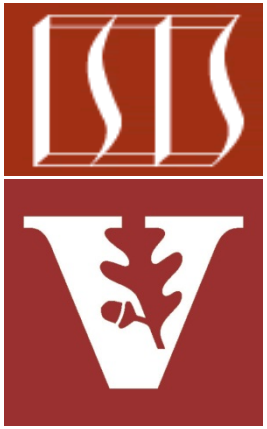
Course Overview: Part 1

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



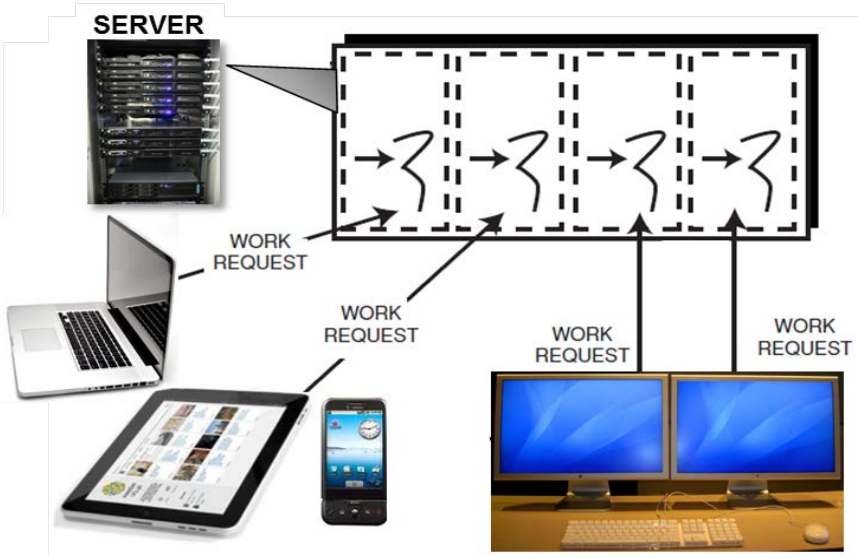
Professor of Computer Science

Institute for Software
Integrated Systems

Vanderbilt University
Nashville, Tennessee, USA

Topics Covered in this Part of the Module

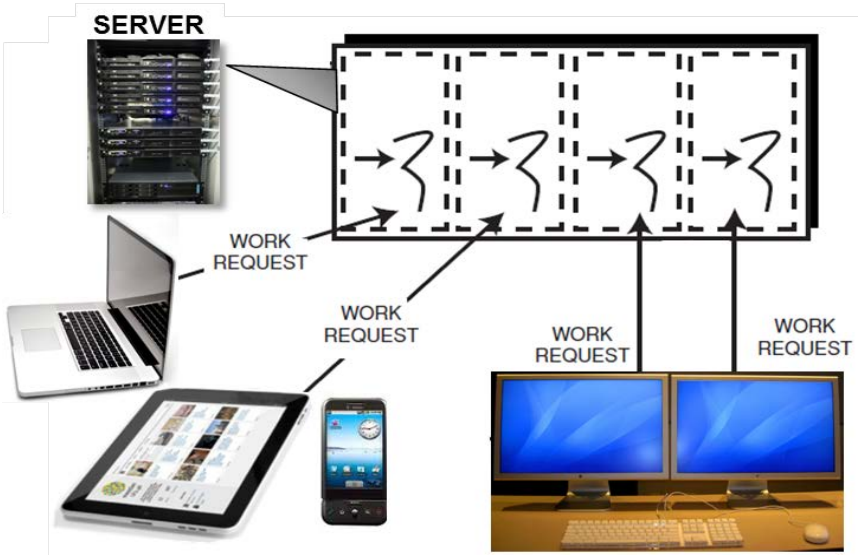
- Explore the motivations for & challenges of concurrent & networked software

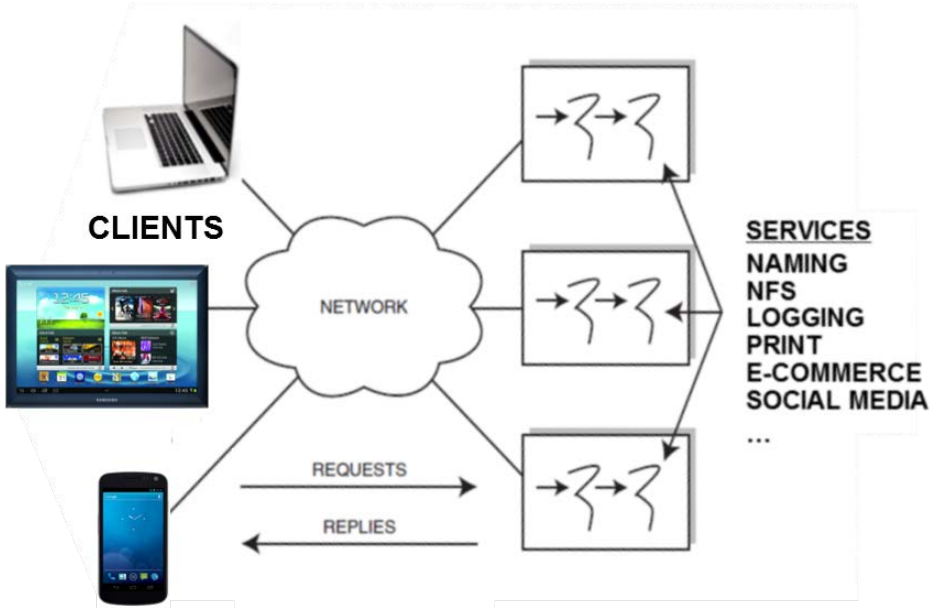


  - *Concurrent software* can simultaneously run multiple computations that potentially interact with each other

Topics Covered in this Part of the Module

- Explore the motivations for & challenges of concurrent & networked software



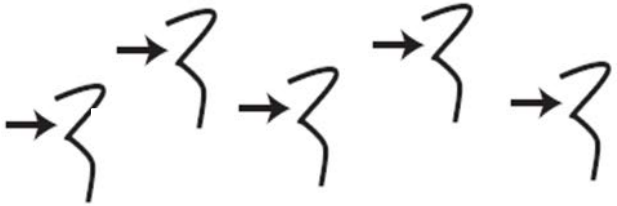


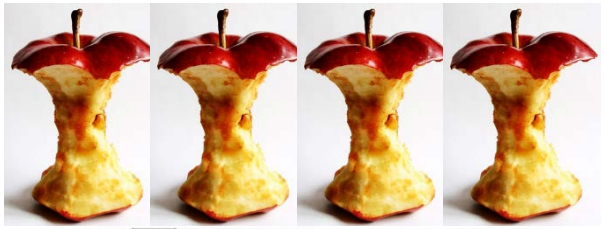
- *Concurrent software* can simultaneously run multiple computations that potentially interact with each other

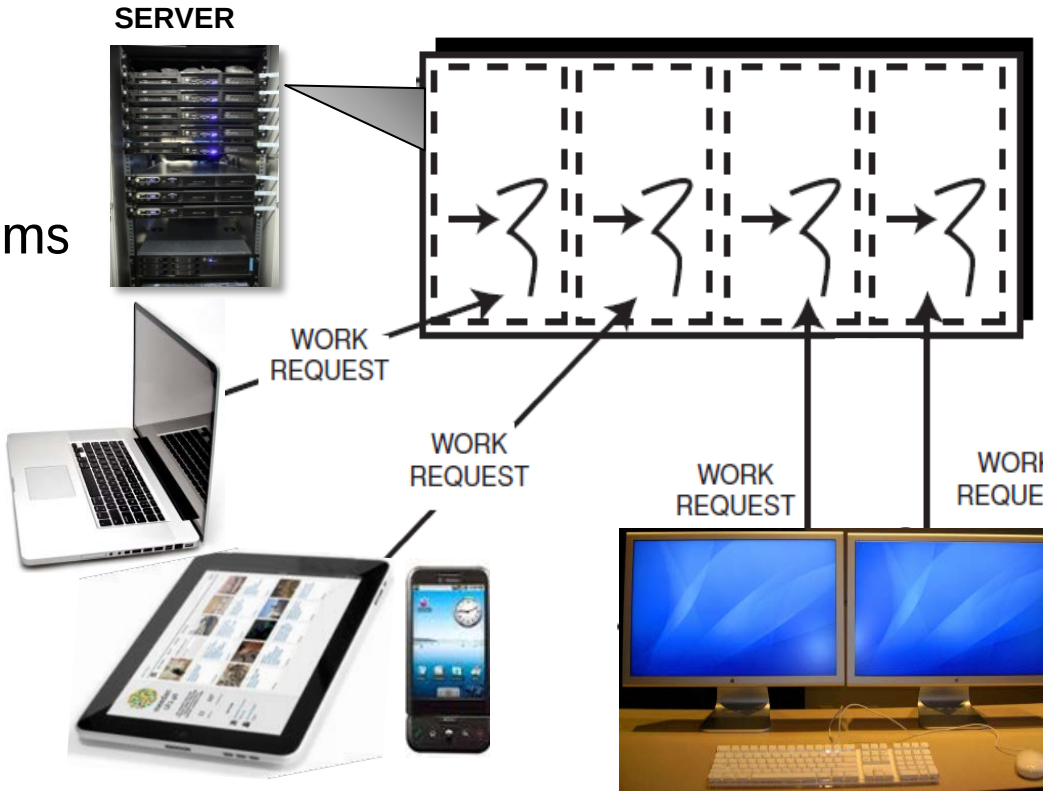
- *Networked software* defines protocols that enable computing devices to exchange messages & perform services remotely

Motivations for Concurrent Software

- Leverage hardware/software advances
  - e.g., multi-core processors & multi-threaded operating systems

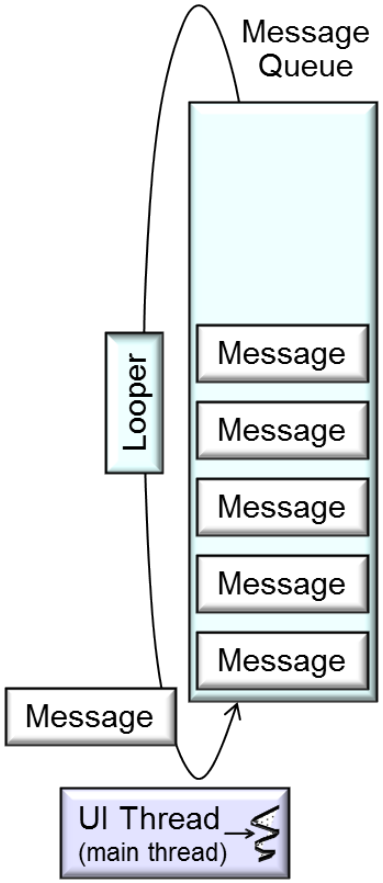


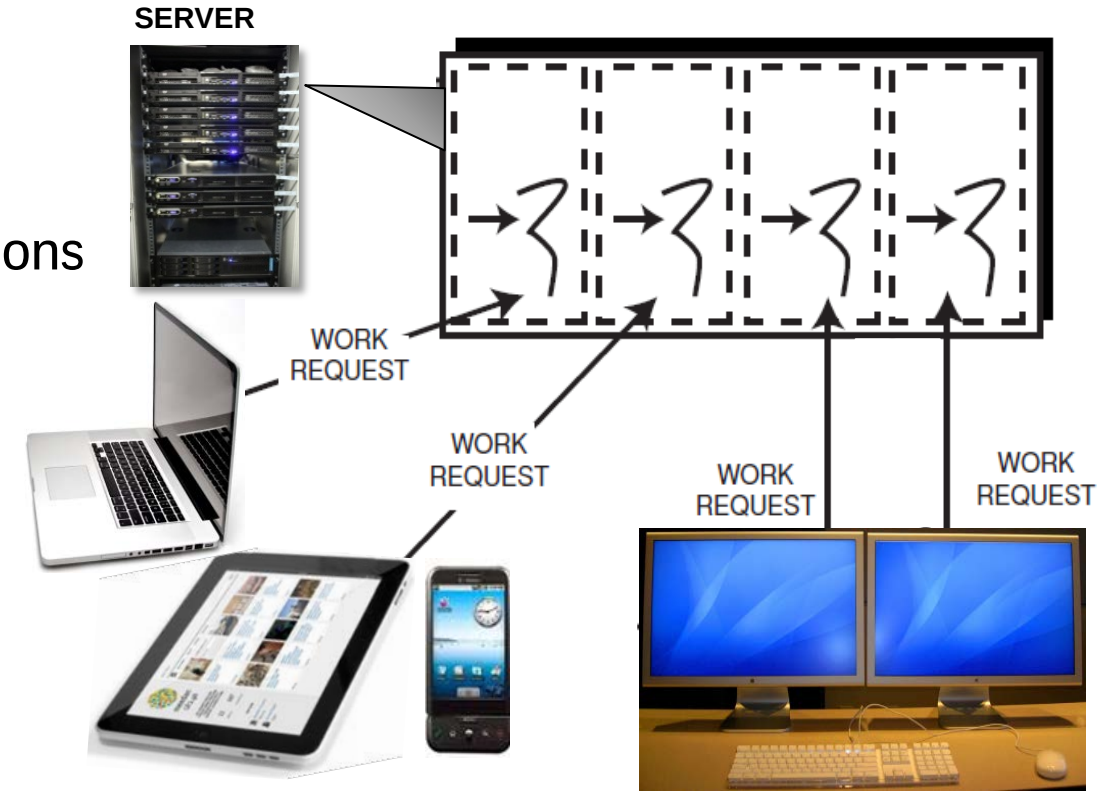




Motivations for Concurrent Software

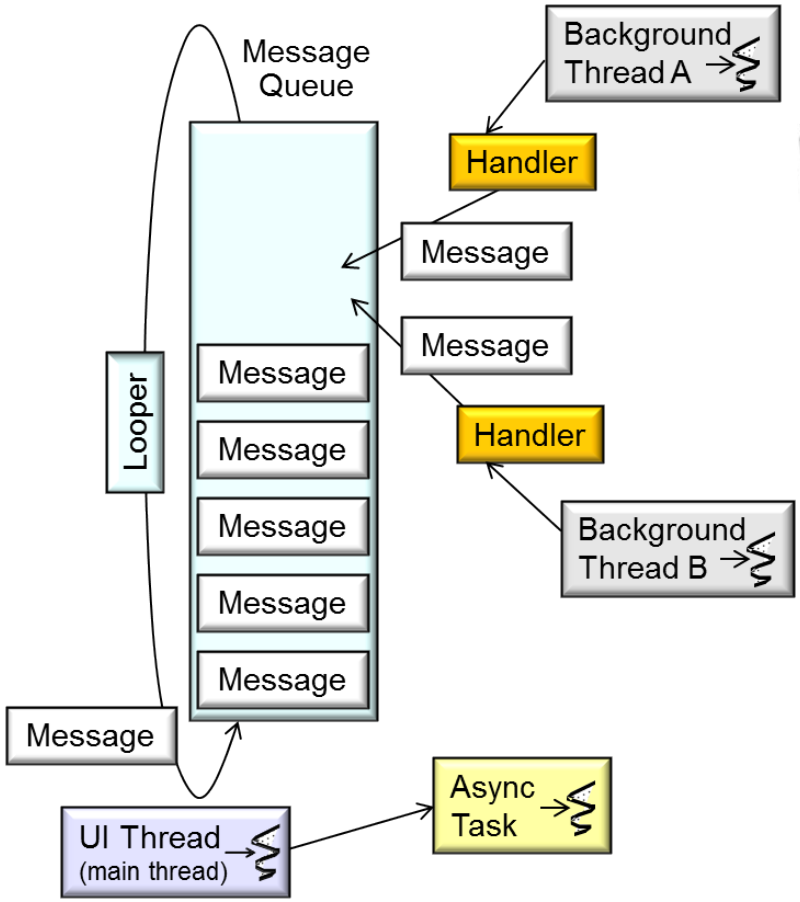
- Leverage hardware/software advances
- Simplify program structure
  - e.g., by allow blocking operations

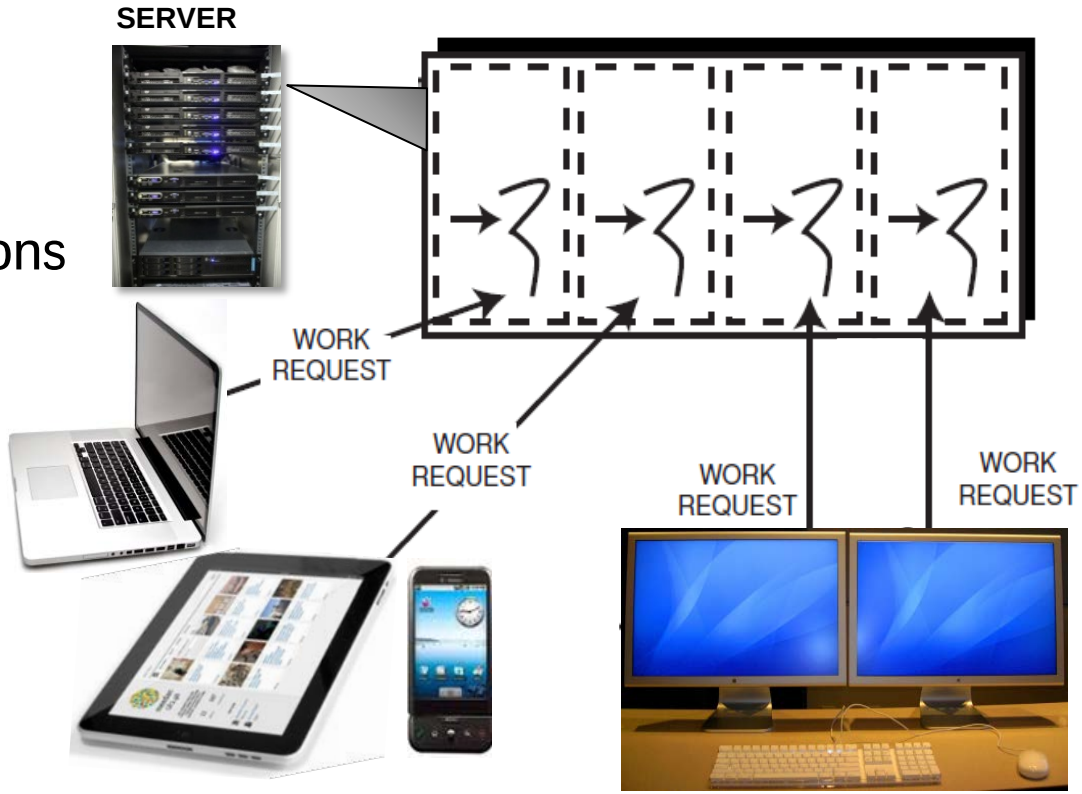




Motivations for Concurrent Software

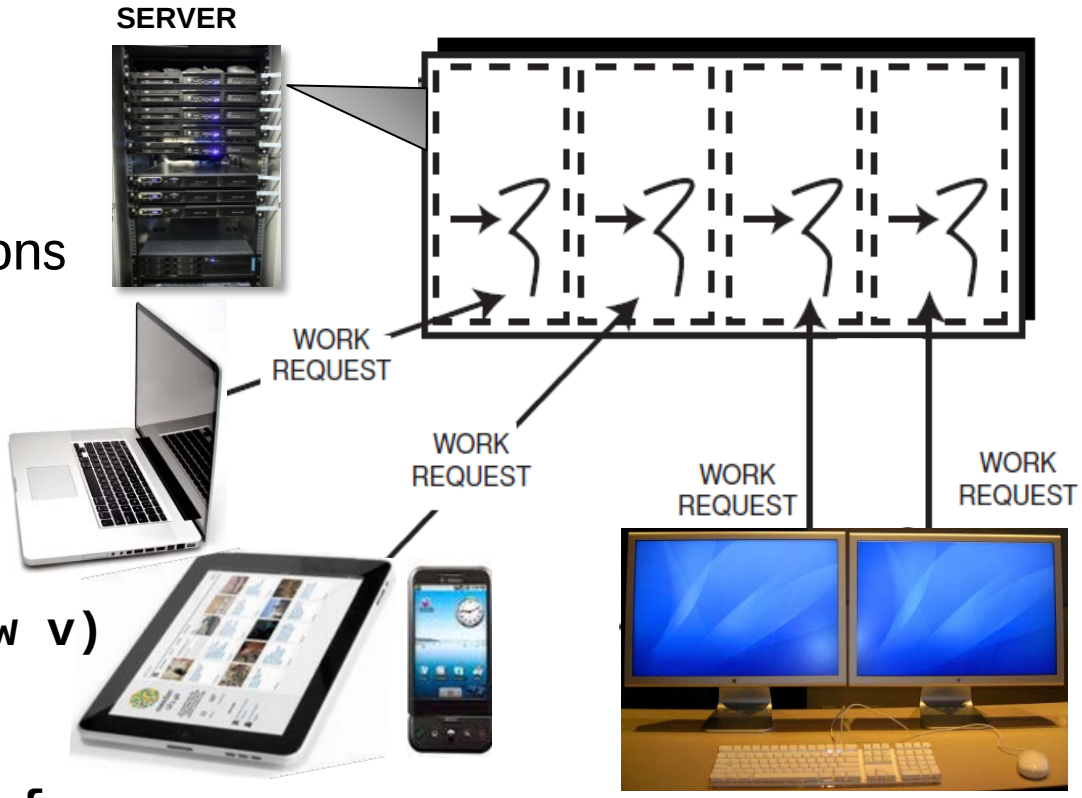
- Leverage hardware/software advances
- Simplify program structure
  - e.g., by allow blocking operations





Motivations for Concurrent Software

- Leverage hardware/software advances
- Simplify program structure
  - e.g., by allow blocking operations




```
private Bitmap bitmap;
final ImageView iview = ...
final Button button = ...
button.setOnClickListener
  (new OnClickListener() {
    public void onClick(View v)
      { new Thread(
          new Runnable() {
          public void run() {
            bitmap = downloadImage(URI);
            iview.post(new Runnable() {
              public void run() { iview.setImageBitmap(bitmap);}
            });
          }
      }).start();
```

