Overview of Concurrent Programming Concepts

Douglas C. Schmidt
d.schmidt@vanderbilt.edu
www.dre.vanderbilt.edu/~schmidt

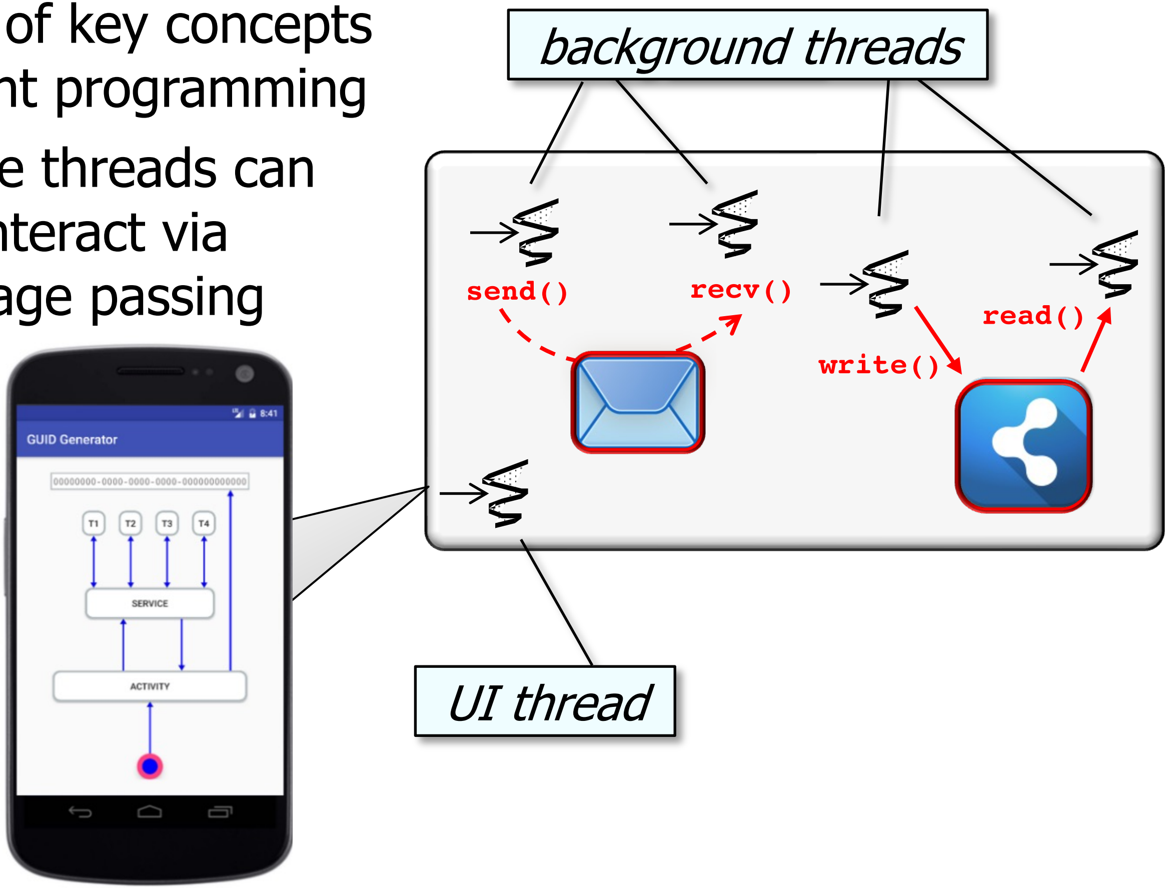
Professor of Computer Science

Institute for Software
Integrated Systems

Vanderbilt University
Nashville, Tennessee, USA

Learning Objectives in this Part of the Lesson

- Understand the meaning of key concepts associated with concurrent programming
  - e.g., where two or more threads can run simultaneously & interact via shared objects & message passing



Concurrent programming helps address some 'cons' of sequential programming

An Overview of Concurrent Programming

An Overview of Concurrent Programming

- Concurrent programming is a form of computing where 2+ threads can run simultaneously & compete for resources

See en.wikipedia.org/wiki/Concurrency_(computer_science)

An Overview of Concurrent Programming

- Concurrent programming is a form of computing where 2+ threads can run simultaneously & compete for resources

