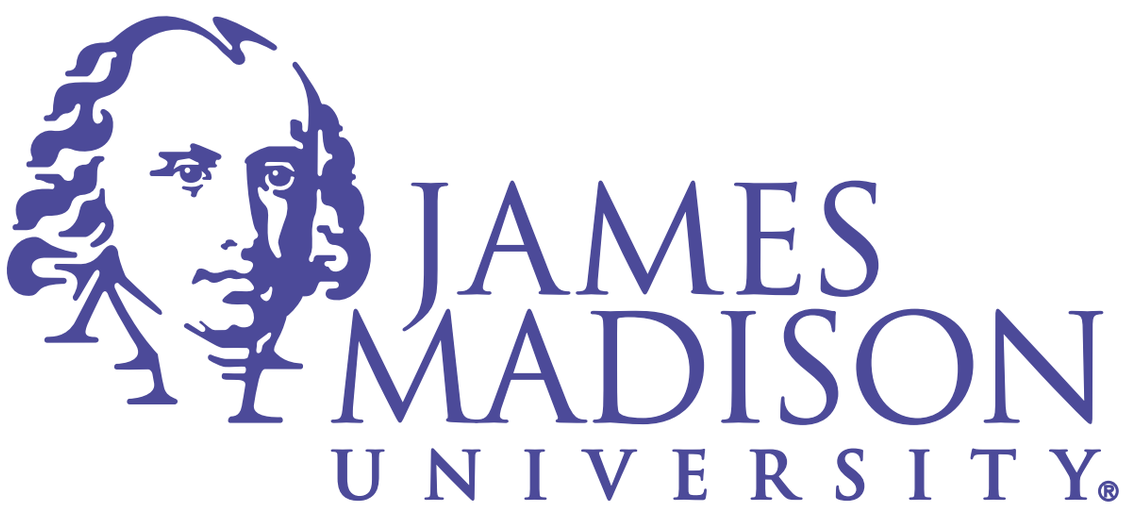
Backward Design

An Integrated Approach to a Systems Curriculum

Michael S. Kirkpatrick
Mohamed Aboutabl
David Bernstein
Sharon Simmons



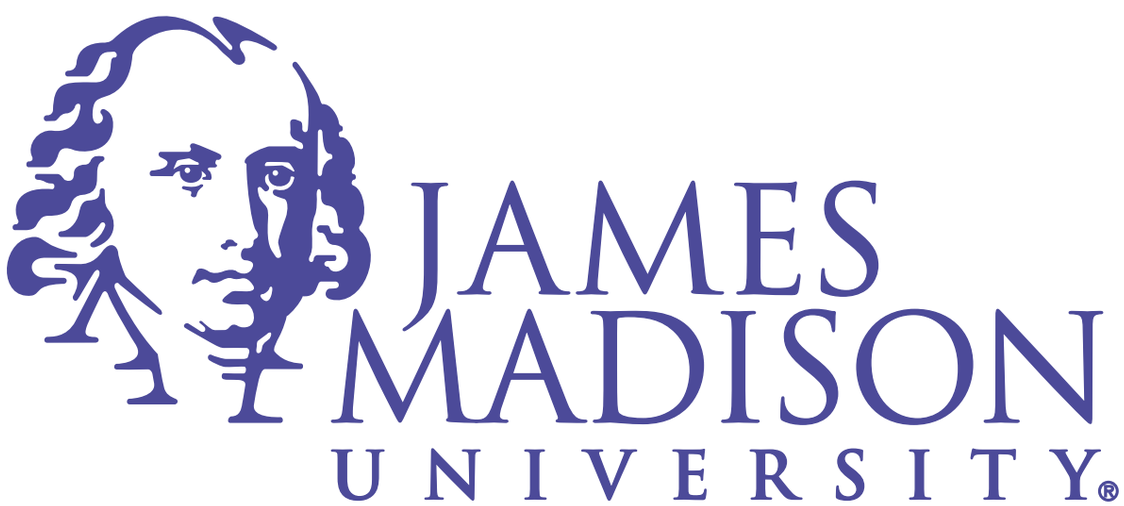
JAMES MADISON UNIVERSITY®

JAMES MADISON UNIVERSITY®

Backward Design

The Long and Rewarding Path to an ACM 2013 Systems Fundamentals Examplar

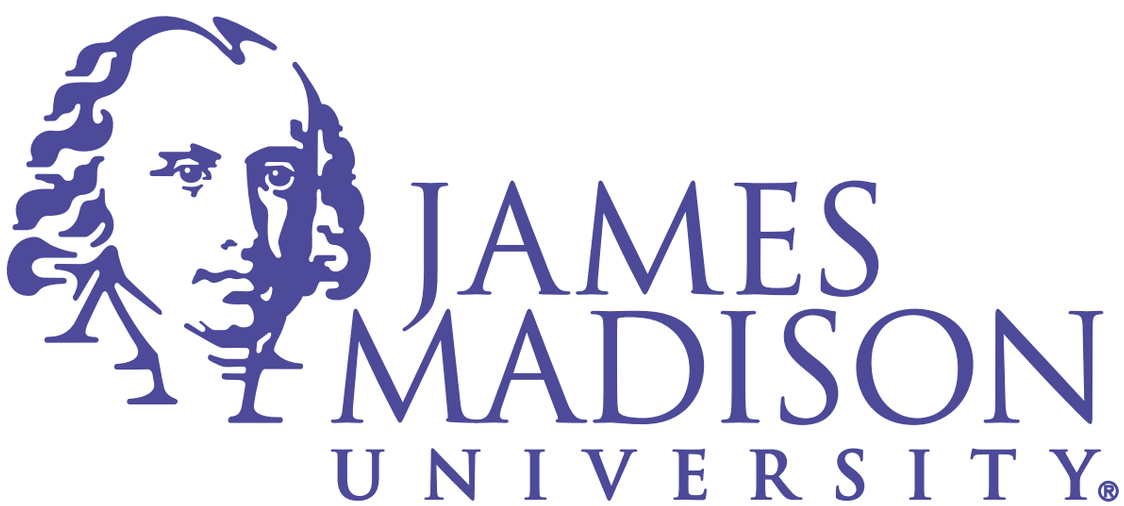
Michael S. Kirkpatrick
Mohamed Aboutabl
David Bernstein
Sharon Simmons



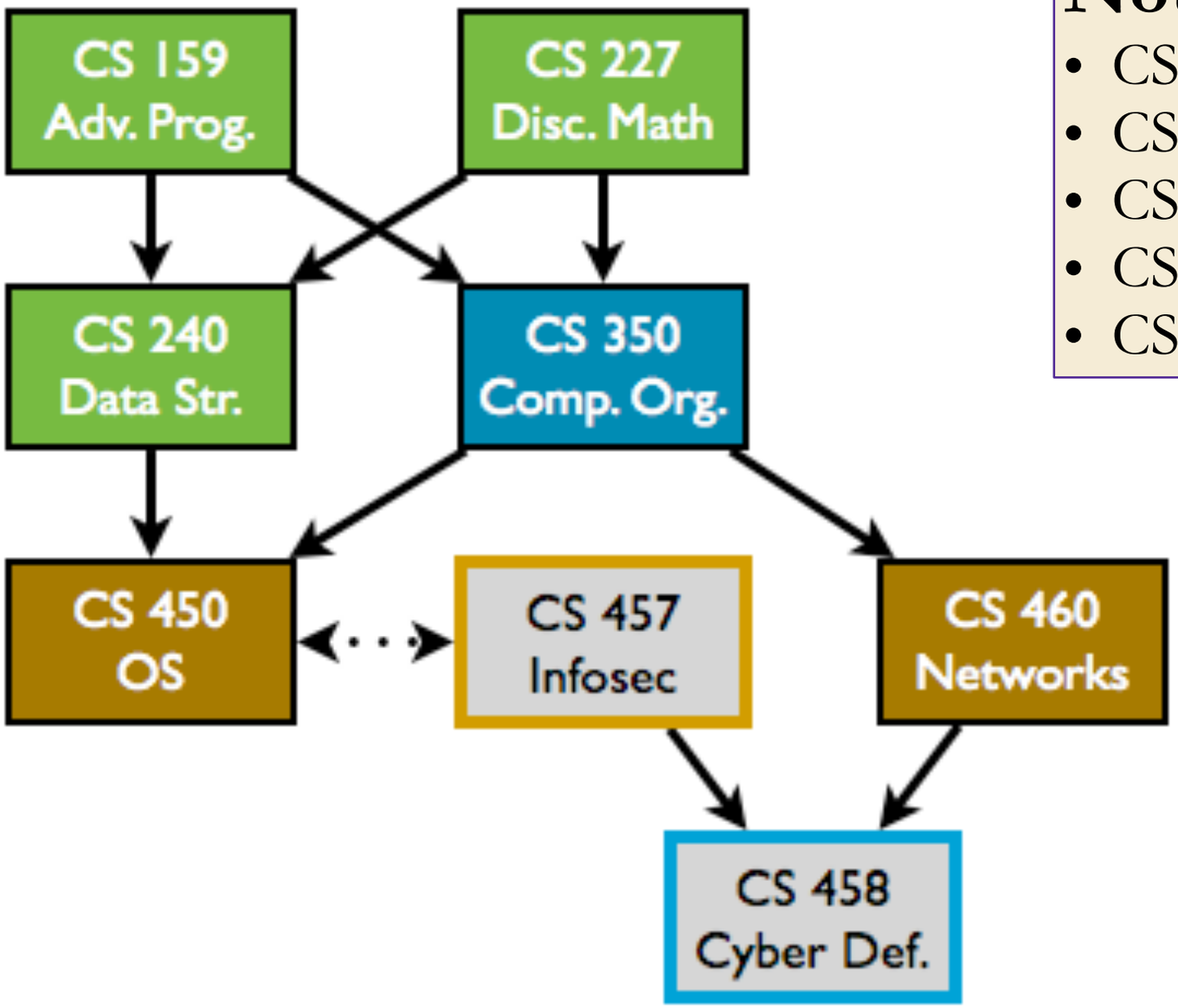
Backward Design

The Long and Rewarding Path to an ACM 2013 Systems Fundamentals Examplar (ish)