Handles button clicks

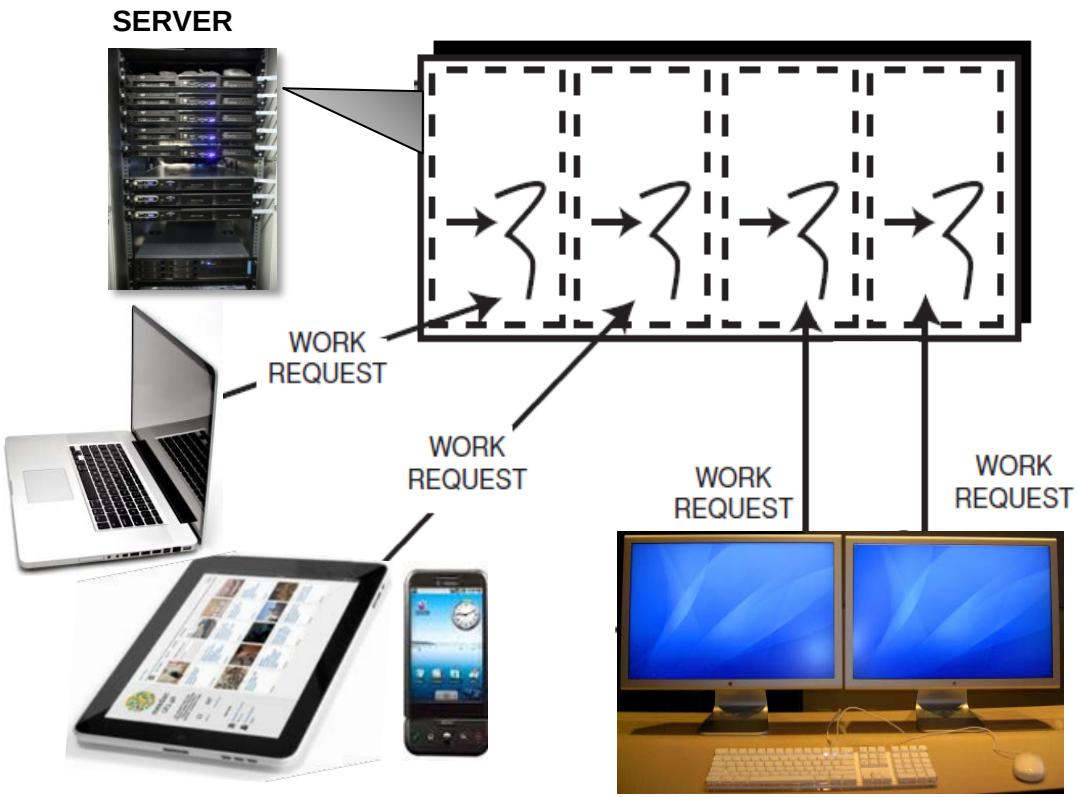
Download an image

Display bitmap in the UI thread

Start a new thread

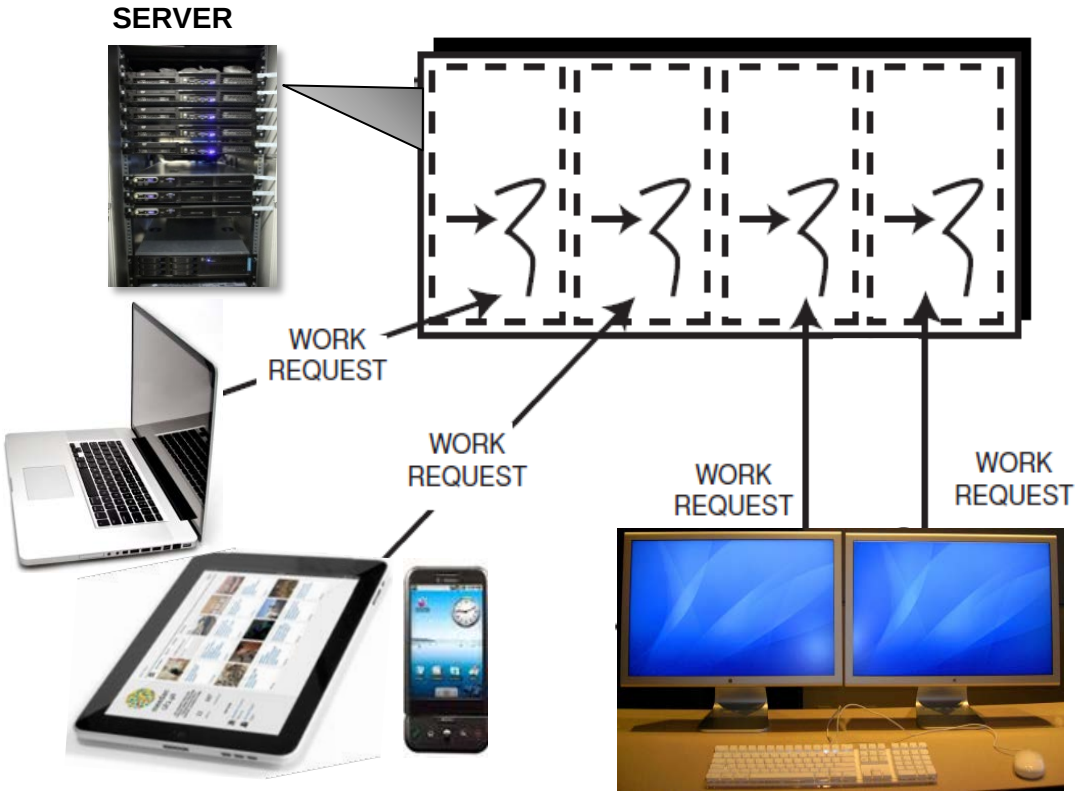
Motivations for Concurrent Software

- Leverage hardware/software advances
- Simplify program structure
- Increase performance
  - Parallelize computations & communications



Motivations for Concurrent Software

- Leverage hardware/software advances
- Simplify program structure
- Increase performance
- Improve response-time
  - e.g., don't starve the UI thread



Challenges for Concurrent Software

- *Accidental Complexities*

Stem from limitations with development tools & techniques

Challenges for Concurrent Software

- *Accidental Complexities*
  - Low-level APIs
    - Tedious, error-prone, & non-portable

Challenges for Concurrent Software

- *Accidental Complexities*
  - Low-level APIs

```
typedef struct
{ char message_[20]; int thread_id_; } PARAMS;

void *print_hello_world (void *ptr) {
  PARAMS *params = (PARAMS *) ptr;
  printf ("%s from thread %d\n",
          params->message_, params->thread_id_);
}

int main (void) {
  pthread_t thread; PARAMS params;
  params.thread_id_ = 1; strcpy (params.message_, "Hello World");

  pthread_create (&thread, 0, &print_hello_world,
                  (void *) &params);
  /* ... */
  pthread_join(thread, 0);
  return 0;
}
```

Cast from void *

Not portable to non-POSIX platforms

Pointer-to-function

Cast to void *

"Quasi-typed" thread handle

Challenges for Concurrent Software

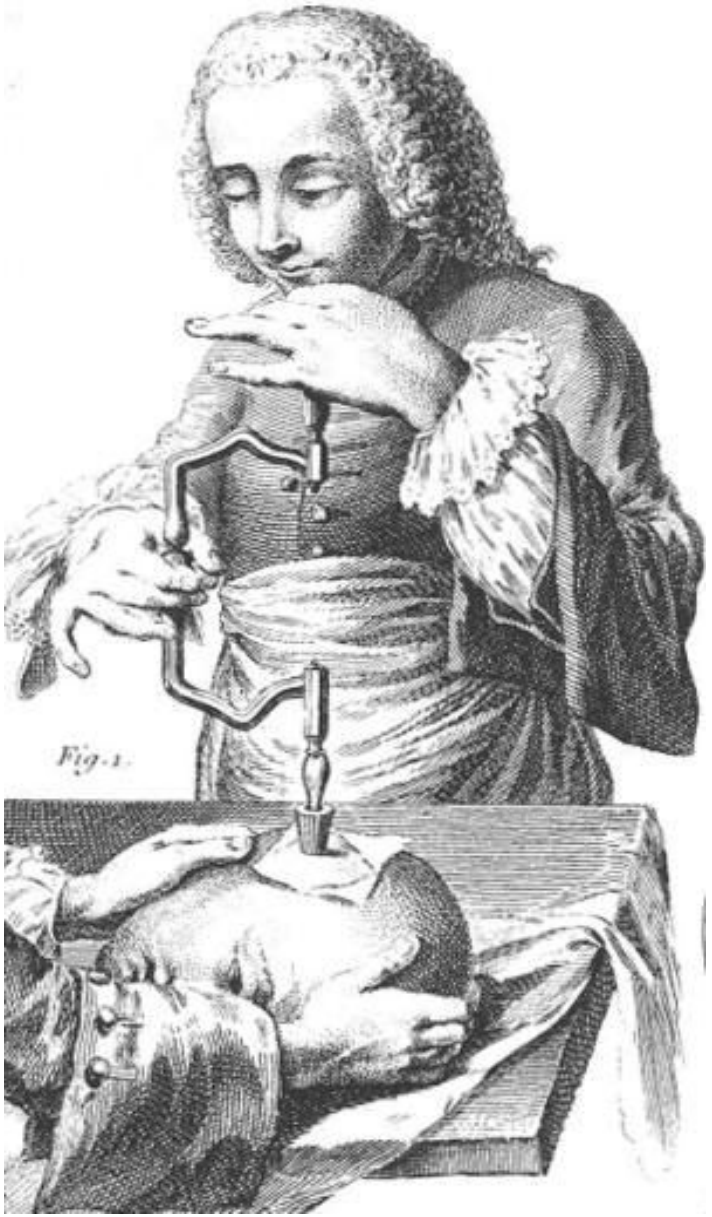
- *Accidental Complexities*
  - Low-level APIs

```
typedef struct
{ char message_[20]; int thread_id_; } PARAMS;

void *print_hello_world (void *ptr) {
  PARAMS *params = (PARAMS *) ptr;
  printf ("%s from thread %d\n",
          params->message_, params->thread_id_);
}

int main (void) {
  pthread_t thread; PARAMS params;
  params.thread_id_ = 1; strcpy (params.message_, "Hello World");


  pthread_create (&thread, 0, &print_hello_world,
                  (void *) &params);
  /* ... */
  pthread_join(thread, 0);
  return 0;
}
```

Other C threading APIs have similar accidental complexities

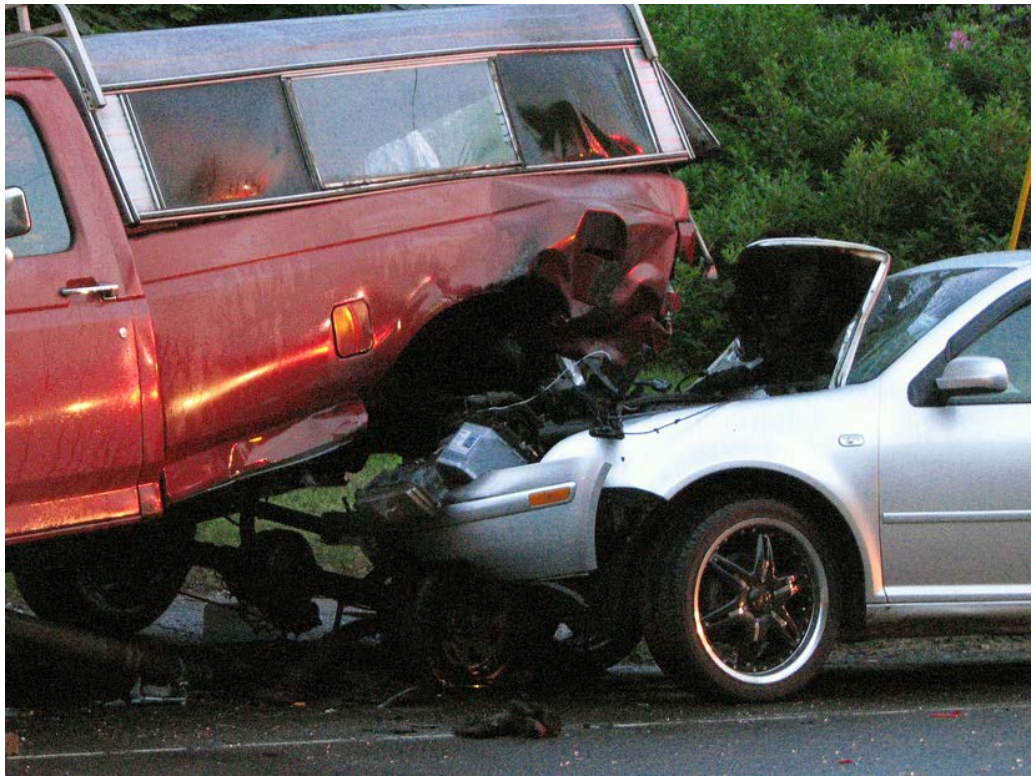
Challenges for Concurrent Software

- *Accidental Complexities*
  - Low-level APIs
  - Limited debugging tools



Challenges for Concurrent Software

- *Accidental Complexities*
  - Low-level APIs
  - Limited debugging tools



See www.dre.vanderbilt.edu/~schmidt/PDF/DSIS.pdf & www.fluid.cs.cmu.edu

Challenges for Concurrent Software

- *Accidental Complexities*
  - Low-level APIs
  - Limited debugging tools
- *Inherent Complexities*

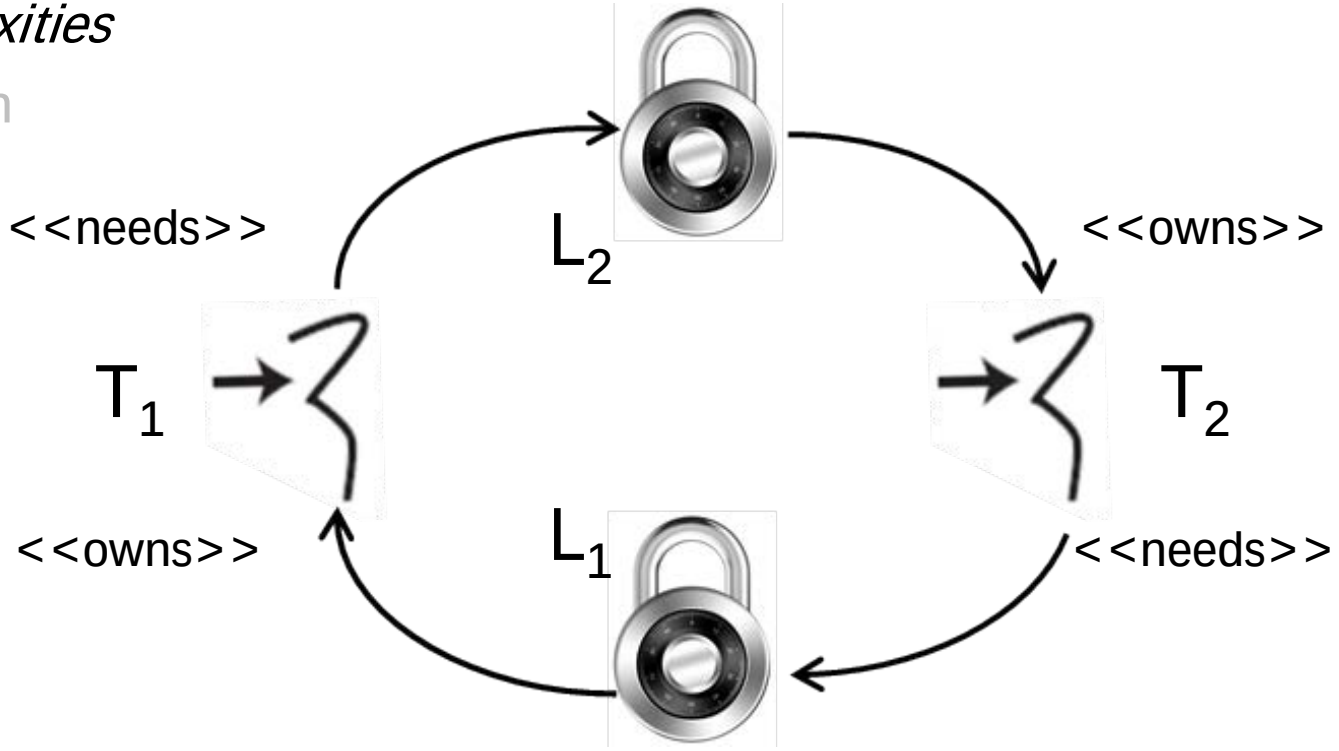
Stem from fundamental domain challenges

Challenges for Concurrent Software

- *Accidental Complexities*
  - Low-level APIs
  - Limited debugging tools
- *Inherent Complexities*
  - Synchronization
  - Scheduling

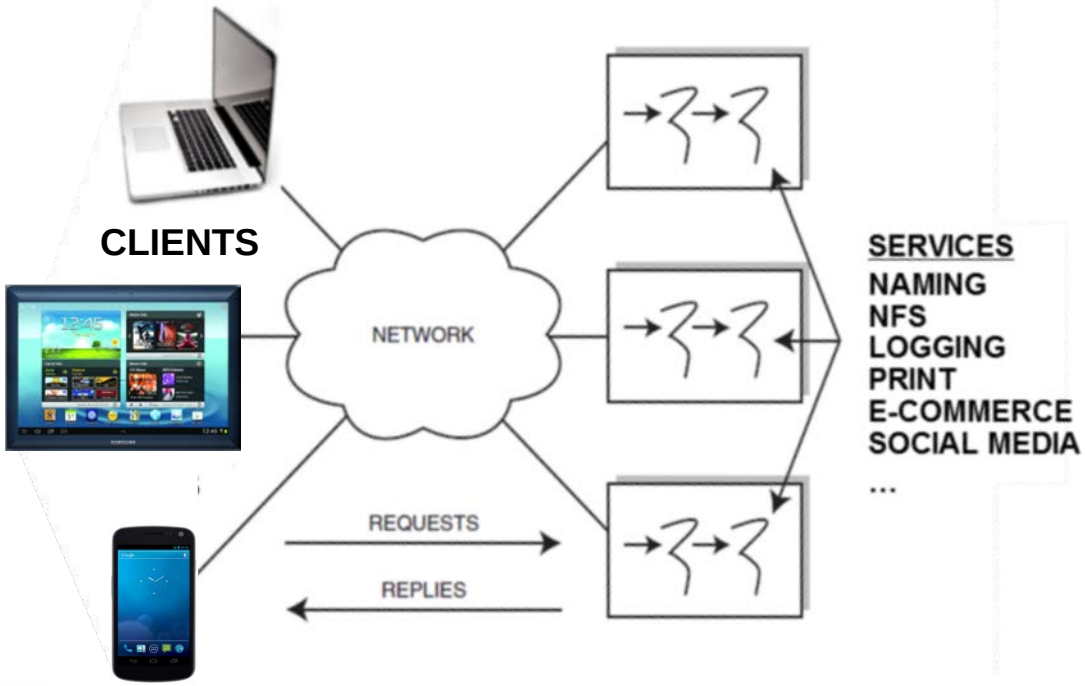
Challenges for Concurrent Software

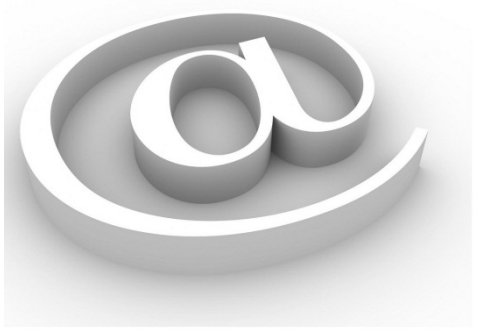
- *Accidental Complexities*
  - Low-level APIs
  - Limited debugging tools
- *Inherent Complexities*
  - Synchronization
  - Scheduling
  - Deadlocks



Motivations for Networked Software

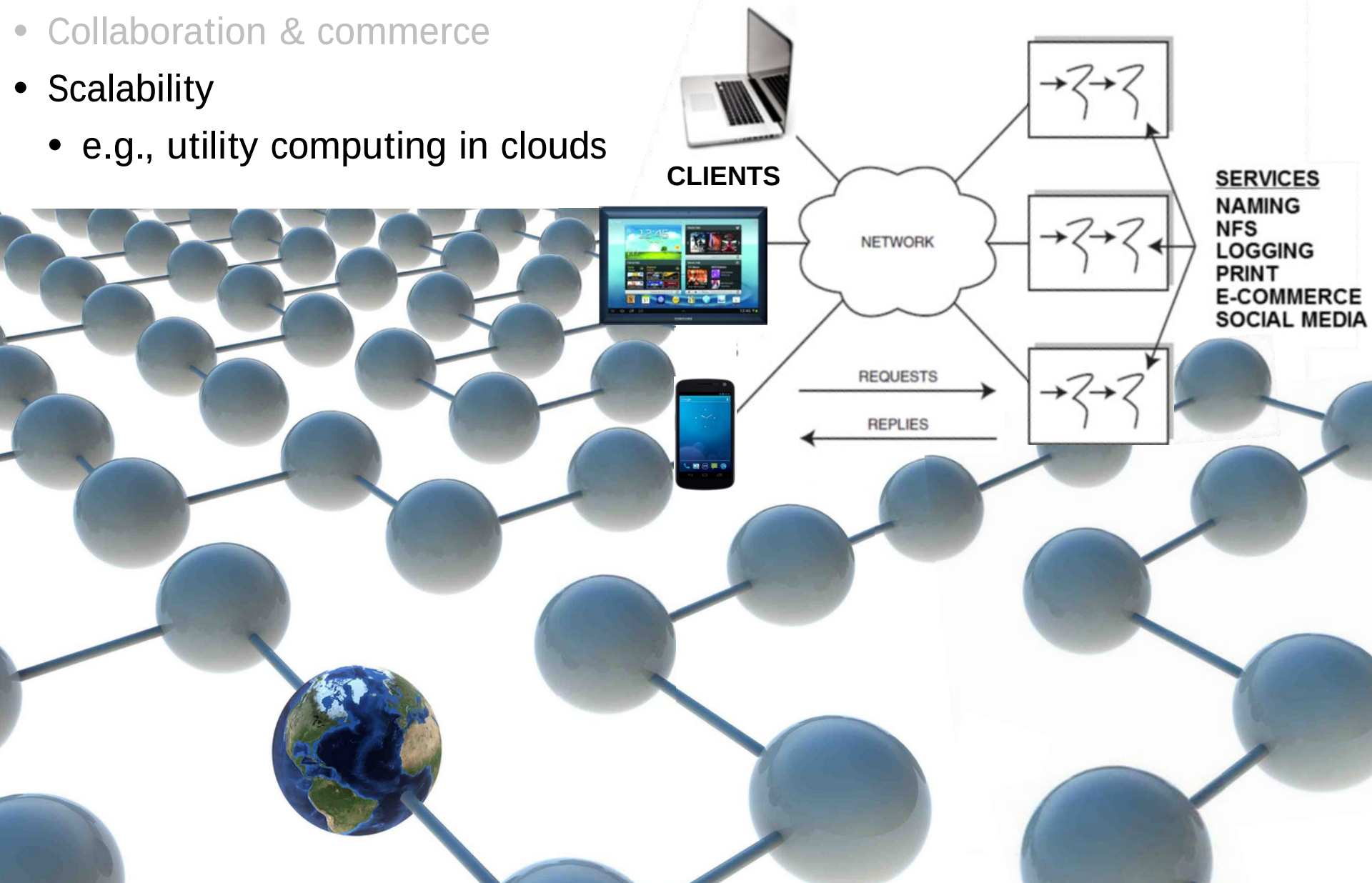
- Collaboration & commerce
  - e.g., file sharing, social media, e-commerce online transaction processing, B2B supply chain management, etc.





