SAND2014-4107P

Sandia National Laboratories

Exceptional service in the national interest

Fault-tolerant programming at the extreme-scale

Work by Janine Bennett, John Floren, Hemanth Kolla, Nicole Slattengren, Keita Teranishi, Jeremiah Wilke

May 12, 2014

U.S. DEPARTMENT OF ENERGY

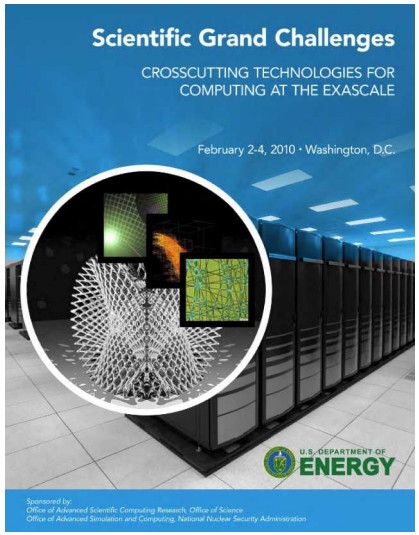
NNSA National Nuclear Security Administration

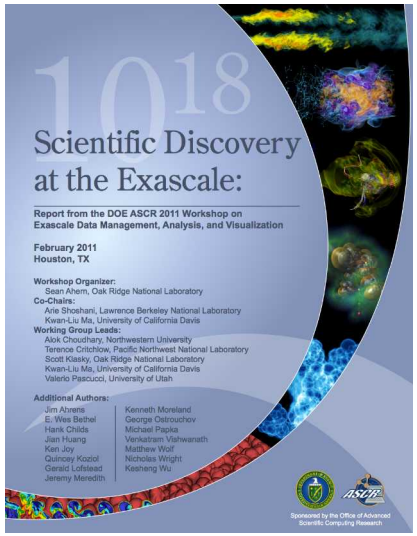
Sandia National Laboratories is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000. SAND NO. 2011-XXXXP

Next generation platforms will have significant increases in concurrency

Sandia National Laboratories

System Parameter	2011	2018		Factor Change
System Peak	2 Pf/s	1 Ef/s		500
Power	6 MW	≤20 MW		3
System Memory	0.3 PB	32-64 PB		100-200
Total Concurrency	225K	1 BX10	1B X100	40000-400000
Node Performance	125 GF	1 TF	10 TF	8-80
Node Concurrency	12	1000	10000	83-830
Network Bandwidth	1.5 GB/s	100 GB/s	1000 GB/s	66-660
System Size (nodes)	18700	1000000	100000	50-500
I/O Capacity	15 PB	30-100 PB		20-67
I/O Bandwidth	0.2 TB/s	20-60 TB/s		10-30





The increase in concurrency is driving us to change how we think about programming

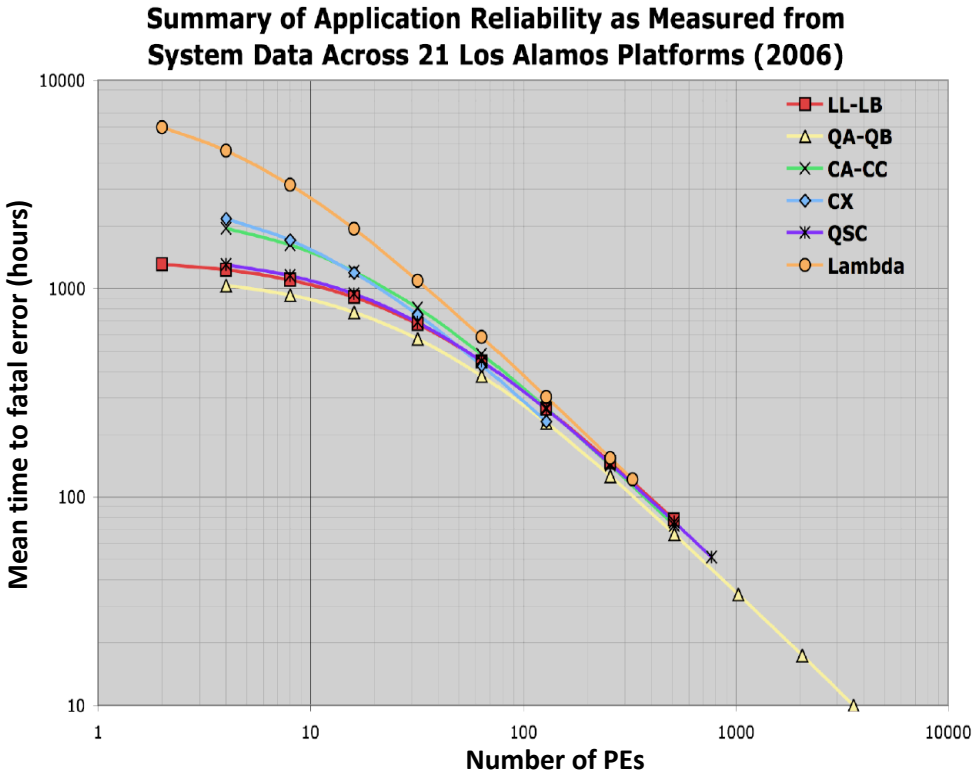
Sandia National Laboratories

- MPI + X hybrid models
  - Focus on extracting on-node parallelism
  - Cuda, OpenCL, Cilk+, OpenMP, Kokkos, …
- Asynchronous many-task programming models
  - Over-decompose problem
  - Overlap of communication and computation
  - Charm++, Uintah, Legion, Scioto, Dague, Intel CnC, Tasks & Chunks, …
- Others: Chapel, Fortress, X10, ParalleX/HPX, UPC, …

Next generation platforms will experience errors/faults much more frequently

Sandia National Laboratories

- Significant increase in number of components
- Insufficient improvements in mean time between failures (MTBF) for each component
- Majority of failures: single node
  - Today's rate: ~2-10 a day
  - 2020: every 30-60 minutes?



