# ACM SIGACT News Distributed Computing Column 26

Sergio Rajsbaum*

**Abstract**

The Distributed Computing Column covers the theory of systems that are composed of a number of interacting computing elements. These include problems of communication and networking, databases, distributed shared memory, multiprocessor architectures, operating systems, verification, Internet, and the Web. This issue consists of:

- "A Collection of Kinesthetic Learning Activities for a Course on Distributed Computing," by Paolo Sivilotti and Scott Pike.

Many thanks to them for their contributions to this issue.

**Request for Collaborations:** Please send me any suggestions for material I should be including in this column, including news and communications, open problems, and authors willing to write a guest column or to review an event related to theory of distributed computing.

**Congratulations to Nancy Lynch:** Finally, it is fitting in this column to congratulate Nancy Lynch, who is the winner of the 8th Knuth Prize, awarded to her for her seminal and influential contributions to the theory of distributed computing.

*Instituto de Matemáticas, Universidad Nacional Autónoma de México. Ciudad Universitaria, Mexico City, D.F. 04510. rajsbaum@math.unam.mx

# A Collection of Kinesthetic Learning Activities for a Course on Distributed Computing

Paolo A. G. Sivilotti [1] and Scott M. Pike[2]

**Abstract**

Kinesthetic learning is a process where students learn by actively carrying out physical activities rather than by passively listening to lectures. Pedagogical research has indicated that kinesthetic learning is a fundamental, powerful, and ubiquitous learning style. It resonates with many students across all disciplines and levels of education. The adoption of kinesthetic approaches in college classrooms, however, has been hampered by the difficulty of designing effective activities, as well as the perceived challenges of coordinating these activities. On the other hand, courses on distributed computing, by the very nature of the material they cover, are uniquely suited to exploiting this learning technique. We have developed and piloted a collection of kinesthetic activities for a senior undergraduate or graduate-level course on distributed systems. We give detailed descriptions of these exercises and discuss factors that contribute to their success (or failure). Our hope is that others will adopt these particular activities in their own distributed systems courses as well as use these examples as a pattern for developing new activities.

## 1 Introduction

A kinesthetic learning activity (KLA) is a pedagogical tool involving physical movement by students. As part of such an activity, students might stand, walk, talk, point, or even work with props. The key characteristics of a KLA are that (i) students are actively, physically engaged in the exposition and assimilation of classroom material, and (ii) this engagement directly supports some specific learning objective.

These activities can be effective in practically any learning environment, from elementary school to the college classroom, from small seminar groups to large lecture auditoriums, from the liberal arts to the sciences and professional schools, and from theoretical academic graded courses to applied industrial training sessions [Gri94, Zim03, Ros87, BLWH00].

Most college courses rely primarily on the traditional lecture-based format for instruction. Even when supplemented with visual slides, the traditional lecture-based format is primarily a passive form of education. As such, this format often suffers from decreased student engagement, frequent student inattention, and the exclusion of nonverbal learning modalities [HD78, Cas85, Bon96, Hak98].[3]

[1]CSE Department, The Ohio State University, Columbus OH, 43210. paolo@cse.ohio-state.edu

[2]CS Department, Texas A&M University, College Station TX, 77843. pike@cs.tamu.edu

[3]Not all traditional lectures suffer from these inadequacies. A well-designed and well-delivered lecture is a thing of beauty and is a powerful learning event for the student. However, such masterful lectures appear to be the exception rather than the norm.

KLAs serve to offset these shortcomings. They can be used in the middle of a long lecture to re-energize and re-focus the class by creating a new perspective from which to consider the topic. Beyond the short-term effect, including KLAs on a regular basis can have a fundamental impact on the classroom culture of interaction. As a side effect of these activities, students learn each other's names and become more comfortable asking questions and participating in group discussions. This raises the level of engagement during the periods of traditional lecturing. Finally, while traditional lectures appeal primarily to a single learning style, research in pedagogy indicates that multiple modalities are more effective [FKM03, LD06], so incorporating KLAs broadens the scope of students who achieve positive learning outcomes.

Recently, kinesthetic learning has enjoyed increased prominence within the computer science education community [BLWH00, BGW04, BWF07]. A workshop on designing kinesthetic learning activities has been held at the annual SIGCSE conference each year since 2004. Two templates have emerged as common themes in the design of KLAs for computer science topics: algorithms enacted by people and data structures built from people. Both of these templates naturally promote an awareness of concurrency (since participants can be simultaneously active) and locality (since cognitive and physical constraints limit how much a single participant can do). It is not surprising, then, that KLAs are *particularly* well-suited to a course on distributed computing, where concurrency and locality are ubiquitous and fundamental themes.

In this paper, we present a collection of KLAs we have designed and piloted for a senior undergraduate or graduate-level course on distributed systems. (Although our focus is on college education, some of these activities have also been adapted for use in outreach activities to K-12 audiences.) Based on the experiences of piloting these activities in our own courses, we give detailed directions for conducting each activity including materials and logistics. For each activity, we describe the learning objectives it is meant to support, and delineate the most important elements contributing to its success. We also give some general advice about how to design new KLAs. An abridged version of this paper, conaining a subset of the the KLAs described here, appeared at SIGCSE 2007[SP07].

Our hope is that others will adopt these particular activities in their own distributed systems courses as well as use the examples in this collection as a pattern for developing new activities.

# 2 Background

## 2.1 Personality Types and Learning Styles

The modern approach to psychological typology traces its origins to the work of C. G. Jung, who introduced a classic taxonomy of human personality [Jun92]. This work became the foundation for the Myers-Briggs Type Indicator, which divides personality types along four dimensions: introvert-extrovert, sensing-intuitive, thinking-feeling, and perceiving-judging [MMQH98]. This popular instrument is used today in many settings including career counseling, team formation, and conflict resolution.

