CS3600 — Systems and Networks

NORTHEASTERN UNIVERSITY

Lecture 8: Deadlocks

Prof. David Choffnes (choffnes@ccs.neu.edu)

[Prepared by Prof. Alan Mislove (amislove@ccs.neu.edu)]

Deadlock

- A set of blocked processes each holding a resource and waiting to acquire a resource held by another process in the set

The Deadlock Problem

- A set of blocked processes each holding a resource and waiting to acquire a resource held by another process in the set

- Example
  - System has 2 disk drives
  - P_1 and P_2 each hold one disk drive and each needs another one

- Example
  - semaphores *A* and *B*, initialized to 1

P_0	P_1
wait (A);	wait(B);
wait (B);	wait(A);

The Deadlock Problem

- A set of blocked processes each holding a resource and waiting to acquire a resource held by another process in the set

- Example
  - System has 2 disk drives
  - P_1 and P_2 each hold one disk drive and each needs another one

- Example
  - semaphores *A* and *B*, initialized to 1

P_0	P_1
wait (A);	wait(B);
wait (B);	wait(A);

The Deadlock Problem

- A set of blocked processes each holding a resource and waiting to acquire a resource held by another process in the set

- Example
  - System has 2 disk drives
  - P_1 and P_2 each hold one disk drive and each needs another one

- Example
  - semaphores *A* and *B*, initialized to 1

P_0	P_1
wait (A);	wait(B);
wait (B);	wait(A);

The Deadlock Problem

- A set of blocked processes each holding a resource and waiting to acquire a resource held by another process in the set

- Example
  - System has 2 disk drives
  - P_1 and P_2 each hold one disk drive and each needs another one

- Example
  - semaphores *A* and *B*, initialized to 1

P_0	P_1
wait (A);	wait(B);
wait (B);	wait(A);

System Model

- Resource types $R_1, R_2, \ldots, R_m$

  CPU cores, memory space, I/O devices

- Each resource type R_i has W_i instances.

- Each process utilizes a resource as follows:
  - **request**
  - **use**
  - **release**

Deadlock Characterization

Deadlock can arise if four conditions hold simultaneously.

Deadlock Characterization

Deadlock can arise if four conditions hold simultaneously.

- **Mutual exclusion:** only one process at a time can use a resource

Deadlock Characterization

Deadlock can arise if four conditions hold simultaneously.

- **Mutual exclusion:** only one process at a time can use a resource
- **Hold and wait:** a process holding at least one resource is waiting to acquire additional resources held by other processes

Deadlock Characterization

Deadlock can arise if four conditions hold simultaneously.

- **Mutual exclusion:** only one process at a time can use a resource
- **Hold and wait:** a process holding at least one resource is waiting to acquire additional resources held by other processes
- **No preemption:** a resource can be released only voluntarily by the process holding it, after that process has completed its task

Deadlock Characterization

Deadlock can arise if four conditions hold simultaneously.

- **Mutual exclusion:** only one process at a time can use a resource
- **Hold and wait:** a process holding at least one resource is waiting to acquire additional resources held by other processes
- **No preemption:** a resource can be released only voluntarily by the process holding it, after that process has completed its task
- **Circular wait:** there exists a set $\{P_0, P_1, \ldots, P_n\}$ of waiting processes such that P_0 is waiting for a resource that is held by P_1, P_1 is waiting for a resource that is held by P_2, …, P_{n-1} is waiting for a resource that is held by P_n, and P_n is waiting for a resource that is held by P_0.

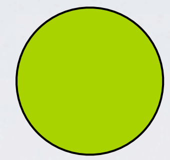
Resource-Allocation Graph

A set of vertices V and a set of edges E.

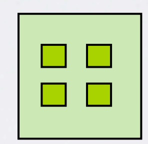
- V is partitioned into two types:
  - $P = \{P_1, P_2, \ldots, P_n\}$, the set consisting of all the processes in the system
  - $R = \{R_1, R_2, \ldots, R_m\}$, the set consisting of all resource types in the system
- **request edge** – directed edge $P_i \rightarrow R_j$
- **assignment edge** – directed edge $R_j \rightarrow P_i$

Resource-Allocation Graph (Cont.)