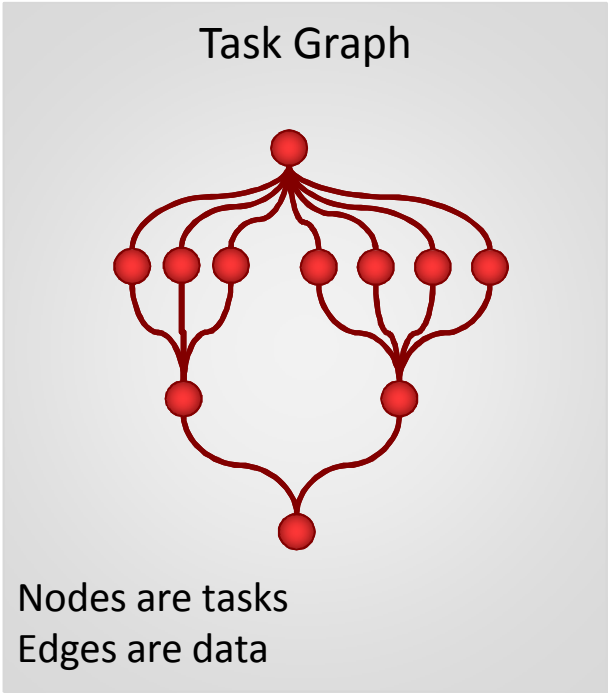
(Courtesy of John Daly)

Traditional checkpoint/restart timings are projected to exceed mean time to error

Future programming models need to support fault-tolerance and recovery

Sandia National Laboratories

- Asynchronous many-task programming models
  - + Show promise at sustaining performance
  - + Work stealing enables load balancing
  - + Failed tasks can be re-executed
- Recovery (beyond checkpoint/restart) is challenging
  - - Enormous distributed coherency problem
  - - Care is required to identify lost tasks due to work-stealing and asynchrony



We are developing a holistic fault-tolerant AMT programming model solution

Sandia National Laboratories

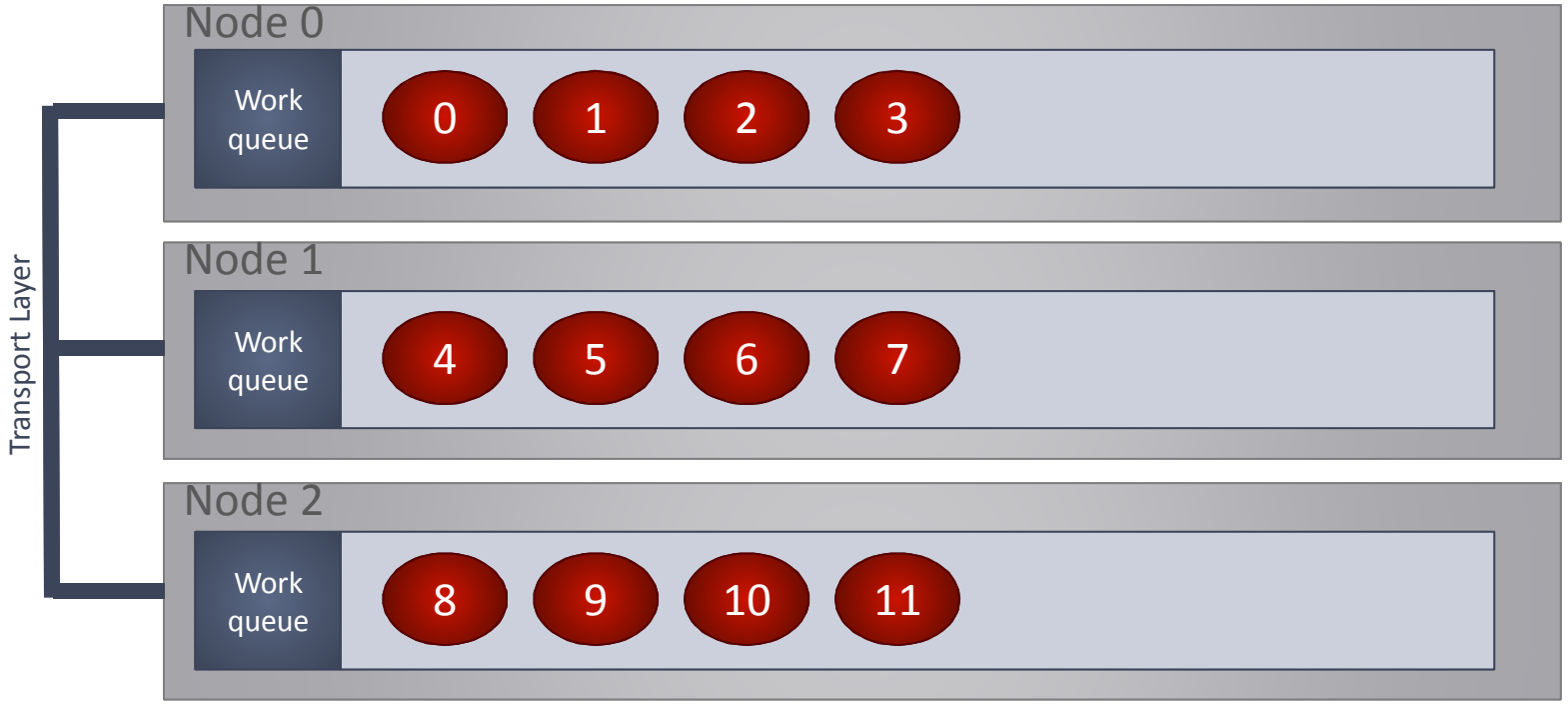
- ***Task collections*** provide a mechanism to support a deferred consistency model
  - Comprise independent tasks distributed across nodes
  - A single collective (global transaction) identifies failures and creates globally consistent view of work queue

- Recent work in this area:

Dinan, J., A. Singri, et al. (2010). **Selective Recovery from Failures in a Task Parallel Programming Model**. 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing (CCGrid): 709-714.