Personality type theory has also been applied in the educational setting, where evidence suggests the existence of a variety of learning styles (and complementary teaching styles) [ANH78, BEH$^+$56]. Several models of student learning styles have been developed, along with instruments for assessing an individual's learning style. Examples include: the Kolb Learning Style

Model [Kol83], the Herrmann Brain Dominance Instrument [Her89], the Felder-Silverman Learning Style Model [FS88], and the VARK learning style inventory [Fle02]. All of these models have been successfully applied to higher education in general and to engineering education in particular [McC90, Sti87, LL95, Fel93, BJY99].

Although these various models differ in the particular dimensions they use to classify learning styles, they share in the recognition that no single learning type fits everyone. While some students assimilate visual information best, others prefer auditory information. While some students prefer information to be structured as facts about things, others prefer a structure based on relationships among things. While some students prefer starting with first principles and using deductive reasoning, others prefer starting with examples and using inductive or abductive approaches.

In order to accommodate this variety of learning styles, educators should strive for a balance of teaching styles. Multimodal delivery increases the chance of a match between teaching and learning styles [CS95, Dai94, RRS+95]. Incorporating kinesthetic teaching in the classroom is a step toward achieving this balance.

## 2.2 Common Myths About Kinesthetic Activities

**Myth #1.** *Kinesthetic activities take too long to prepare.*

While developing a new activity from scratch can be time consuming, adopting an existing activity (such as one presented in this article) reduces the start-up time commitment. Even an existing activity, however, does require some preparation time before running it for the first time. It is important to examine its description carefully, to understand the logistics thoroughly, and to anticipate potential problems. Time spent in preparation can make the difference between a smooth, effective learning activity and chaos. This effort, however, is amortized over many subsequent offerings. With each iteration, one becomes more comfortable with the activity, and the time needed to prepare decreases.

**Myth #2.** *Kinesthetic activities take up too much class time, resulting in less material being covered.*

Although it is true that some lecture time must be sacrificed in order to complete a kinesthetic learning activity, often these activities can be quite short. Some activities take only a few minutes to complete, even allowing for set-up and tear-down time. Furthermore, the primary aim of a course should not be to simply cover material, but rather to educate. The sacrifice of a few minutes of lecture time is well worth making if, as a result, students achieve a deeper understanding of an important course concept.

**Myth #3.** *Kinesthetic activities are too disruptive, resulting in a loss of control of the class, especially for large sections.*

A carefully prepared kinesthetic learning activity is a controlled, directed, and focussed form of class involvement. Although such involvement often energizes a class, the resulting heightened engagement is a positive, not negative, outcome. This assessment is especially true for large sections, where student detachment is more likely to be a problem. Running a kinesthetic activity for a large section does involve special planning, but is not significantly more challenging than for smaller sections.

**Myth #4.** *Computer science students are more technical and less kinesthetic.*

Although the distribution of learning styles among engineering students is not perfectly uniform (for example, a majority appear to be visual learners), all styles are, in fact, represented in significant proportions [Zyw03, FB05, Ros99]. Furthermore, kinesthetic activities have many beneficial side-effects, such as increasing student engagement and promoting open classroom discussion, which in turn facilitate other, non-kinesthetic, teaching methods. Finally, there are genuine opportunities for cooperation and team building via kinesthetic learning activities, which may help address a common weakness among CS students.

### 2.3 The Perils of Anthropomorphism

In 1985, Dijkstra famously warned of the dangers of anthropomorphism in science in general [Dij85] and in distributed computing in particular:

> ...it became a driving force behind the habit of describing computing systems in anthropomorphic terminology. ("When this guy wants to speak to that guy..." in reference to two components of a computer network.) Having that habit is a severe handicap, and too many people suffer from it. As long as we don't shed it, computing science will remain immature.[DS90, p.122]

One concern is that anthropomorphisms may encourage students to view algorithms operationally, rather than assertionally. Students should be encouraged to understand concurrent algorithms in terms of invariants, metrics, progress properties, etc., rather than in terms of sequences of actions and interleavings of events. A second concern is that anthropomorphic metaphors can be so compelling that students may be tempted to make incorrect inferences based on the metaphor.

KLAs for computing concepts frequently put students in the roles of processors or data. Therefore, concerns about anthropomorphisms can easily be exacerbated through the careless use of such activities. However, kinesthetic learning does not, in itself, inherently carry these shortcomings. If properly designed and judiciously applied, KLAs can support assertional, not operational, reasoning. The powerful metaphors they promote can serve as mnemonic hooks for the most important concepts. The designer of a KLA and the instructor conducting the activity are responsible for ensuring that it is a constructive aid supporting a desired learning objective rather than a pedagogically harmful distraction. The activities described in this paper are exemplars of this approach.

## 3 Sample Kinesthetic Activities

In this section, we present activities that we have developed and piloted in our own courses on distributed systems. For each activity, we give a brief description of the algorithm or core concept being addressed, followed by a detailed description of how the activity is conducted. We then give some specific learning objectives and discuss how the activity supports achieving each objective. Finally, we point out some of the significant, but perhaps subtle, elements of the activity that contribute strongly to its success.

## 3.1 Coffee Can Problem

**The Algorithm.** A can contains some coffee beans, each of which is either white or black. An arbitrary pair of beans is removed from the can and if the beans are of the same color, a black bean is placed in the can. If the beans are of different colors, a white bean is placed in the can. This action is repeated until a single bean remains in the can. The problem is to predict the color of the single remaining bean. [Gri81]

The key to solving this problem is the observation that the action either leaves the same number of white beans in the can, or reduces this number by two. Thus, the parity of the white beans is invariant. The final bean is white if and only if there are an odd number of white beans in the can initially.

**Description of the activity.** This activity provides a nice introduction to the importance of assertional reasoning for distributed systems. It can be used as a bridge from students' previous courses which are likely to be focussed on sequential programming. As such, the activity can be prefaced with a discussion that explicitly draws on this background in deterministic sequential algorithms. In particular, students are asked how to check whether a long string of bits has an even or odd number of 1's. The obvious solution[4] is to iterate over the string and count the number of 1's. The class can then be guided to a refinement of this algorithm by being asked to improve the storage complexity: Instead of keeping the total count of 1's, a single bit suffices to encode whether or not the examined prefix has an odd number of 1's.