A thread is a unit of execution for a stream of instructions that can run concurrently on one or more processor cores over its lifetime

Processor cores

See docs.oracle.com/javase/tutorial/essential/concurrency/threads.html

An Overview of Concurrent Programming

- Concurrent programming is a form of computing where 2+ threads can run simultaneously & compete for resources

A thread typically runs in a process, which allocates & manages resources & prevents corruption from threads in other processes

Processor cores

See en.wikipedia.org/wiki/Process_(computing)

An Overview of Concurrent Programming

- Concurrent programming is a form of computing where 2+ threads can run simultaneously & compete for resources

Resources managed by processes include cores, files, memory, network connections, & synchronizers, which threads compete for with other threads

Processor cores

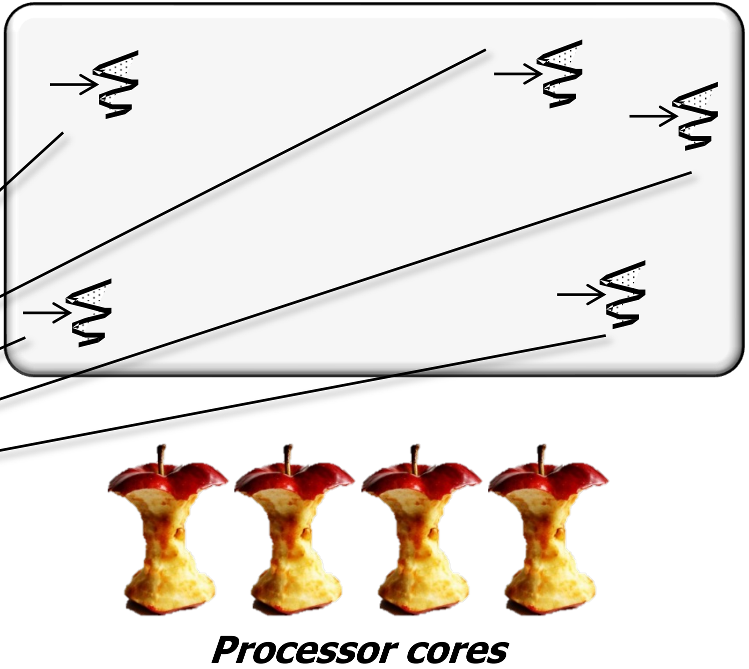
See en.wikipedia.org/wiki/Computational_resource

An Overview of Concurrent Programming

- Concurrent programming is a form of computing where 2+ threads can run simultaneously & compete for resources

```
for (int i = 0; i < 5; i++)
  new Thread(() ->
                  someComputation()).
                  start();
```

This code snippet creates/starts 5 Java Thread objects that concurrently run "someComputation" on 4 processor cores

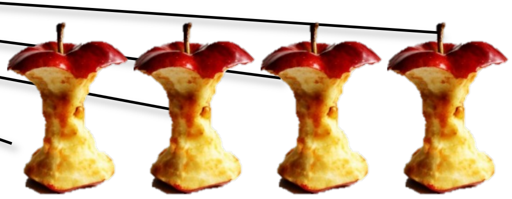


An Overview of Concurrent Programming

- Concurrent programming is a form of computing where 2+ threads can run simultaneously & compete for resources

```
for (int i = 0; i < 5; i++)
  new Thread(() ->
                 someComputation()).
                 start();
```

A Java Thread object needn't run on the same core throughout its lifetime, but instead it can be "multiplexed" across multiple cores via "time-slicing"



Processor cores

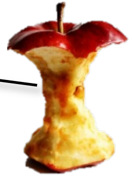
See scalibq.wordpress.com/2012/06/01/multi-core-and-multi-threading

An Overview of Concurrent Programming

- Concurrent programming is a form of computing where 2+ threads can run simultaneously & compete for resources

```
for (int i = 0; i < 5; i++)
  new Thread(() ->
             someComputation()).
             start();
```