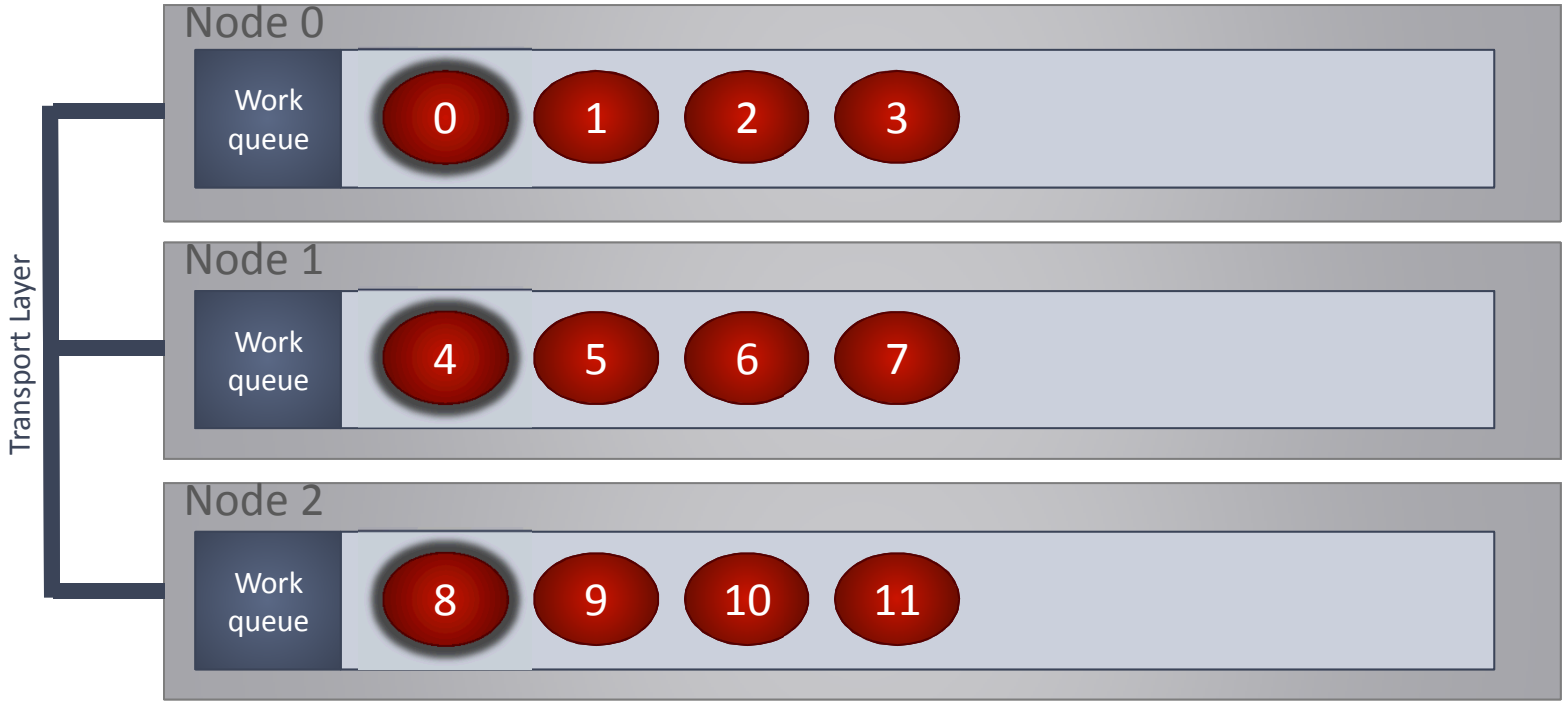
Ma, W. and S. Krishnamoorthy (2012). **Data-Driven Fault Tolerance for Work Stealing Computation**s. 26th ACM international conference on Supercomputing. San Servolo Island, Venice, Italy, ACM: 79-90.

A simple single task collection example



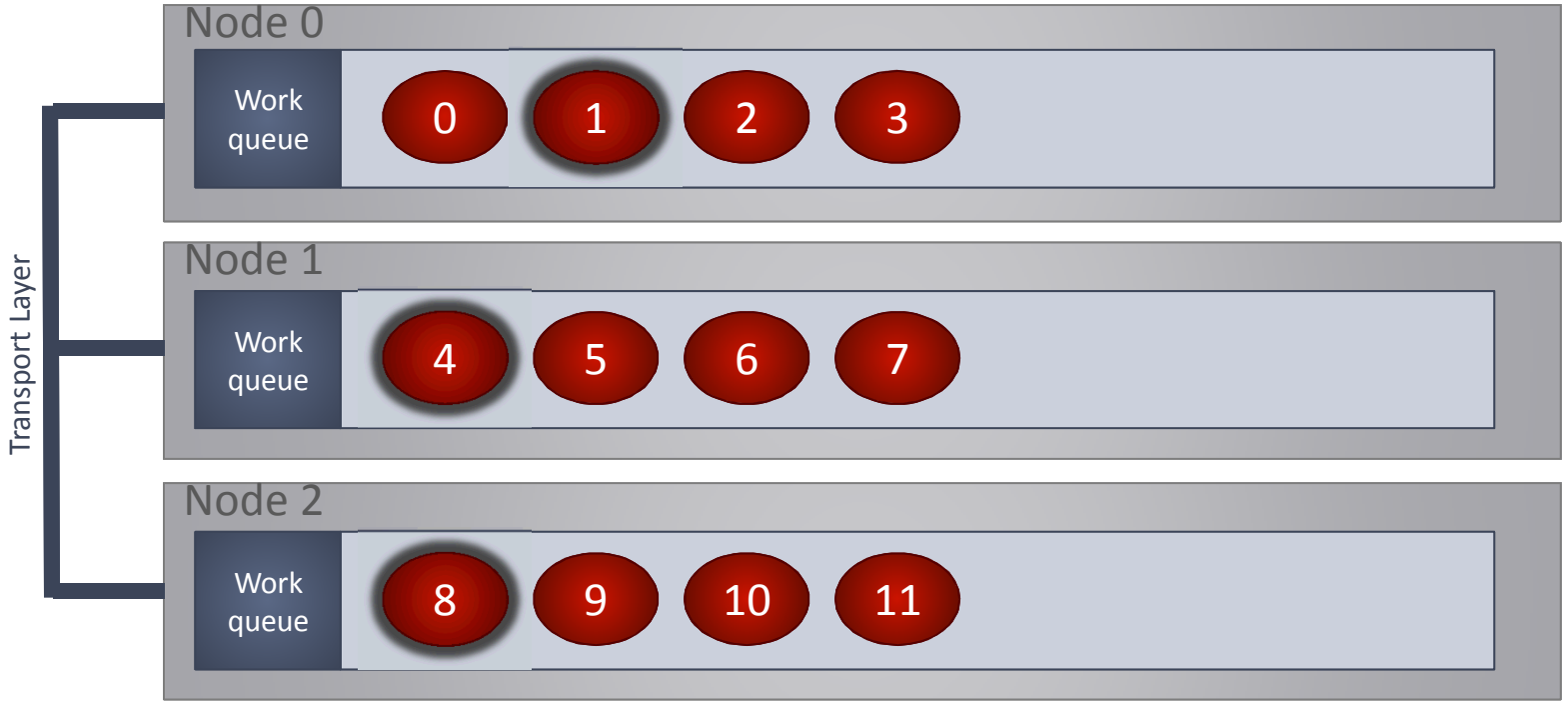
- Tasks are independent and distributed across nodes
- A global address space is assumed