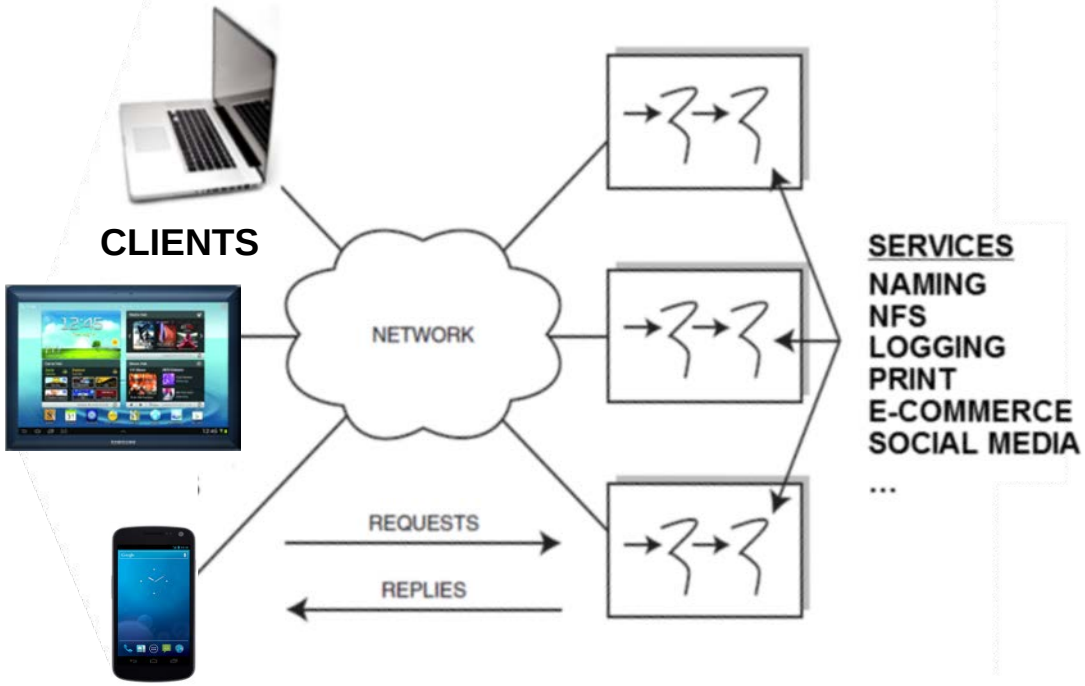
Motivations for Networked Software

- Collaboration & commerce
- Scalability
  - e.g., utility computing in clouds



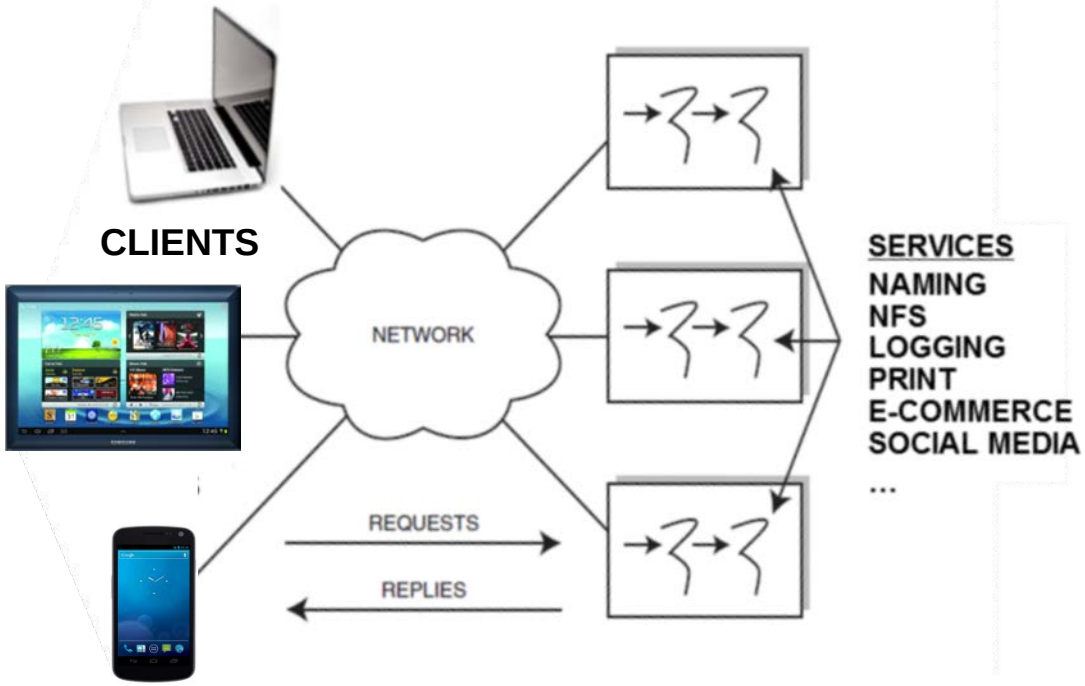
Motivations for Networked Software

- Collaboration & commerce
- Scalability
- Availability
  - e.g., minimizing single points of failure via replication



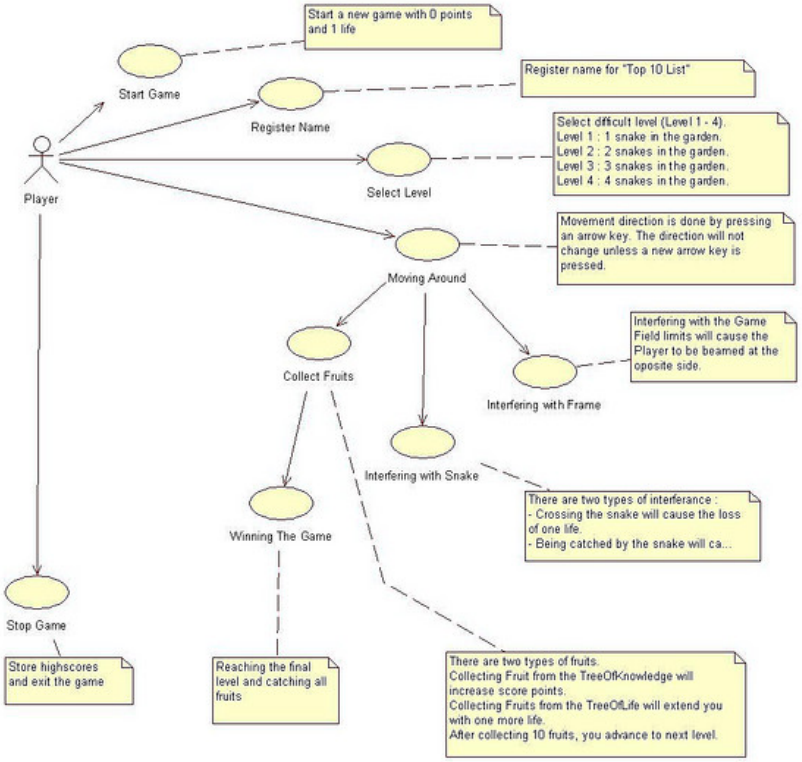
Motivations for Networked Software

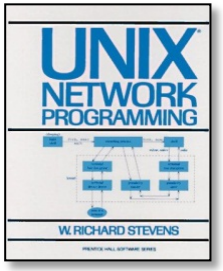
- Collaboration & commerce
- Scalability
- Availability
- Cost effectiveness
  - e.g., via shared resources

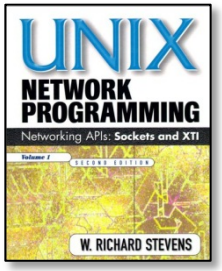


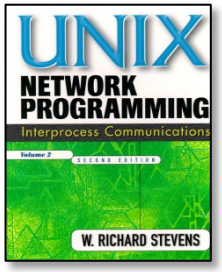
Challenges for Networked Software

- *Accidental Complexities*
  - Algorithmic decomposition









Challenges for Networked Software

- *Accidental Complexities*
  - Algorithmic decomposition
  - Continuous re-discovery & re-invention of core components

See steve.vinoski.net/pdf/IEEE-Middleware_Dark_Matter.pdf for more

Challenges for Networked Software

- *Accidental Complexities*
  - Algorithmic decomposition
  - Continuous re-discovery & re-invention of core components
- *Inherent Complexities*
  - Latency & jitter

Challenges for Networked Software

- *Accidental Complexities*
  - Algorithmic decomposition
  - Continuous re-discovery & re-invention of core components
- *Inherent Complexities*
  - Latency & jitter
  - Reliability & partial failure

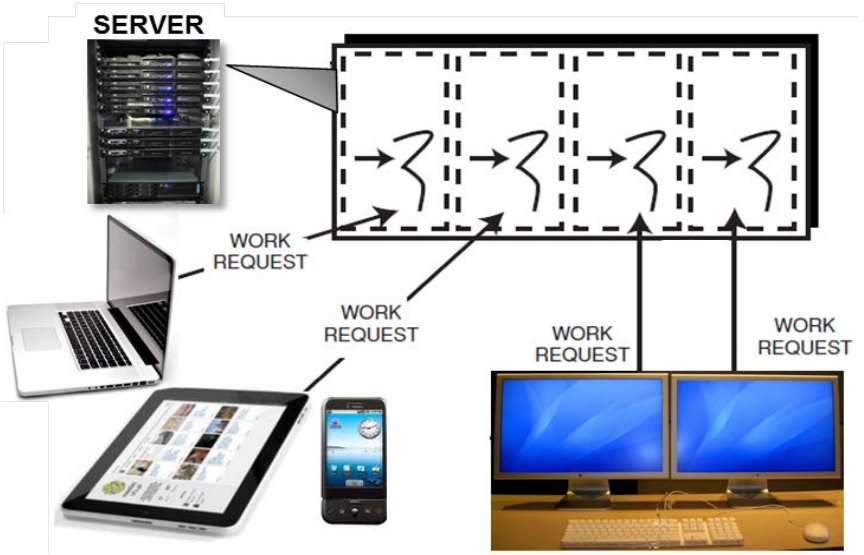
Challenges for Networked Software

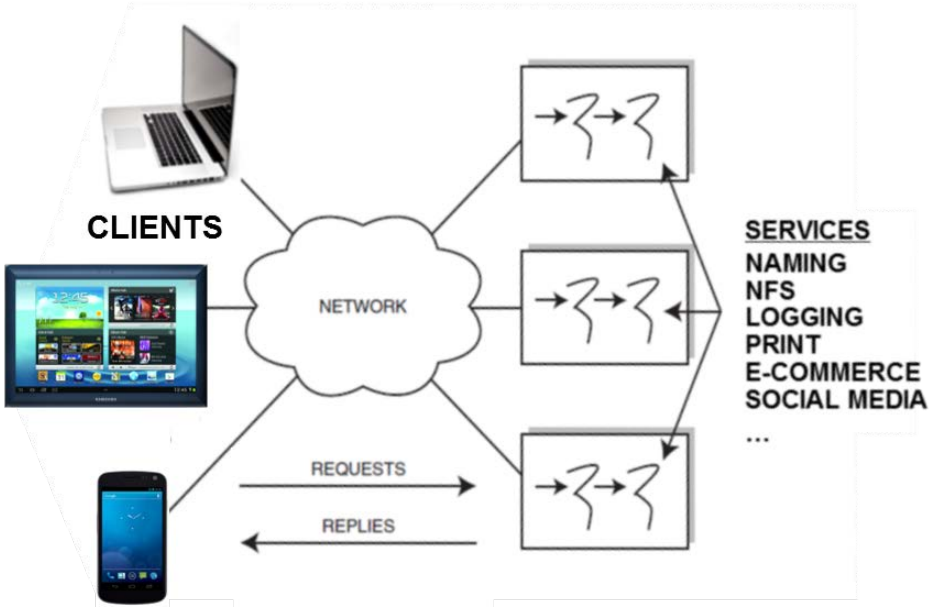
- *Accidental Complexities*
  - Algorithmic decomposition
  - Continuous re-discovery & re-invention of core components
- *Inherent Complexities*
  - Latency & jitter
  - Reliability & partial failure
  - Security

See www.dre.vanderbilt.edu/~schmidt/PDF/pdcp-editorial.pdf for more info

Summary

- Concurrent & networked software helps
  - Leverage advances in hardware & networking technology
  - Meet the quality & performance needs of apps & services



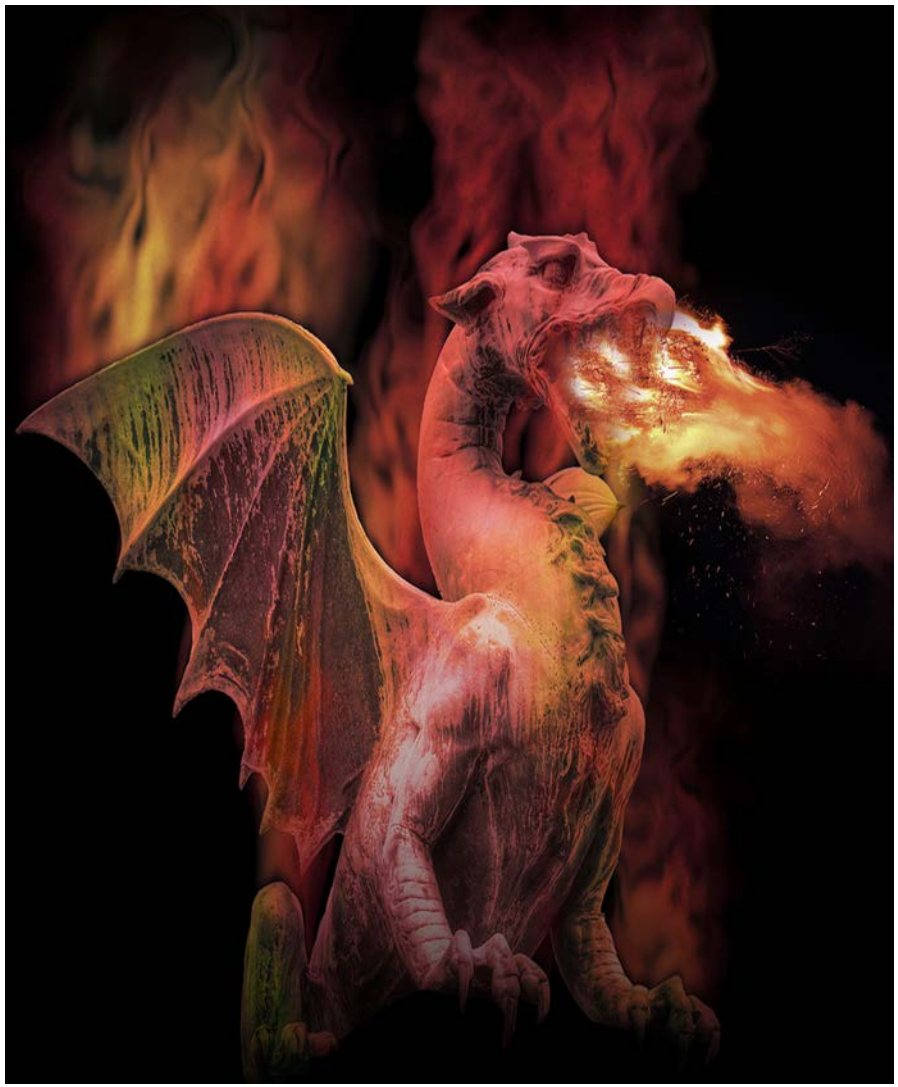


Summary

- Concurrent & networked software helps
  - Leverage advances in hardware technology
  - Meet the quality & performance needs of apps & services
- Successful concurrent & networked software solutions must address key *accidental* & *inherent* complexities arising from
  - Limitations with development tools/techniques
  - Fundamental domain challenges

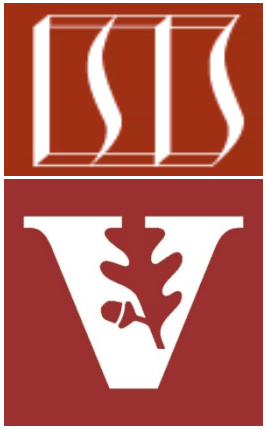
Summary

- Concurrent & networked software helps
  - Leverage advances in hardware technology
  - Meet the quality & performance needs of apps & services
- Successful concurrent & networked software solutions must address key *accidental* & *inherent* complexities arising from
  - Limitations with development tools/techniques
  - Fundamental domain challenges
- As concurrent & networked systems have grown in scale & functionality they must cope with a broader & more challenging set of complexities



Course Overview: Part 2

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



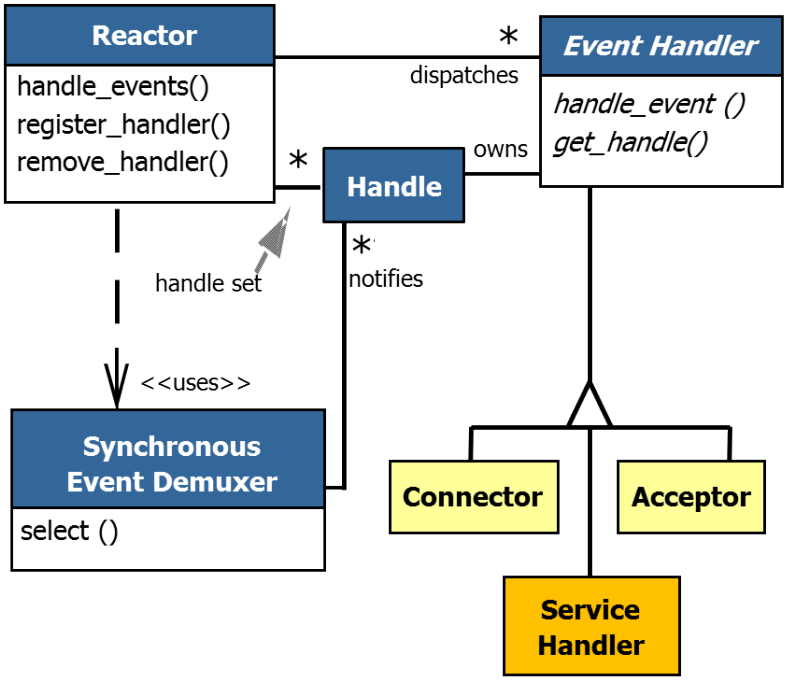
Professor of Computer Science

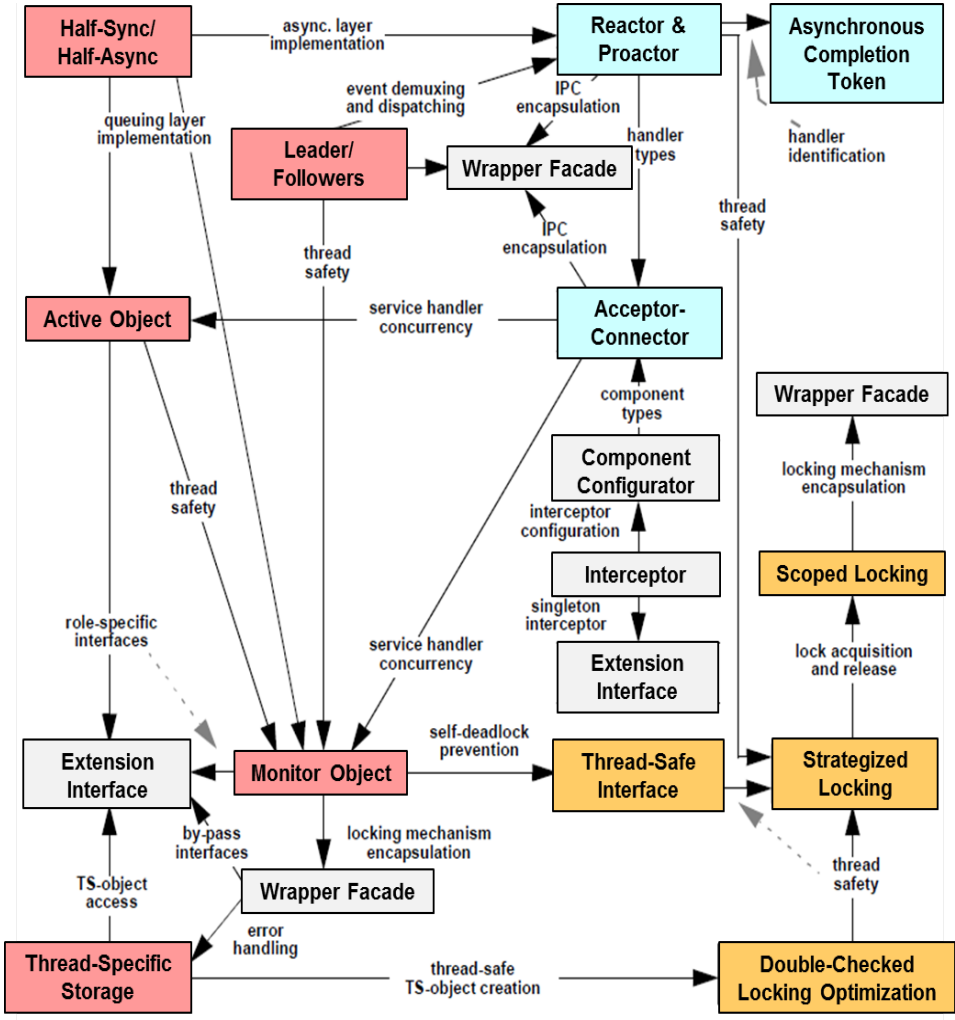
Institute for Software
Integrated Systems

Vanderbilt University
Nashville, Tennessee, USA

Topics Covered in this Part of the Module

- Explore the motivations for & challenges of concurrent & networked software
- Describe how *patterns* & *frameworks* help address challenges of concurrent & networked software





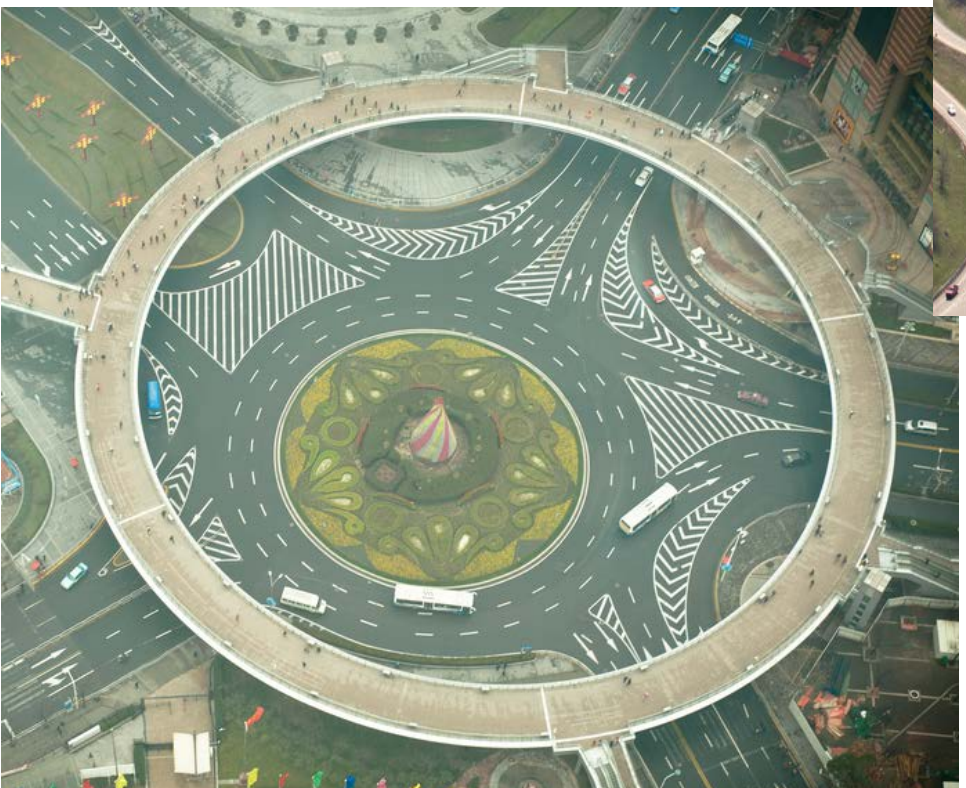
This section gives a high-level overview of patterns & frameworks

