# Overview of Sequential Programming Concepts

Douglas C. Schmidt

d.schmidt@vanderbilt.edu

www.dre.vanderbilt.edu/~schmidt

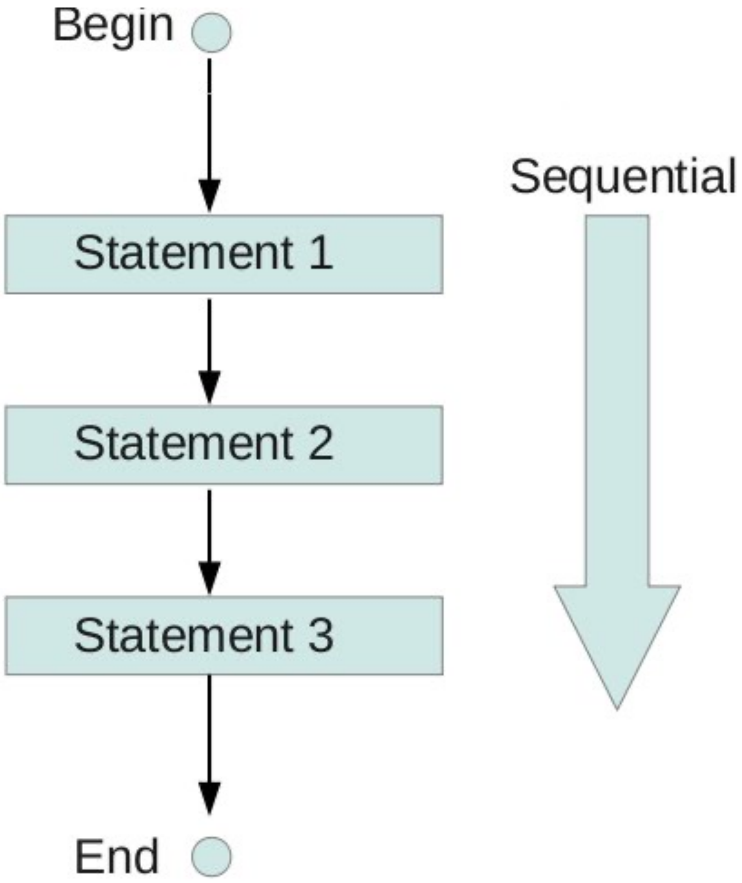
Professor of Computer Science

Institute for Software
Integrated Systems

Vanderbilt University
Nashville, Tennessee, USA

# Learning Objectives in this Lesson

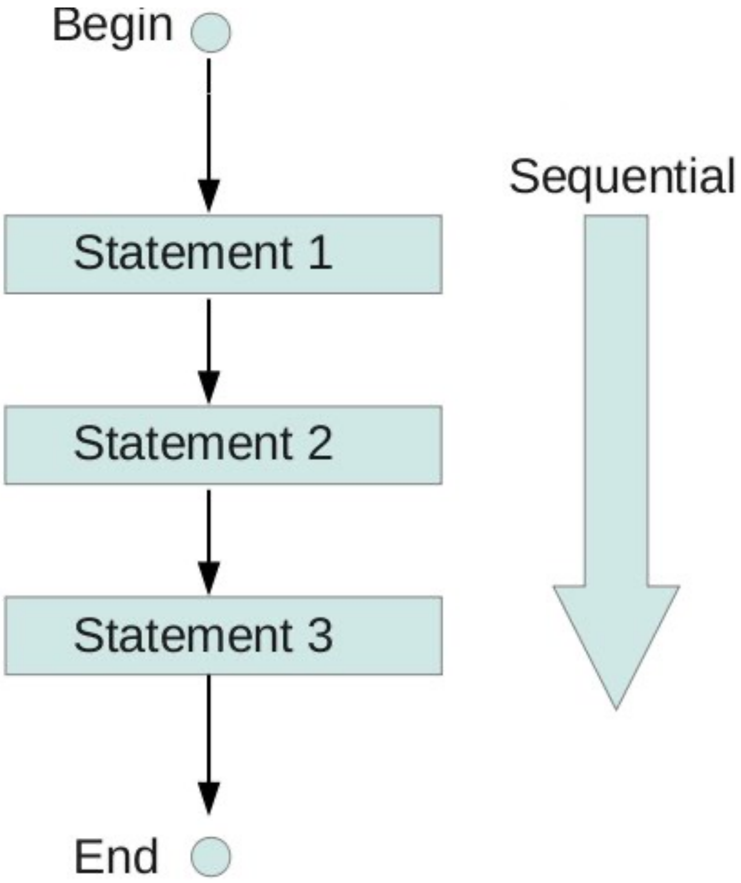
- Understand the meaning of key concepts associated with sequential programming
  - e.g., each step in a program is executed in order one at a time



# Learning Objectives in this Lesson

- Understand the meaning of key concepts associated with sequential programming
  - e.g., each step in a program is executed in order one at a time

*Mastering these concepts is essential before trying to learn more advanced concurrent & parallel programming concepts*



