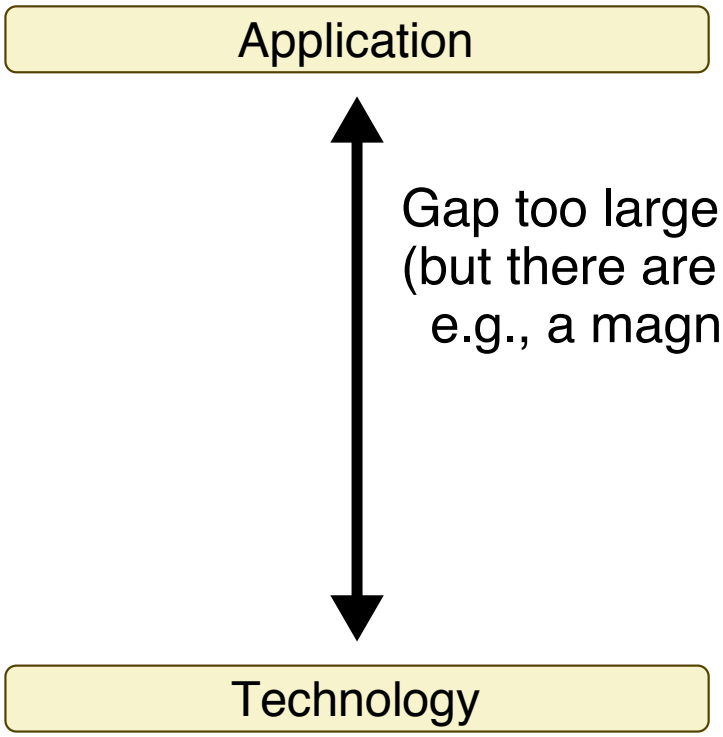
ECE 3400 Guest Lecture
Design Principles and Methodologies in Computer Architecture

Christopher Batten

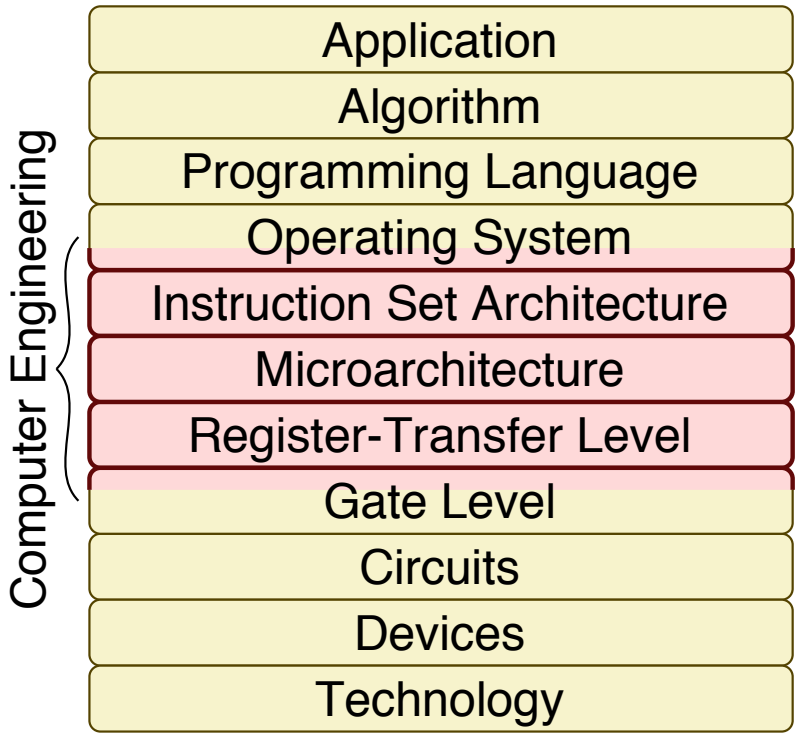
Computer Systems Laboratory
School of Electrical and Computer Engineering
Cornell University

The Computer Engineering Stack



Gap too large to bridge in one step
(but there are exceptions,
e.g., a magnetic compass)

The Computer Engineering Stack



Sort an array of numbers

2,6,3,8,4,5 -> 2,3,4,5,6,8

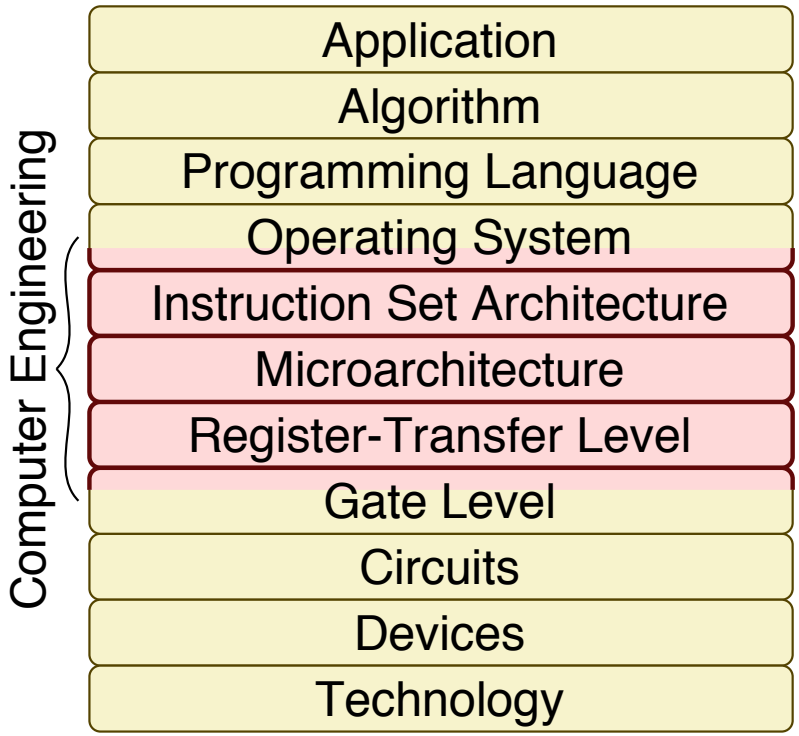
Insertion sort algorithm

1. Find minimum number in input array
2. Move minimum number into output array
3. Repeat steps 1 and 2 until finished

C implementation of insertion sort

```
void isort( int b[], int a[], int n ) {
  for ( int idx, k = 0; k < n; k++ ) {
    int min = 100;
    for ( int i = 0; i < n; i++ ) {
      if ( a[i] < min ) {
        min = a[i];
        idx = i;
      }
    }
    b[k]   = min;
    a[idx] = 100;
  }
}
```