A simple single task collection example



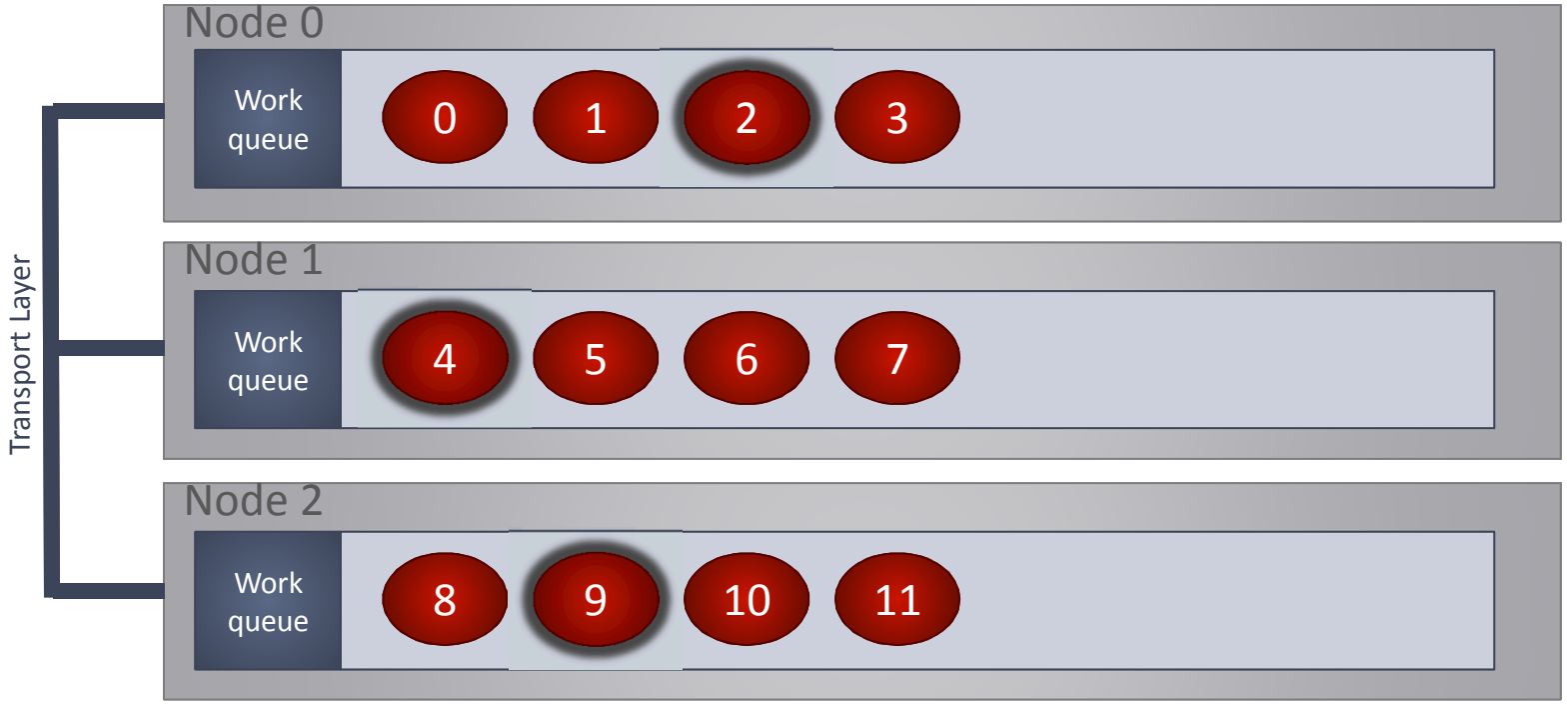
- Within a collection nodes execute tasks asynchronously

A simple single task collection example



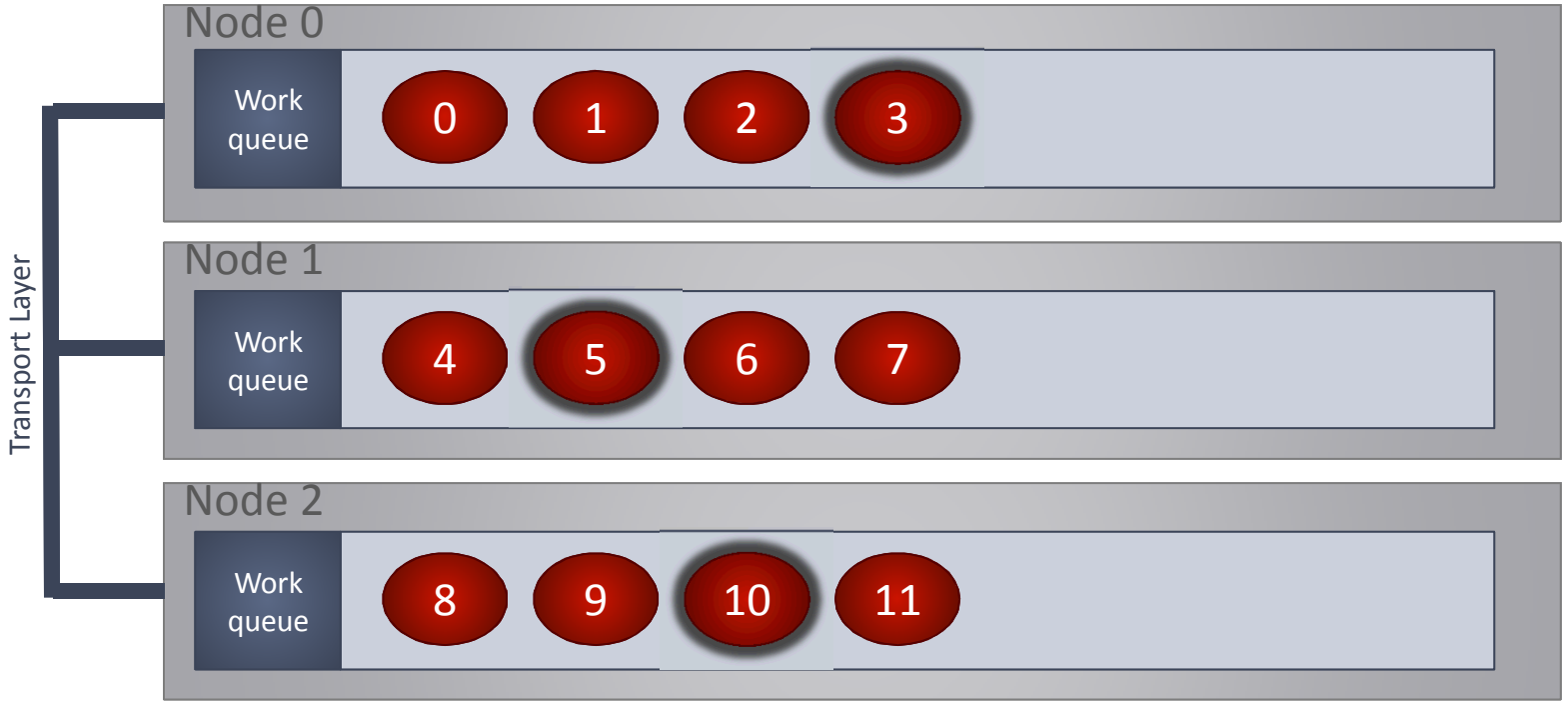
- Within a collection nodes execute tasks asynchronously