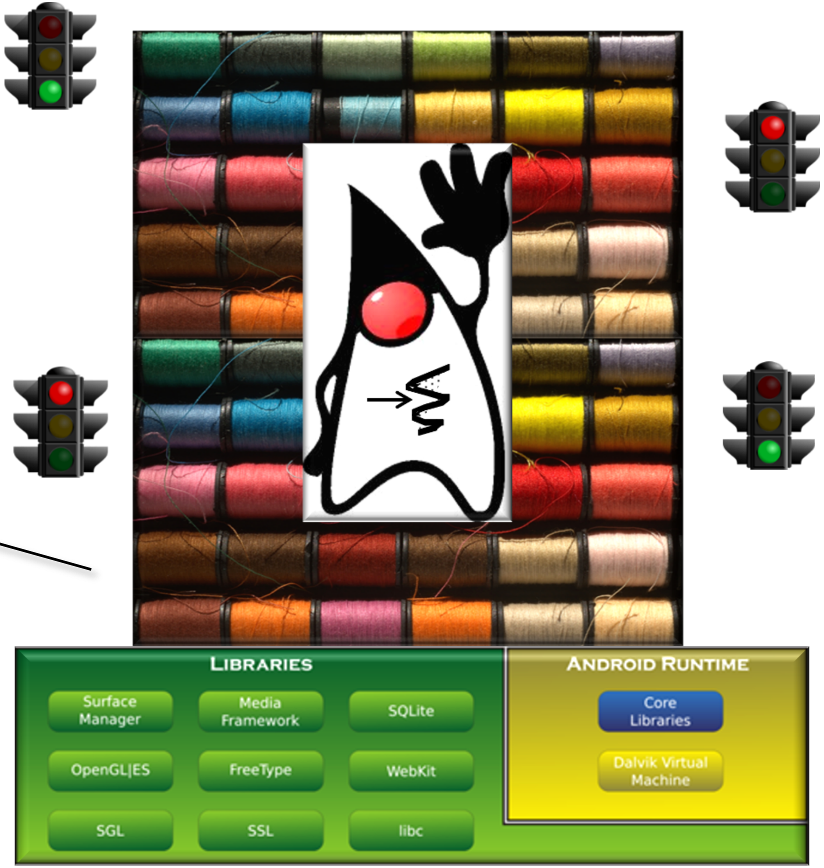
# Learning Objectives in this Lesson

- Understand the meaning of key concepts associated with sequential programming
- Recognize the pros & cons of sequential programming

# Learning Objectives in this Lesson

- Understand the meaning of key concepts associated with sequential programming
- Recognize the pros & cons of sequential programming

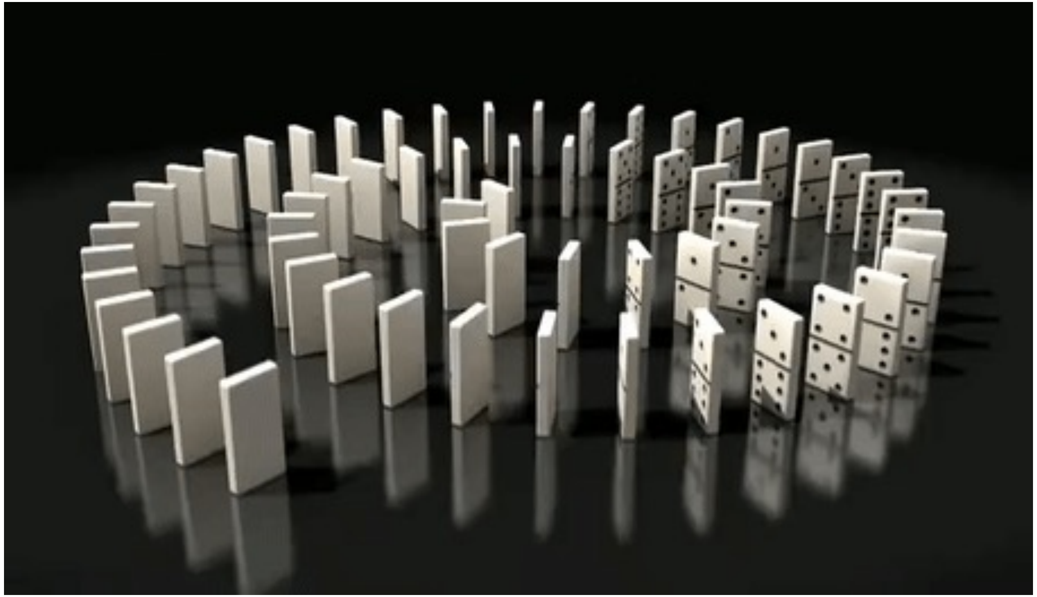
*Overcoming these 'cons' motivates our upcoming focus on concurrent & parallel programming techniques for the Java & Android platforms*



# An Overview of Sequential Programming

# An Overview of Sequential Programming

- Sequential programming is a form of computing that executes the same sequence of instructions & always produces the same results
  - i.e., execution is *deterministic*



See en.wikipedia.org/wiki/Sequential_algorithm

# An Overview of Sequential Programming

- Sequential programming is a form of computing that executes the same sequence of instructions & always produces the same results
  - i.e., execution is *deterministic*

*Given a certain input, the same output will always be produced in the same order*

See en.wikipedia.org/wiki/Deterministic_algorithm

# An Overview of Sequential Programming

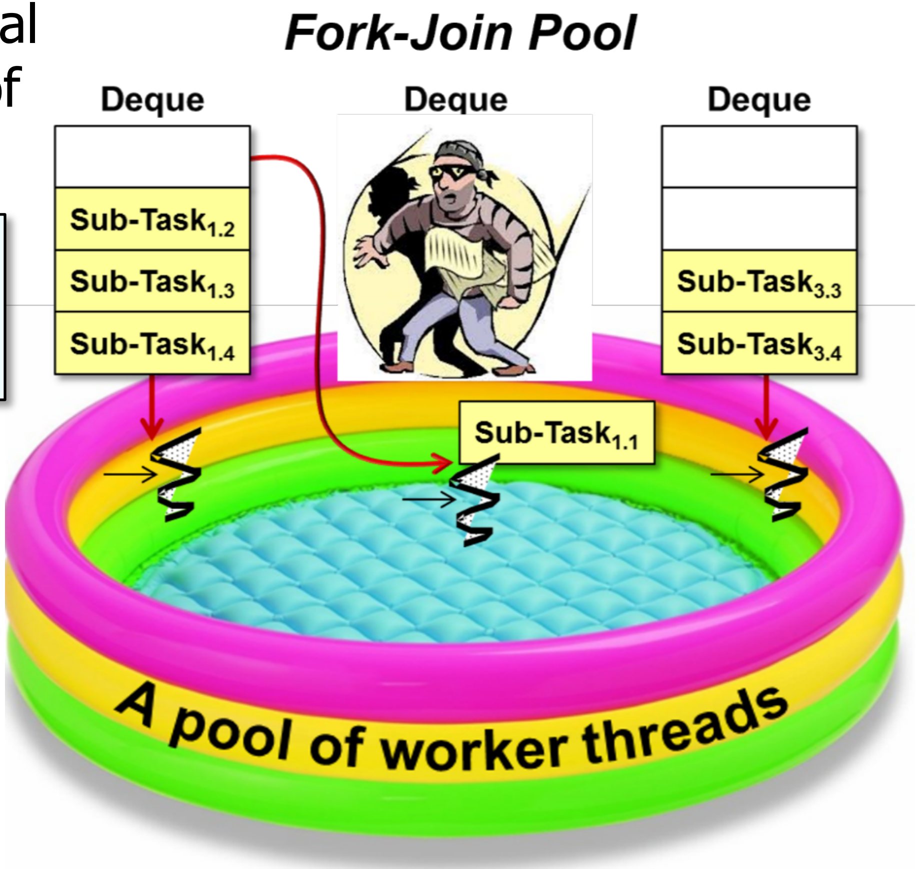
- The deterministic behavior of sequential programs assumes no deliberate use of randomness, of course

