Concurrency In Java

Programming Language Presentation

CS 5314 - Concepts of Programming Languages

Dr. James D. Arthur - Spring 1999

2 March 1999

Reena Khosla and Saverio Perugini

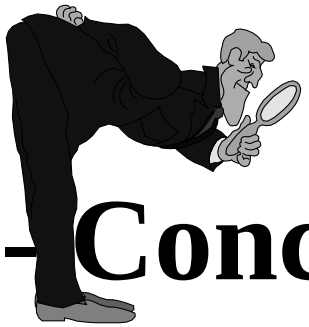
Presentation Outline

- Overall Objectives of Java
- Our Main Focus - Concurrency in Java
- How Java solves some of problems associated with concurrency
- Conclusion

Overall Objectives of Java

- Object-Oriented
- Architecture neutral
- Multithreaded
- Simple and Free

Our Main Focus - Concurrency



- What is Concurrency?
  - The ability of a program to perform multiple tasks simultaneously.
  - Mimics real world
- Example:
  - Web browser

Why does Java provide Concurrency?

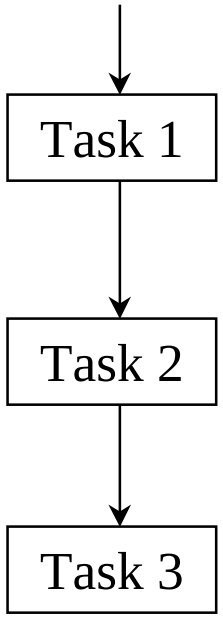
- It must, implicitly, because it is inherently,
- Architecture neutral
  - intended to shield programmer from idiosyncrasies of
    - operating systems and
    - hardware platforms

How Java achieves Concurrency

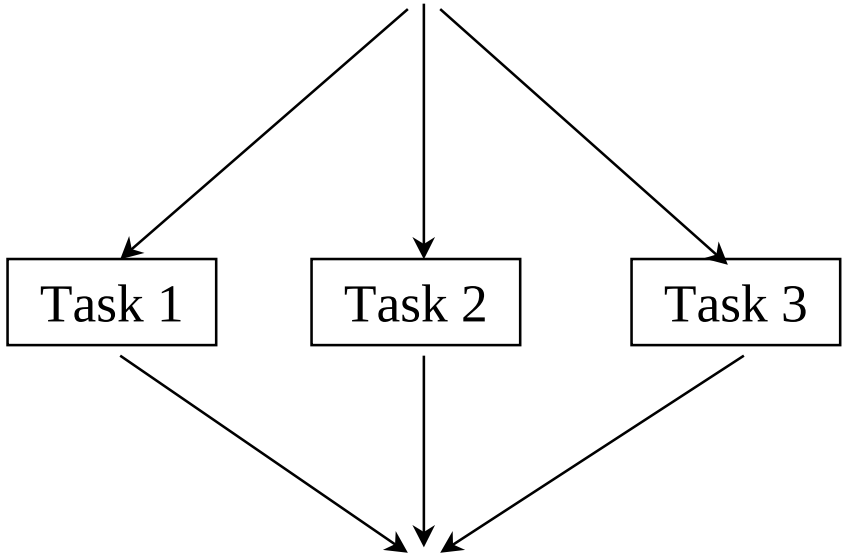
- Multithreaded
  - What are Threads?
    - Light-weight processes
  - Why use Threads?
    - Parallel Processing
    - Concurrent Processing
      - Improves Efficiency
  - Threads are built in Java - unlike C or C++

