Scalable Graph Algorithms for Cluster Finding

Raghavendra Kanakagiri, Laxmikant Kale,
Karthik Senthil and Edgar Solomnik
Parallel Programming Laboratory
University of Illinois Urbana Champaign

PPL UIUC PARALLEL PROGRAMMING LABORATORY

ChaNGa – Computational Cosmology

ChaNGa (CHArm++ N-body GrAvity solver) is built as a Charm++ application that employs a tree algorithm to represent the simulation space.

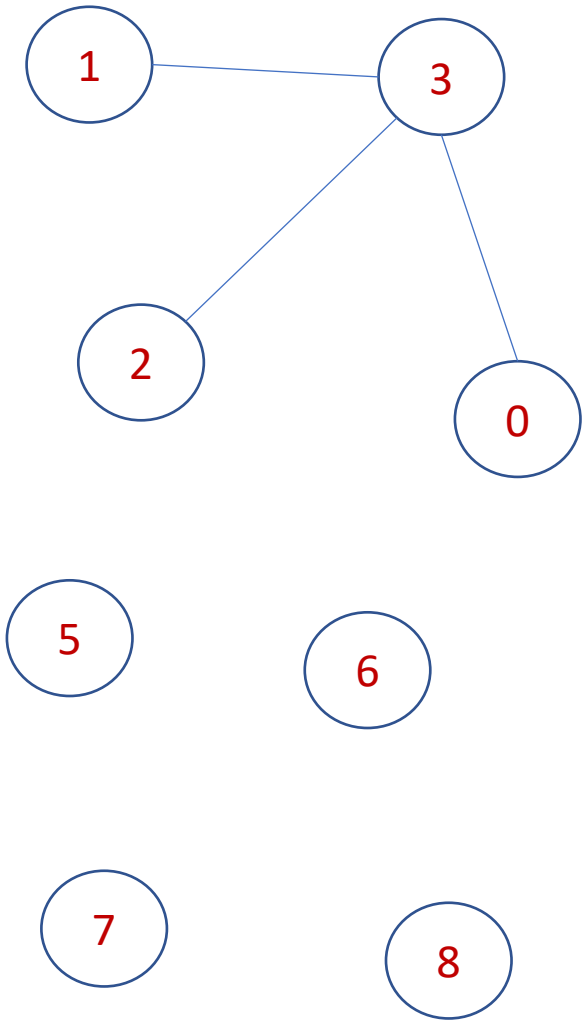
Edges are discovered during tree traversals.

Cluster finding.

Component Pruning: Tinier components need to be deleted.

Connected components detection

Vertices:



Vertices:
{0, 1, 2, 3, 5, 6, 7, 8}

Edges:
(1, 3)

(2, 3)

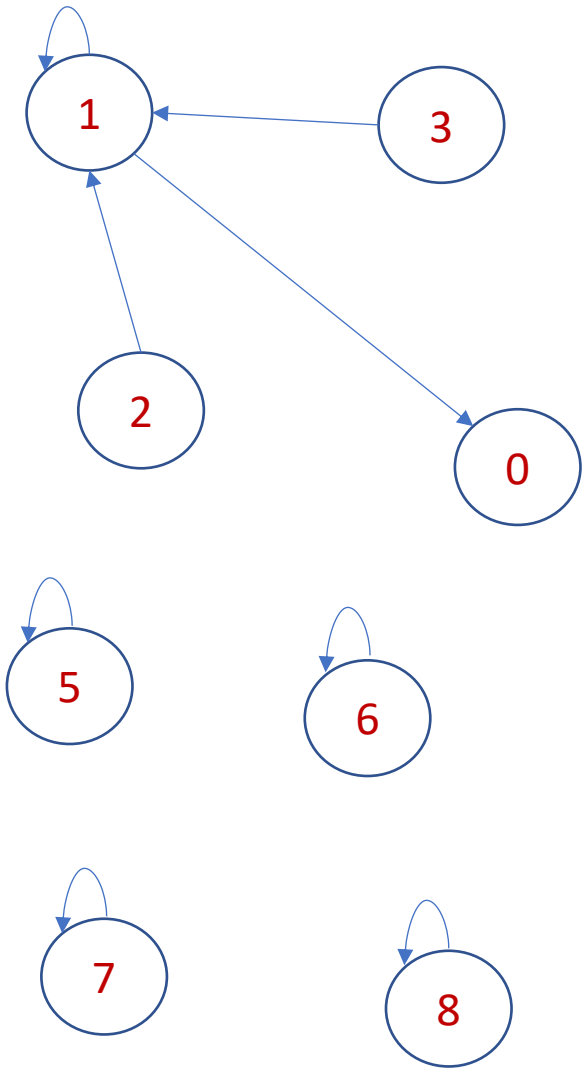
(0, 3)

Connected components: 5

- Detect connected components.
- Label these components.
- Prune components (1).