See en.wikipedia.org/wiki/Randomized_algorithm

# An Overview of Sequential Programming

- The deterministic behavior of sequential programs assumes no deliberate use of randomness, of course

*See upcoming lessons on the Java Fork-Join framework for coverage of how randomness is applied in concurrent & parallel programs*



See gee.cs.oswego.edu/dl/papers/fj.pdf

# An Overview of Sequential Programming

- Sequential programs have two characteristics

See www.doc.ic.ac.uk/~jnm/concurrency/online/concurrent/tsld007.htm

# An Overview of Sequential Programming

- Sequential programs have two characteristics:
  - The textual order of statements specifies their order of execution

```
public E get(int index) {
  rangeCheck(index);

  return elementData
               (index);
}
```

# An Overview of Sequential Programming

- Sequential programs have two characteristics:
  - The textual order of statements specifies their order of execution

```
public E get(int index) {
  rangeCheck(index);

  return elementData
               (index);
}
```

*e.g., chaos & insanity will occur in Java's ArrayList get() method if rangeCheck() is not called before elementData()!!!*

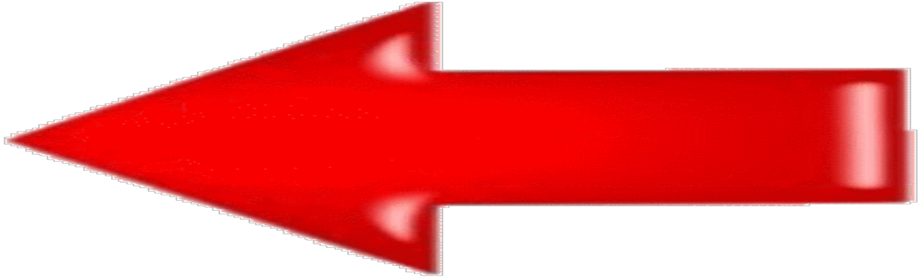
See src/share/classes/java/util/ArrayList.java

# An Overview of Sequential Programming

- Sequential programs have two characteristics:
  - The textual order of statements specifies their order of execution
  - Successive statements must execute without any temporal overlap visible to programs

Consider the code sequence

```
a = b + c
d = e - a
```



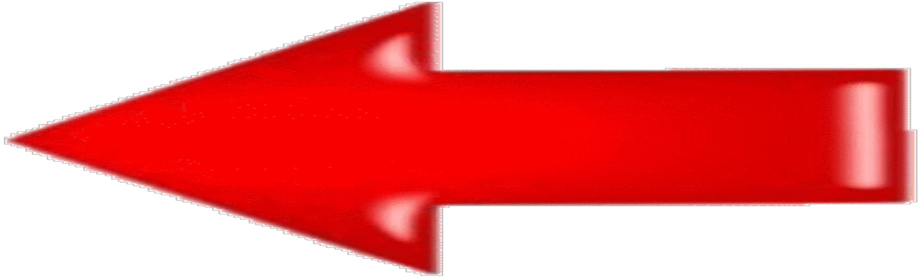
# An Overview of Sequential Programming

- Sequential programs have two characteristics:
  - The textual order of statements specifies their order of execution
  - Successive statements must execute without any temporal overlap visible to programs

Consider the code sequence

```
a = b + c
d = e - a
```

*The value of 'a' must be assigned before the value of 'd' is assigned*

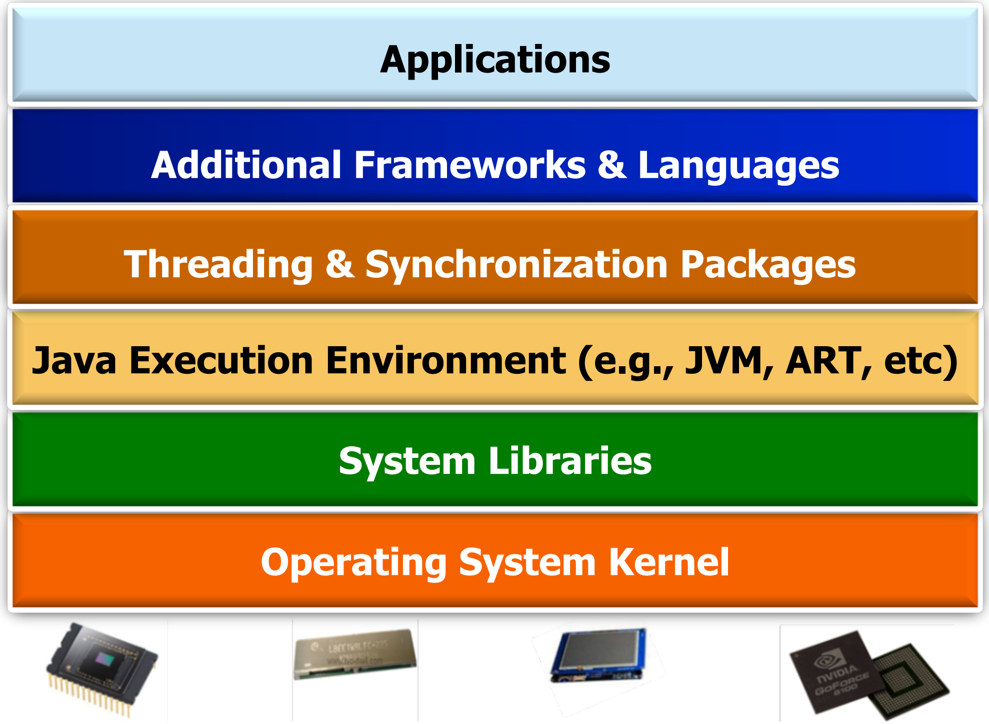


# An Overview of Sequential Programming

- Sequential programs have two characteristics:
  - The textual order of statements specifies their order of execution
  - Successive statements must execute without any temporal overlap visible to programs
    - However, lower layers in the solution stack can reorder instructions transparently