Sequential vs. Concurrent Processing

Sequential Flow

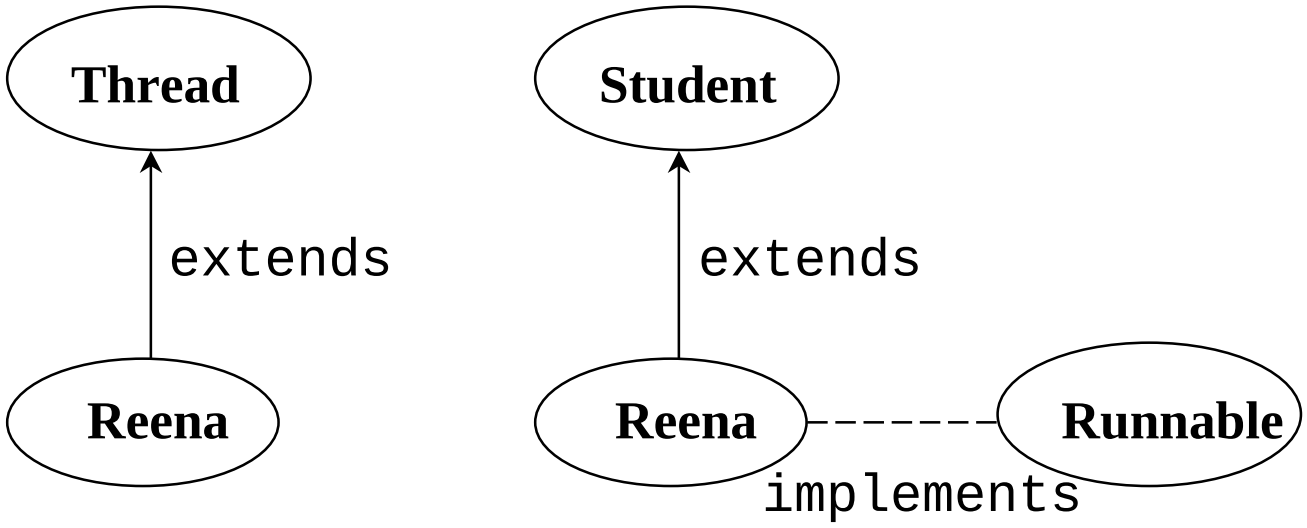


Parallel / Concurrent Flow

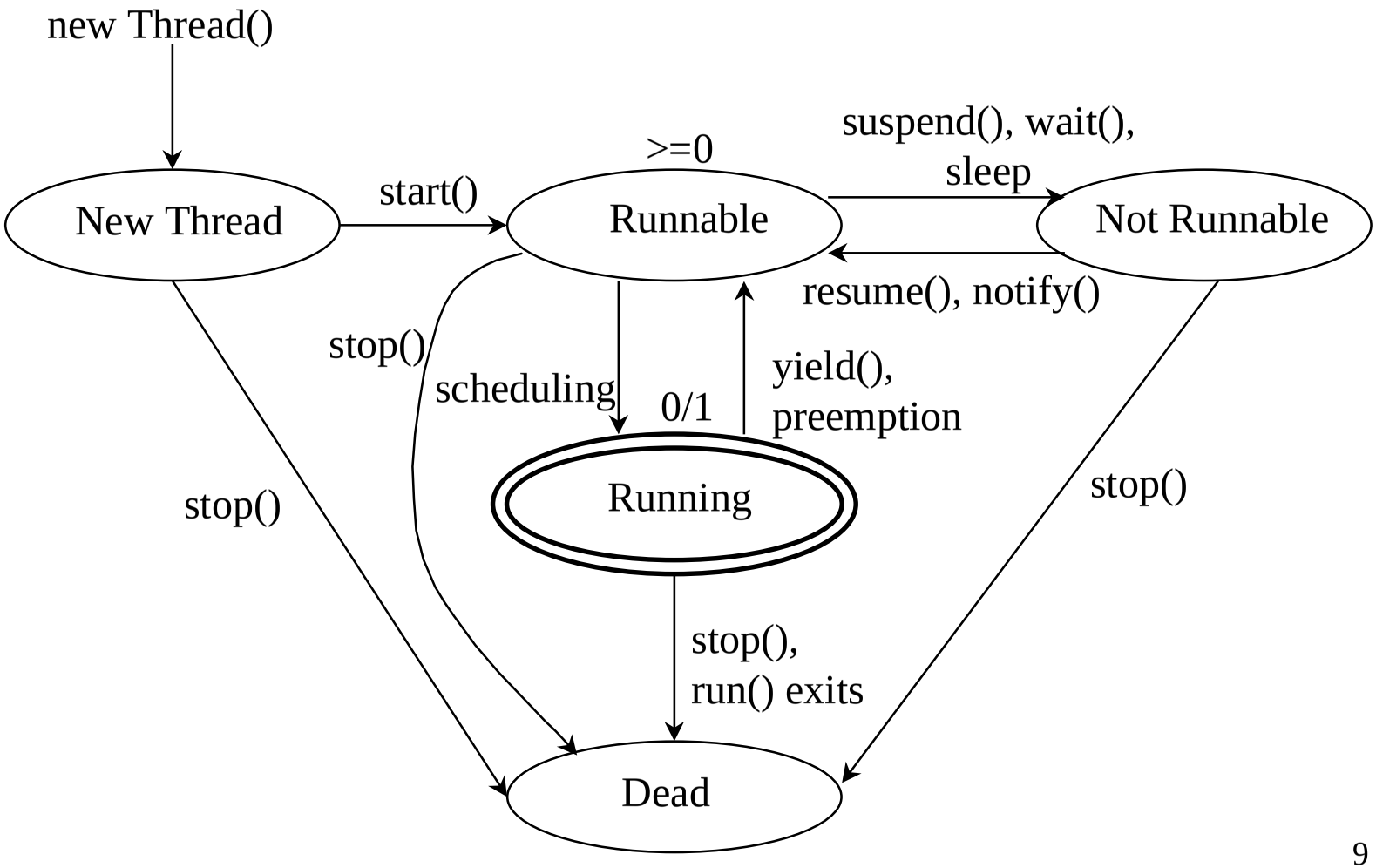


Two ways to create Threads

- Directly inherit from the **`Thread`** class
- Implement the **`Runnable`** interface

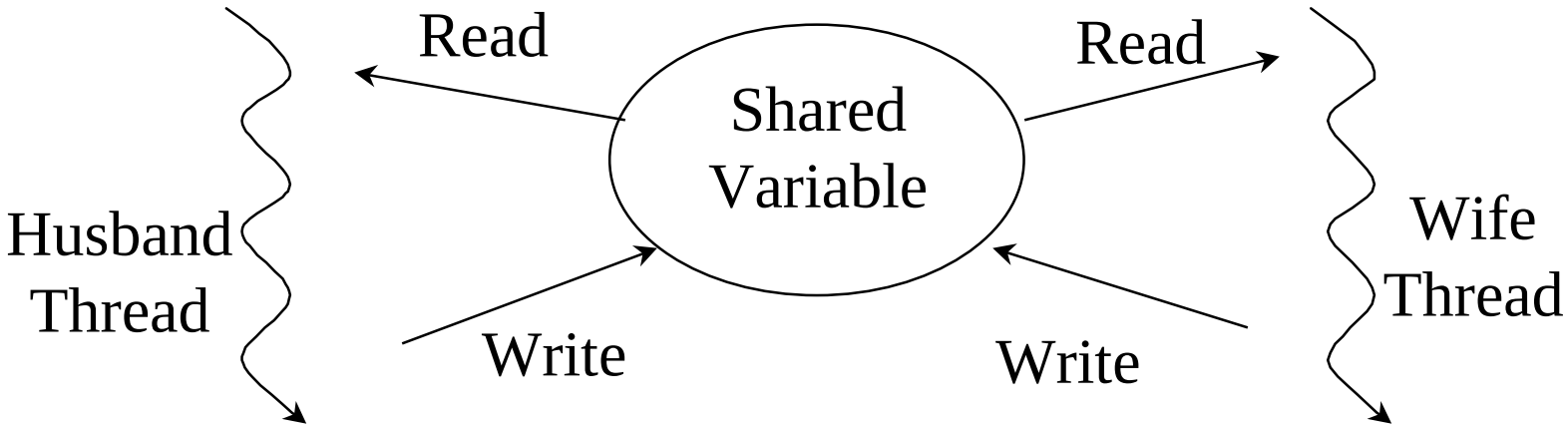


Thread States and Priorities



Threads - Problem?

