: Which ever process’s pages are scanned later
  - The order of scanning dictates page placement !

Observation 2: Subversion of priority goals



- High priority process may suffer high remote memory accesses
- No way to tune the system

Objective:

Enhance Linux's KSM (de-duplication) to contain performance variations and avoid priority subversion

Proposal:

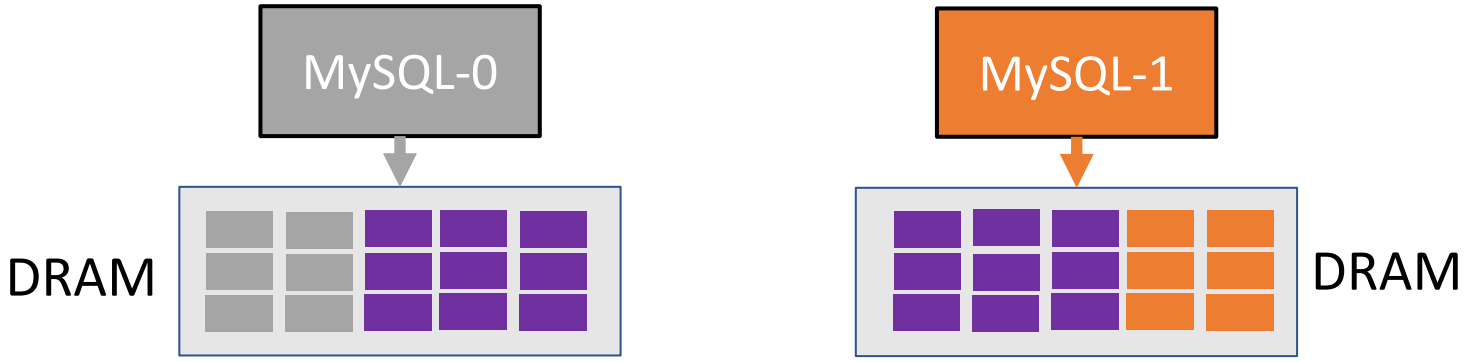
nuKSM: NUMA-aware Memory De-duplication on Multi-socket Servers

Code: https://github.com/csl-iisc/nuKSM-pact21-artifact

nuKSM: Judicious placement of de-duplicated pages

Idea: Place de-duplicated page on the node that accesses it more frequently

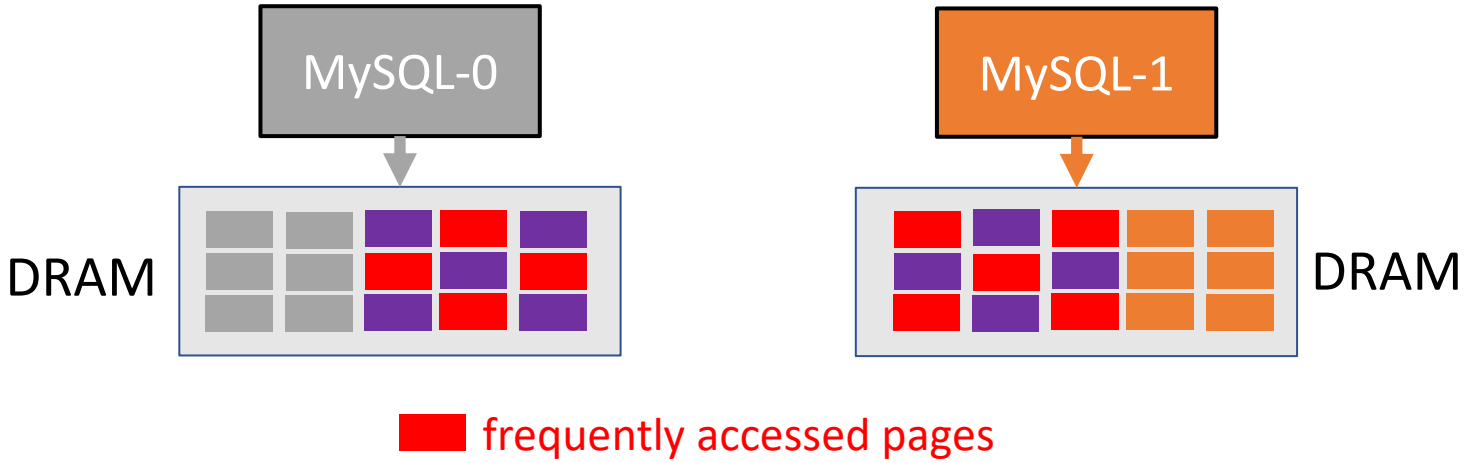
- Minimizes aggregate impact of remote accesses



nuKSM: Judicious placement of de-duplicated pages

Idea: Place de-duplicated page on the node that accesses it more frequently

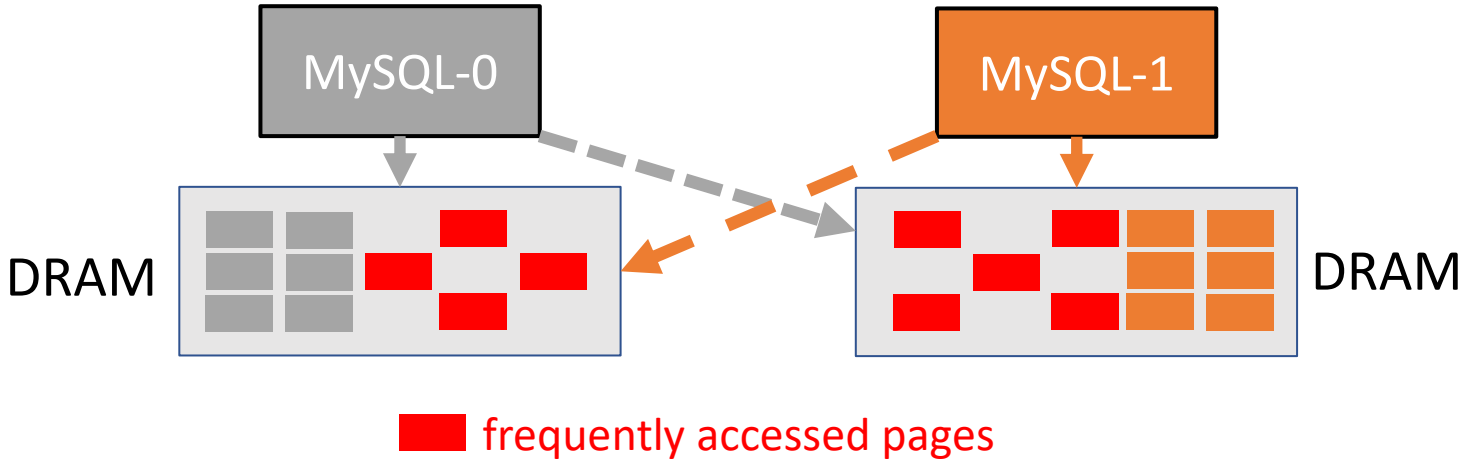
- Minimizes aggregate impact of remote accesses