A simple single task collection example



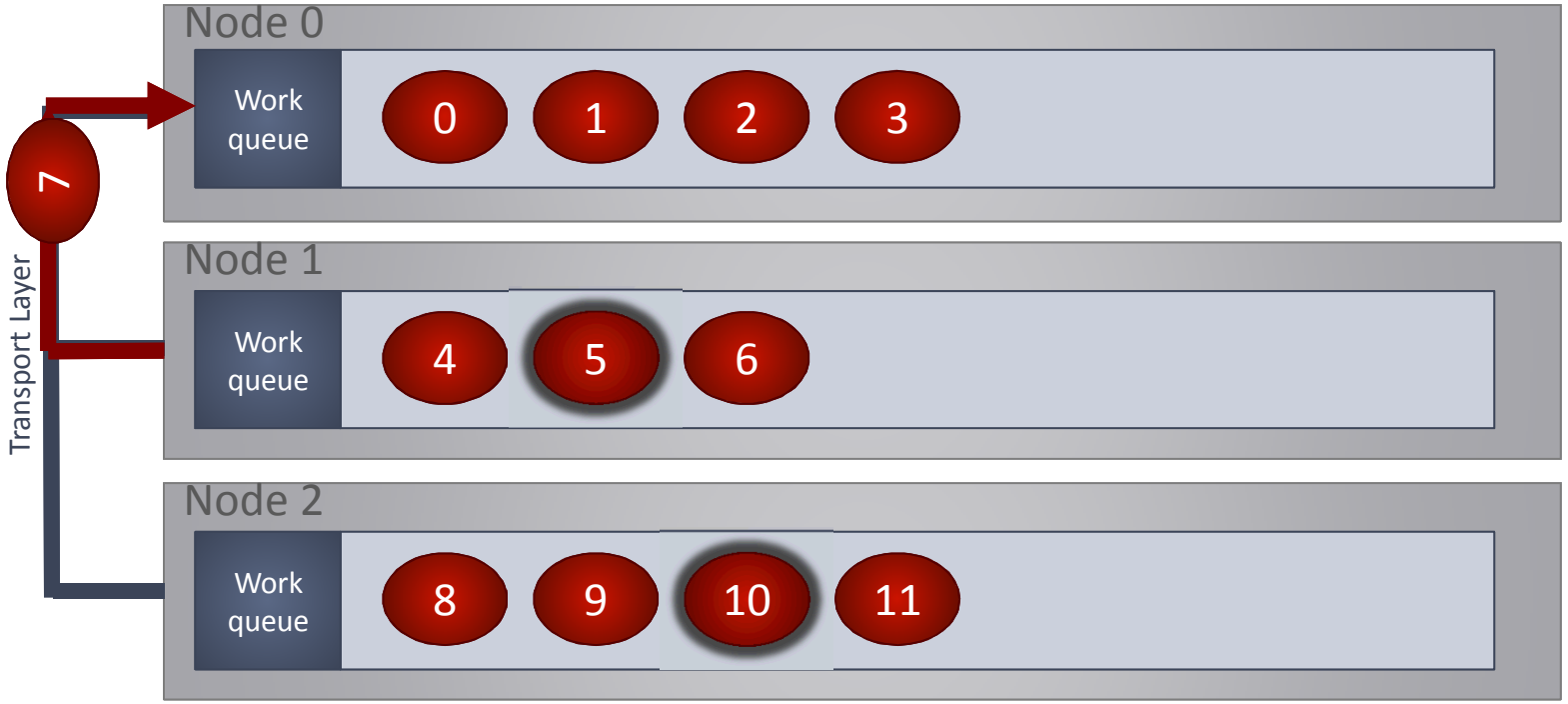
- Within a collection nodes execute tasks asynchronously

A simple single task collection example



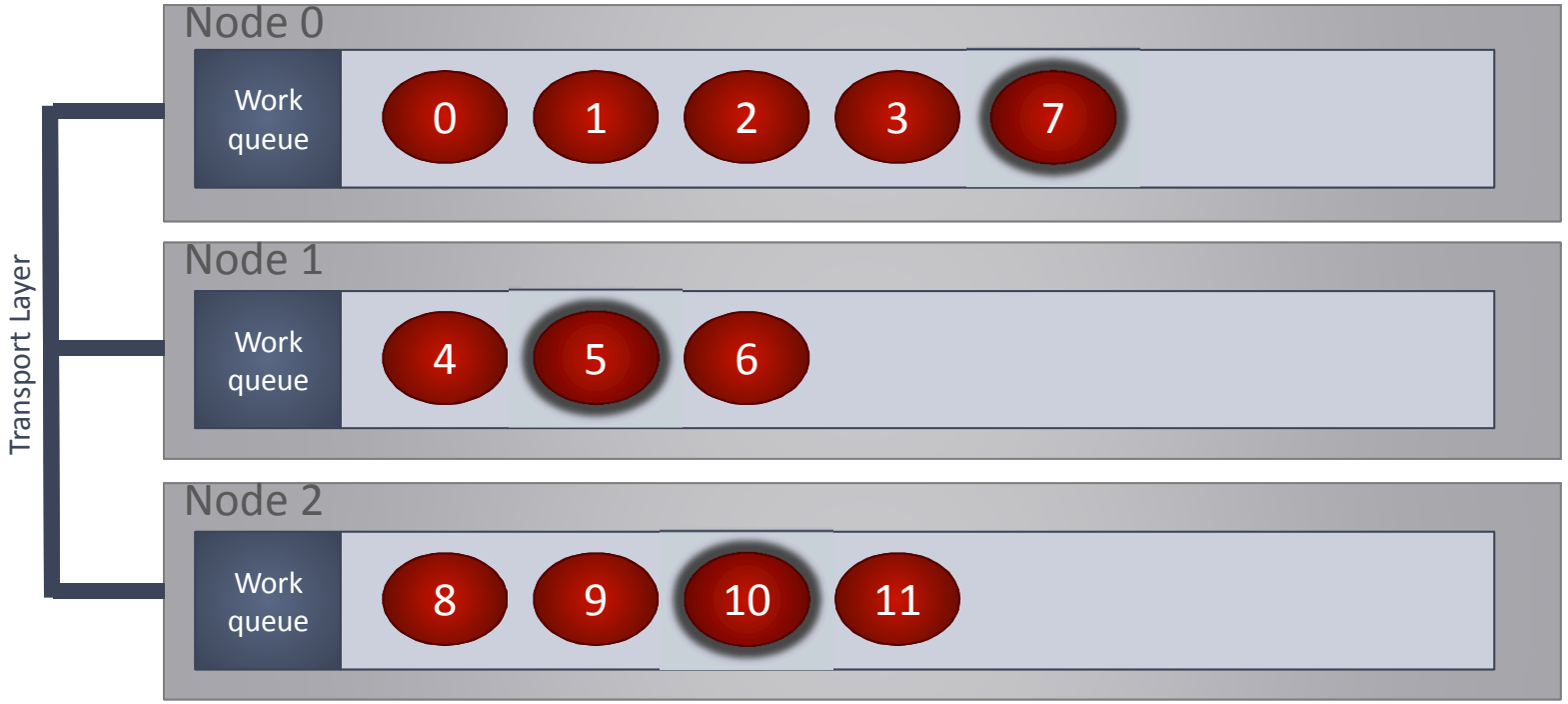
- Within a collection nodes execute tasks asynchronously