- What if two threads modify a ***shared variable*** at the same time?



Bank Account!

Threads - Problem? (con't)

East Branch		West Branch
EBwithdraw = 500	balance	WBwithdraw = 300

fetch balance	1000	fetch balance
fetch EBwithdraw		fetch WBwithdraw
sub balance, EBwithdraw ⟶	500	
	700	⟵ sub balance, WBwithdraw

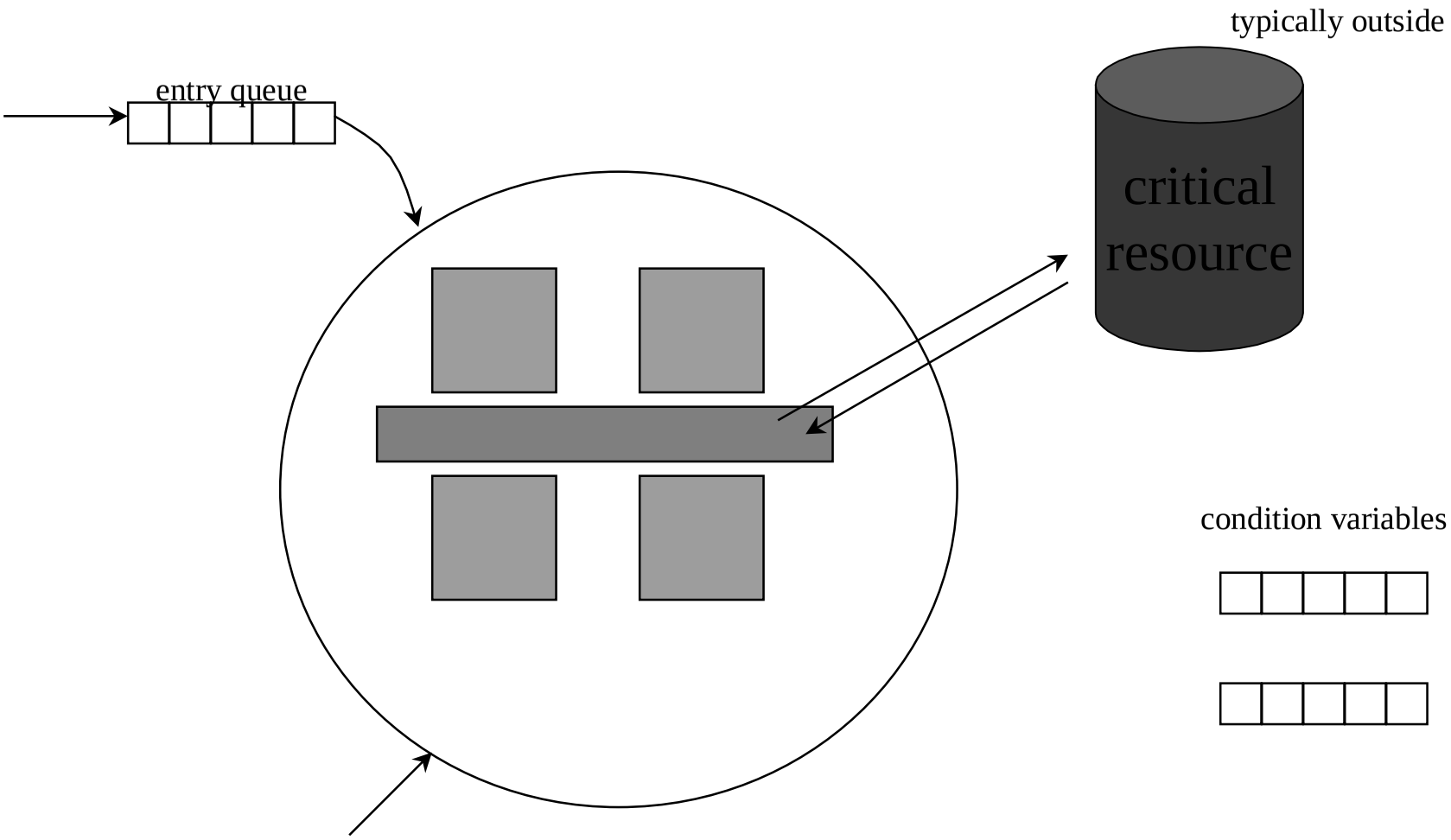
Mutual Exclusion and Synchronization

- First solution **- Boolean variable**
  - Problems
    - busy waiting, non-preemption, and complicated
- A solid solution **- Monitor**
  - mutual exclusion - achieved solely by use of monitor
  - synchronization - achieved by use of monitor and methods