Motivation

To slay the dragons of accidental & inherent complexity associated with concurrent & networked software we need to codify *proven software experience*

Part of the Solution: Patterns

- Present solutions to common problems arising within a context



Part of the Solution: Patterns

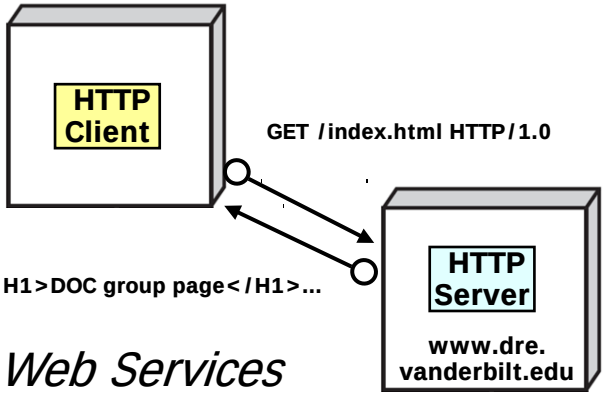
- Present solutions to common problems arising within a context

Electronic Trading

Mobile Devices

Aerospace

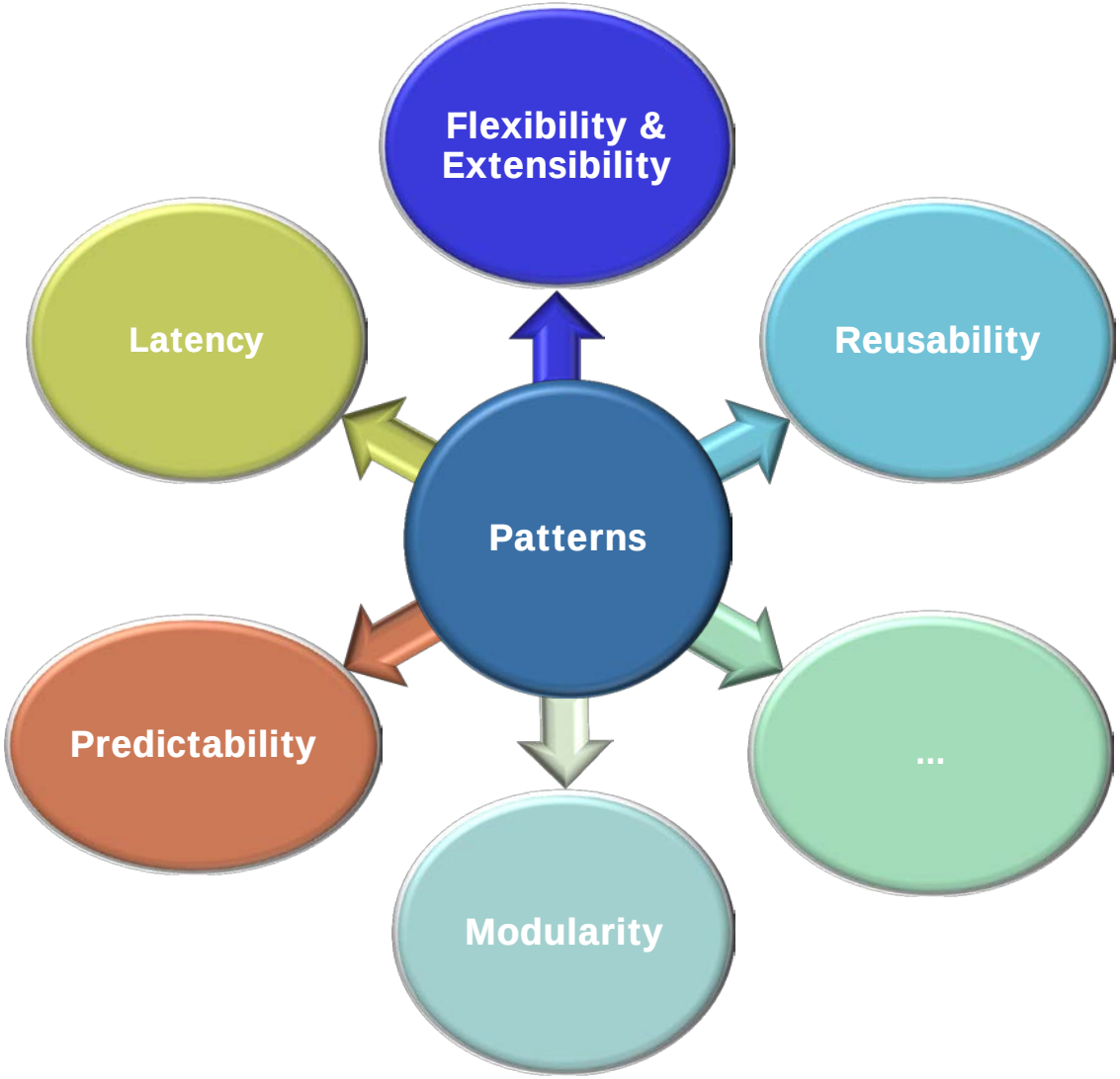
Automotive



Web Services

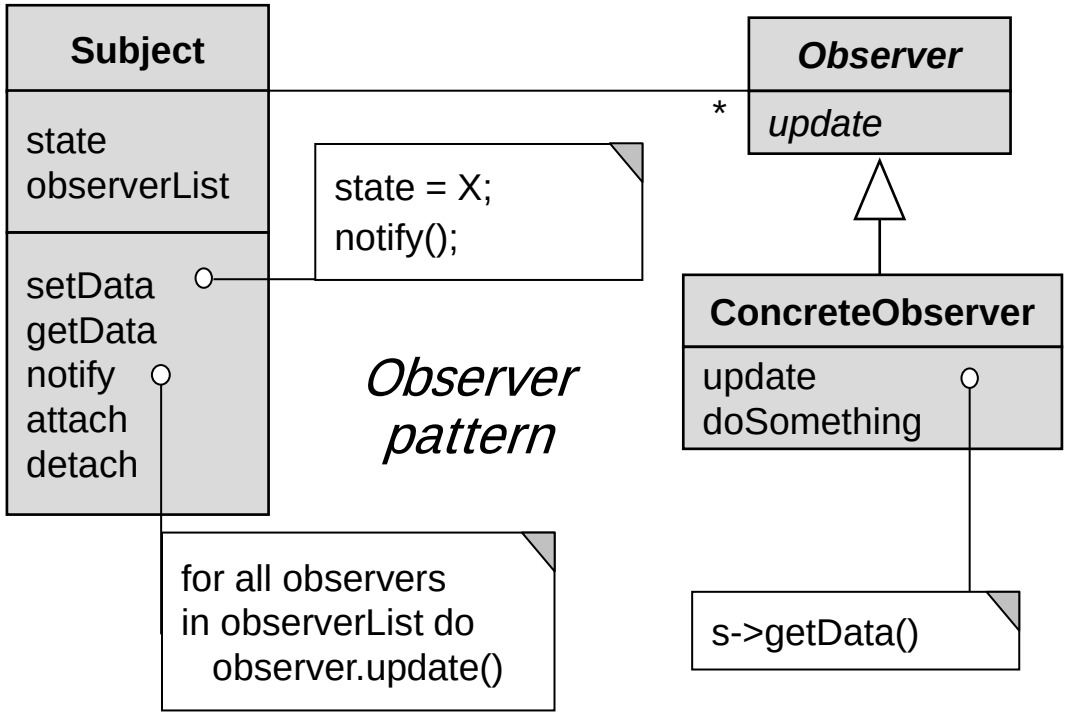
Part of the Solution: Patterns

- Present solutions to common problems arising within a context
- Help balance & resolve key software design forces



Part of the Solution: Patterns

- Present solutions to common problems arising within a context
- Help balance & resolve key software design forces
- Capture recurring structures & dynamics among software elements to facilitate reuse of successful designs



Intent: "Define a one-to-many dependency between objects so that when one object changes state, all dependents are notified & updated"

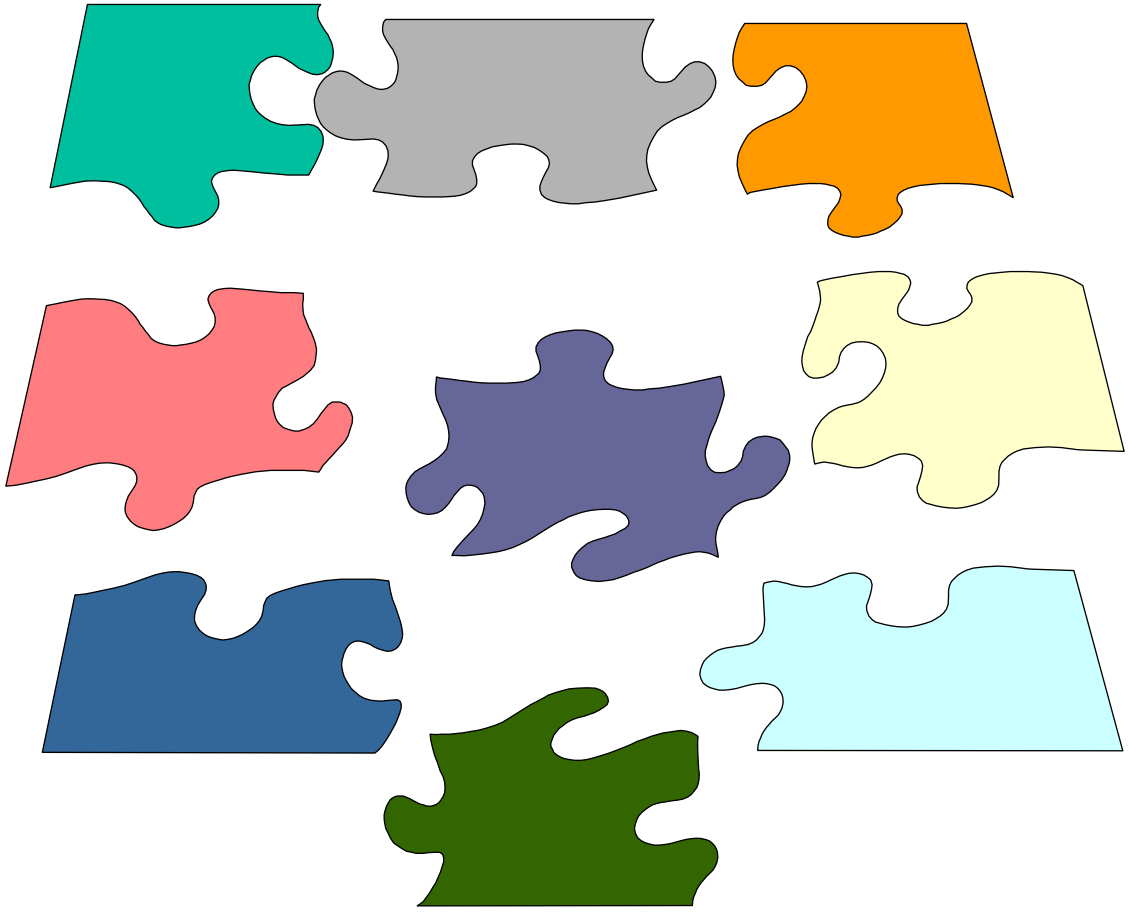
Part of the Solution: Patterns

- Present solutions to common problems arising within a context
- Help balance & resolve key software design forces
- Capture recurring structures & dynamics among software elements to facilitate reuse of successful designs
- Codify expert knowledge of design strategies, constraints & “best practices”

www.dre.vanderbilt.edu/~schmidt/CACM-patterns.html has more info

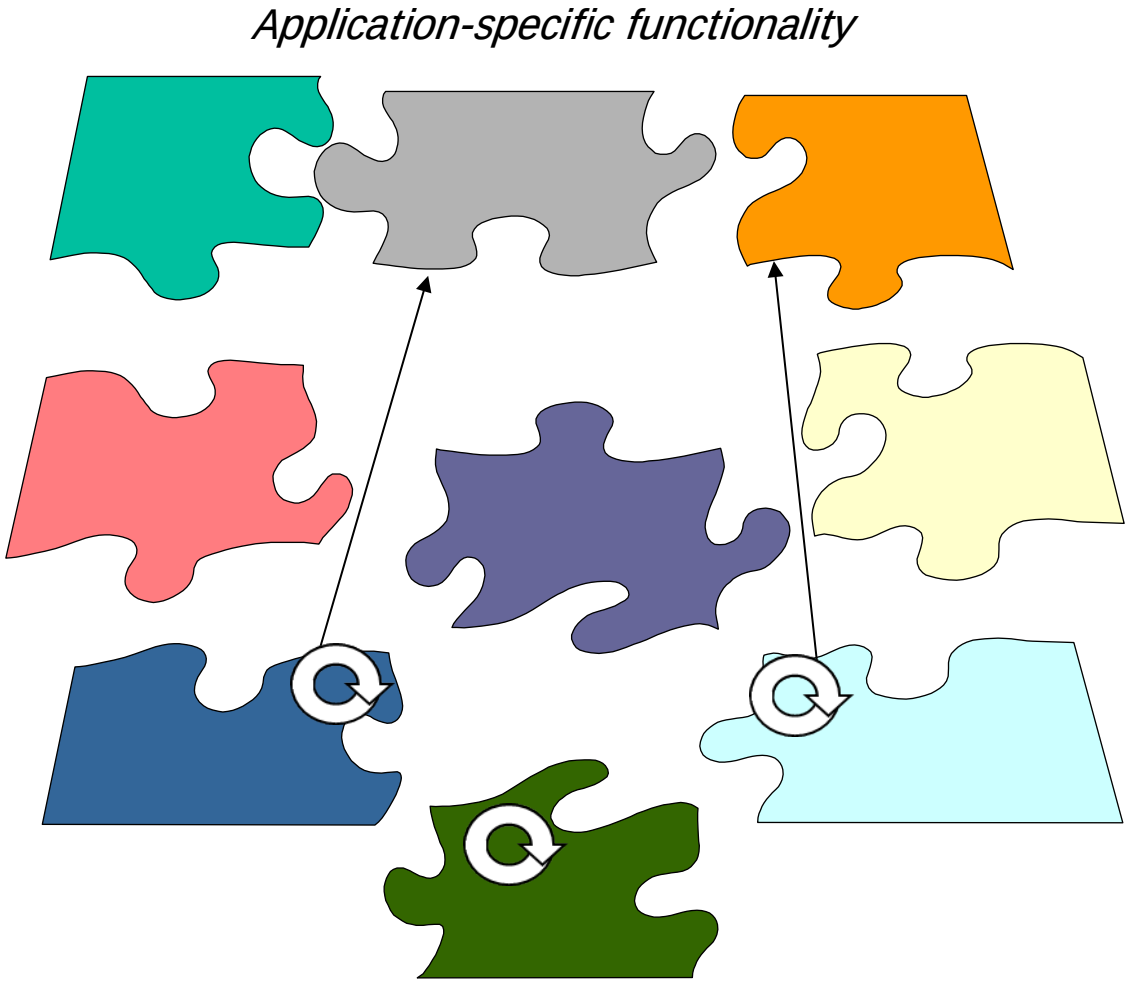
Another Part of the Solution: Frameworks

Frameworks are integrated sets of software components that collaborate to provide reusable architectures for families of related applications



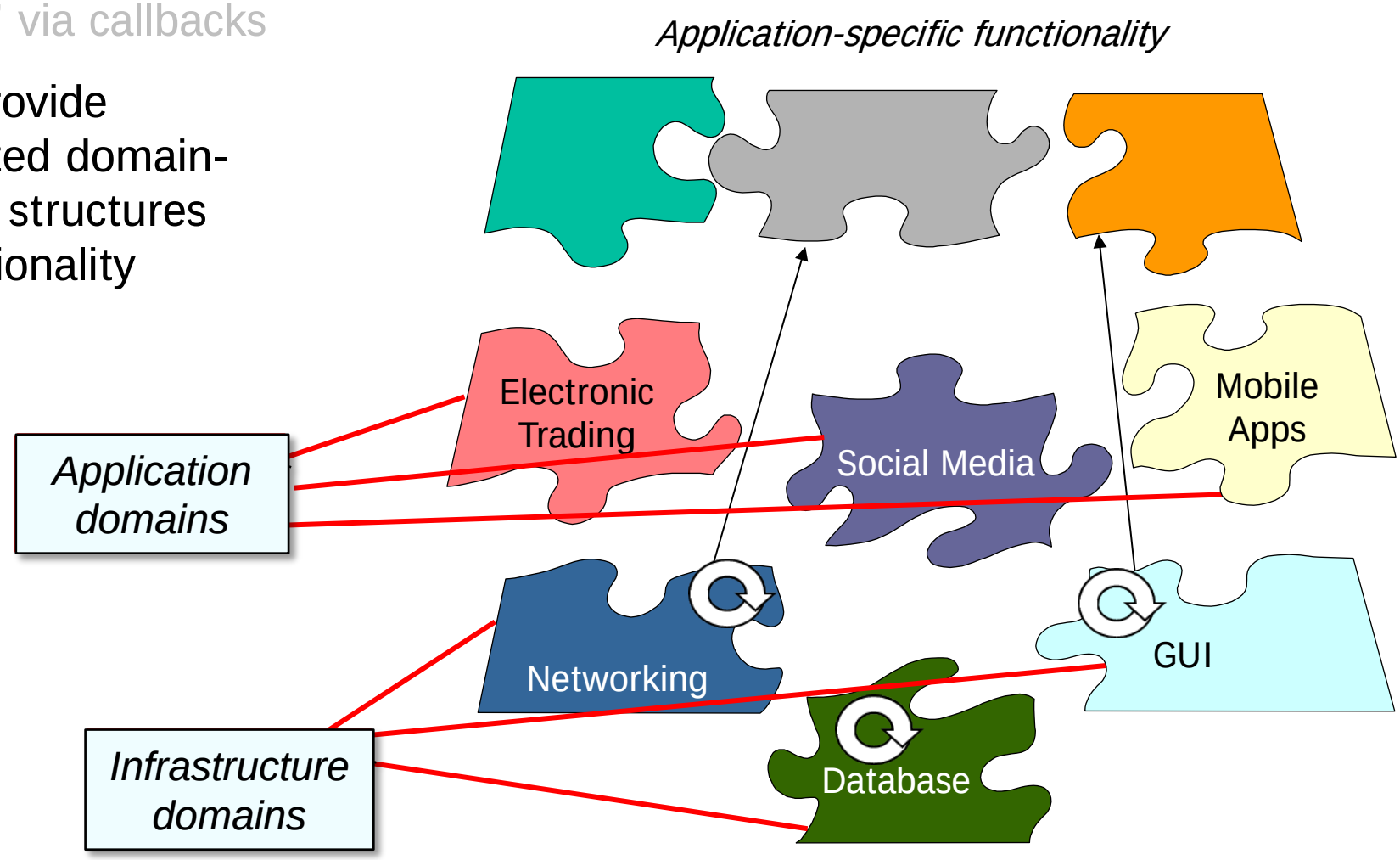
Another Part of the Solution: Frameworks

- They exhibit “inversion of control” via callbacks



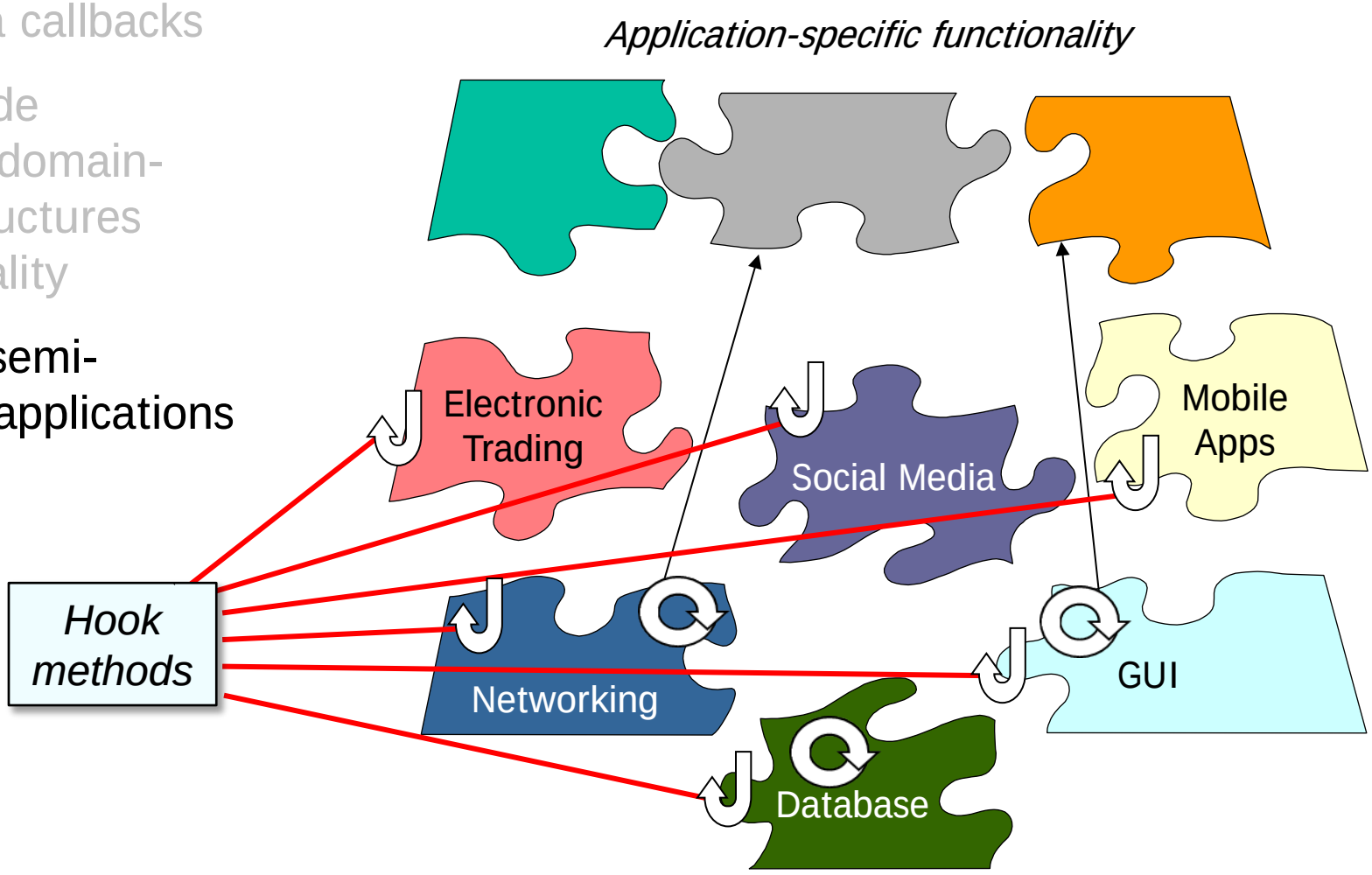
Another Part of the Solution: Frameworks