nuKSM: Judicious placement of de-duplicated pages

Idea: Place de-duplicated page on the node that accesses it more frequently

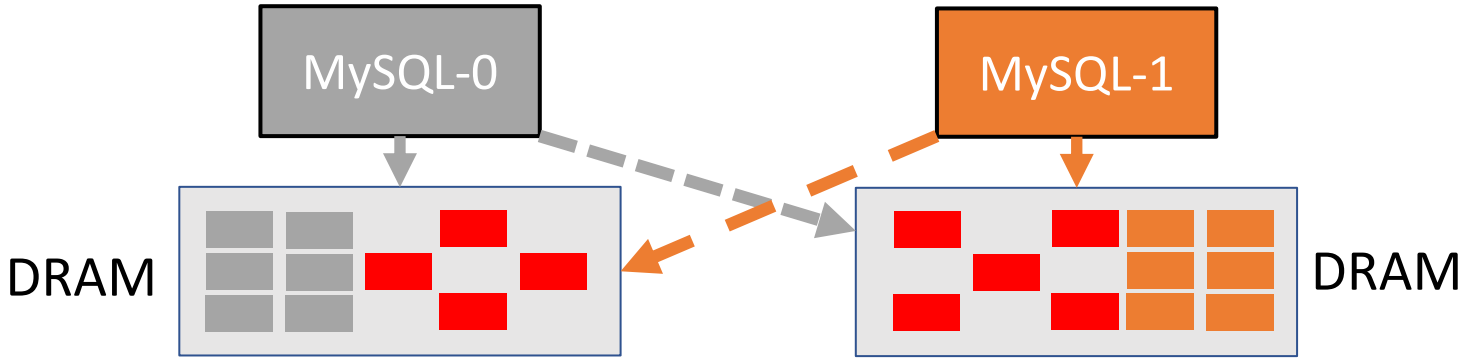
- Minimizes aggregate impact of remote accesses



nuKSM: Judicious placement of de-duplicated pages

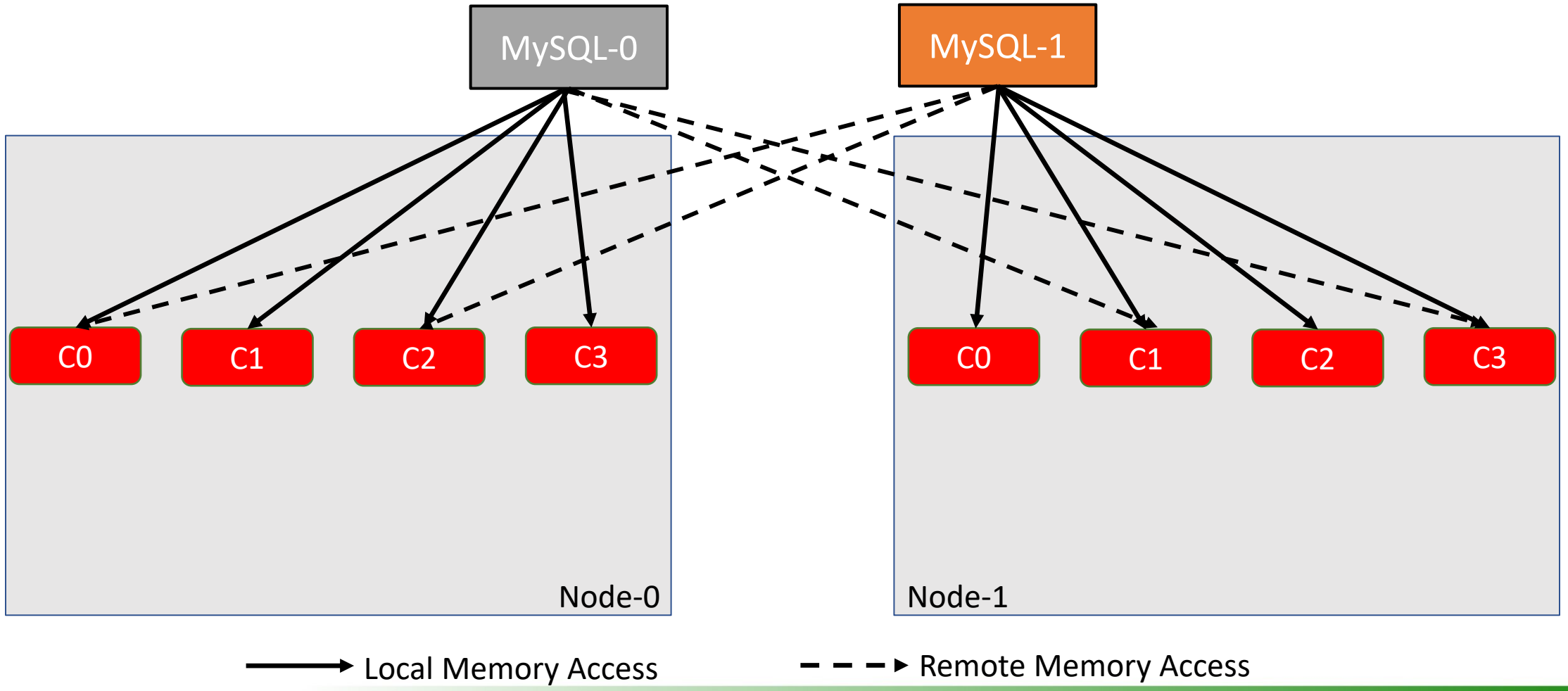
Idea: Place de-duplicated page on the node that accesses it more frequently