Michael S. Kirkpatrick
Mohamed Aboutabl
David Bernstein
Sharon Simmons



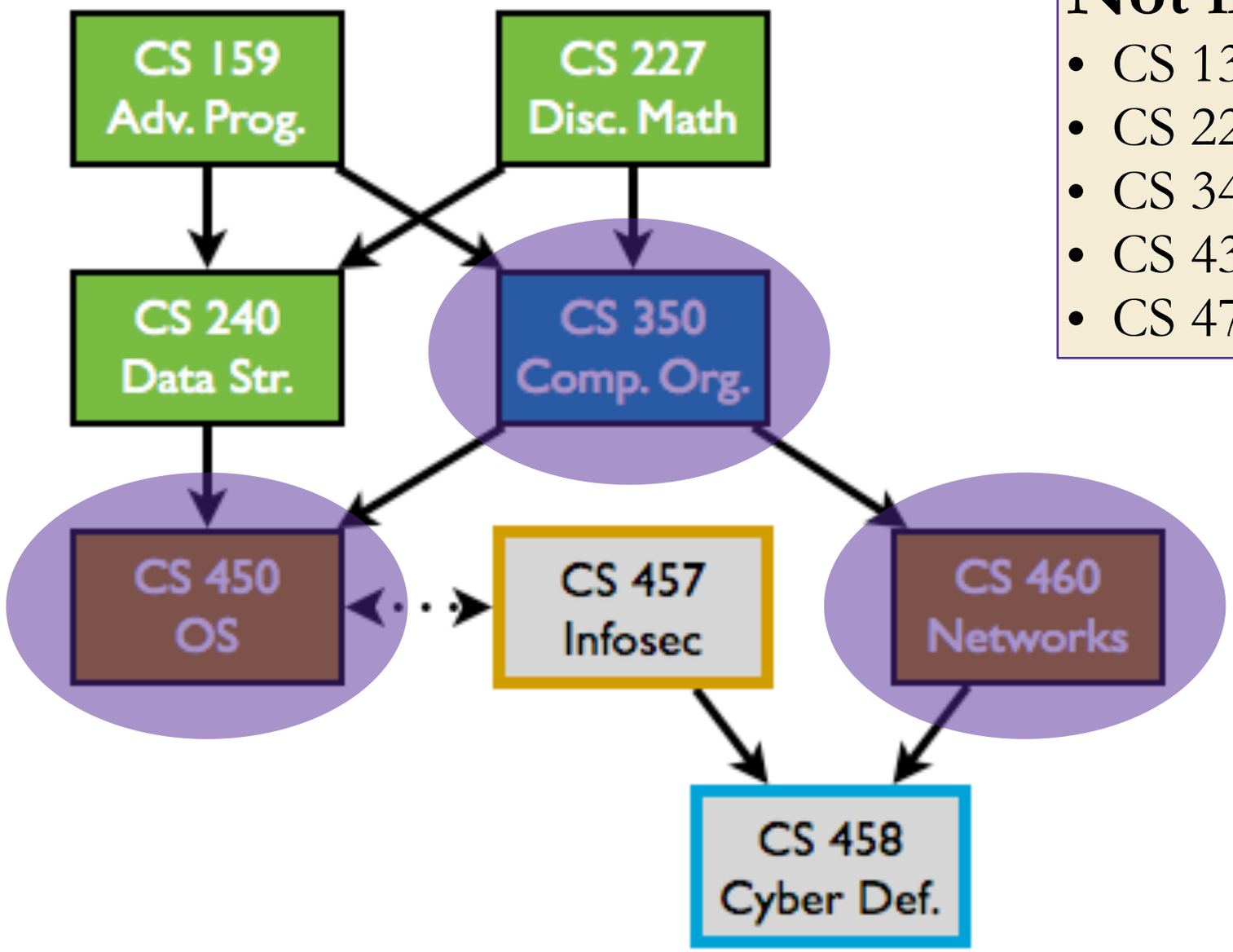
Context - how we got here



Not listed:

- CS 139: Programming Fundamentals
- CS 228: Discrete Structures II
- CS 345: Software Engineering
- CS 430: Programming Languages
- CS 474: Databases

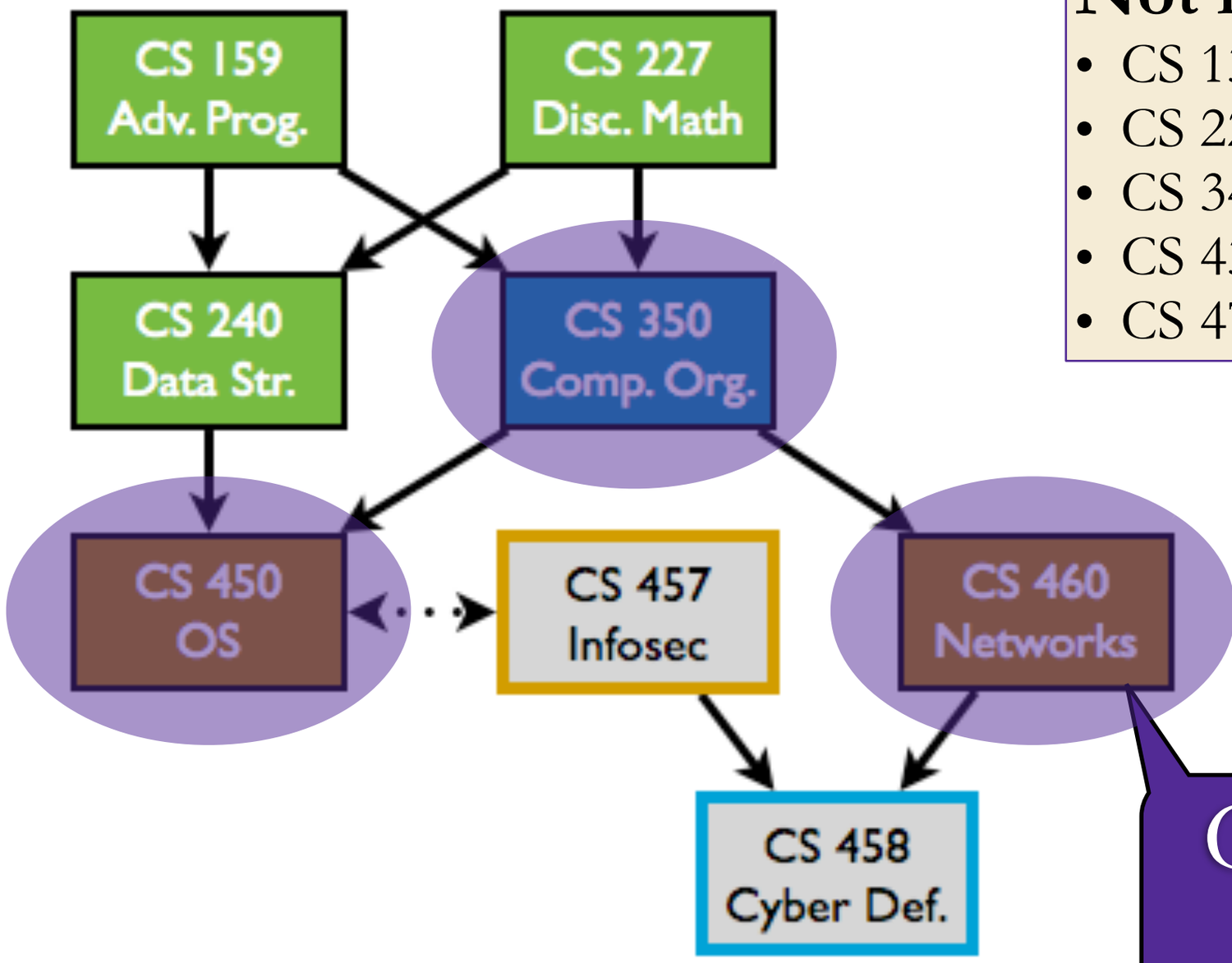
Context - how we got here



Not listed:

- CS 139: Programming Fundamentals
- CS 228: Discrete Structures II
- CS 345: Software Engineering
- CS 430: Programming Languages
- CS 474: Databases

Context - how we got here

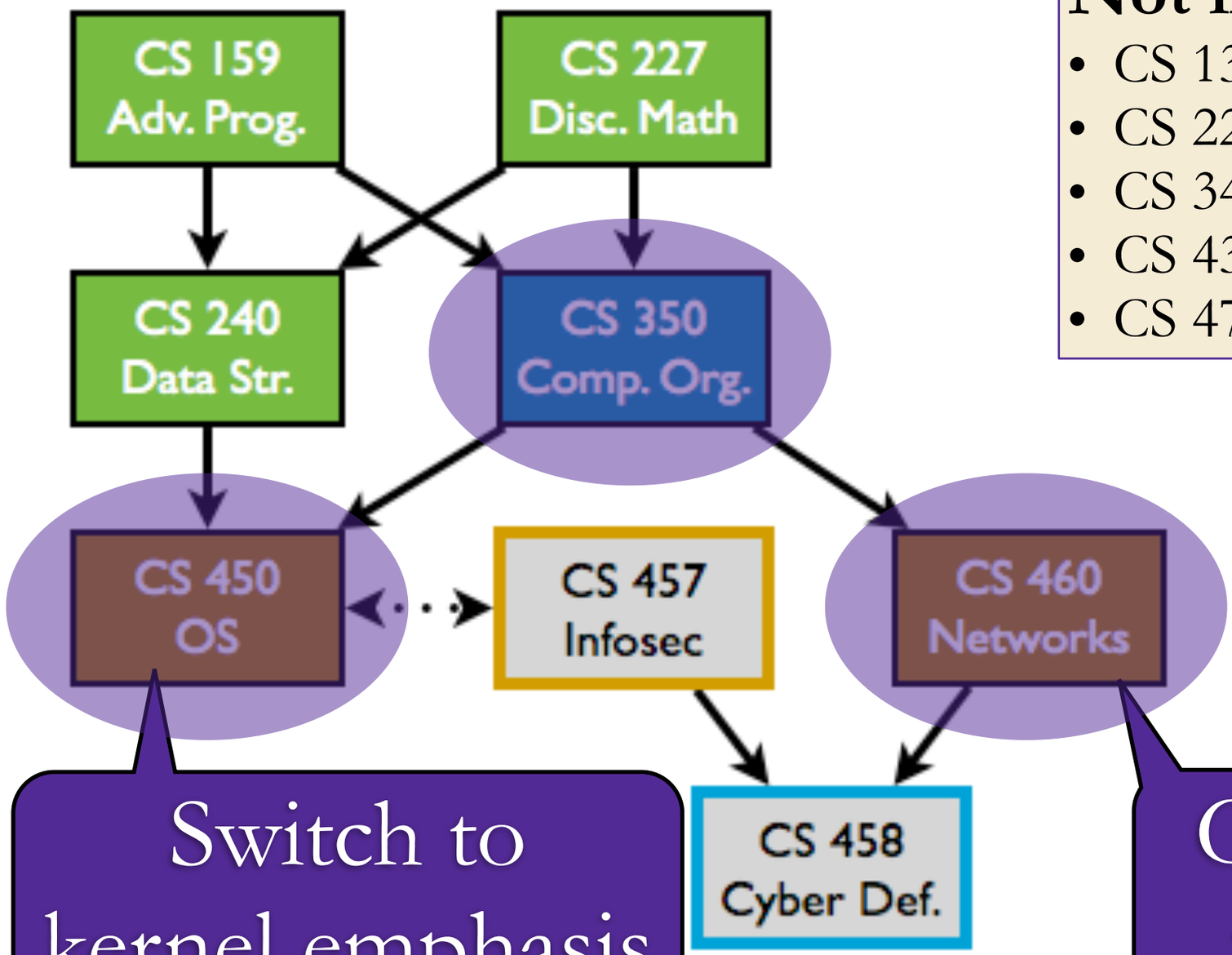


Not listed:

- CS 139: Programming Fundamentals
- CS 228: Discrete Structures II
- CS 345: Software Engineering
- CS 430: Programming Languages
- CS 474: Databases