- They exhibit "inversion of control" via callbacks
- They provide integrated domain-specific structures & functionality



Another Part of the Solution: Frameworks

- They exhibit "inversion of control" via callbacks
- They provide integrated domain-specific structures & functionality
- They are "semi-complete" applications



www.dre.vanderbilt.edu/~schmidt/CACM-frameworks.html has more info

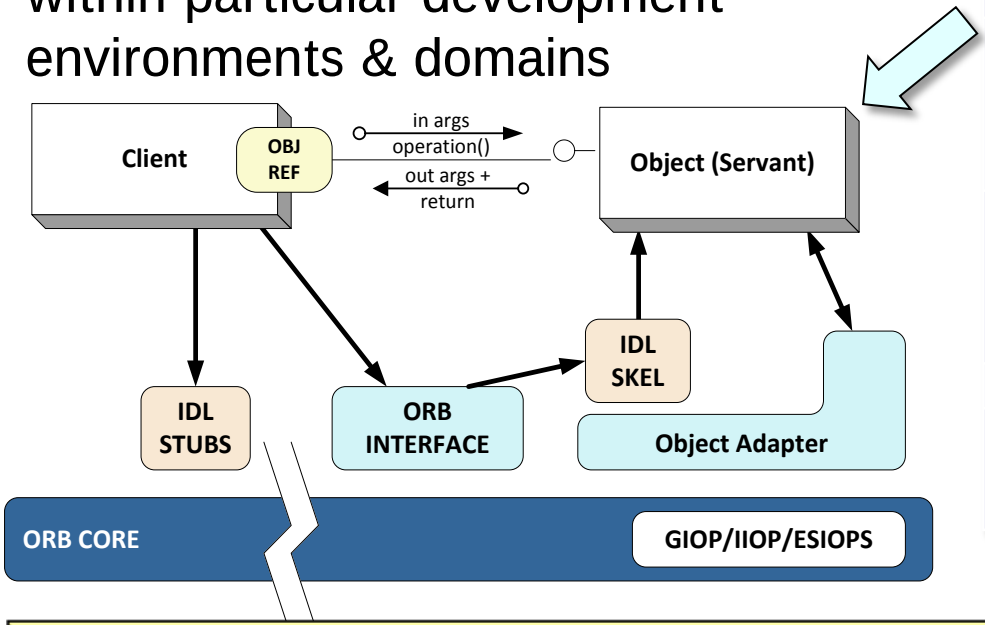
Key Benefit: Enable Systematic Software Reuse

- Patterns & frameworks codify proven experience to enable
  - **Design reuse** – Match problems to relevant structures/dynamics & patterns in a domain

Problem	Pattern
Encapsulating low-level system functions to enhance portability	Wrapper Façade
Demuxing broker core events efficiently	Reactor
Managing broker connections efficiently	Acceptor-Connector
Enhancing broker scalability by processing requests concurrently	Half-Sync/ Half-Async
Efficiently synchronize the Half-Sync/Async request queue	Monitor Object
...	...
Defining the broker's base-line architecture	Broker

Key Benefit: Enable Systematic Software Reuse

- Patterns & frameworks codify proven experience to enable
  - **Design reuse** – Match problems to relevant structures/dynamics & patterns in a domain
  - **Code reuse** – Reify proven designs within particular development environments & domains

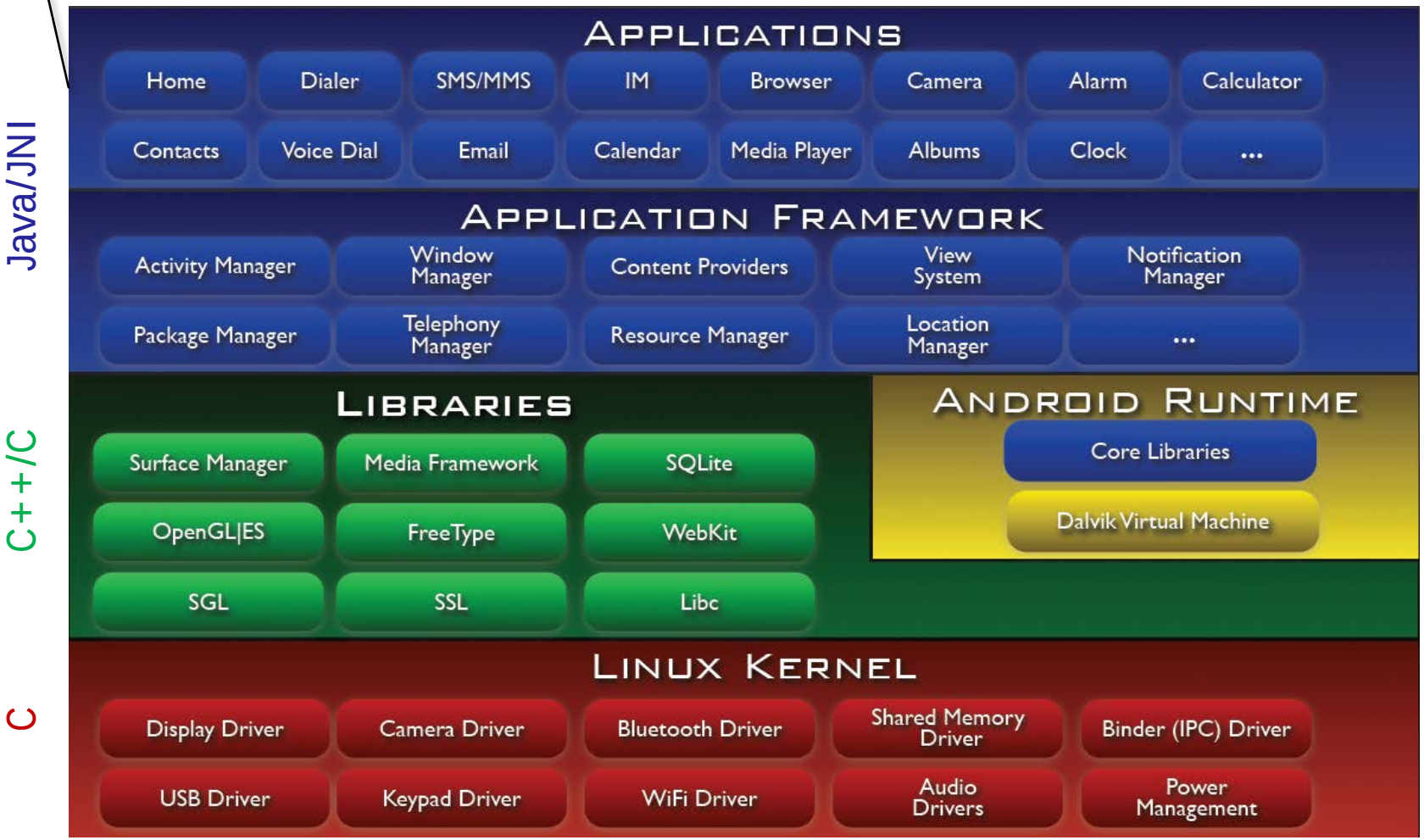


Problem	Pattern
Encapsulating low-level system functions to enhance portability	Wrapper Façade
Demuxing broker core events efficiently	Reactor
Managing broker connections efficiently	Acceptor-Connector
Enhancing broker scalability by processing requests concurrently	Half-Sync/ Half-Async
Efficiently synchronize the Half-Sync/Async request queue	Monitor Object
…	…
Defining the broker's base-line architecture	Broker

Patterns & frameworks help avoid "reinventing the wheel" for many accidental & inherent complexities of concurrent & networked software

Examples of Systematic Software Reuse

Android is a pattern-oriented software stack for mobile device apps

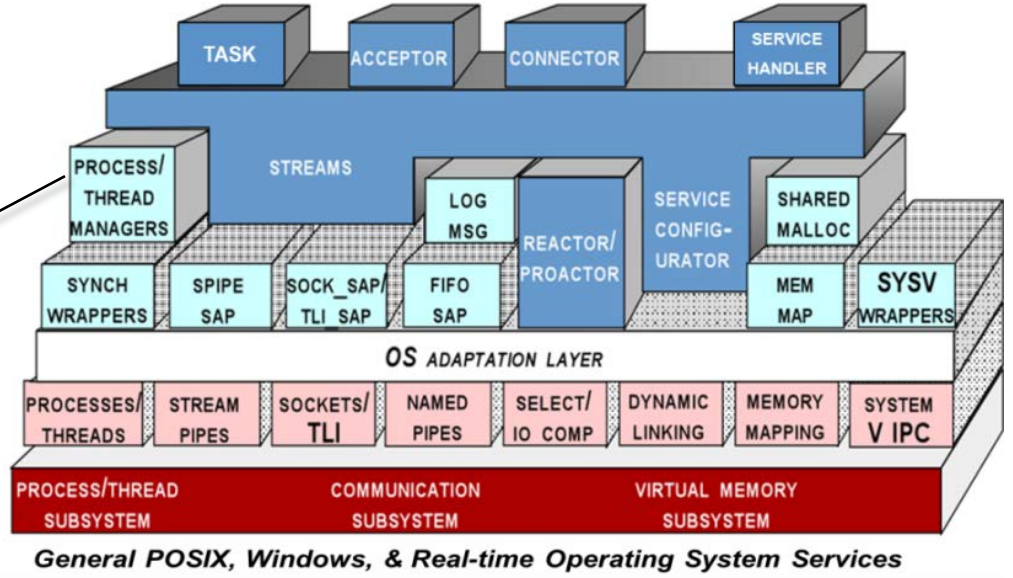


developer.android.com/guide/basics/what-is-android.html

Examples of Systematic Software Reuse

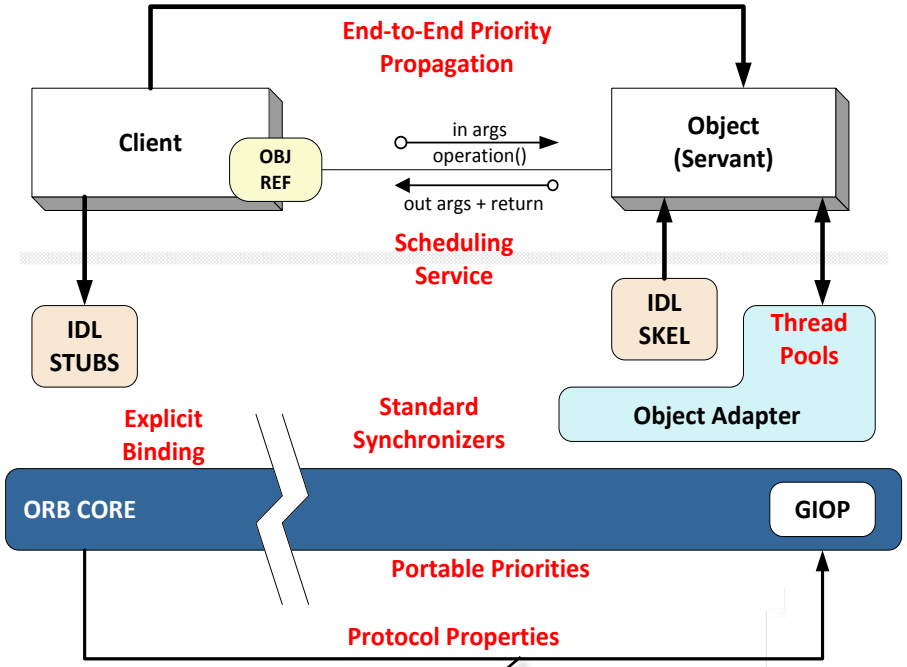
www.dre.vanderbilt.edu/ACE

ACE is a pattern-oriented toolkit containing portable components & frameworks for concurrent & networked software



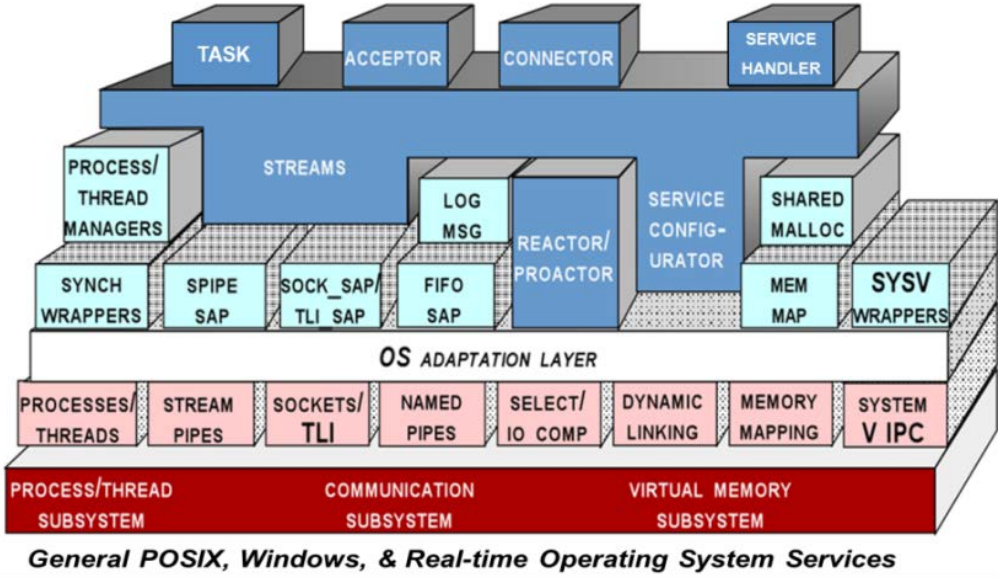
Examples of Systematic Software Reuse

www.dre.vanderbilt.edu/TAO



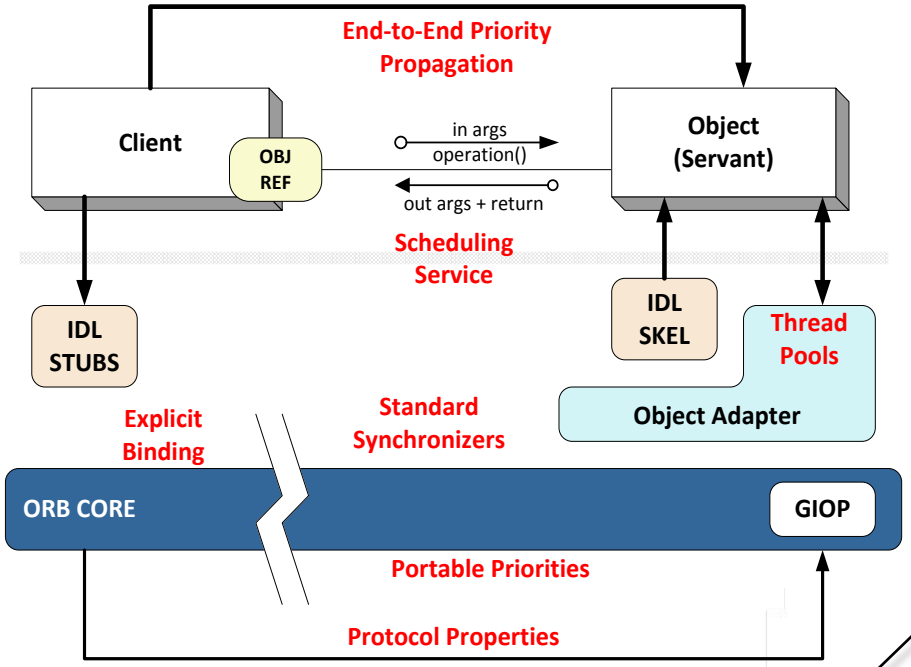
The ACE ORB (TAO) is a pattern-oriented real-time object-request broker

www.dre.vanderbilt.edu/ACE

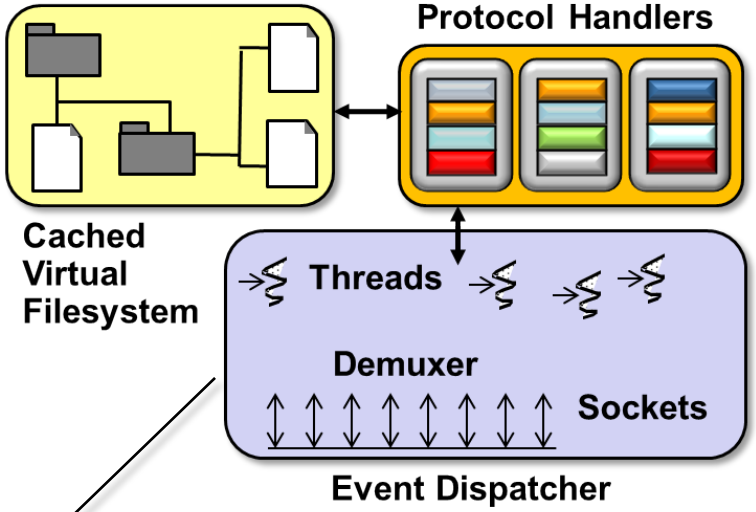


Examples of Systematic Software Reuse

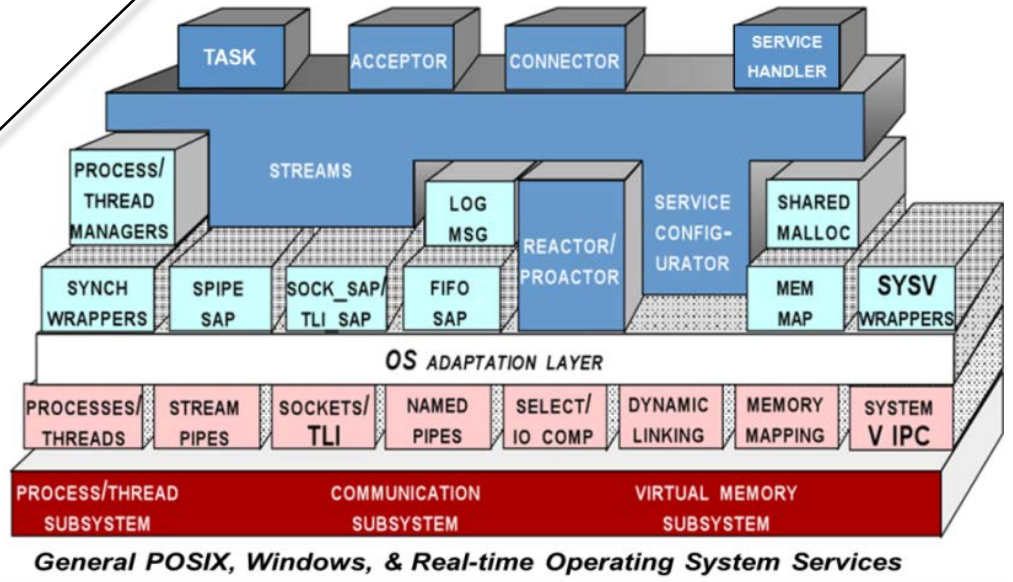
www.dre.vanderbilt.edu/TAO



www.dre.vanderbilt.edu/JAWS



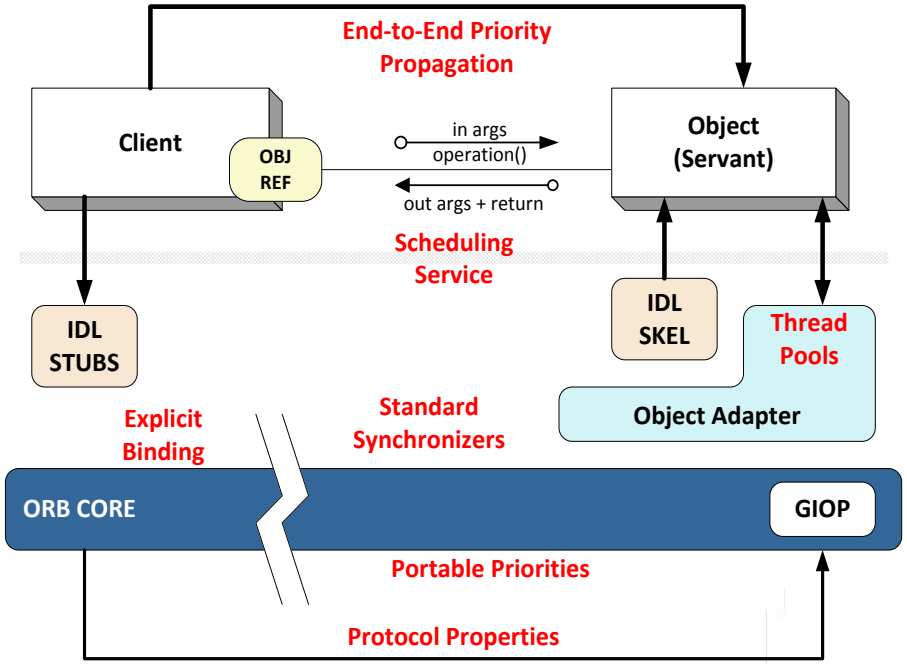
www.dre.vanderbilt.edu/ACE



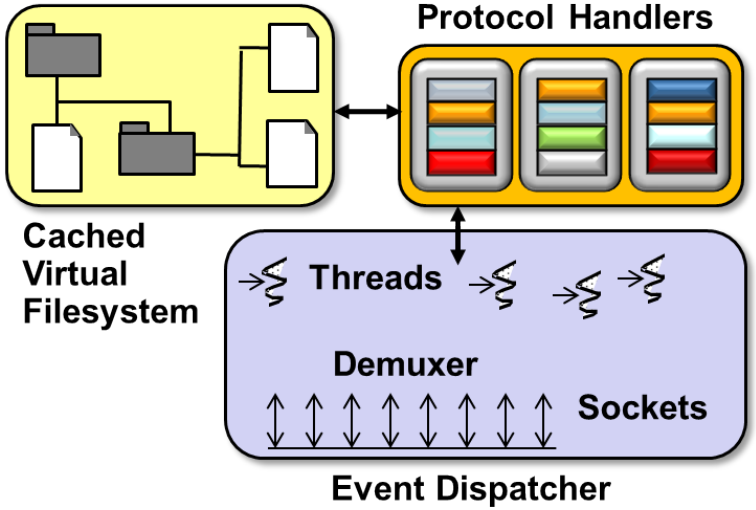
JAWS is a pattern-oriented high-performance HTTP web server