Anchor Algorithm

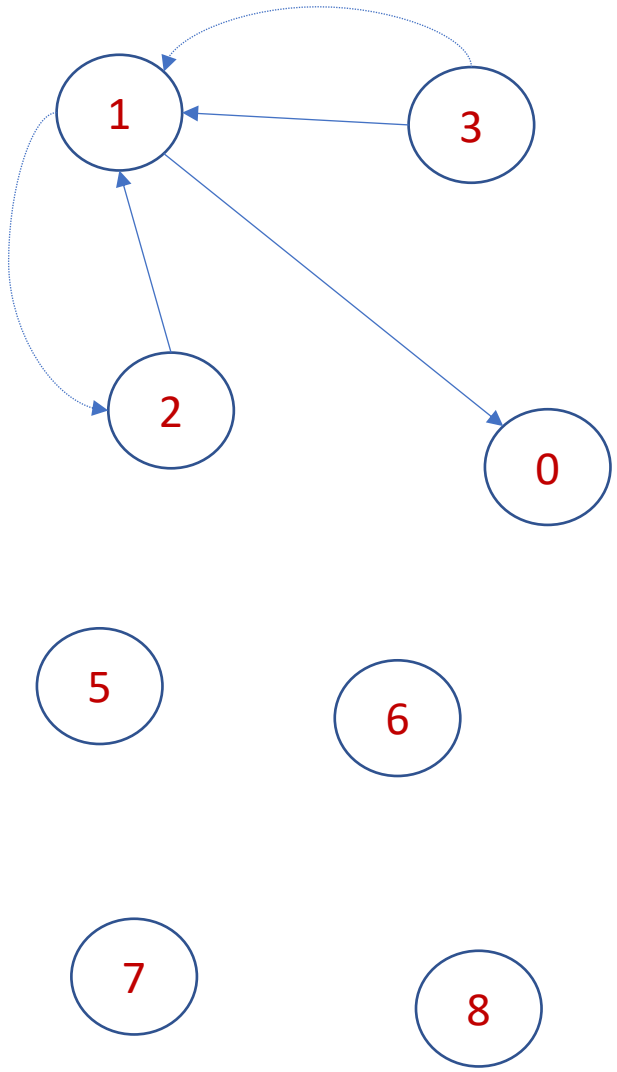
Vertices:



- Each vertex has a parent.
- Union Operation: Edge (a, b) -> union (a, b)
- Challenge: To prevent race condition
- Anchor (hook-up):
  - a -> b ; if a > b
  - b -> a ; if b <= a
- Union operations are executed in parallel.
- Edges can be generated on any PE.

Parallel Union-Find (Cluster finding)

Vertices:



Edges:

(1, 3)

Send a message to 3 to anchor itself to 1.

(2, 3)

Send a message to 3 to anchor itself to 2.

But, 3 has a parent, so it sends a message to its parent (1) to anchor to 2 (tree climbing).

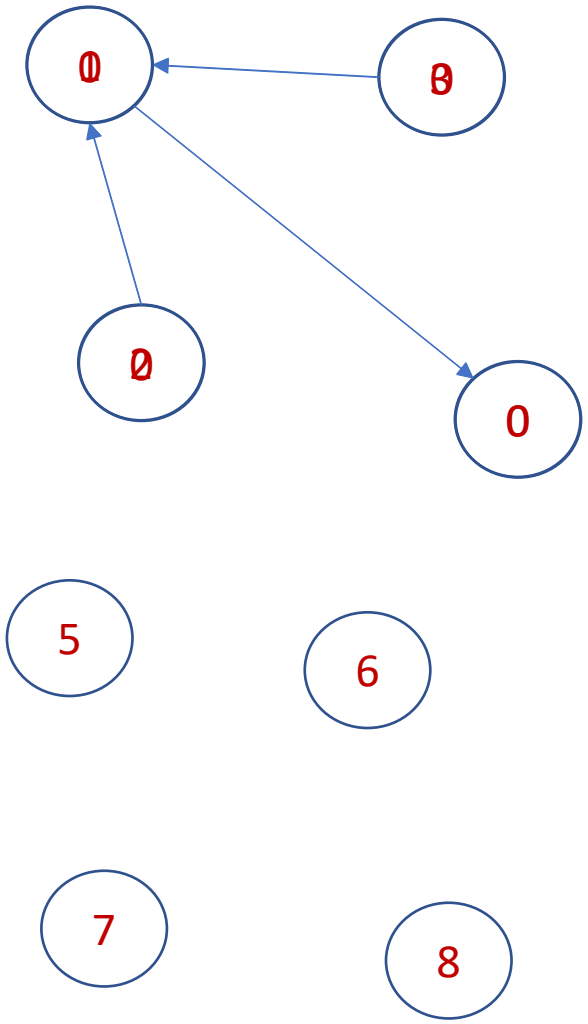
1 sees that 2 is > than itself, so it in-turn sends a message to 2 to anchor itself to 1.

2 obliges. (If 2 had a parent, the tree climbing would have happened here)

(0, 3)

Parallel Union-Find (Cluster finding)

Components:



Phases:

