Banker's Algorithm

Banker's Algorithm: Data

```
class ResourceManager {
  int n;          // # threads 0 to n
  int m;          // # resources 0 to m
  avail[m],       // # of available resources of each type
  max[n,m],       // # of resources that each thread may need
  alloc[n,m],     // # of each resource that each thread is using
  need[n,m],      // # of resources that each thread might still
                  // request (i.e., max - alloc)
```

Banker's Algorithm: Allocation

```
1  public void allocate(int request[m], int i) {
2    // thread i wants request[m] new resources
3    if (request > need[i])  // vector comparison
4      error();
5    else while (request[i] > avail)
6      wait();
7
9    avail = avail - request;  // vector operations
10   alloc[i] = alloc[i] + request;
11   need[i] = need[i] - request;
12
13    while (!safeState()) {
15      <undo changes to avail, alloc[i], and need[i]>
16      wait();
17      <redo changes to avail, alloc[i], and need[i]>
18    }
19 }
```


Banker's Algorithm: Safety Check

```
private boolean safeState() {
  boolean work[m] = avail[m];  // accommodate all resources
  boolean finish[n] = false;   // none finished yet

  // find a process that can complete its work now
  while (find i such that !finish[i]
          and need[i] <= work) { // vector operations
    work = work + alloc[i];
    finish[i] = true;
  }

  return (finish[i] for all i);
}
```

Banker's Algorithm: Exercise

Total Resources: (9, 5, 7)

	Maximum	Allocation
	A B C	A B C
P_0	7 5 3	0 1 0
P_1	5 2 2	2 0 0
P_2	9 0 2	3 0 2
P_3	2 2 2	2 1 1
P_4	4 3 3	0 0 2

Granted or blocked? Assume each from this start configuration.

a. P_4 requests (3, 1, 0)

b. P_3 requests (0, 1, 2)

c. P_1 requests (1, 0, 1)

d. P_0 requests (2, 2, 0)

Recap: Deadlocks

- Necessary conditions
  - Mutual Exclusion
  - Hold and Wait
  - No Resource Preemption
  - Circular Wait
- Deadlock prevention
- Deadlock detection
  - Resource Allocation Graphs
- Deadlock avoidance
  - Banker's Algorithm