The Computer Engineering Stack

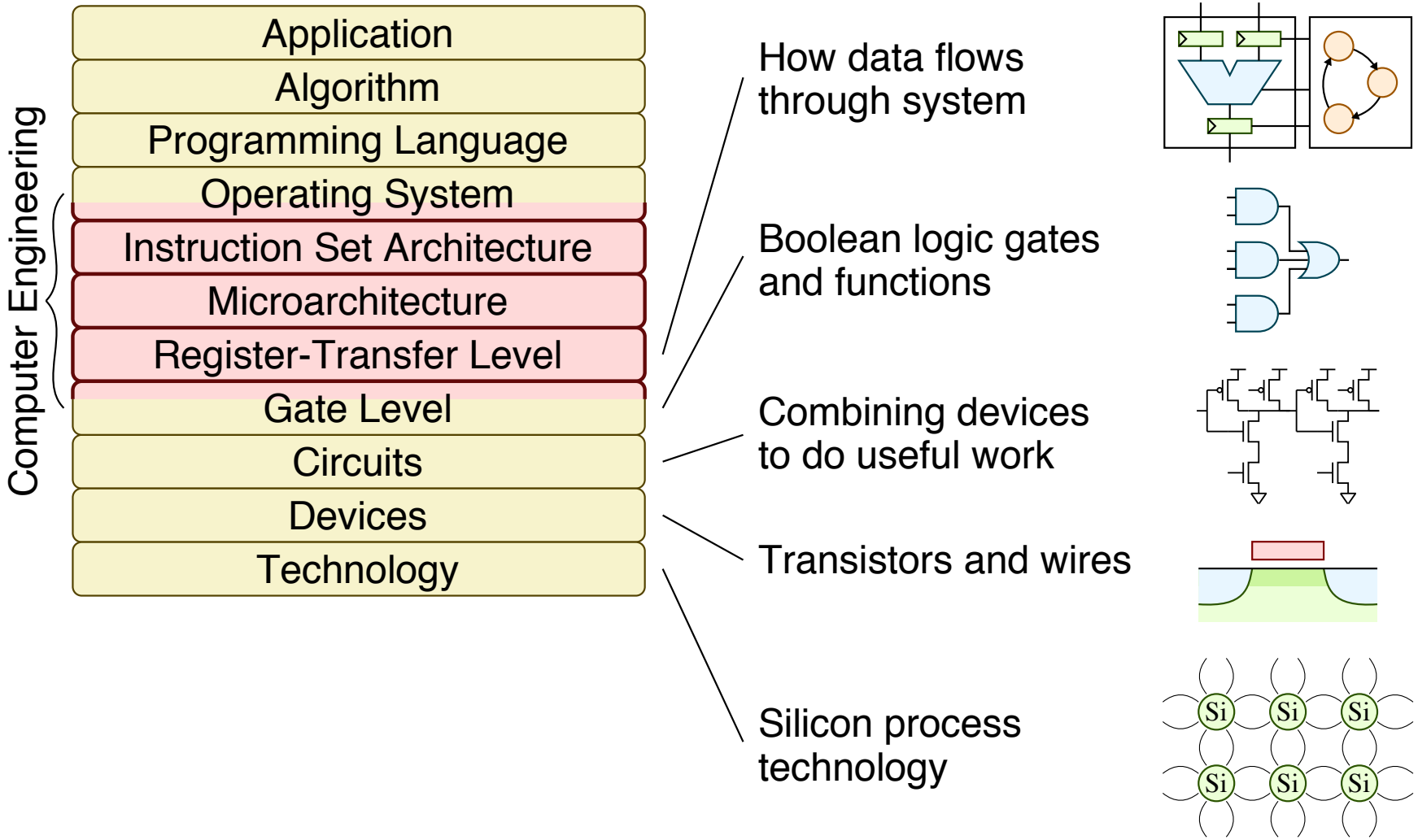


Mac OS X, Windows, Linux
Handles low-level hardware management

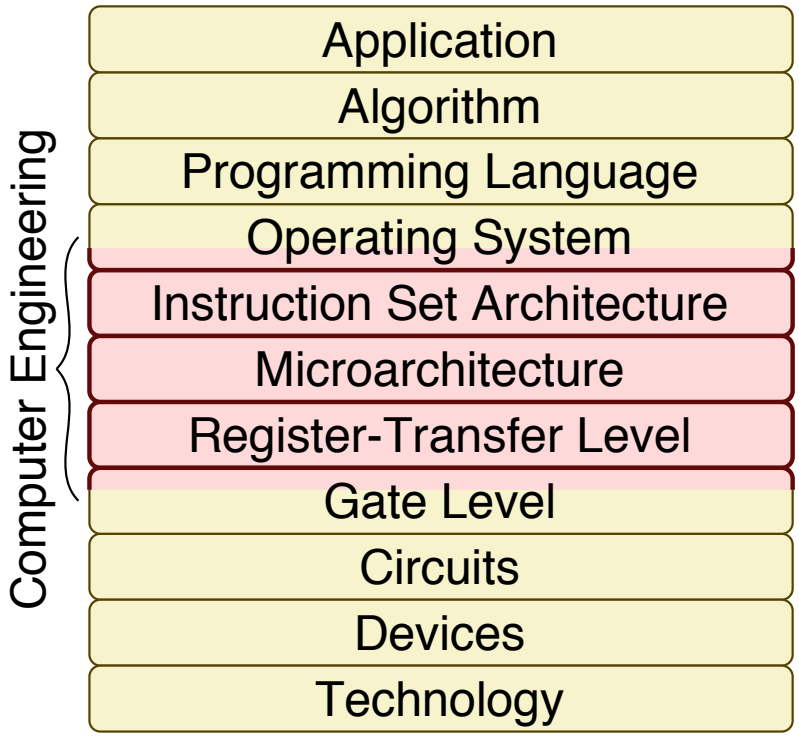
MIPS32 Instruction Set
Instructions that machine executes

```
blez  $a2, done
move  $a7, $zero
li    $t4, 99
move  $a4, $a1
move  $v1, $zero
li    $a3, 99
lw    $a5, 0($a4)
addiu $a4, $a4, 4
slt   $a6, $a5, $a3
movn  $v0, $v1, $a6
addiu $v1, $v1, 1
movn  $a3, $a5, $a6
```

