USF UNIVERSITY OF SAN FRANCISCO
CHANGE THE WORLD FROM HERE

Liveness

CS 272 Software Development

Professor Sophie Engle
Department of Computer Science

Motivation

- We want *healthy* threads (i.e. **thread liveness**)
  - Thread should execute in a timely manner
- Several situations to avoid (i.e. **liveness problems**)
  - Threads can *stop* prematurely (deadlock)
  - Threads can *starve* and take a long time (starvation)
  - Threads can be too *distracted* (livelock)

http://docs.oracle.com/javase/tutorial/essential/concurrency/liveness.html

Deadlock

Deadlock

- Occurs when two or more threads must wait for each other to finish work
- Threads are indefinitely blocked and never complete
  - The threads are effectively dead (hence deadlock)
  - Similar effect as an infinite loop

http://docs.oracle.com/javase/tutorial/essential/concurrency/deadlock.html

Deadlock Example

```
1. void transfer(Account to, Account from, int amount) {
2.   lock(to);
3.   lock(from);
4.
5.   withdraw(from, amount);
6.   deposit(to, amount);
7.
8.   unlock(from);
9.   unlock(to);
10. }
```

Deadlock Example

#	transfer(ann, bev, amount)	transfer(bev, ann, amount)
1	lock(**ann**);	lock(**bev**);
2	lock(**bev**);	lock(**ann**);
3	withdraw(**bev**, amount);	withdraw(**ann**, amount);
4	deposit(**ann**, amount);	deposit(**bev**, amount);
5	unlock(**bev**);	unlock(**ann**);
6	unlock(**ann**);	unlock(**bev**);
7	*Will this finish?*	

Deadlock Example

#	transfer(ann, bev, amount)	transfer(bev, ann, amount)
1	lock(**ann**);	lock(**bev**);
2	lock(**bev**); // must wait	lock(**ann**); // must wait
3	~~*withdraw(**bev**, amount);*~~	~~*withdraw(**ann**, amount);*~~
4	~~*deposit(**ann**, amount);*~~	~~*deposit(**bev**, amount);*~~
5	~~*unlock(**bev**);*~~	~~*unlock(**ann**);*~~
6	~~*unlock(**ann**);*~~	~~*unlock(**bev**);*~~
7	*DEADLOCK on line 2!*	

Deadlock Avoidance

- Deadlock **detection** and **prevention** difficult
  - Must turn to heuristics for **avoidance**
- Avoid obtaining multiple locks if possible
- Try to obtain locks in same order
- Avoid dependencies and cycles

Starvation and Livelock

Starvation

- Occurs when a higher priority thread prevents a lower priority thread from accessing a resource
  - Resource may be CPU time or something else
  - Often caused by overzealous synchronization
- Lower priority threads are starved of the resource, and take too long (or never) complete

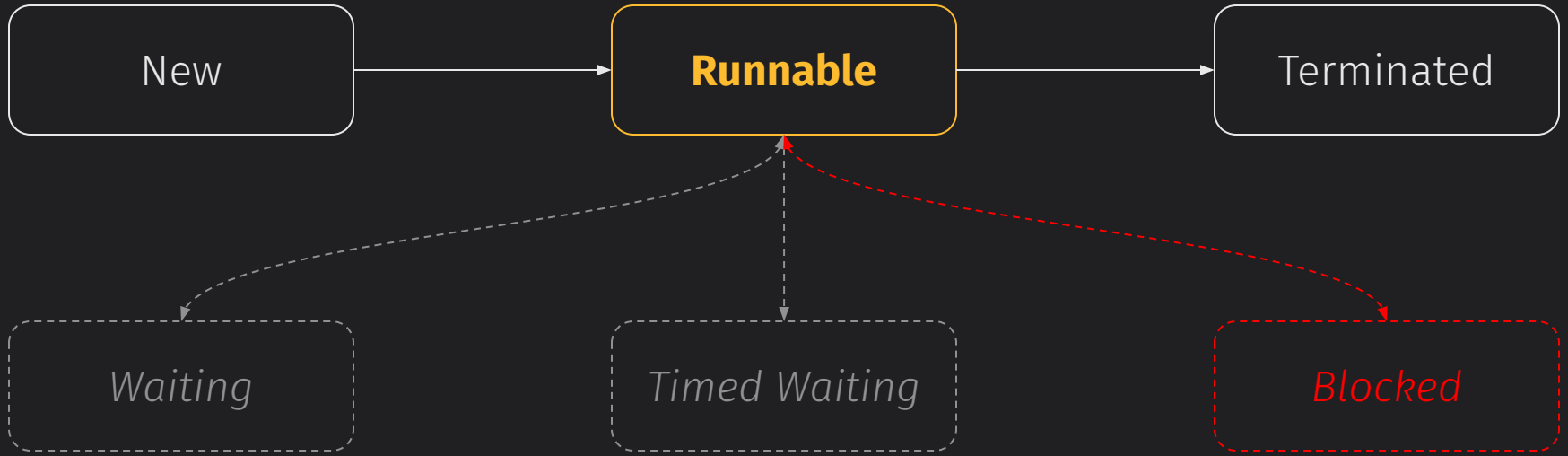
http://docs.oracle.com/javase/tutorial/essential/concurrency/starvelive.html

Livelock

- Occurs when a thread triggers another thread, which triggers the previous thread, and so on
- Threads spend all effort on responding to each other
  - Threads are not blocking each other, so still "lively" but locked in a loop preventing progress
  - *Sometimes caused by deadlock prevention!*

http://docs.oracle.com/javase/tutorial/essential/concurrency/starvelive.html

Thread States



https://www.cs.usfca.edu/~cs272/javadoc/api/java.base/java/lang/Thread.State.html

CHANGE THE WORLD FROM HERE

Software Development
Department of Computer Science

Professor Sophie Engle
sjengle.cs.usfca.edu