Cross-listed with ISAT

Context - how we got here



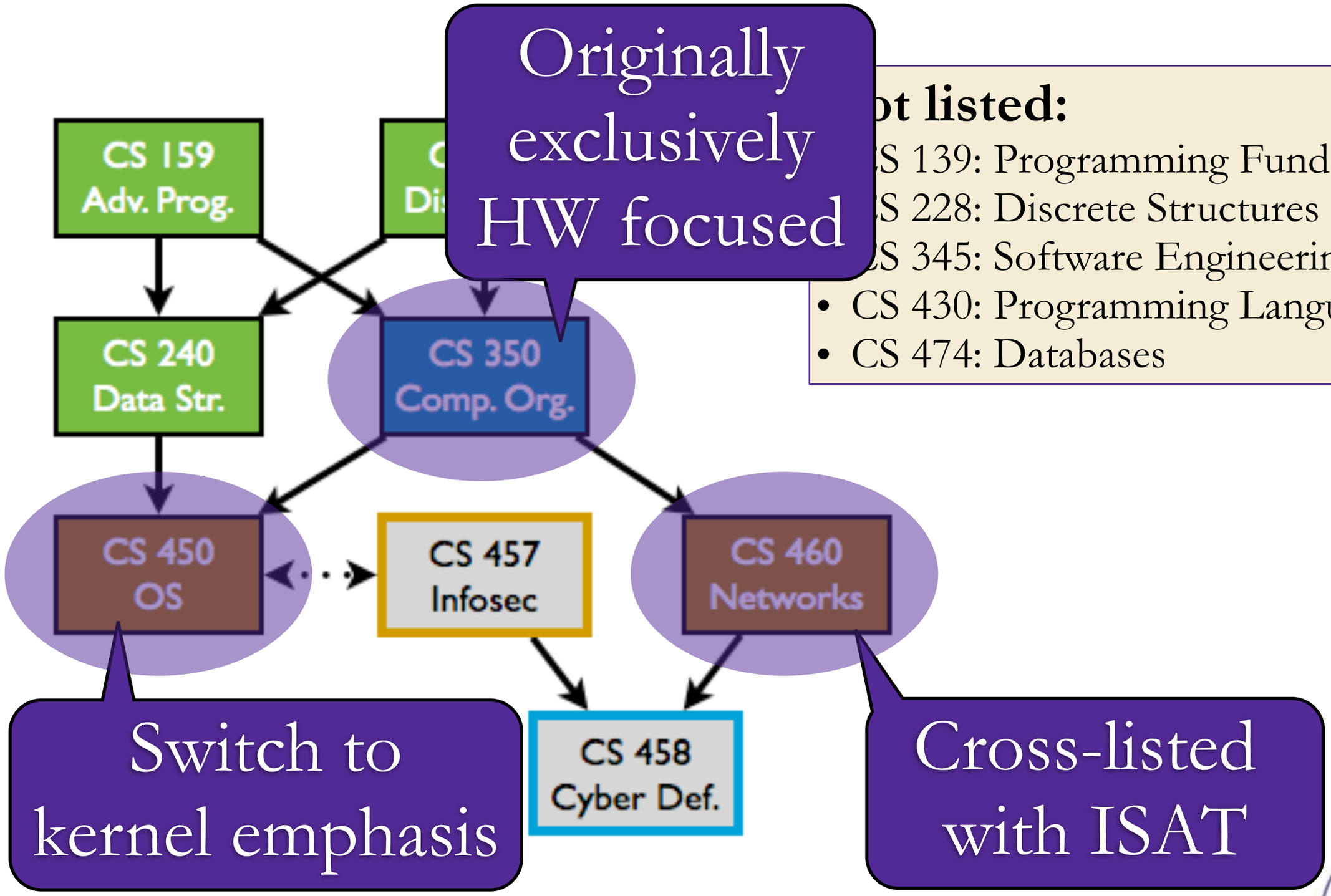
Not listed:

- CS 139: Programming Fundamentals
- CS 228: Discrete Structures II
- CS 345: Software Engineering
- CS 430: Programming Languages
- CS 474: Databases

Switch to kernel emphasis

Cross-listed with ISAT

Context - how we got here



ot listed:

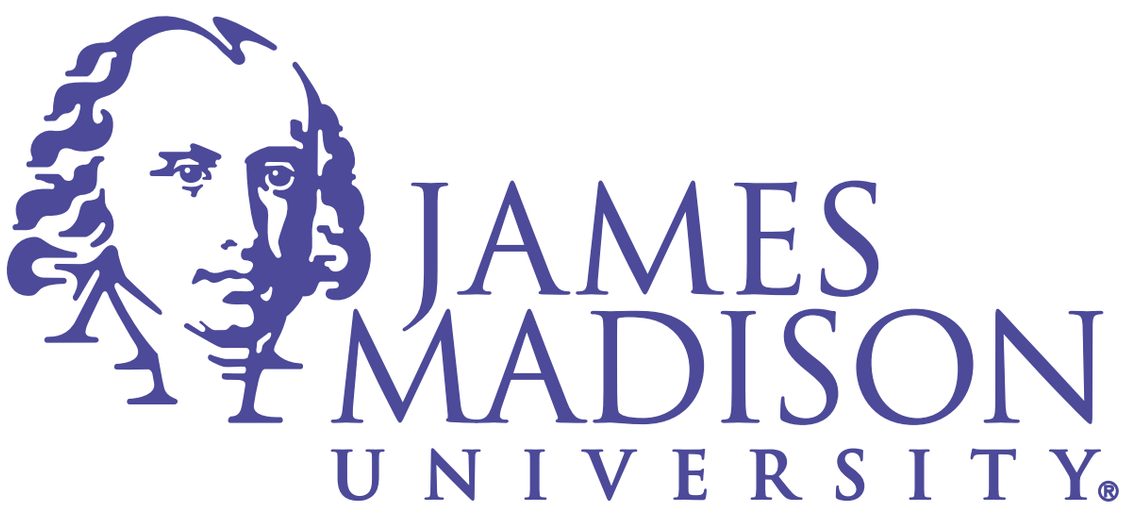
- S 139: Programming Fundamentals
- S 228: Discrete Structures II
- S 345: Software Engineering
- CS 430: Programming Languages
- CS 474: Databases

JAMES MADISON UNIVERSITY®

Backward Design

Fixing the Problems that Certain New Faculty Introduced when Hired

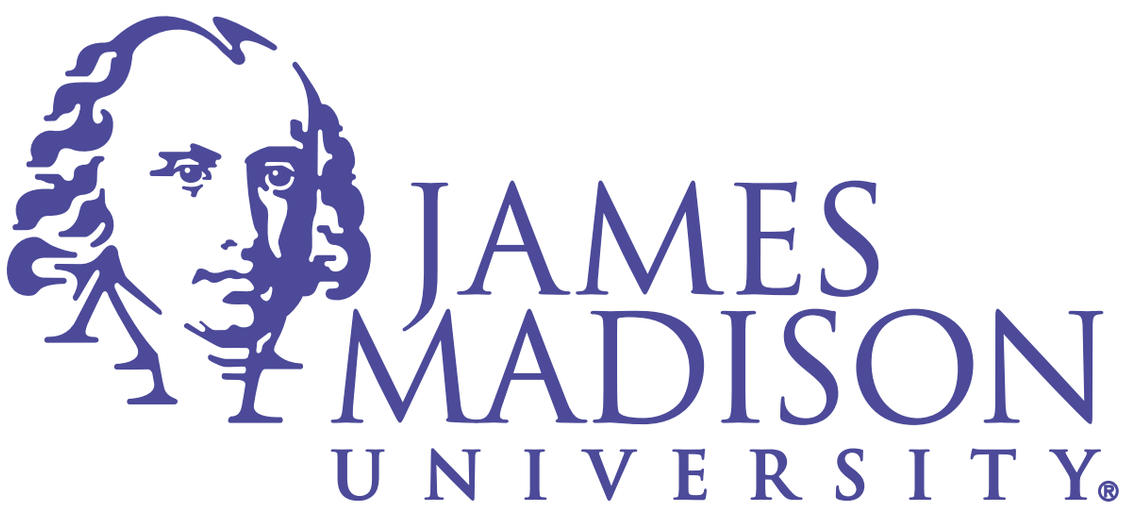
Michael S. Kirkpatrick
Mohamed Aboutabl
David Bernstein
Sharon Simmons



