SAND2014-1287P

Fault-Tolerant Computing at Exascale
– A Quiet Revolution in Progress –

Robert L. Clay, Ph.D.
Manager, Scalable Modeling and Analysis Systems
Sandia National Laboratories

CMU Seminar
Feb 19, 2014
Pittsburgh, PA

Sandia National Laboratories is a multi-program laboratory operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin company, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.

Sandia National Laboratories

First…

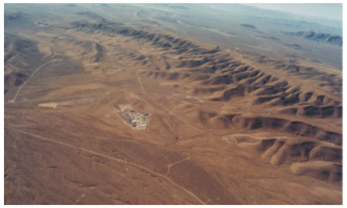
A little background about Sandia National Labs

Sandia National Laboratories: a mission-driven, multi-program laboratory

~8400 employees
>11,000 people
~ 1500 Ph.D. staff
~$2.4B budget

Albuquerque, New Mexico

Livermore, California



Yucca Mountain, Nevada

WIPP, New Mexico

Kauai Test Facility, Hawaii

Pantex, Texas

Tonopah Test Range, Nevada

Vision: Sandia is the provider of innovative, science-based, systems-engineering solutions to our Nation's most challenging national security problems.

Robert L. Clay, CMU-2014

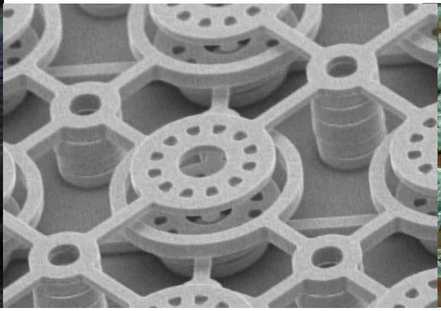
Sandia is a key U.S. government research and development laboratory

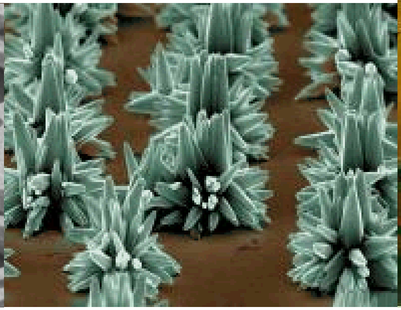
- **Core Purpose:** Help our nation secure a peaceful and free world through technology
- **Corporate mission statement:** Exceptional service in the national interest
- **Key mission areas:**
  - Nuclear Weapons
  - Defense Systems
  - Energy
  - Nonproliferation
  - Homeland Security

Science-Based Engineering

Strate

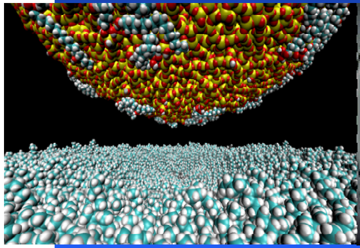
High-Performance Computing


Microsystems


Nanotechnology

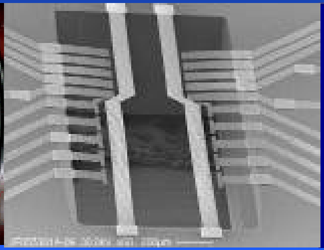
Extreme Environments

Materials

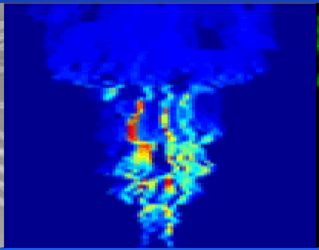


Computer Science

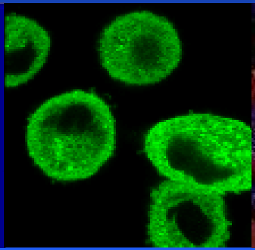
Micro Electronics



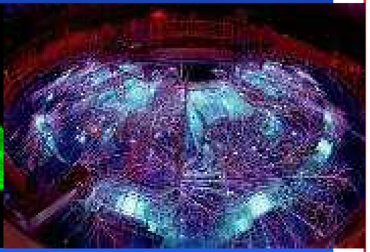
Engineering Science



Bio



Pulsed Power



Research Foundations

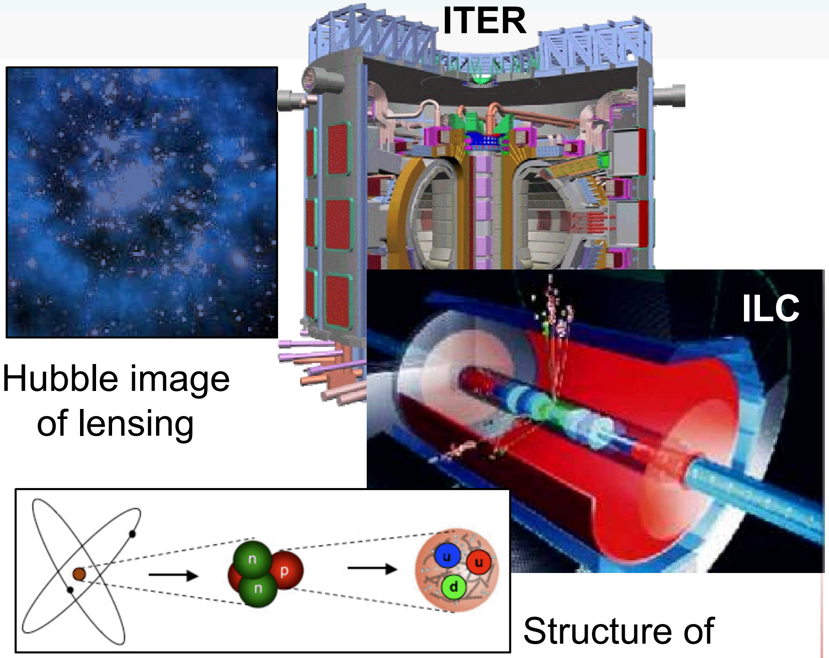
Back to the exascale problem…

Science and engineering are the real drivers.

Exascale simulation will enable fundamental advances in basic science

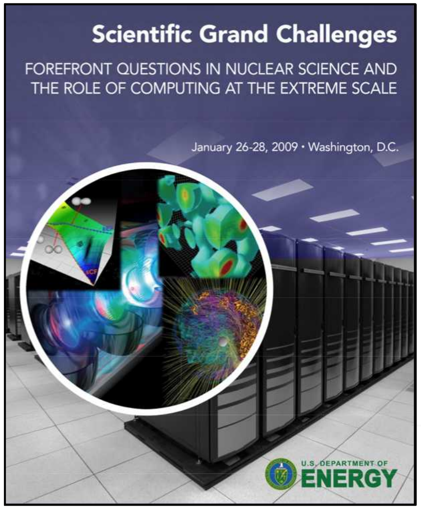
- **High Energy & Nuclear Physics**
  - **Dark-energy and dark matter**
  - **Fundamentals of fission fusion reactions**
- **Facility and experimental design**
  - **Effective design of accelerators**
  - **Probes of dark energy and dark matter**
  - **ITER shot planning and device control**
- **Materials / Chemistry**
  - **Predictive multi-scale materials modeling: observation to control**
  - **Effective, commercial technologies in renewable energy, catalysts, batteries and combustion**
- **Life Sciences**
  - **Better biofuels**
  - **Sequence to structure to function**

These breakthrough scientific discoveries and facilities require exascale applications and resources.



Hubble image of lensing

Structure of nucleons

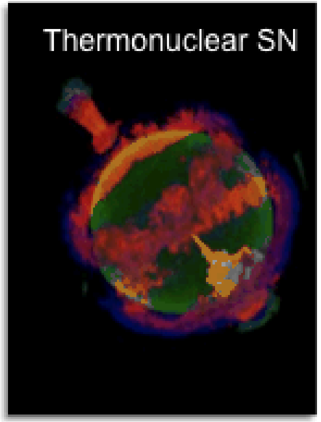


Scientific Grand Challenges

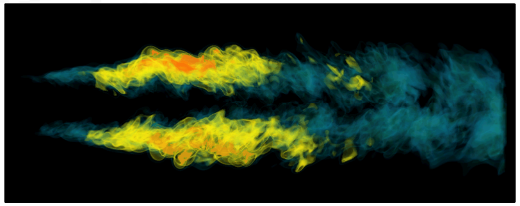
FOREFRONT QUESTIONS IN NUCLEAR SCIENCE AND THE ROLE OF COMPUTING AT THE EXTREME SCALE

January 26-28, 2009 • Washington, D.C.

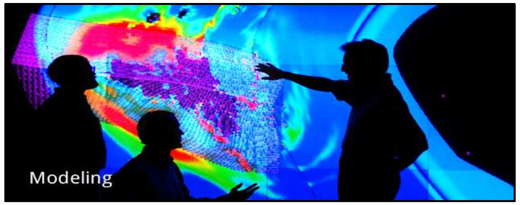
U.S. DEPARTMENT OF ENERGY



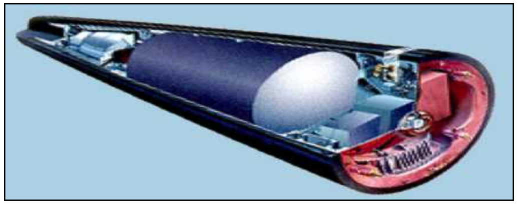
Leadership-class HPC compute capabilities are required for DOE policy and decision making



Energy: Reduce U.S. reliance on foreign energy, reduce carbon footprint



Climate change: Understand, mitigate, and adapt to the effects of global warming

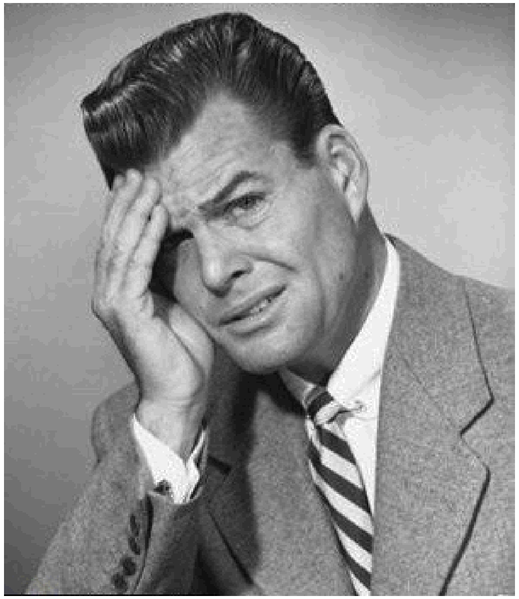


National Nuclear Security: Maintain a safe, secure, and reliable nuclear stockpile

Exascale computing and beyond is required to simulate complex phenomena that characterize the DOE mission space

Exascale computing presents serious technical challenges

How am I going to scale my codes to exascale?



Key Technical Challenges

- DOE's Exascale Initiative Steering Committee and DARPA identified technology gaps that need to be addressed to reach Exascale later this decade
  - Power, memory and storage, parallelism and locality, resilience, scalability, programming models

- Co-development (or co-design) of hardware, system software and applications is a key element of our strategy
  - Codes will need to adapt to manage billion-way parallelism, data locality, resilience and perhaps energy

What is HPC Resilience?

- **We define resilient HPC as *correct and efficient computations at scale despite system degradations and failures*.**

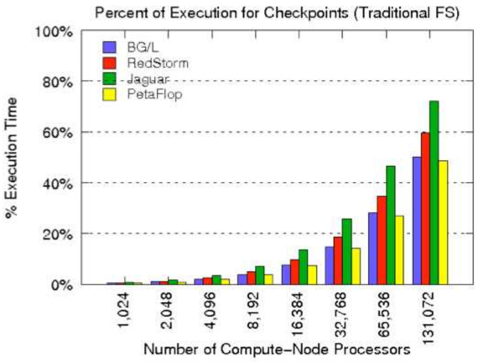
- **Resilience is a cross cutting issue:**
  - **Hardware**
  - **Operating System**
  - **System Management**
  - **Runtime (Execution Model)**
  - **Application / Algorithms**
  - **Multi-layer (any/all combinations of the above)**

Why does resilience matter?

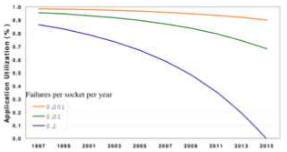
- "Without research into new fault management techniques and the development of supporting resilience technologies, DOE's mission critical applications may not be able to run to completion, or worse, will complete but get the wrong result due to undetected errors". — US DOE Fault Management Workshop Final Report, August 13, 2012.
- "One of the main roadblocks to exascale is the likelihood of much higher error rates, resulting in systems that fail frequently and make little progress in computations or in systems that may return erroneous results. Although such systems might achieve high nominal performance, they would be useless". — Addressing Failures in Exascale Computing, ANL-MCS-TM-332, March 31, 2013.

Computers are Reliable Digital Machines, Aren’t They?

Checkpoint trend isn’t good

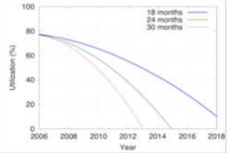


Oldfield et al., *Modeling the Impact of Checkpoints on Next-Generation Systems.* MSST, 2007



Effective application utilization (including checkpoint overhead) at 3 rates of hardware failure

(Courtesy of Lucy Nowell & Sonia Sachs)

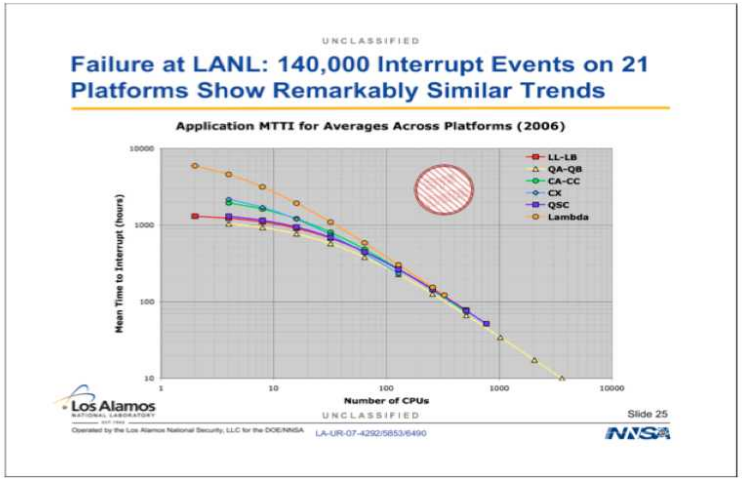


Schroeder and Gibson, *Understanding Failures in Petascale Computers.* Journal of Physics, 2007

(assuming that the number of cores per socket grows by a factor of 2 every 18, 24 and 30 months)

Machine utilization is going to zero! (Not really)

MTTI is shrinking as # cores grows



UNCLASSIFIED

Failure at LANL: 140,000 Interrupt Events on 21 Platforms Show Remarkably Similar Trends

Application MTTI for Averages Across Platforms (2006)

Los Alamos
NATIONAL LABORATORY

UNCLASSIFIED

Slide 25

Operated by the Los Alamos National Security, LLC for the DOE/NNSA

LA-UR-07-4292/5853/6490

NNSA

(Courtesy of John Daly)

Outlook has improved (e.g., on-node NVM, improved CP/R), but still not reasonable to assume the systems will be reliable and static.

Robert L. Clay, CMU-2014

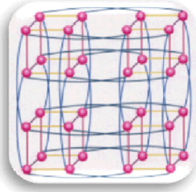
LET’S REVIEW KEY ASSUMPTIONS

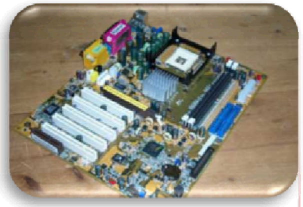
Are future computers similar to today’s?

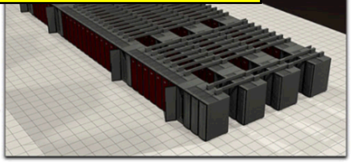
Some Current HPC Assumptions

	Current	New
Nodes	**Persist** for duration of job	**Will fail,** and so will other HW
Hardware	Can build **sufficiently reliable**	Too **expensive / impractical**
Pr **M**	CSP BSP (MPI)	
Machines	Capability **fundamentally different** than capacity	Capability **=** capacity?

We need to rethink the problem, and the solution!







Robert L. Clay, CMU-2014

So, what are we going to do about it?

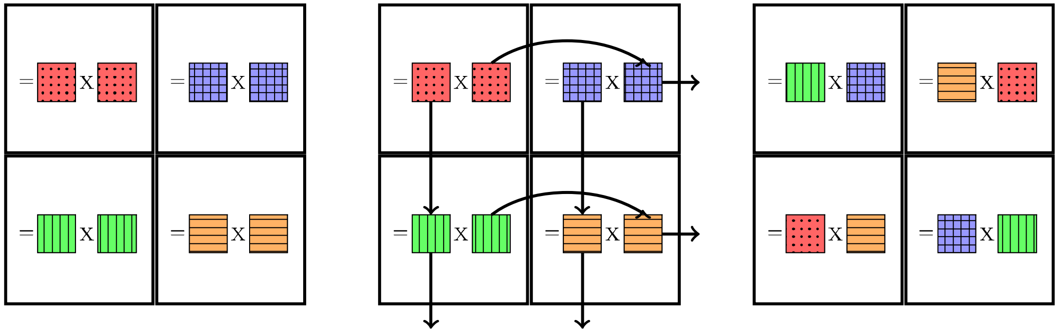
- First, analysis to understand the problem (and co-design).
  - SST/Macro performance simulator (Wilke et al)
- Scalable defensive programming (LFLR).
  - Local fail, local restart (Heroux and Teranishi)
- Ref on.
  - R
- Alternative, fault-tolerant programming models.
  - Pmodels AMT FT programming model (Slattengren et al)

Scalable, fault-tolerant computing is the goal.

Analysis of HPC System Performance

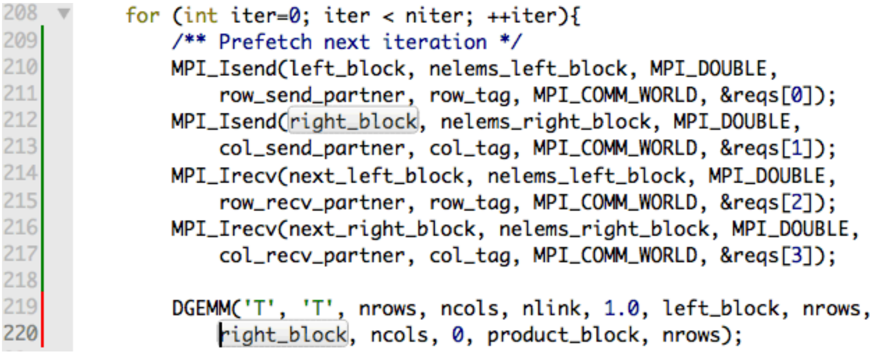
Scalability and resilience studies with SST/macro

Programming model exploration for resilience with simulation



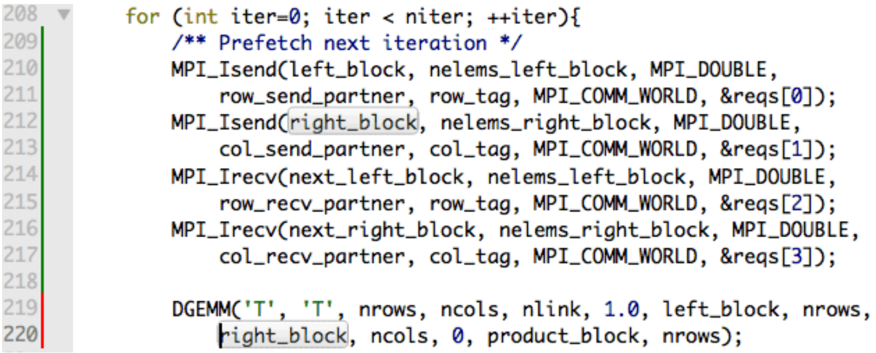
Systolic matrix-matrix multiplication involves “synchronous” migration of matrix blocks.
Start with MPI.

Actual MPI code



```
for (int iter=0; iter < niter; ++iter){
    /** Prefetch next iteration */
    MPI_Isend(left_block, nelems_left_block, MPI_DOUBLE,
        row_send_partner, row_tag, MPI_COMM_WORLD, &reqs[0]);
    MPI_Isend(right_block, nelems_right_block, MPI_DOUBLE,
        col_send_partner, col_tag, MPI_COMM_WORLD, &reqs[1]);
    MPI_Irecv(next_left_block, nelems_left_block, MPI_DOUBLE,
        row_recv_partner, row_tag, MPI_COMM_WORLD, &reqs[2]);
    MPI_Irecv(next_right_block, nelems_right_block, MPI_DOUBLE,
        col_recv_partner, col_tag, MPI_COMM_WORLD, &reqs[3]);

    DGEMM('T', 'T', nrows, ncols, nlink, 1.0, left_block, nrows,
        right_block, ncols, 0, product_block, nrows);
```

Simulator code



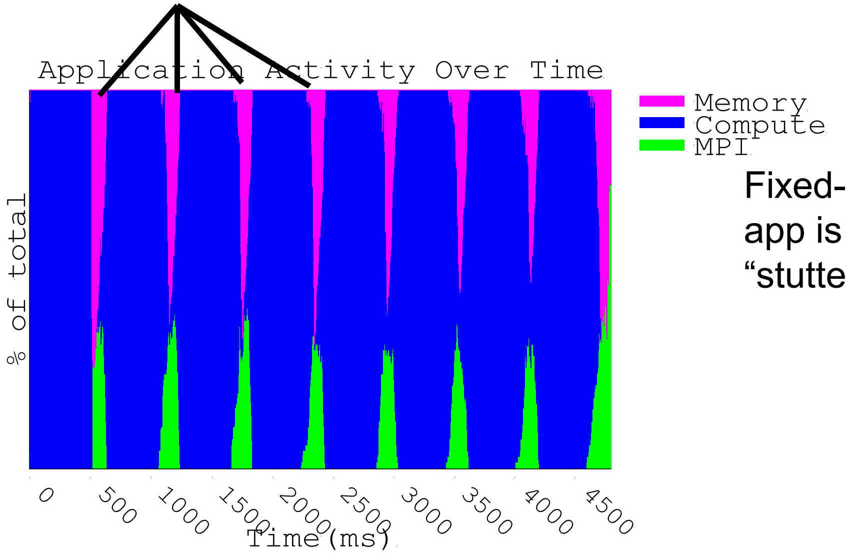
```
for (int iter=0; iter < niter; ++iter){
    /** Prefetch next iteration */
    MPI_Isend(left_block, nelems_left_block, MPI_DOUBLE,
        row_send_partner, row_tag, MPI_COMM_WORLD, &reqs[0]);
    MPI_Isend(right_block, nelems_right_block, MPI_DOUBLE,
        col_send_partner, col_tag, MPI_COMM_WORLD, &reqs[1]);
    MPI_Irecv(next_left_block, nelems_left_block, MPI_DOUBLE,
        row_recv_partner, row_tag, MPI_COMM_WORLD, &reqs[2]);
    MPI_Irecv(next_right_block, nelems_right_block, MPI_DOUBLE,
        col_recv_partner, col_tag, MPI_COMM_WORLD, &reqs[3]);

    DGEMM('T', 'T', nrows, ncols, nlink, 1.0, left_block, nrows,
        right_block, ncols, 0, product_block, nrows);
```

With a few linker tricks, you get direct compilation of source code. No DSL! Only one source to maintain!

Programming model exploration for resilience : simulator results

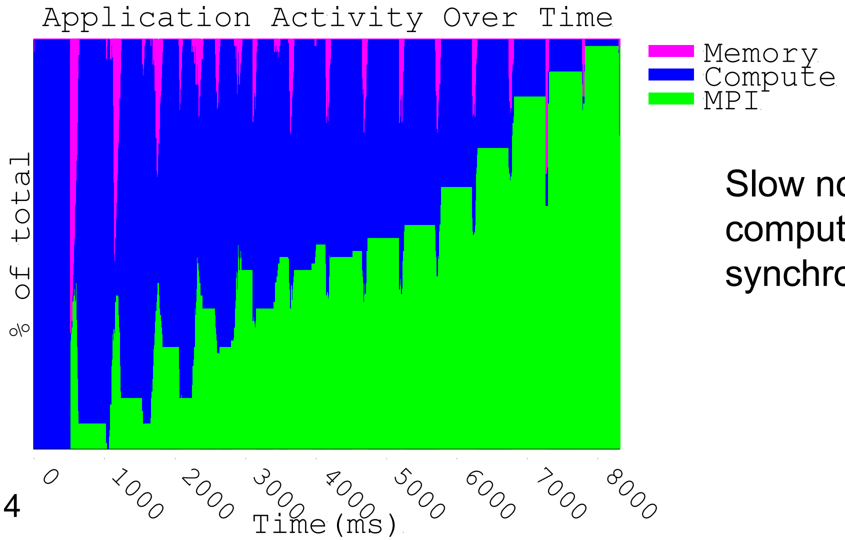
Synchronous MPI data exchange

If all nodes the same speed…



Fixed-time quanta (FTQ) shows where app is spending time. Here MPI “stutters” during synchronous exchange

If one node overheats or has bad DIMM and slows down…

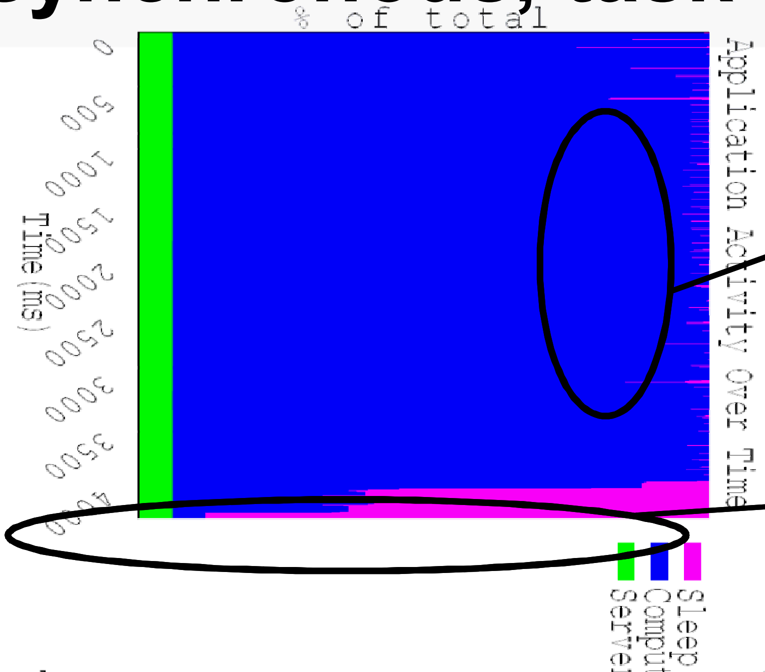


Slow node gradually chokes off computation due to MPI synchronization…

Robert L. Clay, CMU-2014

Programming model exploration: Sandia asynchronous, task-DAG model

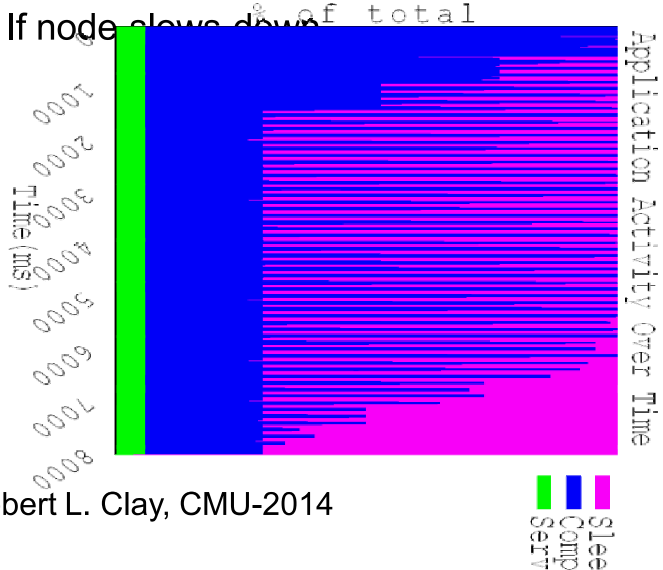
If all nodes the same speed…



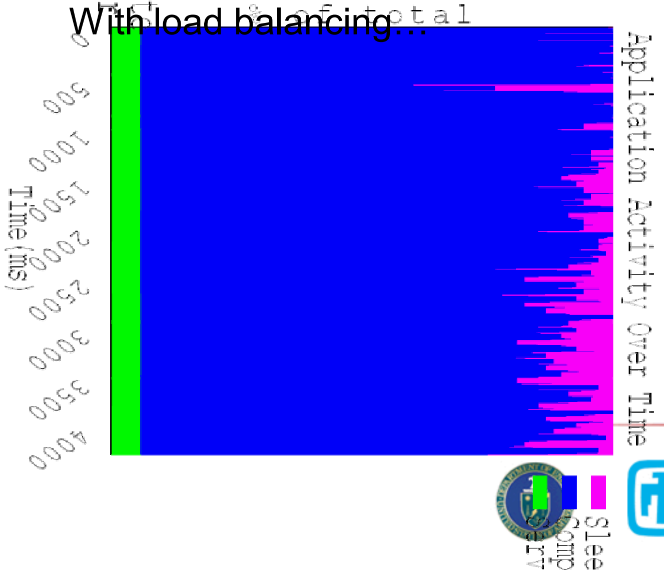
Termination detection/ work stealing needs to be
optimized

Data movement service is constant overhead – single thread dedicated to communication

If node slows down…



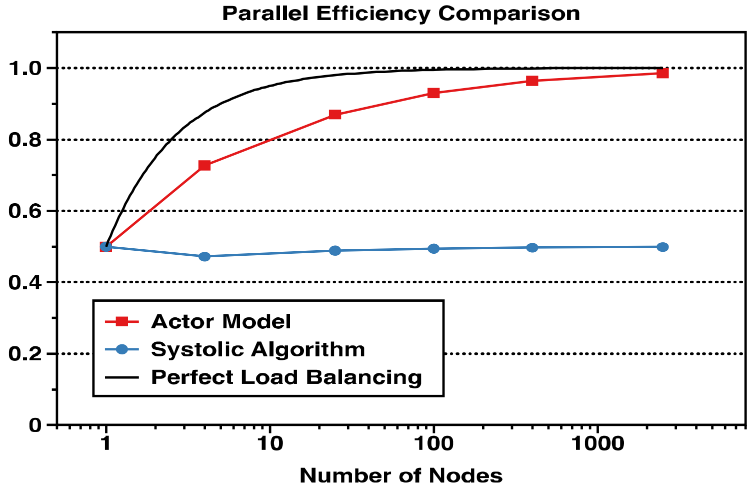
With load balancing…



Asynchronous many-task programming models are fault tolerant!