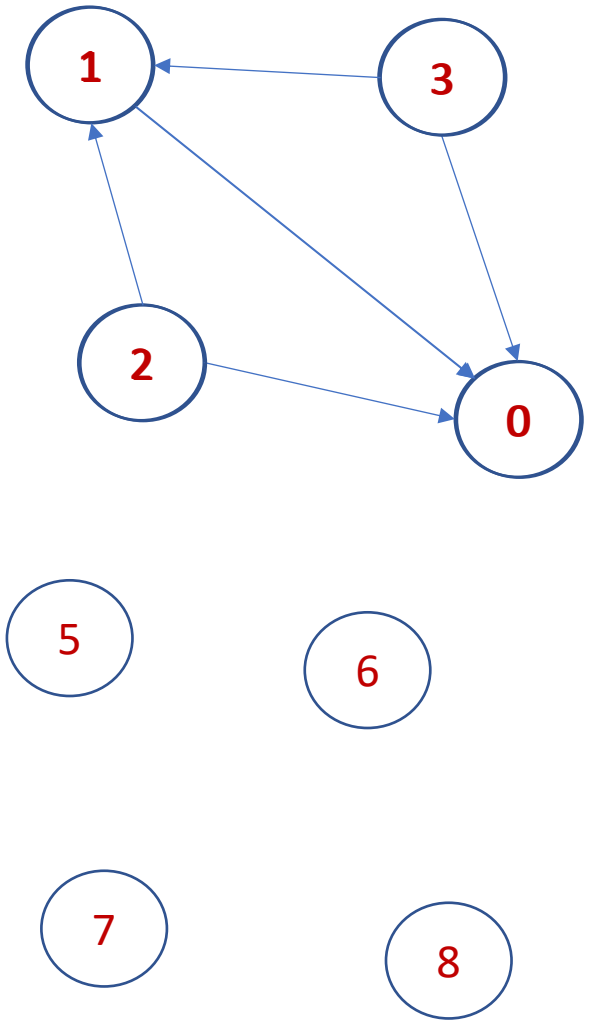
Tree construction

Component Labeling

Pruning (ChaNGa specific)

Parallel Union-Find (Cluster finding)

Vertices:



Path compression

Local path compression : within PE

Global path compression

Parallel Union-Find (Cluster finding)

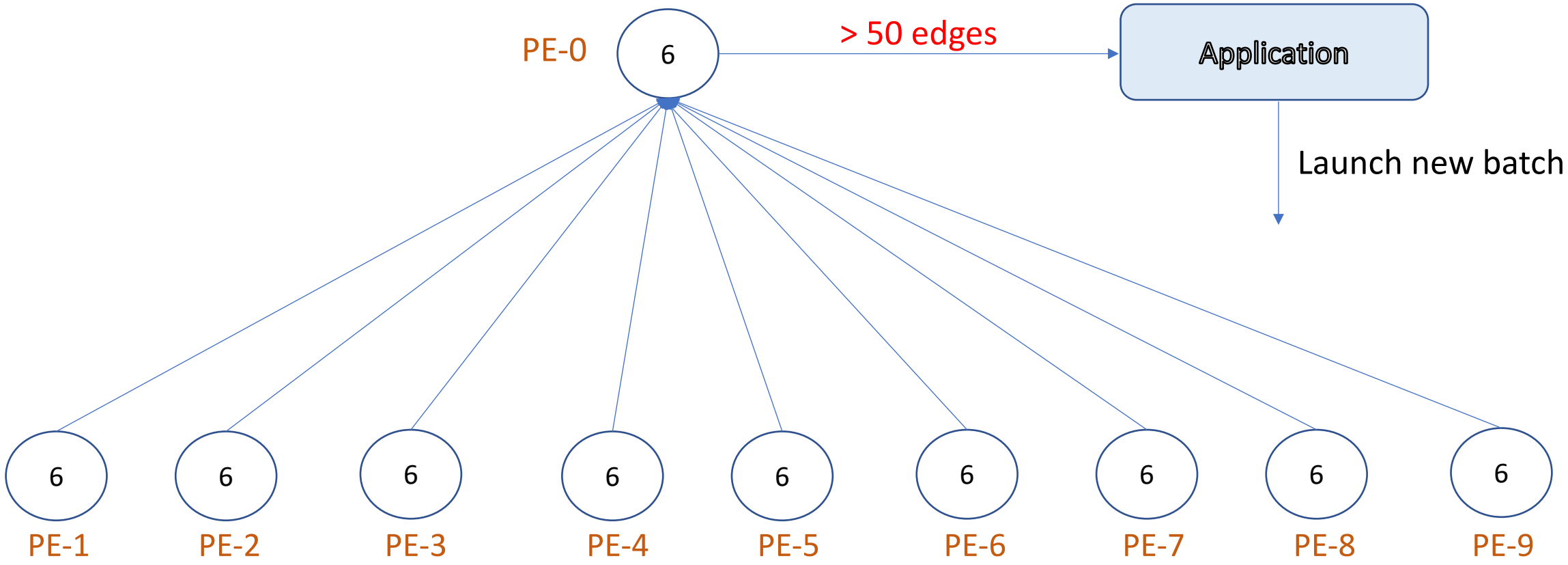
Problem: Too many messages clog the system.

Batch processing of edges in the union-find tree construction.

(Phase 1: batch 1) -> (Phase 1: batch 2) -> (Phase 1: batch N) -> Phase 2 -> Phase 3

Not known:

- How many total edges will be processed when the run starts.
- On which PE a single find traversal will terminate.
  - All union-find operations might terminate on a single PE!
- Who will inform the application to start a new batch?
  - None of the PEs know what the other PEs are doing.
- When to start a new batch?