Consider the code sequence

```
a = b + c
d = e - a
```



Applications

Additional Frameworks & Languages

Threading & Synchronization Packages

Java Execution Environment (e.g., JVM, ART, etc)

System Libraries

Operating System Kernel

See [en.wikipedia.org/wiki/Solution_stack](https://en.wikipedia.org/wiki/Solution_stack)

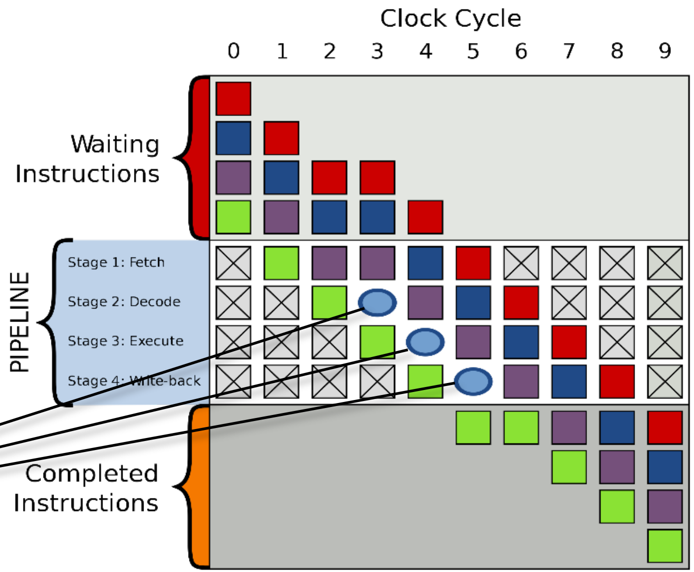
# An Overview of Sequential Programming

- Sequential programs have two characteristics:
  - The textual order of statements specifies their order of execution
  - Successive statements must execute without any temporal overlap visible to programs
    - However, lower layers in the solution stack can reorder instructions transparently

Consider the code sequence

a = b + c
d = e - a

Assuming a, b, c, d, & e are in memory & loads/ stores take one clock cycle out-of-order, then instruction scheduling eliminates pipeline stalls



*e.g., out-of-order execution is used to avoid "pipeline stalls" that delay instruction execution*

See en.wikipedia.org/wiki/Out-of-order_execution & en.wikipedia.org/wiki/Pipeline_stall

# An Overview of Sequential Programming

- Sequential programs have two characteristics:
  - The textual order of statements specifies their order of execution
  - Successive statements must execute without any temporal overlap visible to programs
    - However, lower layers in the solution stack can reorder instructions transparently

Consider the code sequence

a = b + c
d = e - a

Assuming a, b, c, d, & e are in memory & loads/ stores take one clock cycle out-of-order, then instruction scheduling eliminates pipeline stalls

*Original code with stalls:*

```
LD    Rb, b
LD    Rc, c
  stall ●
ADD   Ra, Rb, Rc
SD    Ra, a
LD    Re, e
  stall ●
SUB   Rd, Re, Ra
SD    Rd, d
```

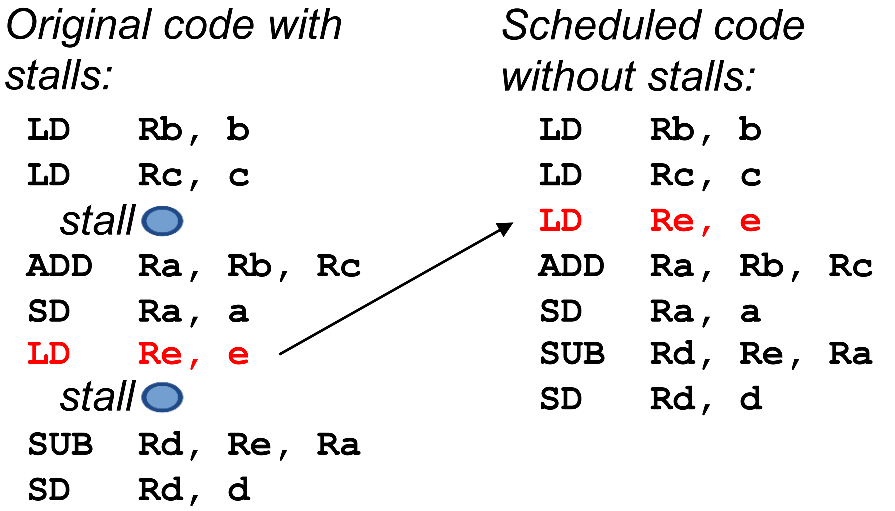
# An Overview of Sequential Programming

- Sequential programs have two characteristics:
    - The textual order of statements specifies their order of execution
    - Successive statements must execute without any temporal overlap visible to programs
        - However, lower layers in the solution stack can reorder instructions transparently

Consider the code sequence

$$a = b + c$$
$$d = e - a$$

Assuming a, b, c, d, & e are in memory & loads/stores take one clock cycle out-of-order, then instruction scheduling eliminates pipeline stalls



*Original code with stalls:*

```
LD    Rb, b
LD    Rc, c
  stall
ADD   Ra, Rb, Rc
SD    Ra, a
LD    Re, e
  stall
SUB   Rd, Re, Ra
SD    Rd, d
```