Examples of Systematic Software Reuse

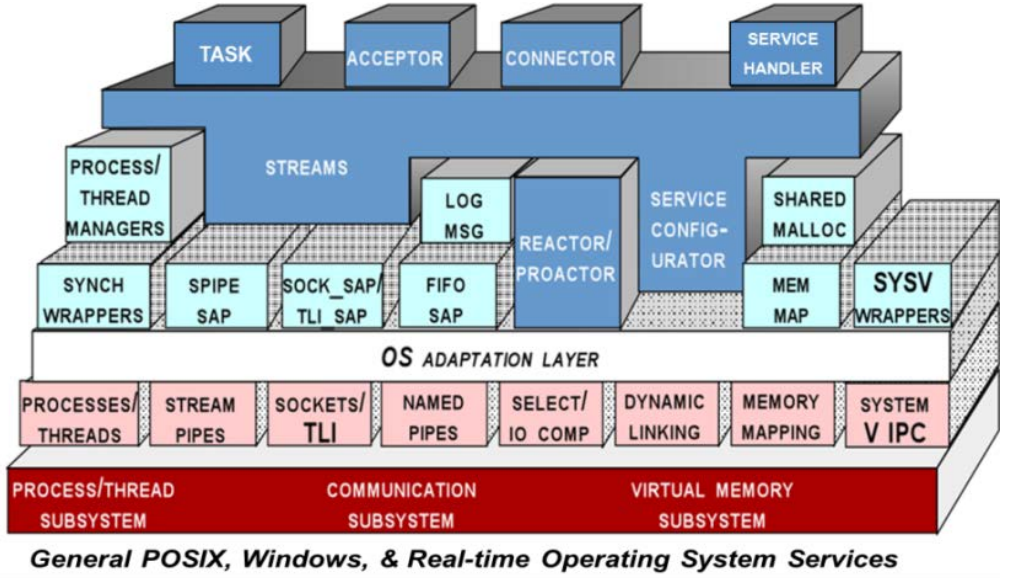
www.dre.vanderbilt.edu/TAO



www.dre.vanderbilt.edu/JAWS



www.dre.vanderbilt.edu/ACE



Android, ACE, TAO, & JAWS are all freely available in open-source form

Summary

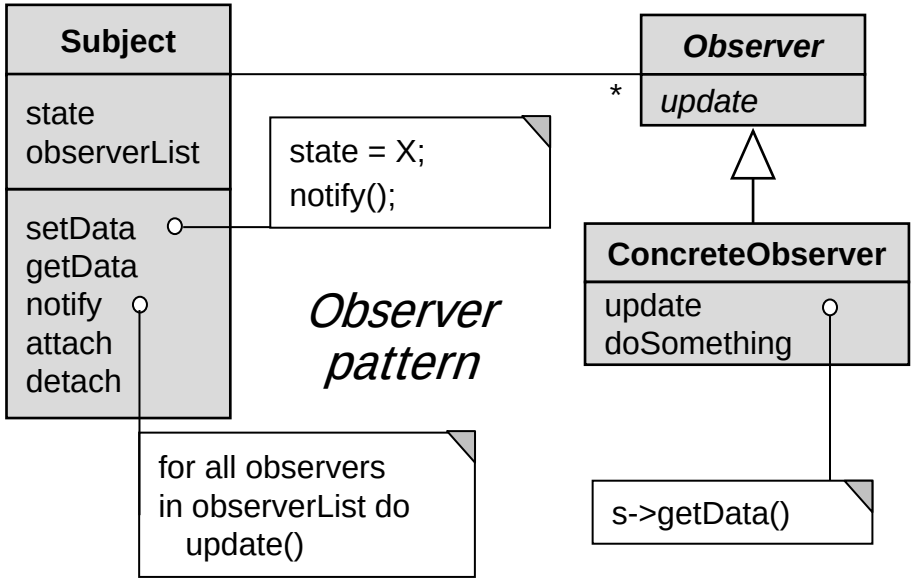
This course will show by example how patterns & frameworks can help to

- **Codify good software design & implementation practices**
  - Distill & generalize experience
  - Aid to novices & experts alike

Summary

This course will show by example how patterns & frameworks can help to

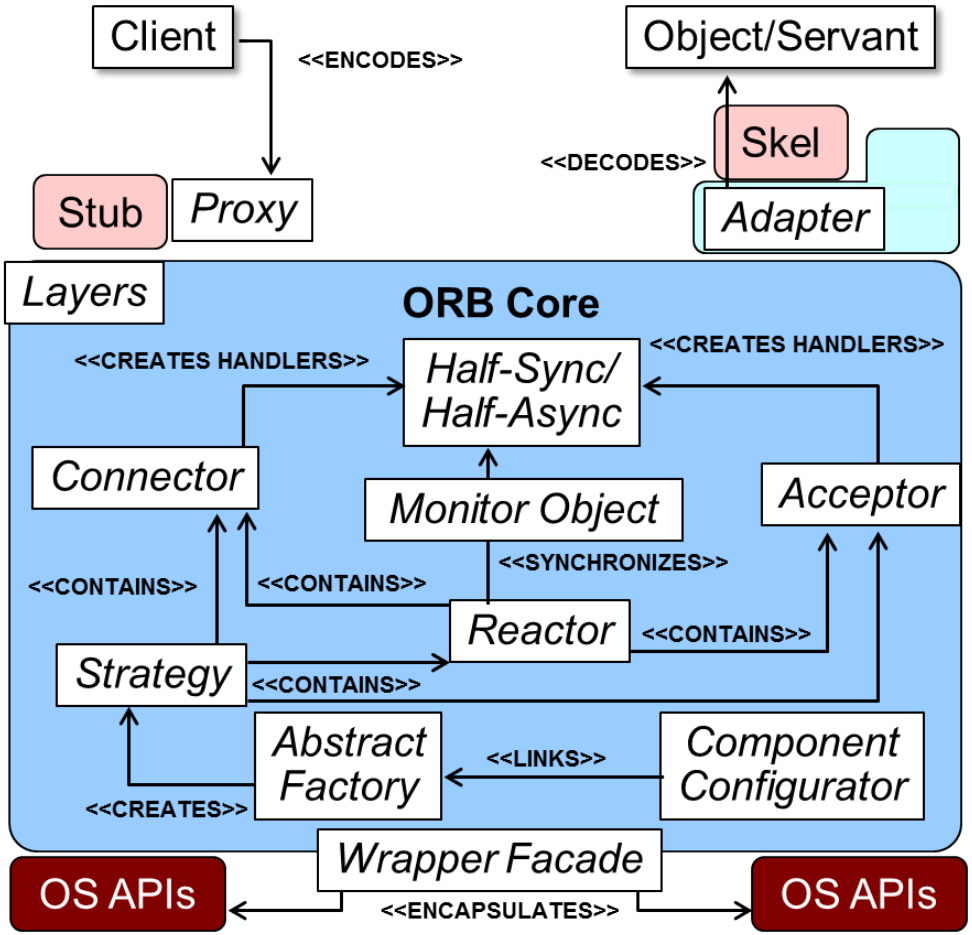
- **Codify good software design & implementation practices**
  - Distill & generalize experience
  - Aid to novices & experts alike
- **Give explicit & intuitive names to common design structures**
  - Shared vocabulary
  - Reduced complexity
  - Greater expressivity



Summary

This course will show by example how patterns & frameworks can help to

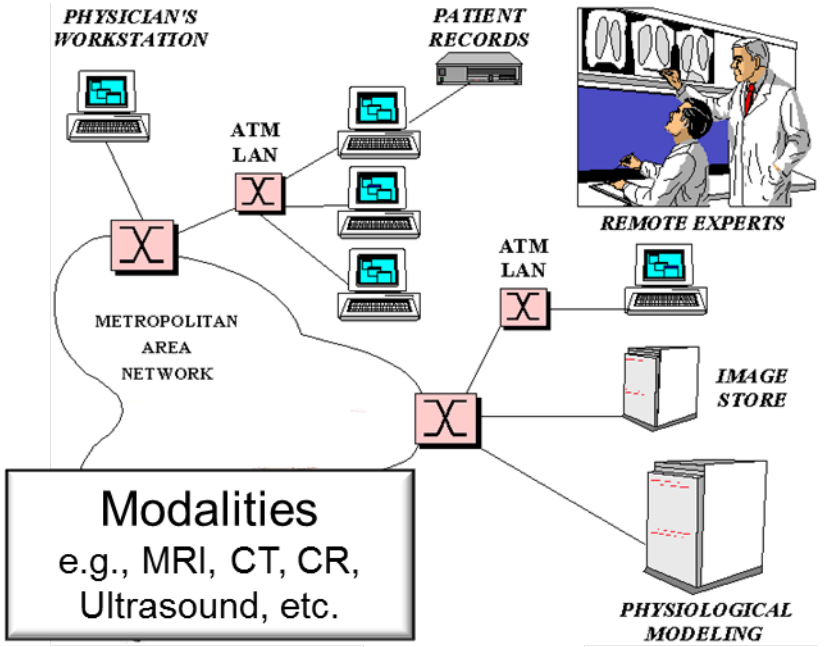
- **Codify good software design & implementation practices**
  - Distill & generalize experience
  - Aid to novices & experts alike
- **Give explicit & intuitive names to common design structures**
  - Shared vocabulary
  - Reduced complexity
  - Greater expressivity
- **Capture & preserve design & implementation knowledge**
  - Articulate key decisions succinctly
  - Improve documentation

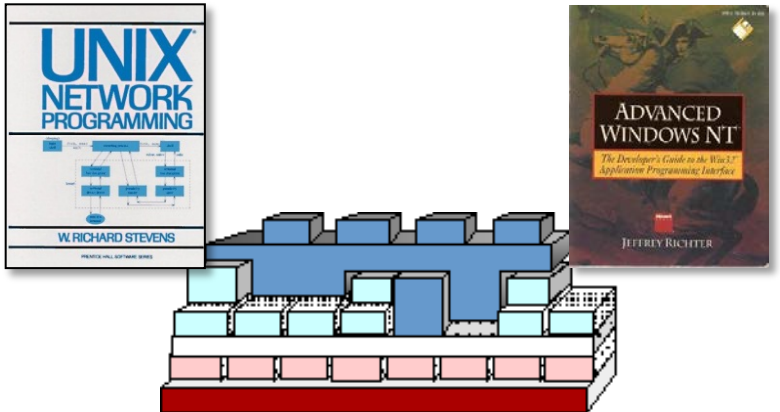


Summary

This course will show by example how patterns & frameworks can help to

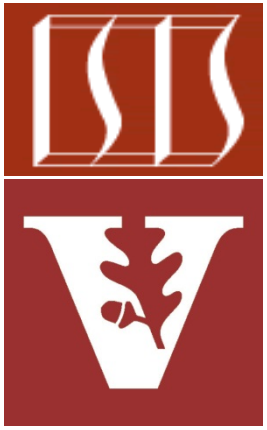
- **Codify good software design & implementation practices**
  - Distill & generalize experience
  - Aid to novices & experts alike
- **Give explicit & intuitive names to common design structures**
  - Shared vocabulary
  - Reduced complexity
  - Greater expressivity
- **Capture & preserve design & implementation knowledge**
  - Articulate key decisions succinctly
  - Improve documentation
- **Facilitate restructuring / refactoring**
  - Enhance reuse & productivity
  - Adapt gracefully to the unexpected





Course Overview: Part 3

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



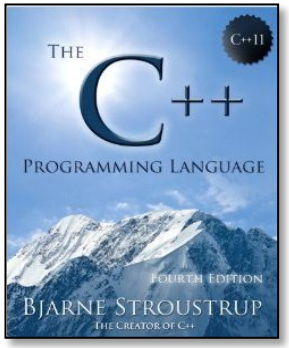
Professor of Computer Science

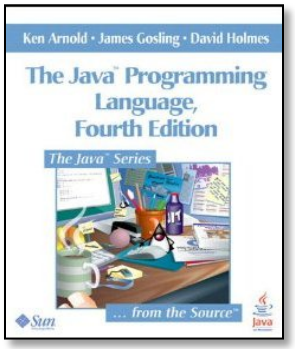
Institute for Software Integrated Systems

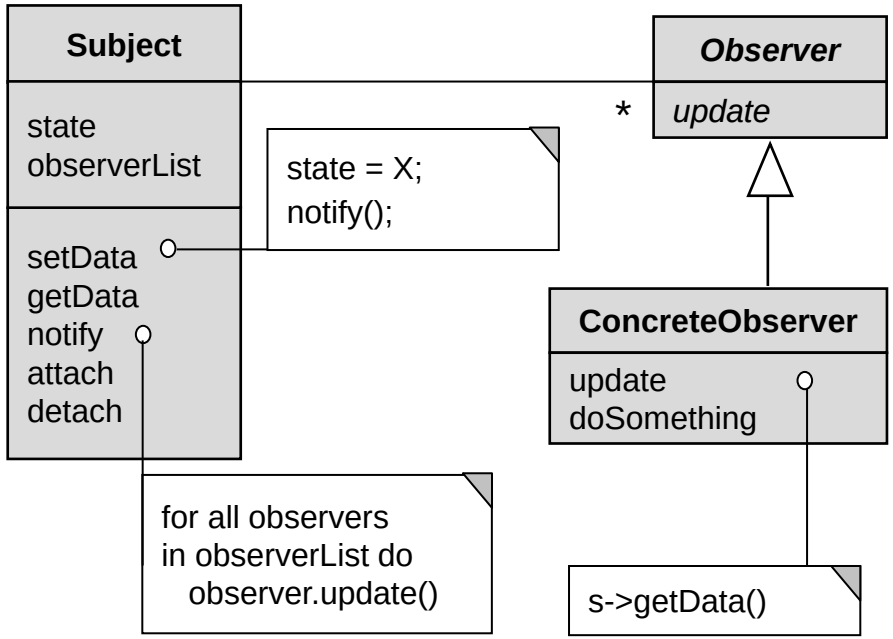
Vanderbilt University
Nashville, Tennessee, USA

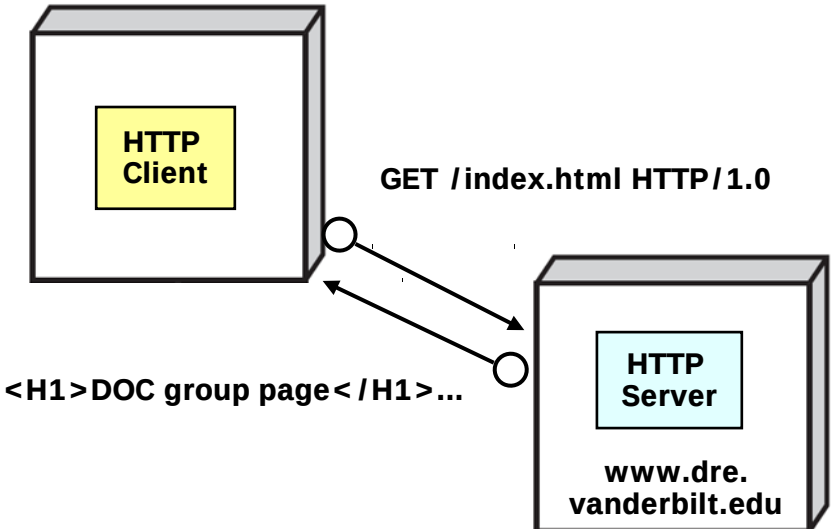
Topics Covered in this Part of the Module

- Explore the motivations for & challenges of concurrent & networked software
- Describe how *patterns* & *frameworks* help address challenges of concurrent & networked software
- Outline the course "prerequisities"





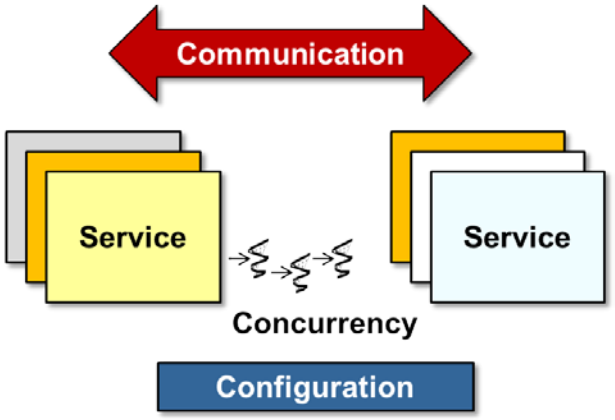




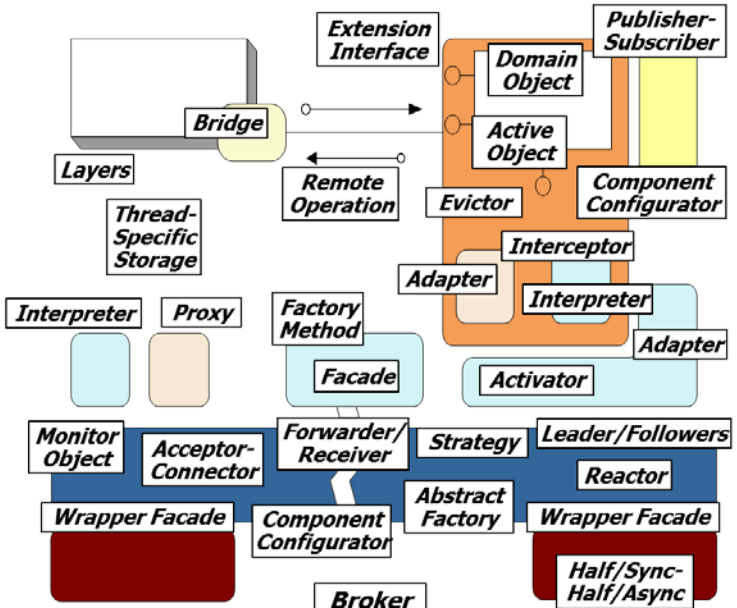
Topics Covered in this Part of the Module

- Explore the motivations for & challenges of concurrent & networked software
- Describe how *patterns* & *frameworks* help address challenges of concurrent & networked software
- Outline the course "prerequisities"
- Present an overview of the three sections in the course syllabus

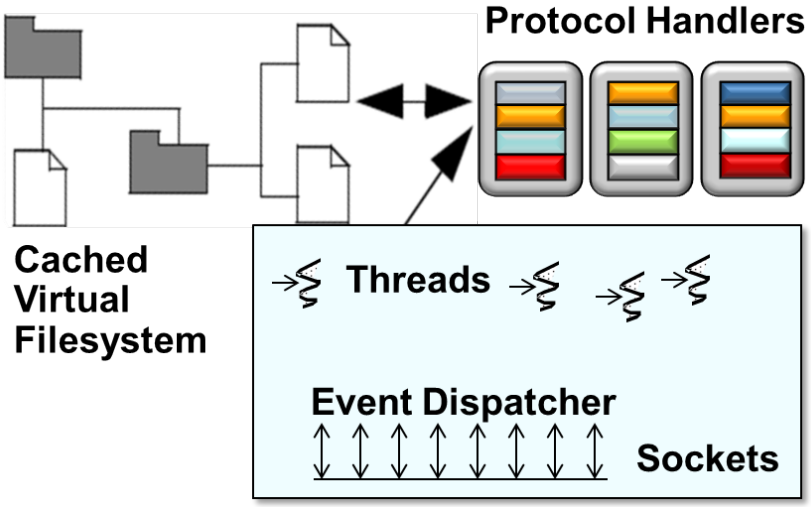
Section 1



Section 2



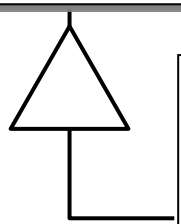
Section 3



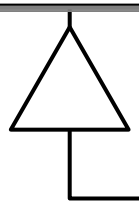
Course "Prerequisites"

- When I teach courses on patterns & frameworks at Vanderbilt University, I can rely on the students having taken the prerequsities

CS 251. Intermediate Software Design. High quality development and reuse of architectural patterns, design patterns, and software components. Theoretical and practical aspects of developing, documenting, testing, and applying reusable class libraries and object-oriented frameworks using object-oriented and component-based programming languages and tools. Prerequisite: CS 201. FALL, SPRING. [3]



CS 201. Program Design and Data Structures. Continuation of CS 101. The study of elementary data structures, their associated algorithms and their application in problems; rigorous development of programming techniques and style; design and implementation of programs with multiple modules, using good data structures and good programming style. Prerequisite: CS 101. FALL, SPRING. [3]



CS 101. Programming and Problem Solving. An intensive introduction to algorithm development and problem solving on the computer. Structured problem definition, top down and modular algorithm design. Running, debugging, and testing programs. Program documentation. FALL, SPRING. [3]

Course "Prerequisites"

- Ideally, you are at least somewhat familiar with
  - **OO programming languages**
    - e.g., classes, inheritance, dynamic binding, & parameterized types available in Java, C++, C#, etc.

```
template <typename T>
class Event_Handler :
  public Observer<T> {
public:
   virtual void update
     (Observable<T> obs,
      const T &arg)
   { /* … */ }
   …
```

```
template <typename T> class Event_Source :
  public Observable<T>, public ACE_Task_Base { public:
    virtual void svc () { /* … */ notify_observers (/* … */); }
    …
Event_Source<Data_Block> event_source;
Event_Handler<Data_Block> event_handler;
event_source->add_observer (event_handler);
Event_Task<Data_Block> task (event_source);
task->activate ();
…
```