The Computer Engineering Stack



What is Computer Architecture?



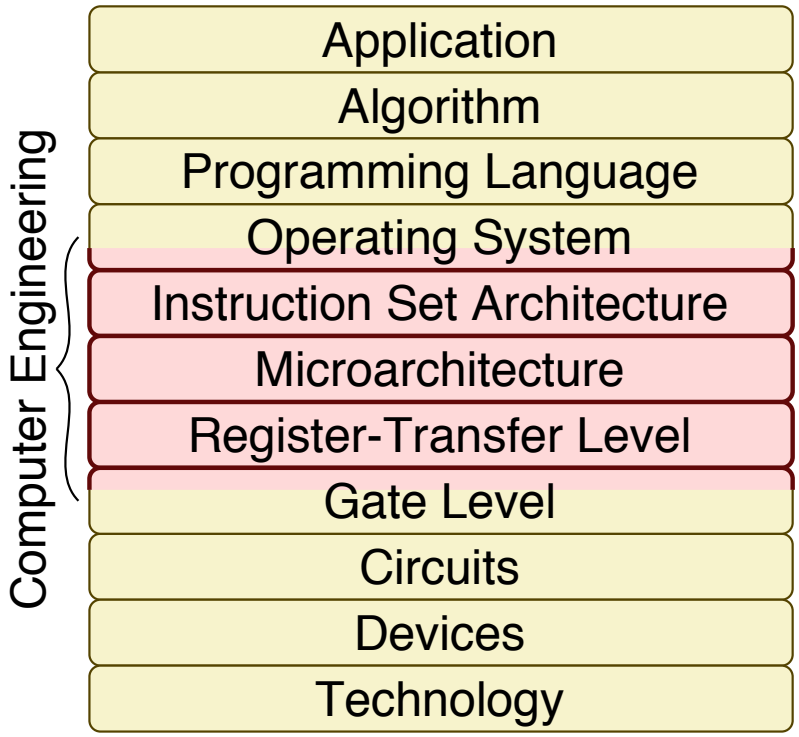
Application Requirements
- Suggest how to improve architecture
- Provide revenue to fund development

Computer engineers provide feedback to guide application and technology research directions

Technology Constraints
- Restrict what can be done efficiently
- New technologies make new arch possible

In its broadest definition, computer engineering is the development of the abstraction/implementation layers that allow us to execute information processing applications efficiently using available manufacturing technologies