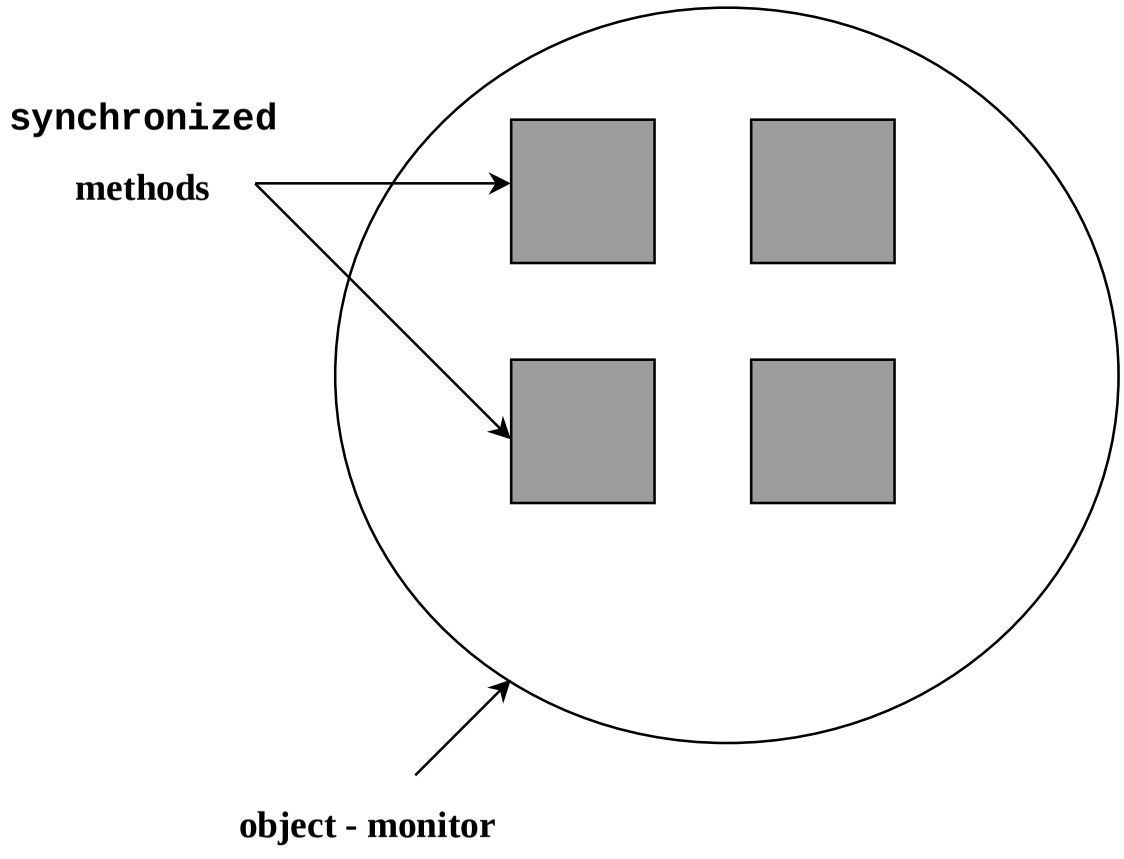
Monitors

- First proposed by Brinch Hansen
- Later refined, designed, and developed by C.A. R. Hoare in 1974
- An ADT
- Java has a slightly different model
- Used it to achieve mutual exclusion