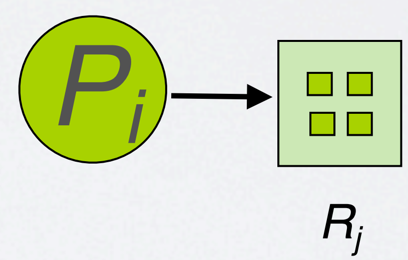
- Process



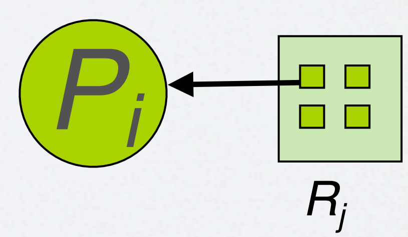
- Resource Type with 4 instances



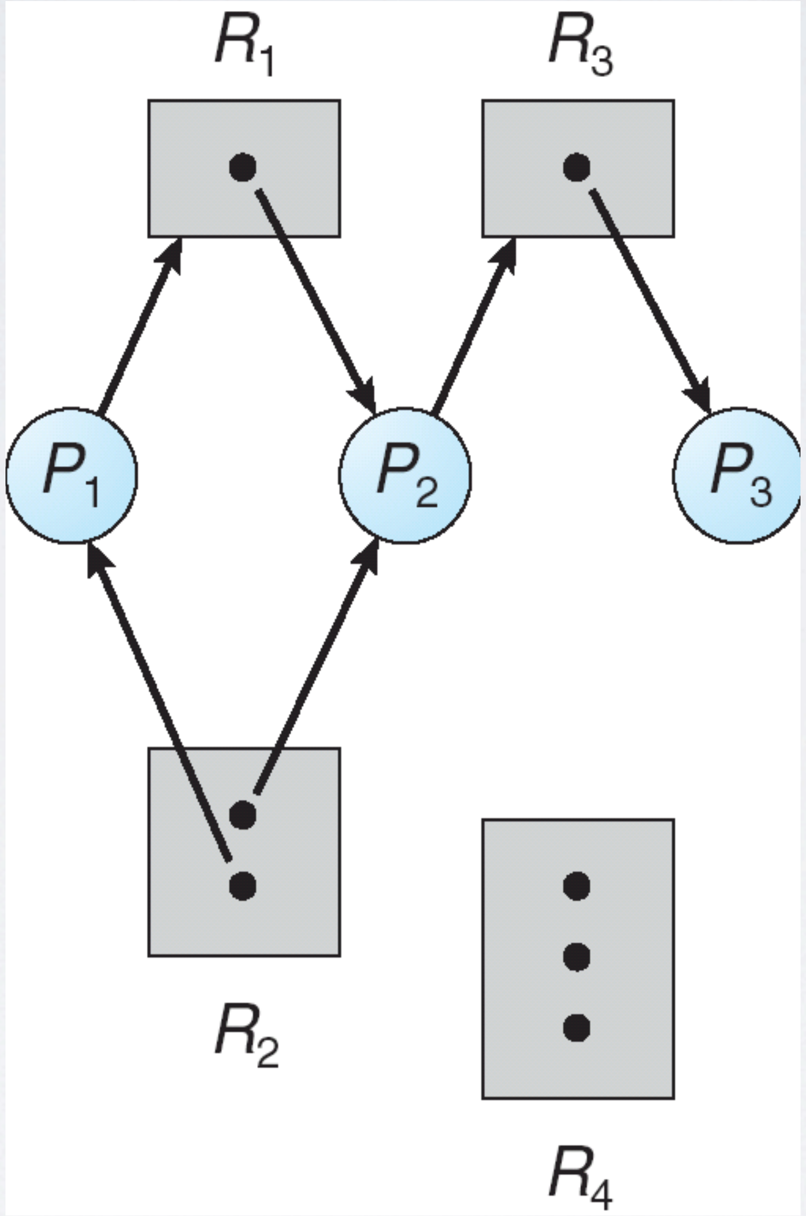