Multiple threads can also be multi-plexed over a single-core processor

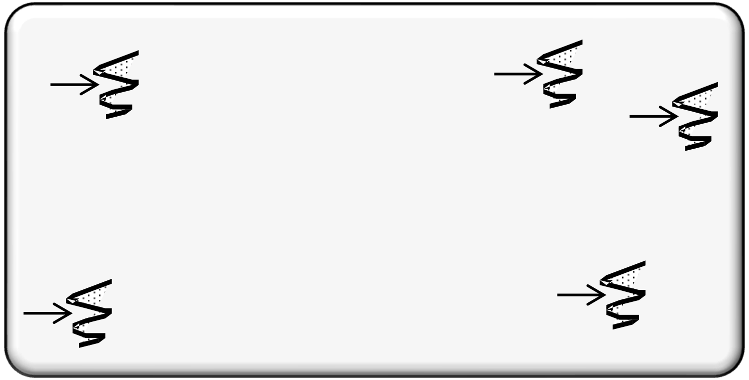


See en.wikipedia.org/wiki/Single-core

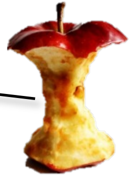
An Overview of Concurrent Programming

- Concurrent programming is a form of computing where 2+ threads can run simultaneously & compete for resources

```
for (int i = 0; i < 5; i++)
  new Thread(() ->
                   someComputation()).
                   start();
```



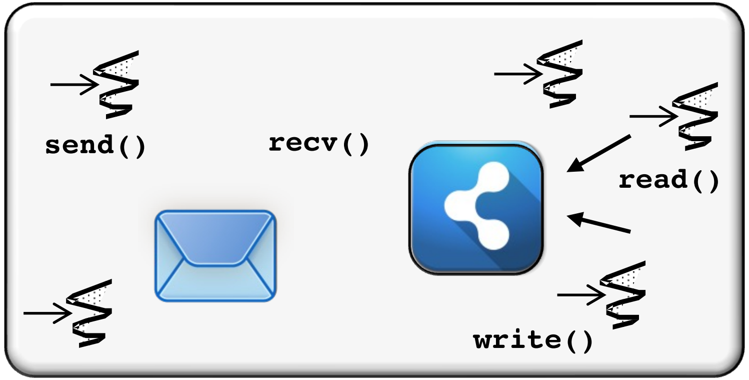
However, single-core processors are becoming rare for general-purpose computing devices..



See www.quora.com/Are-single-core-CPUs-still-produced

An Overview of Concurrent Programming

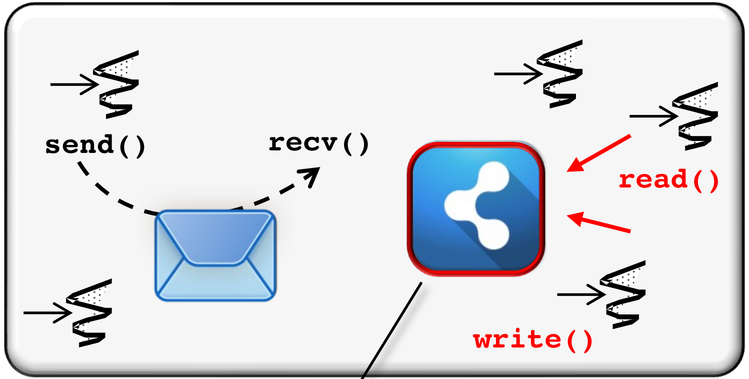
- Concurrent threads typically interact via shared objects (synchronizers) & message passing



See upcoming lesson on "*Overview of How Concurrent Programs are Developed in Java*"

An Overview of Concurrent Programming

- Concurrent threads typically interact via shared objects (synchronizers) & message passing



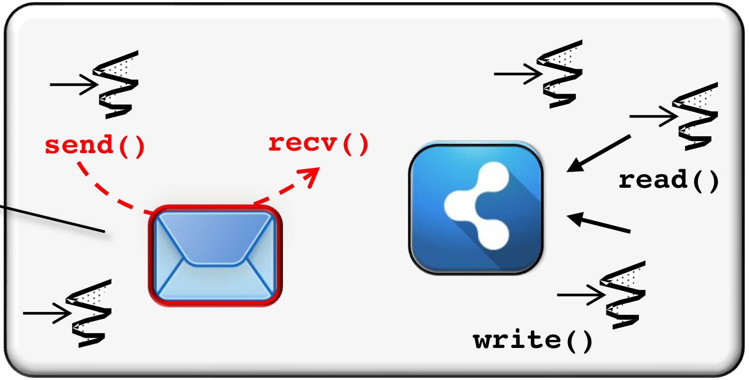
Shared objects (synchronizers) can be used to ensure mutual exclusion between—& coordination amongst—multiple threads