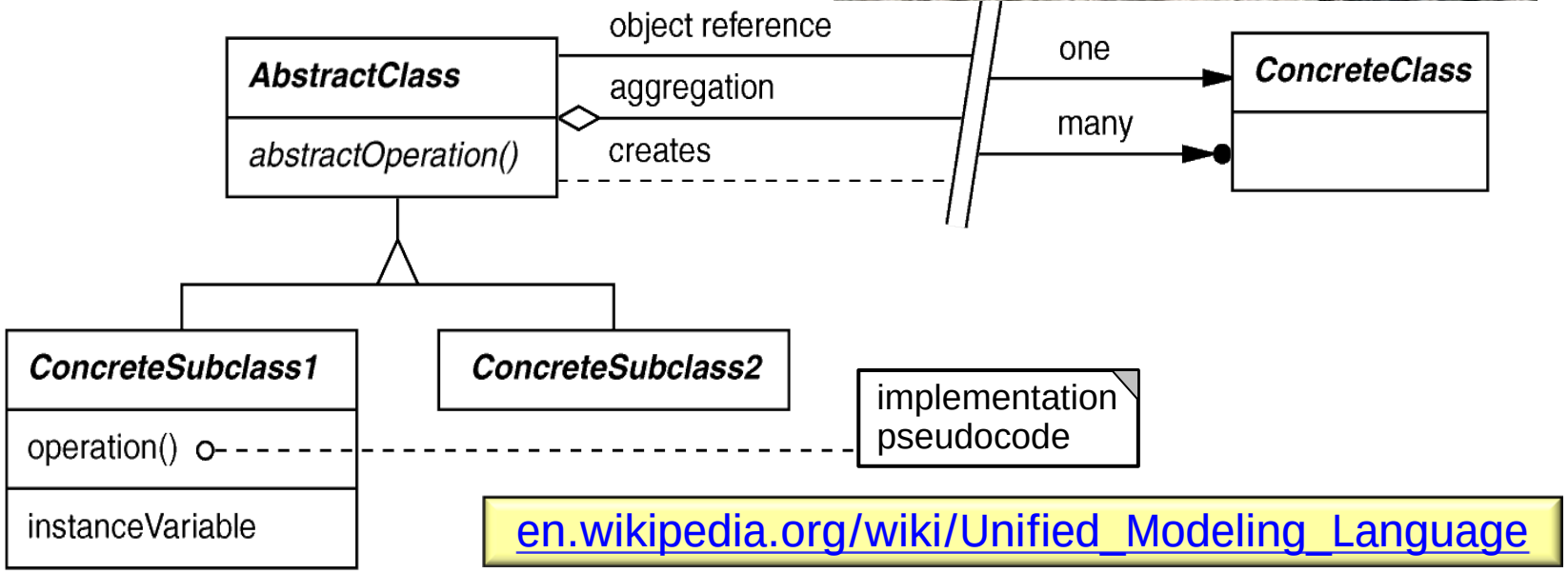
If you don't know Java or C++ you'll struggle with later sections of the course

Course "Prerequisites"

- Ideally, you are at least somewhat familiar with
  - **OO programming languages**
    - e.g., classes, inheritance, dynamic binding, & parameterized types available in Java, C++, C#, etc.
  - **OO design concepts & notations**
    - e.g., encapsulation, abstraction, polymorphism, extensibility

Course "Prerequisites"

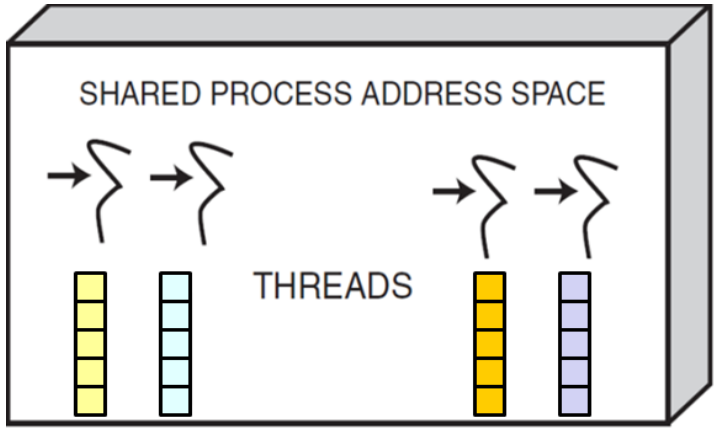
- Ideally, you are at least somewhat familiar with
  - **OO programming languages**
    - e.g., classes, inheritance, dynamic binding, & parameterized types available in Java, C++, C#, etc.
  - **OO design concepts & notations**
    - e.g., encapsulation, abstraction, polymorphism, extensibility, & UML

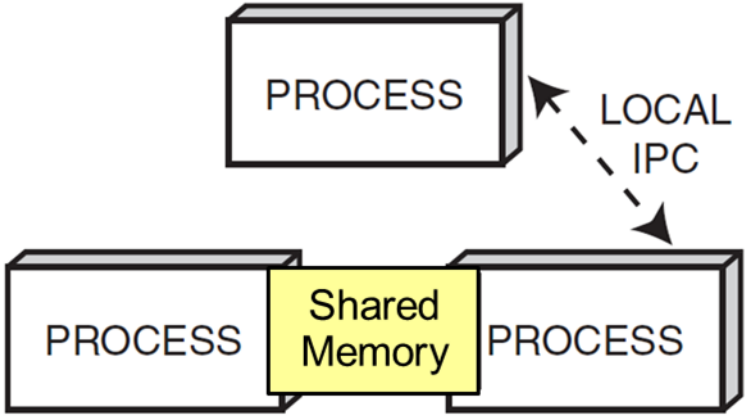


en.wikipedia.org/wiki/Unified_Modeling_Language

Course "Prerequisites"

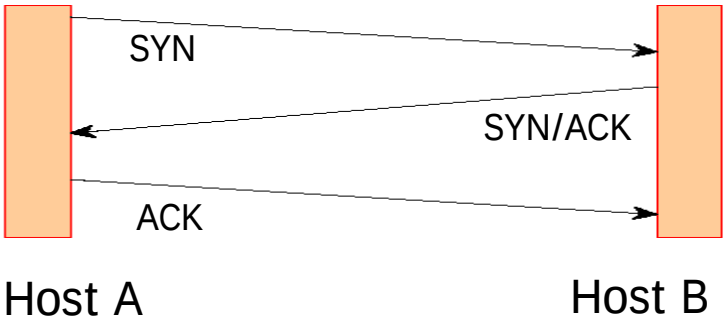
- Ideally, you are at least somewhat familiar with
  - **OO programming languages**
    - e.g., classes, inheritance, dynamic binding, & parameterized types available in Java, C++, C#, etc.
  - **OO design concepts & notations**
    - e.g., encapsulation, abstraction, polymorphism, extensibility, & UML
  - **Systems programming concepts**
    - e.g., event (de)muxing, processes/ threads, synchronization, & inter-process communication (IPC)



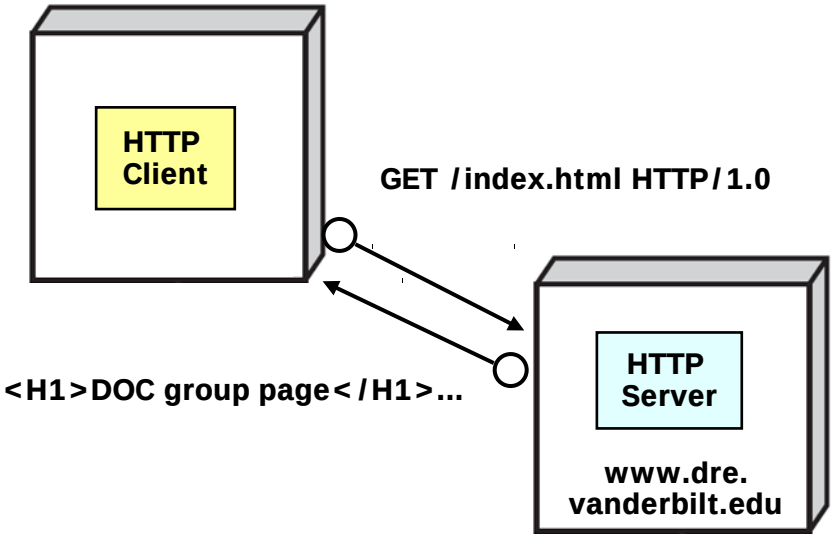


Course "Prerequisites"

- Ideally, you are at least somewhat familiar with
  - **OO programming languages**
    - e.g., classes, inheritance, dynamic binding, & parameterized types available in Java, C++, C#, etc.
  - **OO design concepts & notations**
    - e.g., encapsulation, abstraction, polymorphism, extensibility, & UML
  - **Systems programming concepts**
    - e.g., event (de)muxing, processes/ threads, synchronization, & inter-process communication (IPC)
  - **Networking concepts**
    - e.g., client/server & peer-to-peer architectures, TCP/IP, & layering

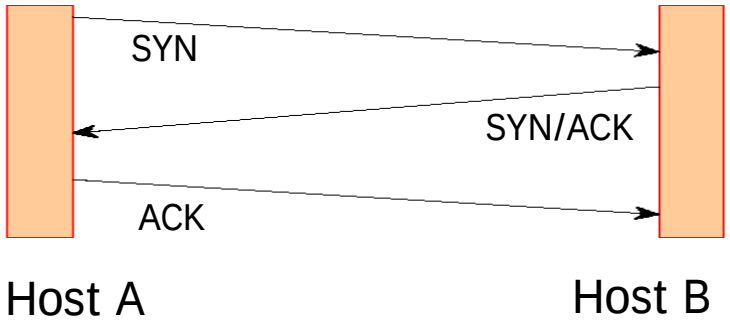


3-way handshake in TCP/IP

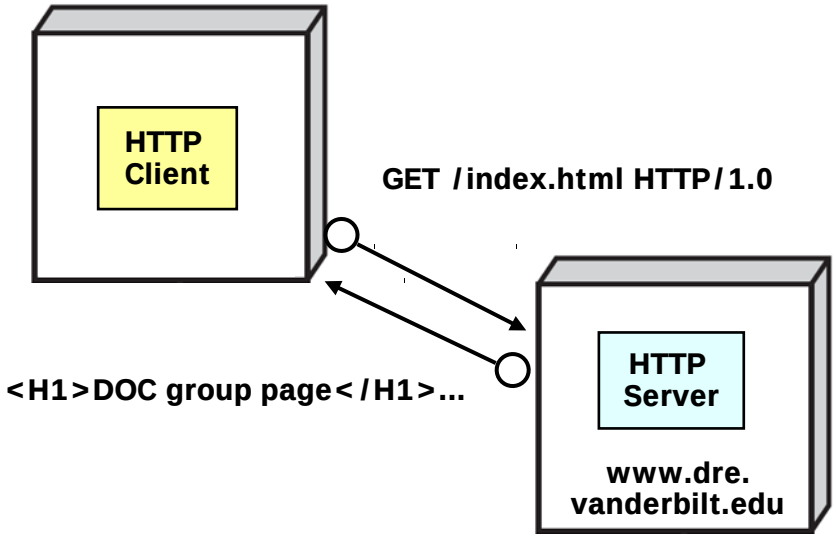


Course "Prerequisites"

- Ideally, you are at least somewhat familiar with
  - **OO programming languages**
    - e.g., classes, inheritance, dynamic binding, & parameterized types available in Java, C++, C#, etc.
  - **OO design concepts & notations**
    - e.g., encapsulation, abstraction, polymorphism, extensibility, & UML
  - **Systems programming concepts**
    - e.g., event (de)muxing, processes/ threads, synchronization, & inter-process communication (IPC)
  - **Networking concepts**
    - e.g., client/server & peer-to-peer architectures, TCP/IP, & layering



3-way handshake in TCP/IP



We'll review key OO design, systems programming, & networking concepts

Course Syllabus

- This course is composed of three Sections

Course Syllabus

- This course is composed of three Sections
  - Each Section is composed of Modules

Course Syllabus

- This course is composed of three Sections
  - Each Section is composed of Modules
  - Each Module is composed of Parts

Course Syllabus

- This course is composed of three Sections
  - Each Section is composed of Modules
  - Each Module is composed of Parts
  - Each Part is ~20 minutes long

Course Syllabus

- This course is composed of three Sections
  - Each Section is composed of Modules
  - Each Module is composed of Parts
  - Each Part is ~20 minutes long
  - Every ~5 minutes there's a quiz

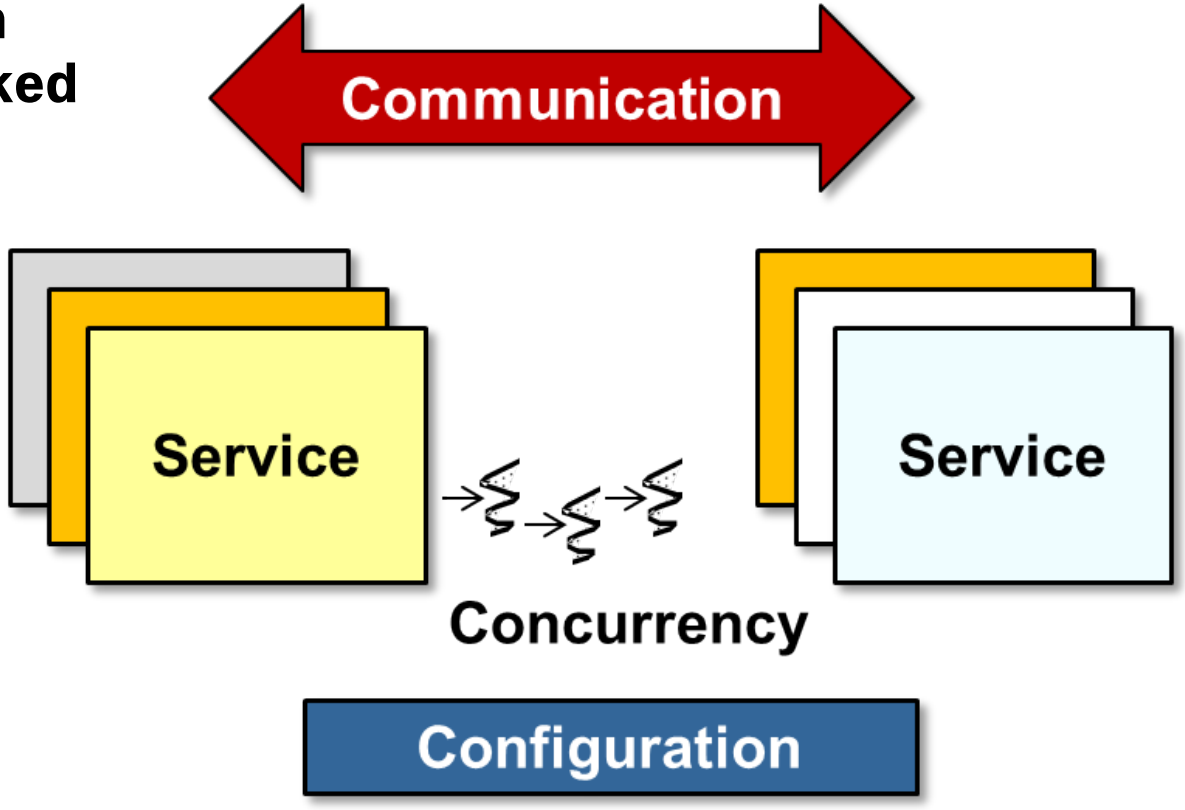
Course Syllabus

- This course is composed of three Sections
  - Each Section is composed of Modules
  - Each Module is composed of Parts
  - Each Part is ~20 minutes long
  - Every ~5 minutes there's a quiz
  - We'll have several programming assignments that can be written in C++ or Java

A Statement of Accomplishment requires you to do the quizzes & assignments

Course Syllabus

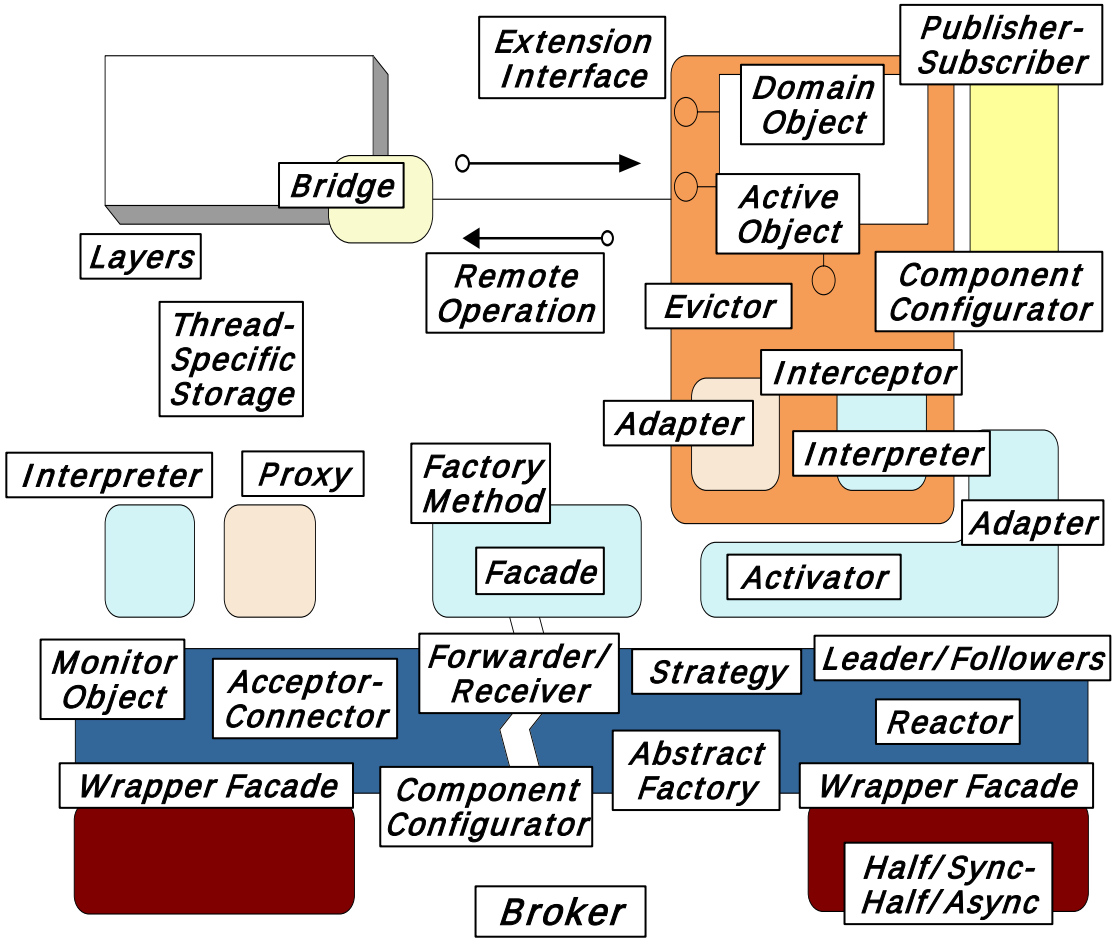
- **Section 1 – Introduction to concurrent & networked software**
  - Layers
  - Design dimensions
  - OS programming mechanisms
  - Android programming mechanisms



This section focuses primarily on concepts – there's little/no code/patterns here

Course Syllabus

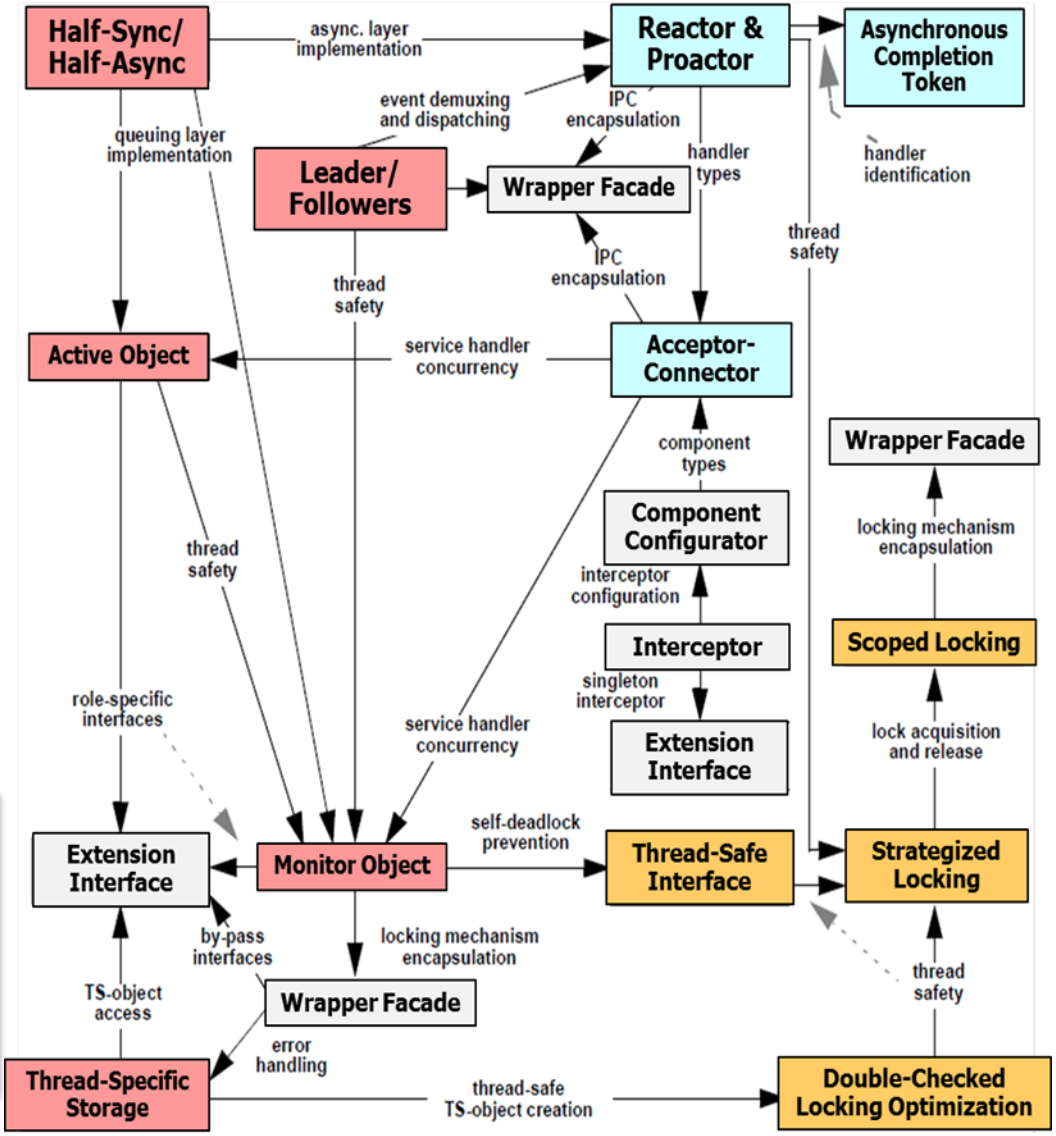
- **Section 2 – Introduction to patterns & frameworks** (with emphasis on concurrent & networked software)
  - Pattern concepts
  - Pattern examples
  - Pattern relationships
  - Framework concepts
  - Framework examples
  - Pros & cons of patterns & frameworks
  - Reference material