The notion of concurrent, distributed execution can be introduced by asking the class how to calculate the parity of 1's more quickly by working together. Students easily see a variety of parallel algorithms based on subdividing the string, calculating the parity of each substring independently, then merging the subsolutions.

The class is then divided into groups of three or four and each group is given a large can or jar and a collection of nuts and screws. It is not necessary—indeed, it is not desirable—for all groups to have the same initial allotment of hardware. Each group sets aside ten nuts and ten screws from their allotment, placing the rest in their can. They are then given instructions to repeatedly remove an arbitrary pair of objects from their can, replacing the two chosen items with a screw if they are the same, and a nut if they are not. Students are then asked to predict the final state of their can before executing the algorithm. After completing execution, each group is polled for the final state of its can, then (by a show of hands) students report whether or not their prediction for their own group's can was correct.

The exercise can be repeated by each group resetting its own can to its original state. Some students will be surprised that the outcome is the same. Their intuition may suggest that the outcome is probabilistic, and this hypothesis may even be supported by the observation that different groups have different outcomes. However, each repetition performed by any given group results in the same final state for that group. Students are eventually compelled to wonder why the outcome is deterministic.

After discussing the application of assertional reasoning techniques (see below), a follow-up activity can be used to reinforce these ideas. A third kind of item, washers, is added to the groups' hardware allotment. Again the students place ten items of each kind aside, putting the rest in

[4] When a solution is too obvious, as in this case, students may be reluctant to offer it.

their can. The algorithm again consists of repeatedly removing an arbitrary pair of items. The replacement rule, however, is modified as follows: If the pair of items is of the same kind, nothing is placed back in the can, but if the pair of items is different, an item of the remaining kind is placed back in the can. Again, students are asked to predict the outcome for their can before executing the algorithm.

**Learning Objective.** Assertional reasoning, based on invariants and metrics, simplifies reasoning about nondeterministic programs.

Seeing that the coffee can algorithm is guaranteed to terminate is relatively straight-forward: Each action decreases the number of items in the can by one. Predicting the final outcome, however, is considerably more subtle. An operational approach, exhaustively tracing the possible execution paths from a given initial configuration, is tiresome. A probabilistic approach, weighing the likelihood of each outcome, is needlessly complicated. Students easily appreciate the relative simplicity of the assertional approach, where the parity of the nuts is shown to be invariant.

There are several natural tie-in points for the coffee can problem with the introductory discussion about sequential parity-checking algorithms. The most important of these points is the key invariant for the parity-checking algorithm that uses a single bit of storage: The parity of the examined prefix together with the extra bit is always even. This observation generally provides enough guidance for students to deduce the appropriate invariant in the case of the coffee can problem. Another point that can be made about the parity-checking algorithms is that the order in which they examine bits doesn't matter. The parity of the entire string, and hence the calculated result, remains the same. This observation helps students see that determinism is not necessary for the construction of a proof of correctness. If choosing arbitrary bits to examine for a parity-checking algorithm appears to the class to be just gratuitous nondeterminism, it can be motivated by the distributed parity-checking algorithm where the parity of substrings is calculated independently. The merging of these subsolutions can be done in any order, which yields an answer faster than following a fixed, deterministic merge tree.

The follow-up activity builds on this use of parity in constructing a loop invariant, but in this case the invariant is more subtle. In this activity, there are three types of objects (nuts, screws, and washers). Every action of the program either preserves the parity of each object type (when two objects of the same type are chosen), or reverses the parity of all three object types (when two objects of differing types are chosen).

Thus, the algorithm terminates with an empty jar (all object types have the same parity) if and only if the three types of objects have the same parity initially. If the three types do not have the same initial parity, exactly one must have a different parity than the other two. In that case, the algorithm terminates with one item in the jar of that distinguished type.

**Tips for success.** An initial can configuration with 12–15 items is large enough to frustrate predicting the final state through exhaustive state exploration, but small enough to allow several repetitions. While each group can begin with slightly different distributions of hardware, it is important that every group begins by setting aside precisely ten screws and ten nuts. This allows a group to reset its can to precisely the same initial state before repeating execution. Finally, cans with sharp metal edges should be avoided for obvious reasons. Coffee cans actually work quite well, as do jars with mouths wide enough to admit a hand.

## 3.2 Nondeterministic Sorting

**The Algorithm.** For a given array of integers, the following action is repeated: A pair of elements is chosen, compared, and swapped if they are not in order [CM88]. The sequence in which pairs are chosen is not specified by the algorithm, but weak fairness requires that every pair be chosen infinitely often.

**Description of the activity.** A group of 10 students is chosen to represent the data array. Each student represents a single element in the array and is given a sign indicating their position (0–9) in the array. The sign should be easily visible, so a sheet of paper with an attached loop of string and worn around the neck works well. In addition, each student is given a small index card on which they write a random number. While the sign with the position should be easily visible to everyone, the value on the index card should be private. A student keeps the same array position (sign) for the entire activity but will swap data values (index cards).

During the activity, students mill around and arbitrarily select other students with which to compare data values. If the values on their respective index cards are out of order with respect to the students' positions in the array, the index cards are exchanged. At the beginning of the activity, students are instructed to raise their hands when they believe the array is sorted. When all hands have been raised, the students line up in order of array position and then read out the data values on their index cards to confirm that the array has been sorted.

An additional, optional, element of the activity is to use the rest of the class as "comparison processors". In this variant, students representing array elements are not allowed to see other students' index cards, even during a comparison. Instead, the pair of students go to a comparison processor and hand over their respective index cards. Only in the case of a swap do they discover the data value held by the other processor. This variant can lead into an interesting discussion of termination detection, but generally takes much longer to complete (even with many comparison processors).