- Simulation permits straightforward investigation of alternative programming models
- Work-stealing approaches will play a role in dealing with large-scale machines lacking perfect homogeneity

Actor Model Matrix Multiplication (asynchronous, many task)



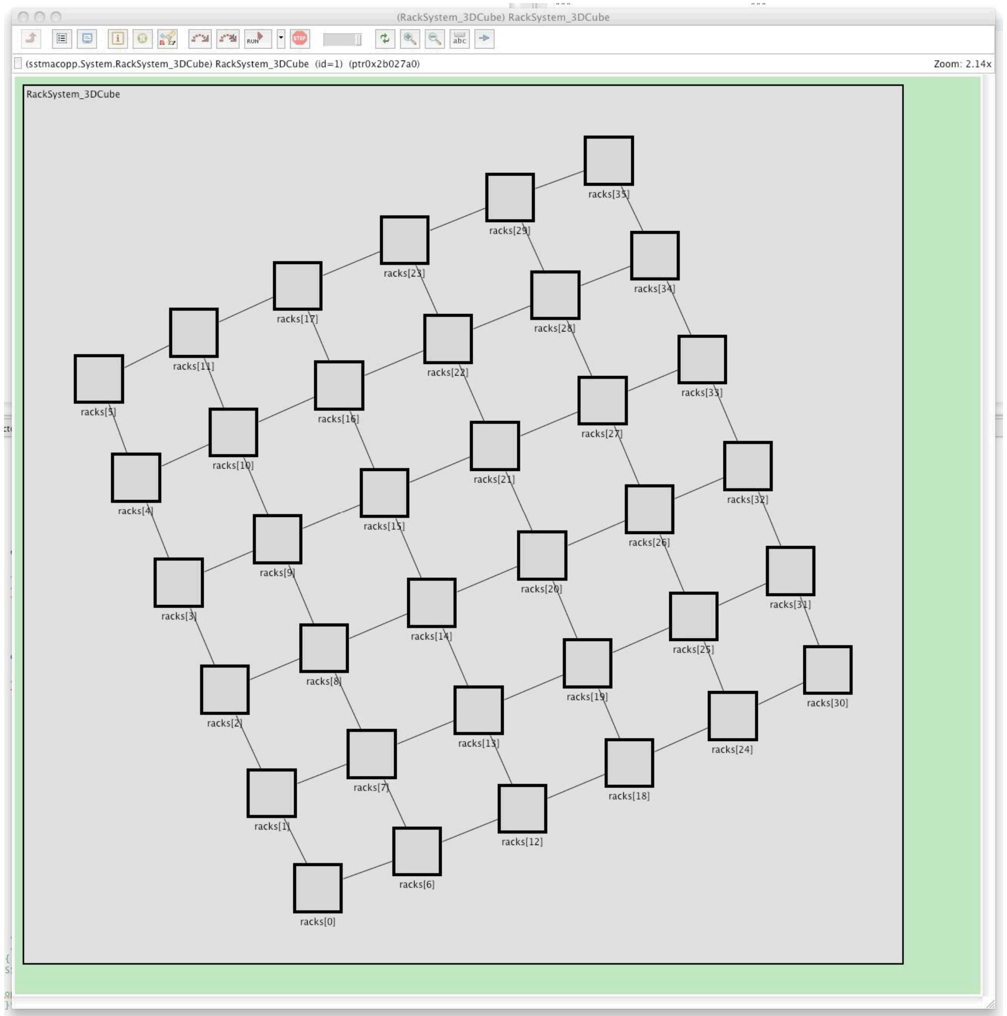
- Research Questions:
  - Is MPI+X (*global* checkpoint/restart) enough?
  - If not, what programming models can reach what scales?
  - If no programming model can reach scales of interest for a given application without algorithmic changes, how might algorithms be adapted?
  - Co-design of architecture tradeoffs between memory, I/O, power, and application performance

SST Experiment: Actor Load Balancing

Legend

- Black - iniitializing
- Green – working
- Yellow border – prefetching
- Red – idle
- Purple – work stealing

Asynchronous, task-based programming model with work stealing balances load under dynamic conditions, including faults and degradation.



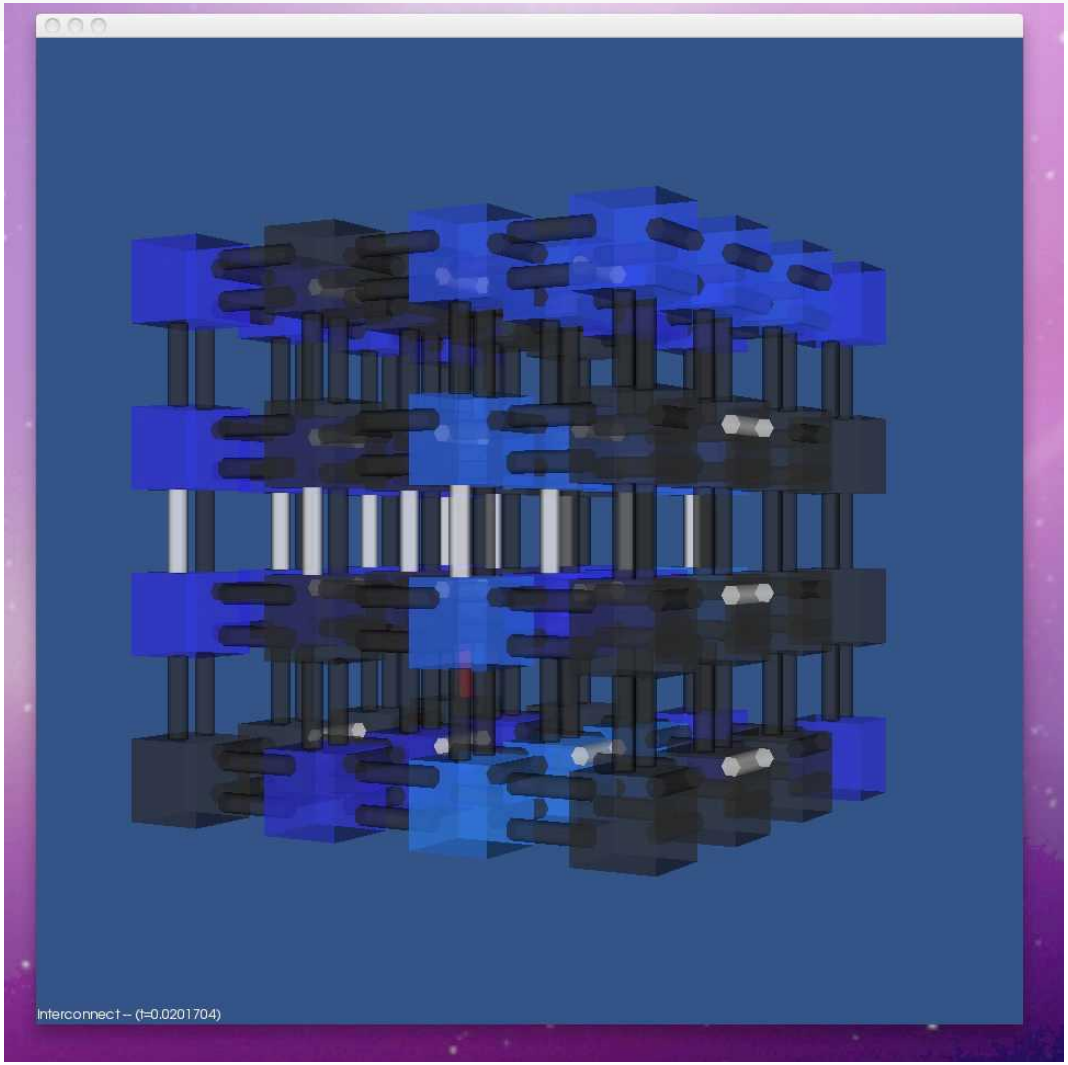
(RackSystem_3DCube) RackSystem_3DCube

(sstmacopp.System.RackSystem_3DCube) RackSystem_3DCube (id=1) (ptr0x2b027a0)

Zoom: 2.14x

RackSystem_3DCube

SST Network Traffic Visualization with VTK



Scalable Defensive Programming

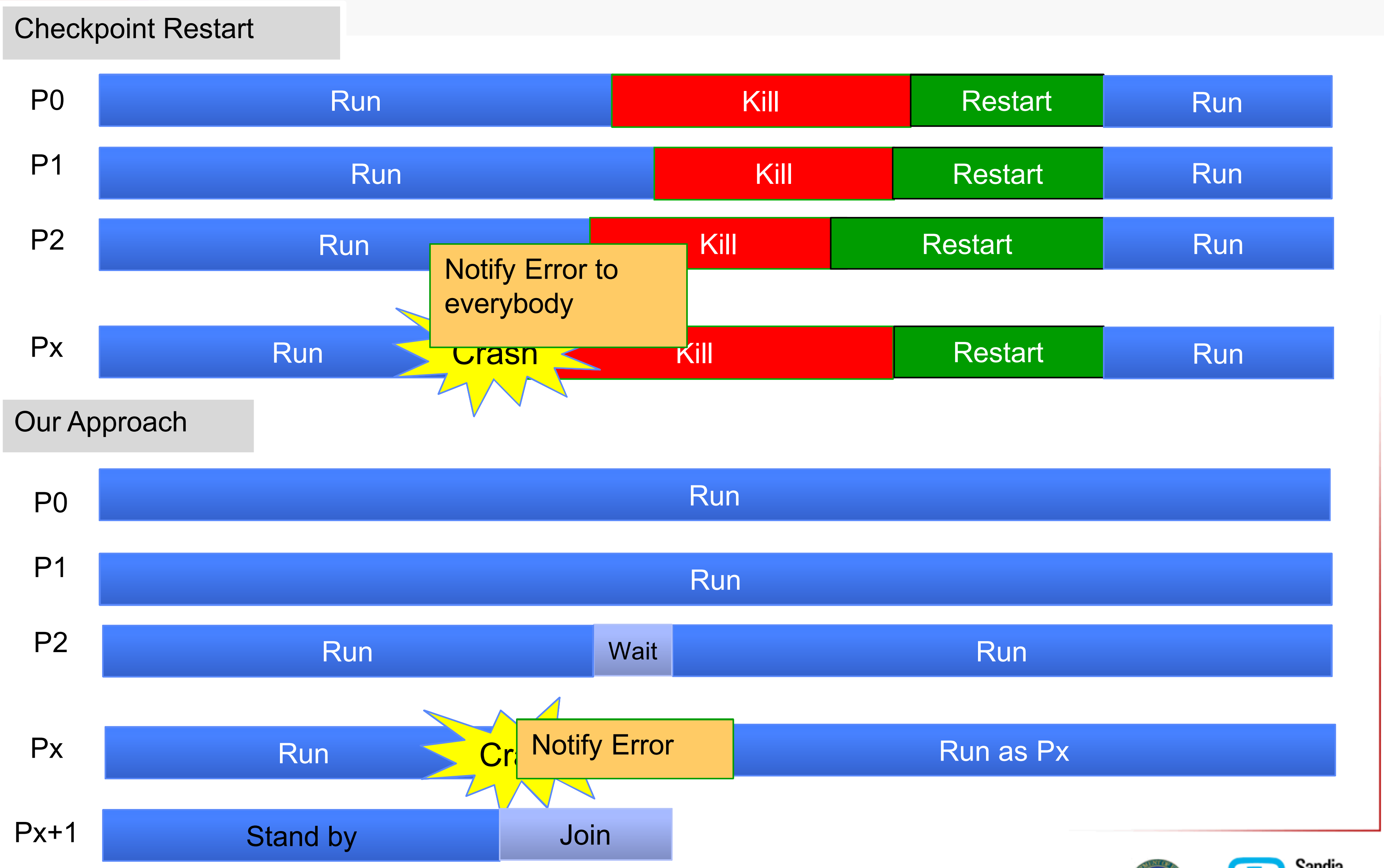
Local Fail, Local Restart – Proportionate Response To Local Failure

Global Checkpoint/Restart: Disproportional Response to Local Failures

- Single node failures account for the major HPC system failures
  - 85% on LLNL clusters (Moody et al. 2010)
  - ~2/3 on Titan (ORNL)
- Short MTBFs due to the increase of error-prone components
  - Titan crashes twice a day
  - Will it get worse?
- Current
  disprop
  - Kill all
  - Recove
  - Depen ate

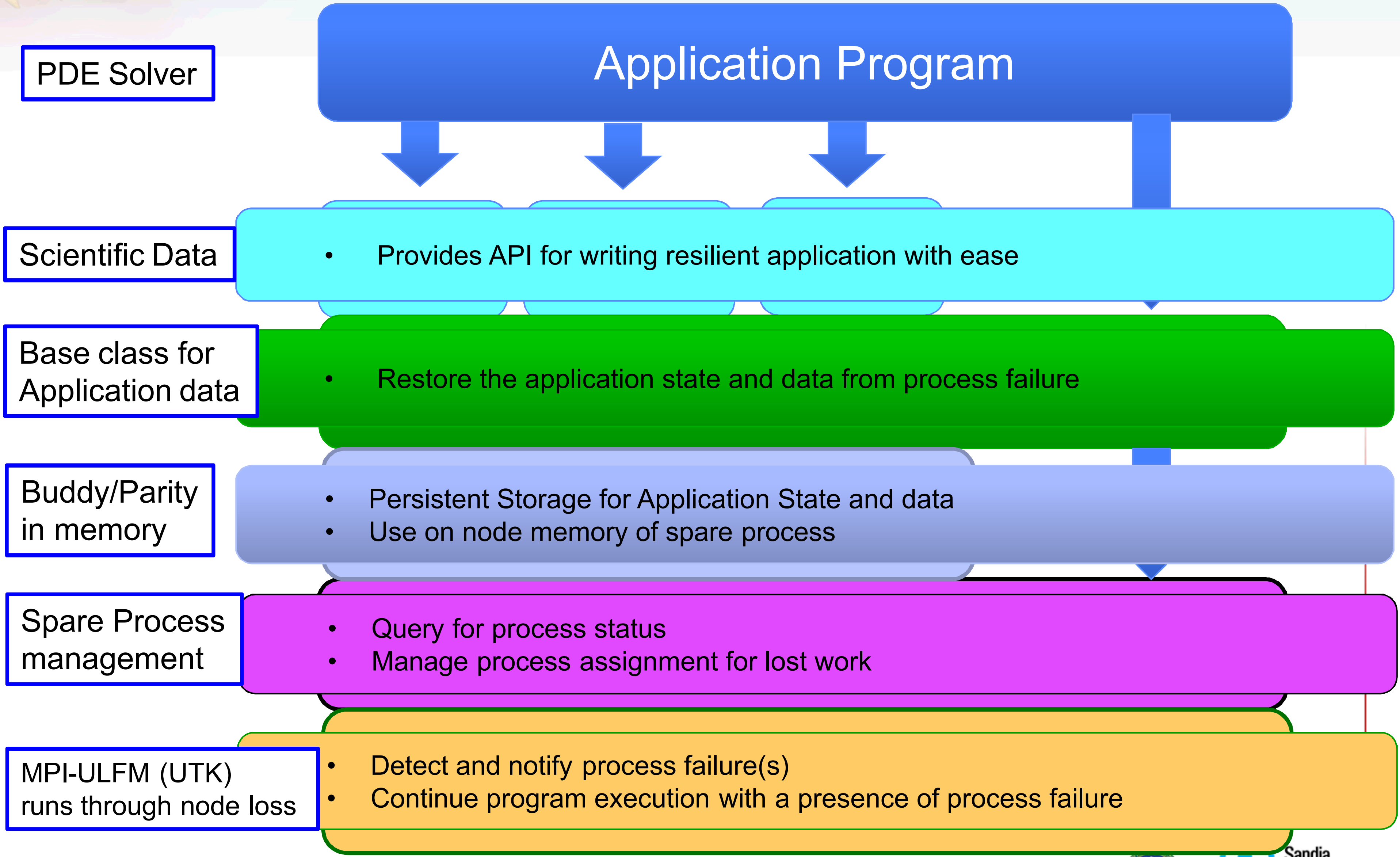
We seek a Local Failure Local Recovery (LFLR) resilient programming model to allow proportional response to single node/process failure.

LFLR Programming Model



Robert L. Clay, CMU-2014

LFLR Architecture



Application Program

PDE Solver

Scientific Data

- Provides API for writing resilient application with ease

Base class for Application data

- Restore the application state and data from process failure

Buddy/Parity in memory

- Persistent Storage for Application State and data
- Use on node memory of spare process

Spare Process management

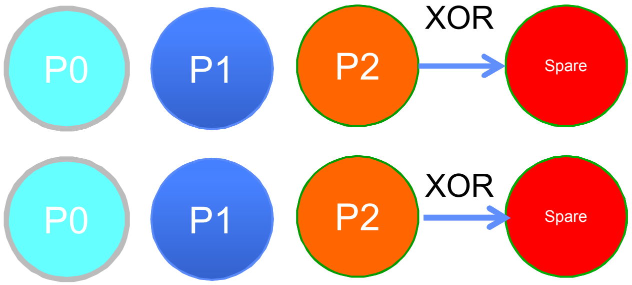
- Query for process status
- Manage process assignment for lost work

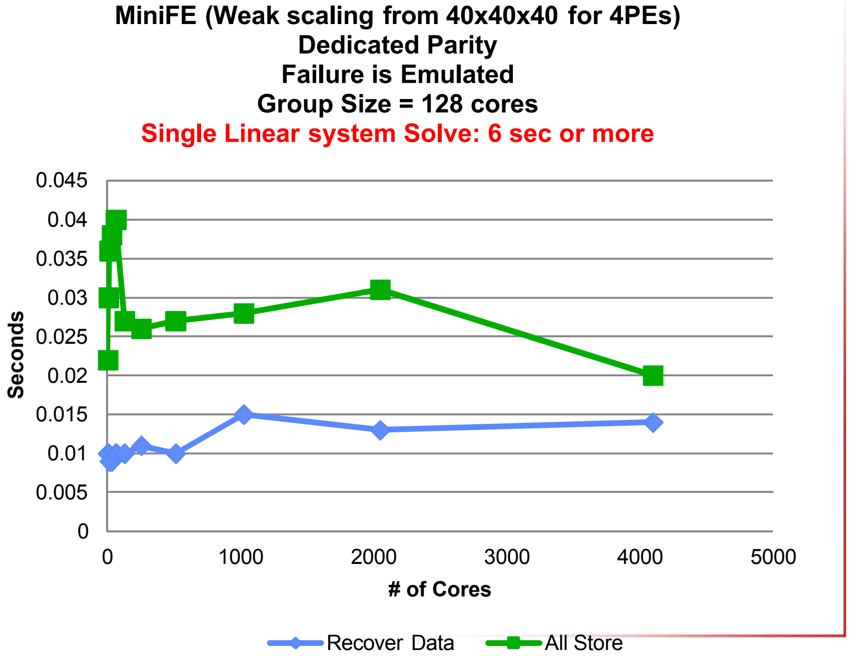
MPI-ULFM (UTK) runs through node loss

- Detect and notify process failure(s)
- Continue program execution with a presence of process failure

LFLR Scalable Recovery

- In-memory checkpoint (persistent-store)
  - Buddy system
    - Duplication of each piece
  - Dedicated Parity
    - Spare processes keep the parity of distributed data
- The data structure is bind to its primary source (application state)
  - Temporary data structure (matrix) is never stored in the storage
    - Created on the fly
  - Reduce the persistent storage size by 90% (or more)
- Performance
  - Fast in in memory persistent data store
  - Good scalability




MiniFE (Weak scaling from 40x40x40 for 4PEs)
Dedicated Parity
Failure is Emulated
Group Size = 128 cores
Single Linear system Solve: 6 sec or more

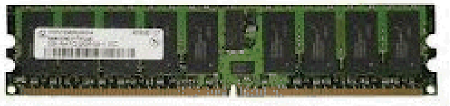
Robert L. Clay, CMU-2014

Reformulating the Problem

“Robust Stencils” – Handling SDC for PDEs

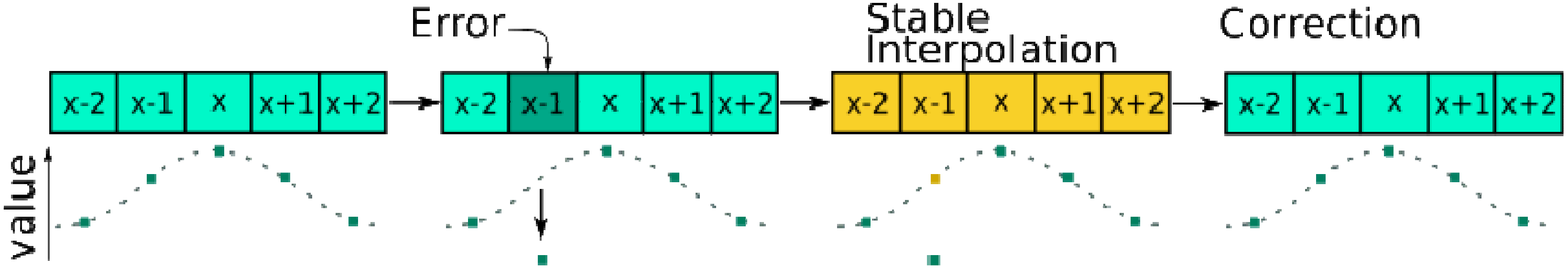
Error-Correcting Algorithms Can Mitigate Silent Errors & Offer New Co-design Options

- Even at commodity scale, ECC memory & ECC processors show the rising need for error correction



ECC memory

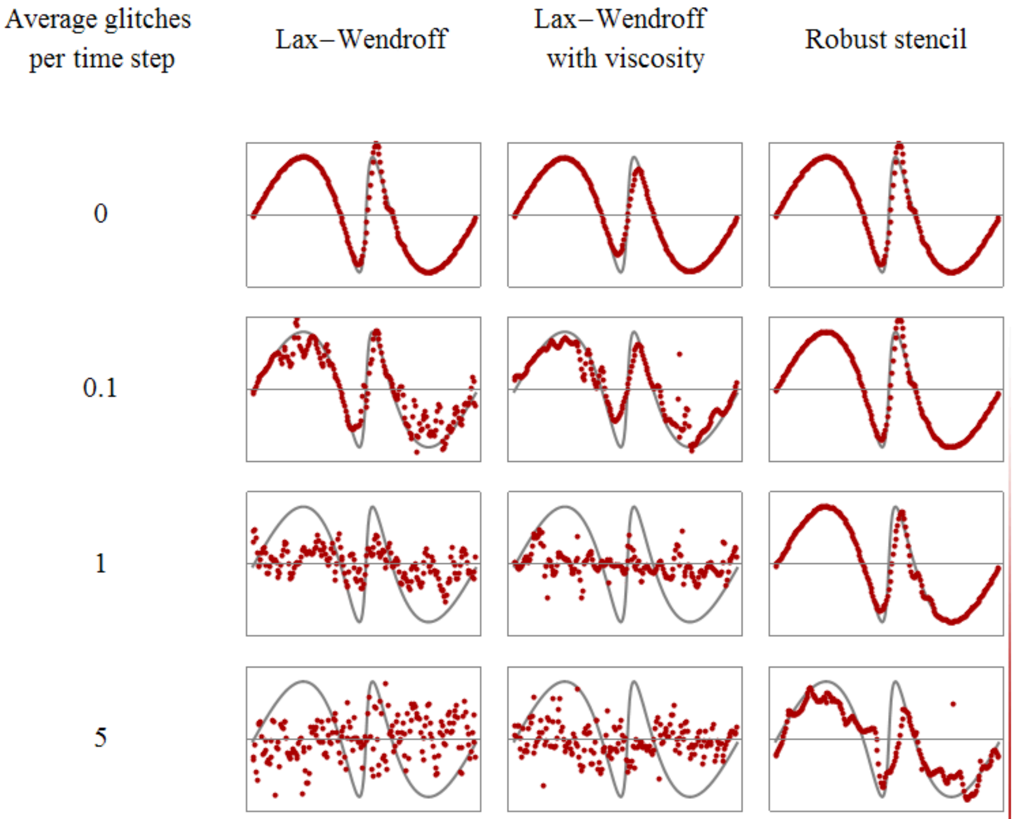
- With increasing scale and with power limitations, errors can occur "silently" without indication that something is wrong
- Numerical algorithms already deal with error from truncation, etc.; specially designed algorithms can mitigate silent bit flips as well



- These robust stencil algorithms not only address scale-up of current silent-error rates, but may enable new "lossy" architecture options with more power-efficient accelerators or reduced latency