Computer Architecture in the ECE/CS Curriculum



CS 4410 Operating Systems
CS 4420 Compilers
ECE 3140 Embedded Systems

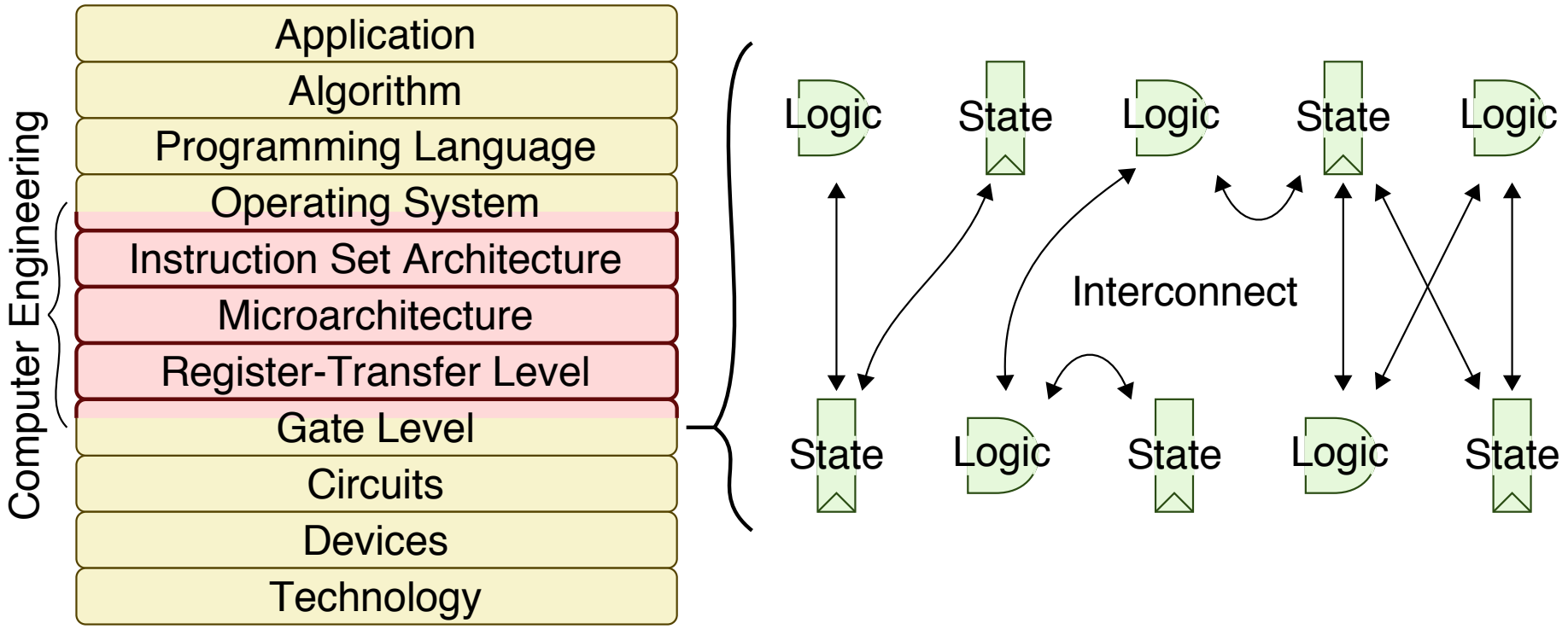
ECE 4750 Computer Architecture

ECE 2300 Digital Logic & Computer Org
ECE 4740 Digital VLSI Design

Related Graduate Courses

- ECE 5760 Advanced Microcontroller Design
- ECE 5750 Advanced Computer Architecture
- ECE 5730 Memory Systems
- ECE 5770 Resilient Computer Systems
- ECE 5745 Complex Digital ASIC Design
- ECE 5775 High-Level Design Automation

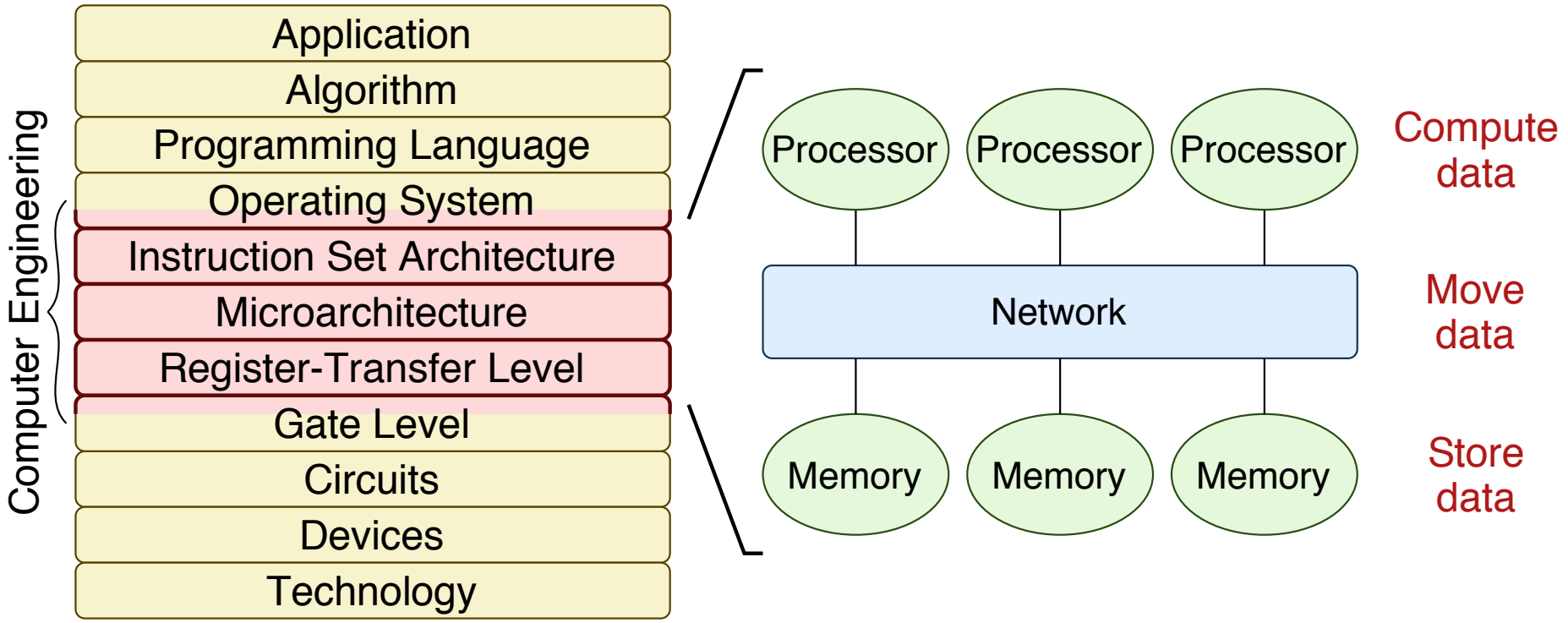
Logic, State, and Interconnect



Digital systems are implemented with three basic building blocks

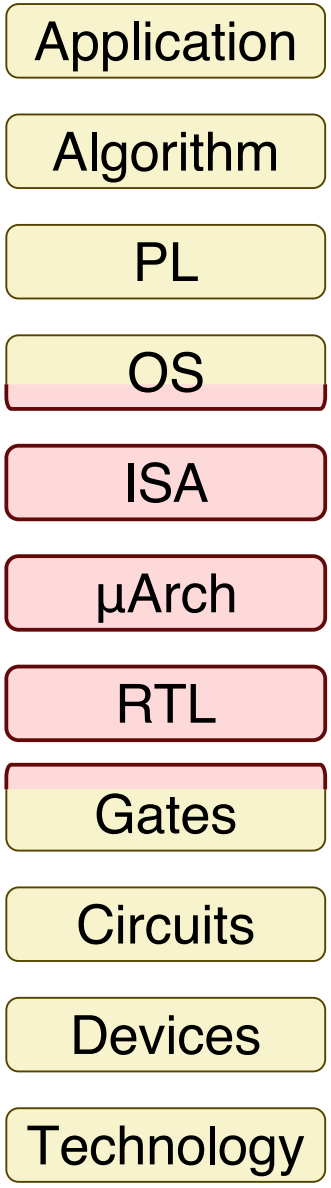
- Logic to process data
- State to store data
- Interconnect to move data

Processors, Memories, and Networks



In computer engineering we more generally think of using

- Processors for computation
- Memories for storage
- Networks for communication



Agenda

What is Computer Architecture?

Design Example

Design Principles

Design Methodologies

What do computer architects actually do?