**Learning Objective 1.** A deterministic, sequential algorithm is often an over specification.

Students can easily see that there are many possible execution sequences for this nondeterministic sorting algorithm. Some of these sequences correspond to well-known sequential algorithms, such as bubblesort or mergesort. In fact, every deterministic, in-place, comparison-based sorting algorithm corresponds to some execution sequence for the nondeterministic sorting program. Indeed, some students will naturally attempt a bubble-like execution, comparing their own data value with each array position in turn.

Of course, the important claim is the converse: Every execution sequence of the nondeterministic algorithm yields a sorted array. As in the coffee bean problem, students can be guided through the careful assertional proof of this claim based on the proper invariant (that the array is a permutation of the original) and metric.

**Learning Objective 2.** A good metric is not always obvious.

In the coffee bean problem, the metric to establish termination is easy to see, but the invariant is subtle. The nondeterministic sort activity provides a contrasting example, where the invariant is easy to see, but the metric is subtle. Participants have an intuitive sense of convergence towards

the correct data value, but often have a difficult time making that intuition precise. Some metrics students have proposed include: (i) the number of elements in their correct (final) position, (ii) the sum of distances that data values are from their correct (final) position, and (iii) the length of the longest sorted prefix. The first two are simply not true, and the last is too coarse-grained to be useful (i.e., lots of good work can happen without affecting the length of the longest sorted prefix).

One observation that participants can easily make is that it becomes increasingly unlikely that they will swap index cards as the algorithm proceeds. Initially, about half of the comparisons result in swaps, but that fraction gradually decreases to zero. This observation leads directly to the formation of a correct metric: the number of out-of-order pairs. This number is monotonically non-increasing, bounded below, and decreased by every swap.

**Learning Objective 3.** An action system has terminated when it reaches a fixed point.

The execution of an action system consists of an infinite sequence of (nondeterministicly chosen) actions. The program is defined to have terminated when it reaches a fixed point: In the case of this sorting algorithm, no further swaps of data values can occur. It is generally not, however, immediately obvious to any one participant that this condition holds! This difficulty of locally recognizing the fixed point is reflected in the fact that participants hesitate in raising their hands to indicate they believe the algorithm has terminated.

To test for termination, participants tend to methodically compare with each element in the array, keeping track of each value. After the activity, they can be asked to precisely characterize the condition under which they can raise their hands. This question is even more interesting under the variant where a comparison processor is used, so participants do not see the other data value when no swap occurs. Again, the process of formalizing their intuition into a more rigorous statement is a useful one. For example, students can be asked to prove or disprove the following (erroneous) claim: A participant has their final value if they have compared with all other participants and found they did not need to swap. Post-activity questions of this kind can be used to highlight the nature of rigorous assertional reasoning.

**Tips for success.** Participants should not be able to easily guide the algorithm to completion by intelligently selecting comparisons. To this end, data values should be kept small (e.g., placed on index cards as described above) and private. If participants can see data values from a distance, they will tend to perform many implicit comparisons as they mill around looking for someone with whom to swap values. This mode of making many implicit comparisons creates the false impression that relatively few comparisons are needed to complete execution of the algorithm and that almost all comparisons are effective comparisons, i.e., resulting in a swap of data values. Uniqueness of data values is not necessary. In fact, the presence of duplicate values will reduce the time needed for the algorithm to complete.

It may be tempting to increase participation by running the activity with a larger array size. However, when the array is too large, the activity takes too long to complete. Conversely, if the array is too small, the amount of interesting work to be done in sorting is small as well. In our experience, an array size around 10 (and no more than 15) works best.

## 3.3 Parallel Garbage Collection

**The Algorithm.** For a directed graph with a single distinguished vertex, called the *root*, vertices reachable from the root are termed *food*. All other vertices are termed *garbage*. The task is to distinguish food and garbage so that the latter can be collected. The challenge lies in accomplishing this task concurrently with a mutator process that is allowed to modify the edges (but not the set of vertices). The mutator process is allowed to (i) delete any edge and (ii) add any edge so long as the new edge is directed toward a vertex that is already food.

The trivial algorithm of marking all vertices is not correct because it marks vertices that are initially garbage (and therefore remain garbage). The obvious algorithm of marking the root and then propagate marks to any neighbor of a marked node is also not correct, because the mutation of the graph could undermine the diffusion of the marks. Some food may never get marked. The correct algorithm extends the mark propagating approach by also marking a node when an edge is created that is directed towards it [CM88].

**Description of the activity.** A group of 15 to 20 students is chosen to represent the vertices in the graph. Each student is given a hat[5] and a stack of approximately 12 index cards. One student is distinguished as the root. Each student then writes their own name, once on each index card in their stack. Three index cards are collected from each student, shuffled, then redistributed to the group. The redistributed cards received by a student are *edge cards* and should be held in one's hand while the cards that were never collected are *reserves* and can be kept in one's pocket. An edge card represents an edge directed from the student holding the card to the student whose name is on the card.

Graph mutation is accomplished by the students themselves. Edges are always added and deleted by the *source* vertex. A student can delete an edge, at any time, by simply discarding an edge card from their own hand. (These discards should also be brought back periodically to the students whose names appear on the cards, just to replenish students' reserves.) To add an edge, a student must first choose a destination vertex that is reachable from the root. This is done by examing the edge cards of the root and choosing one vertex, then examing the edge cards held by that vertex and again choosing one vertex, and so on, stopping after as many hops as the student wishes. The student then takes a card from the destination vertex's reserves, thus creating the edge.

In addition to the students representing the graph and carrying out the mutation, one more volunteer is chosen to be the marker. This student attempts to mark all food vertices (by placing their hat on their head) while not marking any vertices that were garbage at the beginning of execution. The naive mark-and-sweep algorithm should be simulated where the marker begins by marking the root, and then recursively marks each vertex in the root's edge cards.