Robust stencils can discard outliers to mitigate bit flips in PDE solving

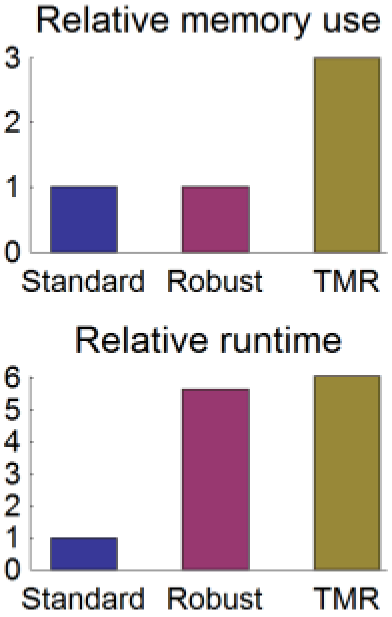
- **A simple 1D advection equation $\partial u/\partial t = \partial u/\partial x$ illustrates the behavior of finite-difference schemes**
- **The robust stencil here computes a second-order u at position *i* from one of these subsets after discarding the most extreme value:**
  - $\{\, i-3,\ i-1,\ i+1,\ i+3 \,\}$
  - $\{\, i-2,\ i,\ i+2 \,\}$
  - $\{\, i-1,\ i,\ i+1 \,\}$

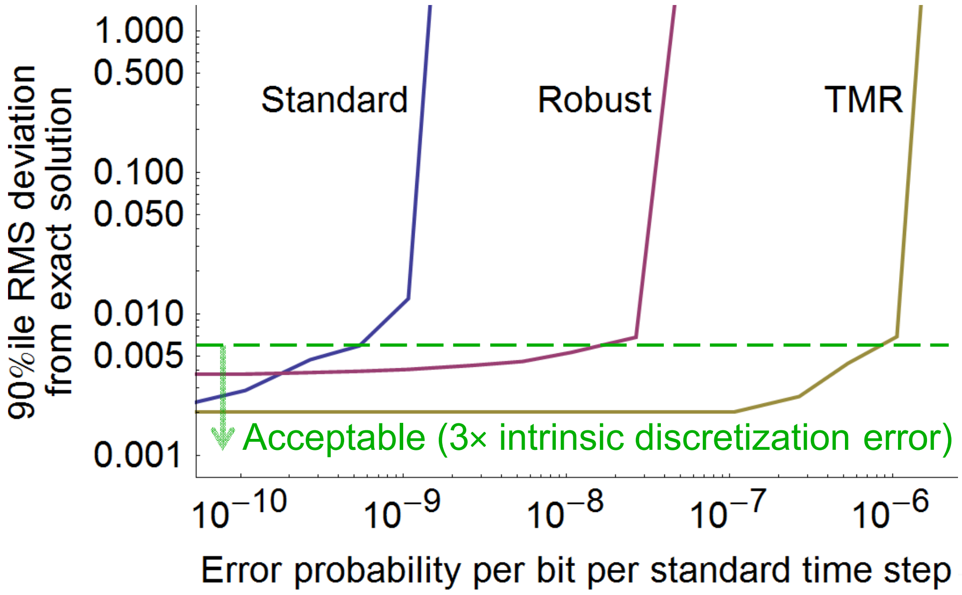


Simple demo in Mathematica

Bit-flip Injection at Machine Level Confirms Effectiveness of Our Robust Stencil

- Focus on silent-error models affecting floating-point
  - Relaxing FP correctness may benefit designs (e.g., GPUs)
- Test: During C++ PDE simulation, asynchronously perform raw memory bit flips in the FP solution array
  - Can also be a proxy for *processor* bit flips that corrupt FP ops
- Compare brute-force triple modular redundancy (TMR)



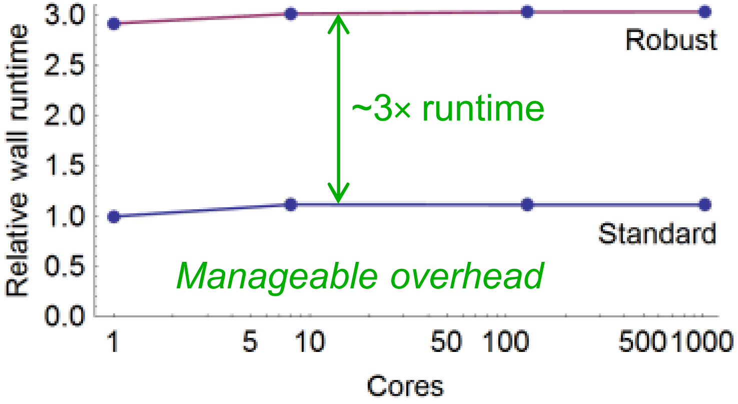


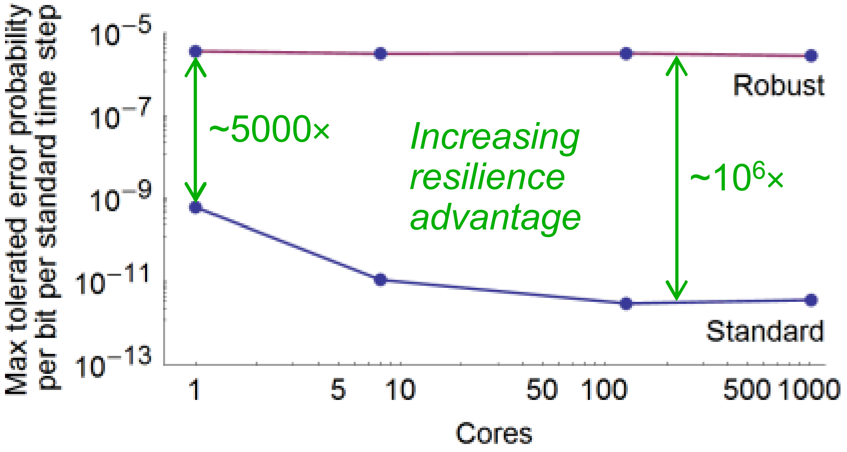
Here, the robust stencil provides substantial bit-flip tolerance at lower cost than TMR

Robert L. Clay, CMU-2014

Preliminary Weak-Scaling Experiments Show Favorable Trends for Robust Stencil

- **As a research tool for ongoing use, we have implemented a modular C++/MPI framework for explicit Cartesian PDE solvers**
  - **Captures "halo exchange" pattern in generic form**
- **Preliminary results from many short runs, 10^6 grid cells per core**





- **Further questions:**
  - How does resilience scale with longer runs and more realistic PDEs?
  - How realistic is our way of emulating memory bit flips?
  - What happens if bit flips also occur in message communication?

Rethinking the Programming Model

Asynchronous, Many-Task Provide Scalability and Fault Tolerance

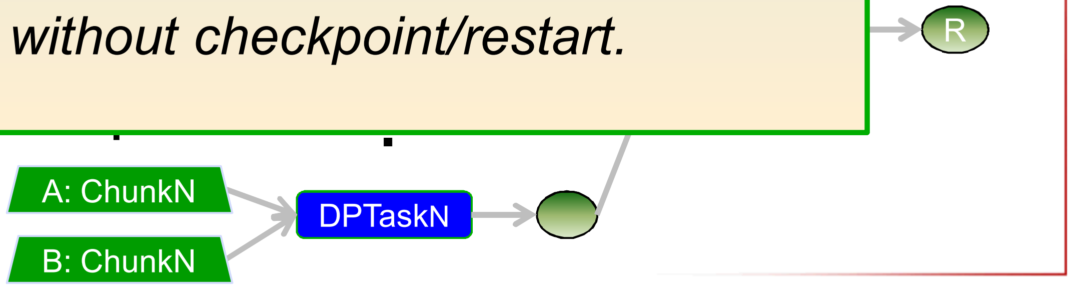
Can asynchronous, many-task programming models facilitate scalable resilience on extreme-scale systems?

- **Our approach:**
  - ***Dynamically scheduled, asynchronous tasks:*** maximize use of resources by load balancing and redistributing work from failed nodes
  - ***Locality and minimal data movement:*** move work to data; multithreaded, NUMA-aware scheduling on each node in distributed environment
  - ***Automatic data repair:*** silent data corruption is detected and repaired using triple modular redundancy or 2D checksums
  - ***Automatic task recovery:*** transaction-like semantics allow task replay after data is corrected

AMT programming models enable marching toward the correct solution in the face of both soft and hard faults without checkpoint/restart.

Example (

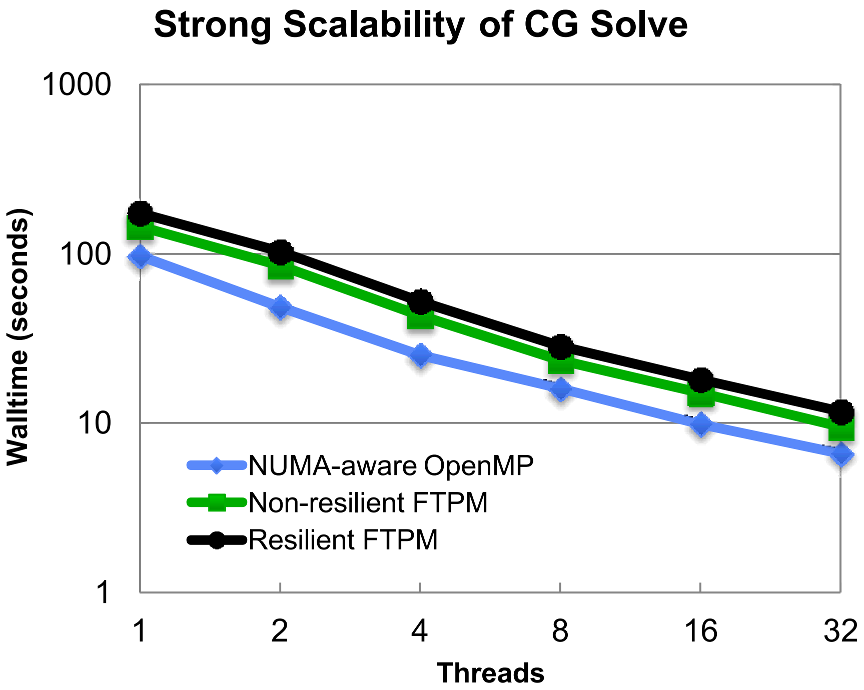
Dot proc
over-dec
and *B* to produce result *R*



Robert L. Clay, CMU-2014

Demonstrated resilience to silent data corruption in our on-node, task-based conjugate gradient solver driven by miniFE proxy app

- *Automatically* detected/corrected multi-bit silent data corruption in user data structures using triple-modular redundancy for scalars and 2D checksums for vectors and matrices (application/algorithm agnostic)



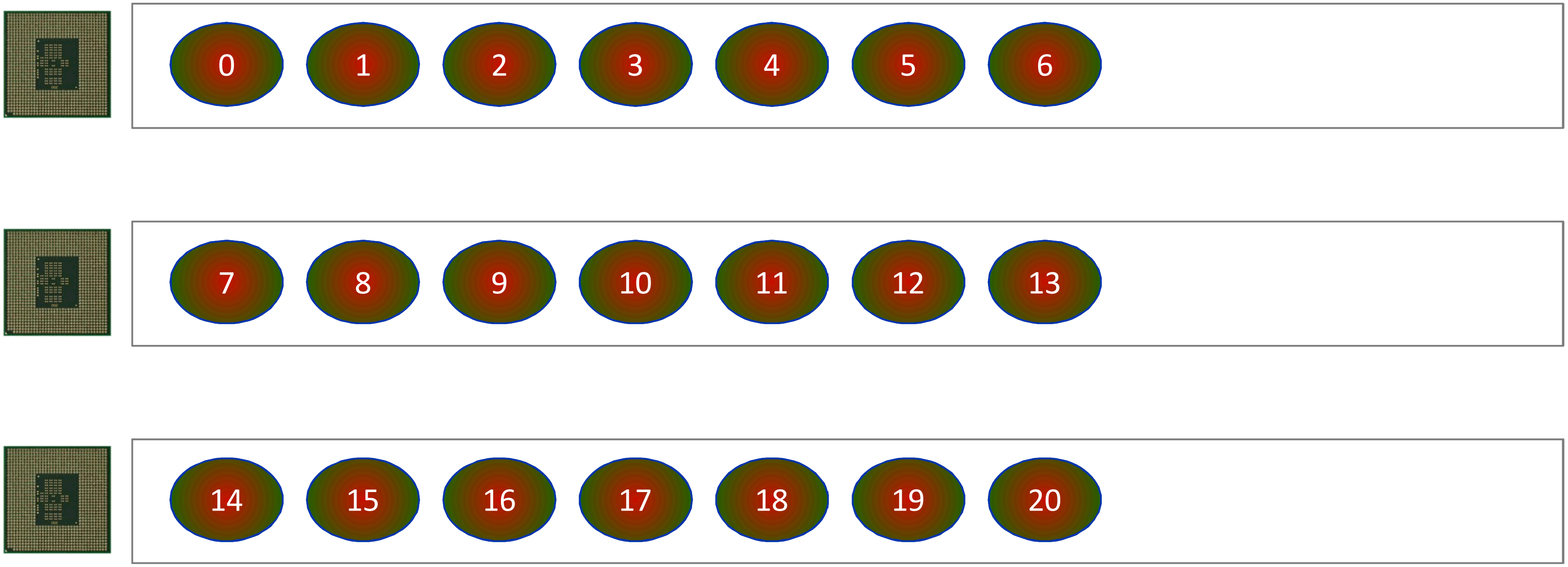
- Technique applied selectively by self-stabilizing CG algorithm in order to lower protection cost
  - 0.8% memory overhead on protected data structures
  - 20% increase in runtime due to checksum validation on every 20th iteration

Benchmarks from SGI Altix UV 10 with four 8-core Nehalem EX and 512 GB globally-shared memory

Task collections are a good first step towards resilient task-based distributed programming

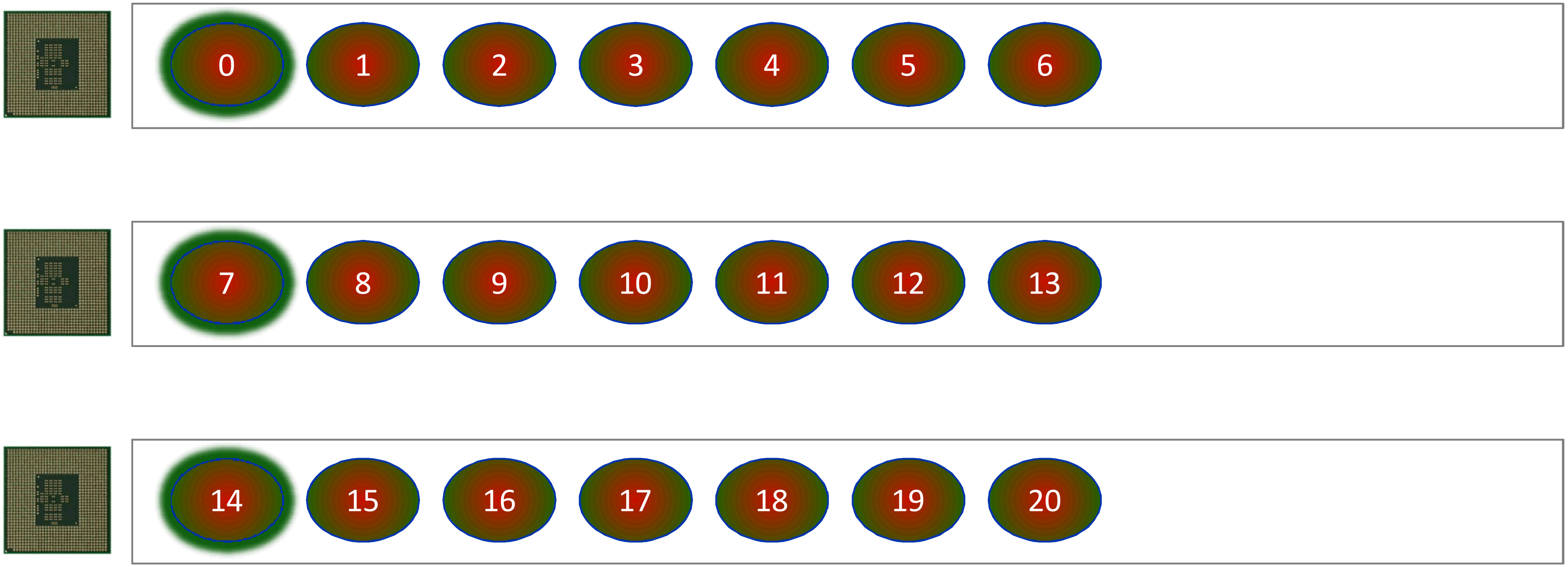
- **Recent work in this area:**
  - Dinan, J., A. Singri, et al. (2010). Selective Recovery from Failures in a Task Parallel Programming Model. 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing (CCGrid): 709-714.
  - Ma, W. and S. Krishnamoorthy (2012). Data-Driven Fault Tolerance for Work Stealing Computations. 26th ACM international conference on Supercomputing. San Servolo Island, Venice, Italy, ACM: 79-90.
- **Key advantages over other techniques:**
  - Maintenance of coherent state at frequency less than MTBF
  - Relatively simple book-keeping

Task collections are a good first step towards resilient task-based distributed programming



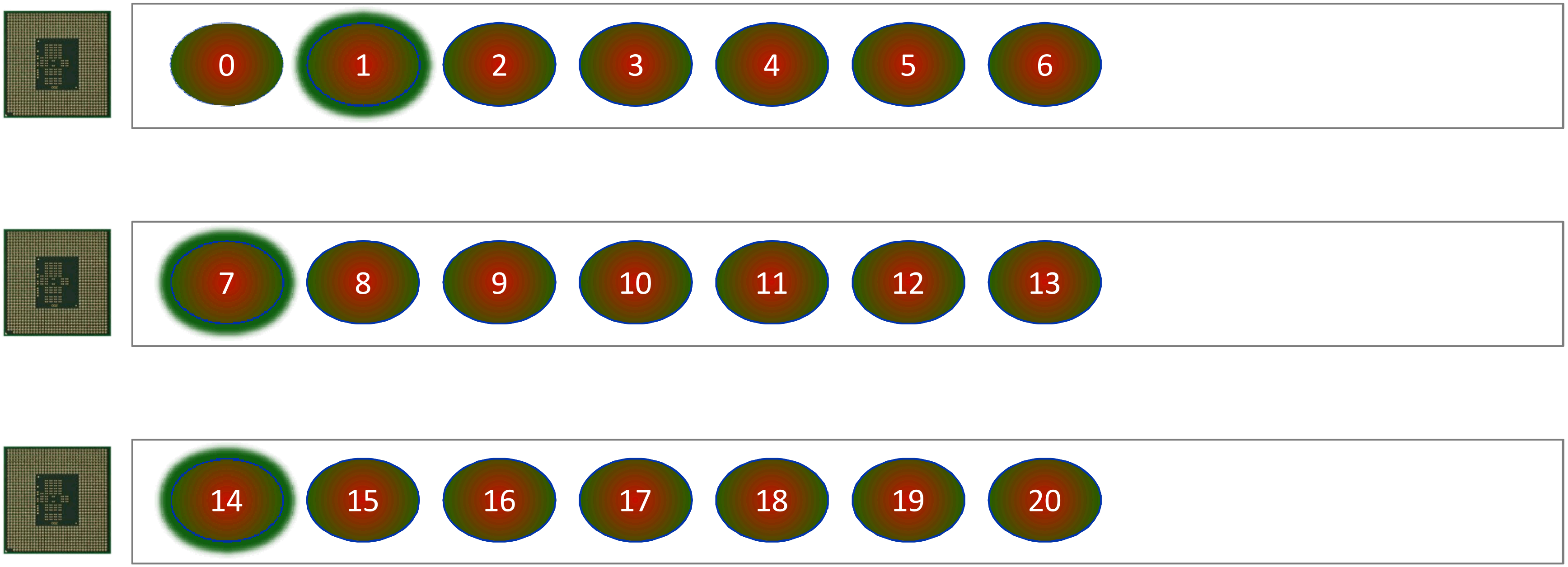
- A single collection executes at a time
- Tasks are independent and distributed across nodes
- A global address space is assumed

Task collections are a good first step towards resilient task-based distributed programming



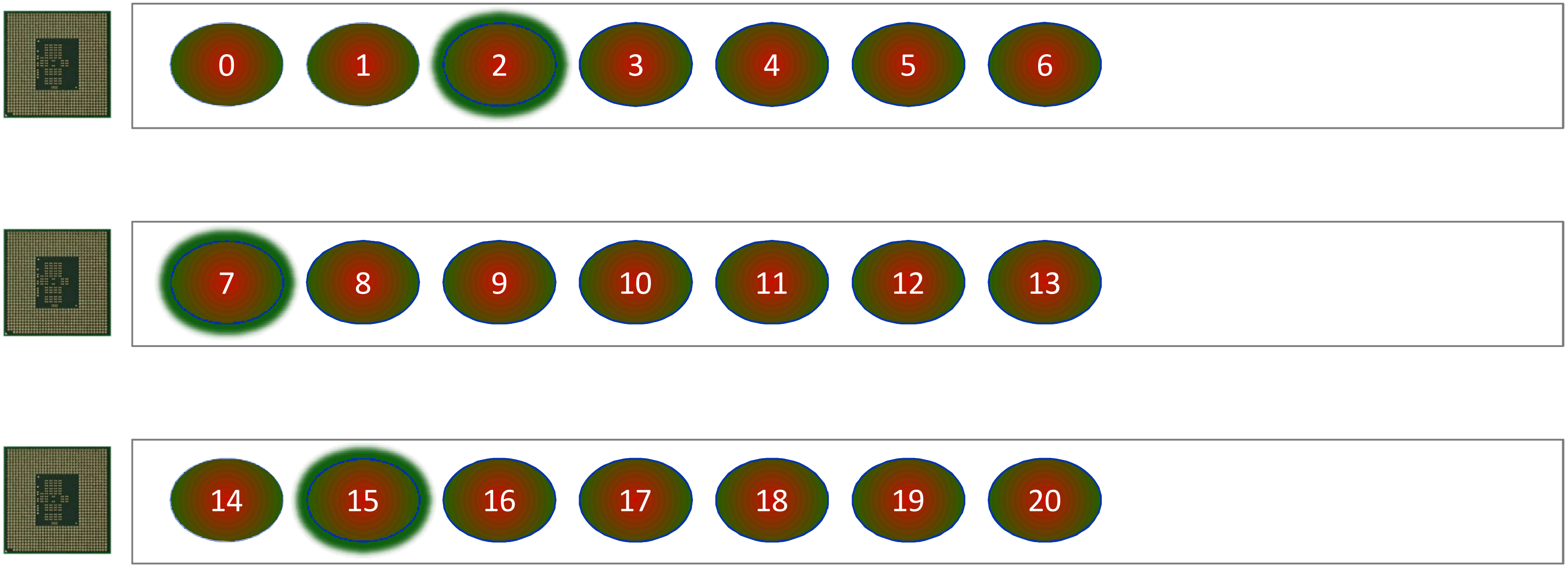
- Within a collection nodes execute tasks asynchronously

Task collections are a good first step towards resilient task-based distributed programming



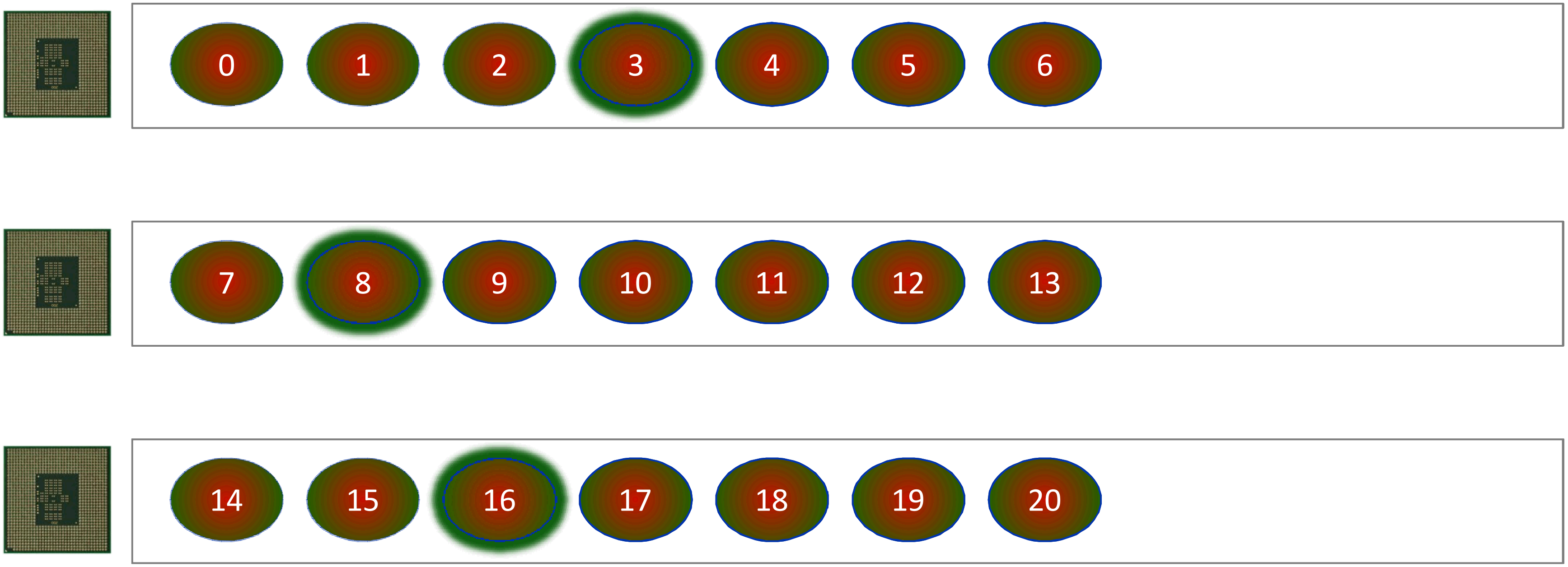
- Within a collection nodes execute tasks asynchronously

Task collections are a good first step towards resilient task-based distributed programming



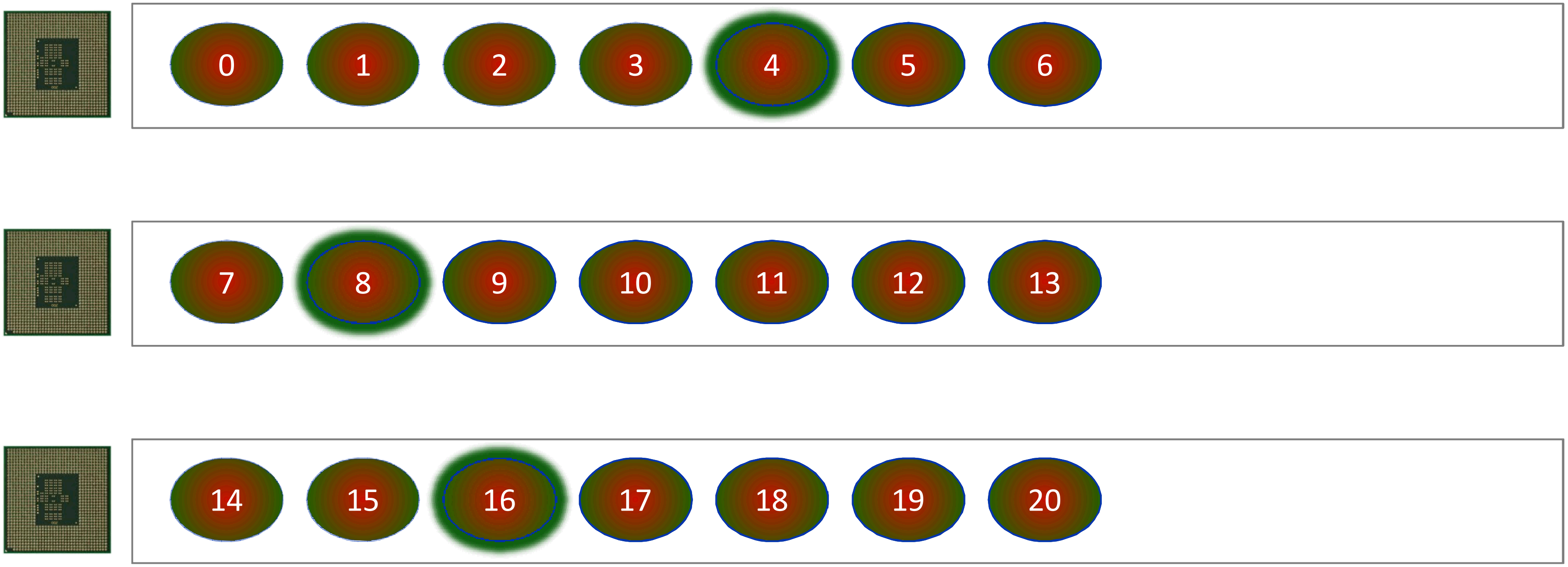
- Within a collection nodes execute tasks asynchronously