10 PEs, 100 edges batch size.

<u>Can there be a deadlock?</u>

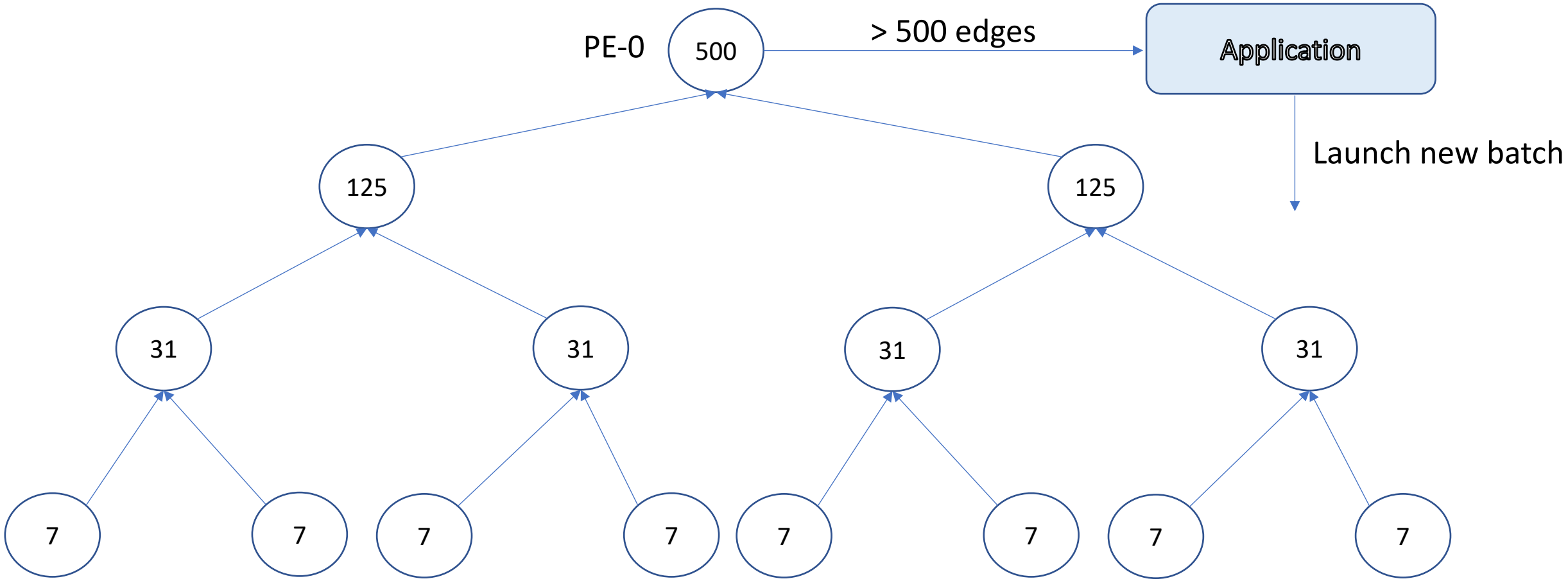
Batch Processing

Example scenario: 10 PEs with a batch size of 100 edges.

<u>Algorithm</u>:

- Each PE informs PE[0] as soon as it sees 6 union-find terminations.
- PE[0] informs the application to start a new batch as soon as it receives 50 union-find terminations.

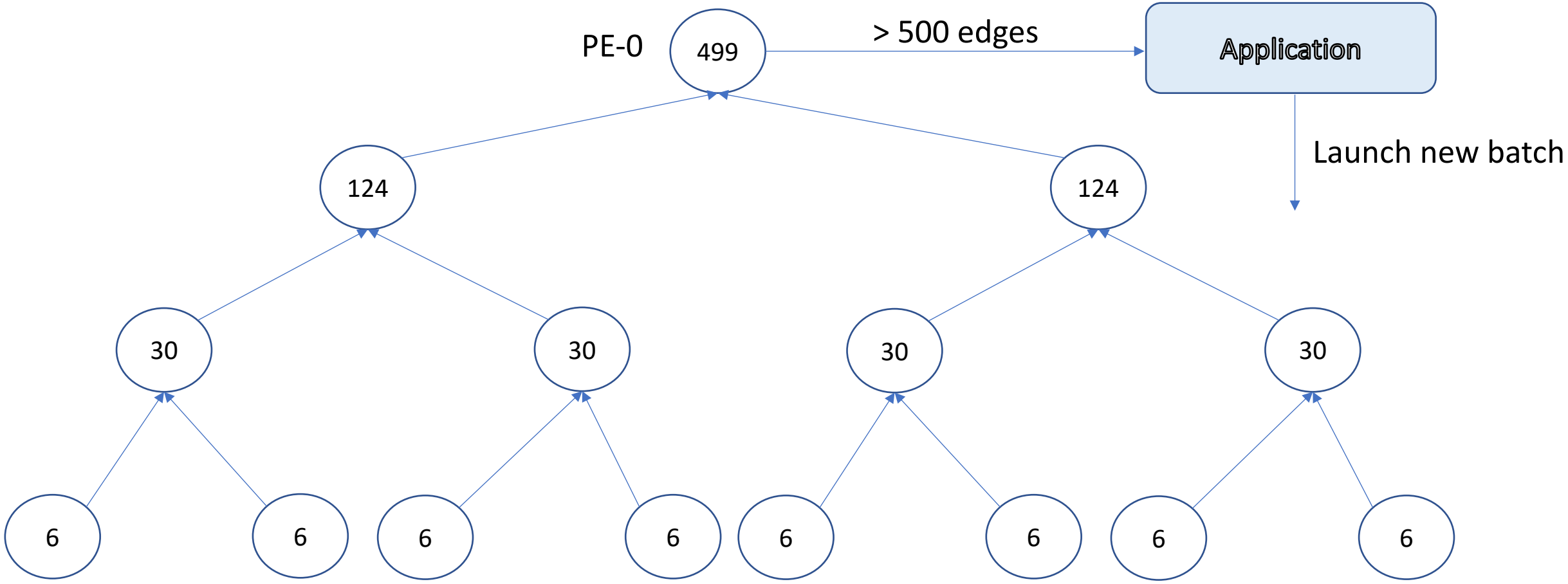
- In general (avg + 1) as the trigger point to send a message to PE-0 will work, where
  avg = batch size/# PEs. The trigger point should be changed if the trigger point for PE-0 to inform the application to start a new batch changes (but can similarly be derived).