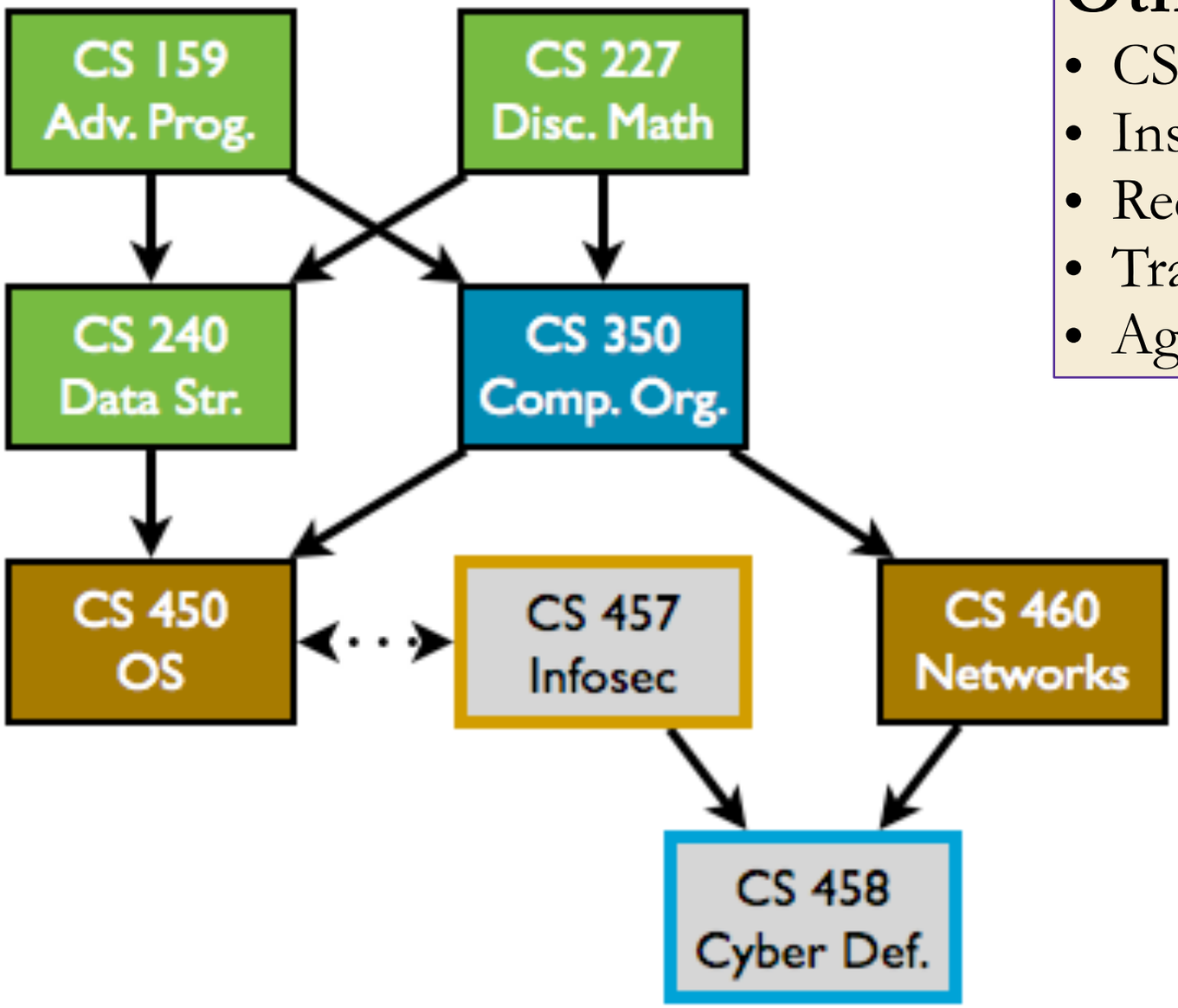
Backward Design

Fixing the Problems that Certain New Faculty Introduced when Hired

Michael S. Kirkpatrick
Mohamed Aboutabl
David Bernstein
Sharon Simmons



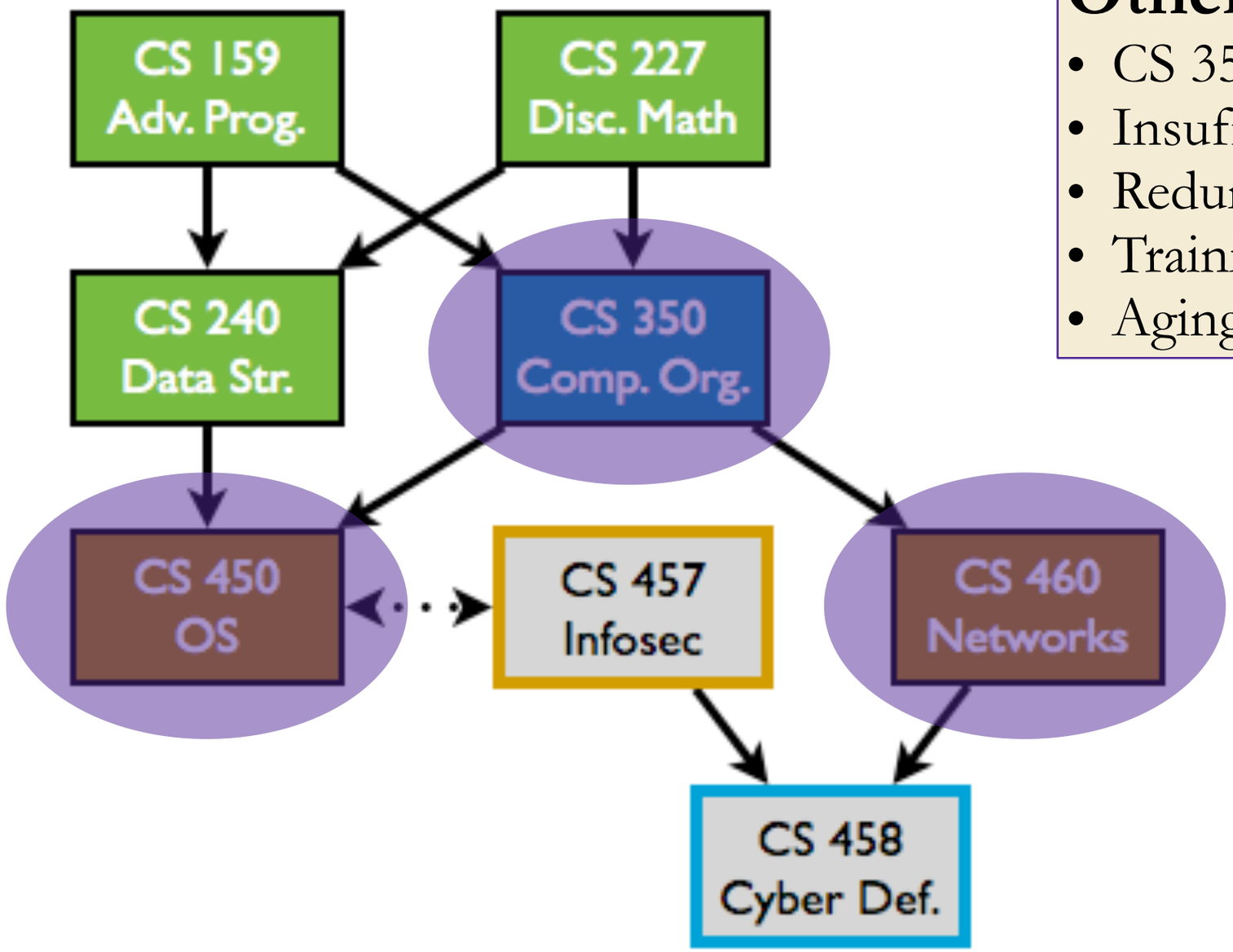
Context - how we got here



Other problems:

- CS 350 bottleneck
- Insufficient scaffolding
- Redundancies between siloes
- Training vs. education
- Aging curriculum

Context - how we got here



Other problems:

- CS 350 bottleneck
- Insufficient scaffolding
- Redundancies between siloes
- Training vs. education
- Aging curriculum

Backward Design

Identify situational factors

Backward Curricular Design

Identify situational factors

Backward Curricular Design

Identify situational factors

Starting points:

- What are our goals for **OUR** students?
- What are our strengths?
- What do we want to improve?
- What limitations and constraints exist?
- What is backward design?

Backward Curricular Design

Starting points:

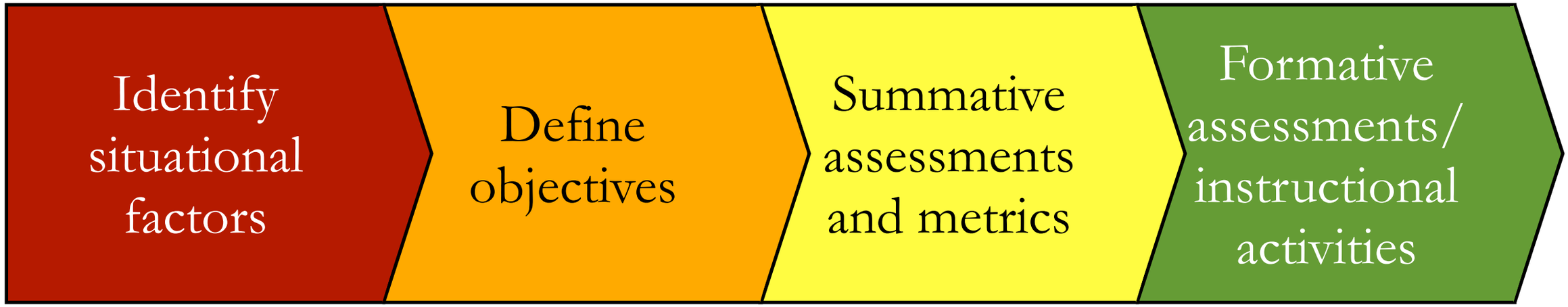
- What are our goals for **OUR** students?
- What are our strengths?
- What do we want to improve?
- What limitations and constraints exist?
- What is backward design?

Backward Curricular Design

Starting points:

- What are our goals for **OUR** students?
- What are our strengths?
- What do we want to improve?
- What limitations and constraints exist?
- What is backward design?

Backward Curricular Design



Starting points:

- What are our goals for **OUR** students?
- What are our strengths?
- What do we want to improve?
- What limitations and constraints exist?
- What is backward design?

Situational factors

Situational factors

Show-stopper constraints

- Three-section, two-prep teaching structure
- Virginia Community College System (VCCS) relationship
- Faculty interests and staffing

Situational factors

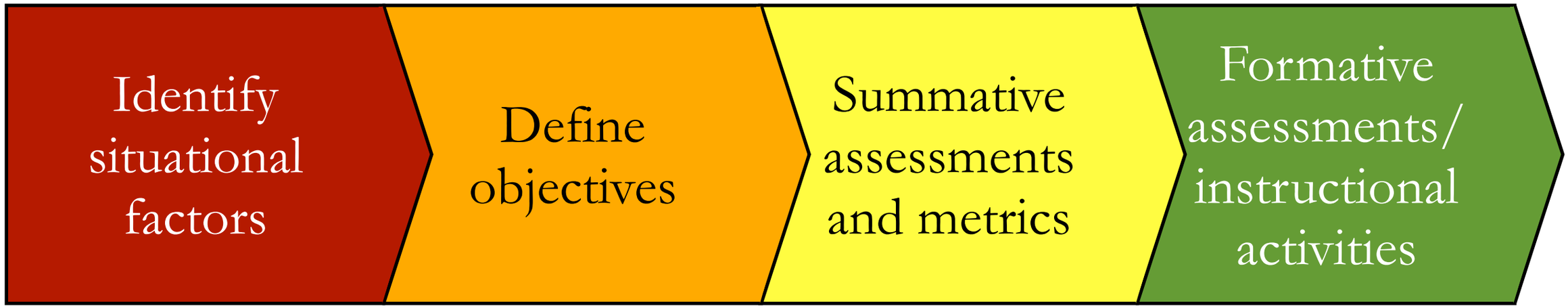
Show-stopper constraints

- Three-section, two-prep teaching structure
- Virginia Community College System (VCCS) relationship
- Faculty interests and staffing

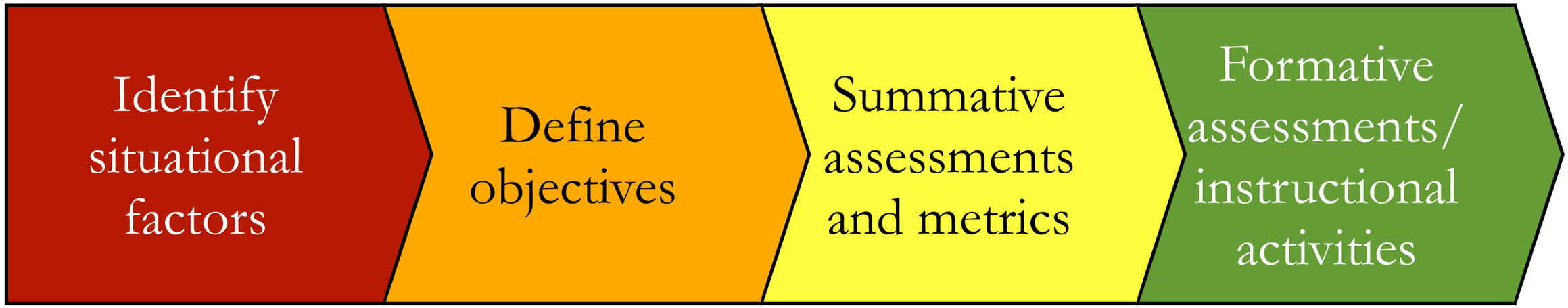
Other considerations

- Prolonged CS1 sequence (assumes no background)
- Infosec/Cyber-defense emphasis and strength
- Student career aspirations and proximity to DC
- Courses have reputations
- Liberal arts university - credit hour questions

Defining objectives



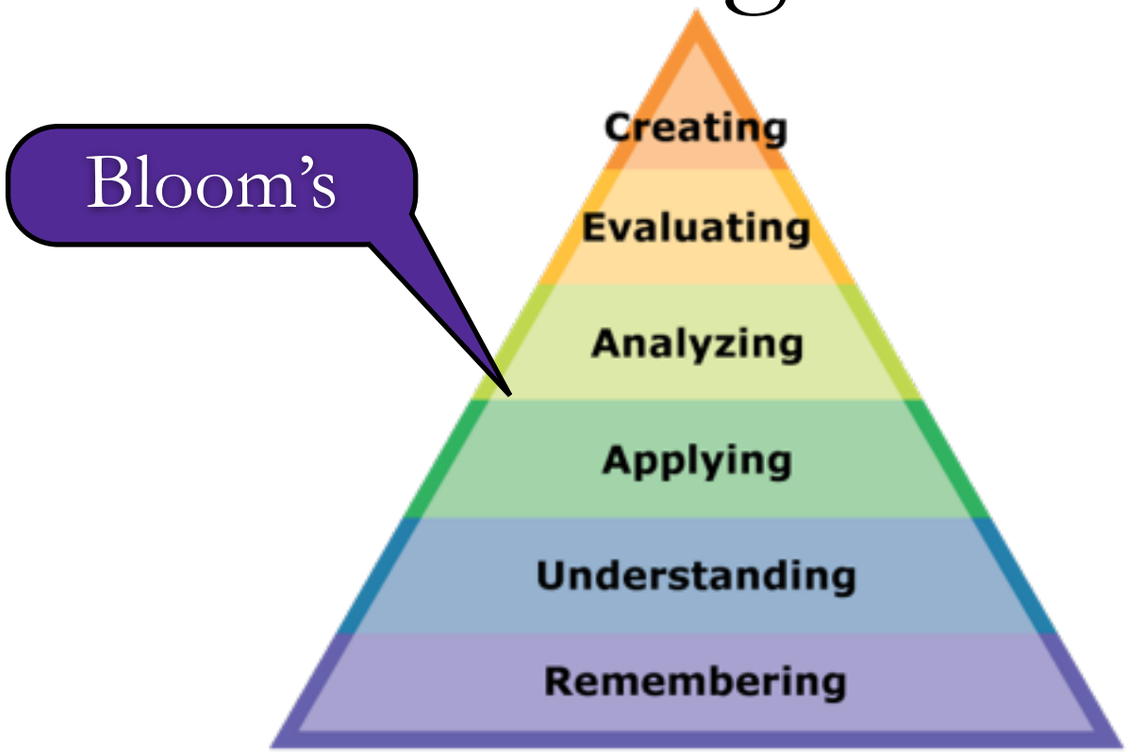
Defining objectives



Starting points for objectives:

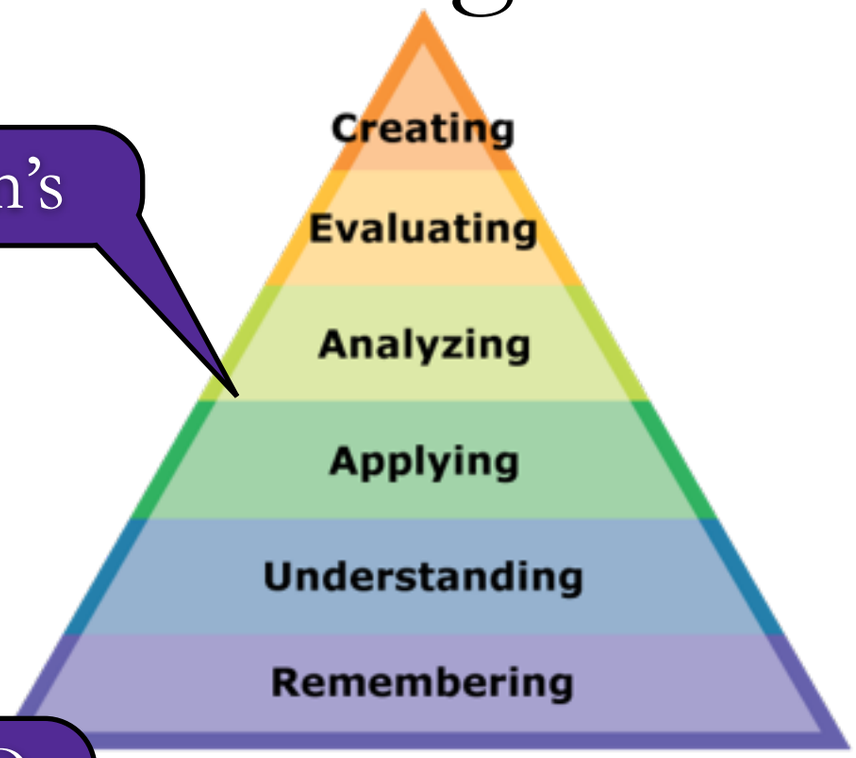
- What vision do we have for our students?
- How do we make our vision explicit and measurable?
- Where are our students starting from?
- What should they learn beyond just CS?

Taxonomies galore

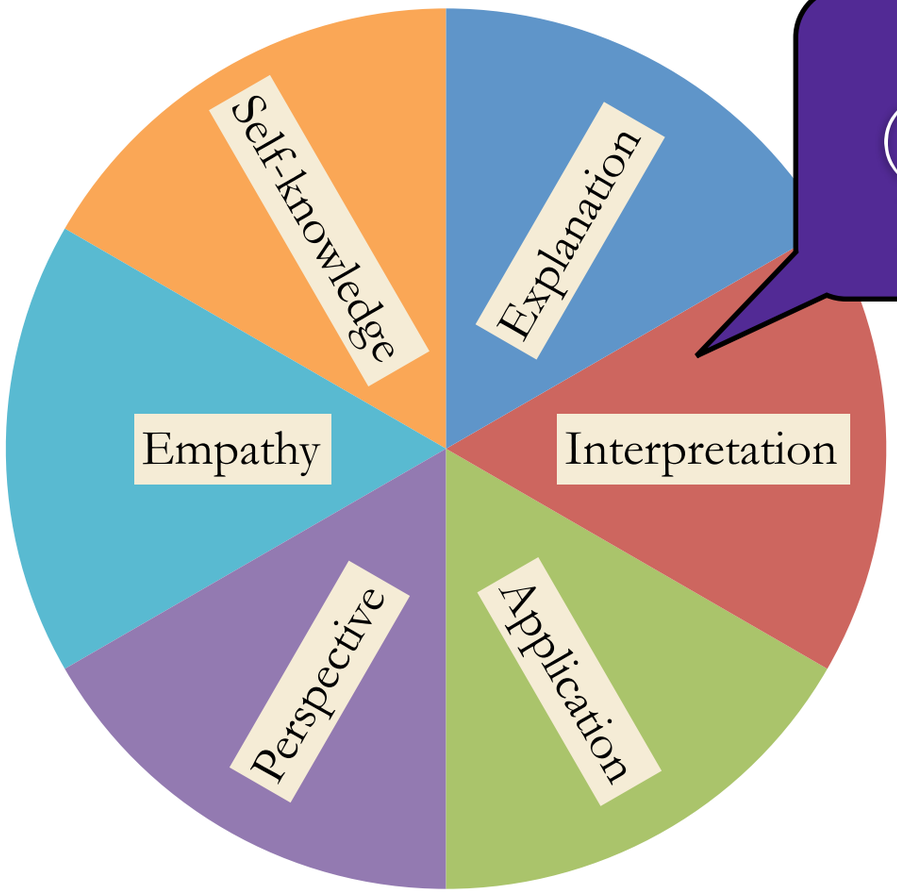


Taxonomies galore

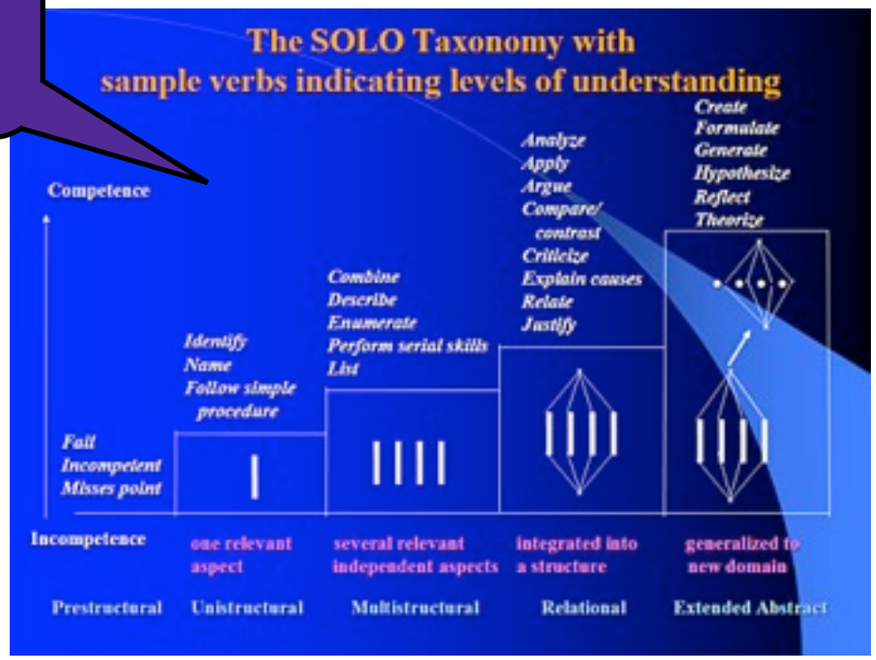
Bloom's



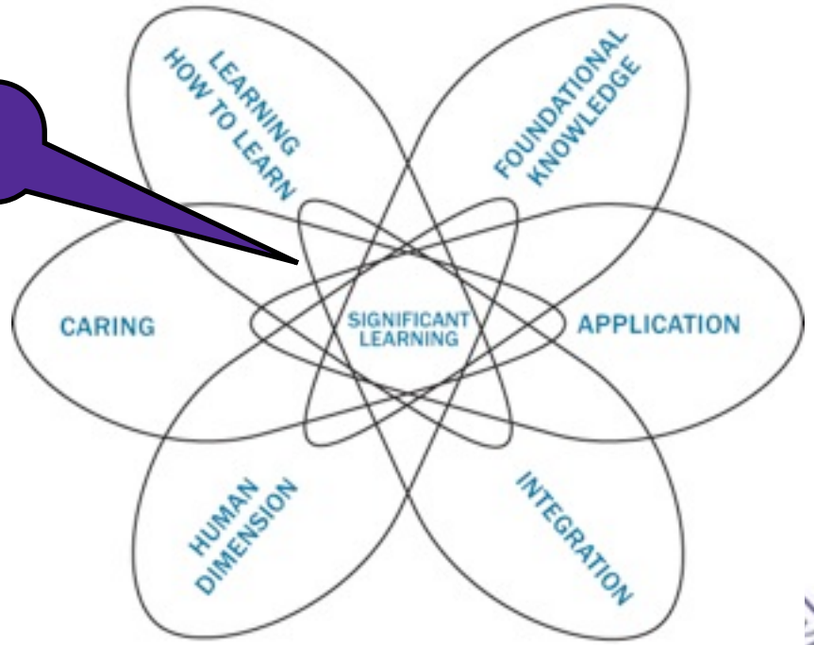
UbD (Wiggins & McTighe)



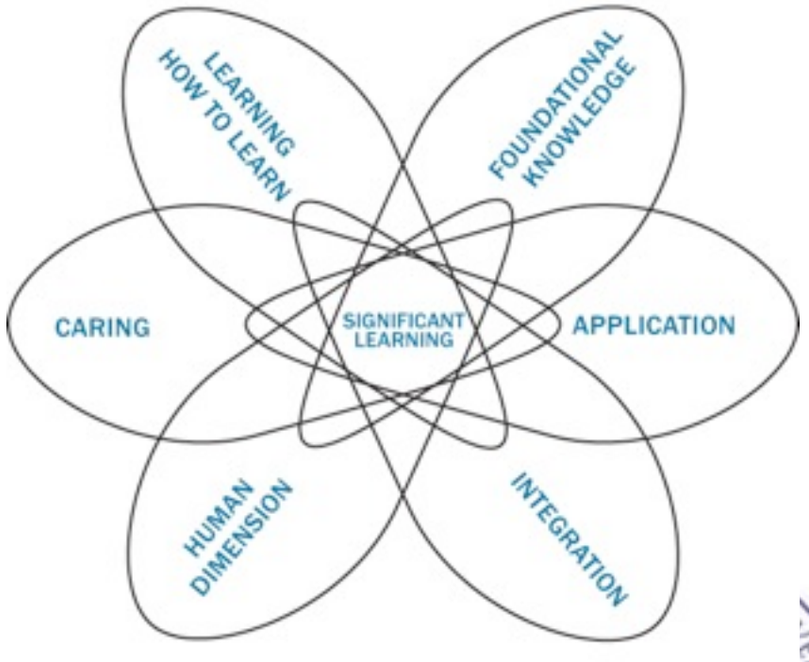
SOLO (Biggs)