*Scheduled code without stalls:*

```
LD    Rb, b
LD    Rc, c
LD    Re, e
ADD   Ra, Rb, Rc
SD    Ra, a
SUB   Rd, Re, Ra
SD    Rd, d
```

See en.wikipedia.org/wiki/Instruction_scheduling

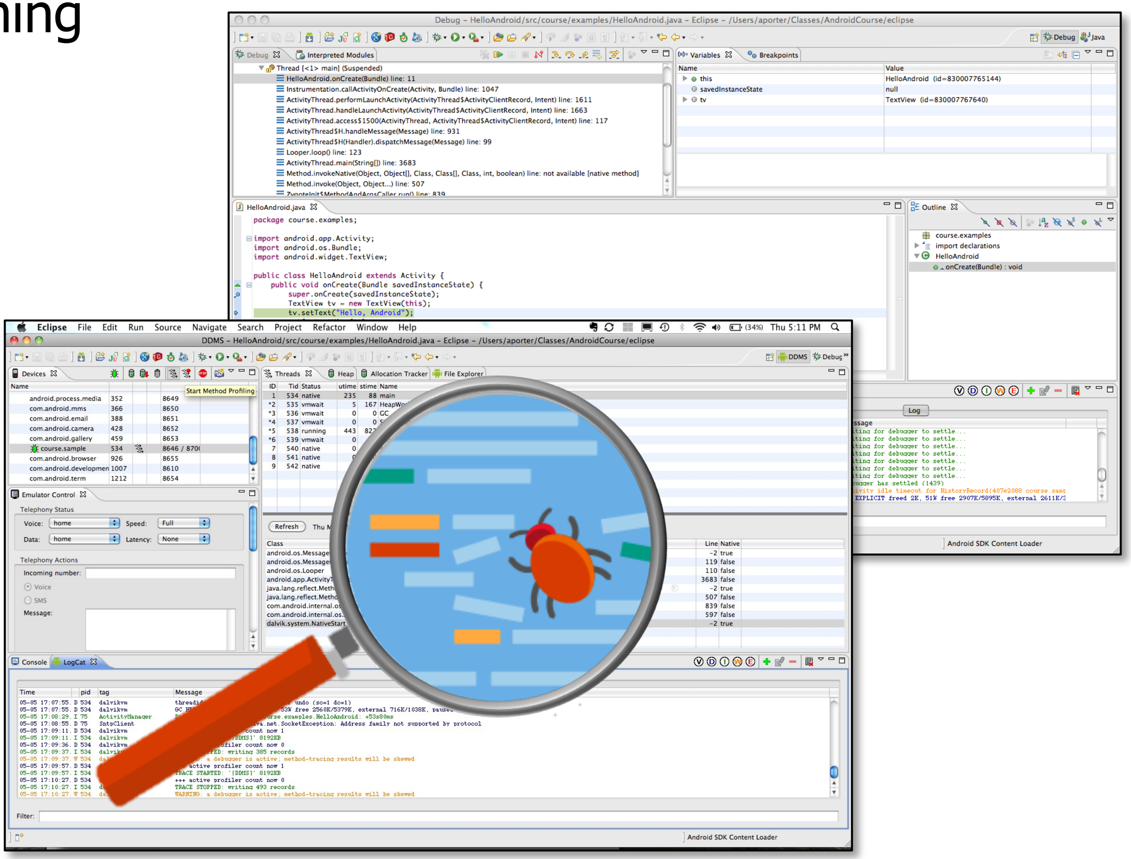
# Evaluating the Pros & Cons of Sequential Programming

# Evaluating the Pros & Cons of Sequential Programming

- Pros of sequential programming

# Evaluating the Pros & Cons of Sequential Programming

- Pros of sequential programming
  - Easy to program & debug


Debug – HelloAndroid/src/course/examples/HelloAndroid.java – Eclipse – /Users/aporter/Classes/AndroidCourse/eclipse
Debug
Interpreted Modules
Variables
Breakpoints
Outline
Eclipse File Edit Run Source Navigate Search Project Refactor Window Help
Thu 5:11 PM
DDMS – HelloAndroid/src/course/examples/HelloAndroid.java – Eclipse – /Users/aporter/Classes/AndroidCourse/eclipse
DDMS
Debug
Devices
Start Method Profiling
Threads
Heap
Allocation Tracker
File Explorer
Emulator Control
Console
LogCat
Refresh
Android SDK Content Loader

```
Thread [<1> main] (Suspended)
    HelloAndroid.onCreate(Bundle) line: 11
    Instrumentation.callActivityOnCreate(Activity, Bundle) line: 1047
    ActivityThread.performLaunchActivity(ActivityThread$ActivityClientRecord, Intent) line: 1611
    ActivityThread.handleLaunchActivity(ActivityThread$ActivityClientRecord, Intent) line: 1663
    ActivityThread.access$1500(ActivityThread, ActivityThread$ActivityClientRecord, Intent) line: 117
    ActivityThread$H.handleMessage(Message) line: 931
    ActivityThread$H(Handler).dispatchMessage(Message) line: 99
    Looper.loop() line: 123
    ActivityThread.main(String[]) line: 3683
    Method.invokeNative(Object, Object[], Class, Class[], Class, int, boolean) line: not available [native method]
    Method.invoke(Object, Object...) line: 507
    ZygoteInit$MethodAndArgsCaller.run() line: 839
```

| Name | Value |
|---|---|
| this | HelloAndroid (id=830007765144) |
| savedInstanceState | null |
| tv | TextView (id=830007767640) |

HelloAndroid.java

```
package course.examples;

import android.app.Activity;
import android.os.Bundle;
import android.widget.TextView;

public class HelloAndroid extends Activity {
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        TextView tv = new TextView(this);
        tv.setText("Hello, Android");
```