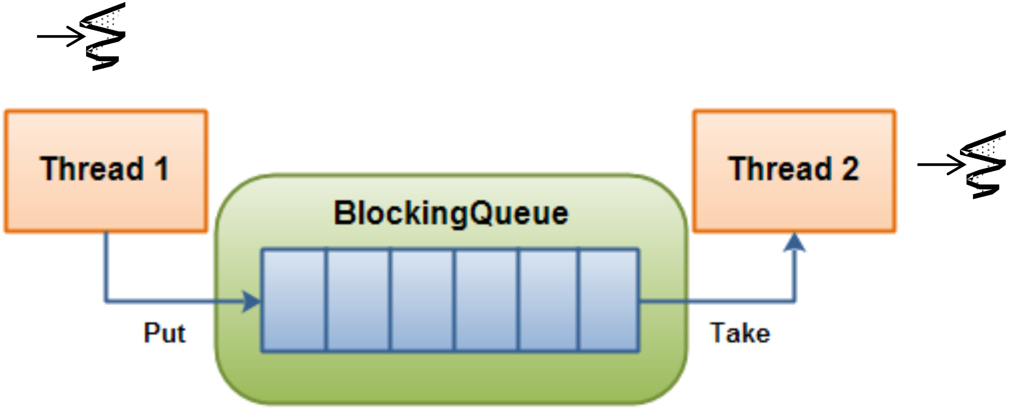
See upcoming lesson on "*Overview of How Concurrent Programs are Developed in Java*"

An Overview of Concurrent Programming

- Concurrent threads typically interact via shared objects (synchronizers) & message passing

Multiple threads can pass messages via queues that are properly synchronized





See upcoming lesson on "*Overview of How Concurrent Programs are Developed in Java*"

An Overview of Concurrent Programming

- Unlike sequential programming, different executions of a concurrent program may produce different orderings of instructions:

See earlier lesson on "*Overview of Sequential Programming Concepts*"

An Overview of Concurrent Programming

- Unlike sequential programming, different executions of a concurrent program may produce different orderings of instructions:
  - The textual order of the source code doesn't define the order of execution

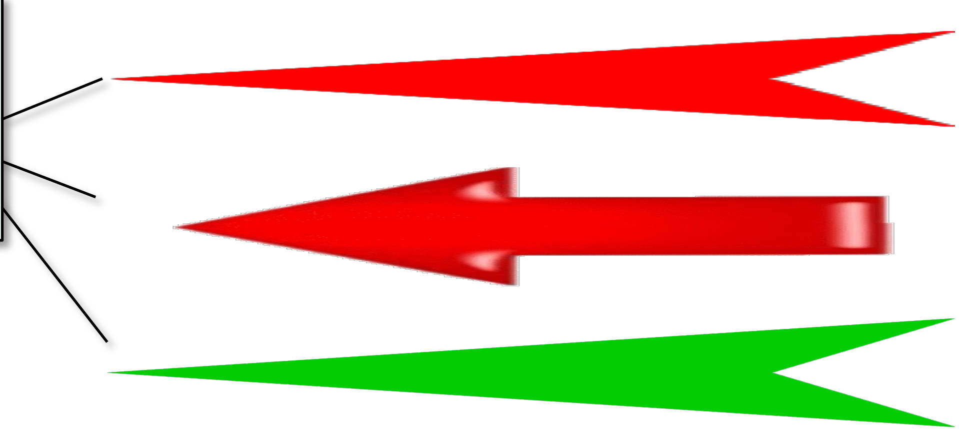
```
new Thread(() ->
               computationA()).
               start();

new Thread(() ->
               computationB()).
               start();

new Thread(() ->
               computationC()).
               start();
```

computationA(), computationB(), & computationC() can run in any order after their threads start up

See en.wikipedia.org/wiki/Indeterminacy_in_concurrent_computation

An Overview of Concurrent Programming

- Unlike sequential programming, different executions of a concurrent program may produce different orderings of instructions:
  - The textual order of the source code doesn't define the order of execution
  - Operations are permitted to overlap in time across multiple cores