General Science

Discover truths about nature

Ask question about nature

↓

Construct hypothesis

↓

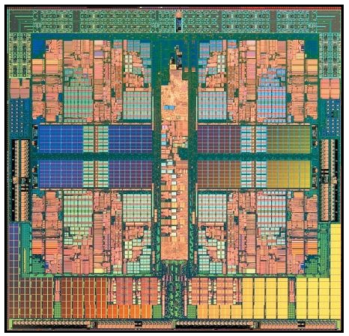
Test with experiment

↓

Analyze results and draw conclusions

Computer Engineering

Explore design space for a new system



Design and model baseline system

↓

Ask question about system

↓

Test with experiment

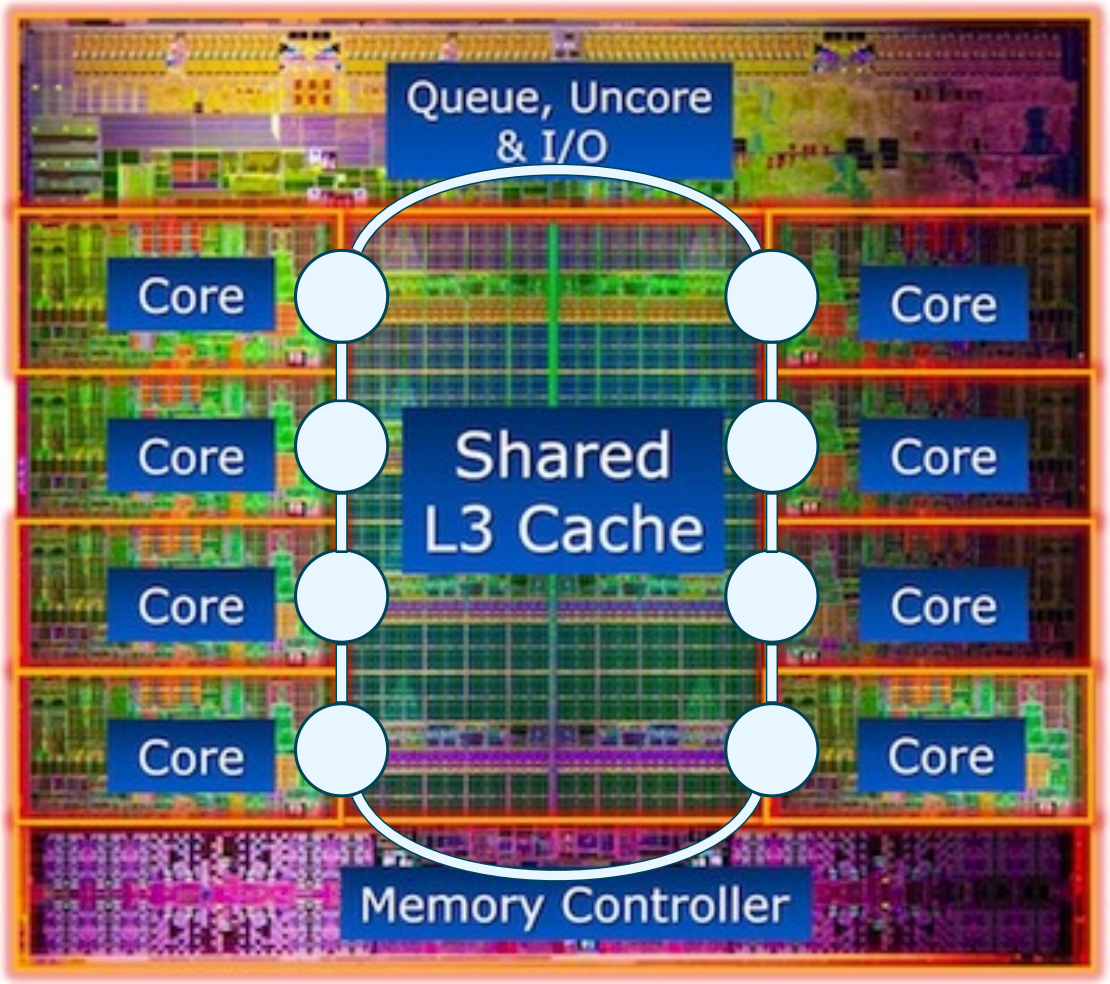
↓

Analyze results and draw conclusions

↙ ↘

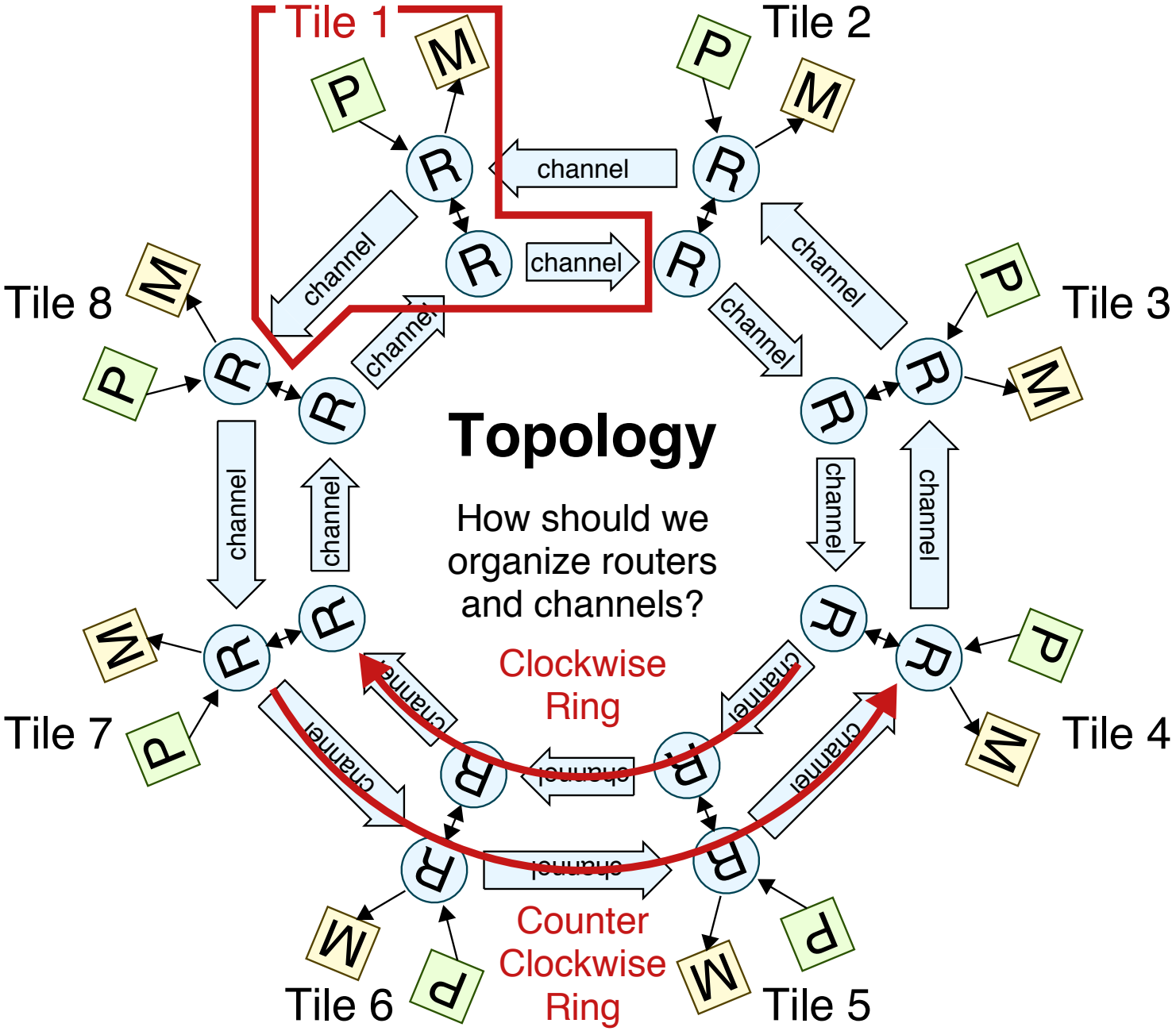