When the marker decides they have completed their task, their work can be checked. One way to do this is to have all students sit down, then perform a depth-first search of the graph beginning with the root and asking each student, as their name is read, to stand up (and read the name on one of their edge cards). At the end, all food vertices are standing and it can be seen whether they have all been marked.

**Learning Objective.** Operational reasoning is dangerous.

[5]These hats can be simply constructed from paper bags or construction paper.

This activity is most effective when used to illustrate an *incorrect* algorithm. The naive algorithm consists of marking the root, then repeatedly examining some marked vertex and marking its neighbors. This algorithm, however, is not guaranteed to mark all food. The run that exposes this error in the algorithm is somewhat pathological and unlikely to arise by random chance. To illustrate the error, then, some collusion among the student volunteers is necessary.

The simplest way to frustrate the marking of all food is to arrange ahead of time with two students to be special vertices, each of which has an edge to the instructor. It is important that (i) no other vertices have edges to the instructor and (ii) both special vertices remain reachable from the root continually through the activity. The first property is ensured by only giving out two edge cards from the instructor, one to each special vertex. The second property is ensured by making both special vertices immediate neighbors of the root and instructing the root to never discard either of their edge cards.

During the activity, each special vertex keeps the instructor edge card in their hands, unless they are about to be checked by the marking process. Before being checked, they casually discard the instructor edge. After being checked, they can safely pick that card back up. Thus, the only time they are not holding the instructor edge card is when they are being checked by the marker process.

At the end of the activity, the entire class (including the marking process) should be surprised that the naive algorithm failed to mark all food. The operational argument, that marks spread throughout the root's connected component, is compelling. This activity is useful, therefore, in illustrating the dangers in anthropomorphic, informal, handwaving arguments.

**Tips for success.** The activity depends on students knowing each other's names, since names on index cards encode edges. Students should be encouraged to mill around and actively add and discard edges.

The class should not be aware of the collusion between the two students described above, so it should be arranged beforehand. During the activity, asking the root not to discard any edge cards generally does not raise suspicions. To further obfuscate the collusion, the two volunteers should not be physically near each other. The goal is for none of the participants to be aware of the special interleaving of concurrent actions being orchestrated to frustrate the marking of a particular food vertex (the instructor).

In addition to identifying all vertices that are food at the end of the activity, a correct marking algorithm is also required to *not* mark vertices that are initially garbage. A random 3-regular directed graph with 15 vertices, as described in this activity, has an astronomically small probability of containing garbage initially. Therefore, the graph must either be deliberately constructed to contain garbage initially, or a smaller degree must be used[6]. One way to deliberately construct the graph with garbage is to partition the students representing vertices into 2 groups and then distribute edge cards (i) within each group, and (ii) from the group containing the root to the other group (thus forming edges pointing *towards* the group with the root). At the end of the activity, students representing vertices that were initially garbage can be asked to stand to verify that none have been marked.

[6]A 2-regular graph with 15 vertices still only has about a 10% chance of containing garbage initially, while a 1-regular graph is virtually guaranteed to contain garbage initially.

## 3.4 Stabilizing Token Ring

**The Algorithm.** In Dijkstra's classic self-stabilizing token ring algorithm [Dij74], $k$ processes are arranged in a ring and each maintains an integer in the range 0 to $k-1$. A process has the token when its left neighbor's value is different than its own. To pass the token on, a process copies its left neighbor's value. One special process uses a different rule: It has the token when its left neighbor's value is the same as its own. To pass the token on it increments its value, modulo $k$.

The legitimate states for this algorithm are the ones in which there is exactly one token in the ring. Once in a legitimate state, the algorithm—in the absence of faults—remains in a legitimate state. The algorithm is self-stabilizing because it converges to a legitimate state from an arbitrary initial state.

**Description of the activity.** We first developed this activity as part of a computer science module for $8^{th}$ grade girls participating in a week-long science and engineering camp at The Ohio State University. A description of the entire module, including this activity, can be found in [SD03]. Since then, we have used variations on this activity for presentations to middle school, high school, undergraduate, and graduate level students, modifying the cognitive level of the associated presentation and discussion as appropriate. We've been pleased to observe a surprising degree of scale invariance: this activity (with variations) can be compelling and effective at each of these levels.

The basic idea underlying this activity is the use of sound (or music) to convey the departure from a legitimate state and the subsequent convergence. Each student is given a simple musical instrument to play when they access the critical section. When the system is in a legitimate state—a single token circulating around the ring—each note is played in turn and the orchestration is obvious. When there are multiple tokens, there is a cacophony of sound. As the number of tokens decreases, the original obvious orchestration returns.

A group of 14 students is chosen to represent the processes in the ring and each is given a one-note xylophone bar, a mallet, a sheet of paper with the numbers 0 and 1, and a coin or similar marker (for placing on the 0 or 1 to indicate state). The xylophone bars are arranged so as to play an ascending then descending major scale. With 14 notes, each tonic appears only once while the intermediate notes appear twice, creating a circular melody with no apparent beginning or end (see Fig. 1 for an example).

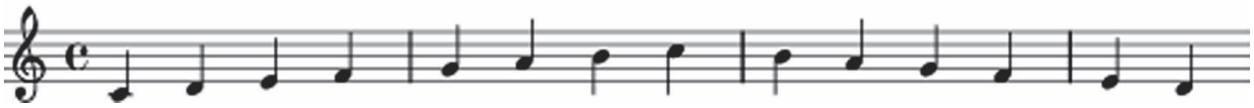

Figure 1: Circular Scale for Binary Token Ring

Each student performs the token ring actions of Dijkstra's algorithm. When their left neighbor's value differs from their own, they play their note and copy that neighbor's value. The special process performs the modified rule of playing the note when the left neighbor's value is the same as their own and then incrementing their own value.