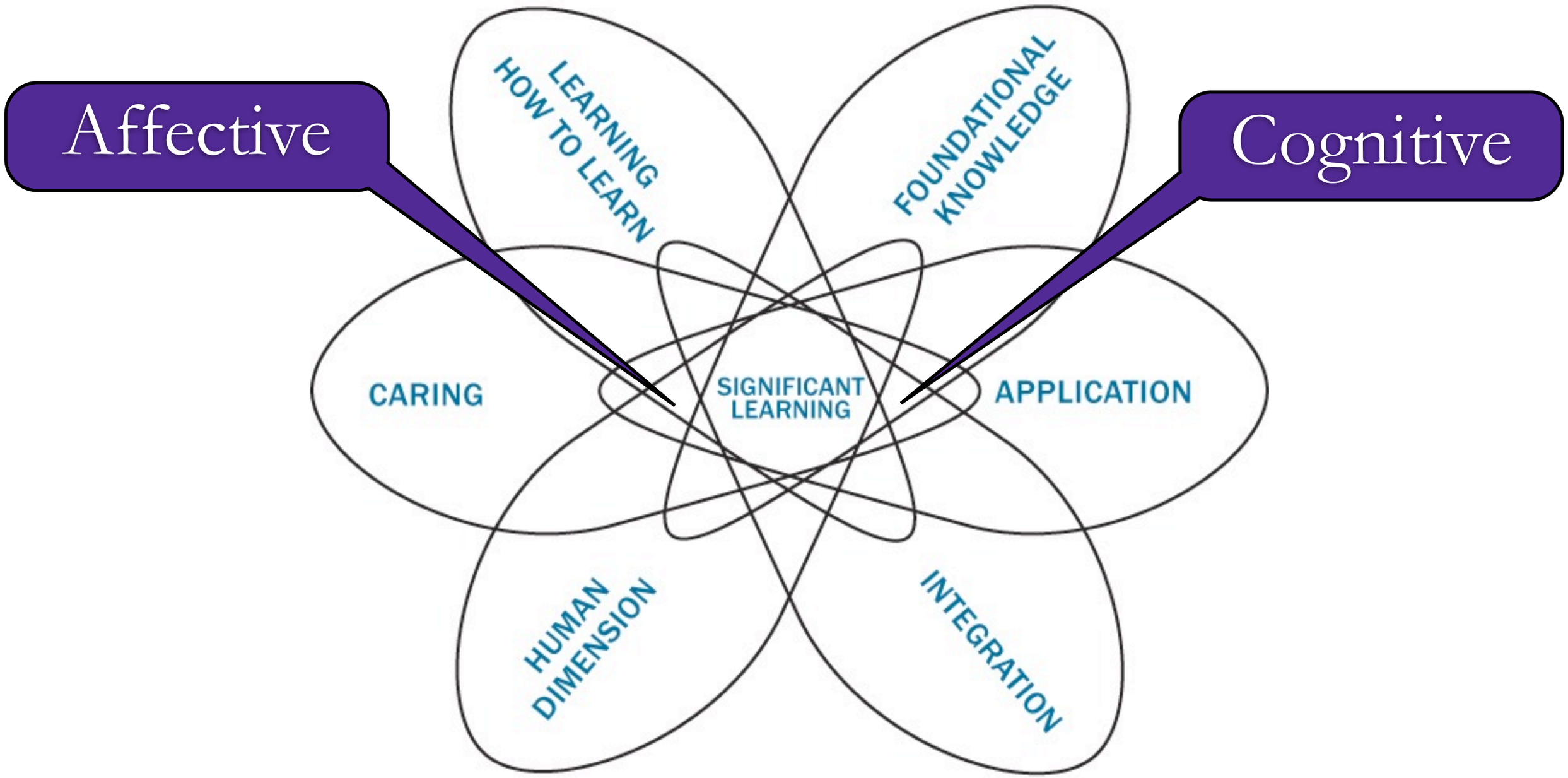
Fink



Taxonomies galore



Taxonomies galore



Defining learning outcomes

Foundational

1. What key information should our students understand and remember in the future?
2. What key ideas or themes are important for students to comprehend and describe?

Application

3. What kinds of thinking (critical, creative, practical) are important for our students to learn?
4. What software development skills do students need to gain?
5. What skills do our students need to learn about managing complex projects?

Integration

6. What connections should students recognize and make within the curriculum or to professional life?

Human dimension

7. What should students learn about themselves as software developers, users of technology, or as students?
8. What should students learn about understanding others and how they interact with others (including through developed software)?

Caring

9. What changes and values should students adopt in regard to interests or ideas?

Learning how to learn

10. What should students learn about how to work with computer systems?
11. What should students learn about becoming a self-directed learner?

Defining learning outcomes

Foundational

1. What key information should our students understand and remember in the future?
2. What key ideas or themes are important for students to comprehend and describe?

Application

3. What kinds of thinking (critical, creative, practical) are important for our students to learn?
4. What software development skills do students need to gain?
5. What skills do our students need to learn about managing complex projects?

Integration

6. What connections should students recognize and make within the curriculum or to professional life?

Human dimension

7. What should students learn about themselves as software developers, users of technology, or as students?
8. What should students learn about understanding others and how they interact with others (including through developed software)?

Caring

9. What changes and values should students adopt in regard to interests or ideas?

Learning how to learn

10. What should students learn about how to work with computer systems?
11. What should students learn about becoming a self-directed learner?

Systems core themes

To reflect trends within the CS field, the JMU CS systems core courses should strive to produce the following **characteristics of CS graduates**:

- technical understanding of computer and network systems
- appreciation of the interplay between theory and practice
- system-level perspective (thinking in levels of abstraction)
- understanding of how to identify and evaluate trade-offs in design and implementation
- ability to identify common problem patterns and apply appropriate solutions
- demonstrable experience with large software projects
- commitment to individual skill development, such as learning new languages
- commitment to professional responsibility
- understand the differences between systems and application programming
- understand the hardware characteristics that dictate the requirements/features of the controlling software (e.g., sampling frequency, signal propagation delays, arithmetic precision)

Systems core themes

To reflect trends within the CS field, the JMU CS systems core courses should strive to produce the following **characteristics of CS graduates**:

- technical understanding of computer and network systems
- appreciation of the interplay between theory and practice
- system-level perspective (thinking in levels of abstraction)
- understanding of how to identify and evaluate trade-offs in design and implementation
- ability to identify common problem patterns and apply appropriate solutions
- demonstrable experience with large software projects
- commitment to individual skill development, such as learning new languages
- commitment to professional responsibility
- understand the differences between systems and application programming
- understand the hardware characteristics that dictate the requirements/features of the controlling software (e.g., sampling frequency, signal propagation delays, arithmetic precision)

Systems core themes

In support of this goal, the following **principles** should guide the systems curriculum:

- open-ended programming assignments that require analysis, design, and testing
- opportunities to apply systems concepts to realistic problems
- a combination of individual- and group-based projects
- experience both reading and writing code
- exposure to standard industry tools and techniques where appropriate
- exploration of the design considerations underlying existing systems software
- emphasis on developing skills for independent learning
- flexible curriculum that serves the needs of students with varying technical talents
- appropriate coverage of fundamental computer and networking concepts

Systems core themes

In support of this goal, the following **principles** should guide the systems curriculum:

- open-ended programming assignments that require analysis, design, and testing
- opportunities to apply systems concepts to realistic problems
- a combination of individual- and group-based projects
- experience both reading and writing code
- exposure to standard industry tools and techniques where appropriate
- exploration of the design considerations underlying existing systems software
- emphasis on developing skills for independent learning
- flexible curriculum that serves the needs of students with varying technical talents
- appropriate coverage of fundamental computer and networking concepts

Systems core themes

Every JMU CS student who satisfactorily completes all required systems courses should be able to:

- Summarize the technical foundations of how software executes on hardware, how parallel and distributed software communicate, what patterns and structures are used to construct systems and concurrent programs, and what layered abstractions support modern computing systems.
- Explain how bits can represent information, how instructions and communication can be seen as a sequence of state transitions, how mathematical models can describe system behavior, and how reactive programming practices used in systems programming differs from other approaches.
- Read and understand existing protocols, critically evaluate existing protocols, and select appropriate protocols.
- Describe the ways in which information can be exchanged between different parts of an existing system, select appropriate ways to exchange information in proposed systems, and implement systems that exchange information.
- Describe the architectural style/high-level design of a system using appropriate terminology, and evaluate the architectural style/high-level design of existing and proposed systems.
- ...

Systems core themes

Every JMU CS student who satisfactorily completes all required systems courses should be able to:

- Summarize the technical foundations of how software executes on hardware, how parallel and distributed software communicate, what patterns and structures are used to construct systems and concurrent programs, and what layered abstractions support modern computing systems.
- Explain how bits can re[illegible]ons and communication can be seen as a sequence of st[illegible] models can describe system behavior, and how react[illegible] systems programming differs from other approaches.
- Read and understand existing protocols, critically evaluate existing protocols, and select appropriate protocols.
- Describe the ways in which information can be exchanged between different parts of an existing system, select appropriate ways to exchange information in proposed systems, and implement systems that exchange information.
- Describe the architectural style/high-level design of a system using appropriate terminology, and evaluate the architectural style/high-level design of existing and proposed systems.
- ...

Core Tier 1 vs.
Core Tier 2

Systems core themes

Enduring Understandings

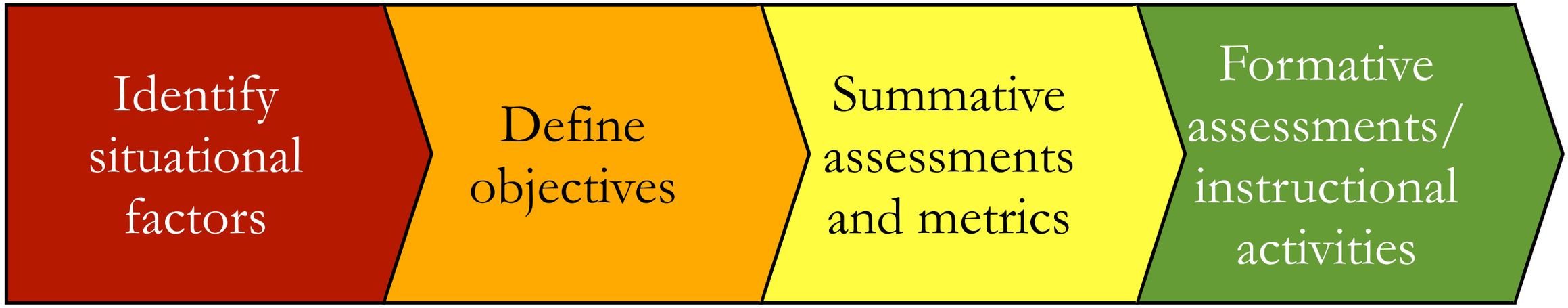
- Information = Bits + Context
- The Semiotics of Systems
- System Design Involves Tradeoffs
- Systems Programs are a Foundation
- Digital/Discrete Models of an Analog/Continuous World
- The Computer as Other
- Appearances can be Deceiving
- From Abstract Specifications to Complete Implementations
- Communication has Multiple Dimensions
- Resources must be Shared
- Reliability can be Elusive
- Use the Right Tool for the Job