- P_i requests instance of R_j

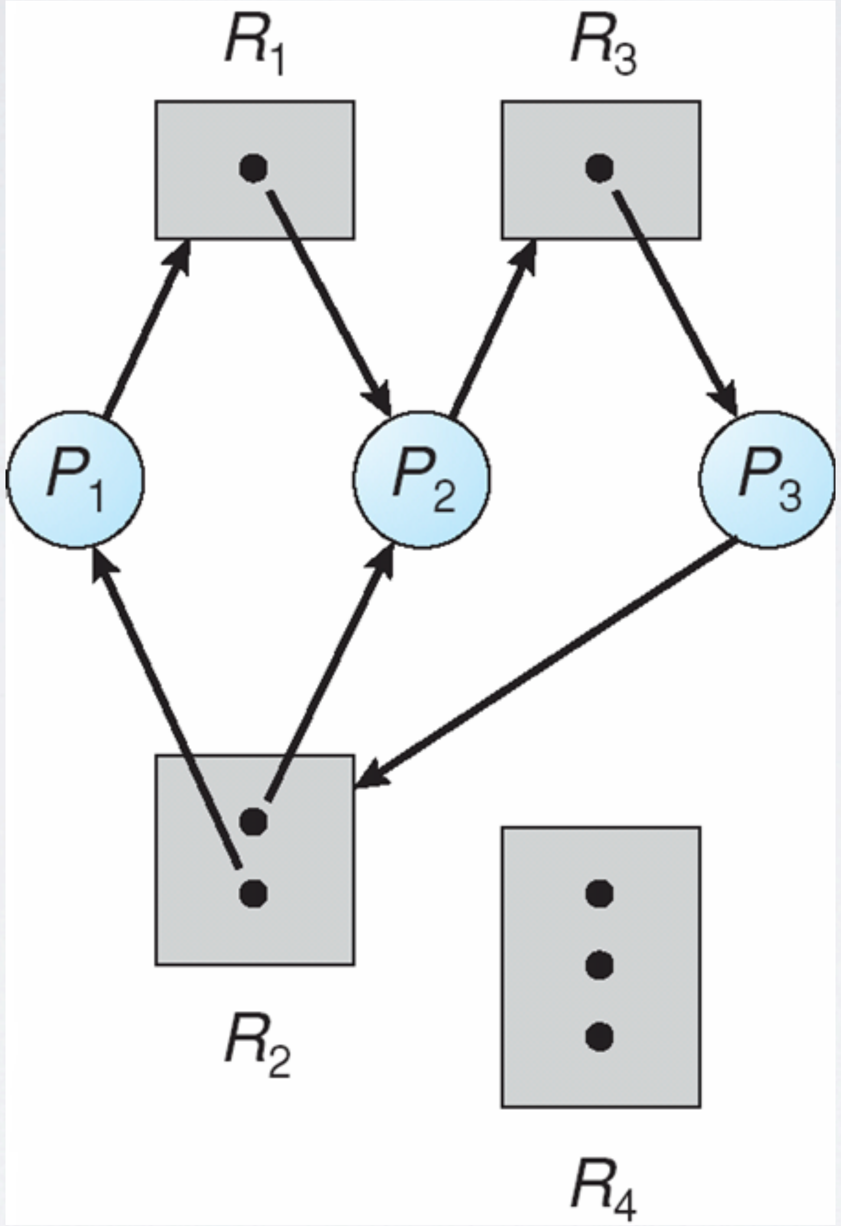


- P_i is holding an instance of R_j

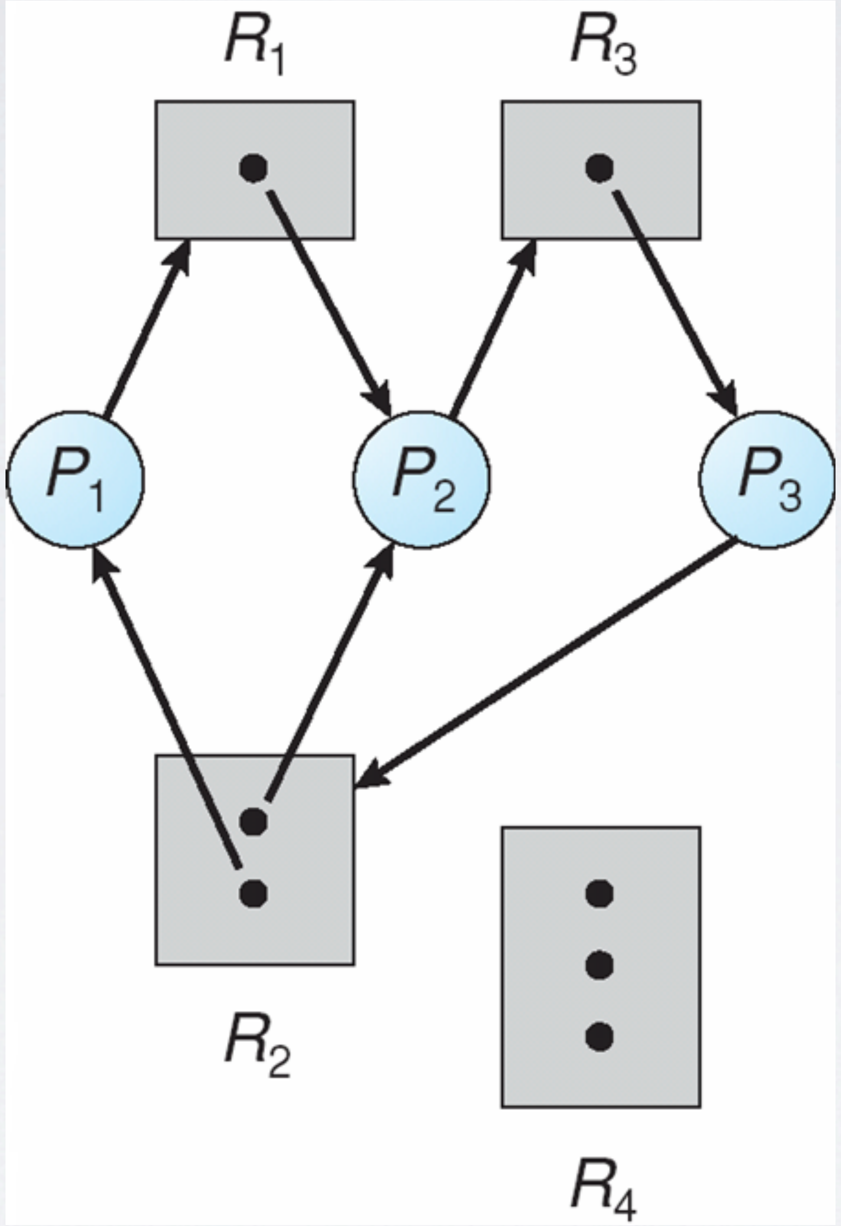