A simple single task collection example



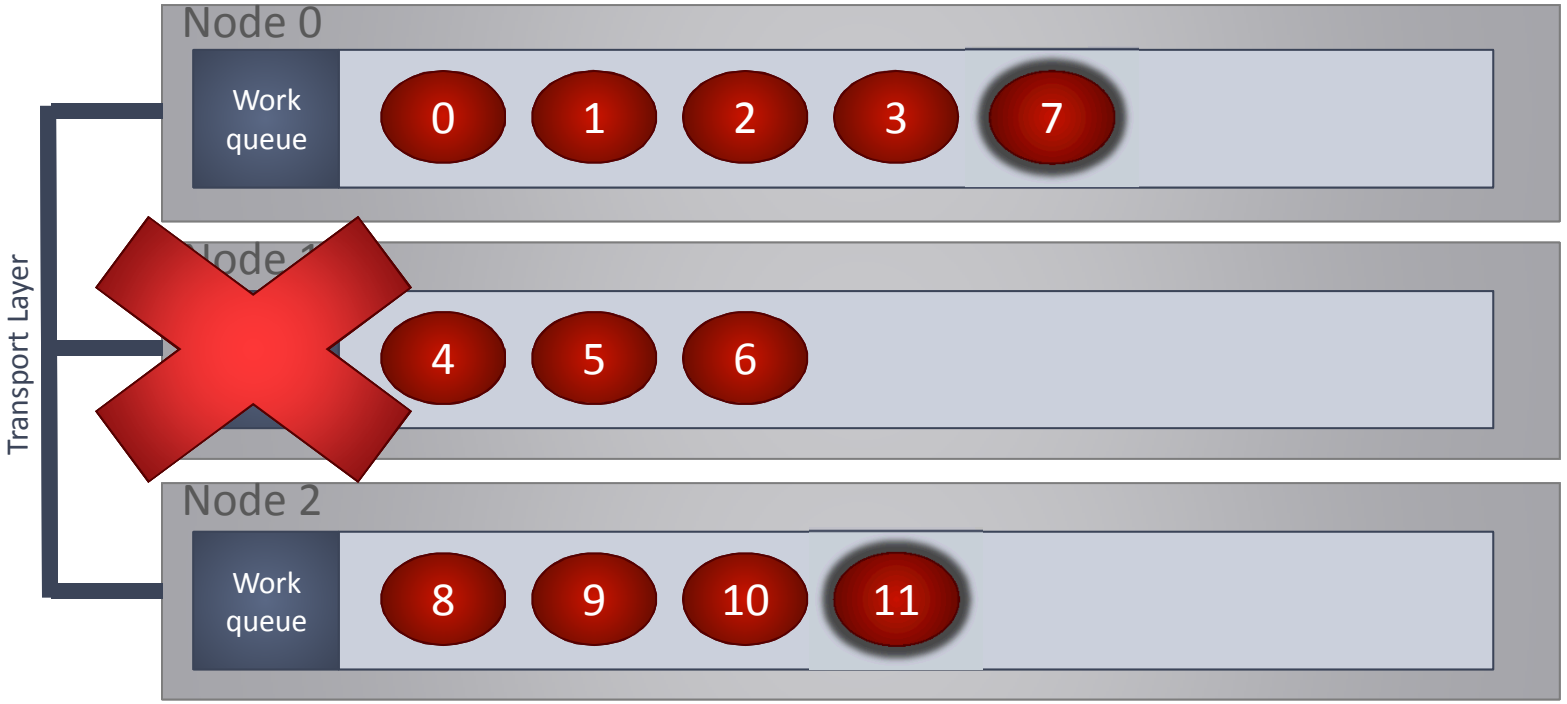
- Work stealing enables tolerance to variations in the execution environment