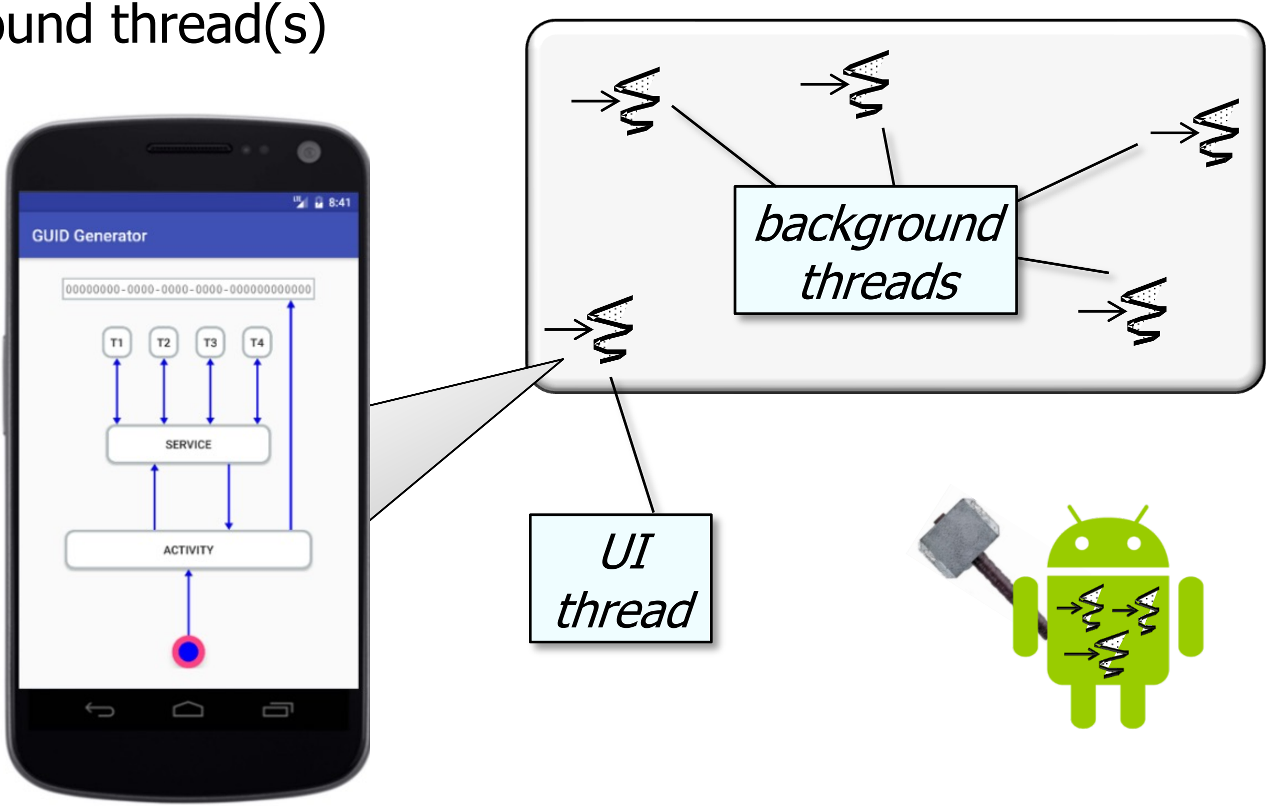


Multiple computations can execute concurrently (during overlapping time periods) instead of sequentially (i.e., one completing before the next starts)