- Minimizes aggregate impact of remote accesses



- **Challenge:** How to identify who accesses a page more frequently?
  - Repurpose Linux's **active** and **inactive** lists (originally for swapping)

nuKSM: Judicious placement of de-duplicated pages

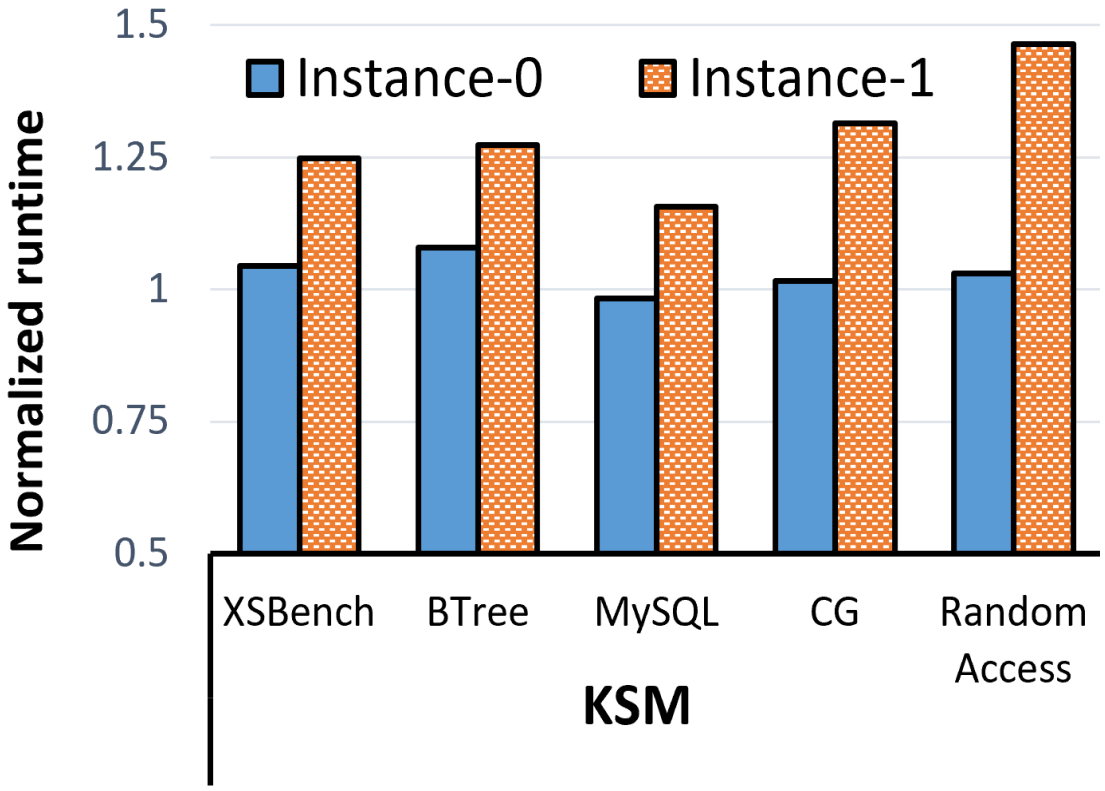


Evaluation Methodology

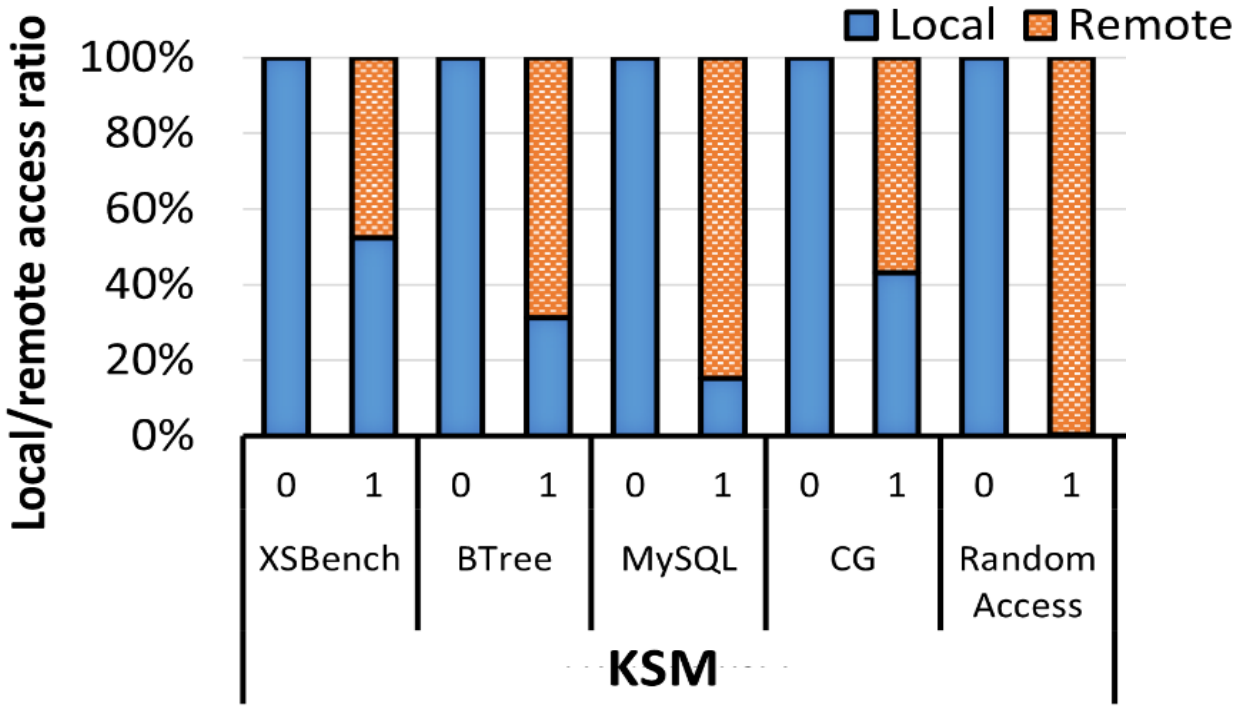
Hardware Platform	
Model	2-socket Intel Skylake
CPU cores	18 cores per socket @ 2.30GHz
Cache	25MiB shared L3 cache
Memory	DDR4-2666, 192GiB per socket

Workloads	
XSBench	11 GiB
MySQL	20 GiB
BTree	5.6 GiB
RandomAccess	2.8 GiB
CG	3.5 GiB

nuKSM avoids performance variations



nuKSM avoids performance variations



nuKSM avoids performance variations

Benchmark	Fairness *	
	KSM	nuKSM
XSBench	0.84	0.98
Btree	0.85	0.98
MySQL	0.85	0.99
CG	0.77	0.99
Random-Access	0.70	0.94

$$* \ Fairness\ (I0, I1) = \frac{\min(slowdown(Instance\ 0), slowdown(Instance\ 1))}{\max(slowdown(Instance\ 0), slowdown(Instance\ 1))}$$