This binary token ring should be initiated in a legitimate state and allowed to run so the students see how the token circulates in a normal execution. This 2-state algorithm, however, is not self-stabilizing. A fault can be introduced by modifying a single student's data value. The result is 3

tokens that follow each other around the ring, a clear and continued departure from the original melody.

The algorithm can then be simply modified to Dijkstra's $k$-state algorithm by changing the sheets of paper to include the values 0 to 13 inclusive. The rules followed by the students regarding when to play their note and how to change their state remain exactly the same. This version, however, is stabilizing. This can be illustrated by having each student begin in an arbitrary state and then executing the algorithm. Initially, almost every student is likely to have a token, so the notes are played chaotically. As tokens are removed, however, the chaos diminishes until the orchestrated melody finally emerges.

**Learning Objective 1.** Self-stabilization is convergence and stability.

The effect of stabilizing from an arbitrary state is dramatic. With each removal of an extra token, the initial cacophony of sound is gradually transformed into a structured melody. Thus, the auditory aspect of this activity not only clearly distinguishes between legitimate and fault states, it also reflects a metric of distance between the current state and a legitimate one. Students can literally hear the convergence occur.

As for stability, the initial execution starting in a legitimate state serves as a kind of "control group" experience that witnesses the stability of legitimate states. Of course, one example execution is not a proof of stability, but the example can serve as a motivating basis for proving the assertion formally by induction.

**Learning Objective 2.** The special process plays an important role in guaranteeing convergence.

During execution of the stabilizing algorithm, students can observe where in the ring extra tokens disappear. This leads to a discussion of how the special process acts to push the system state towards a legitimate state. In this way, the informal observation can be refined into an appreciation for the value of the following lemma: Eventually the value held by the special process is unique. A careful proof of correctness can then be constructed around this key lemma.

**Tips for success.** When injecting a fault into the binary token ring by modifying a student's state, make sure that that student's *neighbor* realizes they have a token (and pass it along) *before* the student at which the fault occurred begins to work with their own, new, spurious token. The idea is to create some spacing between these two spurious tokens to reduce the chance of accidental convergence (i.e., collision). In our experience with this activity, accidental convergence of the binary ring does not happen in practice. Human nature appears to work towards keeping the tokens separated and thus continually circulating.

The convergence of the $k$-state algorithm is most dramatic when the final result is a recognizable tune. After the token ring activity, we reorganize the xylophone bars so as to play *Twinkle Twinkle Little Star* (which also has 14 notes). To do this reorganization quickly, the tone bars are numbered with their position in this tune. The students aren't aware of the effect of the reorganization, so they do not realize what they are playing until the algorithm stabilizes.

For maximum effect, the special process should *not* be aligned with the start of the tune. Placing the special process about a quarter of the way through the tune works well.

Although using a recognizable melody increases the contrast between legitimate states and fault states, xylophone bars and musical tunes are not strictly necessary for this activity to be

effective. For example, we have also used kazoos with some success. These instruments allow greater flexibility in group size, since no particular melody is played as the token circulates.

## 3.5 Stabilizing Leader Election

One challenge of kinesthetic learning activities is the possibility that things can go amiss during the activity itself. For example, algorithmic simulations can witness communication faults whenever students misunderstand each other's speech or handwriting. Sometimes a single mistake can precipitate a global algorithmic failure. In the examples we have tested, this can usually be avoided by careful design and management of the activity itself. This next activity, however, actually leverages the possibility of faulty executions into a learning opportunity for illustrating the self-healing properties of stabilizing systems. Specifically, this activity shows how the impact of data corruptions can be temporally isolated and repaired during the execution of a stabilizing leader-election algorithm.

**The Problem.** Each process in a connected graph has a distinct numeric identifier. The goal is to elect as leader the unique process in the system with the greatest identifier. The fault model assumes that process identifiers are not corruptible, but that transient faults may corrupt the data values of all other variables finitely many times during any run. Such faults can be repaired only by overwriting the corrupted values with fresh values.

**Description of the activity.** This activity simulates two leader election algorithms drawn from [Dol00, pp. 34–36]. The first algorithm is intolerant: It may yield invalid election outcomes in the wake of certain data corruptions. The second is stabilizing: It always detects and recovers from finitely many transient corruptions to non-identifier variables. In both activities, each student represents a process in the graph. Each pair of students with adjacent seats in the classroom shares an undirected edge in the graph. This is usually sufficient to guarantee a simple connected graph, but large classes can be partitioned into two or more disjoint connected graphs for scalability.

In principle, each student needs a distinct process identifier; in practice, only the maximal identifier needs to be unique. The identifiers can be assigned by the instructor for finer control of the activity, but this is usually unnecessary. A simpler approach is for each student to use the last four digits of their telephone number.[7]

**The Intolerant Algorithm.** Each student maintains on a flash card a local variable called *candidate*, which records the maximum identifier witnessed thus far. Upon start-up, each student initializes their candidate variable to the value of their own process identifier. Thereafter, the algorithm proceeds via concurrent gossiping as follows: (1) Every student periodically shares the current value of their candidate variable with each neighbor; (2) After each exchange of information, the value of each local candidate variable gets updated to equal the max of the two values just witnessed.

[7] Duplicate maximal values are improbable using this assignment strategy, but beware of roommates!

**Learning Objective 1.** Single, local faults can precipitate global algorithmic failures.

Not all faults lead to an erroneous result: The spurious candidate value must be globally maximum. One colluding participant can surreptitiously corrupt their candidate value (but not their actual process identifier) to 9999—a value that is globally maximal and is very unlikely to denote any legitimate process identifier in the system. The corrupted value will diffuse throughout the graph until it ultimately gets elected leader. At this point, the instructor can ask for the person with the elected identifier 9999 to stand up. The absence of such a person witnesses the failure of the algorithm, and provides a basis for classroom discussion about how the intolerant algorithm can be amended.