Outline: course.examples; import declarations; HelloAndroid; onCreate(Bundle) : void

| Name | | | | |
|---|---|---|---|---|
| android.process.media | 352 | | | 8649 |
| com.android.mms | 366 | | | 8650 |
| com.android.email | 388 | | | 8651 |
| com.android.camera | 428 | | | 8652 |
| com.android.gallery | 459 | | | 8653 |
| course.sample | 534 | | | 8646 / 870( |
| com.android.browser | 926 | | | 8655 |
| com.android.developmen | 1007 | | | 8610 |
| com.android.term | 1212 | | | 8654 |

| ID | Tid | Status | utime | stime | Name |
|---|---|---|---|---|---|
| 1 | 534 | native | 235 | 88 | main |
| *2 | 535 | vmwait | 5 | 167 | HeapWo |
| *3 | 536 | vmwait | 0 | 0 | GC |
| *4 | 537 | vmwait | 0 | 0 | |
| *5 | 538 | running | 443 | 82 | |
| *6 | 539 | vmwait | 0 | | |
| 7 | 540 | native | 0 | | |
| 8 | 541 | native | | | |
| 9 | 542 | native | | | |

| Class | Line | Native |
|---|---|---|
| android.os.Message | -2 | true |
| android.os.Message | 119 | false |
| android.os.Looper | 110 | false |
| android.app.ActivityT | 3683 | false |
| java.lang.reflect.Meth | -2 | true |
| java.lang.reflect.Metho | 507 | false |
| com.android.internal.os | 839 | false |
| com.android.internal.os. | 597 | false |
| dalvik.system.NativeStart | -2 | true |

Emulator Control

Telephony Status

Voice: home Speed: Full

Data: home Latency: None

Telephony Actions

Incoming number:

○ Voice

○ SMS

Message:

```
Time                pid  tag              Message
05-05 17:07:55. D 534  dalvikvm         threadid                    undo (sc=1 dc=1)
05-05 17:07:55. D 534  dalvikvm         GC HE                  53% free 2560K/5379K, external 716K/1038K, paused
05-05 17:08:29. I 75   ActivityManager  Di                  urse.examples.HelloAndroid: +53s80ms
05-05 17:08:55. D 75   SntpClient                              va.net.SocketException: Address family not supported by protocol
05-05 17:09:11. D 534  dalvikvm                          count now 1
05-05 17:09:11. I 534  dalvikvm                          [DDMS]' 8192KB
05-05 17:09:36. D 534  dalvikvm                         rofiler count now 0
05-05 17:09:37. I 534  dalvik                           PPED: writing 385 records
05-05 17:09:37. W 534  dal                              : a debugger is active; method-tracing results will be skewed
05-05 17:09:57. D 534                                   active profiler count now 1
05-05 17:09:57. I 534                                  TRACE STARTED: '[DDMS]' 8192KB
05-05 17:10:27. D 534                                  +++ active profiler count now 0
05-05 17:10:27. I 534  da                              TRACE STOPPED: writing 493 records
05-05 17:10:27. W 534  da                              WARNING: a debugger is active; method-tracing results will be skewed
```

Filter:

# Evaluating the Pros & Cons of Sequential Programming

- Pros of sequential programming
  - Easy to program & debug
    - "Intuitive" since it matches the steps expressed in algorithms

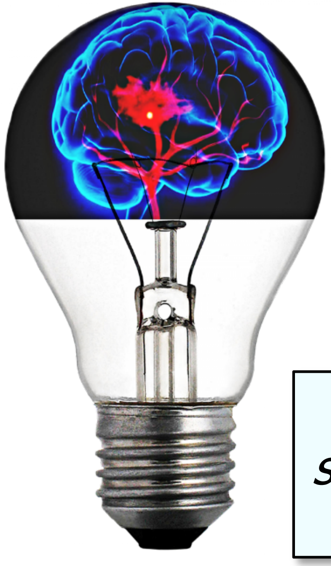
```
int i, j, len = ...;

for (i = 0;
     i < len - 1;
     i++) {
    int min = i;

    for (j = i + 1;
         j < len;
         j++)
      if (a[j] < a[min])
        min = j;

    if (min != i)
      swap(a[i], a[min]);
```

| |
|---|
| 8 |
| 5 |
| 2 |
| 6 |
| 9 |
| 3 |
| 1 |
| 4 |
| 0 |
| 7 |



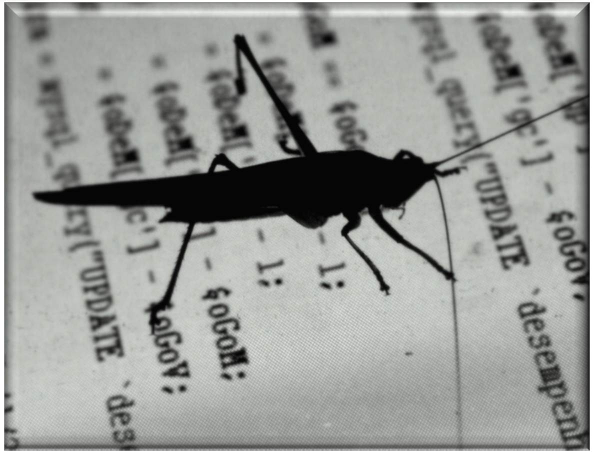
*This algorithm can be understood by reading it as written, i.e., there are no "surprises"*

See jeremymanson.blogspot.com/2007/08/atomicity-visibility-and-ordering.html

# Evaluating the Pros & Cons of Sequential Programming

- Pros of sequential programming
  - Easy to program & debug
    - "Intuitive" since it matches the steps expressed in algorithms
    - The behavior in the debugger reflects actual program behavior