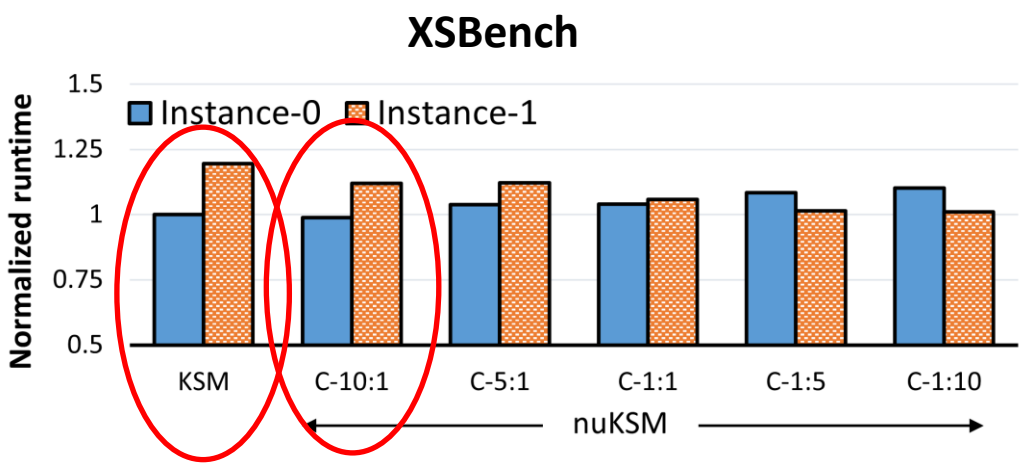
nuKSM: Priority based memory de-duplication

- Distribute NUMA overhead in proportion to priority ratios

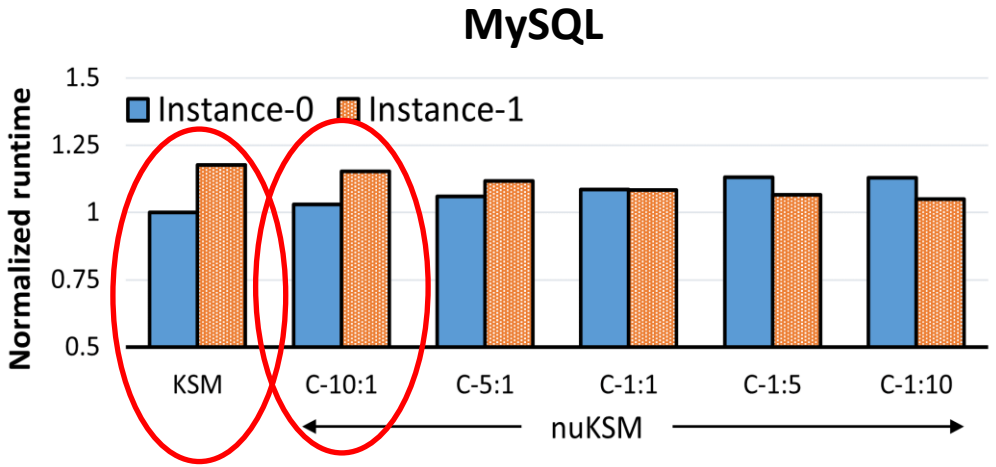
- Derive priority from Linux's *nice* values
  - -20 (highest priority) <= nice <= 19 (lowest priority)
  - *snice (scaled nice)* = nice + 21

- $nushare^{(p)} = 1 - \frac{snice(current)}{\sum_{tasks\ using\ p} snice(task)}$