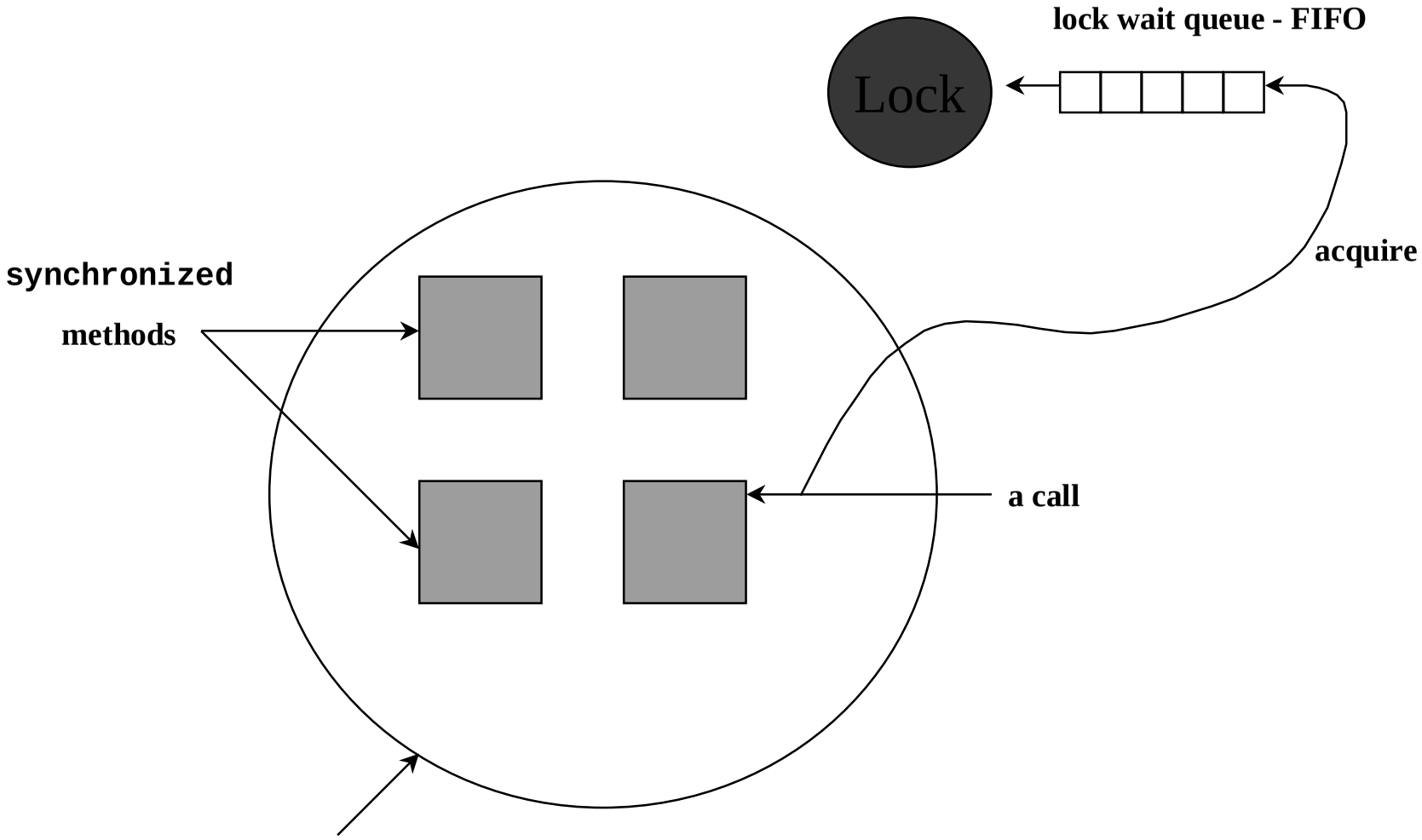
Hoare's Monitor Model



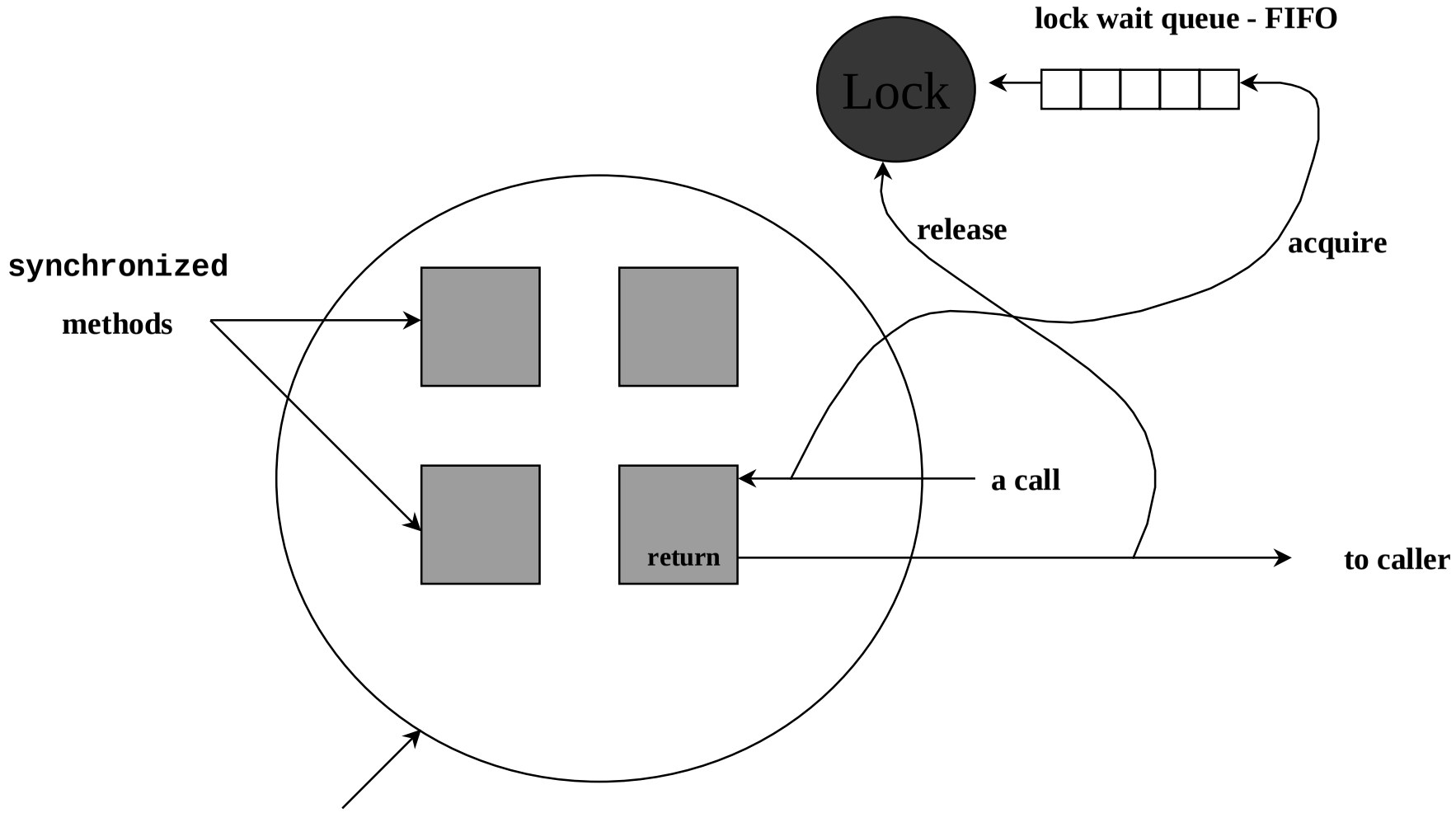
A Java Monitor