**vs.**



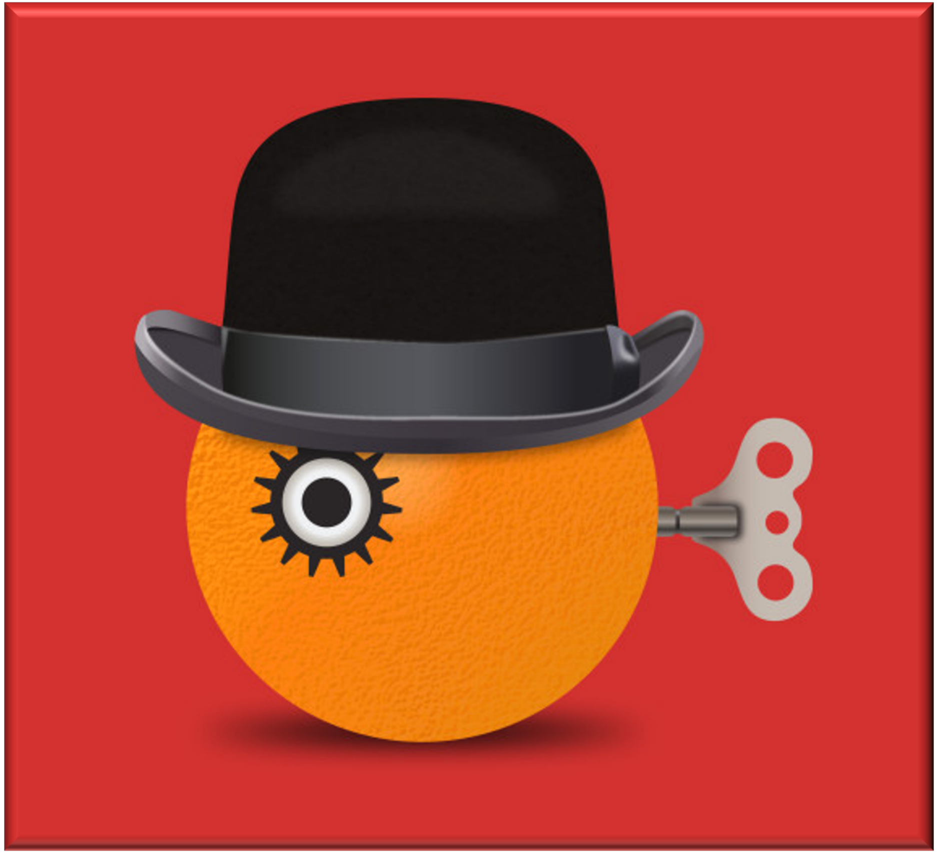
# Evaluating the Pros & Cons of Sequential Programming

- Pros of sequential programming
  - Easy to program & debug
    - "Intuitive" since it matches the steps expressed in algorithms
    - The behavior in the debugger reflects actual program behavior
      - Conversely, the behavior of non-sequential programs often differ when run in a debugger vs. "in the wild"

*These differences stem from perturbations in timing from the different execution contexts*

# Evaluating the Pros & Cons of Sequential Programming

- Pros of sequential programming
  - Easy to program & debug
  - Deterministic execution order simplifies reasoning about & assuring program behavior



See screenprism.com/insights/article/what-is-the-ludovico-technique-and-how-does-it-work

# Evaluating the Pros & Cons of Sequential Programming

- Pros of sequential programming
  - Easy to program & debug
  - Deterministic execution order simplifies reasoning about & assuring program behavior
    - Especially for safety-critical cyber-physical systems

See en.wikipedia.org/wiki/Cyber-physical_system

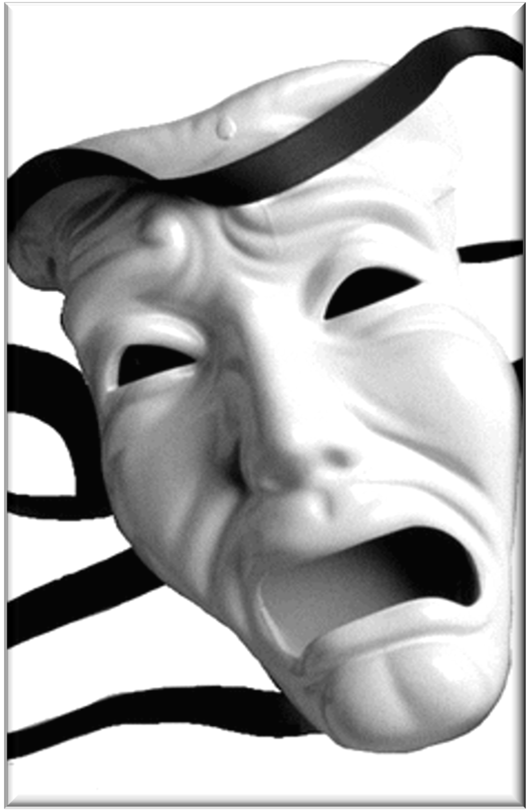
# Evaluating the Pros & Cons of Sequential Programming

- Pros of sequential programming
  - Easy to program & debug
  - Deterministic execution order simplifies reasoning about & assuring program behavior
    - Especially for safety-critical cyber-physical systems

*The right answer delivered too late becomes the wrong answer*

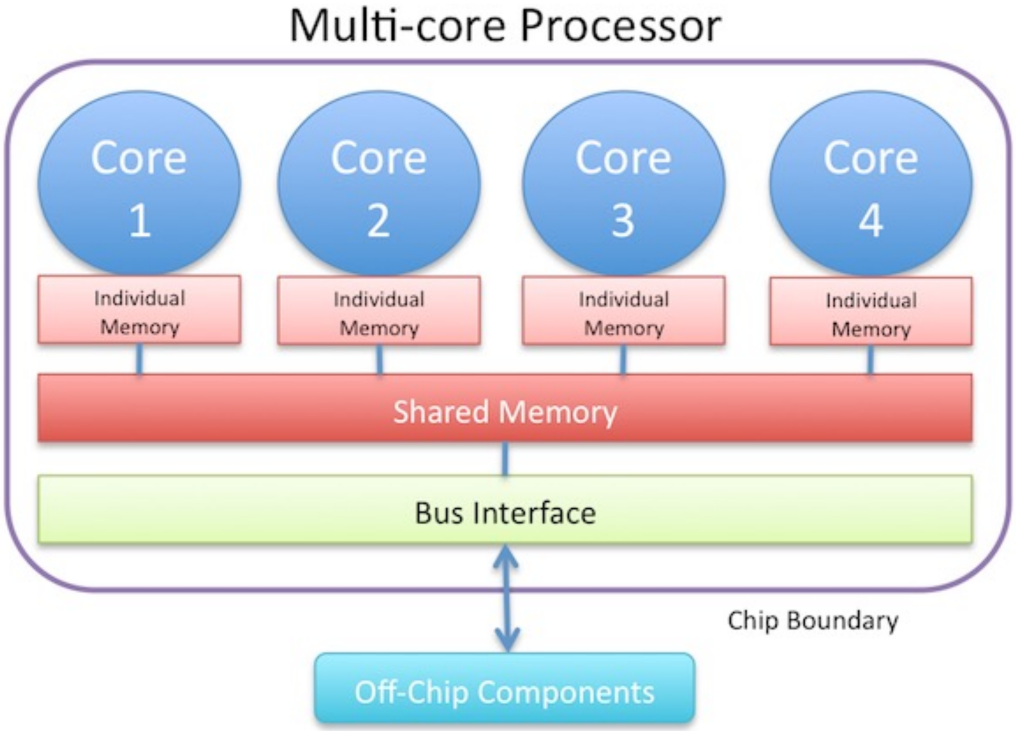
# Evaluating the Pros & Cons of Sequential Programming