Priority based memory de-duplication

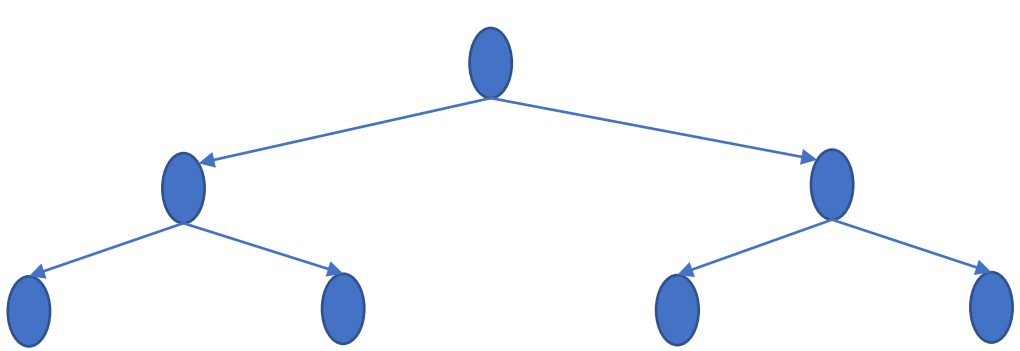


Priority of Instance-0 decreases
Priority of Instance-1 increases

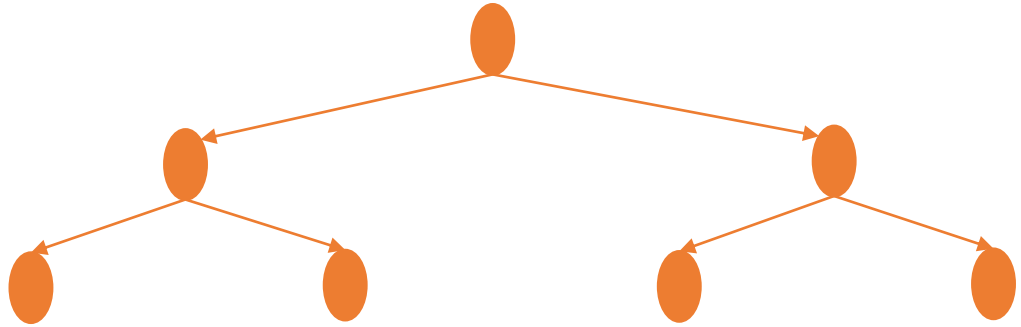


Priority of Instance-0 decreases
Priority of Instance-1 increases

Beyond NUMA: Scaling KSM to larger memory sizes

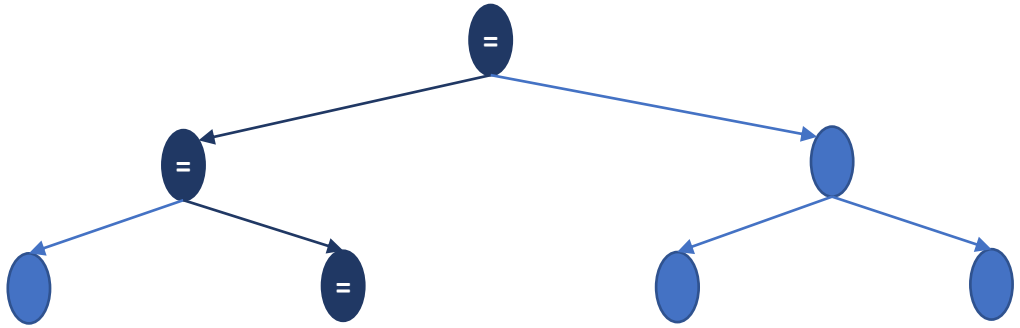