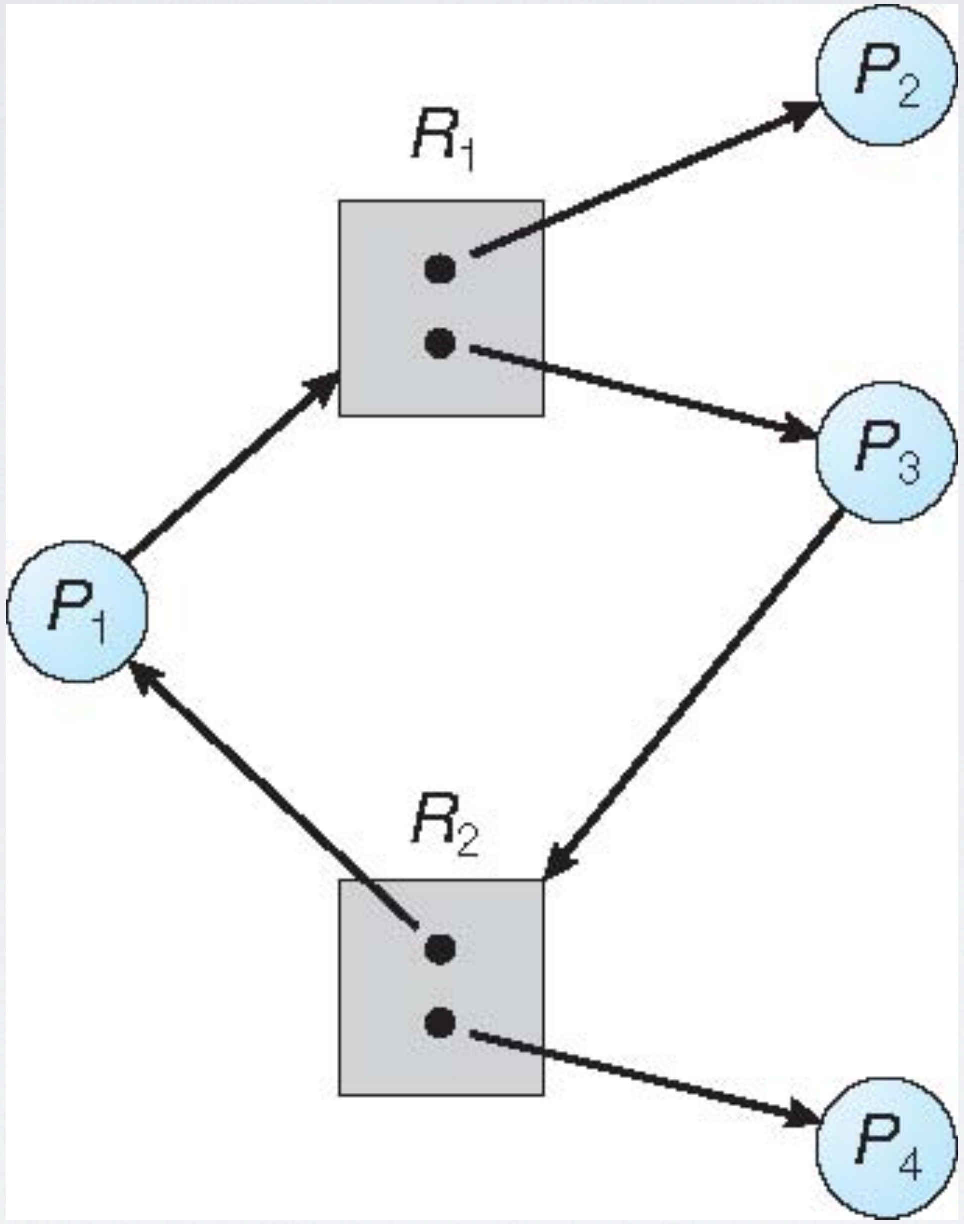
Example of a Resource Allocation Graph




R1
R3
P1
P2
P3
R2
R4

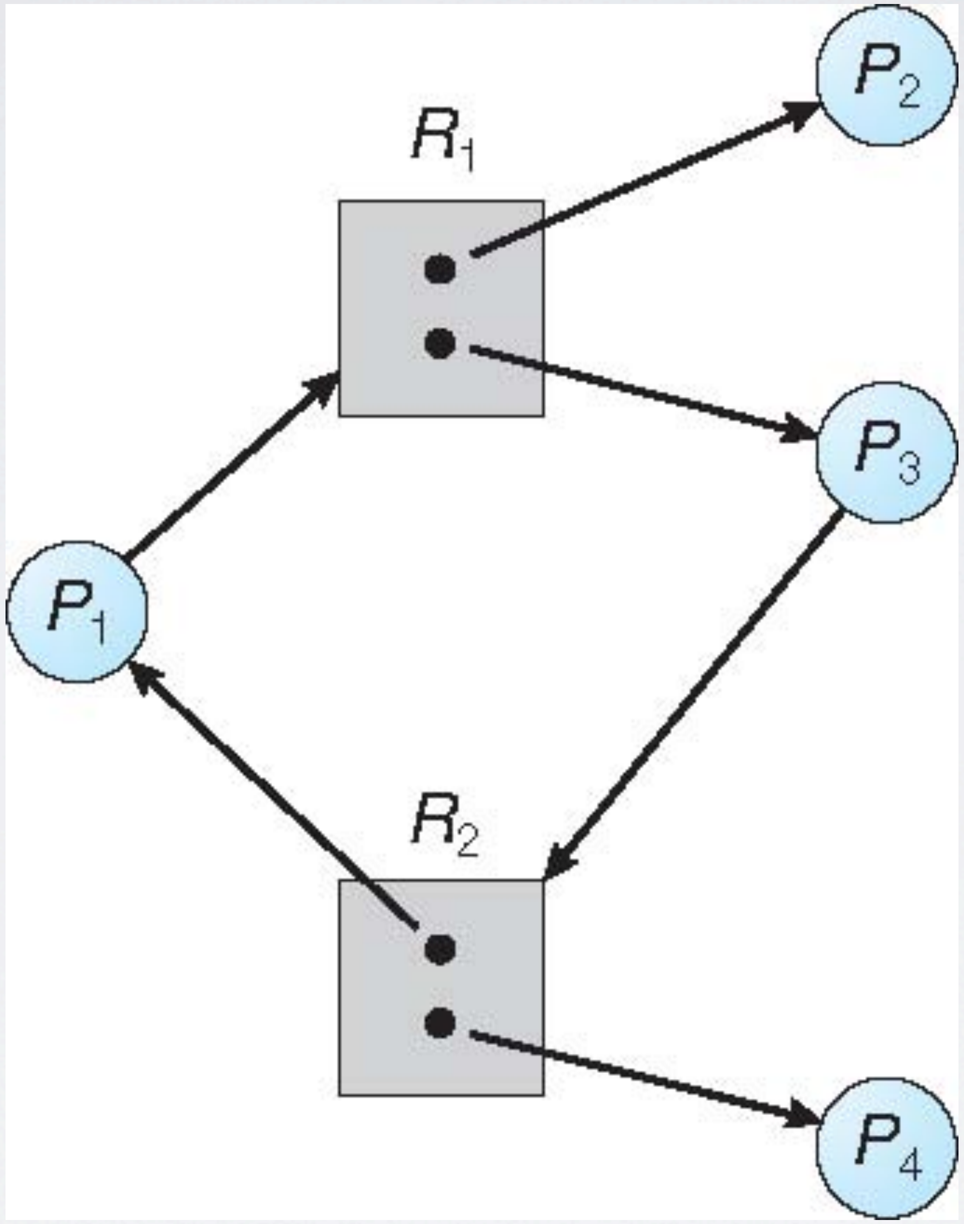
Resource Allocation Graph With A Deadlock