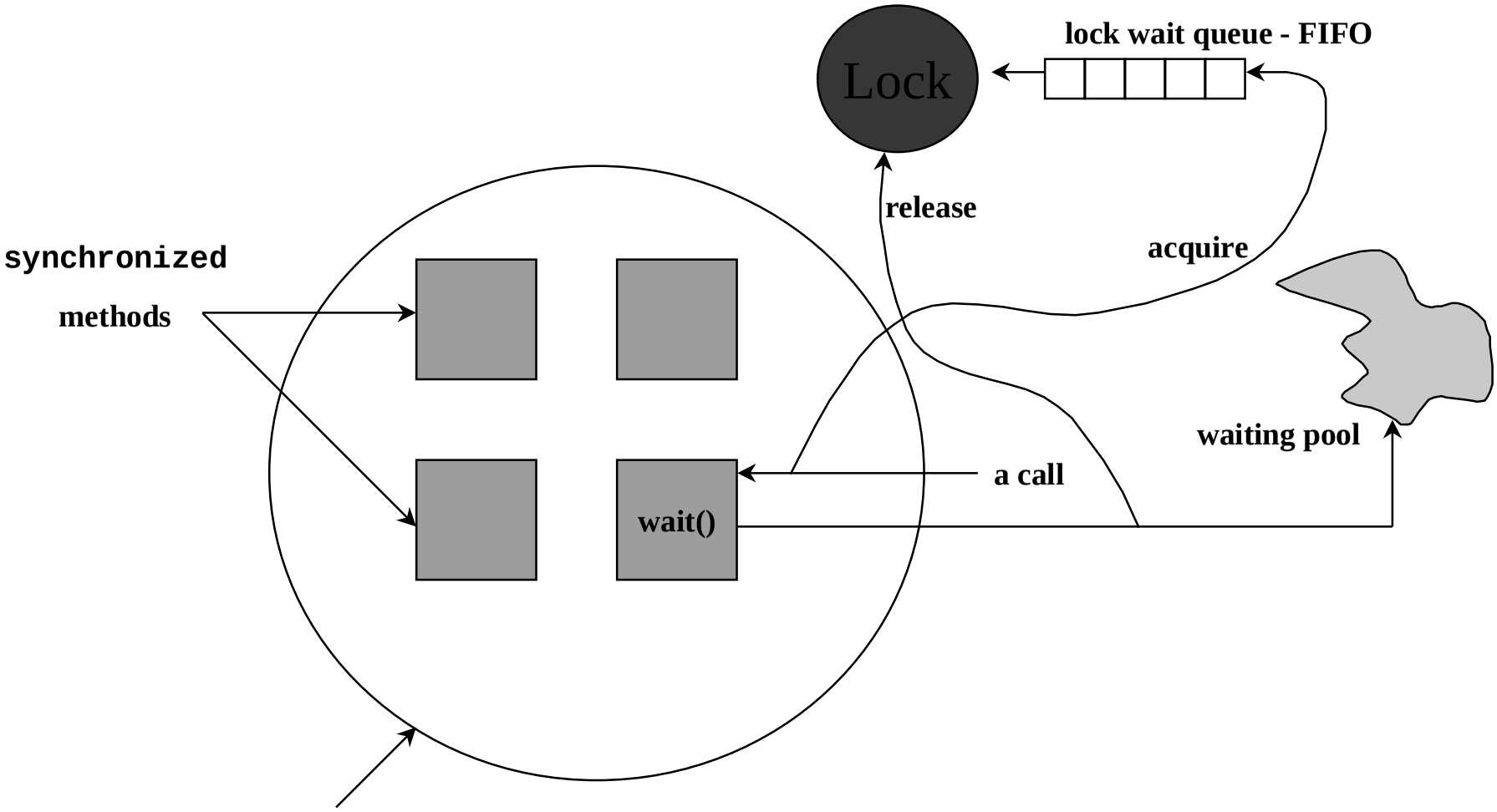
A Java Monitor - lock



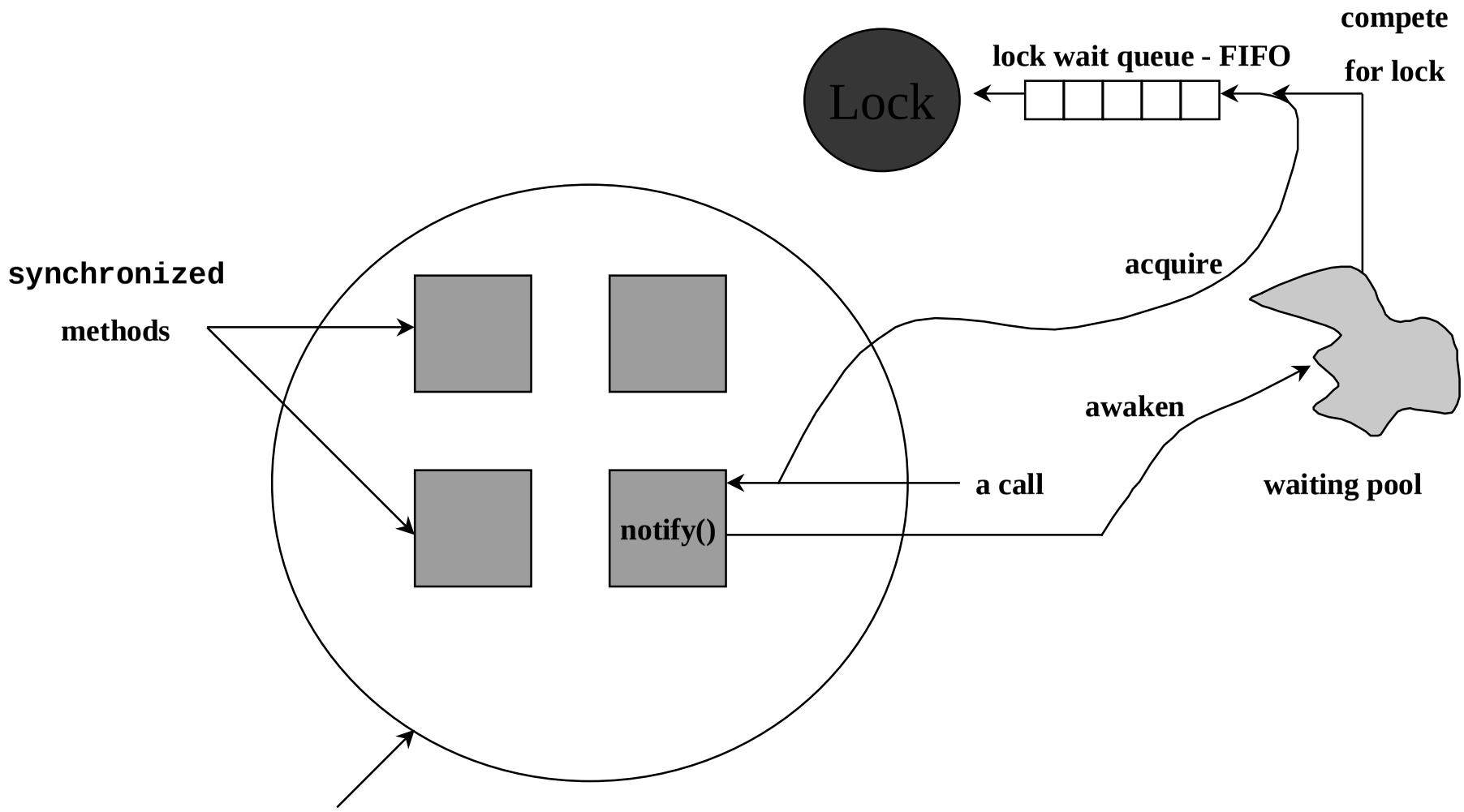
A Java Monitor - lock (con't)