Stable Tree

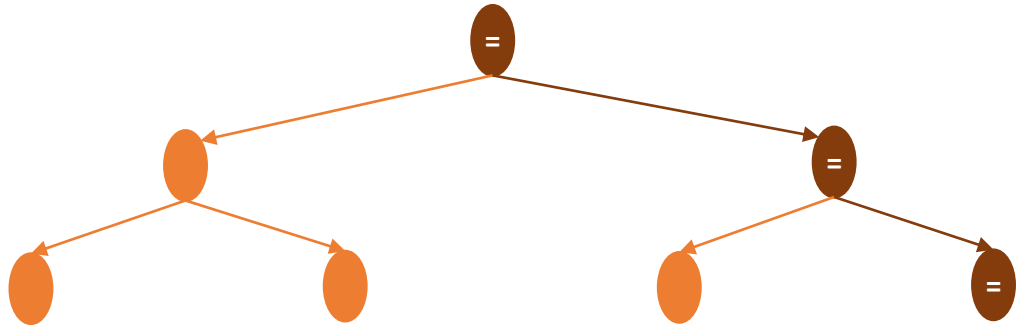


Unstable Tree

Beyond NUMA: Scaling KSM to larger memory sizes



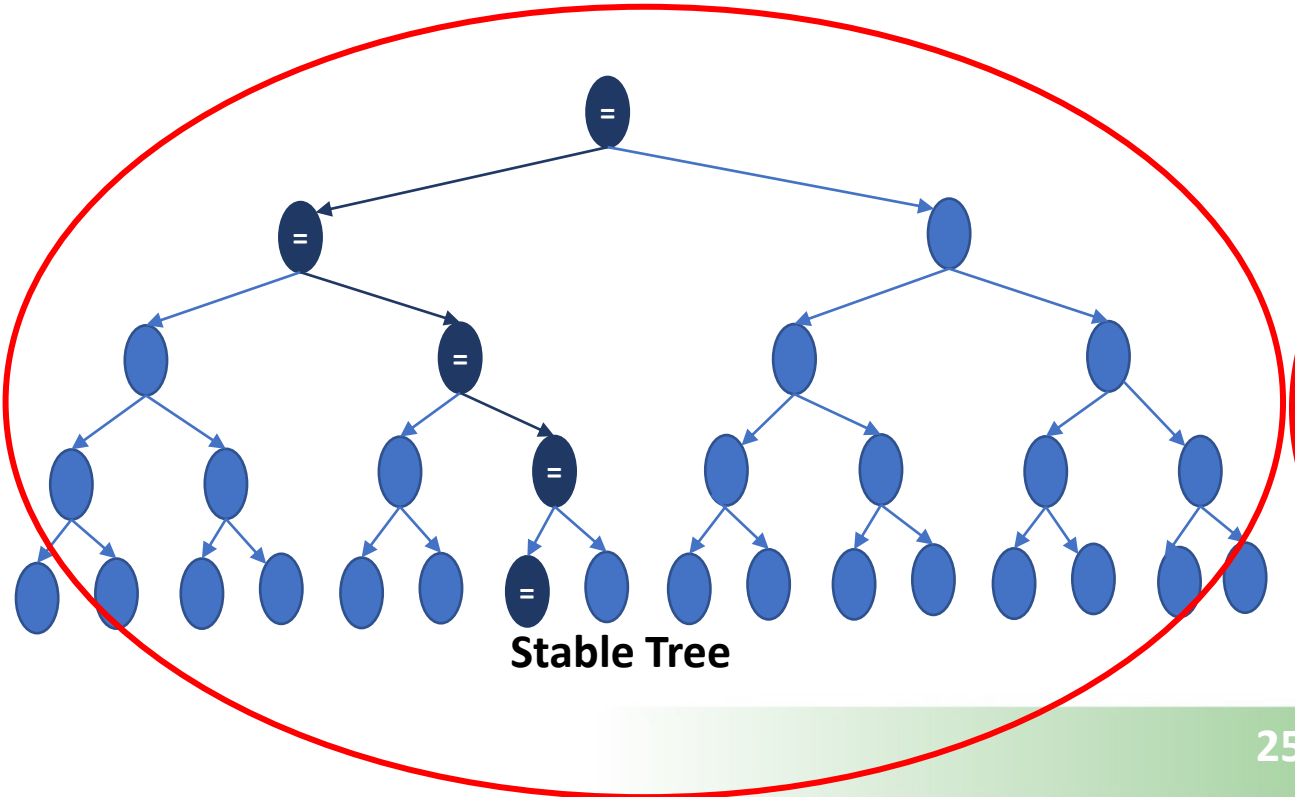
Stable Tree

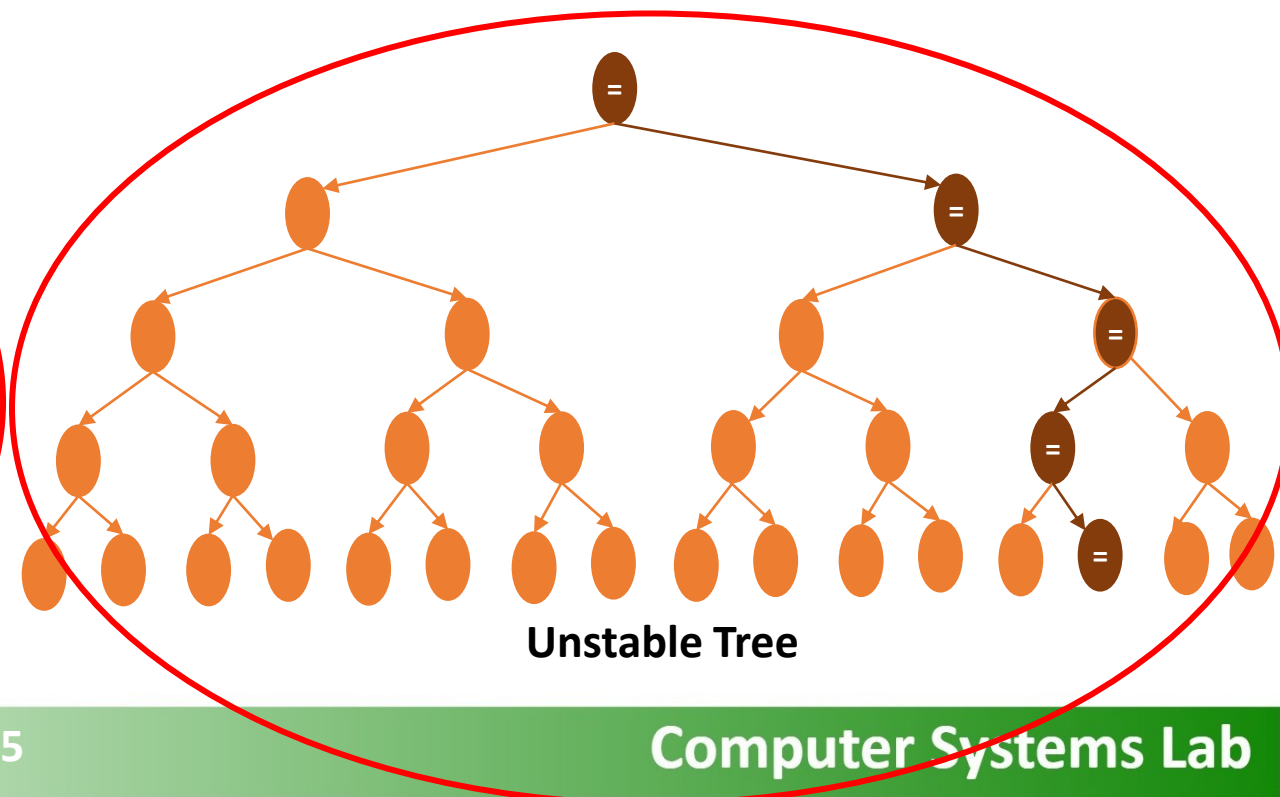


Unstable Tree

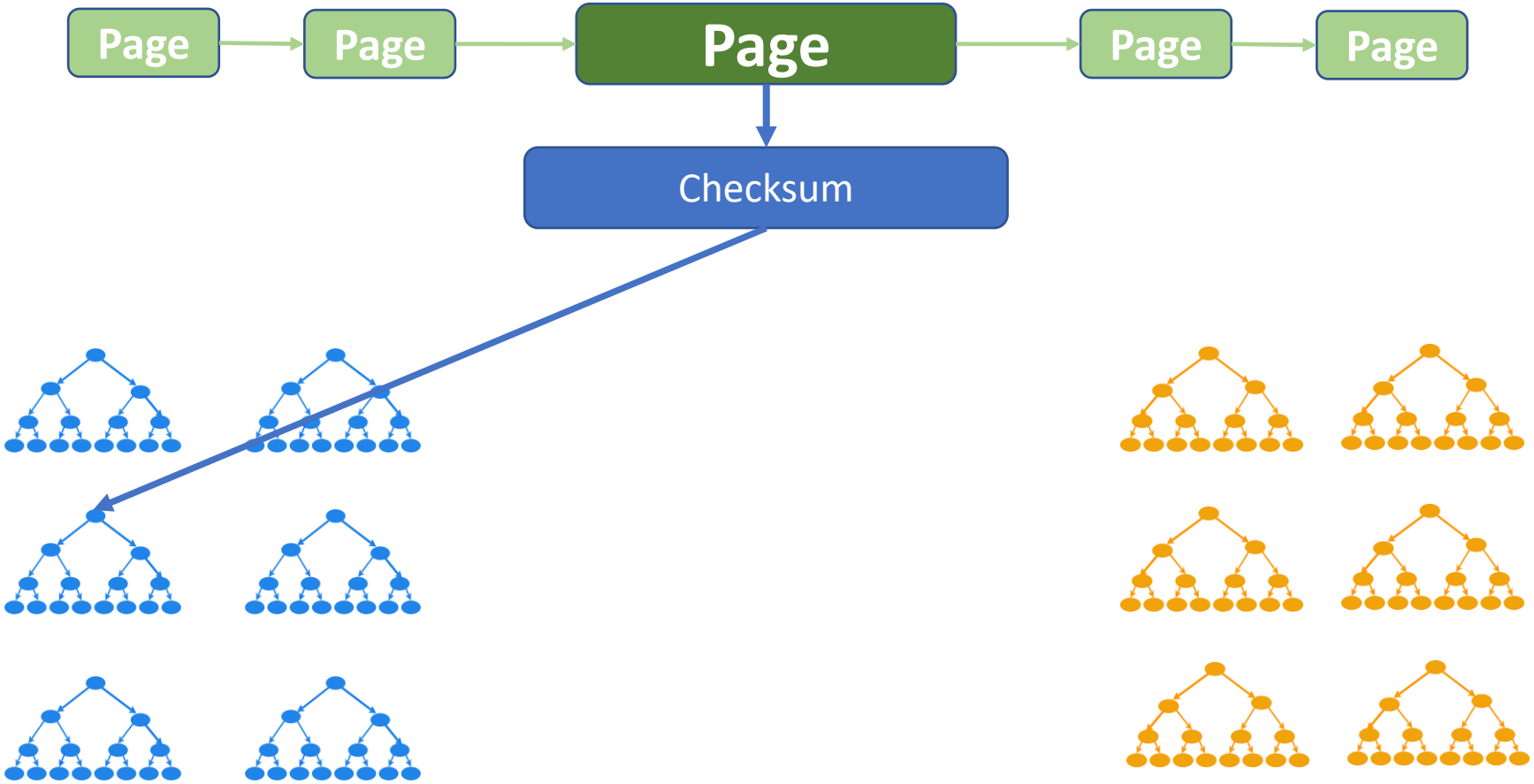
Beyond NUMA: Scaling KSM to larger memory sizes