Systems core themes

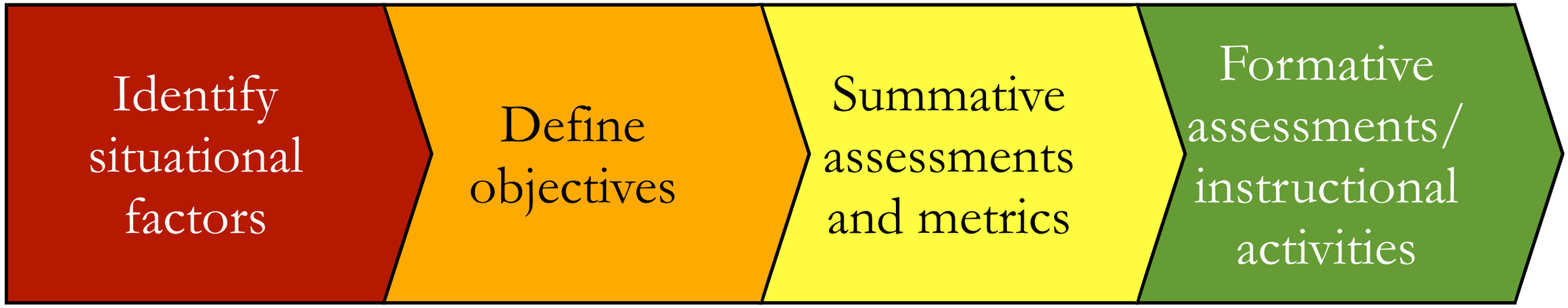
Enduring Understandings

- Information = Bits + Context
- The Semiotics of Systems
- System Design Involves Tradeoffs
- Systems Programs are a Foundation
- Digital/Discrete Models of an Analog/Continuous World
- The Computer as Other
- Appearances can be Deceiving
- From Abstract Specifications to Complete Implementations
- Communication has Multiple Dimensions
- Resources must be Shared
- Reliability can be Elusive
- Use the Right Tool for the Job

Assessment strategies



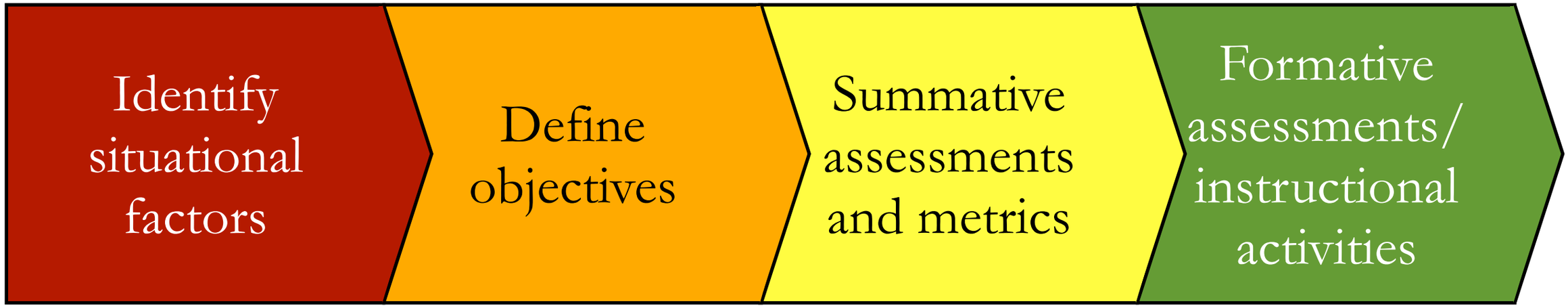
Assessment strategies



Summative assessments

- Question bank for exams
  - Good for program assessment, too
- Common projects - 3-phase semester-long system building
  - Shared rubric encourages consistent expectations

Assessment strategies



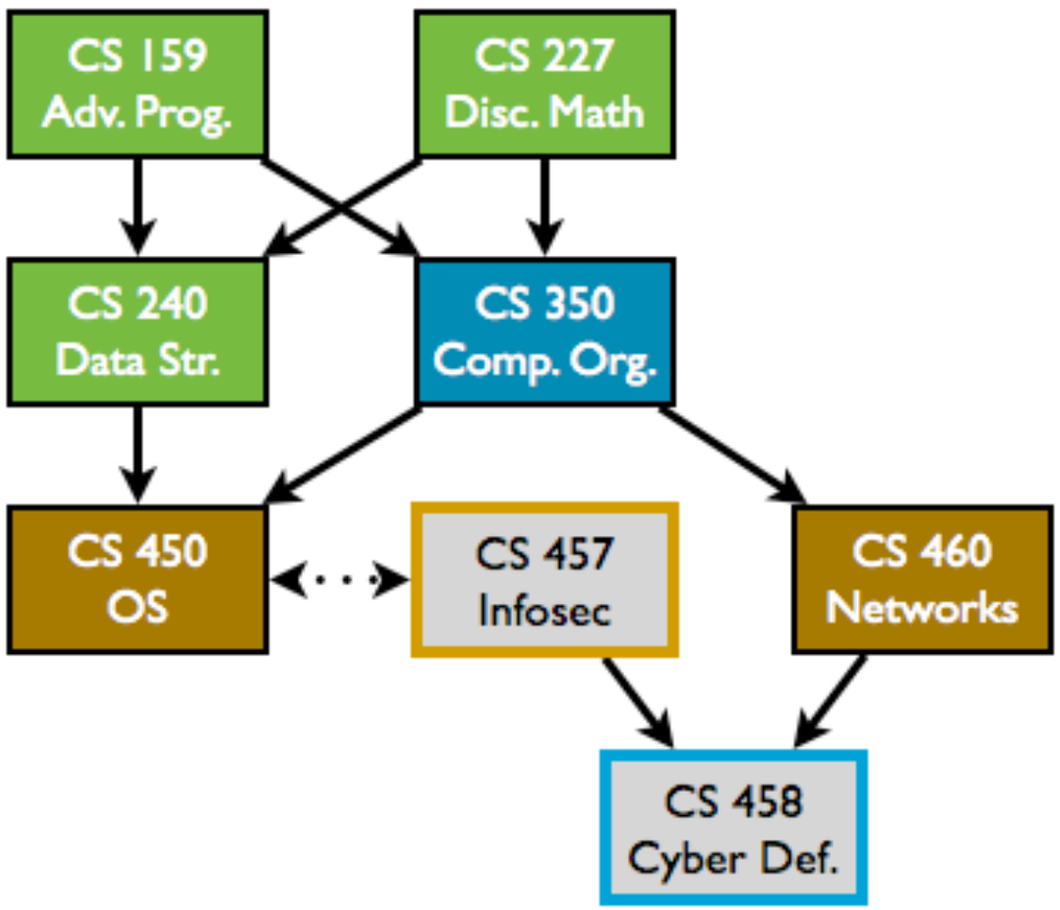
Summative assessments

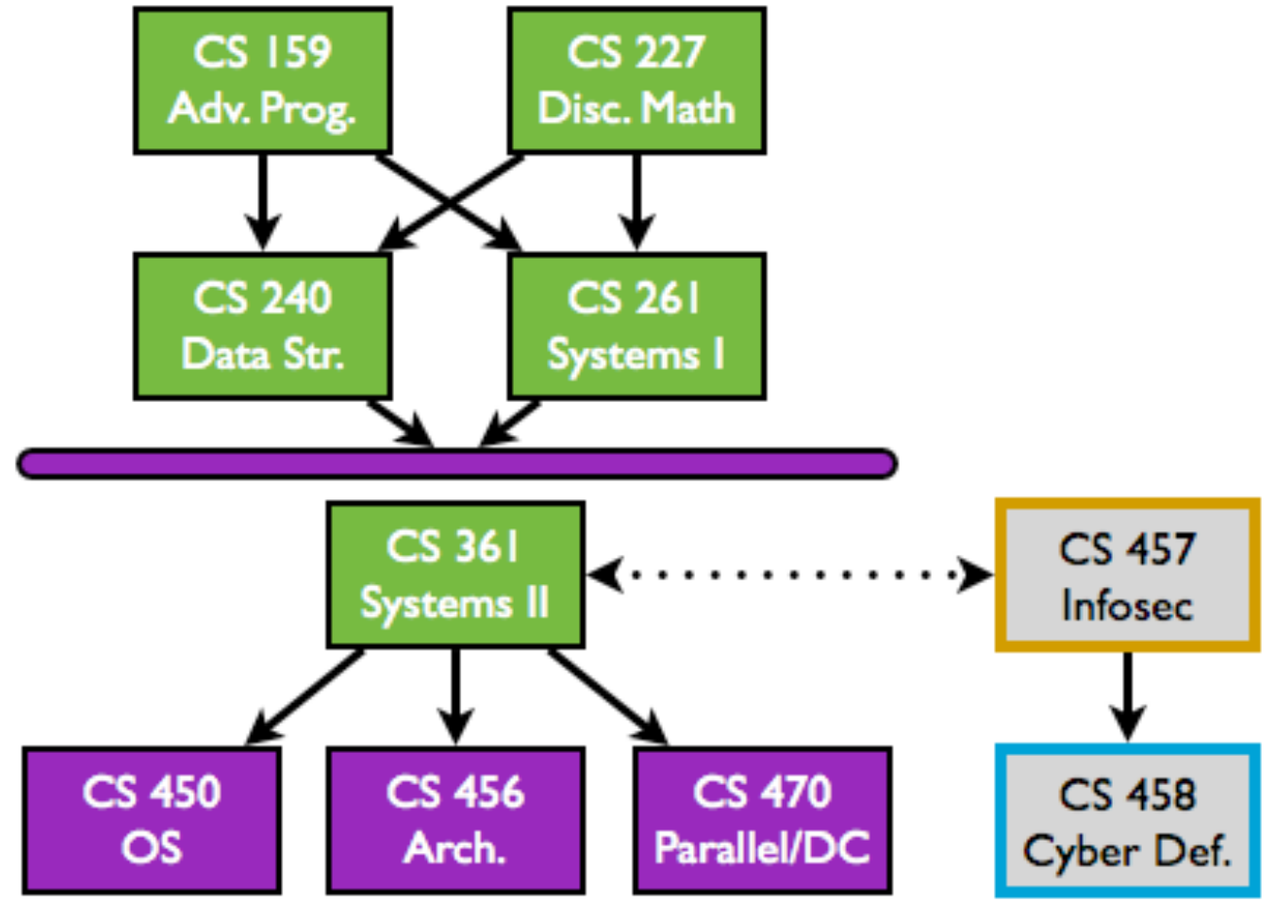
- Question bank for exams
  - Good for program assessment, too
- Common projects - 3-phase semester-long system building
  - Shared rubric encourages consistent expectations