See en.wikipedia.org/wiki/Concurrent_computing

An Overview of Concurrent Programming

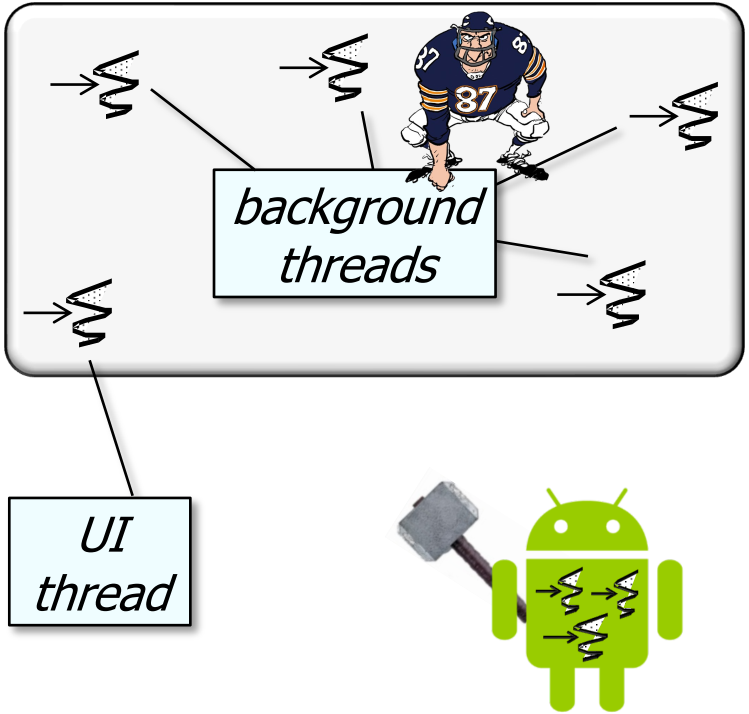
- Concurrent programming can offload work from the user interface (UI) thread to background thread(s)



See developer.android.com/topic/performance/threads.html

An Overview of Concurrent Programming

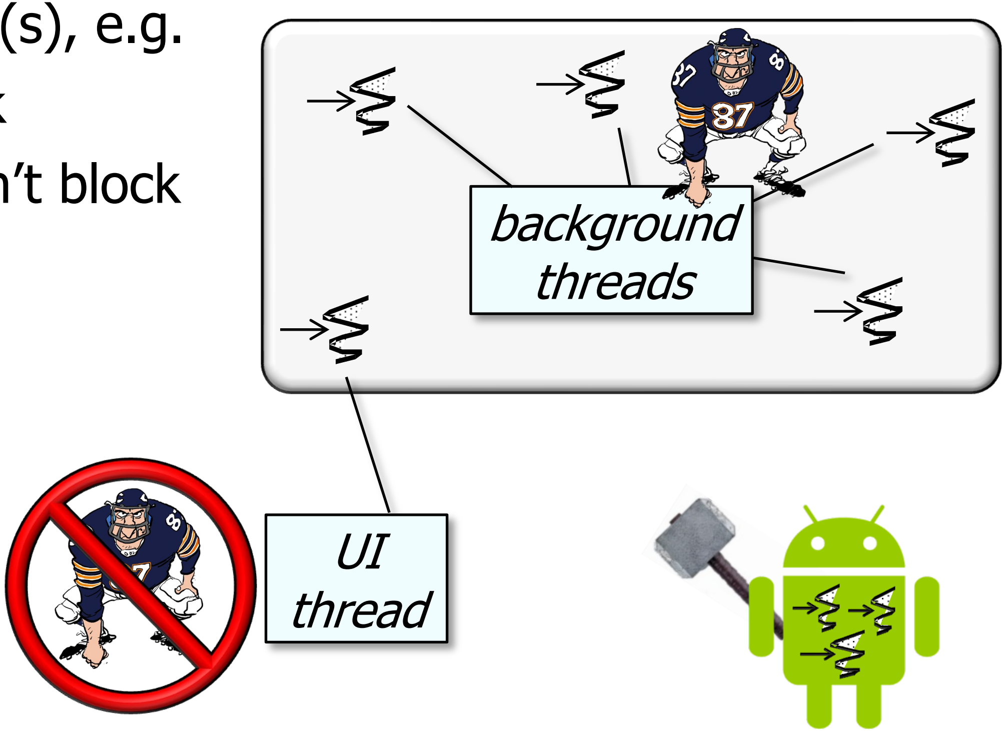
- Concurrent programming can offload work from the user interface (UI) thread to background thread(s), e.g.
  - Background thread(s) can block