Centralized data structures

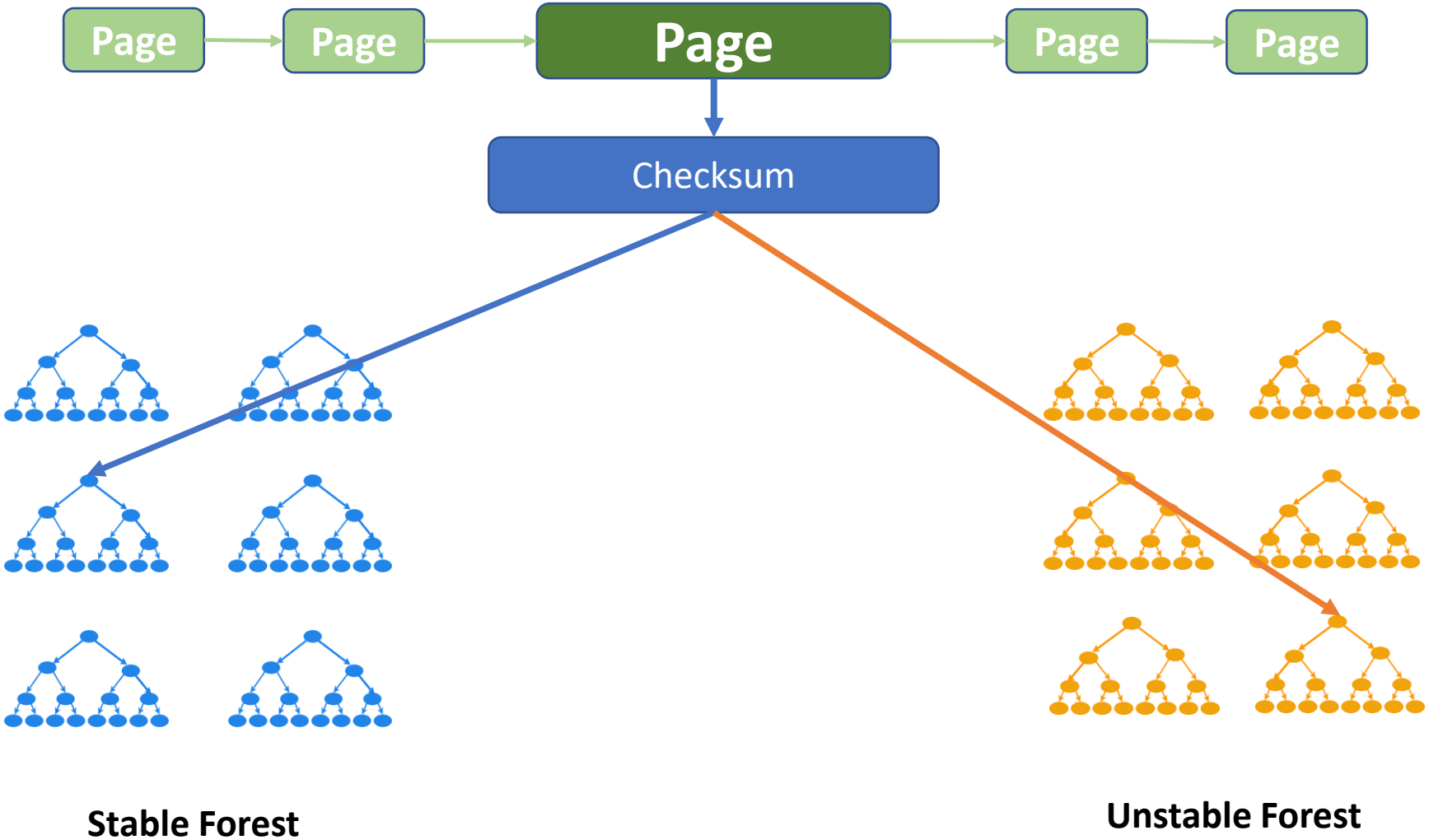




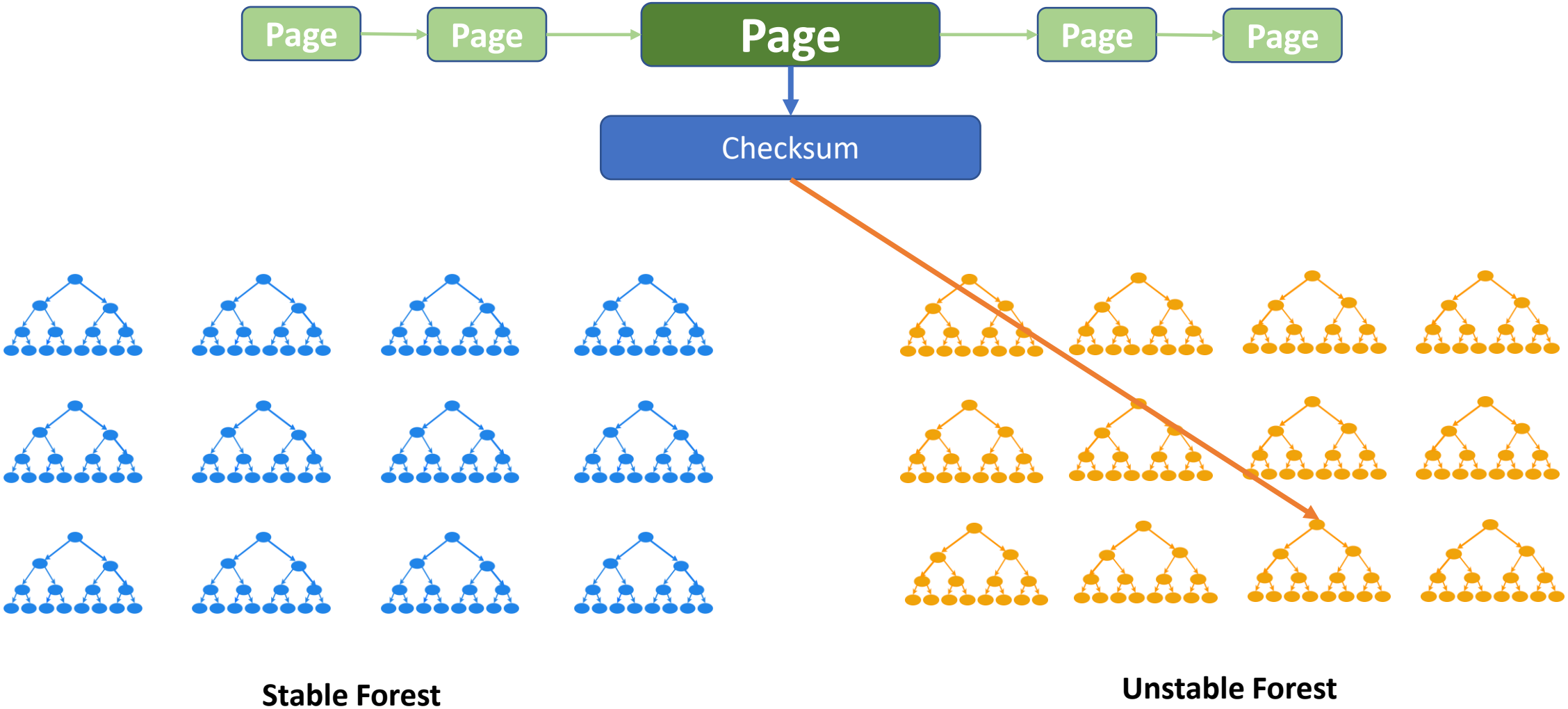
Beyond NUMA: Scaling KSM to larger memory sizes