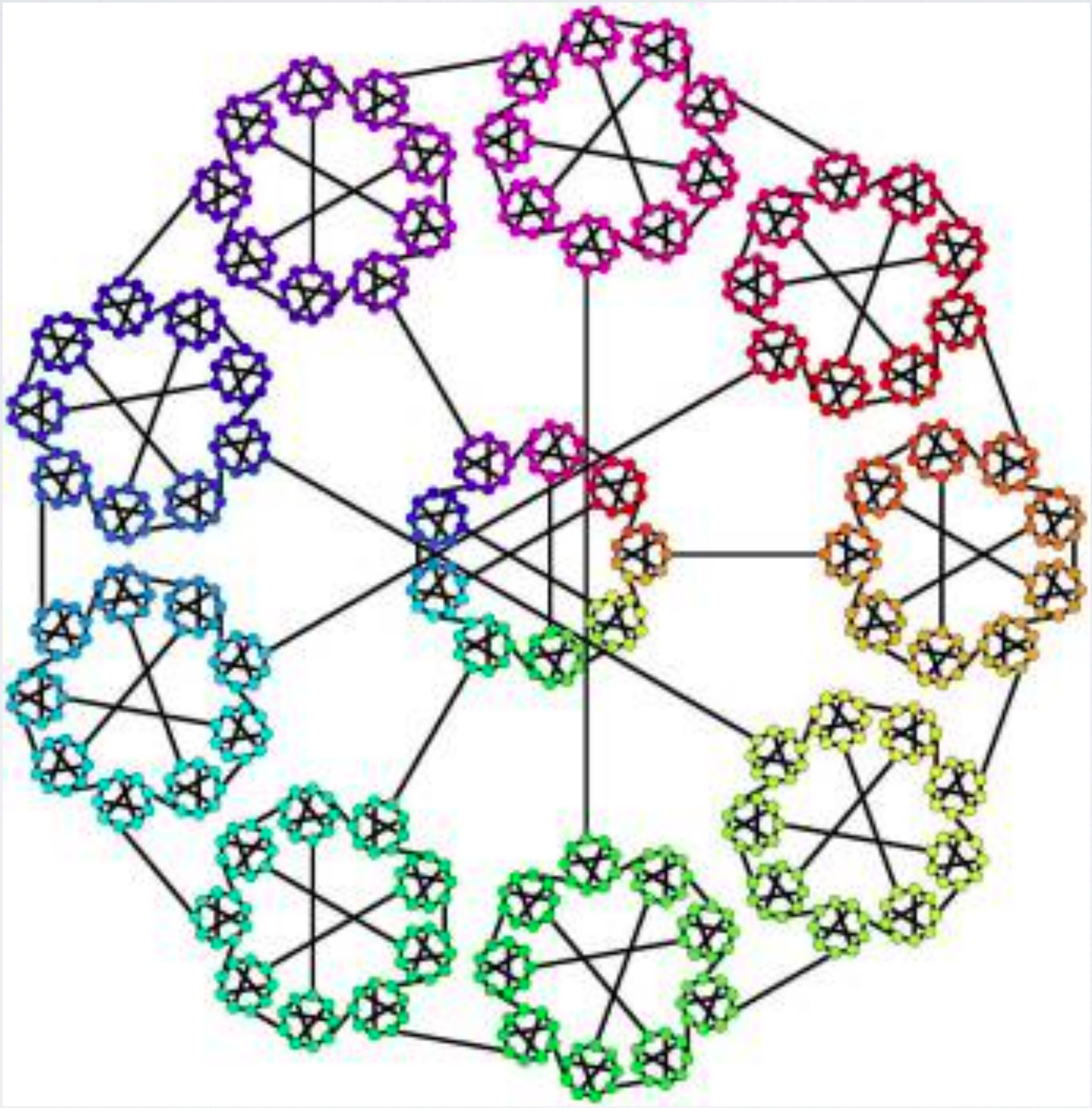




R1
R2
P1
P2
P3
P4

Graph With A Cycle But No Deadlock





Basic Facts

- If graph contains no cycles ⇒ no deadlock
- If graph contains a cycle ⇒
  - if only one instance per resource type, then deadlock
  - if several instances per resource type, possibility of deadlock

Methods for Handling Deadlocks

Methods for Handling Deadlocks

- **Avoidance** - Ensure that the system will ***never*** enter a deadlock state

Methods for Handling Deadlocks

- **Avoidance** - Ensure that the system will ***never*** enter a deadlock state
- **Recovery** - Allow the system to enter a deadlock state and then recover

Methods for Handling Deadlocks

- **Avoidance** - Ensure that the system will ***never*** enter a deadlock state

- **Recovery** - Allow the system to enter a deadlock state and then recover

- ***Ignorance*** - Ignore the problem and pretend that deadlocks never occur in the system; used by most operating systems, including UNIX

Deadlock Avoidance

Requires that the system has some additional *a priori* information available

- Simplest and most useful model requires that each process declare the *maximum number* of resources of each type that it may need
- The deadlock-avoidance algorithm dynamically examines the resource-allocation state to ensure that there can never be a circular-wait condition
- Resource-allocation *state* is defined by the number of available and allocated resources, and the maximum demands of the processes

Safe State

- When a process requests an available resource, system must decide if immediate allocation results in a safe state

- System is in **safe state** if there exists a sequence <P_1, P_2, …, P_n> of ALL the processes in the systems such that for each P_i, the resources that P_i can still request can be satisfied by currently available resources + resources held by all the P_j, with $j < I$

Safe State

- When a process requests an available resource, system must decide if immediate allocation results in a safe state

- System is in **safe state** if there exists a sequence <P_1, P_2, …, P_n> of ALL the processes in the systems such that for each P_i, the resources that P_i can still request can be satisfied by currently available resources + resources held by all the P_j, with $j < I$

- That is:
  - If P_i resource needs are not immediately available, then P_i can wait until all P_j have finished
  - When P_j is finished, P_i can obtain needed resources, execute, return allocated resources, and terminate
  - When P_i terminates, P_{i+1} can obtain its needed resources, and so on

Basic Facts

- If a system is in safe state $\Rightarrow$ no deadlocks
- If a system is in unsafe state $\Rightarrow$ possibility of deadlock
- Avoidance $\Rightarrow$ ensure that a system will never enter an unsafe state.

Avoidance algorithms

- Single instance of a resource type
  - Use a resource-allocation graph
- Multiple instances of a resource type
  - Use the banker's algorithm
  - In book, not discussed in class

Resource-Allocation Graph Scheme

- **Claim edge** $P_i \rightarrow R_j$ indicates that process P_j may request resource R_j; represented by a dashed line