See developer.android.com/training/multiple-threads/communicate-ui.html

An Overview of Concurrent Programming

- Concurrent programming can offload work from the user interface (UI) thread to background thread(s), e.g.
  - Background thread(s) can block
    - However, the UI thread doesn't block



background threads

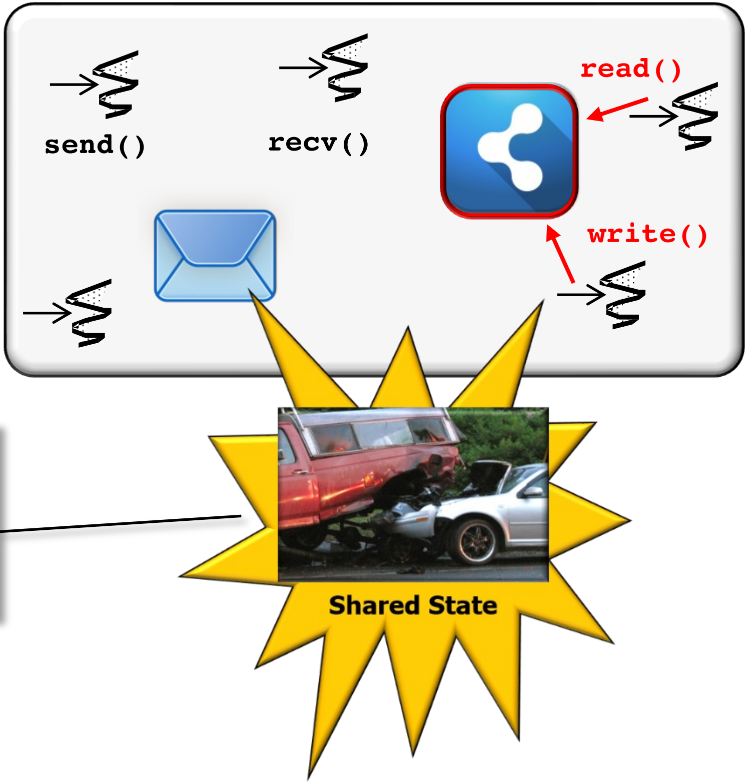
UI thread

See developer.android.com/training/multiple-threads/communicate-ui.html

An Overview of Concurrent Programming

- Concurrent programming can offload work from the user interface (UI) thread to background thread(s), e.g.
  - Background thread(s) can block
  - Any mutable state shared between these threads must be protected to avoid concurrency hazards

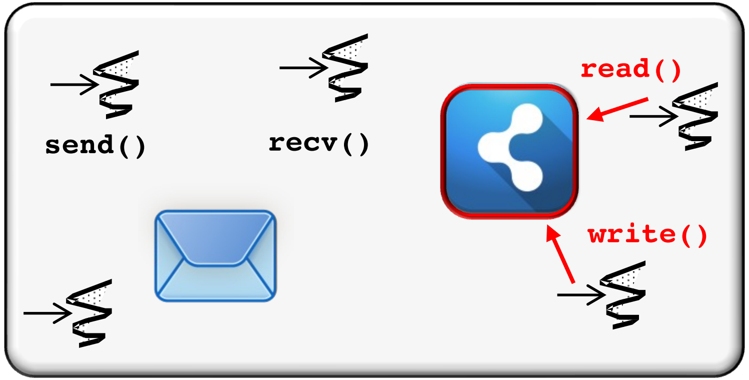
e.g., a "race condition" can occur when a program depends upon the sequence or timing of threads for it to operate properly



See upcoming lesson on "*Overview of Concurrency in Java*"

An Overview of Concurrent Programming

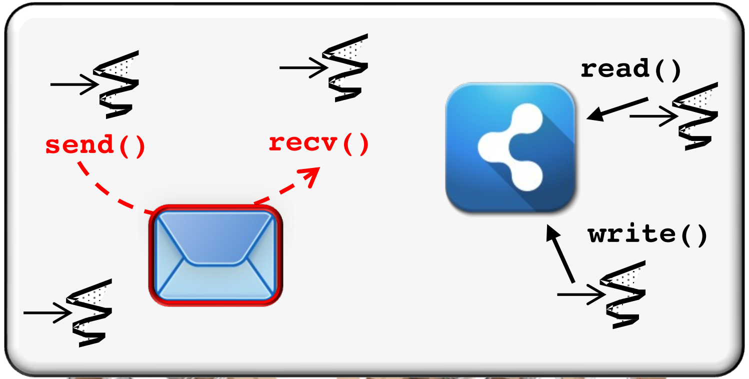
- Concurrent programming can offload work from the user interface (UI) thread to background thread(s), e.g.
  - Background thread(s) can block
  - Any mutable state shared between these threads must be protected to avoid concurrency hazards
    - Motivates the need for various types of Java synchronizers