A simple single task collection example



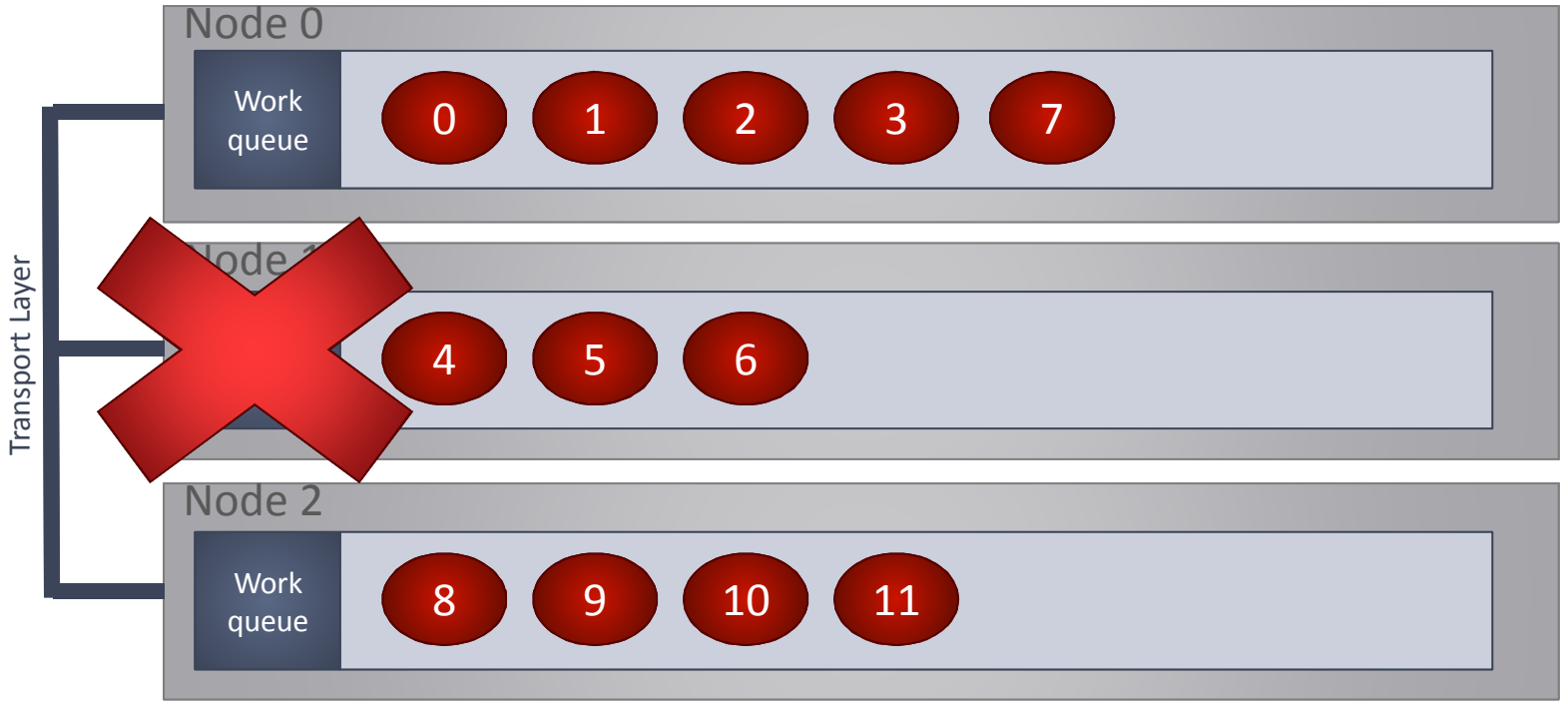
- Work stealing enables tolerance to variations in the execution environment

A simple single task collection example



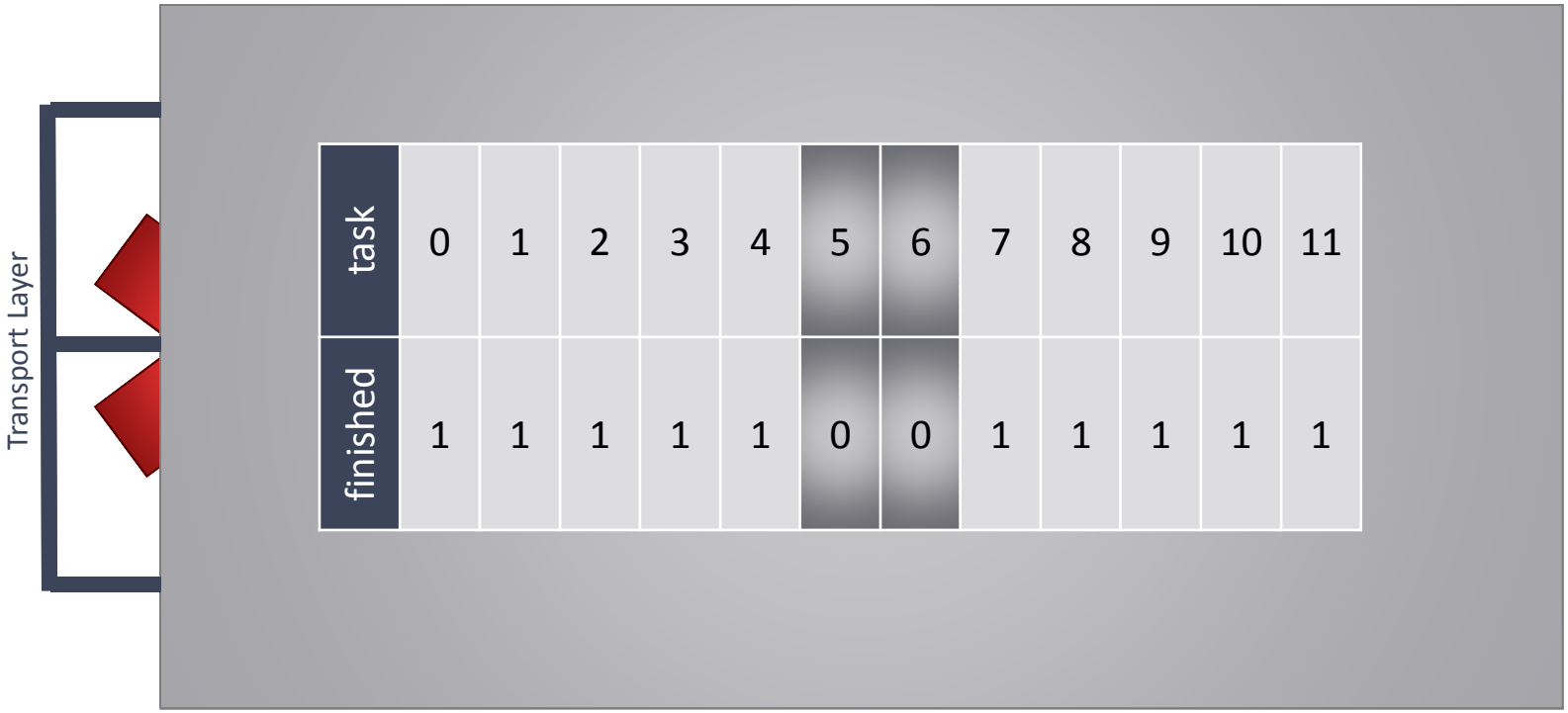
- Recovery is possible when a node goes down
- A simple lazy scheme ignores faults until task collection has terminated

A simple single task collection example



- Recovery is possible when a node goes down
- A simple lazy scheme ignores faults until task collection has terminated

A simple single task collection example



- A single collective identifies failures and creates globally consistent view of work queue
- Highlighted tasks are incomplete