Task collections are a good first step towards resilient task-based distributed programming



- Within a collection nodes execute tasks asynchronously

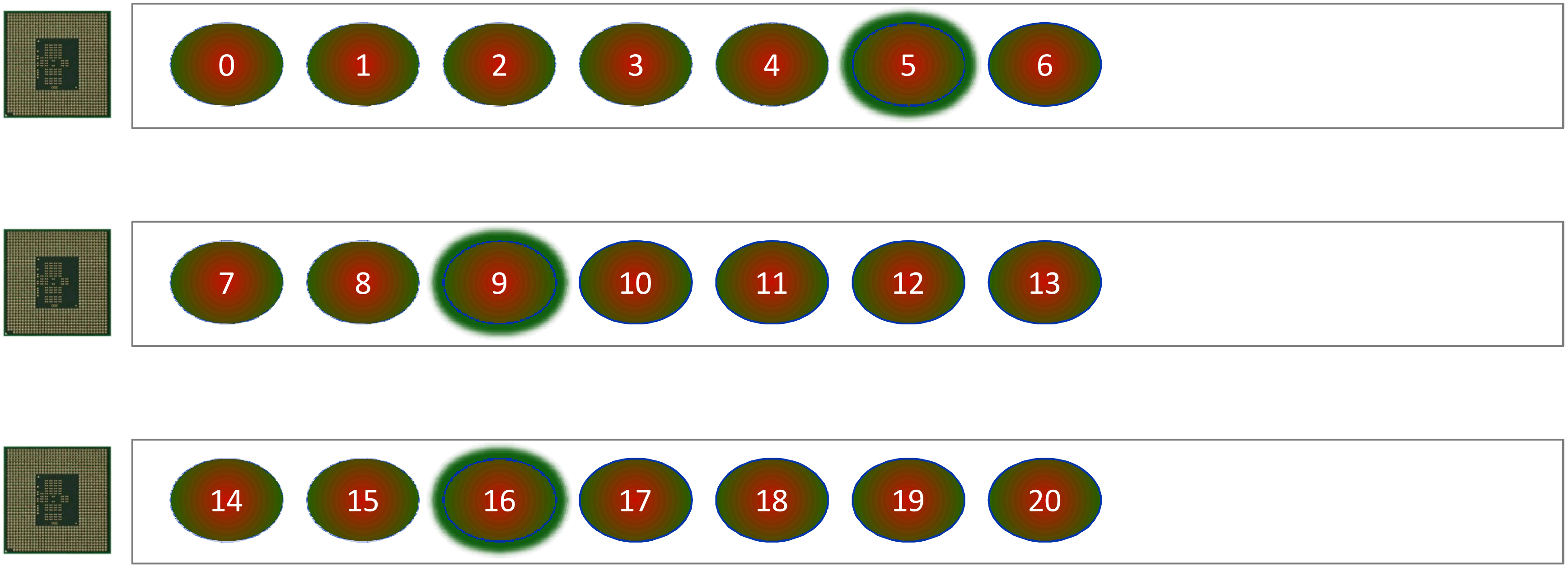
Task collections are a good first step towards resilient task-based distributed programming



- Within a collection nodes execute tasks asynchronously

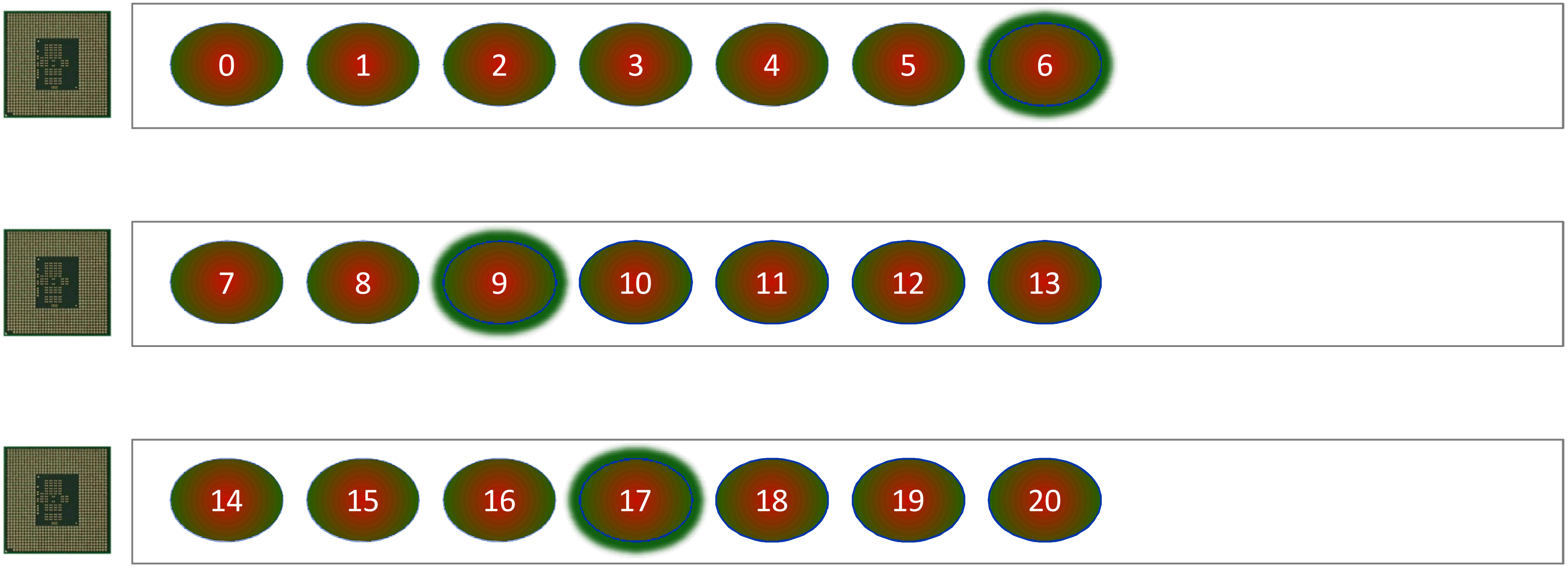
Robert L. Clay, CMU-2014

Task collections are a good first step towards resilient task-based distributed programming



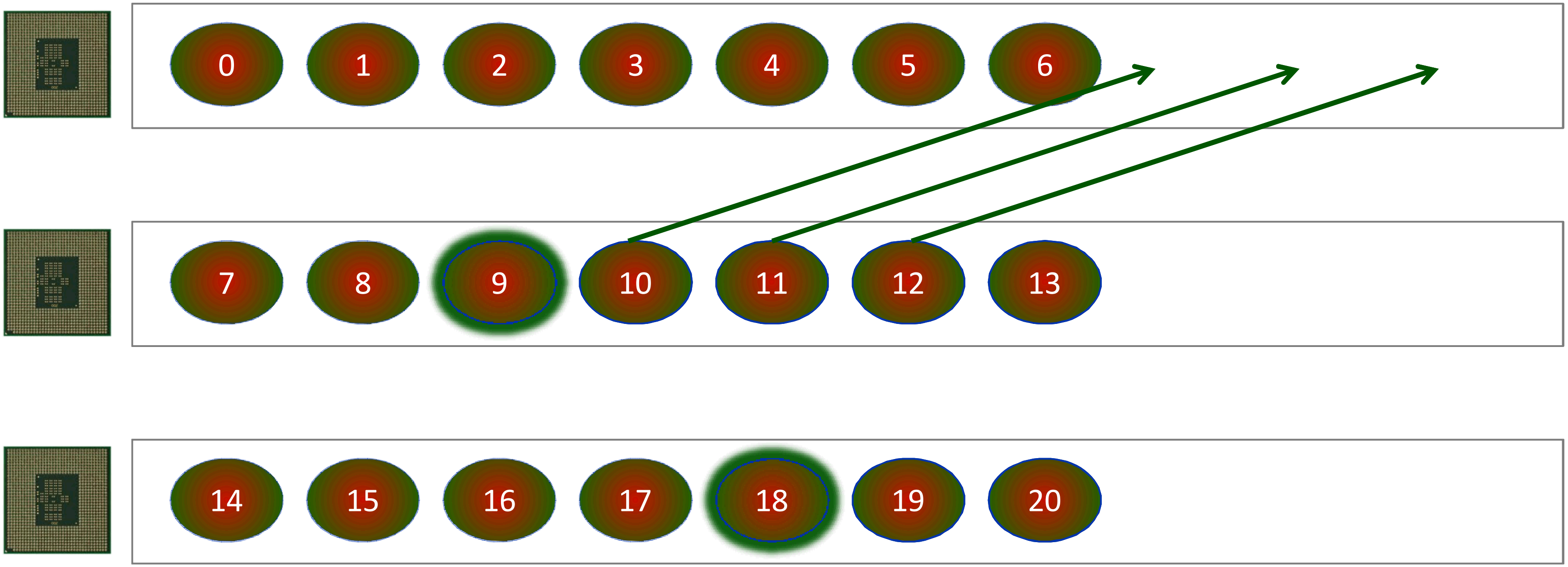
- Within a collection nodes execute tasks asynchronously

Task collections are a good first step towards resilient task-based distributed programming



- Within a collection nodes execute tasks asynchronously

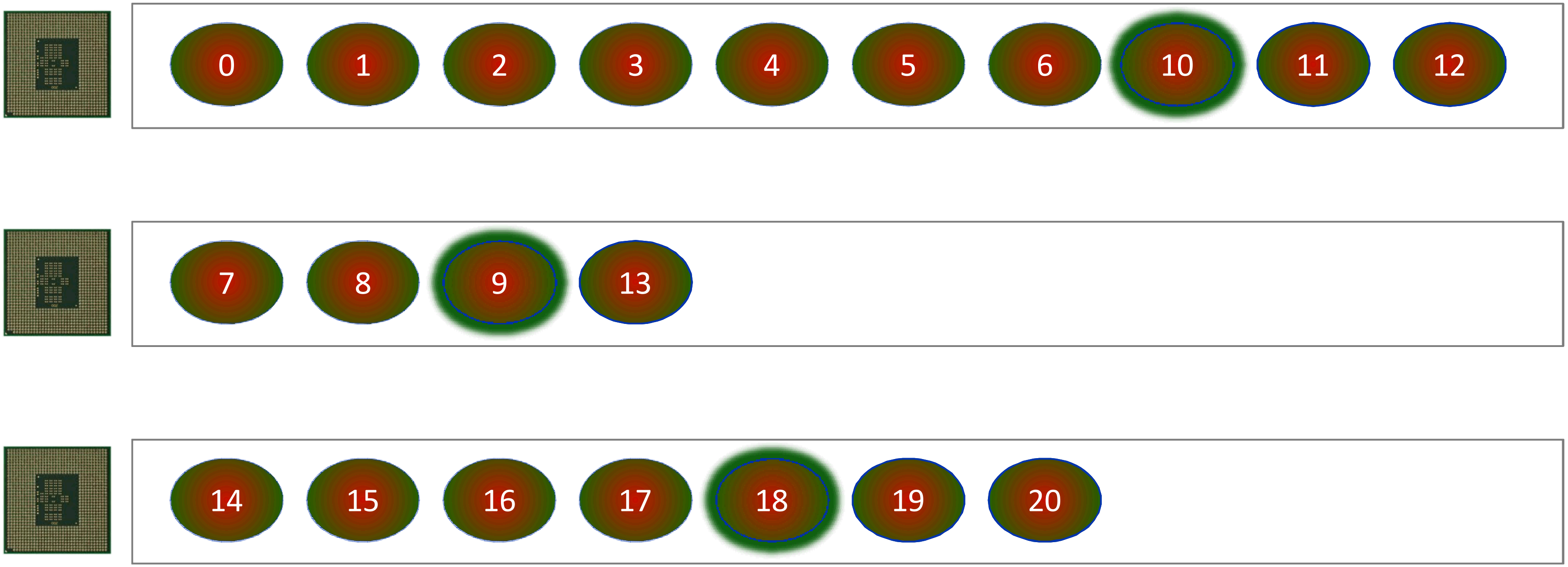
Task collections are a good first step towards resilient task-based distributed programming



- Work stealing enables tolerance to variations in the execution environment

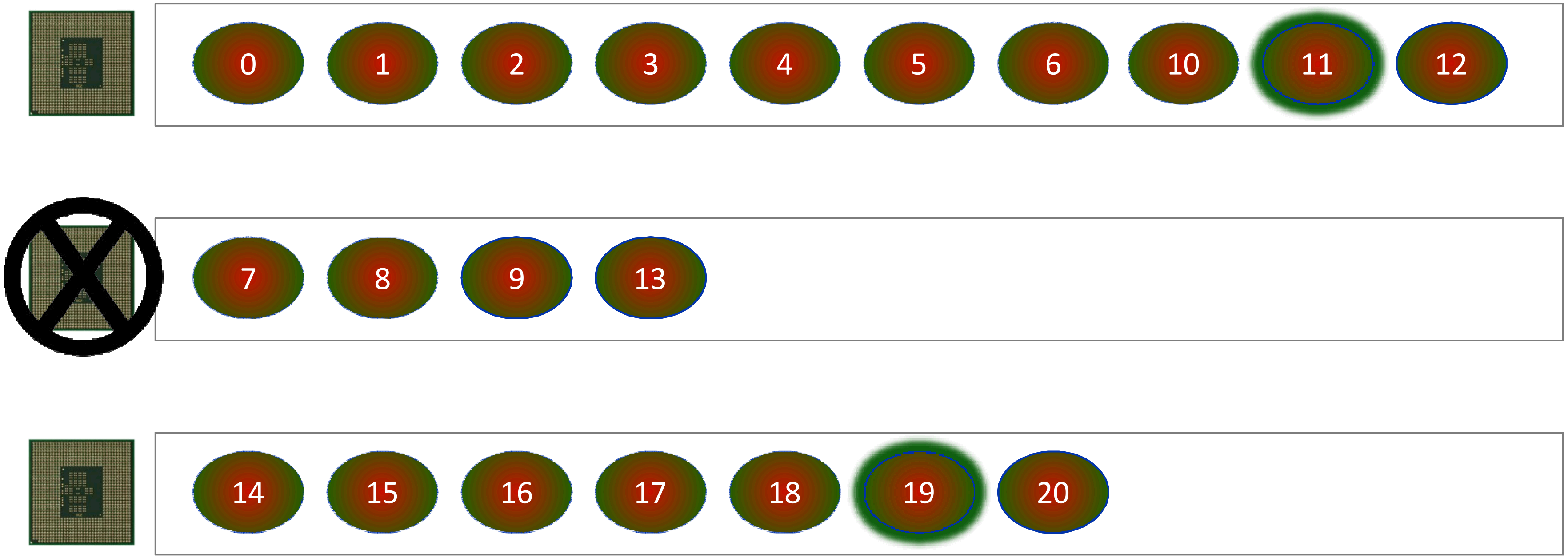
Robert L. Clay, CMU-2014

Task collections are a good first step towards resilient task-based distributed programming



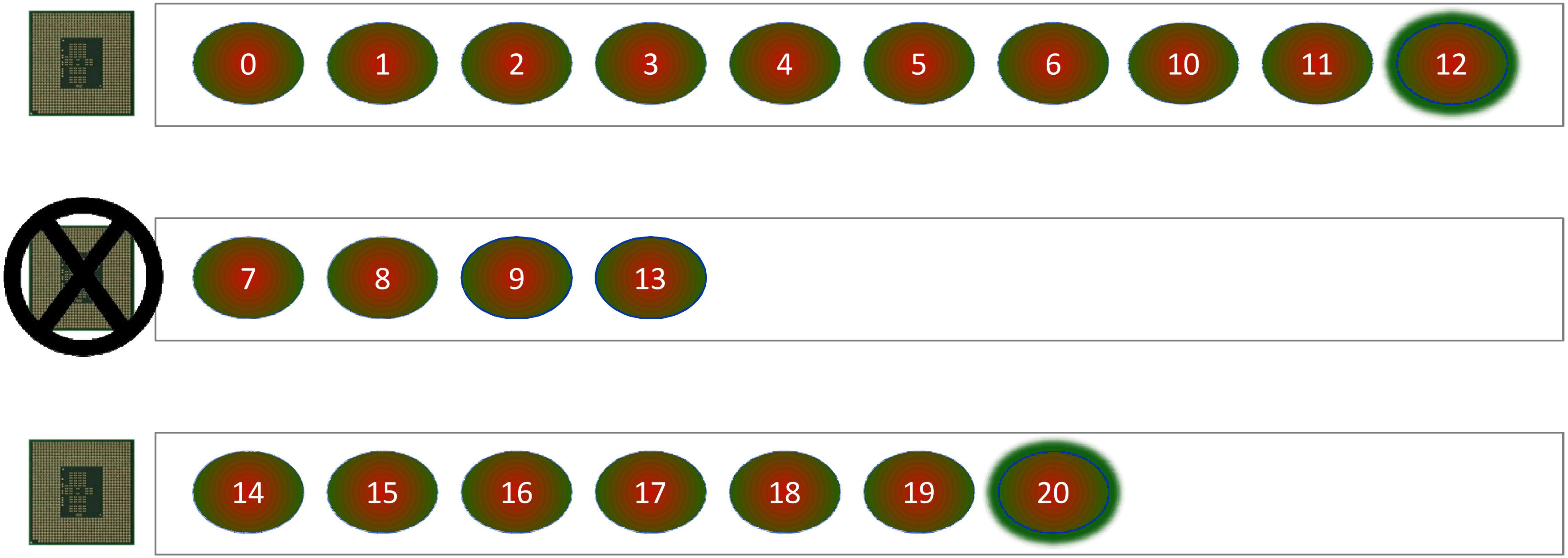
- Work stealing enables tolerance to variations in the execution environment

Task collections are a good first step towards resilient task-based distributed programming



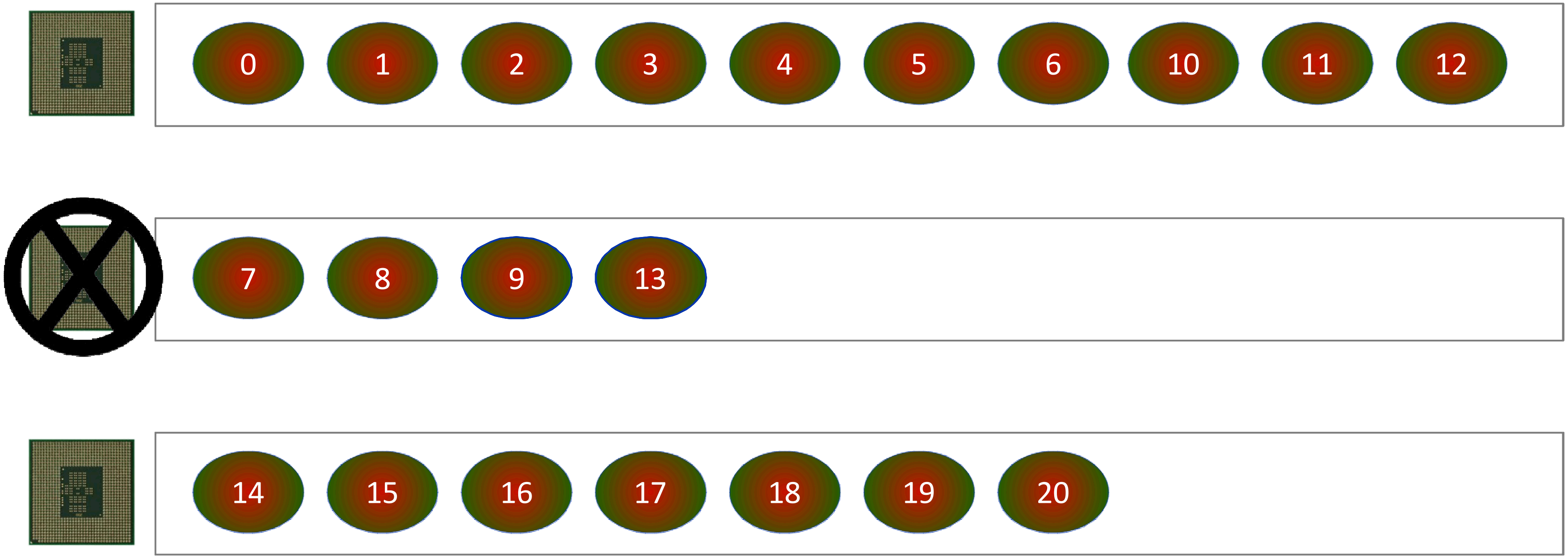
- Recovery is possible when a node goes down
- A simple lazy scheme ignores faults until task collection has terminated

Task collections are a good first step towards resilient task-based distributed programming



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

Task collections are a good first step towards resilient task-based distributed programming



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

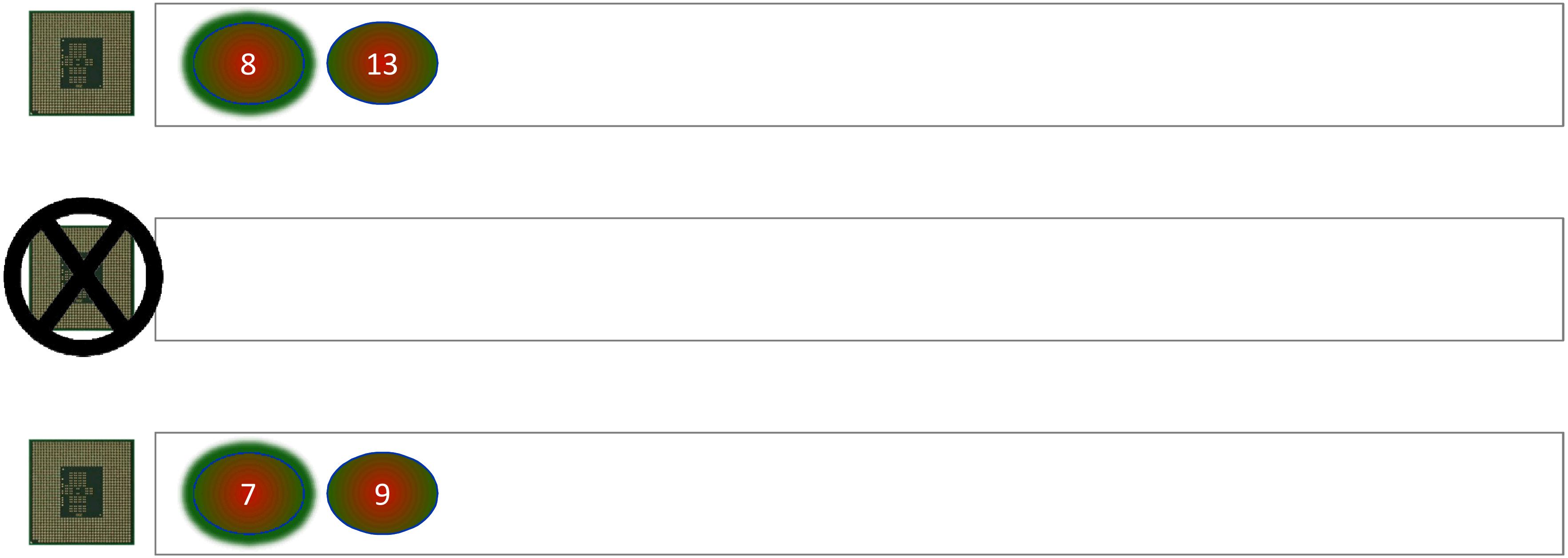
Task collections are a good first step towards resilient task-based distributed programming

Global reduction: Highlighted tasks incomplete

task	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
finished	1	1	1	1	1	1	1	0	0	0	1	1	1	0	1	1	1	1	1	1	1

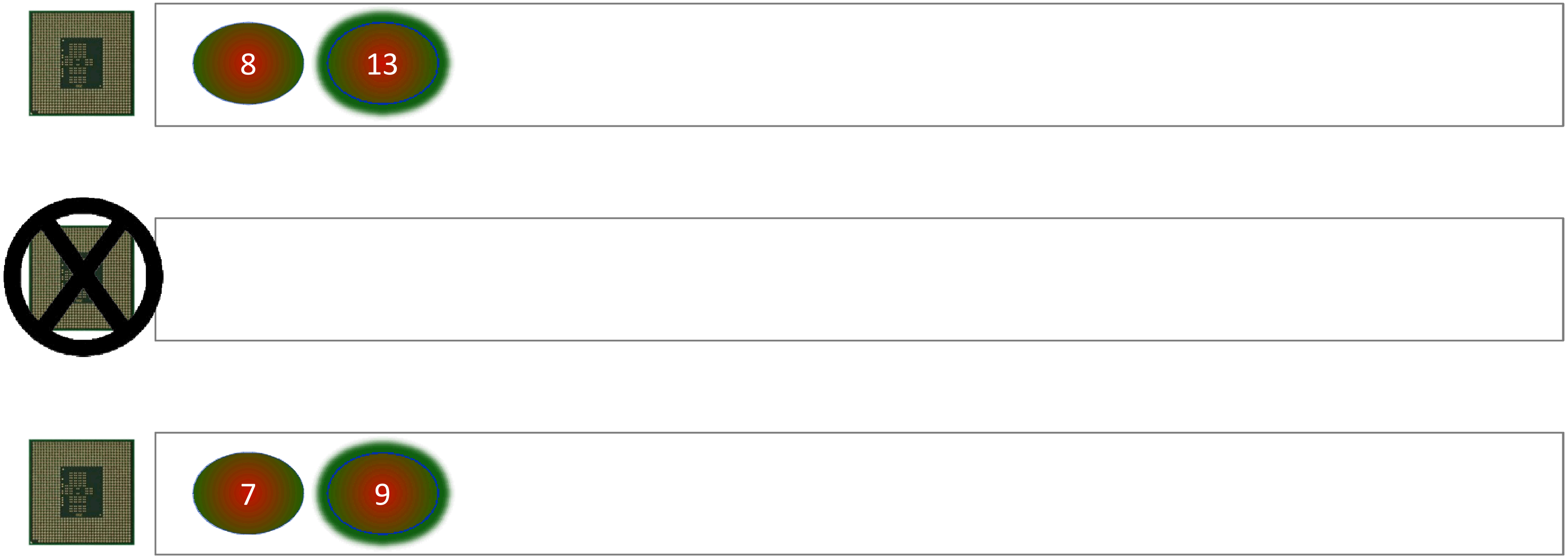
- A global reduction is used to identify incomplete tasks

Task collections are a good first step towards resilient task-based distributed programming