15 PEs, 1000 edges batch size

Varying thresholds. (1/4)

<u>Can there be a deadlock?</u>

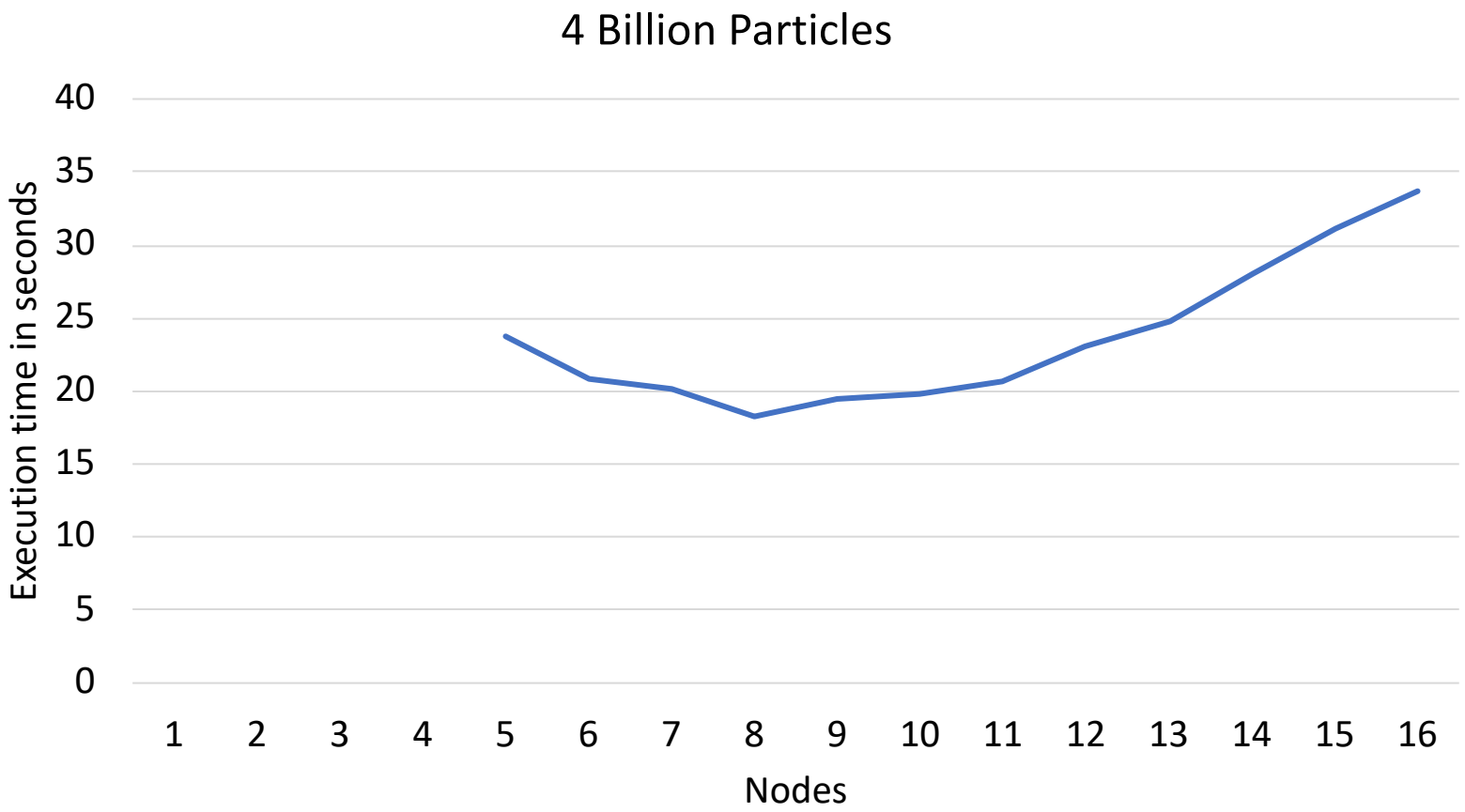


15 PEs, 1000 edges batch size

<u>Scenario 1</u>: 499 + 124 * 2 + 30 * 4 + 6 * 8 = 915

85 edges remaining to be processed.

Parallel Union-Find (Cluster finding)



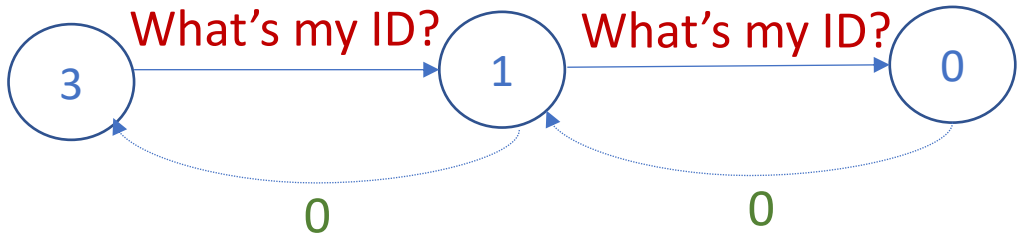
<u>Stampede2</u>

68 cores per node
4 hardware threads
272 threads per node

per node 7 mpi_tasks/logical nodes
+ppn 8
+pemap 1-8,10-17,19-26,28-35,37-44,46-53,55-62
+commap 0,9,18,27,36,45,54
ofi layer

Component Labeling

Iterate over the vertices in the node, and only those vertices that have their parent in a different node send messages to their parent requesting for a component ID. (The number of messages sent is recorded)



Root replies back the component ID.