This section focuses on design – there's some Java/C++/C code shown here

Course Syllabus

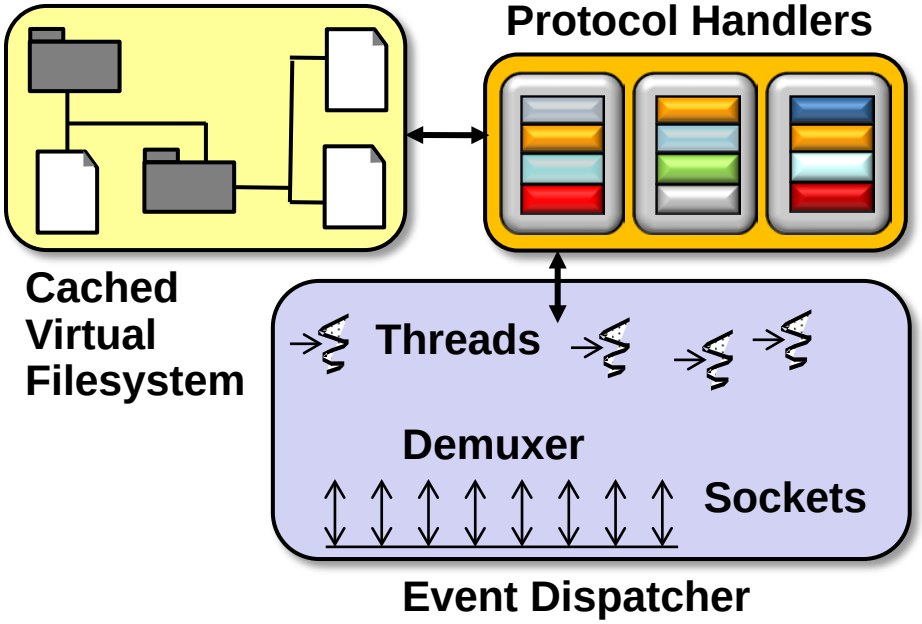
- **Section 3 – Applying patterns & frameworks to concurrent & networked software**
  - Service access & configuration
  - Event handling
  - Concurrency
  - Synchronization

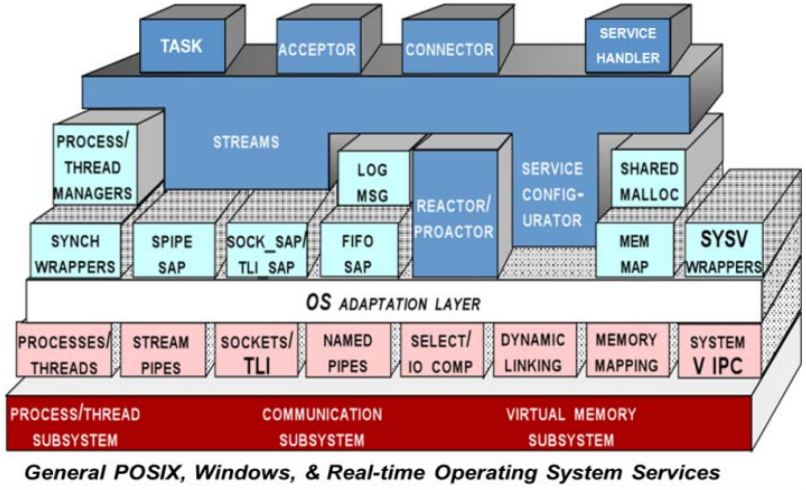


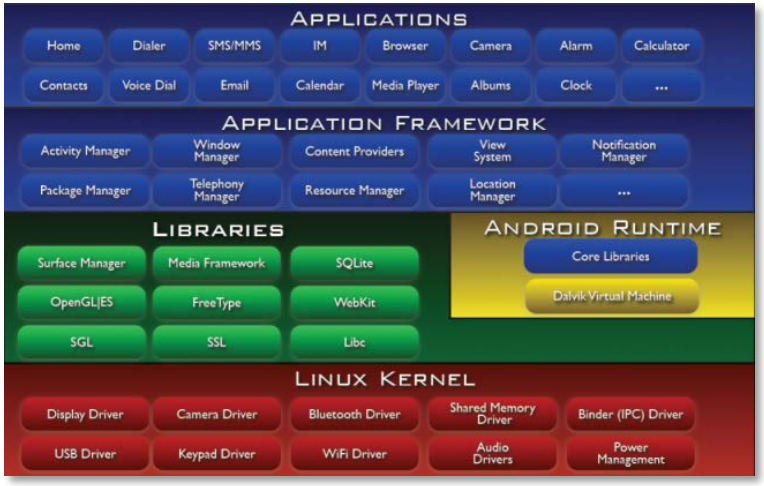
Focus is on applying POSA2 & GoF patterns – lots of C++/Java code examples

Course Syllabus

- **Section 3 – Applying patterns & frameworks to concurrent & networked software**
  - Service access & configuration
  - Event handling
  - Concurrency
  - Synchronization



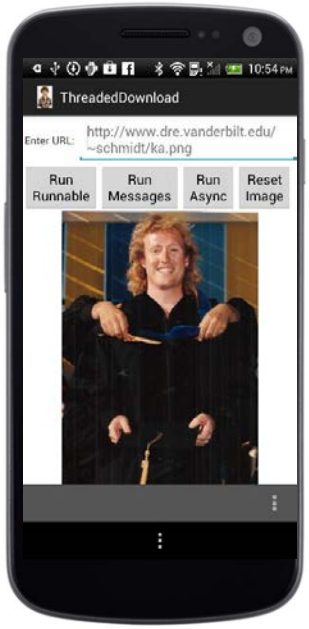


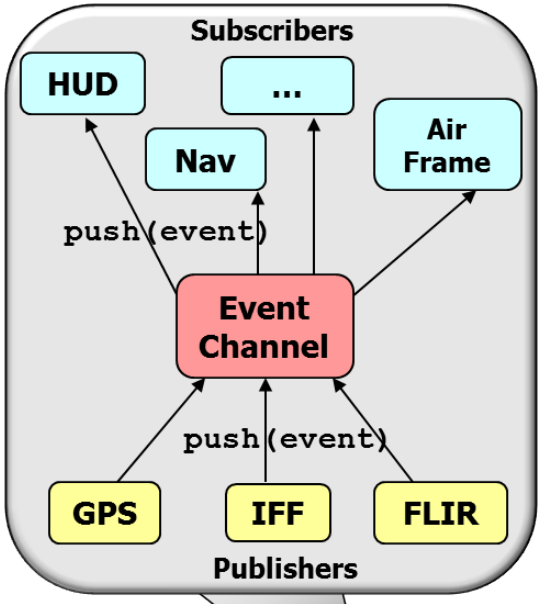


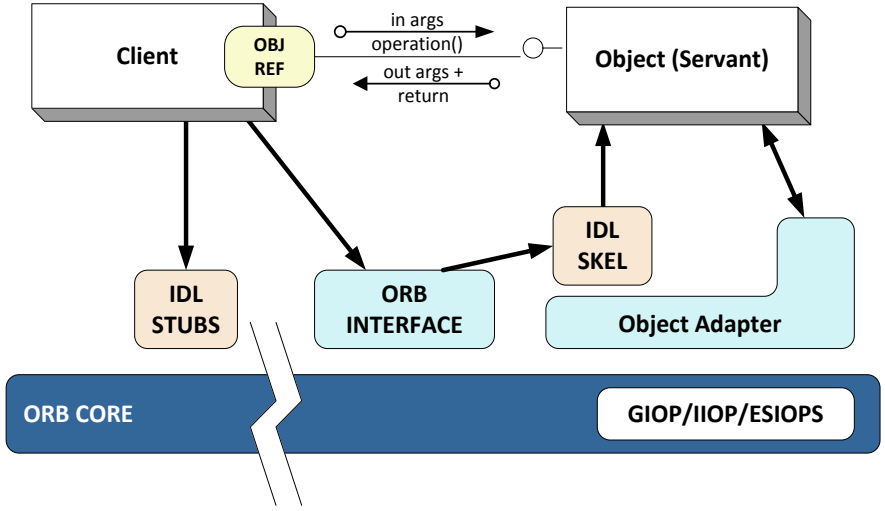
Focus is on a framework-based web server using ACE (C++) & Android (Java)

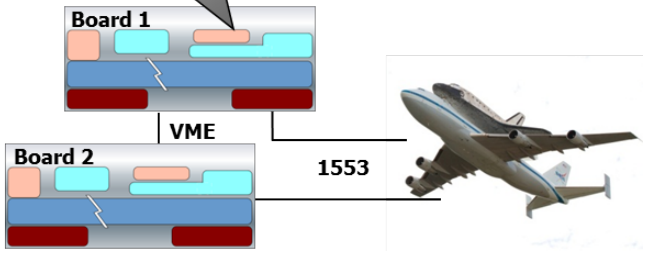
Course Syllabus

- **Section 3 – Applying patterns & frameworks to concurrent & networked software**
  - Service access & configuration
  - Event handling
  - Concurrency
  - Synchronization





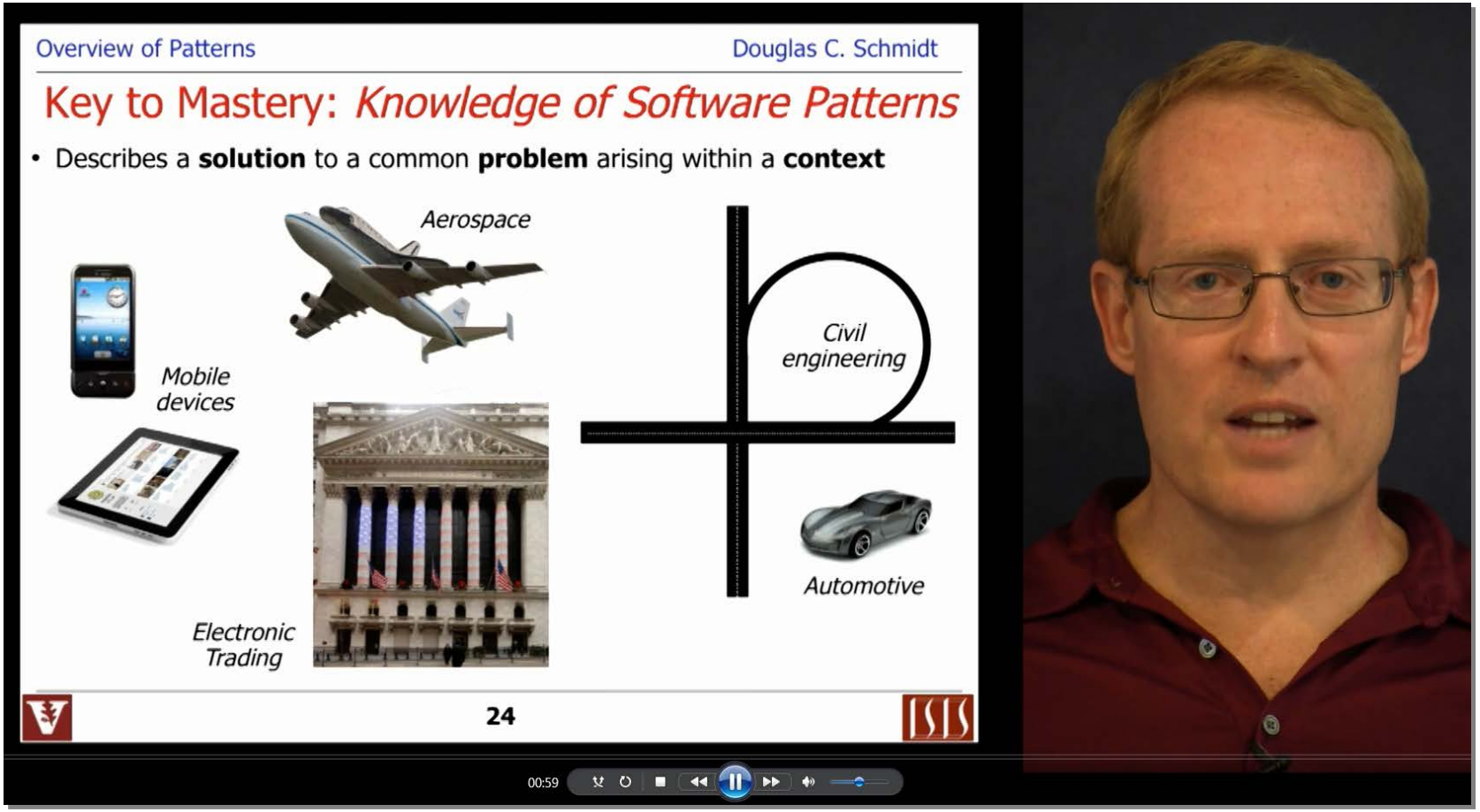




We also cover patterns & frameworks for other domains throughout the course

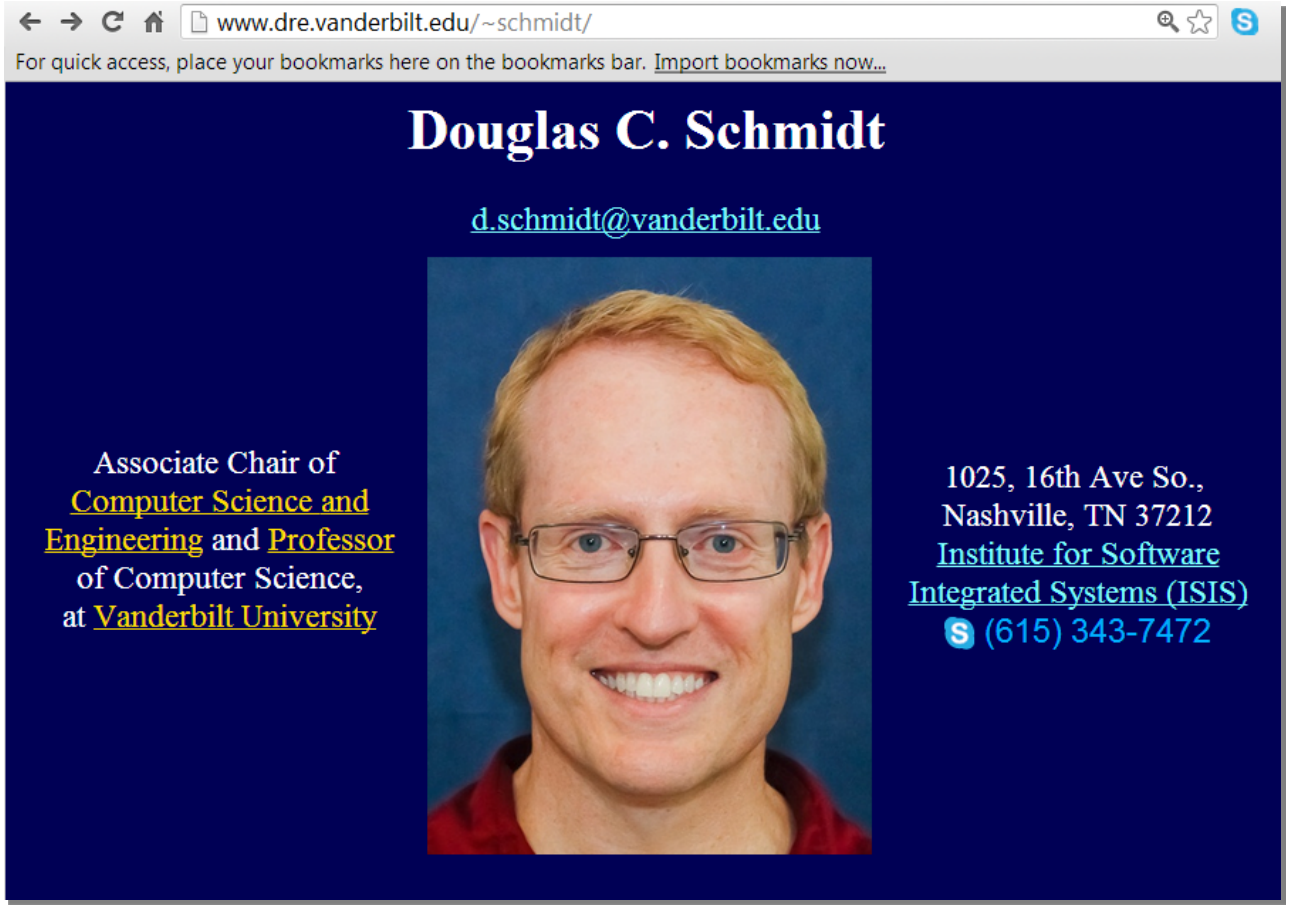
Summary

- Since there's a lot of material in this course—& you may not be familiar with all the prerequisites—I therefore encourage you to
  - Watch the videos multiple times



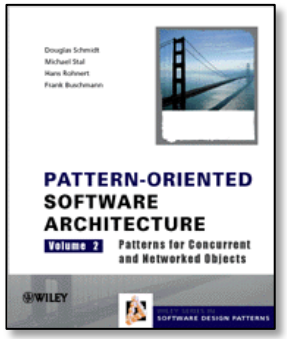
Summary

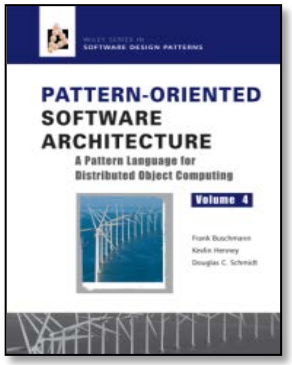
- Since there's a lot of material in this course—& you may not be familiar with all the prerequisites—I therefore encourage you to
  - Watch the videos multiple times
  - Read the various online resources at the provided URLs

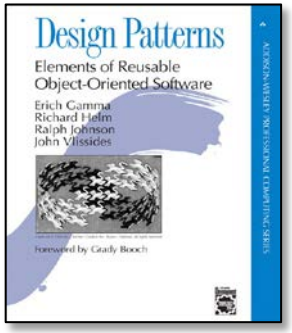


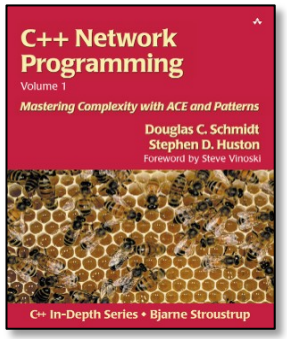
Summary

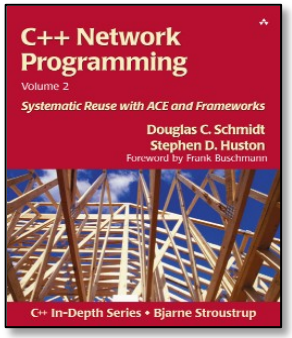
- Since there's a lot of material in this course—& you may not be familiar with all the prerequisites—I therefore encourage you to
  - Watch the videos multiple times
  - Read the various online resources at the provided URLs
  - Read the books that are recommended

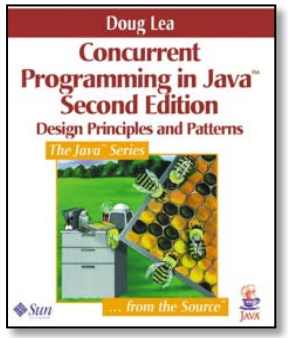












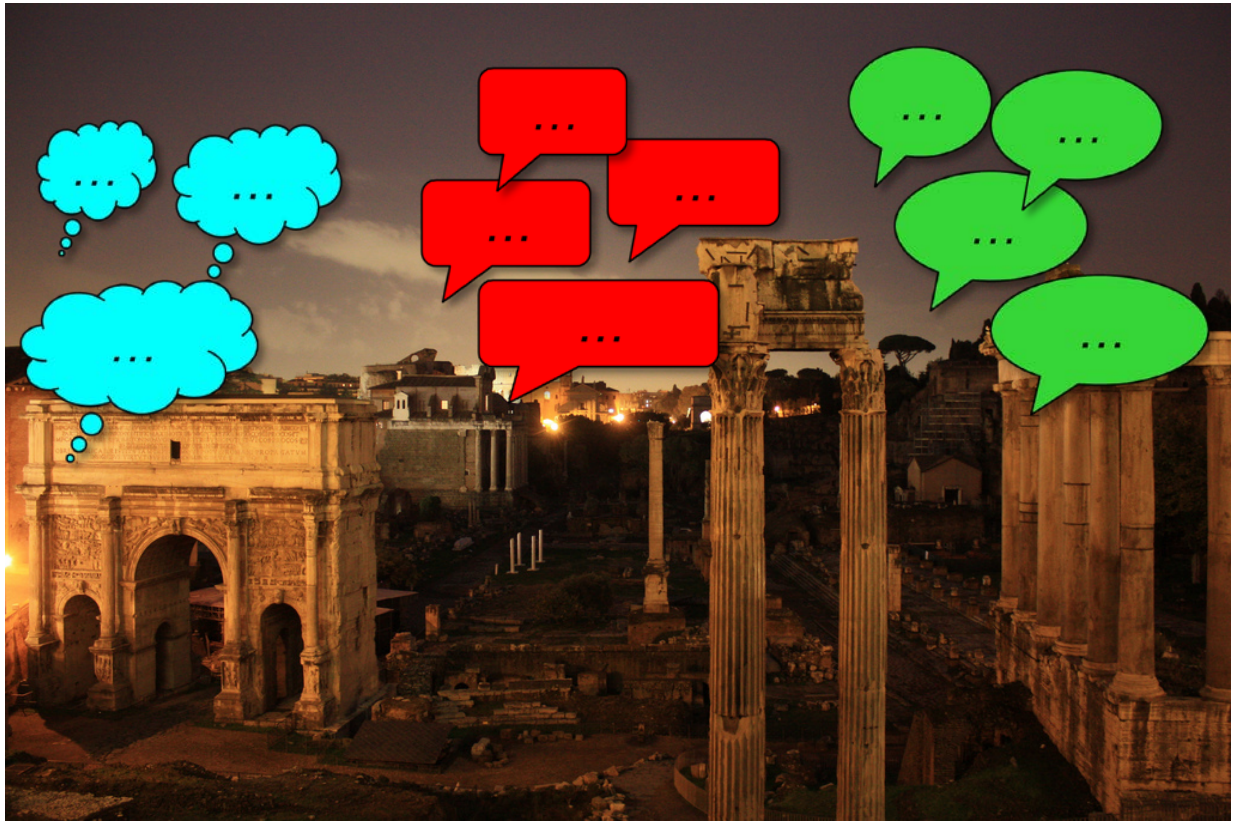
Summary

- Since there's a lot of material in this course—& you may not be familiar with all the prerequisites—I therefore encourage you to
  - Watch the videos multiple times
  - Read the various online resources at the provided URLs
  - Read the books that are recommended
  - Join a "meetup group"

www.meetup.com/Coursera

Summary

- Since there's a lot of material in this course—& you may not be familiar with all the prerequisites—I therefore encourage you to
  - Watch the videos multiple times
  - Read the various online resources at the provided URLs
  - Read the books that are recommended
  - Join a "meetup group"
  - Participate in the course online discussion forum



Summary

- Since there's a lot of material in this course—& you may not be familiar with all the prerequisites—I therefore encourage you to
  - Watch the videos multiple times
  - Read the various online resources at the provided URLs
  - Read the books that are recommended
  - Join a "meetup group"
  - Participate in the course online discussion forum
  - Apply to Vanderbilt University!

Summary

- Since there's a lot of material in this course—& you may not be familiar with all the prerequisites—I therefore encourage you to
  - Watch the videos multiple times
  - Read the various online resources at the provided URLs
  - Read the books that are recommended
  - Join a "meetup group"
  - Participate in the course online discussion forum
  - Apply to Vanderbilt University!

Summary

- Since there's a lot of material in this course—& you may not be familiar with all the prerequisites—I therefore encourage you to
  - Watch the videos multiple times
  - Read the various online resources at the provided URLs
  - Read the books that are recommended
  - Join a "meetup group"
  - Participate in the course online discussion forum
  - Apply to Vanderbilt University!