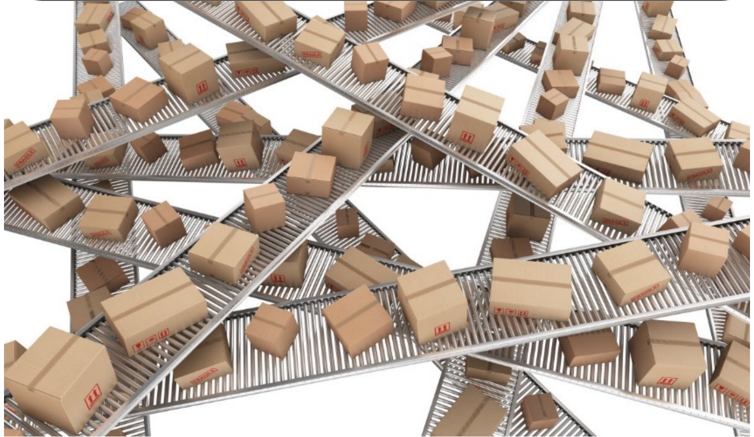


See docs.oracle.com/javase/tutorial/essential/concurrency/sync.html

An Overview of Concurrent Programming

- Concurrent programming can offload work from the user interface (UI) thread to background thread(s), e.g.
  - Background thread(s) can block
  - Any mutable state shared between these threads must be protected to avoid concurrency hazards
  - Message passing can avoid directly sharing state across multiple threads

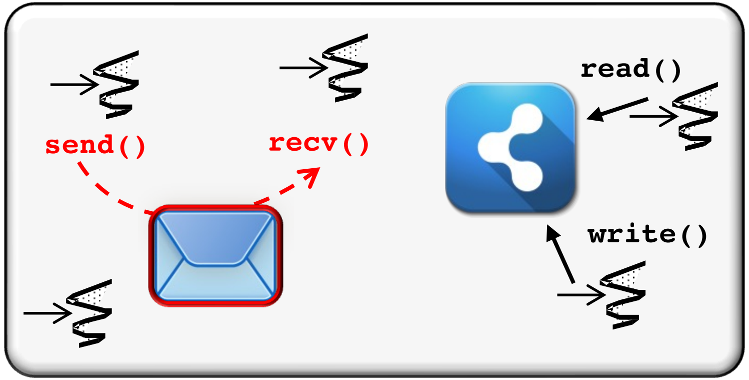


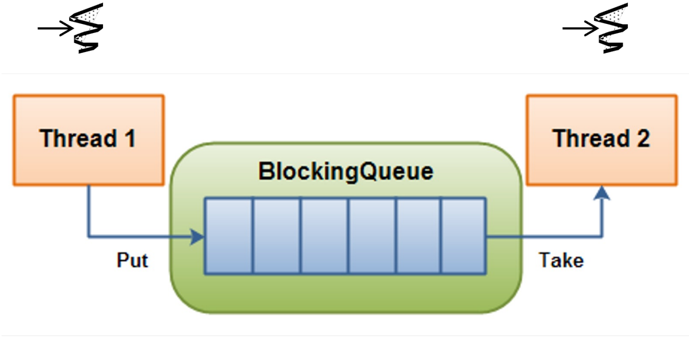


See upcoming lesson on "*Overview of Concurrent Programming in Java*"

An Overview of Concurrent Programming

- Concurrent programming can offload work from the user interface (UI) thread to background thread(s), e.g.
  - Background thread(s) can block
  - Any mutable state shared between these threads must be protected to avoid concurrency hazards
  - Message passing can avoid directly sharing state across multiple threads
    - Combines internal synchronization, memory visibility guarantees, passing immutable state, & encapsulation to ensure threads can interact with each other without sharing mutable state





See dev.to/yiksanchan/implementing-java-util-concurrent-arrayblockingqueue-3k3

End of Overview of Concurrent Programming Concepts

Discussion Questions

1. Which of the following are key goals of concurrency?
   a. *Concurrency is used to offload work from a non-blocking user interface thread to background threads that can block*
   b. *Concurrency is used to efficiently partition tasks into sub-tasks & combine results*
   c. *Concurrency always executes the same sequence of instructions & it will always produce the same results since execution is deterministic*
   d. *Concurrency focuses on optimizing performance by avoiding resource sharing & not blocking*