Only after all the messages are received, iterate over the vertices, and label them.

Quiescence Detection (QD) -> Inform the application.

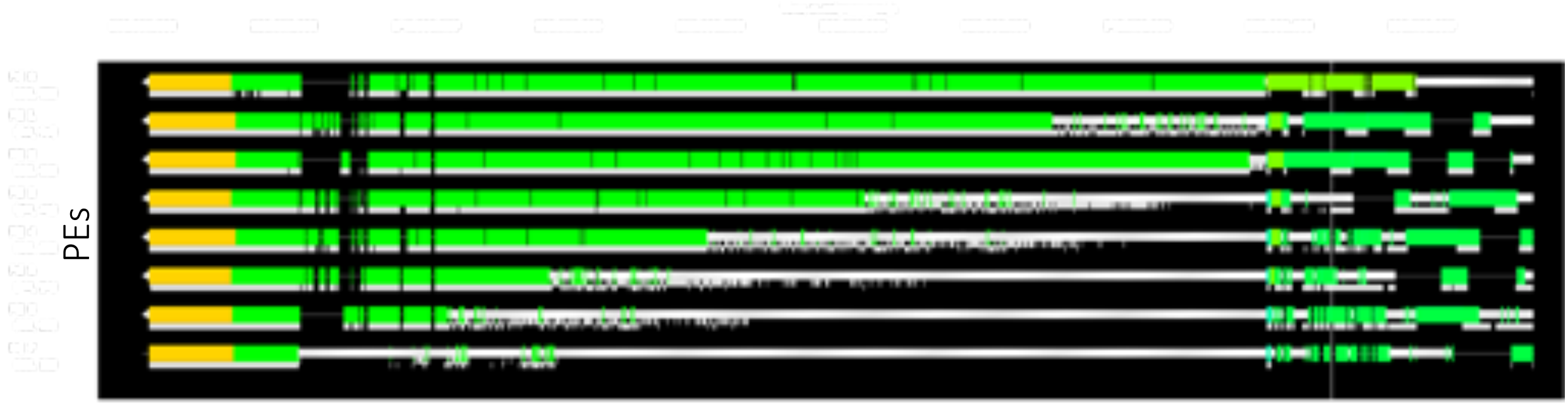
Parallel Union-Find (Cluster finding)

Orkut data set:

	tree construction time (s)	component detection (s)
1 PE	84.3	0.8
64 PEs	80.3	7.4
256 PEs	85.3	3.4
512 PEs (8 nodes)	51.4	1.7
768 PEs	131.3	1.2
1024 PEs	154.5	1

Projections

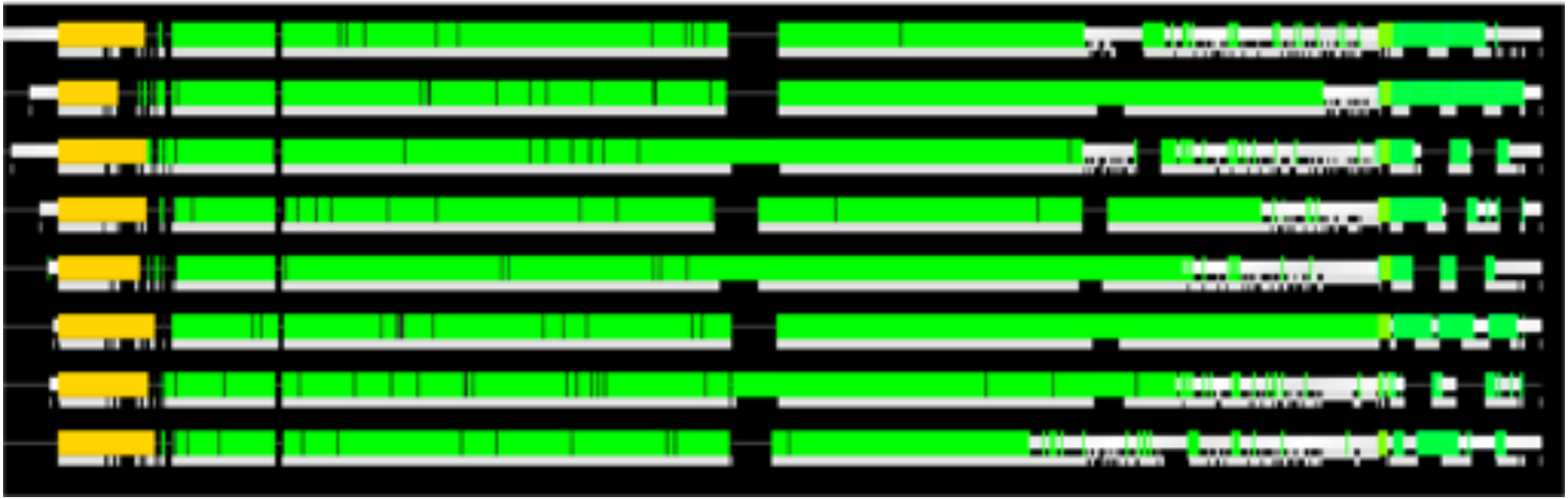
Orkut (8PEs)



Timeline

Projections

Orkut: Strided (8PEs)



Timeline

Distribution of vertices.