**The Stabilizing Algorithm.** This solution recovers from transient data corruptions by recomputing floating outputs—a technique that filters out spurious candidates to prevent phantom leaders from being elected. The key idea is that each legitimate candidate must correspond to some actual, reachable process in the system. For any system with at most $k + 1$ processes, the *minimal* path from any node to a legitimate candidate can never exceed $k$ hops. Each node continually recomputes the shortest distance to the maximal candidate value seen thus far, except that candidates more than $k$ hops away are excluded as spurious.

Each student maintains two corruptible variables: *candidate* and *distance*. Each student continuously monitors their neighbor's candidate and distance variables and updates their own values to maintain the following property: Their own candidate value is equal to the maximum of all neighbors' candidates and their own identifier. In addition, their own distance value is one greater than the *minimum* distance value of a neighbor with the *maximum* candidate value (or 0 if the maximum candidate value is their own identifier). It is important that only neighbors with a distance value less than $k$ are considered during this operation.

While in the intolerant algorithm, students use their own candidate values for comparison, it is important that in the stabilizing algorithm one's own candidate and distance values are *not* used in updating this information.

**Learning Objective 2.** Transient faults can be repaired via computational redundancy.

In the algorithm, any maximal identifier that is spurious will also have a spurious distance. As such, each round of execution will cause the estimated distance to this node to increase. Eventually the estimated distance will exceed a known bound $k$ on the number of nodes in the graph, at which point the spurious identifier is also eliminated from consideration as a candidate for leader. This activity demonstrates how self-repairing algorithms can withstand data corruption using only local communication and coordination.

# 4 Designing Kinesthetic Learning Activities

Developing a new KLA requires careful planning. A poorly planned activity will not only be a waste of class time, but may even be pedagogically harmful, serving to undermine the intended lesson. The design of a KLA should begin with an explicit statement of the learning objectives it is meant to support. The activity should then be designed around these learning objectives.

In addition to this basic principle, the following heuristics are helpful in creating effective KLAs.

- Incorporate an element of surprise. In the coffee bean problem, by first doing the sequential algorithm and then the nondeterministic one, students end up surprised to realize that they are the same computation. In the stabilizing token ring, students are surprised when a recognizable musical tune emerges. With parallel garbage collection, they are surprised that the "obvious" algorithm is not correct.

- Involve multiple senses and dimensions of engagement. In the stabilizing token ring exercise, students are watching their neighbor's state, playing their musical instrument, and *hearing* the violation of the mutual exclusion invariant, as well as convergence towards stabilization. In the leader election activity, students are observing neighbor values while simultaneously modifying their own.

- Anticipate and accommodate mistakes. There will almost certainly be too many concurrent activities to be able to monitor them all. The activity should either be robust enough to tolerate the occasional mistake, or checks should be incorporated to reduce the chances of such a mistake occurring. For example, in the coffee bean problem, students work in groups so the removal and addition of beans is overseen by peers within the group.

- Engage the entire class. If the activity can not be scaled to include every student directly, it should be designed to encourage non-participants to identify with participants, and thus be involved at least vicariously.

- Provide simple directions to participants. If the instructions are complicated, there is a greater chance of mistakes being made. Also, if students are too engrossed in their local computation, the bigger picture can be hard for them to see.

Even if an activity has been carefully planned, a practice run is an invaluable aid in assessing how it will work live, in a classroom setting. A small, friendly group can provide feedback and insight into an activity's dynamics and can help refine the details of the design to improve its chances of success in the classroom.

# 5 Acknowledgements

Many of the activities in the collection presented in this paper benefited from suggestions by colleagues and graduate students. The musical aspect of the stabilizing token ring algorithm was developed together with Murat Demirbas and Hilary Pike, with logistical support by Amy Giles for procuring instruments. All of these activities have been refined through input from Nigamanth Sridhar, Christopher Bohn, Brad Moore, Sandip Bapat, Matt Lang, and Greg Buehrer. Finally, thanks to the editor, Sergio Rajsbaum for both his encouragement and his patience.

# 6 Conclusions

KLAs promote student interactivity and improve student learning by engaging a fundamental and ubiquitous learning style. For this reason, courses across all disciplines can benefit from the inclusion of such activities. For courses on distributed computing in particular, however, KLAs are ideally suited since they naturally incorporate concurrency and locality of computation. Furthermore, self-stabilization can improve the robustness of KLAs to participant error.

We have provided here a collection of KLAs developed for use in our courses on distributed computing. Our hope is that others will adopt these activities for their own courses, as well as use this collection as a template for developing new activities that they later share with the community, too.

# References


[ANH78] David P. Ausubel, Joseph D. Novak, and Helen Hanesian. *Educational Psychology: A Cognitive View*. Holt, Rinehart and Winston, 2nd edition, 1978.

[BEH+56] B. S. Bloom, M. D. Engelhart, H. H. Hill, E. J. Furst, and D. R. Krathwhol, editors. *Taxonomy of Educational Objectives. The Classification of Educational Goals, Handbook I: Cognitive Domain*. David McKay Company, Inc, New York, 1956.

[BGW04] Andrew Begel, Daniel D. Garcia, and Steven A. Wolfman. Kinesthetic learning in the classroom. In *SIGCSE '04: Proceedings of the 35th SIGCSE technical symposium on Computer science education*, pages 183–184, New York, NY, USA, 2004. ACM Press.

[BJY99] Robert Borchert, Daniel Jensen, and David Yates. Development and assessment of hands-on and visualization modules for enhancement of learning in mechanics. In *ASEE Annual Conference & Exposition: Engineering: Education to Serve the World*, pages 20–23, Charlotte, NC, June 1999.

[BLWH00] Paolo Bucci, Timothy J. Long, Bruce W. Weide, and Joe Hollingsworth. Toys are us: Presenting mathematical concepts in CS1/CS2. In *Proceedings of the* $30^{th}$ *ASEE/IEEE Frontiers in Education Conference*. IEEE Computer Society Press, 2000.