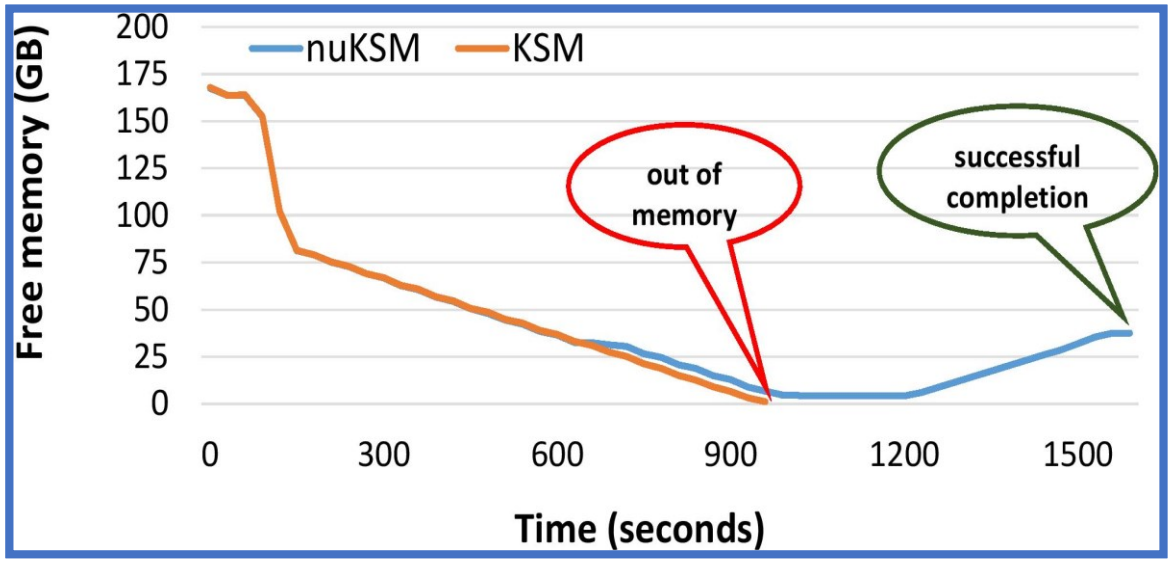
Beyond NUMA: Scaling KSM to larger memory sizes

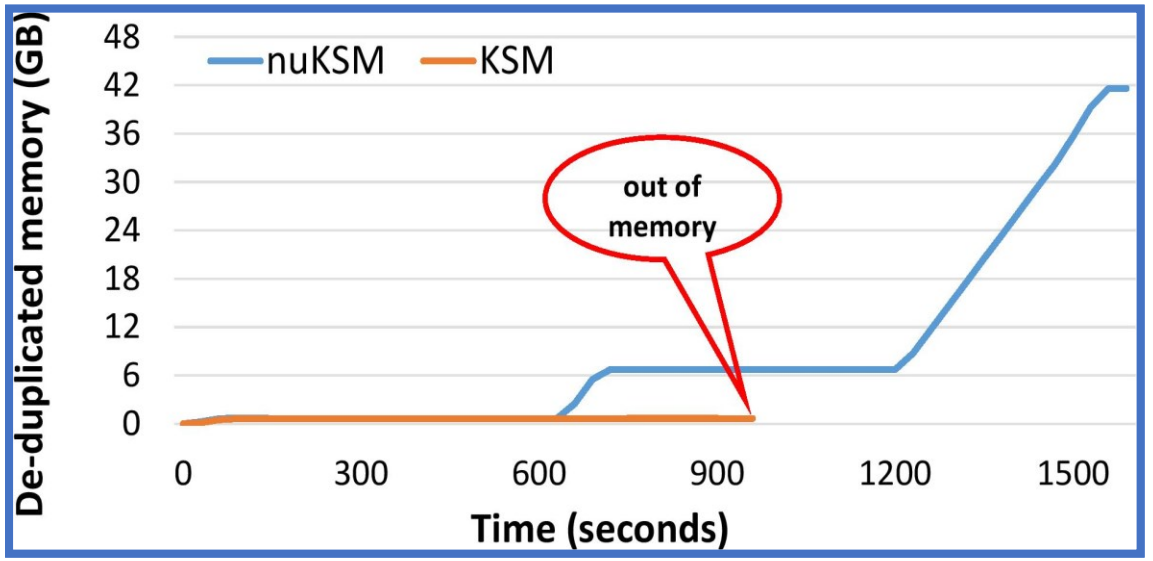


Beyond NUMA: Scaling KSM to larger memory sizes



nuKSM scales better with memory size





Take-aways: NUMA-aware KSM (nuKSM)

- Observation: KSM is NUMA unaware.
  - Arbitrary (uncontrolled) performance variability
  - Subversion of priority goals
  - Low responsiveness with large memory

- Our proposal: nuKSM: NUMA-aware KSM
  - Deterministic performance and fairness
  - Inline with priority objective
  - Enhanced responsiveness

Code: https://github.com/csl-iisc/nuKSM-pact21-artifact

Questions

Code: https://github.com/csl-iisc/nuKSM-pact21-artifact