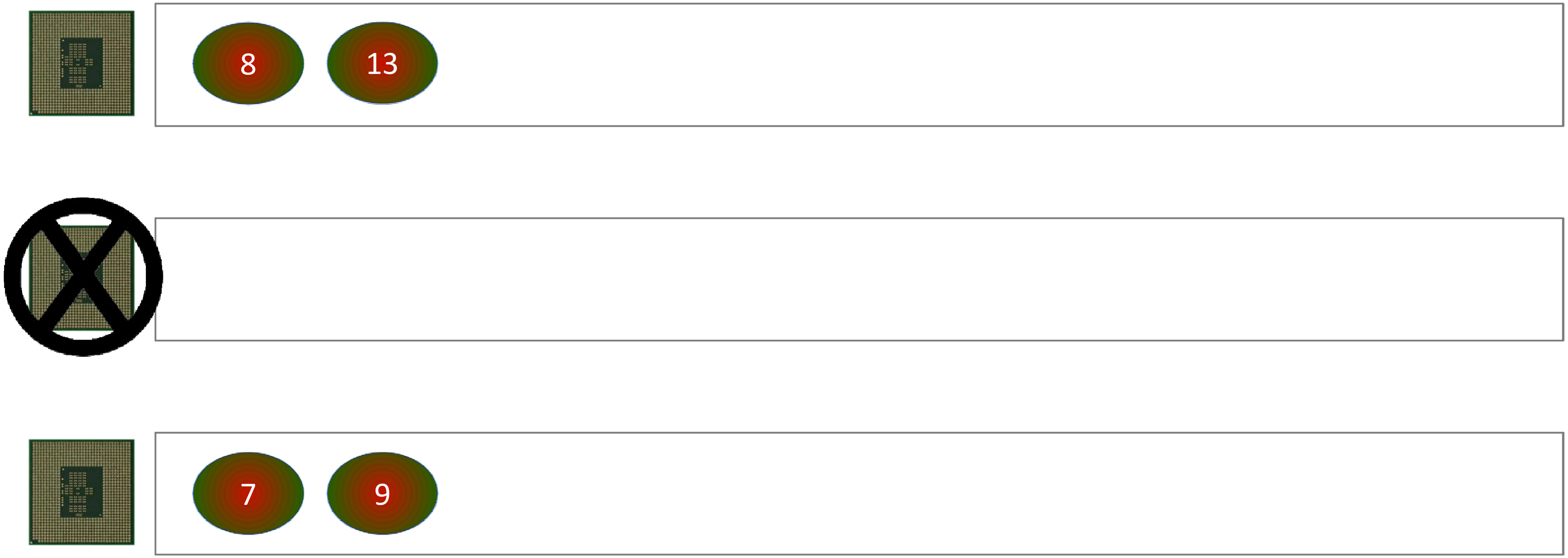
- The incomplete tasks are re-distributed to active nodes
- Execution continues until all tasks have finished

Task collections are a good first step towards resilient task-based distributed programming



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

Task collections are a good first step towards resilient task-based distributed programming



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

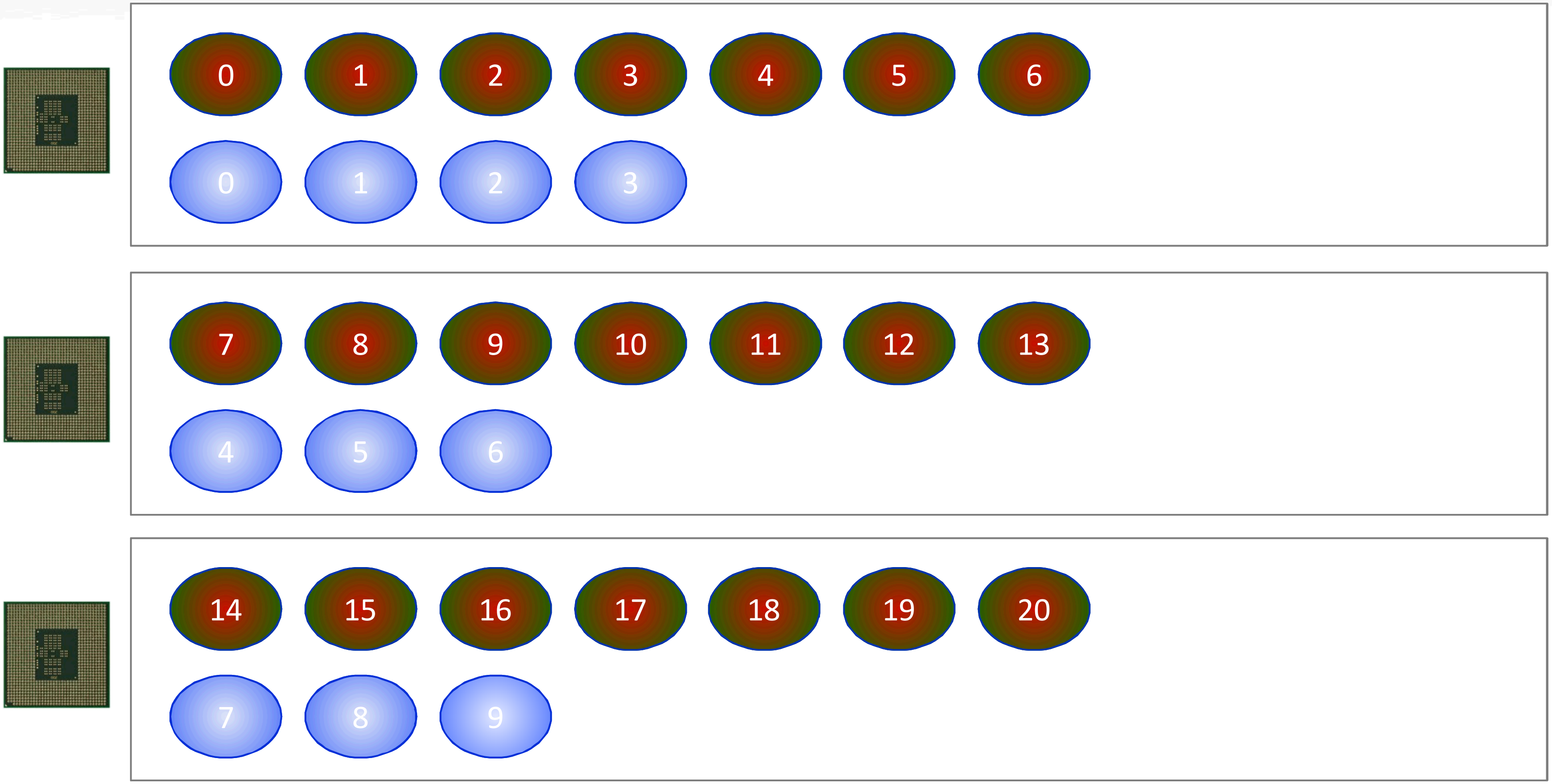
Task collections are a good first step towards resilient task-based distributed programming

Global reduction: All tasks complete

task	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
finished	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

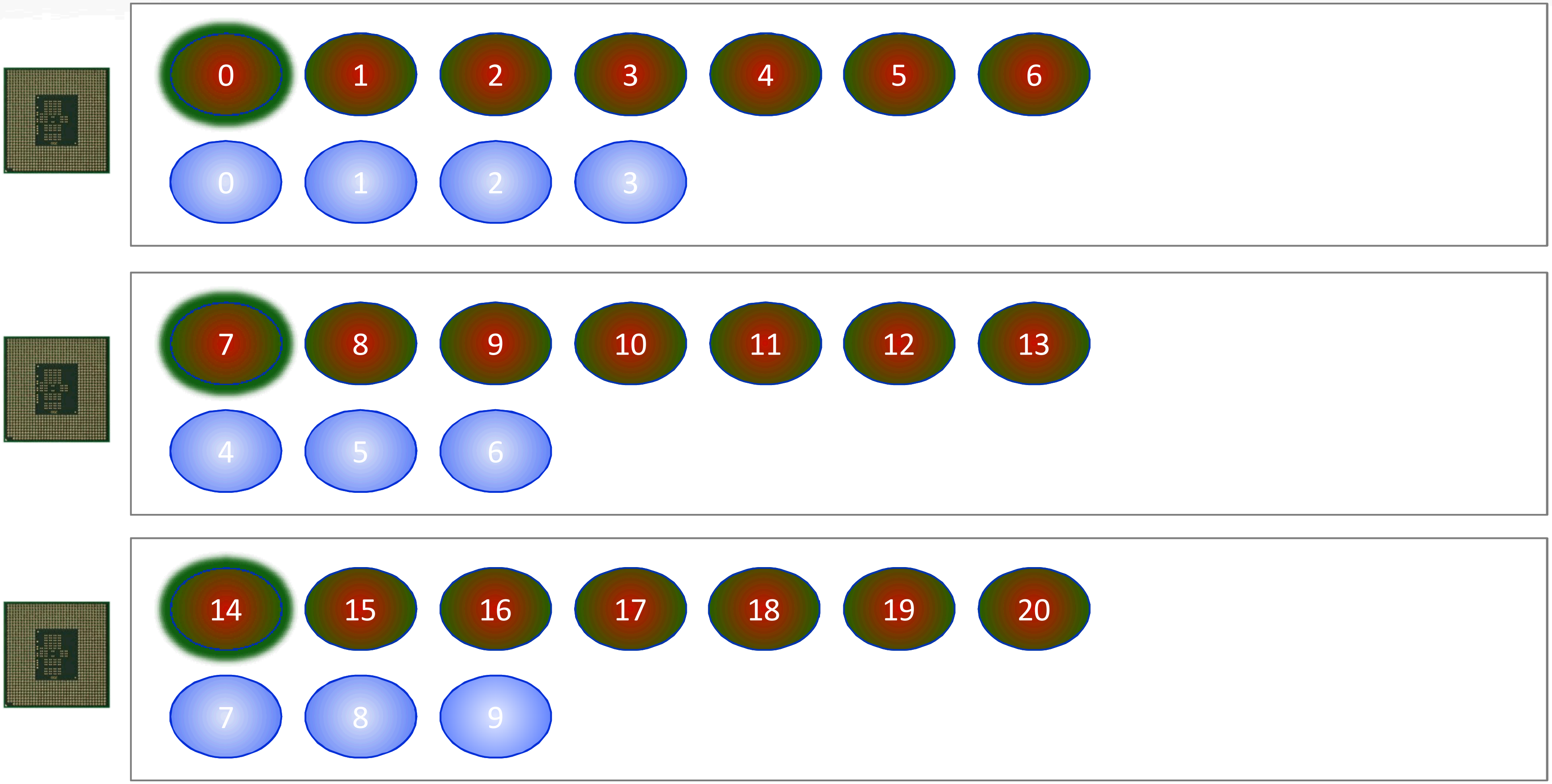
- The incomplete tasks are re-distributed to active nodes
- Execution continues until all tasks have finished

We are extending task collections to support multiple collections operating concurrently



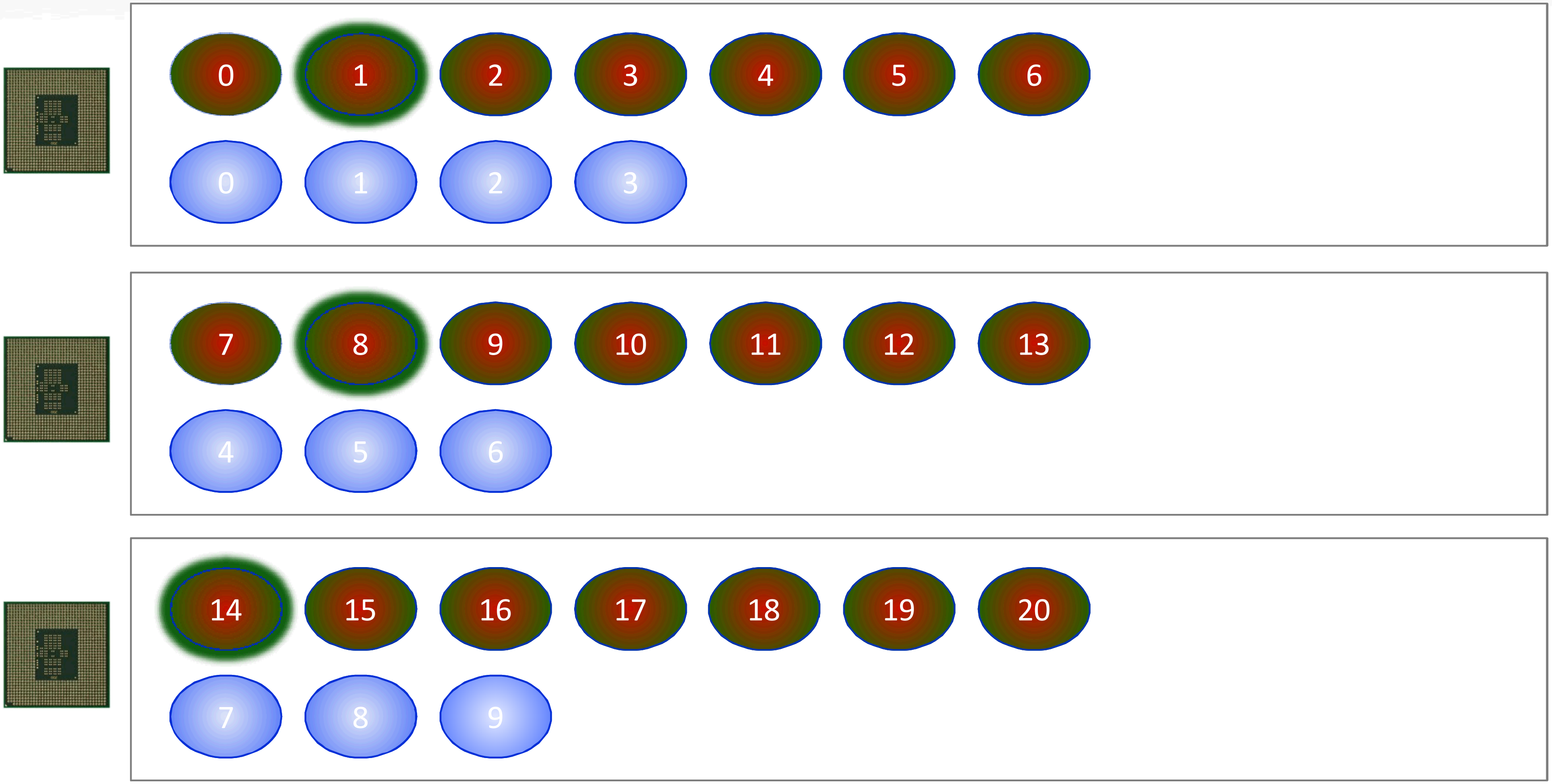
- Priority (**CriticalPathCollection**) > Priority(**AuxiliaryCollection**)
- Increase task parallelism while maintaining critical path

We are extending task collections to support multiple collections operating concurrently



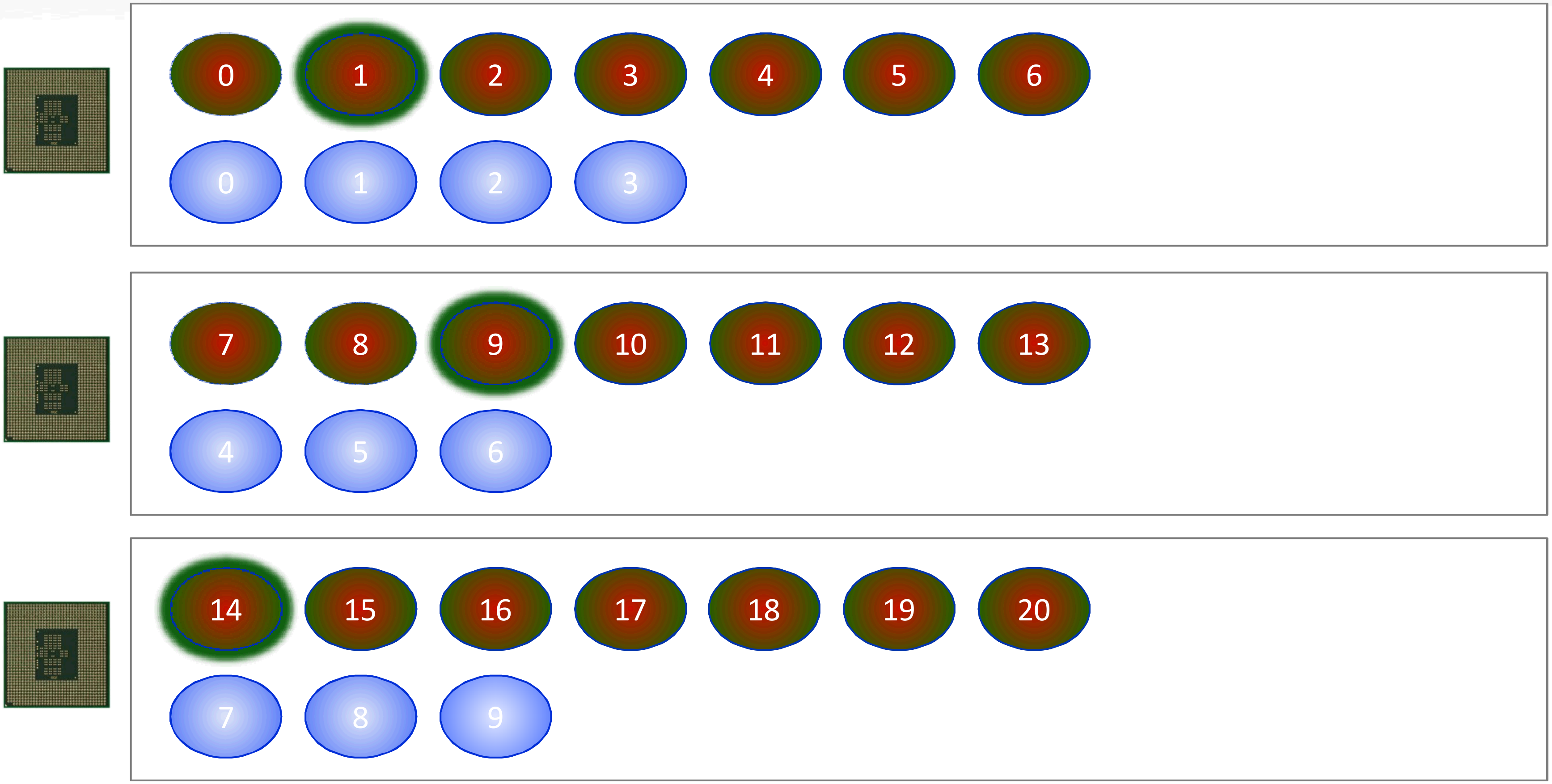
- Priority (**CriticalPathCollection**) > Priority(**AuxiliaryCollection**)
- Increase task parallelism while maintaining critical path

We are extending task collections to support multiple collections operating concurrently



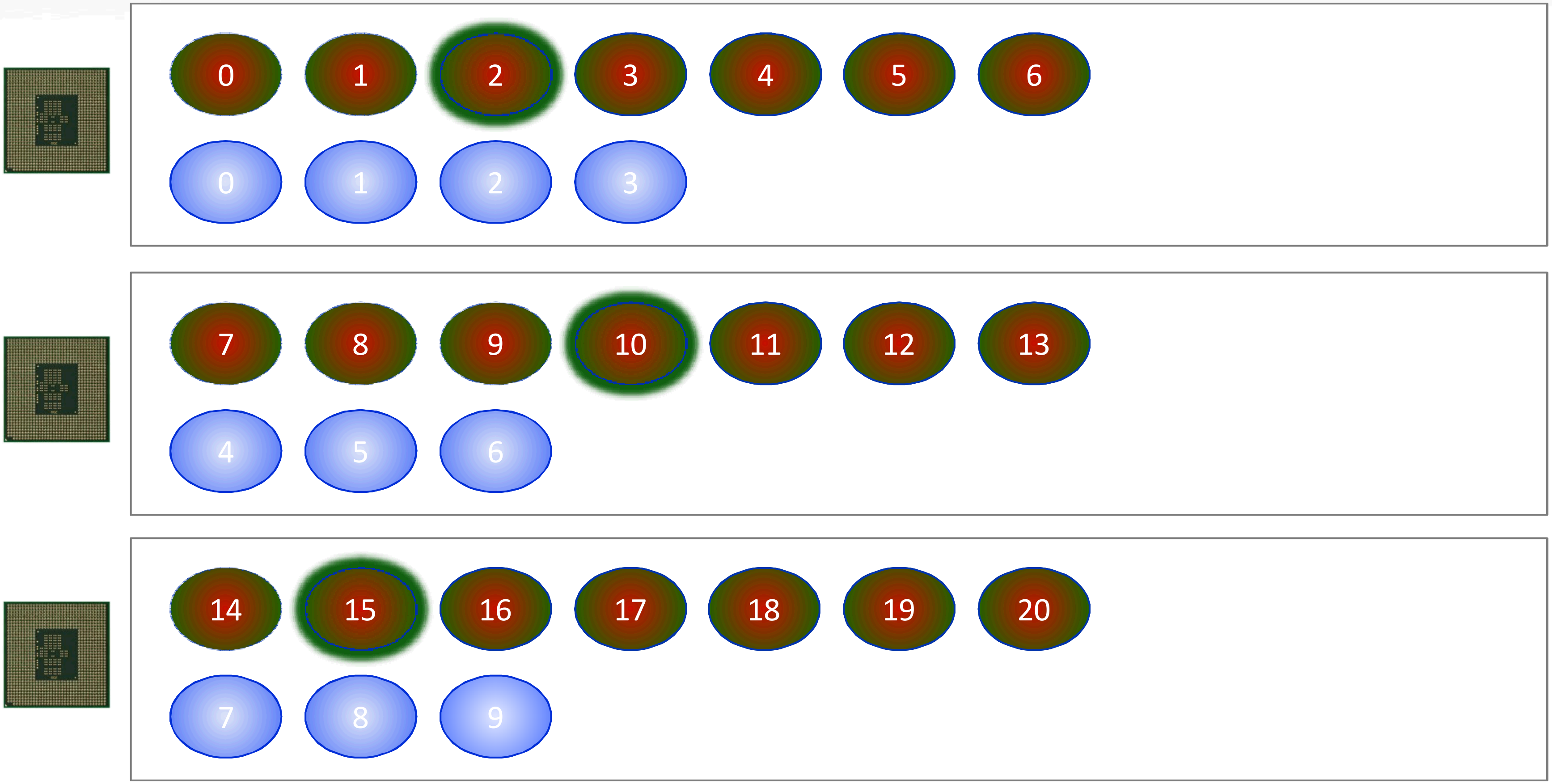
- Priority (**CriticalPathCollection**) > Priority(**AuxiliaryCollection**)
- Increase task parallelism while maintaining critical path

We are extending task collections to support multiple collections operating concurrently



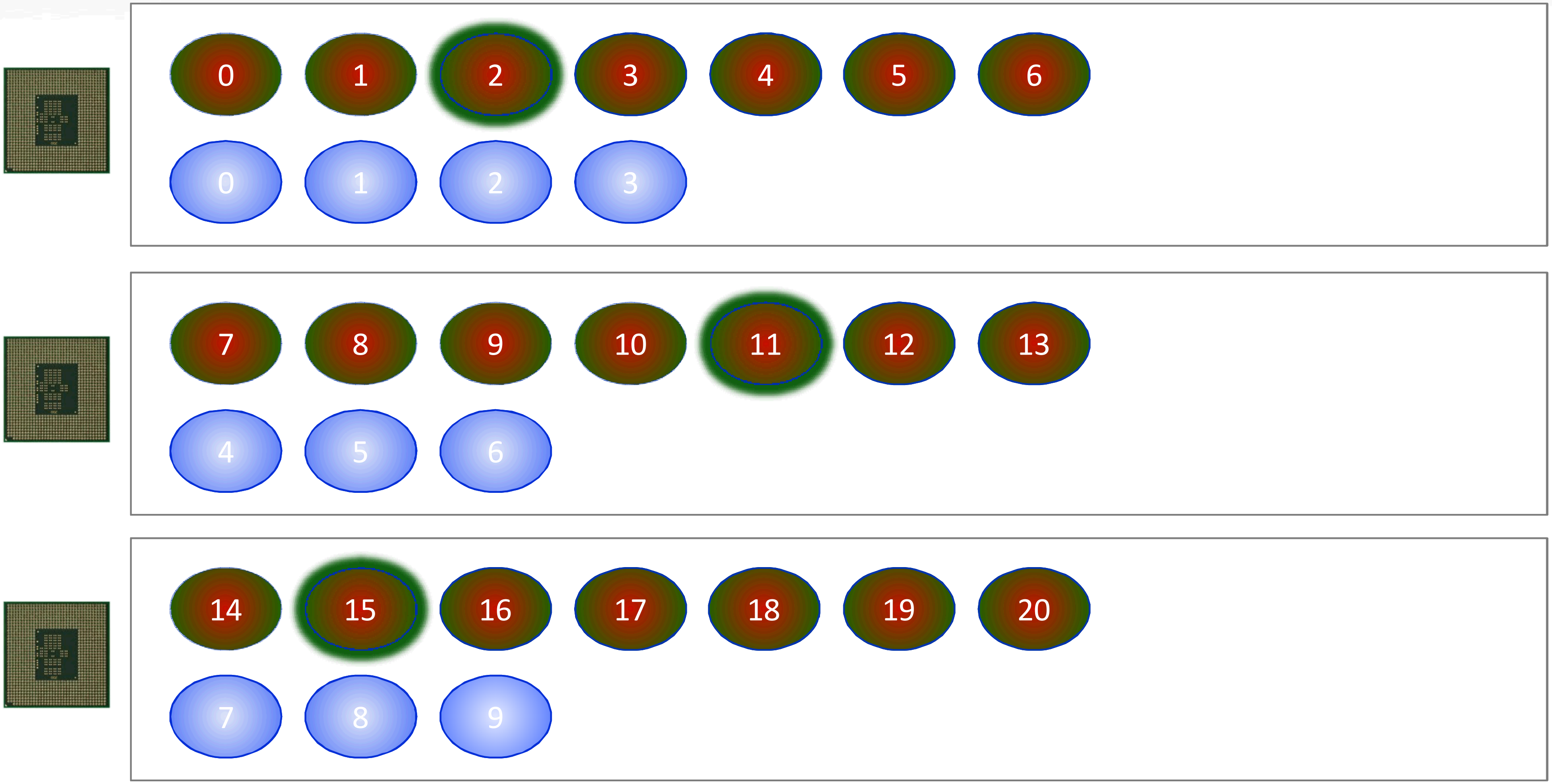
- Priority (**CriticalPathCollection**) > Priority(**AuxiliaryCollection**)
- Increase task parallelism while maintaining critical path

We are extending task collections to support multiple collections operating concurrently