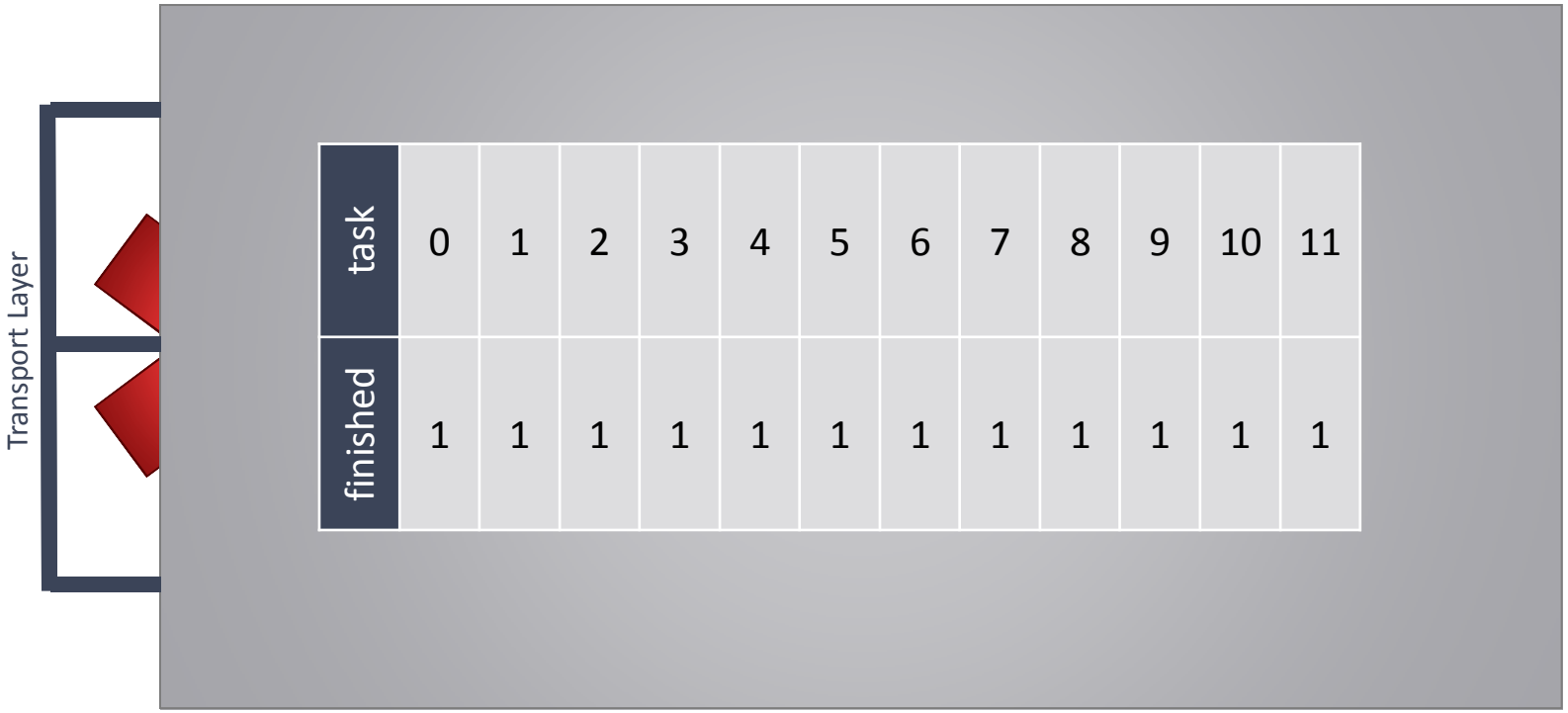
A simple single task collection example

- The incomplete tasks are re-distributed to active nodes
- Execution continues until all tasks have finished

A simple single task collection example

- The incomplete tasks are re-distributed to active nodes
- Execution continues until all tasks have finished

A simple single task collection example

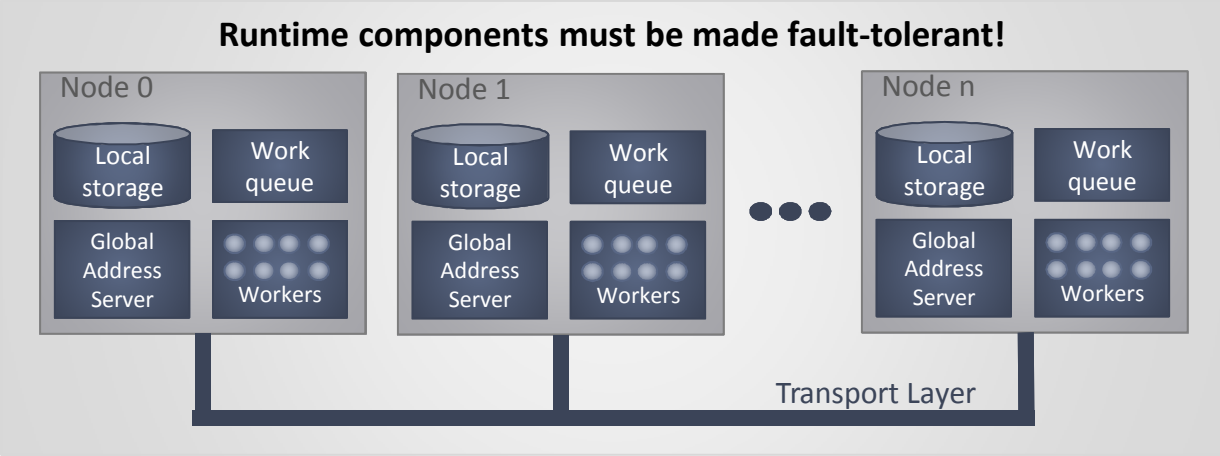


- A global reduction indicates that all tasks have completed

Fault-tolerant task collections at scale requires us to address fundamental research challenges

Sandia National Laboratories

- Redundancy required for data and Global Address Server
- Transport Layer
  - Resilient agreement algorithm: Identification of failed processes
  - Resilient collective communication: Need to terminate in spite of failed processes
  - Virtualized collectives: Allow for workers to take on failed rank's work
- Support for overlapping task collections



We are in the process of addressing each of these challenges using discrete event simulation

Sandia National Laboratories

- Skeletonized mini-apps of explicit and implicit solver
- Runtime system in SST-Macro

Runtime studies	Algorithmic studies
Scalability with no faults (strong and weak)	Task-granularity and decomposition
Performance in the presence of faults	Classification of performance according to compute/communication ratios
Node degradation tests	Algorithmic tradeoffs
Comparison against baseline MPI skeleton	Matrix assembly variants made possible by shift to many-task model

- Demonstration of full-scale implementation of run-time and associated mini-apps on capability-class system next year