[Bon96] Charles C. Bonwell. Enhancing the lecture: Revitalizing the traditional format. *New Directions for Teaching and Learning*, 67:31–44, Fall 1996.

[BWF07] Tim Bell, Ian H. Witten, and Mike Fellows. Computer science unplugged. URL www.unplugged.canterbury.ac.nz, 2007.

[Cas85] William E. Cashin. Improving lectures. IDEA paper No. 14, Kansas State University, Center for Faculty Evaluation and Development, 1623 Anderson Avenue, Manhattan, KS 66502-4098, September 1985.

[CM88] K. Mani Chandy and Jayadev Misra. *Parallel Program Design: A Foundation*. Addison-Wesley Publishing Company, Reading, Massachusetts, 1988.

[CS95] M. E. Crosby and J. Stelovsky. From multimedia instruction to multimedia evaluation. *Journal of Educational Multimedia and Hypermedia*, 4(2/3):147–162, 1995.

[Dai94] B. Daily. Multimedia and its impact on training engineers. *International Journal of Human Computer Interaction*, 6(2):191–204, 1994.

[Dij74] E. W. Dijkstra. Self-stabilizing systems in spite of distributed control. *Communications of the ACM*, 17(11):643–644, November 1974.

[Dij85] Edsger W. Dijkstra. On anthropomorphism in science. EWD936, available at http://www.cs.utexas.edu/users/EWD/ewd09xx/EWD936.PDF, September 1985.

[Dol00] Shlomi Dolev. *Self-Stabilization*. MIT Press, 2000.

[DS90] Edsger W. Dijkstra and Carel S. Scholten. *Predicate Calculus and Program Semantics*. Texts and Monographs in Computer Science. Springer-Verlag, 175 Fifth Avenue, New York, New York 10010, 1990.

[FB05] Richard M. Felder and Rebecca Brent. Understanding student differences. *Journal of Engineering Education*, 94(1):57–72, 2005.

[Fel93] Richard M. Felder. Reaching the second tier—learning and teaching styles in college science education. *Journal of College Science Teaching*, 22(5):286–290, March–April 1993.

[FKM03] Heather Fry, Steve Ketteridge, and Stephanie Marshall, editors. *A Handbook for Teaching & Learning in Higher Education*. Routledge (UK), 2003.

[Fle02] Neil D. Fleming. VARK—A guide to learning styles. www.vark-learn.com, 2002.

[FS88] Richard M. Felder and Linda K. Silverman. Learning and teaching styles in engineering education. *Journal of Engineering Education*, 78(7):674–681, 1988.

[Gri81] David Gries. *The Science of Programming*. Texts and Monographs in Computer Science. Springer-Verlag, 175 Fifth Avenue, New York, New York 10010, 1981.

[Gri94] Susan Griss. Creative movement: A language for learning. *Educational Leadership*, 51(5):78–80, February 1994.

[Hak98] Richard R. Hake. Interactive-engagement versus traditional methods: A six-thousand-student survey of mechanics test data for introductory physics courses. *American Journal of Physics*, 66(1):64–74, January 1998.

[HD78] James Hartley and Ivor K. Davies. Note-taking: A critical review. *Programmed Learning and Educational Technology*, 15(3):207–224, August 1978.

[Her89] Ned Herrmann. *The Creative Brain*. Ned Herrmann Group, revised edition, 1989.

[Jun92] C. G. Jung. *Psychological Types*. Routledge, 1992. Original work published in 1921.

[Kol83] David A. Kolb. *Experiential Learning: Experience as the Source of Learning and Development*. Prentice-Hall, Englewood Cliffs, New Jersey, 1983.

[LD06] Heidi L. Lujan and Stephen E. DiCarlo. First-year medical students prefer multiple learning styles. *Advances in Physiology Education*, 30(1):13–16, March 2006.

[LL95] M. Lumsdaine and E. Lumsdaine. Thinking preferences of engineering students: Implications for curriculum restructuring. *Journal of Engineering Education*, 84(2):193–204, April 1995.

[McC90] M. H. McCaulley. The MBTI and individual pathways in engineering design. *Engineering Education*, 80(5):537–542, 1990.

[MMQH98] Isabel Briggs Myers, Mary H. McCaulley, Naomi L. Quenk, and Allen L. Hammer. *MBTI Manual: A Guide to the Development and Use of the Myers-Briggs Type Indicator*. Consulting Psychologists Press, 3rd edition, 1998.

[Ros87] Colin P. Rose. *Accelerated Learning*. Dell, December 1987.

[Ros99] P. Rosati. Specific differences and similarities in the learning preferences of engineering students. In *Frontiers in Education Conference*, volume 2, pages 12c1–17, 1999.

[RRS+95] M. Recker, A. Ram, T. Shikano, G. Li, and J. T. Stasko. Cognitive media types for multimedia information access. *Journal of Educational Multimedia and Hypermedia*, 4(2/3):183–210, 1995.

[SD03] Paolo A. G. Sivilotti and Murat Demirbas. Introducing middle school girls to fault tolerant computing. In *Proceedings of the 34th SIGCSE Technical Symposium on Computer Science Education*, pages 327–331, Reno, Nevada, February 2003. ACM Press.

[SP07] Paolo A. G. Sivilotti and Scott M. Pike. The suitability of kinesthetic learning activities for teaching distributed algorithms. In *Proceedings of the 38th SIGCSE Technical Symposium on Computer Science Education*, pages 362–366, Covington, Kentucky, March 2007. ACM Press.

[Sti87] J. E. Stice. Using Kolb's learning cycle to improve student learning. *Engineering Education*, 77(5):291–296, February 1987.

[Zim03] Virginia Zimmerman. Moving poems: Kinesthetic learning in the literature classroom. *Pedagogy*, 2(3):409–412, 2003.

[Zyw03] Malgorzata S. Zywno. A contribution to validation of score meaning for Felder-Soloman's index of learning styles. *American Society for Engineering Education*, 2003.