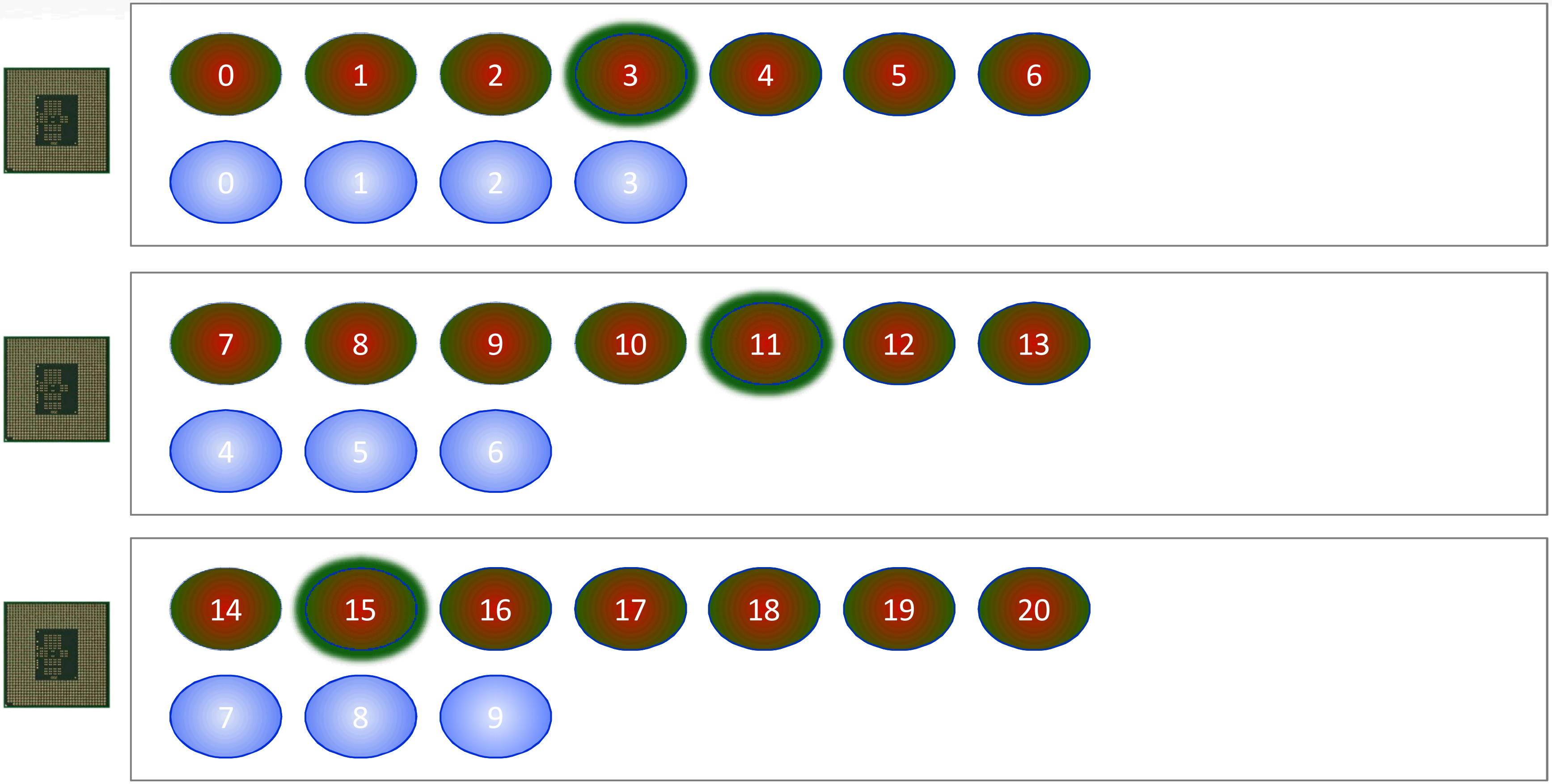
- Priority (**CriticalPathCollection**) > Priority(**AuxiliaryCollection**)
- Increase task parallelism while maintaining critical path

We are extending task collections to support multiple collections operating concurrently



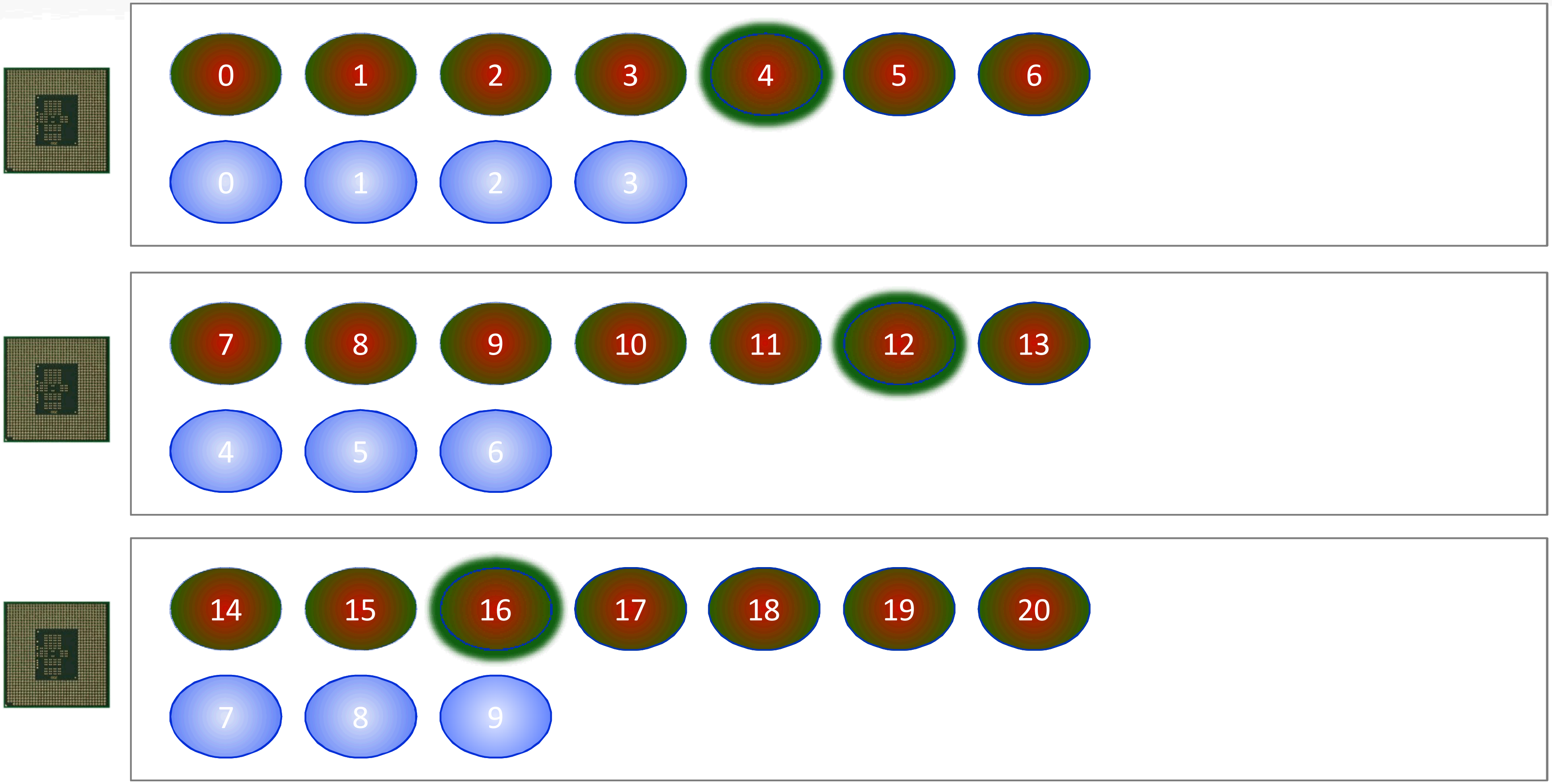
- Priority (**CriticalPathCollection**) > Priority(**AuxiliaryCollection**)
- Increase task parallelism while maintaining critical path

We are extending task collections to support multiple collections operating concurrently



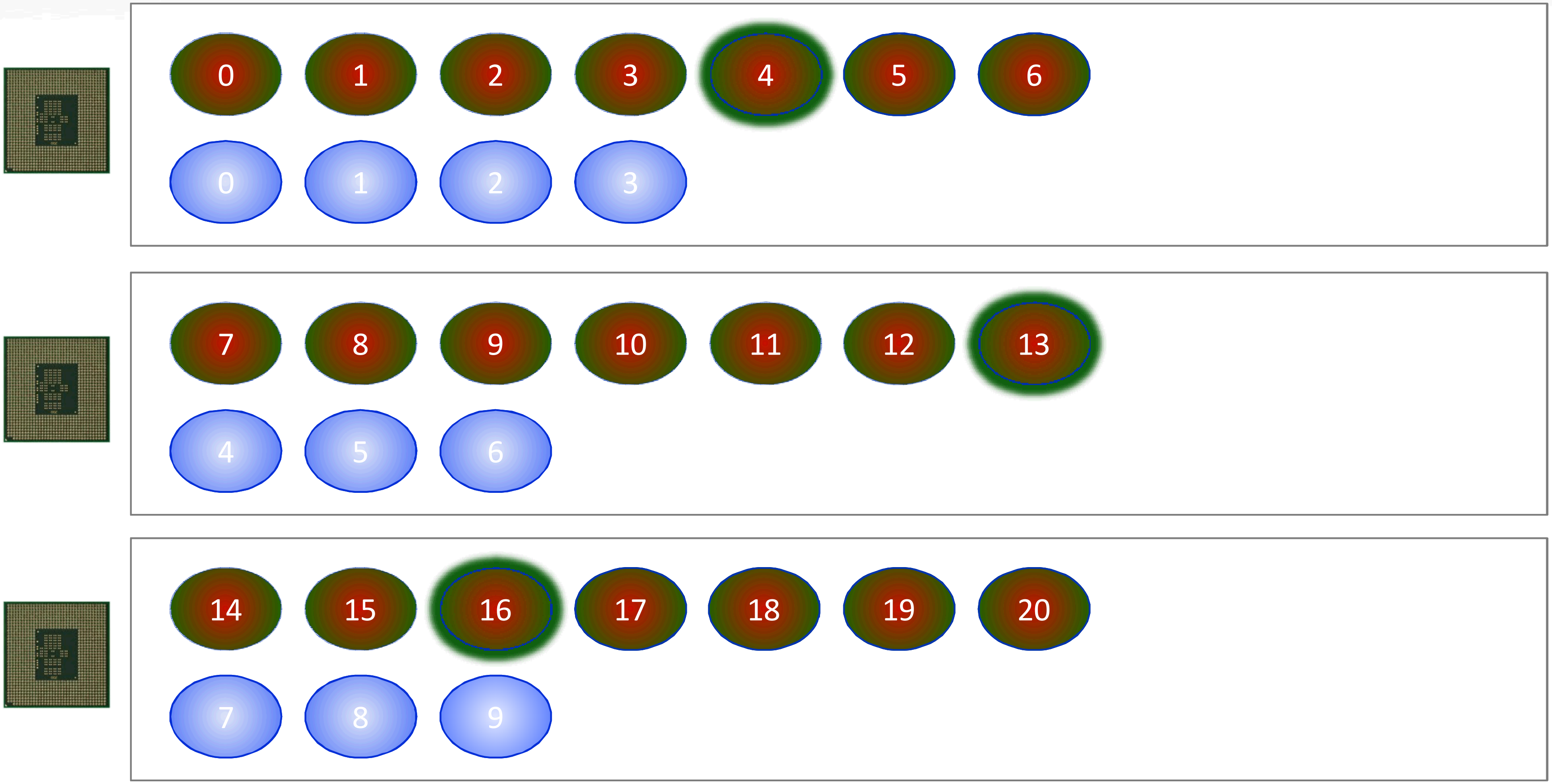
- Priority (**CriticalPathCollection**) > Priority(**AuxiliaryCollection**)
- Increase task parallelism while maintaining critical path

We are extending task collections to support multiple collections operating concurrently



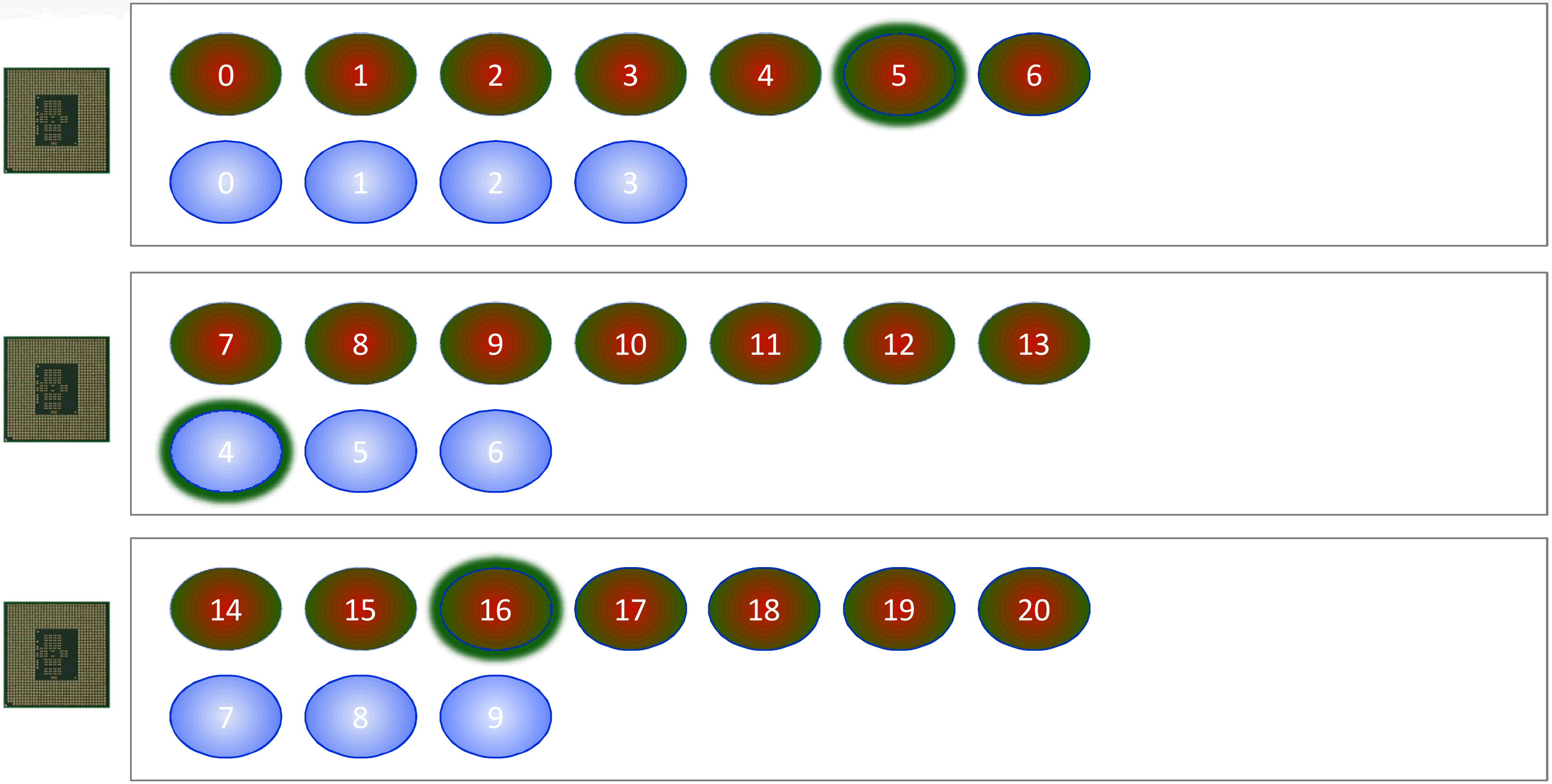
- Priority (**CriticalPathCollection**) > Priority(**AuxiliaryCollection**)
- Increase task parallelism while maintaining critical path

We are extending task collections to support multiple collections operating concurrently



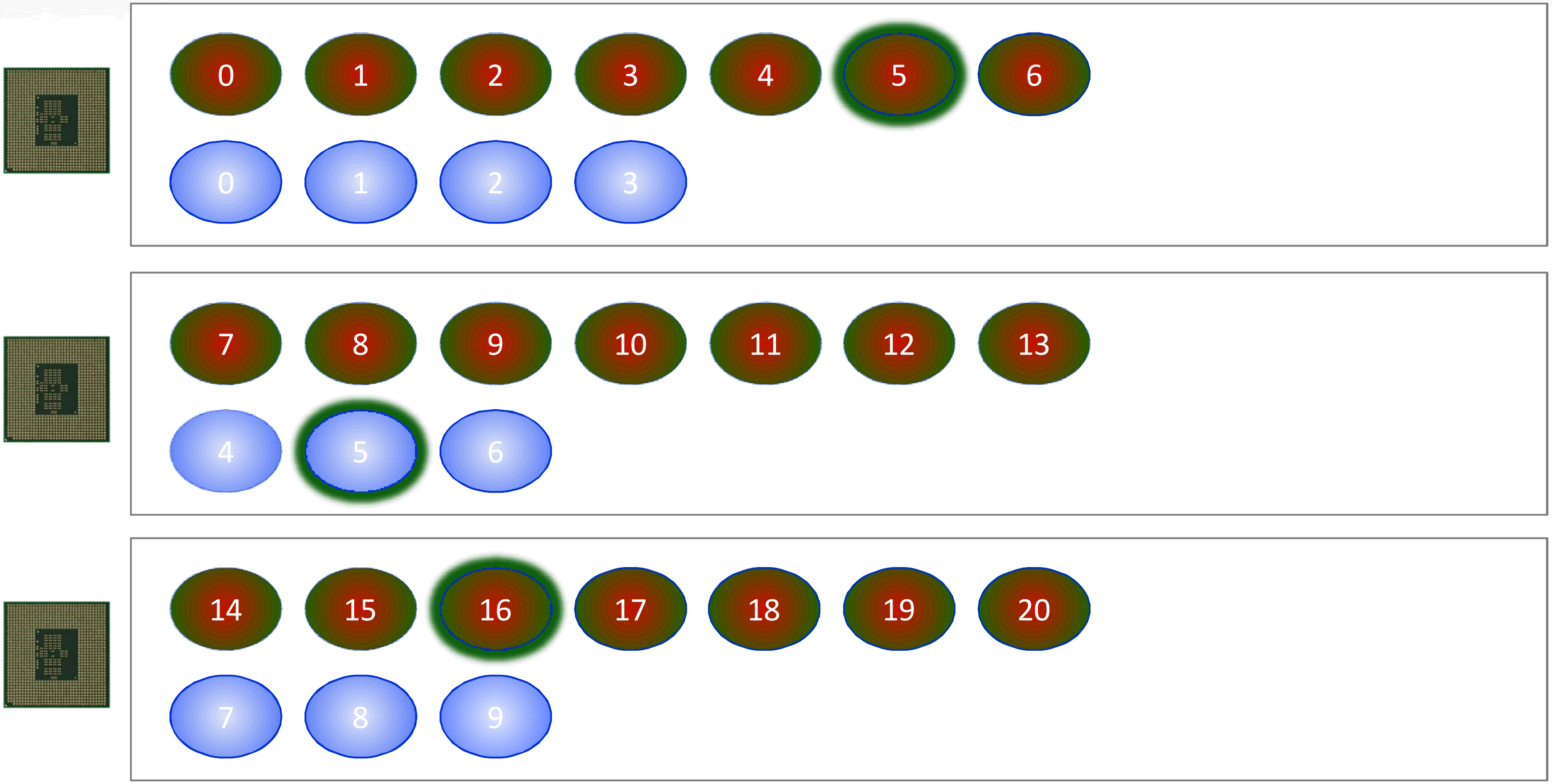
- Priority (**CriticalPathCollection**) > Priority(**AuxiliaryCollection**)
- Increase task parallelism while maintaining critical path

We are extending task collections to support multiple collections operating concurrently



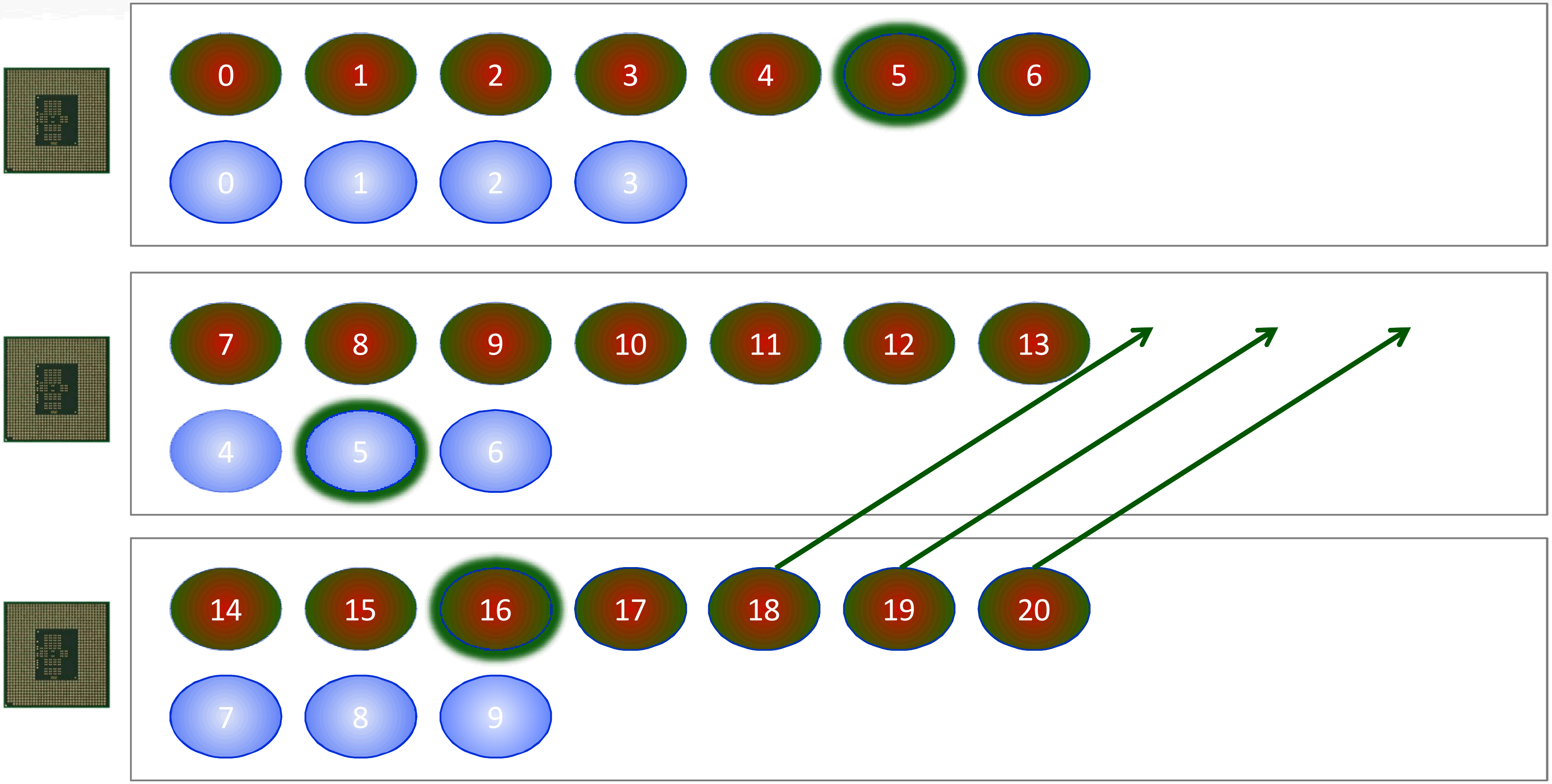
- Priority (**CriticalPathCollection**) > Priority(**AuxiliaryCollection**)
- Increase task parallelism while maintaining critical path

We are extending task collections to support multiple collections operating concurrently



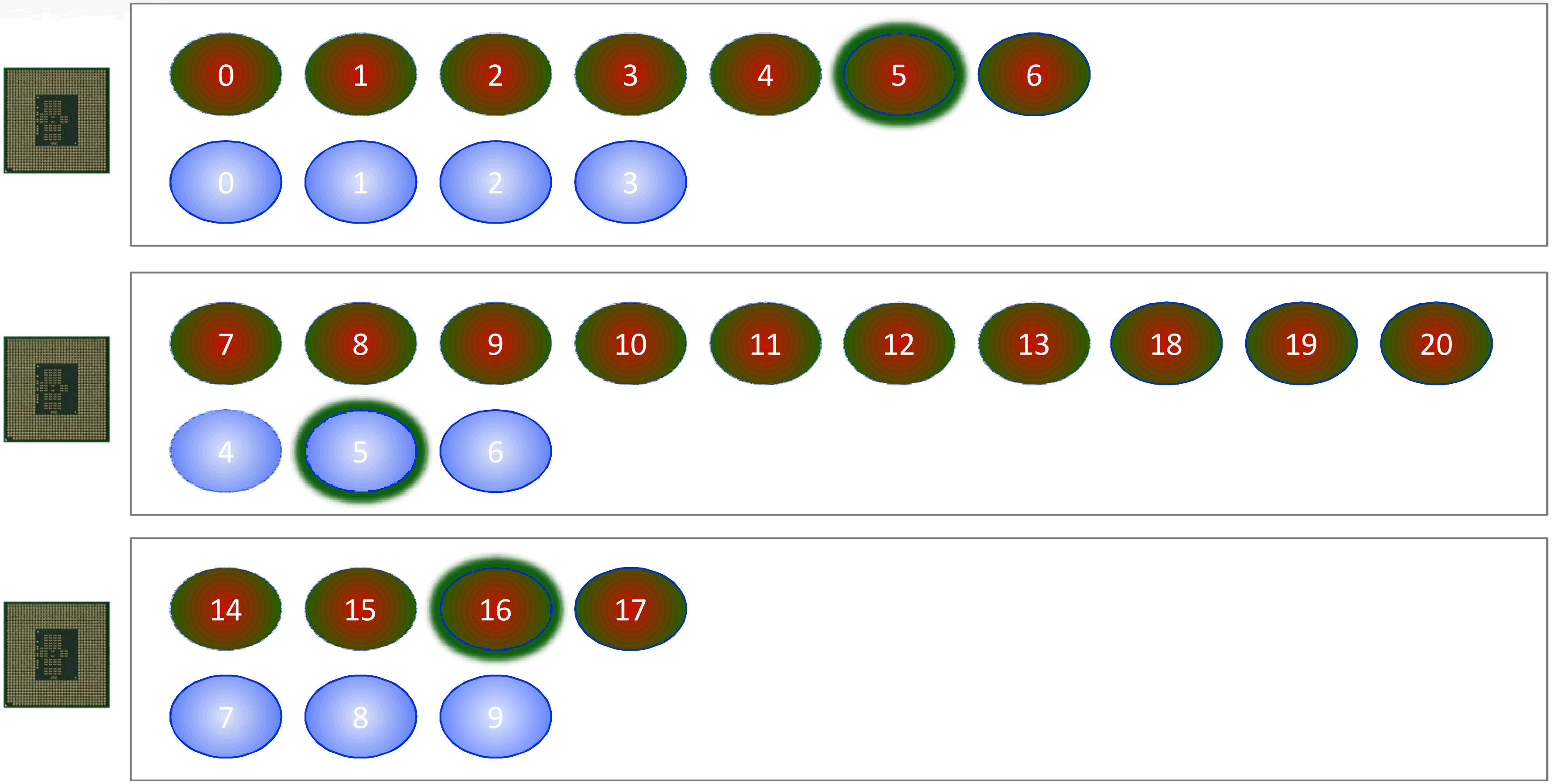
- Priority (**CriticalPathCollection**) > Priority(**AuxiliaryCollection**)
- Increase task parallelism while maintaining critical path

We are extending task collections to support multiple collections operating concurrently