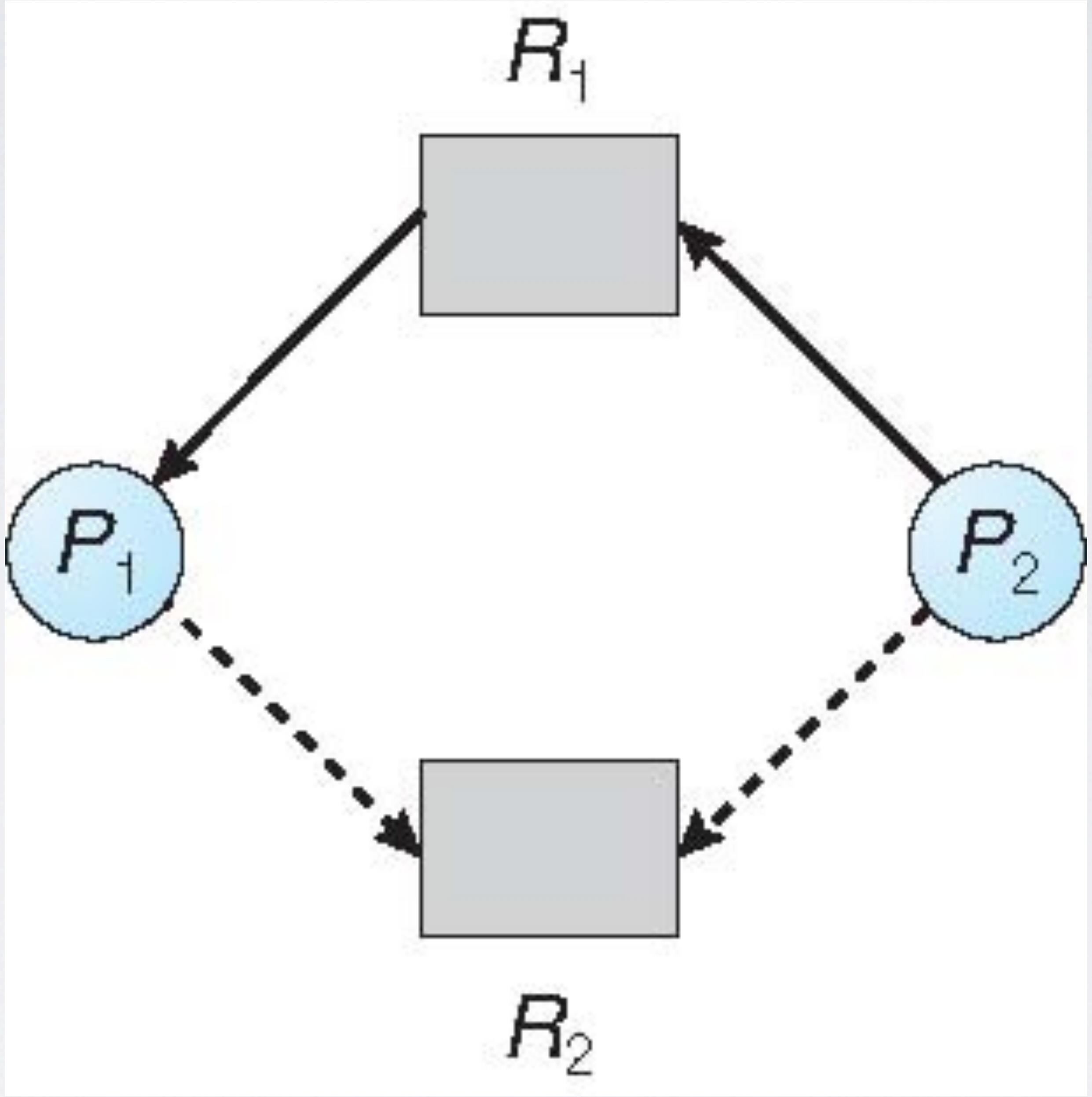
- Claim edge converts to request edge when a process requests a resource

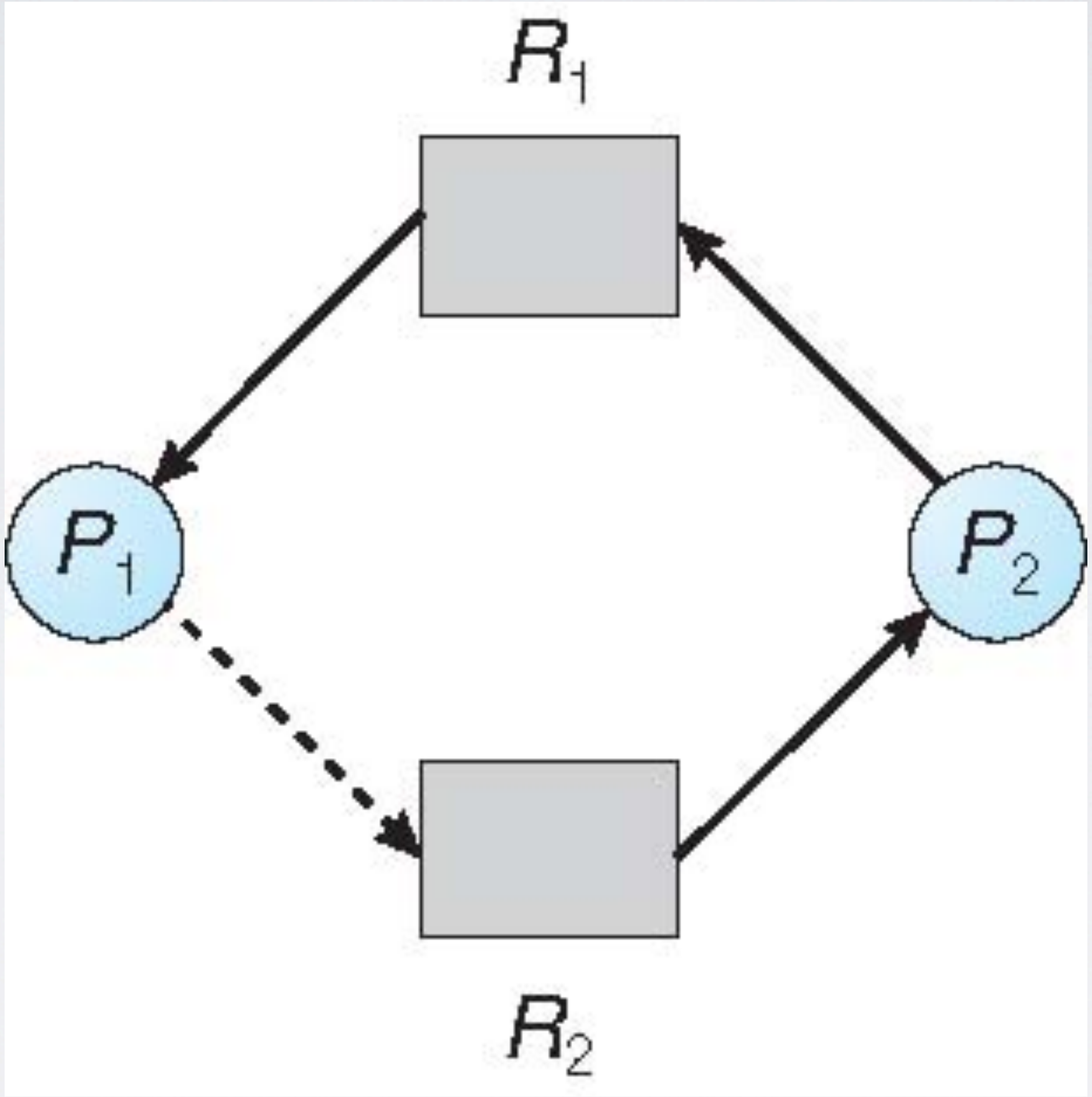
- Request edge converted to an assignment edge when the resource is allocated to the process

