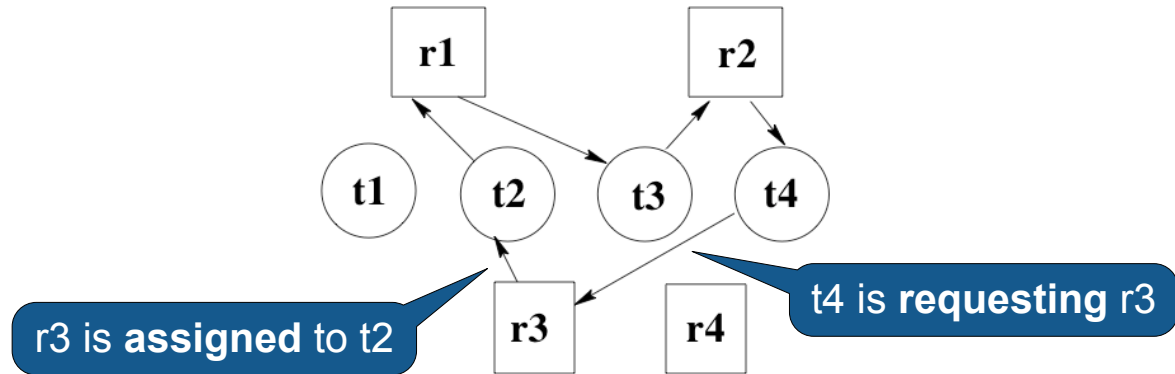


Resource Allocation Graph



Allocation Example (1)

3 threads, 12 interchangeable resources (11 used)

	maximum	current	could request
t ₁	4	3	1
t ₂	8	4	4
t ₃	12	4	8

Safe state?

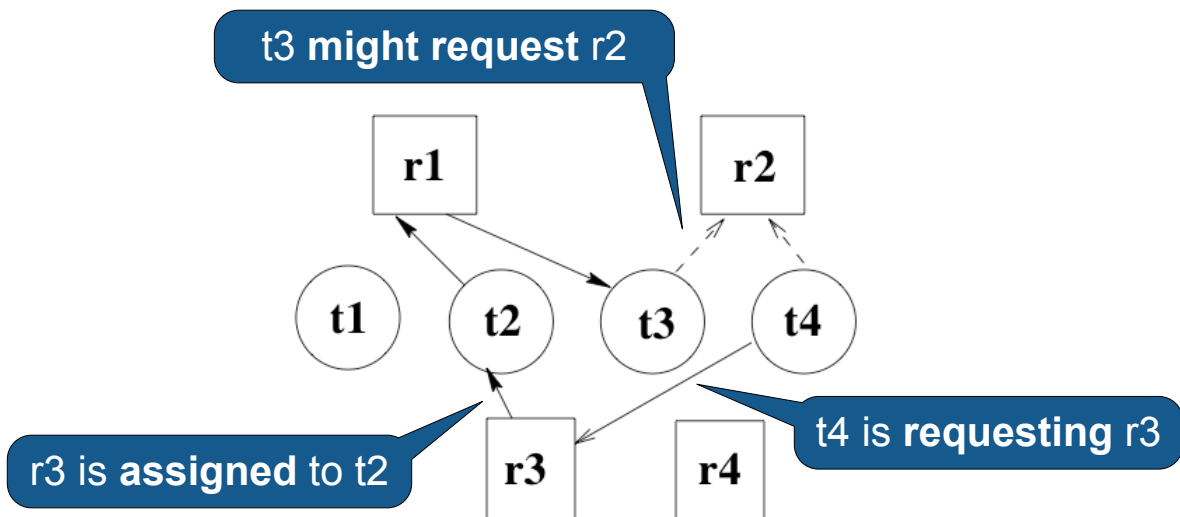
Allocation Example (2)

t_3 requests last resource (now all 12 used)

	maximum	current	could request
t_1	4	3	1
t_2	8	4	4
t_3	12	5	7

Safe state?

Deadlock Avoidance with RAG



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

```
public void synchronized allocate(int request[m], int i) {
    // thread i wants request[m] new resources
    if (request > need[i]) // vector comparison
        error(); // Can't request more than you declared
    else while (request[i] > avail)
        wait(); // Insufficient resources available

    // enough resources exist, see if would lead to unsafe state
    avail = avail - request; // vector operations
    alloc[i] = alloc[i] + request;
    need[i] = need[i] - request;

    while (!safeState()) {
        // if this is an unsafe state, undo the allocation and wait
        <undo the changes to avail, alloc[i], and need[i]>
        wait();
        <redo the changes to avail, alloc[i], and need[i]>
    }
}
```

Banker's Algorithm: Safety Checking

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

    // find someone 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: Example

	Max	Allocation	Available
	A B C	A B C	A B C
P₀	0 0 1	0 0 1	–
P₁	1 7 5	1 0 0	–
P₂	2 3 5	1 3 5	–
P₃	0 6 5	0 6 3	–
Total		2 9 9	1 5 2

Banker's Algorithm: Example 2

	Max	Allocation	Available
	A B C	A B C	A B C
P₀	0 0 1	0 0 1	–
P₁	1 7 5	1 5 2	–
P₂	2 3 5	1 3 5	–
P₃	0 6 5	0 6 3	–
Total		2 14 11	1 0 0

Recap: Deadlocks

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