- Cons of sequential programming



# Evaluating the Pros & Cons of Sequential Programming

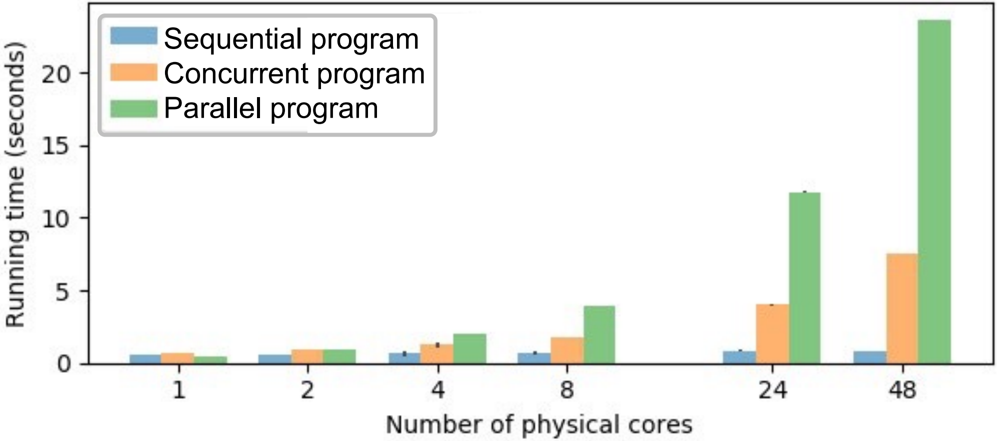
- Cons of sequential programming
  - Cannot leverage the parallelism available in multi-core systems

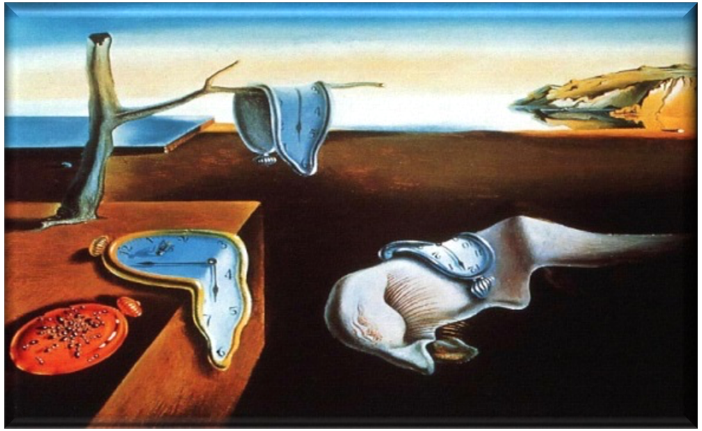


See en.wikipedia.org/wiki/Multi-core_processor

# Evaluating the Pros & Cons of Sequential Programming

- Cons of sequential programming
  - Cannot leverage the parallelism available in multi-core systems
    - Performance may therefore suffer relative to concurrent & parallel programs





# Evaluating the Pros & Cons of Sequential Programming

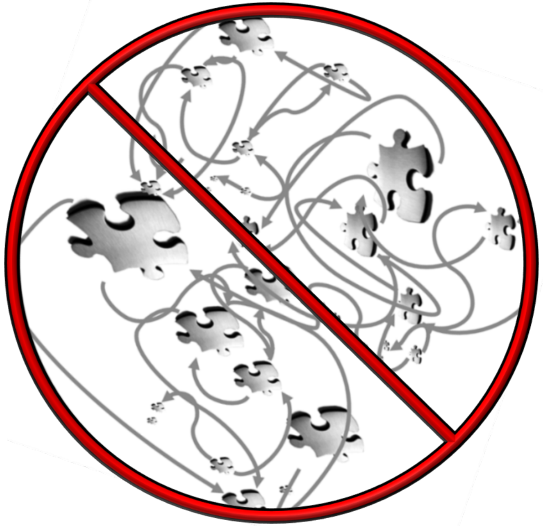
- Cons of sequential programming
  - Cannot leverage the parallelism available in multi-core systems
  - It's hard to be responsive to multiple I/O sources/sinks

*e.g., mouse movement/clicks, touch events, GPS location signals, network connections, asynchronous storage read & write completions, etc.*

See en.wikipedia.org/wiki/Responsiveness

# Evaluating the Pros & Cons of Sequential Programming

- Cons of sequential programming
  - Cannot leverage the parallelism available in multi-core systems
  - It's hard to be responsive to multiple I/O sources/sinks



*Having only a single thread of control complicates the structure of sequential programs for blocking operations*

See en.wikipedia.org/wiki/Event-driven_programming

# Evaluating the Pros & Cons of Sequential Programming

- Cons of sequential programming
  - Cannot leverage the parallelism available in multi-core systems
  - It's hard to be responsive to multiple I/O sources/sinks

*Overcoming these 'cons' motivates all of the concurrency & parallelism topics that we cover henceforth!!!*

# End of Overview of Sequential Programming Concepts