- When a resource is released by a process, assignment edge reconverts to a claim edge

- Resources must be claimed *a priori* in the system

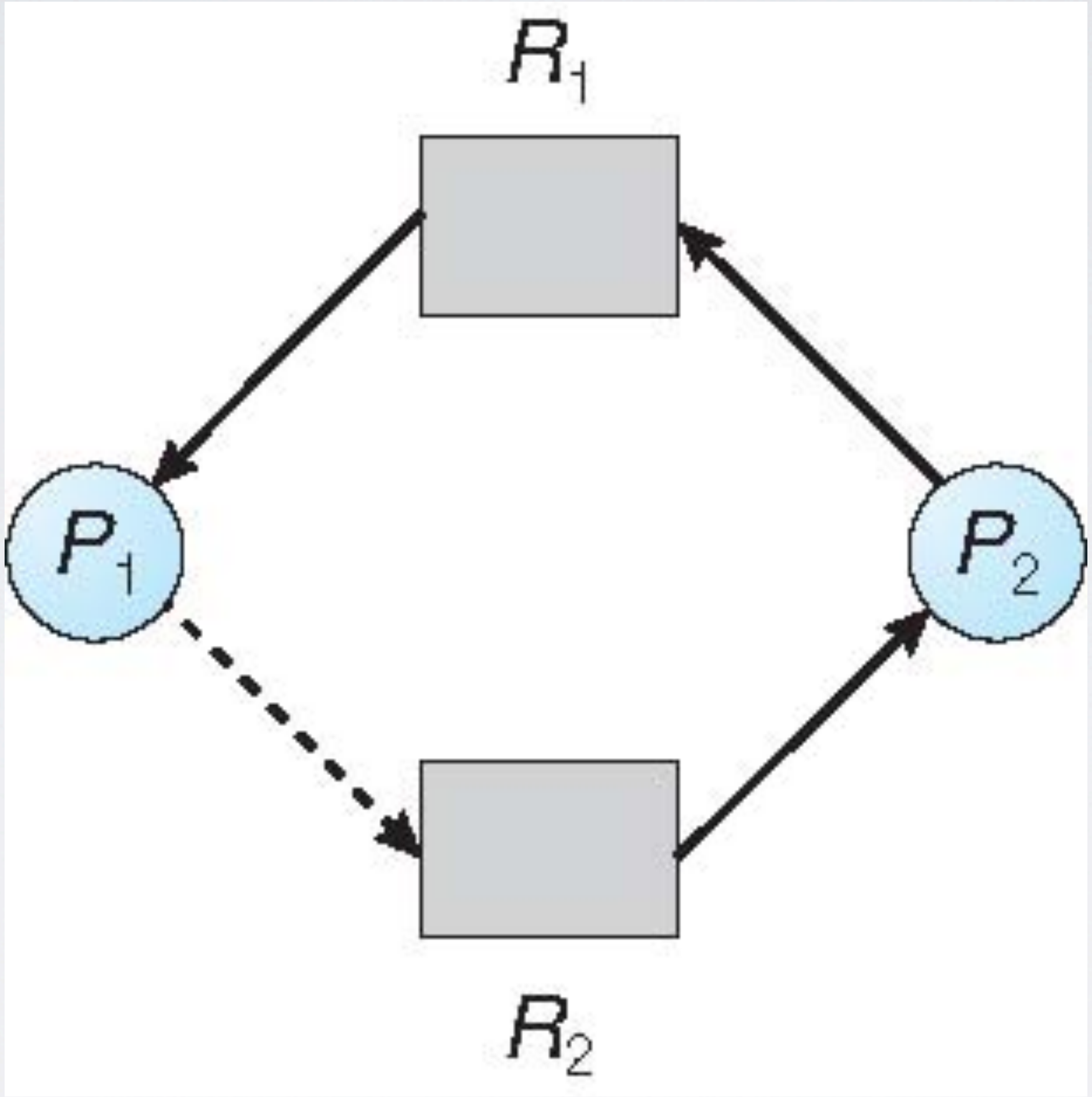
Resource-Allocation Graph





R_1
P_1
P_2
R_2

Unsafe State In Resource-Allocation Graph



Resource-Allocation Graph Algorithm

- Suppose that process P_i requests a resource R_j
- The request can be granted only if converting the request edge to an assignment edge does not result in the formation of a cycle in the resource allocation graph

But enough about deadlock

Research Opportunities for Undergrads

- Why do research in CCIS at NEU?
  - Work on interesting problems
  - You're considering grad school
  - You're curious in general
  - Because you're already done writing your FAT file system

Research Opportunities for Undergrads

- Why do research in CCIS at NEU?
  - Work on interesting problems
  - You're considering grad school
  - You're curious in general
  - Because you're already done writing your FAT file system

- Why do research with me at NEU?
  - Build things
  - Make a difference
  - You want to play with cell phone networks and apps

Quick survey

Quick survey

- Today, have you used your phone to check
  - Facebook?
  - Twitter?
  - E-mail?