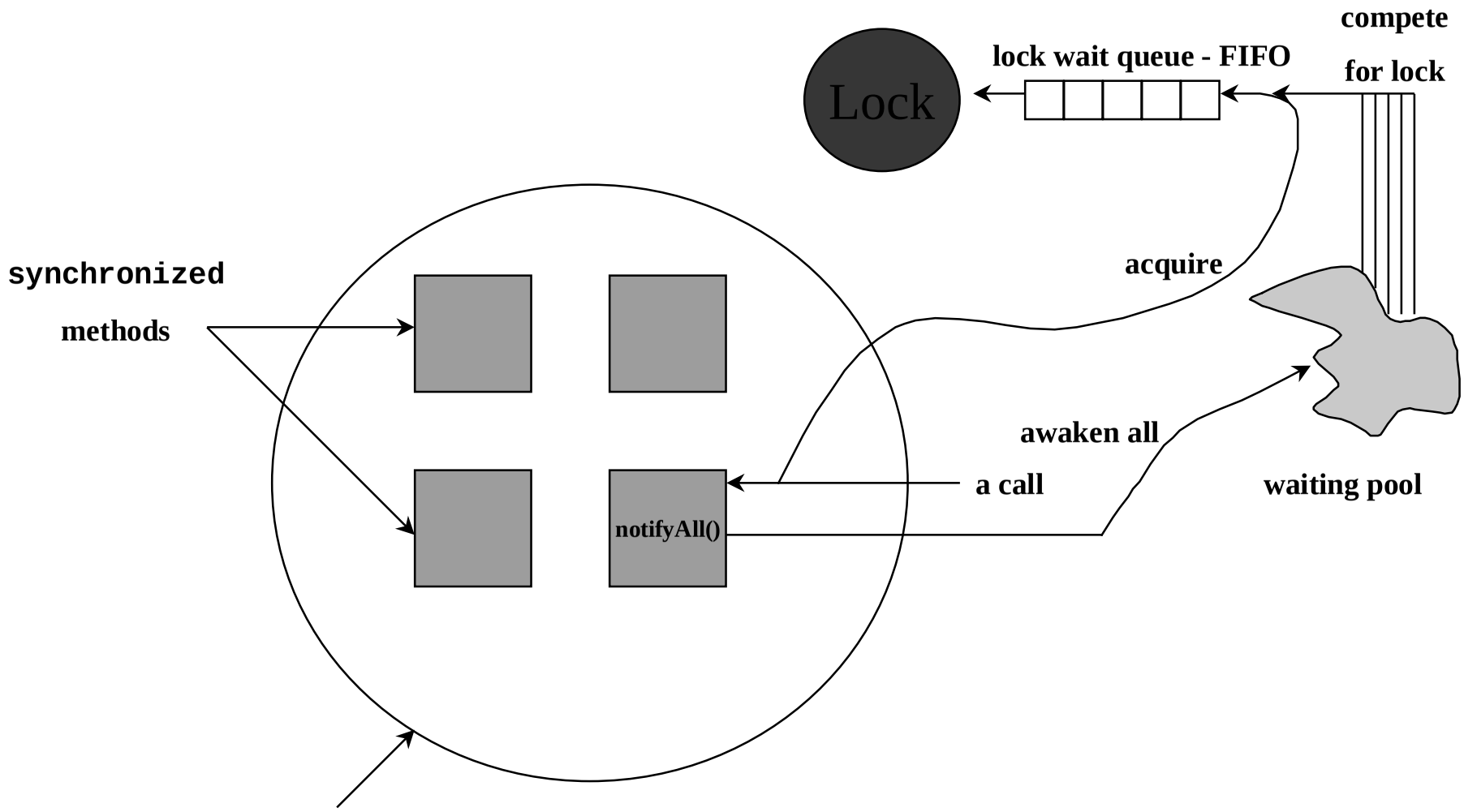
A Java Monitor - wait()



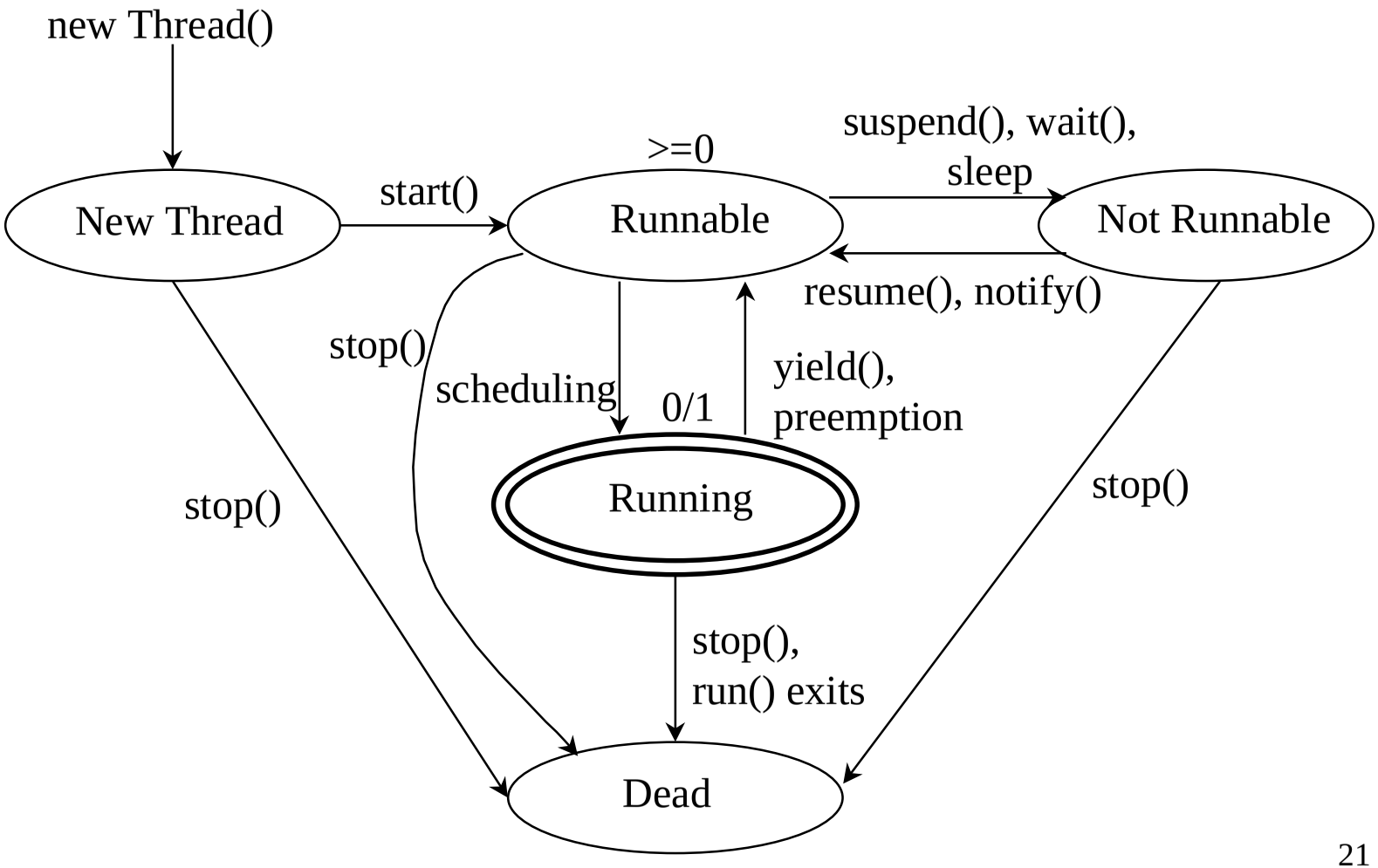
A Java Monitor - notify()