Build prototype or real system

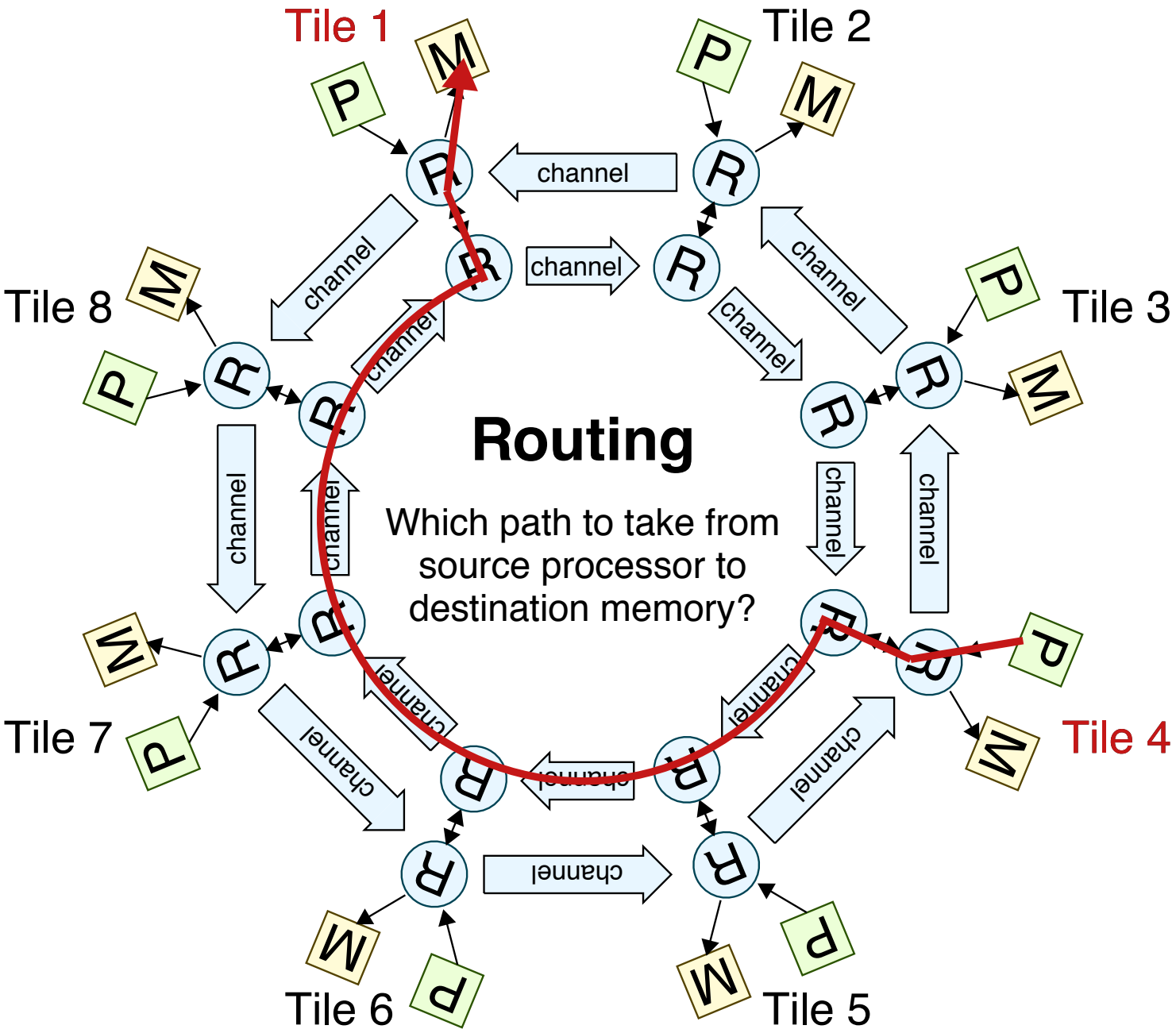
Design and model alternative system

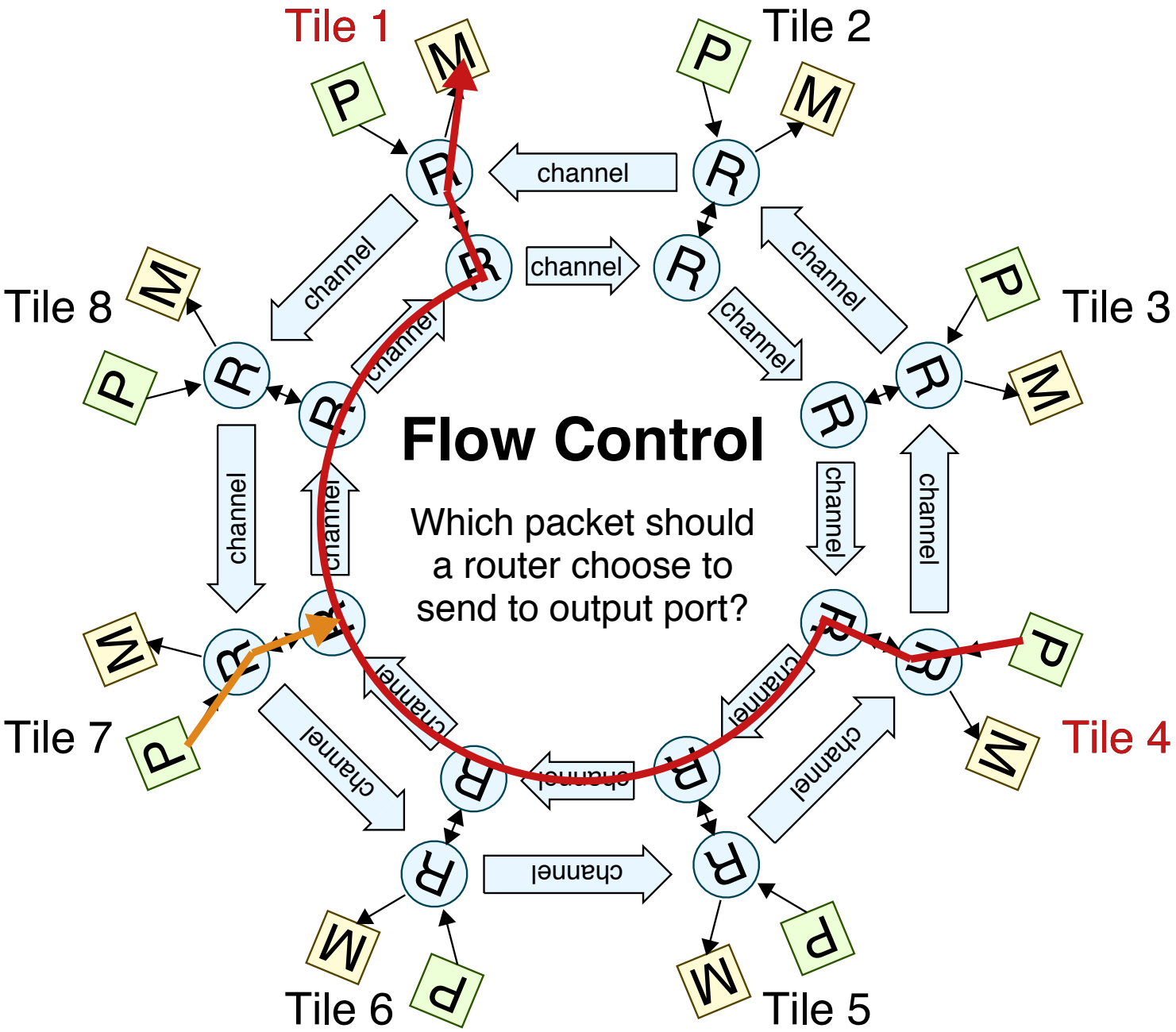


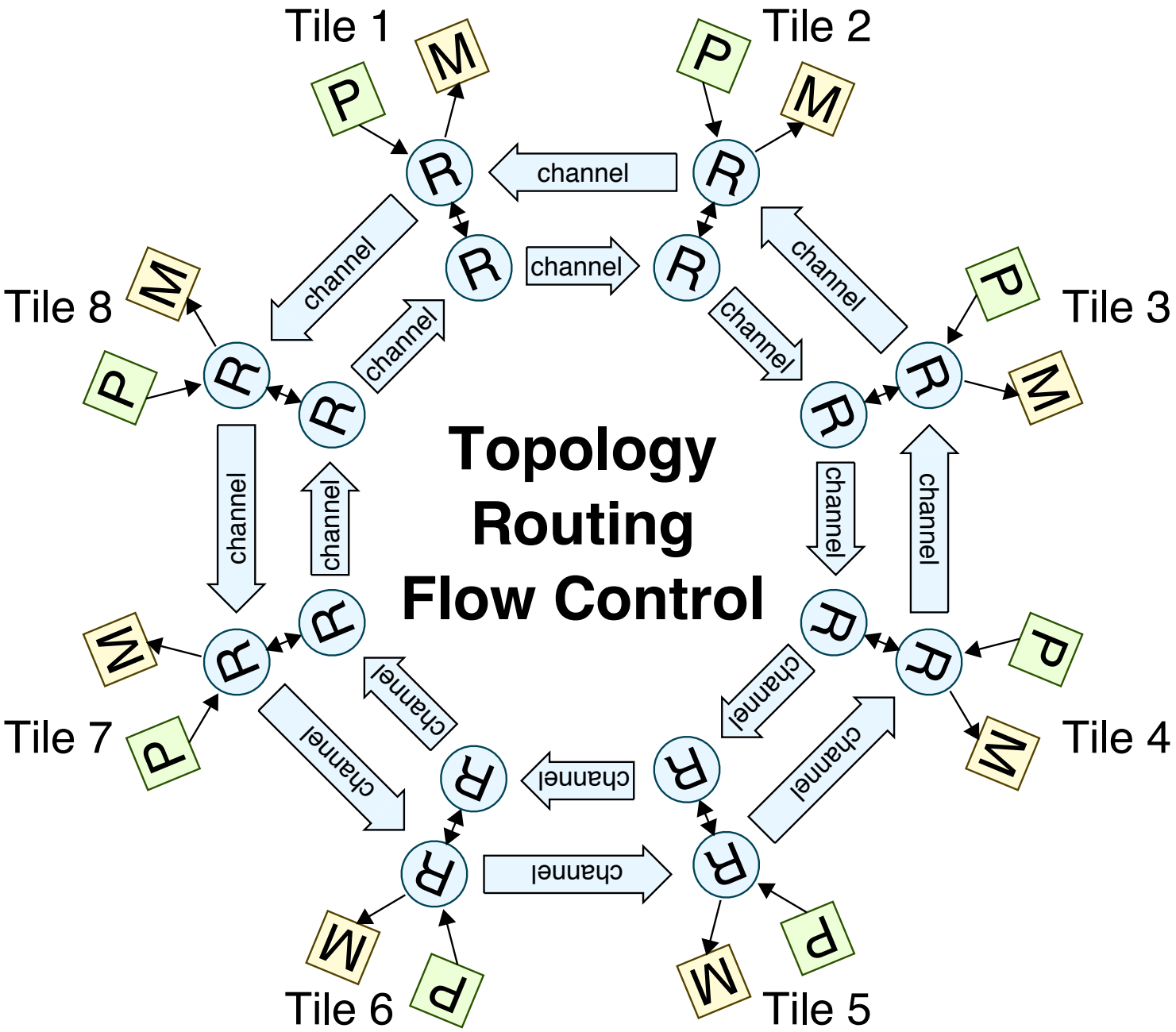
Example Design Problem: On-Chip Interconnection Network