Quick survey

- Today, have you used your phone to check
  - Facebook?
  - Twitter?
  - E-mail?

- How many have made a voice call?

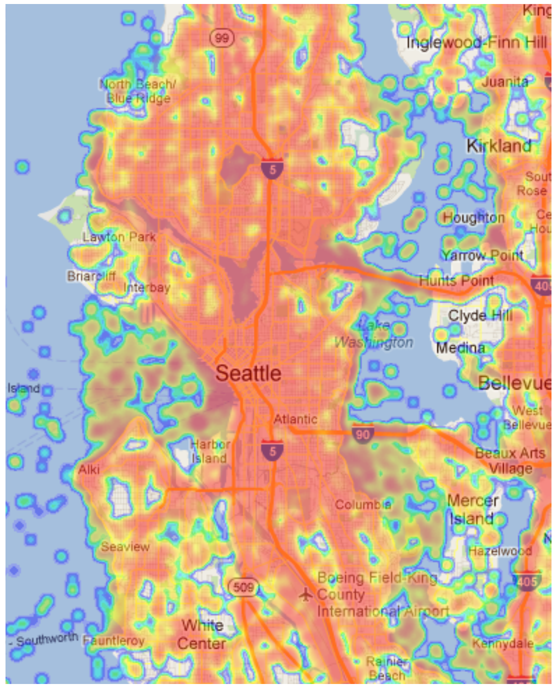
Can you ping me now?

- Phones are increasingly used for data, but designed for voice

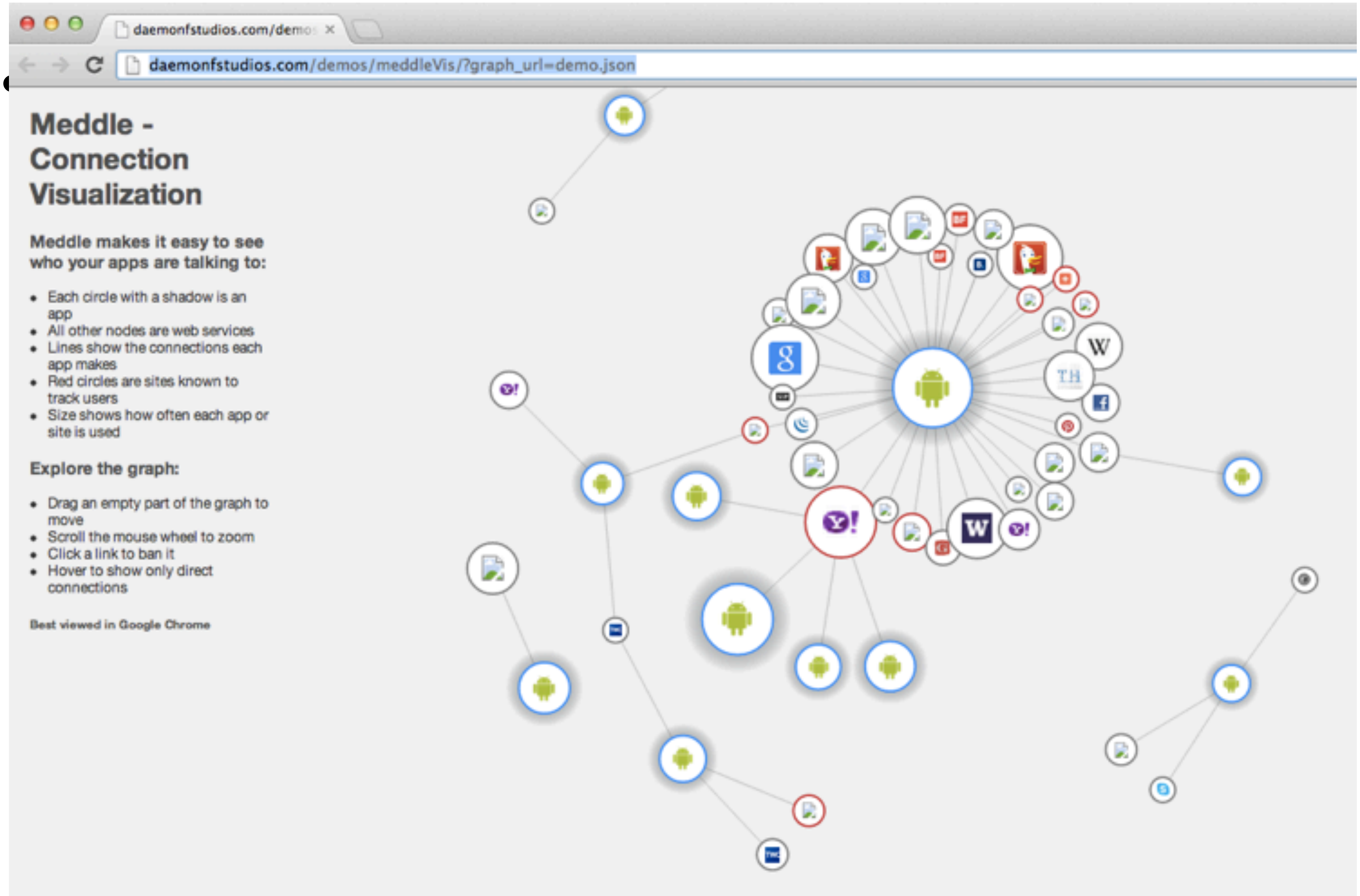
- Performance suffers for a number of reasons
  - –Network is slow
  - –Devices are slow
  - –Too many apps open at once
  - –Apps are poorly written

Apps for the Greater Good

- Goal: Make mobile performance more transparent
  - –Head-to-head comparisons (SpeedBump)
  - –Get what you pay for (ShortChanged)
  - –Mobile network cartography (MapMyNetwork)
- Goal: Use data to improve performance
  - –Comparison shopping (TimeToSwitch)
  - –Performance localization (SpeedSpotter)



Tracking the trackers



http://daemonfstudios.com/demos/meddleVis2/