A Java Monitor - notifyAll()



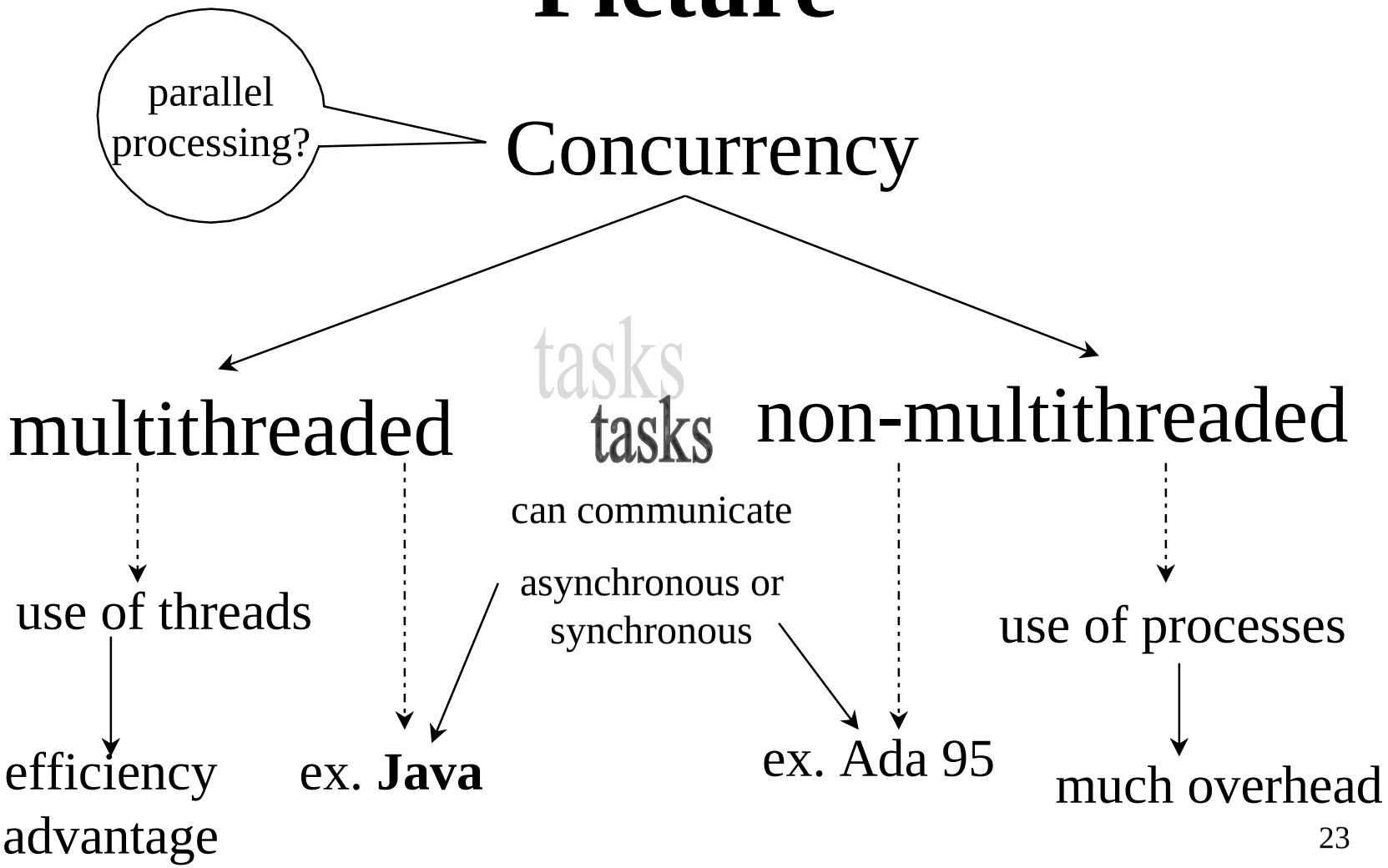
wait(), notify(), notifyAll()



Java & Ada

	Threads Monitor	Tasks Rendezvous
Synchronization between peer activities	No	Yes
Tasks selected randomly	No, FIFO	Yes
Wait/Notify operations	Yes	No
Distributed	No	Yes

Concurrency - The "BIG" Picture



Conclusions

- Need for concurrency
- Issues and problems regarding concurrency
- Different flavors of achieving concurrency
- Different ways of solving problems concurrency presents
- No “right” way - tradeoffs

Citations

- Mutual Exclusion and synchronization in Java - Dr. Dobb's Journal, Jan 1998 v23
- Java Logo - http://java.sun.com
- Advanced Flow of Control - Chapter 14, Book's name
- Parallel Programming in Java - ftp://actor.cs.vt.edu/pub/CS5204/JavaThreadsTutorial.ps

Questions/Comments?