Parallel Union-Find (Cluster finding)

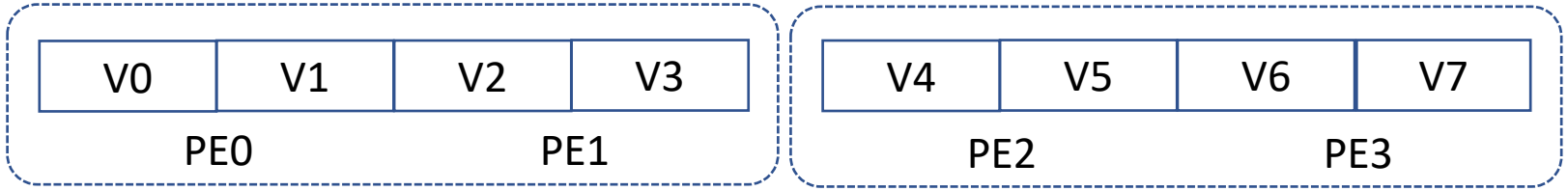
Other optimizations (Results from Orkut):

- Do component labeling (global path compression) at regular intervals.
  Phase 1 -> Phase 2 -> Phase 1 -> …
  Phase 1 -> global path compression -> Phase 2 -> …

512 PEs	
# batches	total_time
1	56.9
2	94.4
3	153.2
5	225.4

- Drop an edge if both its vertices belong to the same component.

512 PEs	
# batches	total_time
2	81.7
3	85.1
5	141.9



- Shared parent data structure across PEs.
  - Phase 1 still sends messages so as to avoid race conditions during tree construction.
  - More edges can be dropped (component label information is treated as read-only).

Parallel Union-Find (Cluster finding)

Other optimizations:

- Overlap Phase 1 and Phase 2.
- Mini batches of edge injection.
- Child pointers at vertices.

Related Work

Related work:

- ParConnect: A parallel connectivity algorithm for de Bruijn graphs in metagenomic applications by Patrick Flick, Chirag Jain, Tony Pan and Srinivas Aluru (SC 2016).
- on de Bruijn graphs: 1.8B vertices; 135B vertices + edges; 22 minutes on 1280 cores.
- LACC: A Linear-Algebraic Algorithm for Finding Connected Components in Distributed Memory by Ariful Azad and Aydin Buluc (IPDPS 2019). (~5.1x faster than ParConnect)

- ChaNGa: Edges are generated dynamically.

Backup Slides

Component Pruning

Up-Pass:

A node sends component count to its respective parent.

Parent knows how many counts it will receive for that component. (Because in the labelling phase the count of the buffered requests was stored).

Adds them up and sends the sum to its parent.

Root finally has the total count.

Down-Pass:

The root sends back the total count. The total count is trickled down. Component pruning.