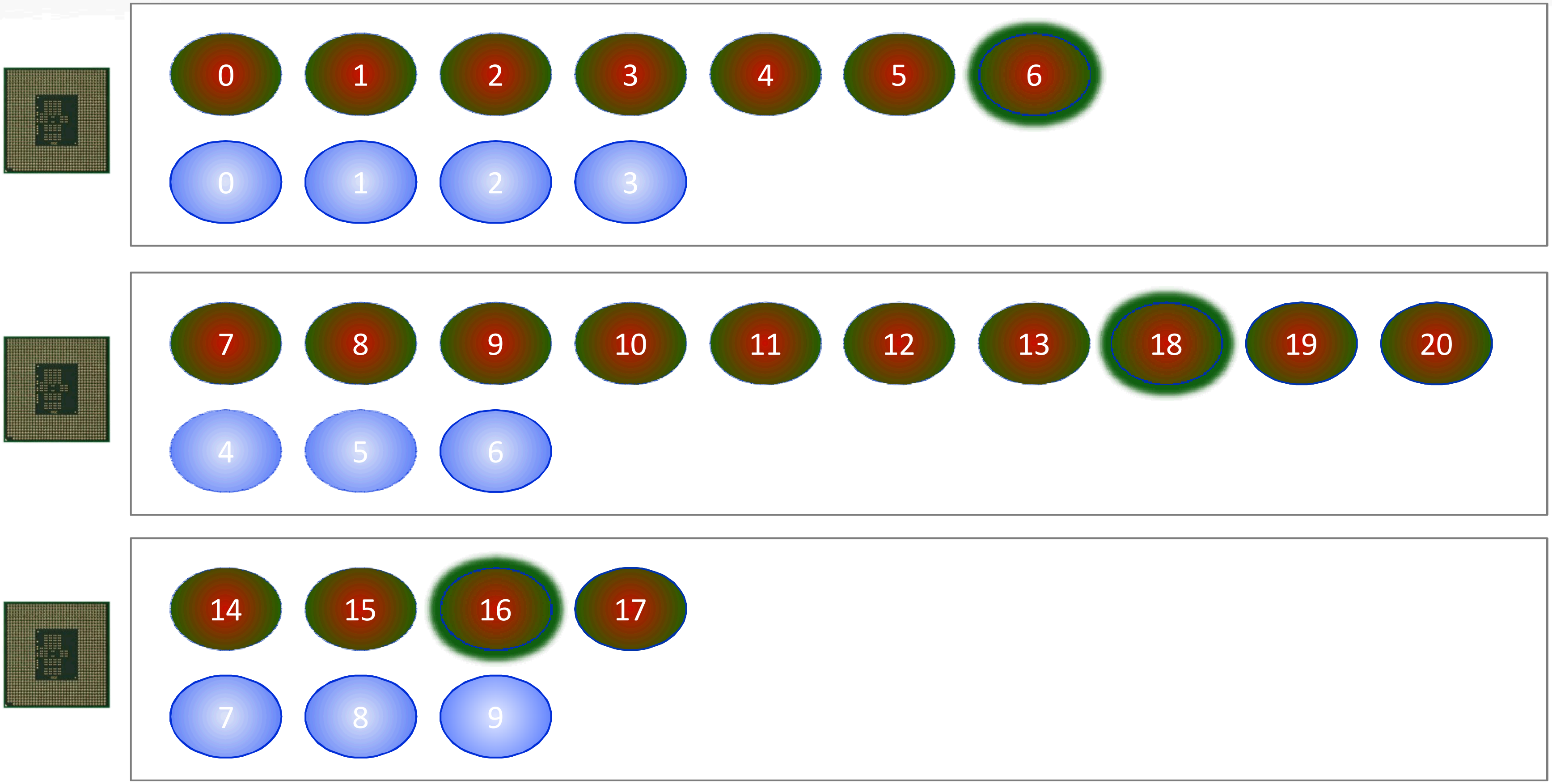
- Priority (**CriticalPathCollection**) > Priority(**AuxiliaryCollection**)
- Increase task parallelism while maintaining critical path

We are extending task collections to support multiple collections operating concurrently



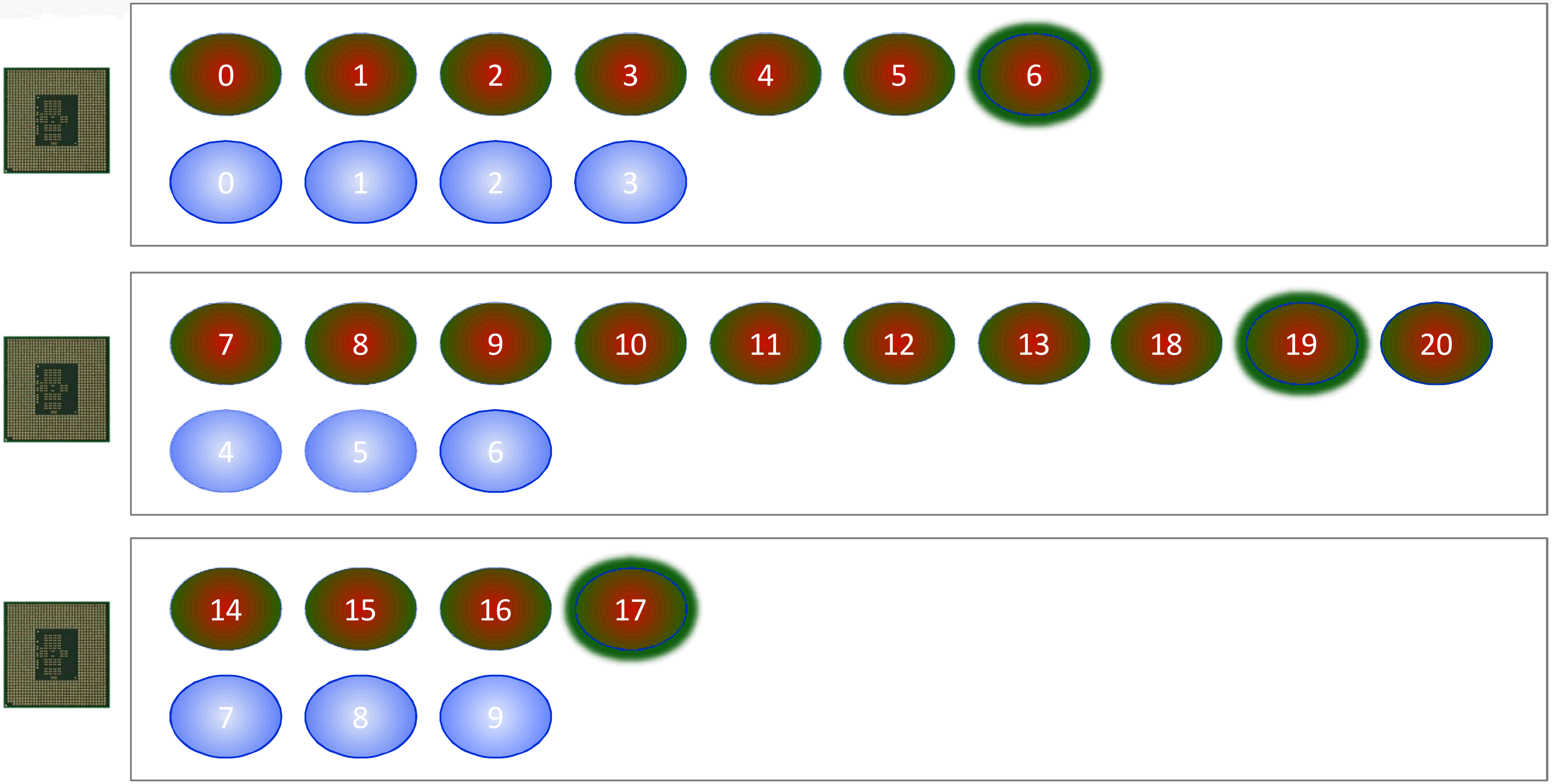
- Priority (**CriticalPathCollection**) > Priority(**AuxiliaryCollection**)
- Increase task parallelism while maintaining critical path

We are extending task collections to support multiple collections operating concurrently



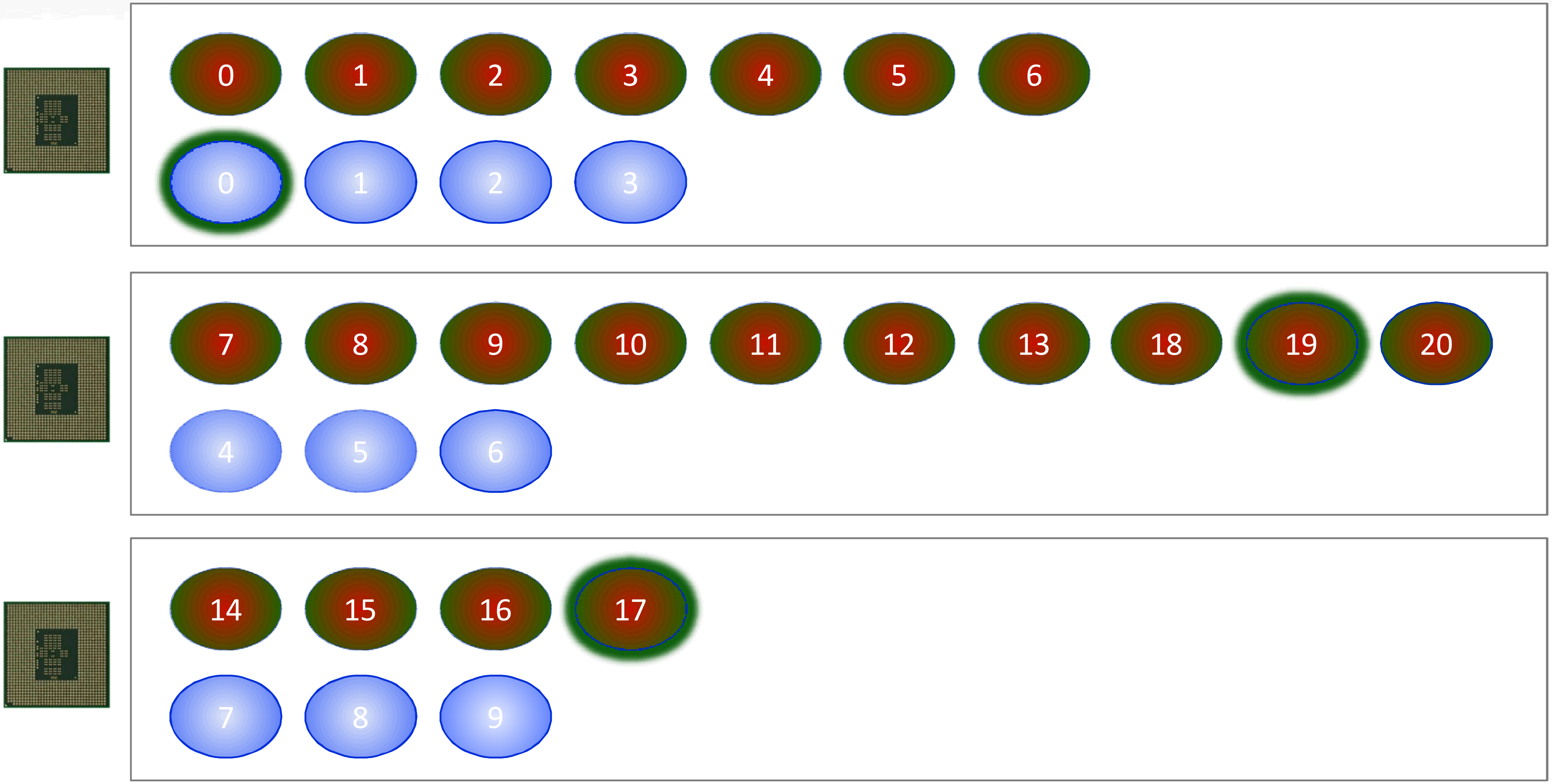
- Priority (**CriticalPathCollection**) > Priority(**AuxiliaryCollection**)
- Increase task parallelism while maintaining critical path

We are extending task collections to support multiple collections operating concurrently



- Priority (**CriticalPathCollection**) > Priority(**AuxiliaryCollection**)
- Increase task parallelism while maintaining critical path

We are extending task collections to support multiple collections operating concurrently

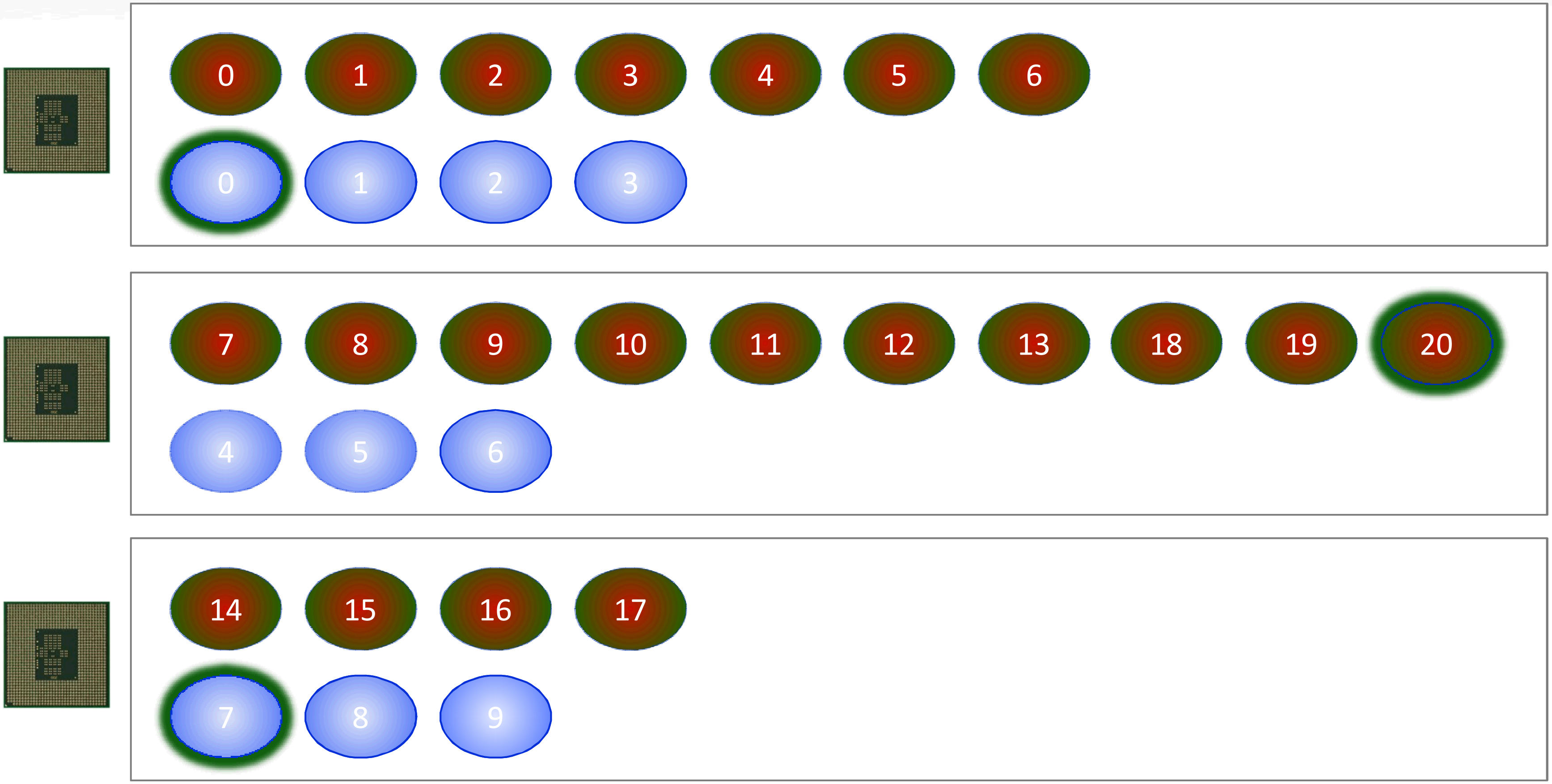


- Priority (**CriticalPathCollection**) > Priority(**AuxiliaryCollection**)
- Increase task parallelism while maintaining critical path

Robert L. Clay, CMU-2014

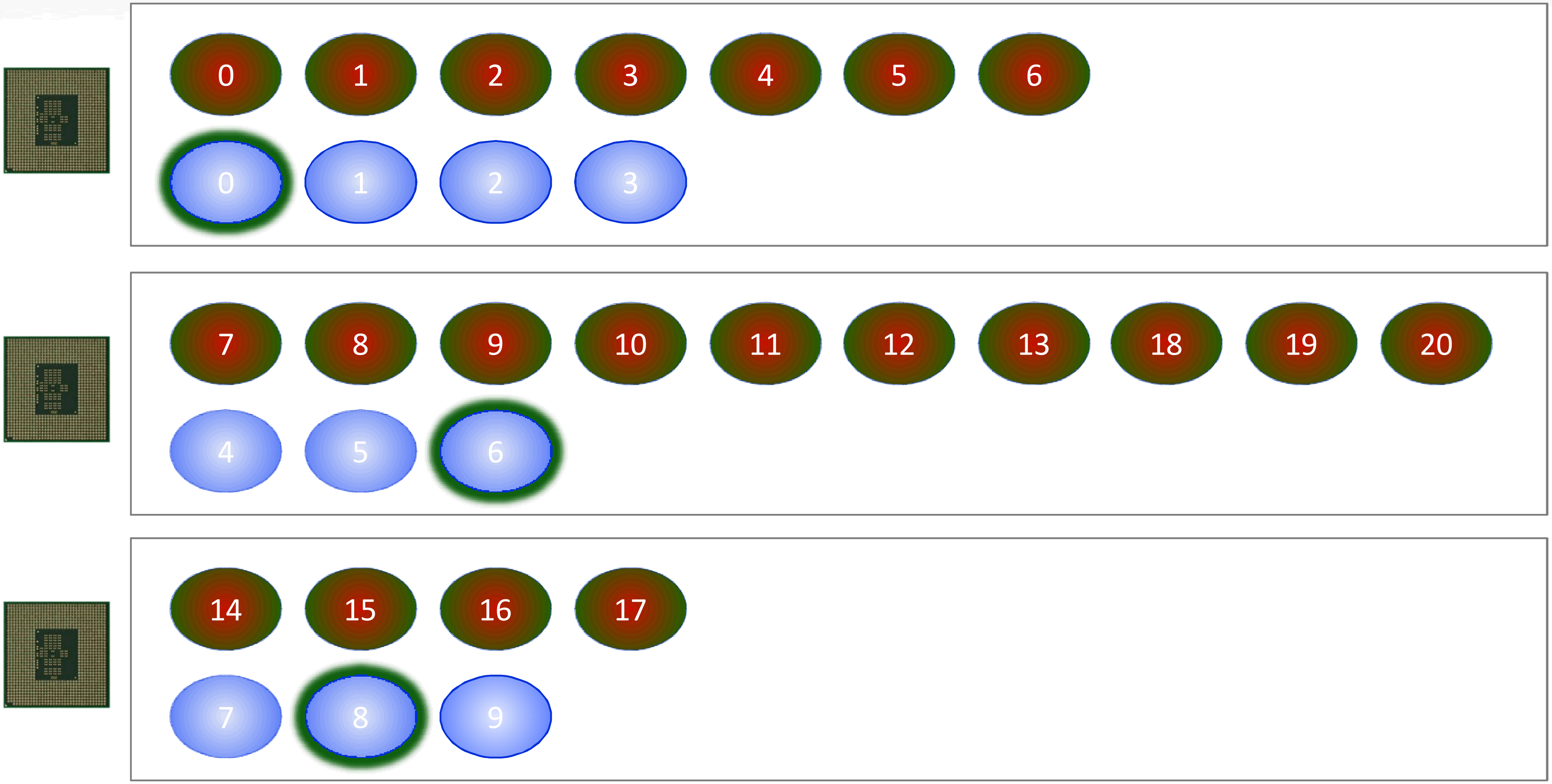
Sandia National Laboratories

We are extending task collections to support multiple collections operating concurrently



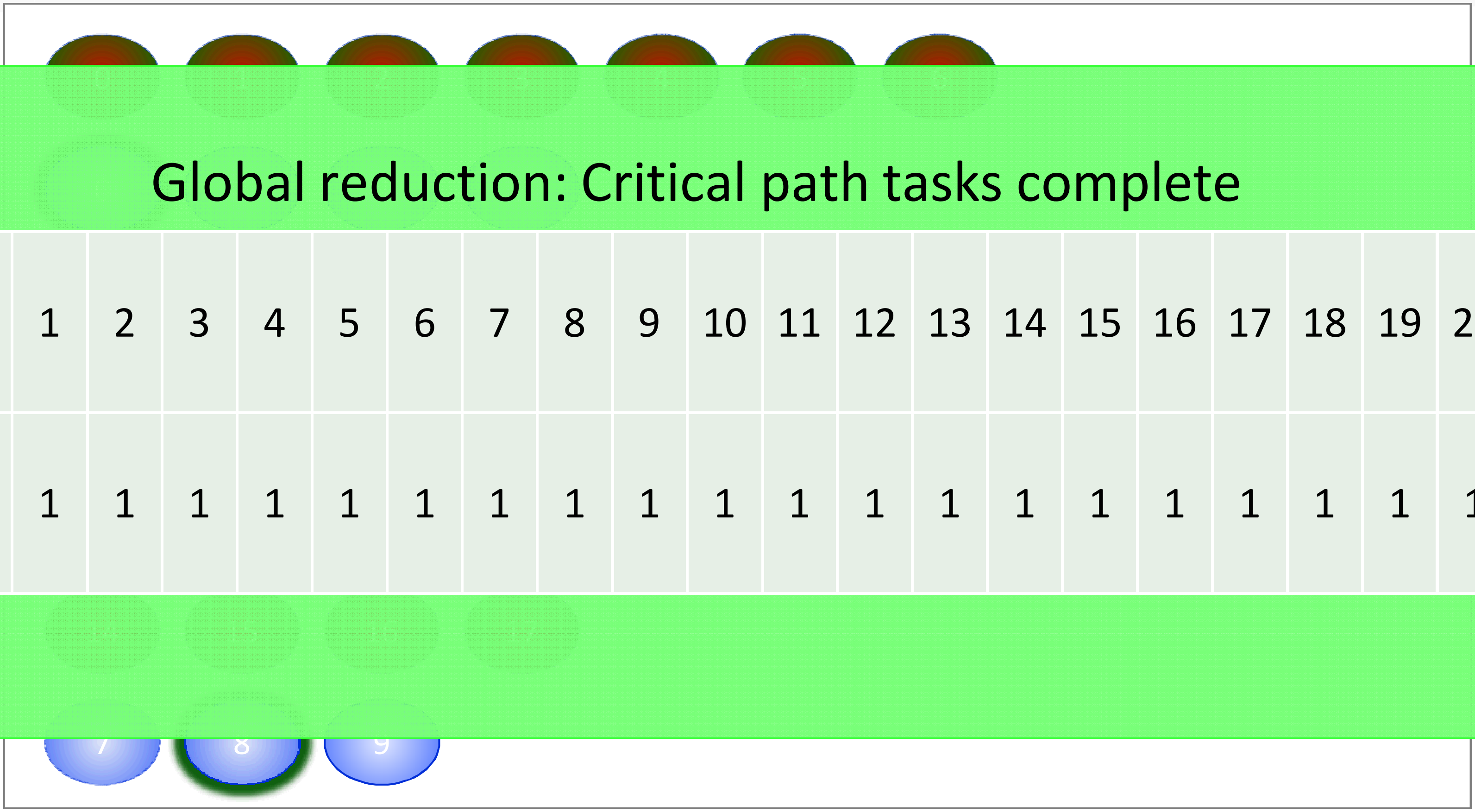
- Priority (**CriticalPathCollection**) > Priority(**AuxiliaryCollection**)
- Increase task parallelism while maintaining critical path

We are extending task collections to support multiple collections operating concurrently



- Priority (**CriticalPathCollection**) > Priority(**AuxiliaryCollection**)
- Increase task parallelism while maintaining critical path

We are extending task collections to support multiple collections operating concurrently

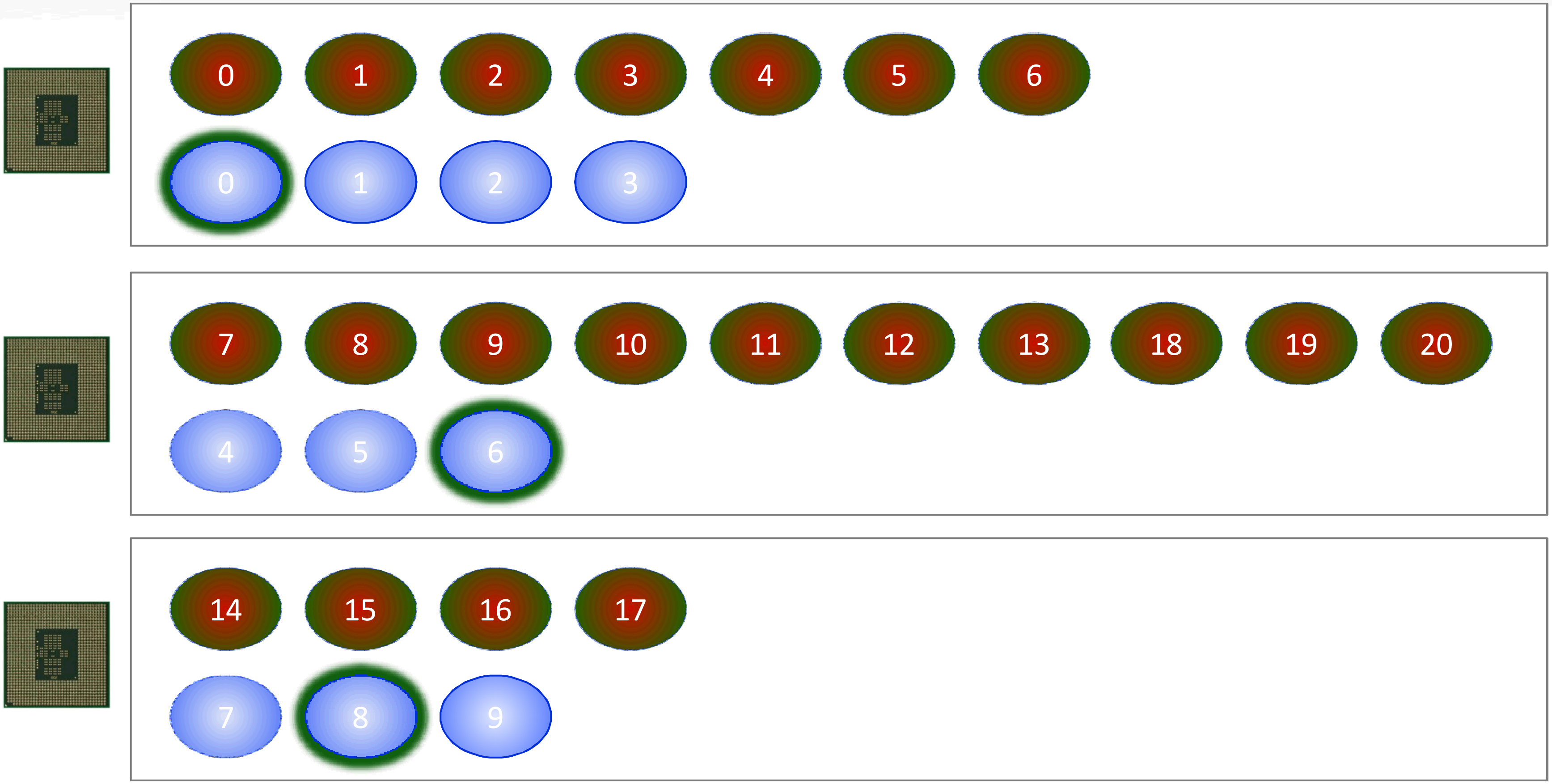


Global reduction: Critical path tasks complete

task	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
finished	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