Formative assessments and activities

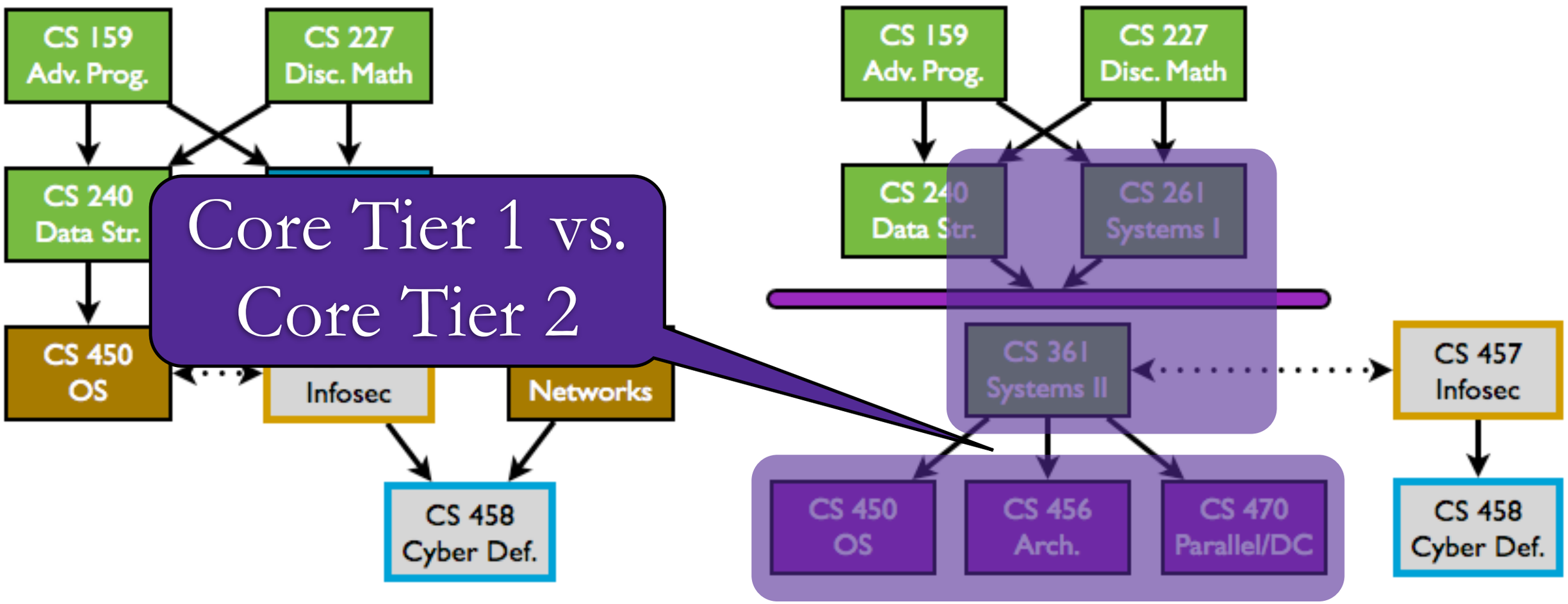
- Primarily at instructor discretion
- Challenge: flipped vs. non-flipped classroom structures

New courses


CS 159
Adv. Prog.
CS 227
Disc. Math
CS 240
Data Str.
CS 350
Comp. Org.
CS 450
OS
CS 457
Infosec
CS 460
Networks
CS 458
Cyber Def.


CS 159
Adv. Prog.
CS 227
Disc. Math
CS 240
Data Str.
CS 261
Systems I
CS 361
Systems II
CS 457
Infosec
CS 450
OS
CS 456
Arch.
CS 470
Parallel/DC
CS 458
Cyber Def.

New courses


CS 159
Adv. Prog.
CS 227
Disc. Math
CS 240
Data Str.
CS 450
OS
Infosec
Networks
CS 458
Cyber Def.
Core Tier 1 vs.
Core Tier 2
CS 159
Adv. Prog.
CS 227
Disc. Math
CS 240
Data Str.
CS 261
Systems I
CS 361
Systems II
CS 457
Infosec
CS 450
OS
CS 456
Arch.
CS 470
Parallel/DC
CS 458
Cyber Def.

New courses

CS 261 Computer Systems I (required)
Introduction to operation of modern interrupt-driven computer systems. Explores the representation of software and information in binary memory, the primary components of a CPU, multithreaded programming, and basic interactions with an Operating System. Prerequisites: Grade of "C-" or better in CS 159.

Module	Hours	Description
C basics	6	Memory model, pointers
Compiler and debugger use	3	GCC/clang, Makefiles, GDB
CPU/memory organization	3	Registers and cache, locality
Binary representation	4.5	Two's complement, IEEE 754, arithmetic encoding
von Neumann cycle	3	Role of CPU and memory, load/store instructions
Basic circuits	6	Logic gates, adders, ALUs, control signals
Threads vs. processes	6	Fork vs. thread creation, unique memory space
Interrupts and OS principles	4.5	Interrupts, system calls, user vs. kernel mode
System software design, evaluation	4.5	Benchmarks, complexity, static/dynamic analysis

New courses

CS 361 Computer Systems II (required)
Intermediate exploration of modern interrupt-driven computer systems. Explores models of computation and complex systems, techniques for communication and synchronization of parallel and concurrent software, and the protocols that make up the Internet. Prerequisites: Grades of "C-" or better in CS 240 and CS 261.

Module	Hours	Description
Architecture analysis, evaluation, design	3	P2P vs. client-server, layered architecture
State models	4.5	Notion of state, UML, FSM
Mathematical modeling	3	Basic systems theory
Information exchange	6	Communication basics (blocking vs. non-blocking, IPC vs. sockets)
Synchronization primitives and problems	6	Locks vs. semaphores, producer-consumer, readers-writers, dining philosophers, deadlock
Parallel decomposition	3	Data vs. task parallelism, Amdahl's law, fork-join pattern, libraries
Protocol analysis, evaluation, design	9	Protocols and services, timing and statechart diagrams, connections, push/pull, flow control, reliability, handshaking, metrics
The Internet model	4.5	HTTP, DNS, DHCP, TCP, UDP, IP, 802.3, 802.11, ARP

New courses

CS 450 Operating Systems
Introduction to the design and implementation of modern operating systems. Explores fundamental concepts of operating systems, memory management, virtualization, resource allocation, file systems, and system protection mechanisms. Course work includes a significant programming component. Prerequisites: Grade of "C-" or better in CS 361.

Module	Hours	Description
Thread and process management	4.5	Review of multithreading, OS structures for representing threads and processes, context switches
OS interface and IPC	4.5	Review of system calls vs. interrupts, pipes, shared memory, other forms of IPC
Synchronization implementation	6	Hardware support for implementation of semaphores, locks, spinlocks
Memory management	6	Paging vs. segmentation, virtual memory, demand paging, implementation of shared memory
Virtualization	6	Virtualization vs. emulation, trap-and-emulate, binary translation, hardware support for virtualization
Scheduling	3	Scheduling policies and evaluation
I/O and file systems	6	Interrupt-driven I/O, DMA, RAID, file system implementation and metadata
Security and protection	6	CIA model, access control mechanisms, malware defense mechanisms

New courses

CS 456 Computer Architecture
Introduction to the design and implementation of modern CPU architectures. Explores hardware-based parallel execution, quantitative performance evaluation, I/O interfacing techniques, and hardware descriptor languages. Course work includes a significant programming component. Prerequisites: Grade of "C-" or better in CS 261.

Module	Hours	Description
Assembly language	6	RISC assembly language and decoding
Building a datapath	6	Logic gates, control unit, ALU construction, register banks, von Neumann implementation
Hardware descriptor languages	3	Verilog, VHDL, RTL
Pipelined datapath and hazards	6	Pipelined datapath and control, data hazards (forwarding vs. stalling), control hazards, exceptions
Memory hierarchy and cache design	6	Quantitative performance measures, cache mapping techniques, cache coherence protocols
Storage and I/O interfacing	4.5	Storage devices, bus protocols, I/O performance
Instruction-level parallelism	4.5	Branch prediction, dynamic scheduling
Data-level parallel architectures	3	Vector, SIMD, GPU architectures
Thread-level parallel techniques	3	Hyperthreading, shared-memory multiprocessors

New courses

CS 470 Parallel and Distributed Systems
Introduction to parallel and distributed systems. Explores shared memory, cluster, grid, peer-to-peer, and cloud computing models along with parallel software patterns, distributed file systems, and performance considerations. Course work includes a significant programming component. Prerequisites: Grade of "C-" or better in CS 361.

Module	Hours	Description
Parallel/distributed concepts	3	Amdahl's law, critical paths, speedup/scalability, data/task decomposition, applications, research challenges
Parallel patterns	6	Naturally (embarrasingly) parallel, nearest-neighbor, communication, producer-consumer, master-workers, pipelines, map/reduce
Parallel systems	9	Shared memory, SMP, SIMD, OpenMP, GPUs/co-processors, race conditions, mutual exclusion, deadlock, cache effects, dense/sparse matrices
Distributed systems	9	MPI, MapReduce, global address spaces, clusters, topologies, synchronization, collectives, clocks, NUMA, hybrid architectures, fault tolerance
Grid, P2P, and cloud computing systems	6	Heterogeneous systems, decentralized computation, consensus, IaaS, virtualization
Distributed file systems	6	RPC, data replication, transactions, consistency
Parallel performance	3	Tools, measurement, scheduling/load balancing, contention, communication overhead, power usage

Possible future courses

Breadth-first scaffolding as foundation for:

- Advanced networking
- Database design and implementation
- Virtualization technologies

ACM 2013 Mapping

ACM 2013 KA	Current	Proposed
Algorithms and Complexity		
Distributed algorithms (T1)	none	470
Architecture and Organization		
Machine-level representation of data (T2)	350	261, 456
Assembly level machine organization (T2)	350	261, 456
Interfacing and I/O strategies (interrupts) (T2)	350	456
Memory architecture (T2)	350	261, 456
Functional organization (ILP/datapaths) (E)	350 (some)	261, 456
Operating Systems		
OS overview (T1)	450	261
OS principles (APIs, processes, interrupts) (T1)	450	261, 450
Concurrency (T2)	450	361, 450
Scheduling (T2)	450	450
Memory management (T2)	450	450
Security and protection (T2)	450	457/450
File systems (E)	450 (some)	450
System performance evaluation (E)	none	261
Network-centric Computing		
Introduction (ISPs, circuit vs. packet) (T1)	460	361
Network applications (HTTP, sockets) (T1)	460 (most)	361, 466, 470
Reliable data delivery (TCP, flow control) (T2)	460	361, 466
Routing vs. forwarding (T2)	460	361, 466
LANs (T2)	460	361, 466
Resource allocation (T2)	460	361, 466
Mobility (T2)	460	361, 466

ACM 2013 KA	Current	Proposed
Parallel and Distributed Computing		
Parallelism fundamentals (T1)	450 (most)	261, 470
Parallel decomposition (T1,T2)	none	361, 470
Communication and coordination (T1,T2)	450 (some)	361, 450, 470
Parallel architectures (T1,T2)	350 (little)	261, 470
Parallel performance (E)	none	470
Distributed systems (E)	none	361, 470
System Fundamentals		
Computational paradigms (T1)	350 (some)	261, 361, 456, 470
Cross-layer communications (T1)	450 (little)	361, 450, 466, 470
States, transitions, state machines (T1)	350 (some)	361, 456, 466, 470
System support for parallelism (T1)	350 (little)	361, 456, 470
Performance (T1)	none	261
Resource allocation and scheduling (T2)	450	450
Proximity (T2)	350	450, 456, 470
Virtualization (T2)	none	450
Reliability through redundancy (T2)	460 (little)	361, 456, 470

Transitional plan

Transitional plan

Fall 2015

- CS 460 replaced by CS 361
- Elaborate tango between CS 361 and CS 450 (OS)

Transitional plan

Fall 2015

- CS 460 replaced by CS 361
- Elaborate tango between CS 361 and CS 450 (OS)

Spring 2016

- CS 350 replaced by CS 261
- First offering of CS 470 (Parallel and Distributed)

Transitional plan

Fall 2015

- CS 460 replaced by CS 361
- Elaborate tango between CS 361 and CS 450 (OS)

Spring 2016

- CS 350 replaced by CS 261
- First offering of CS 470 (Parallel and Distributed)

Fall 2016

- New curriculum offering fully in place
  - CS 261 and CS 361 now offered every semester
- First offering of CS 456 (Architecture)

Lessons learned

Too many to name…

- A well-defined vision statement works wonders
  - Revisit frequently, particularly for conflict resolution
- SMART objectives make writing assessments easier
  - Take the time to do this well
- Transition to new curriculum requires improvisation and flexibility
- Build your own taxonomy
  - Fink and Wiggins/McTighe for aspirational structure
  - Hansen for enumerating points of emphasis
  - Bloom and SOLO for instructional objectives
- The process will take longer than you expect
  - Hidden assumptions, connections, requirements

https://github.com/kirkpams/JMU-CS-Systems