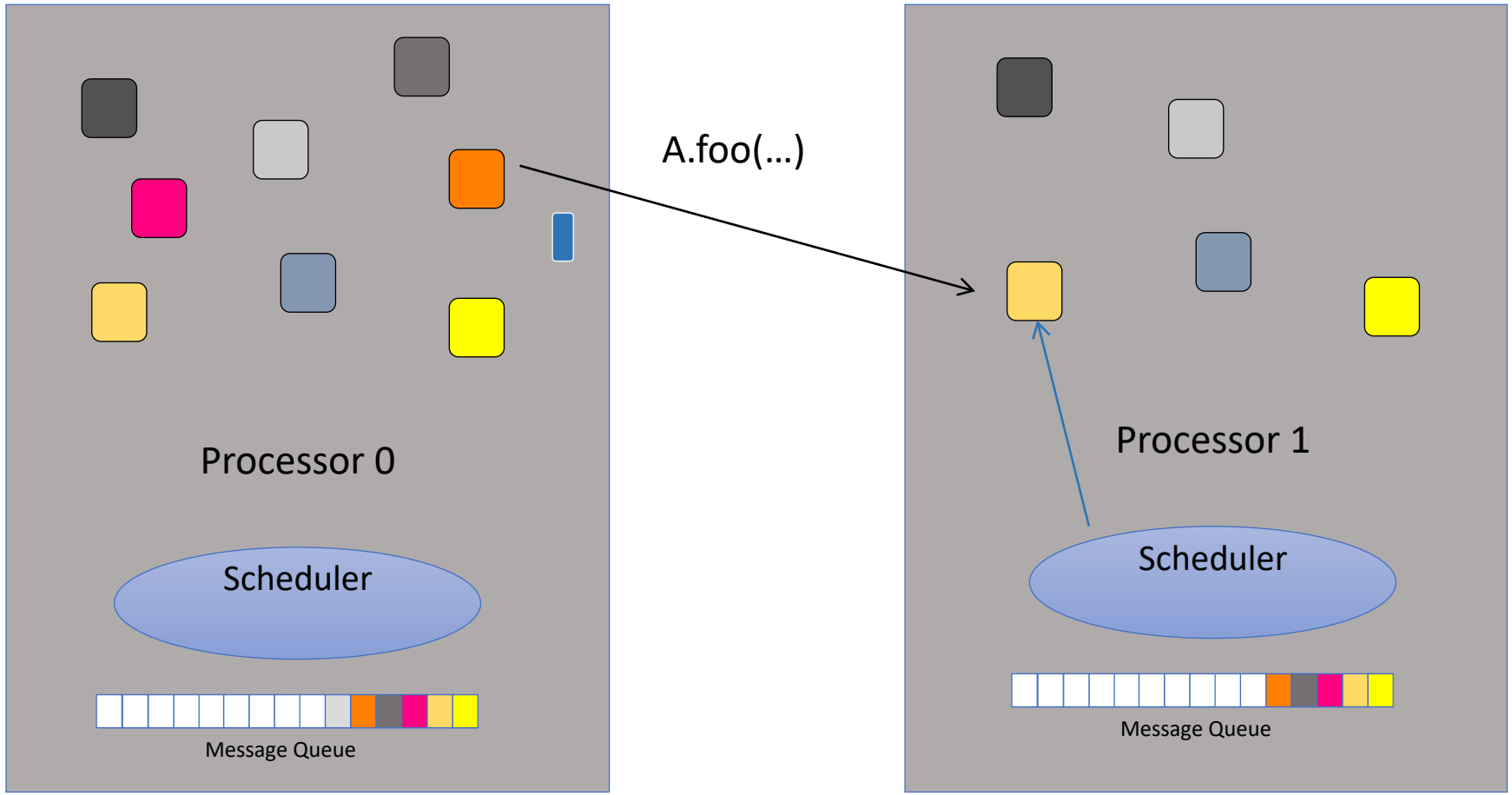
Charm++

Charm++ is a generalized approach to writing parallel programs.

Three design principles: Overdecomposition, Migratability, and Aysnchrony.

Smallest Unit: Chare

Overview



Source: Charm Workshop '18

Charm++

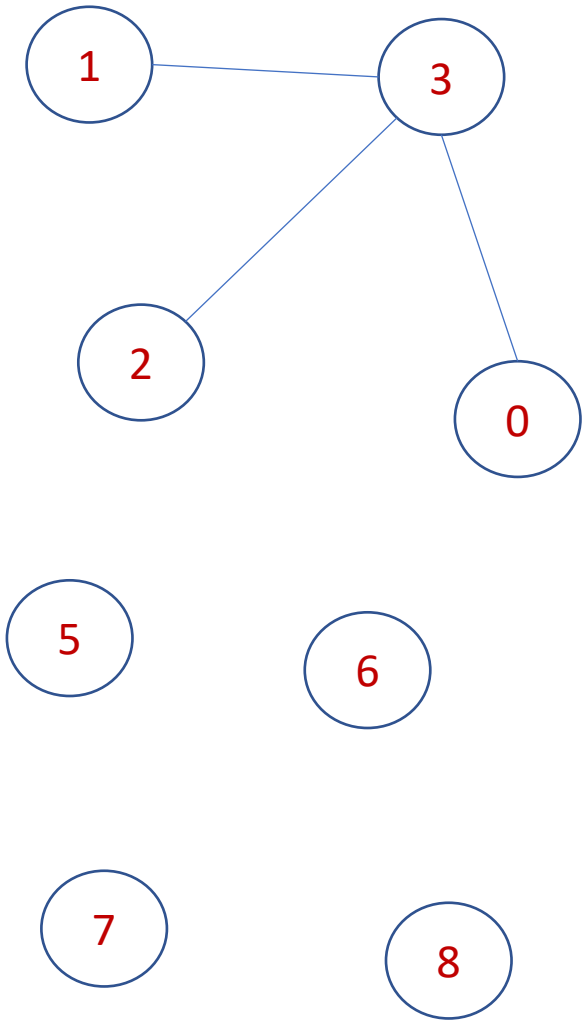
Chare is the smallest unit.

Application execution is distributed across Processing Elements (PEs), which are OS threads or processes. On each PE, there is a scheduler operating with its own private pool of messages.

Group chares when created exist on every PE.

Anchor Algorithm

Vertices:



We leverage a parallel data-structure, the union forest, $T = (V, E_T)$. The union forest leverages a global ordering $\prec$ of the vertices in V, in particular, it ensures that if v is a parent of w, so $(v, w) \in E_T$, then $v \prec w$. Our analysis assumes this ordering is random. The data structure support ANCHOR queries. Each vertex in T knows only its parent,

$$\pi(v) = \begin{cases} u : (u, v) \in E_T \\ v : \nexists u \in V \text{s.t.} (u, v) \in E_T \end{cases} .$$

Algorithm 2 $[u]$ =ANCHOR$\langle w \rangle(T, v)$

Require: $T = (V, E_T)$ is a forest, if $(s, t) \in E_T$, $\pi(t) = s$, if t is a root $\pi(t) = t$. $v, w \in V$. The anchor function is invoked on the vertex w (only $\pi(w)$ is known).

Ensure: If v and w are not in the same tree, an edge is added to E_T to connect the smaller of the two roots (r) of the trees containing v and w, to the first ancestor of the other vertex (p) satisfying $p \prec r$, in particular $\pi(r) = p$.

```
if w ≺ v then
    ANCHOR⟨π(v)⟩(T, w)
else if π(w) = w then
    π(w) = v
else
    ANCHOR⟨π(w)⟩(T, v)
end if
```

Batch Processing

Example scenario: 10 PEs with a batch size of 100 edges.

- Scenario-1: 9 PEs witnessed 5 terminations, so PE-x will observe 55 terminations, hence would have informed PE-0 9 times that it has seen 6 terminations. PE-0 would have informed the application to start a new batch.
- Scenario-2: 4 PEs: 5; 4 PEs: 11. The remaining PEs should see 34 among them. This still does not lead to a deadlock.