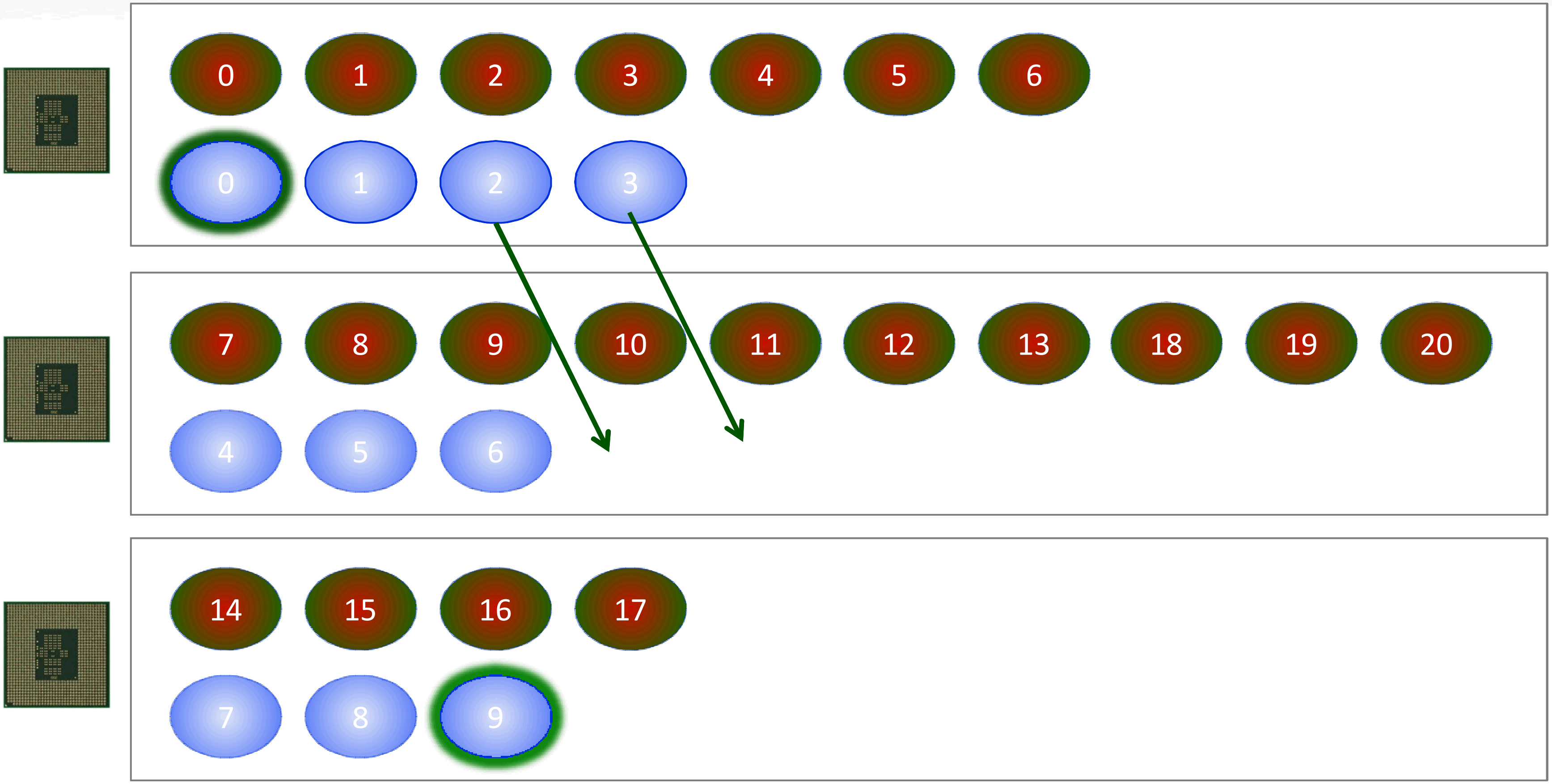
- Priority (**CriticalPathCollection**) > Priority(**AuxiliaryCollection**)
- Increase task parallelism while maintaining critical path

We are extending task collections to support multiple collections operating concurrently



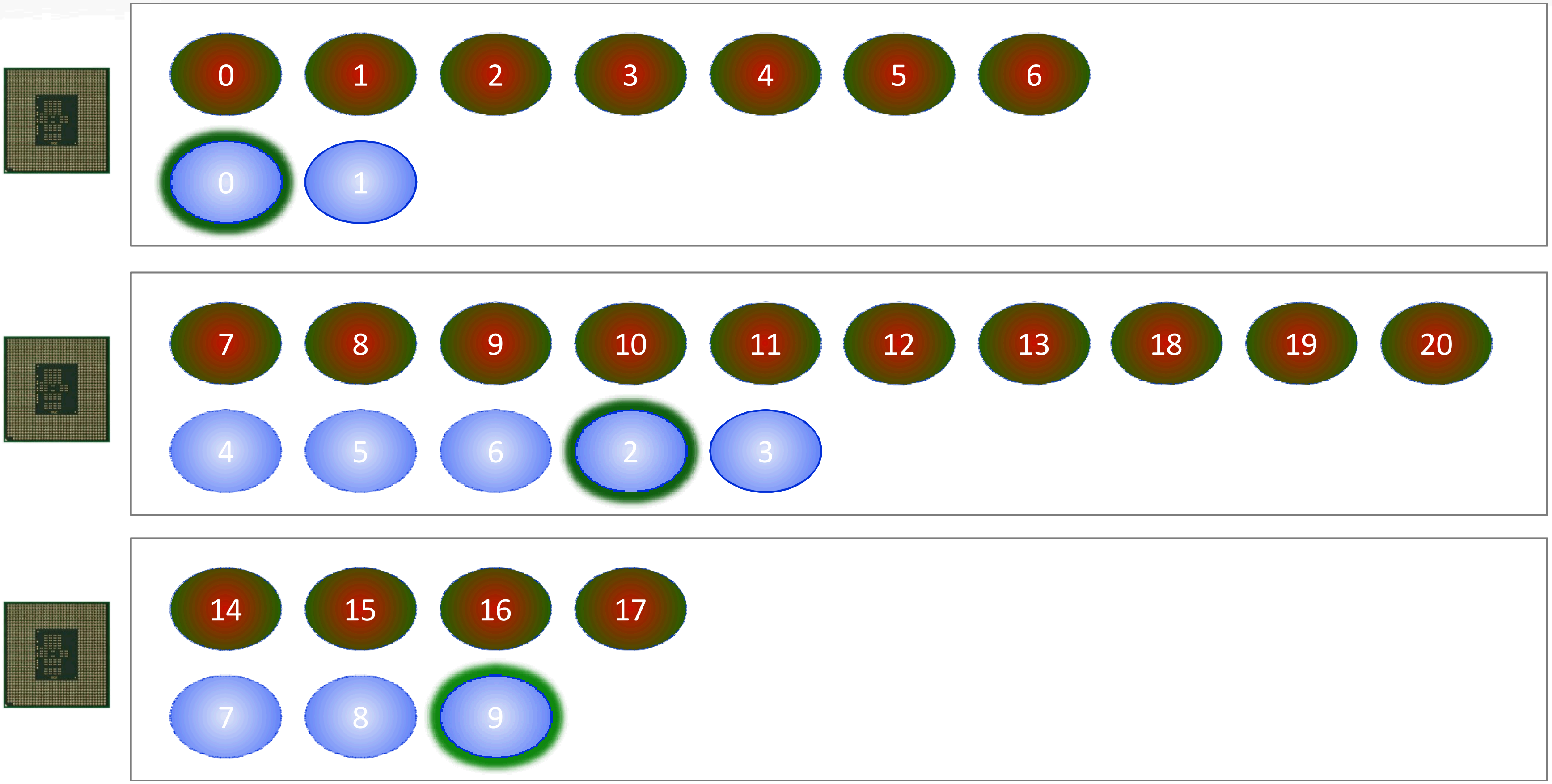
- Priority (**CriticalPathCollection**) > Priority(**AuxiliaryCollection**)
- Increase task parallelism while maintaining critical path

We are extending task collections to support multiple collections operating concurrently



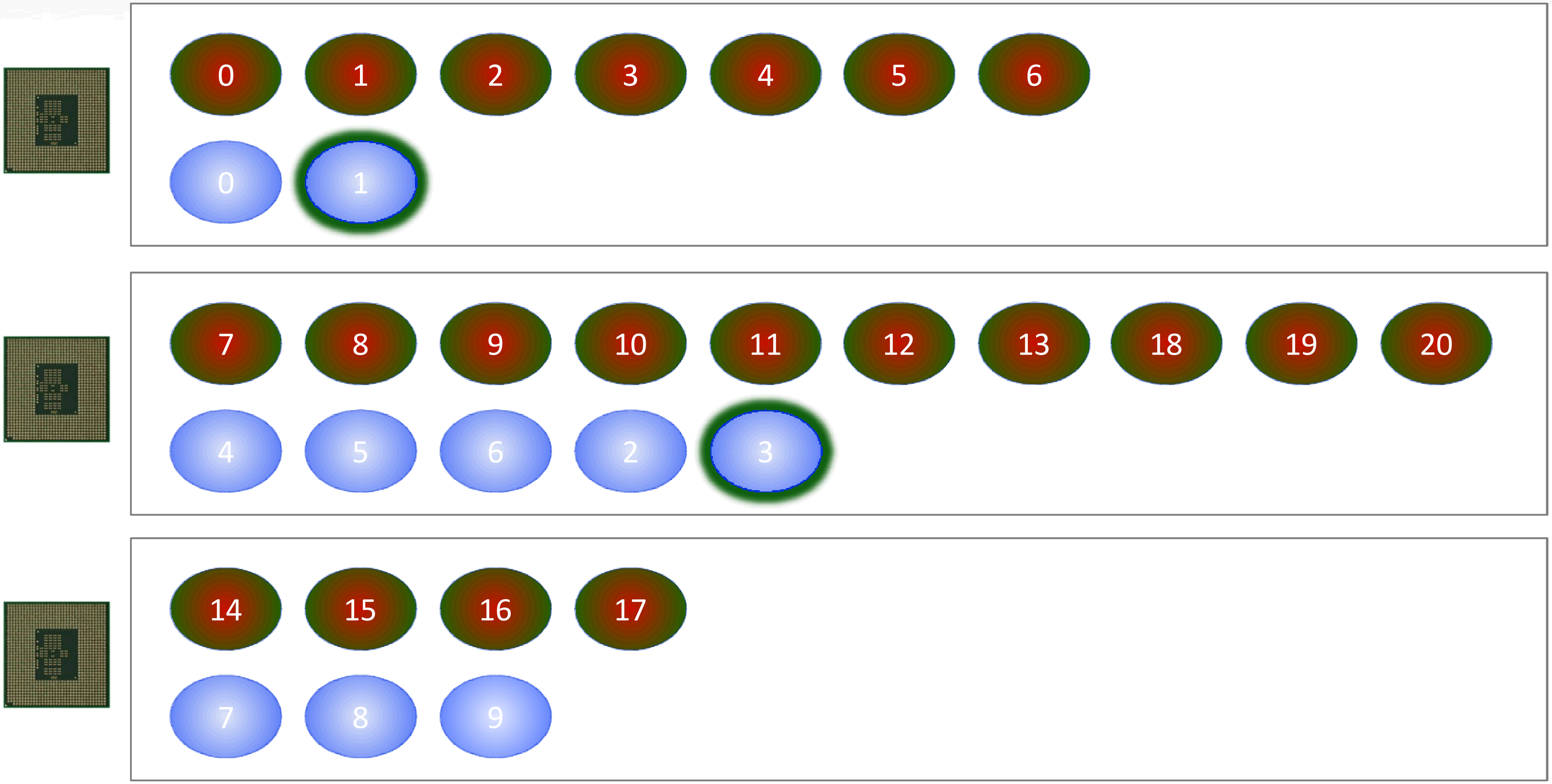
- Priority (**CriticalPathCollection**) > Priority(**AuxiliaryCollection**)
- Increase task parallelism while maintaining critical path

We are extending task collections to support multiple collections operating concurrently



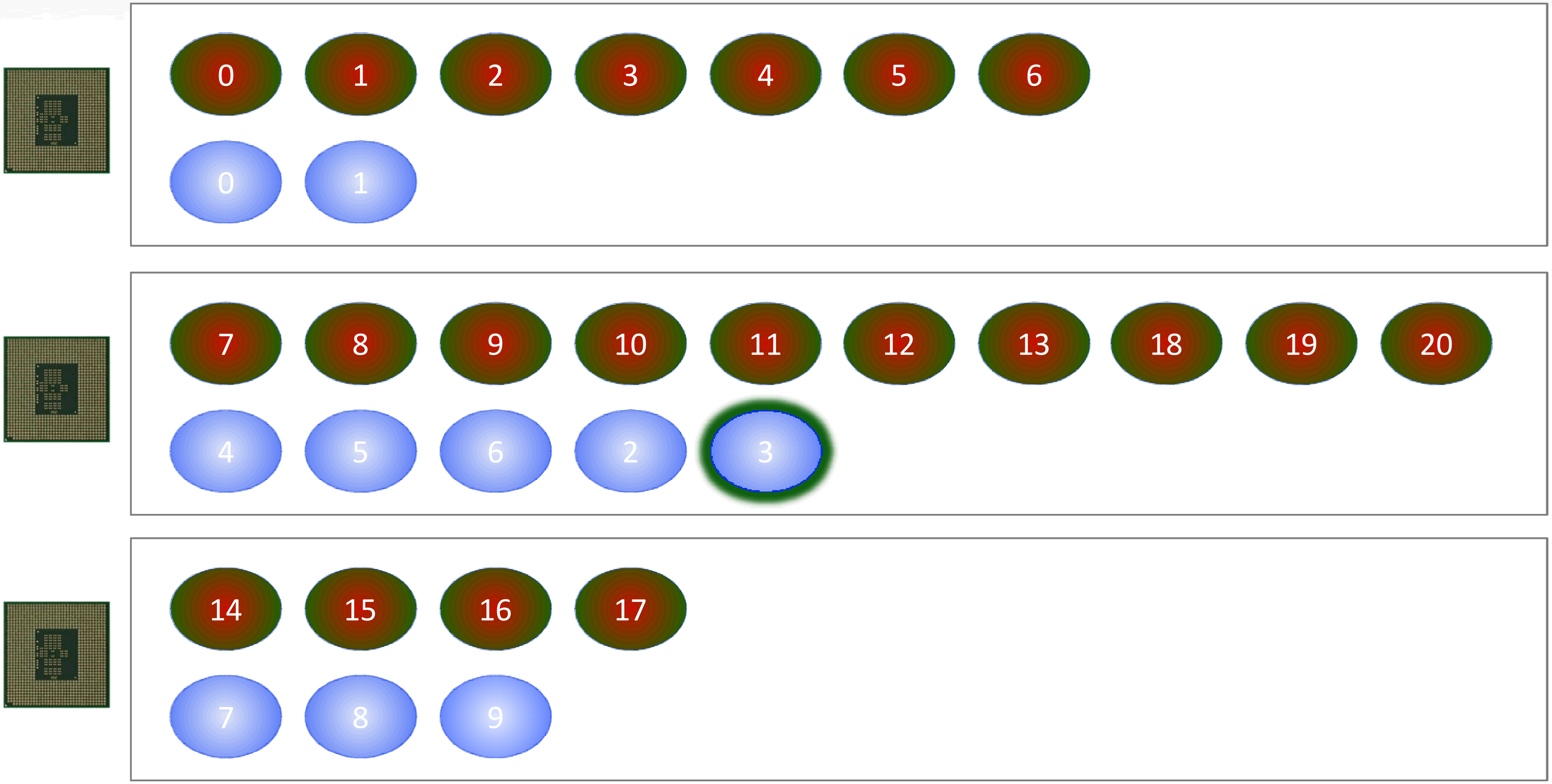
- Priority (**CriticalPathCollection**) > Priority(**AuxiliaryCollection**)
- Increase task parallelism while maintaining critical path

We are extending task collections to support multiple collections operating concurrently



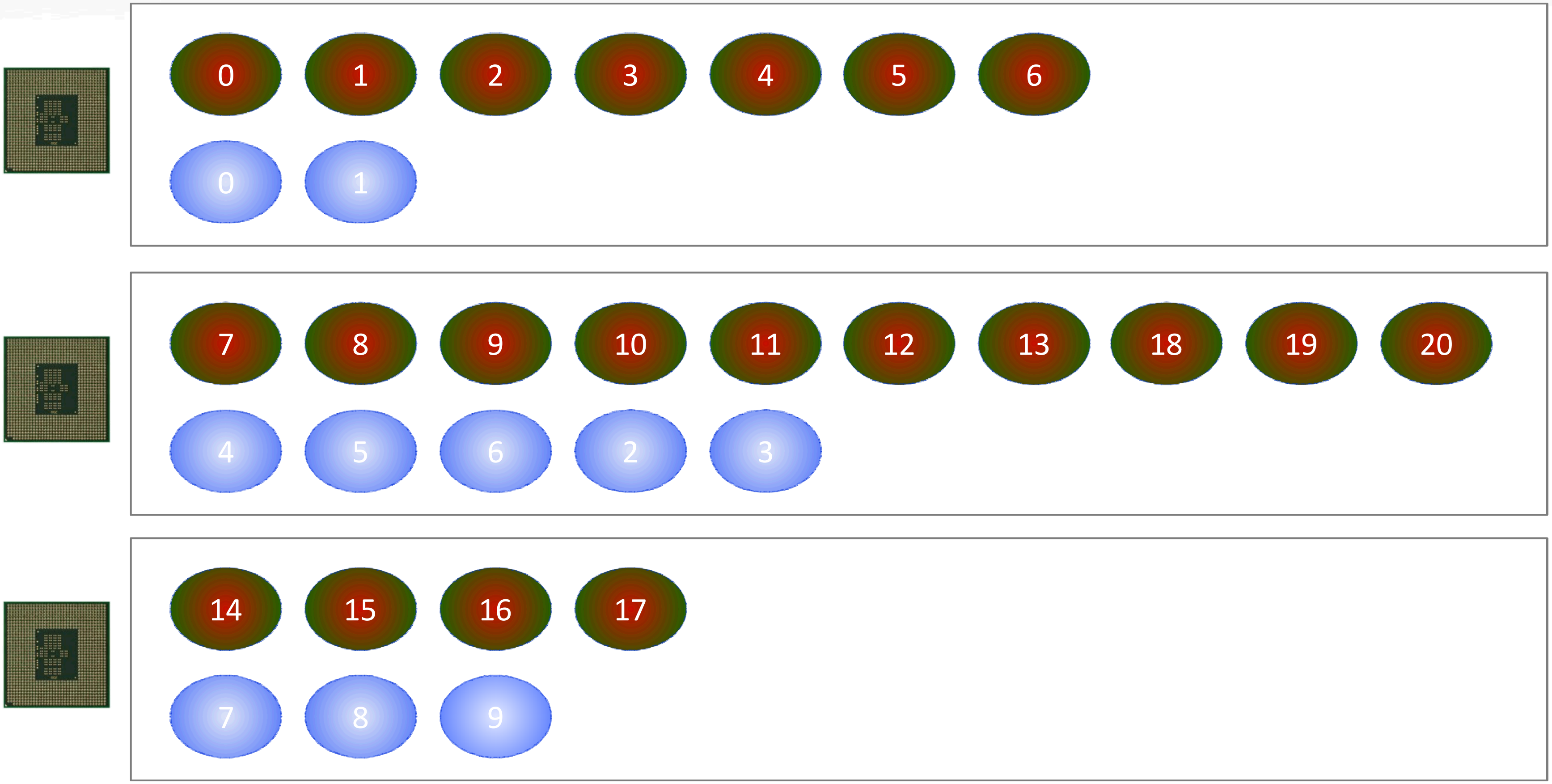
- Priority (**CriticalPathCollection**) > Priority(**AuxiliaryCollection**)
- Increase task parallelism while maintaining critical path

We are extending task collections to support multiple collections operating concurrently



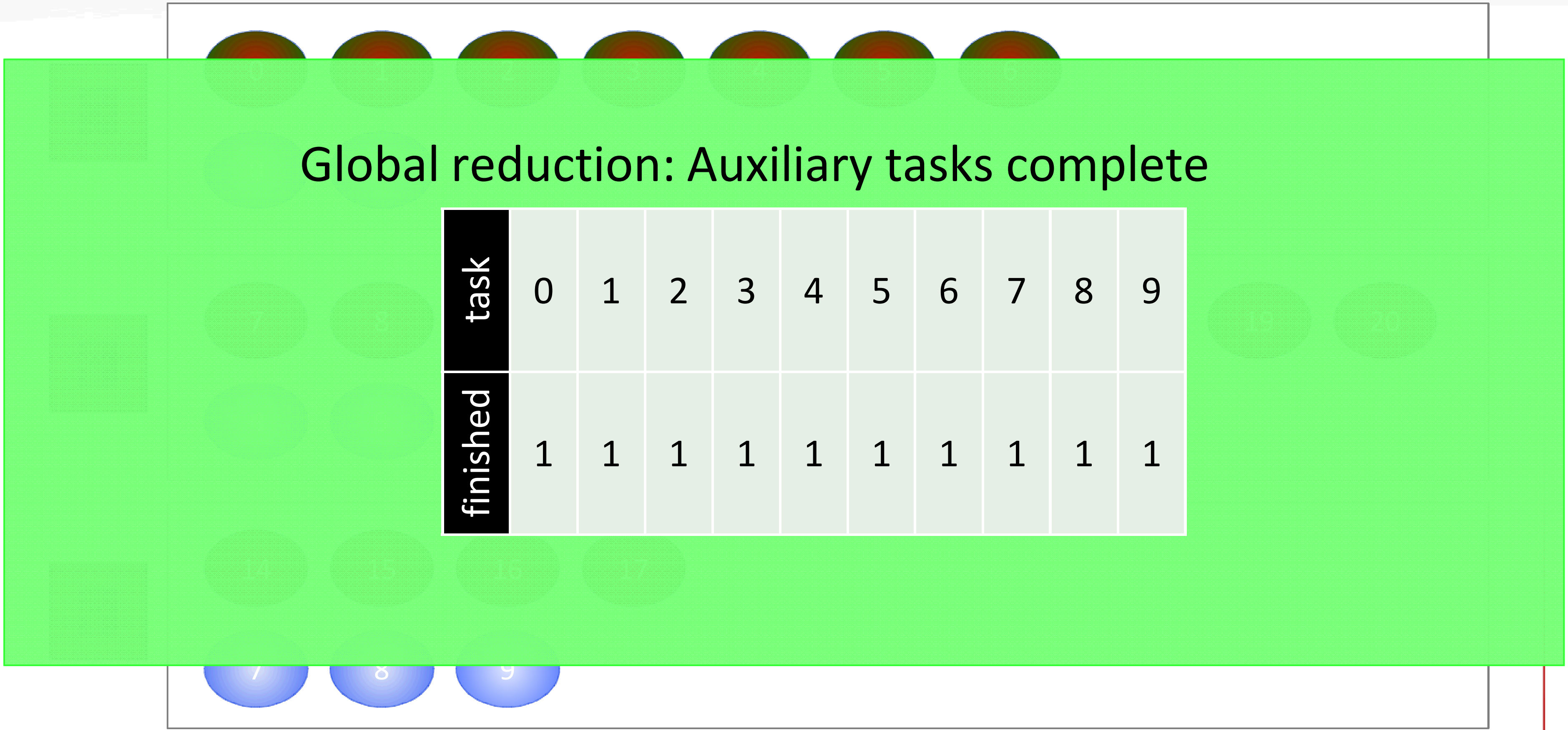
- Priority (**CriticalPathCollection**) > Priority(**AuxiliaryCollection**)
- Increase task parallelism while maintaining critical path

We are extending task collections to support multiple collections operating concurrently



- Priority (**CriticalPathCollection**) > Priority(**AuxiliaryCollection**)
- Increase task parallelism while maintaining critical path

We are extending task collections to support multiple collections operating concurrently



Global reduction: Auxiliary tasks complete

task	0	1	2	3	4	5	6	7	8	9
finished	1	1	1	1	1	1	1	1	1	1

- Priority (**CriticalPathCollection**) > Priority(**AuxiliaryCollection**)
- Increase task parallelism while maintaining critical path

Our goal: Discover the right approach for extreme-scale, fault-resilient computing

Exascale systems are expected to experience errors/faults much more frequently than petascale systems

On future systems we need to develop methods for:

- Reducing mean time to error
- Fault detection & recovery
- Fault-oblivious algorithms

Existing programming models are inherently not fault-resilient	Asynchronous many-task (AMT) programming models can be fault-resilient
Single Program Multiple Data (SPMD), implicitly synchronous algorithms cannot recover from failure nor adapt well to node degradation	Asynchronous execution and redundancy minimize the impact of node degradation/failure and benefit scalability even without failure
Global check-points no longer feasible	Synergistic with local check-pointing

OPPORTUNITY

Sandia has an active hiring program in computational R&D

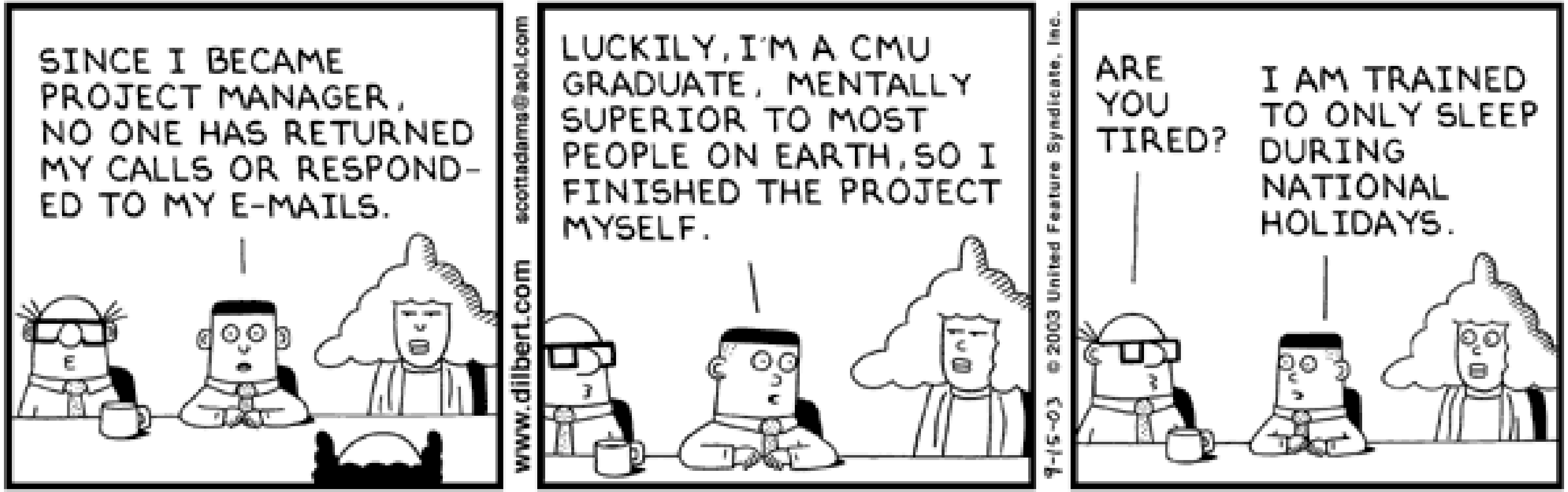
- Student Internships
  - Undergraduate and graduate
  - Typically summers, but not exclusively
- Hire at all levels
  - BS, MS, PhD
  - CS and Engineering concentrations
  - Full time, Limited term, Postdocs
- In addition to scalable computing, we are very active in cyber security.

Acknowledgements

- Rob Armstrong (Robust Stencils)
- Janine Bennett (pmodels)
- Gilbert Hendry (SST/macro)
- Mike Heroux (LFLR)
- Hemanth Kolla (pmodels)
- Jackson Mayo (Robust Stencils)
- Philippe Pebay (SST/macro)
- Nicole Slattengren (pmodels)
- Keita Teranishi (LFLR)
- Jeremiah Wilke (SST/macro)

Thank You

Robert L. Clay
rlclay@sandia.gov
+1 (209) 610-2929


SINCE I BECAME PROJECT MANAGER, NO ONE HAS RETURNED MY CALLS OR RESPONDED TO MY E-MAILS.
LUCKILY, I'M A CMU GRADUATE, MENTALLY SUPERIOR TO MOST PEOPLE ON EARTH, SO I FINISHED THE PROJECT MYSELF.
ARE YOU TIRED?
I AM TRAINED TO ONLY SLEEP DURING NATIONAL HOLIDAYS.
scottadams@aol.com
www.dilbert.com
9-15-03
© 2003 United Feature Syndicate, Inc.