- Intel Sandybridge E Server-Class Processor
- 435 mm^2 in 32 nm technology with 2.27B transistors running at several GHz
- Eight cores and eight memory banks with an on-chip ring network


Tile 1
Tile 2
Tile 3
Tile 4
Tile 5
Tile 6
Tile 7
Tile 8
P
M
R
channel
Topology
How should we organize routers and channels?
Clockwise Ring
Counter Clockwise Ring


Tile 1
Tile 2
Tile 3
Tile 4
Tile 5
Tile 6
Tile 7
Tile 8
P
M
R
channel
Routing
Which path to take from source processor to destination memory?

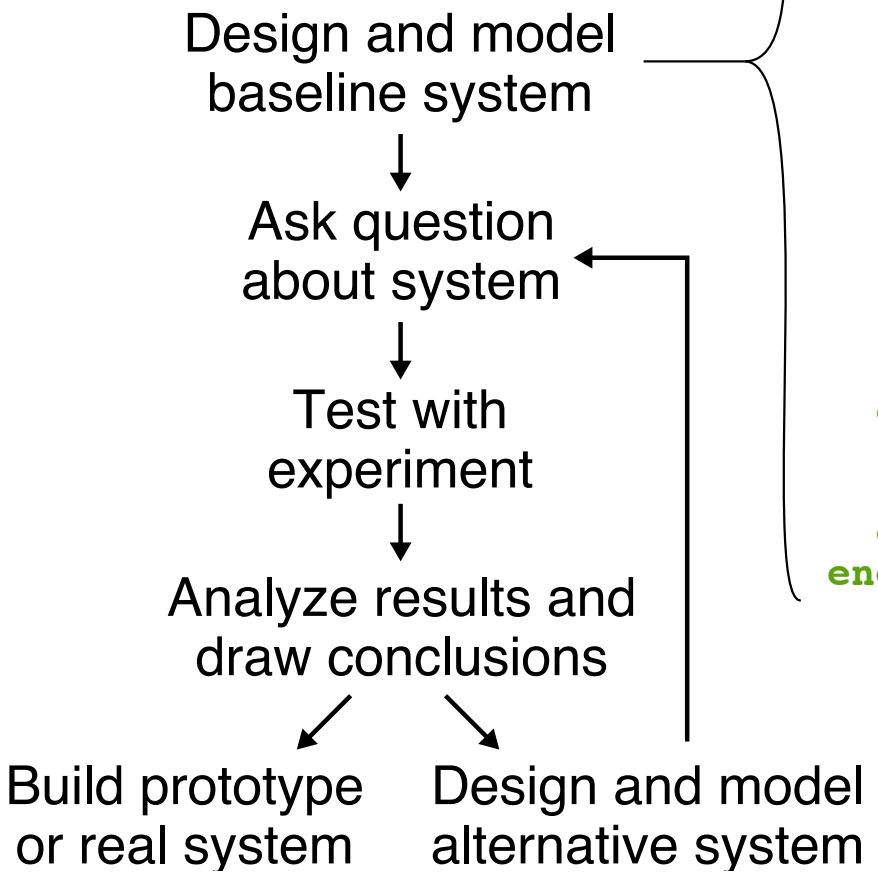

Tile 1
Tile 2
Tile 3
Tile 4
Tile 5
Tile 6
Tile 7
Tile 8
P
M
R
channel
Flow Control
Which packet should a router choose to send to output port?


Tile 1
Tile 2
Tile 3
Tile 4
Tile 5
Tile 6
Tile 7
Tile 8
Topology
Routing
Flow Control
channel
R
P
M

Modeling in Computer Architecture

Computer Engineering

Explore design space for a new system




```
// rdy is OR of the AND of reqs and grants
assign in_rdy = | (reqs & grants);

reg [2:0] reqs;
always @(*) begin
  if ( in_val ) begin

    // eject packet if it is for this tile
    if ( dest == p_router_id )
      reqs = 3'b010;

    // otherwise, just pass it along ring
    else
      reqs = 3'b001;

  end else begin
    // if !val, don't request any ports
    reqs = 3'b000;
  end
end
```

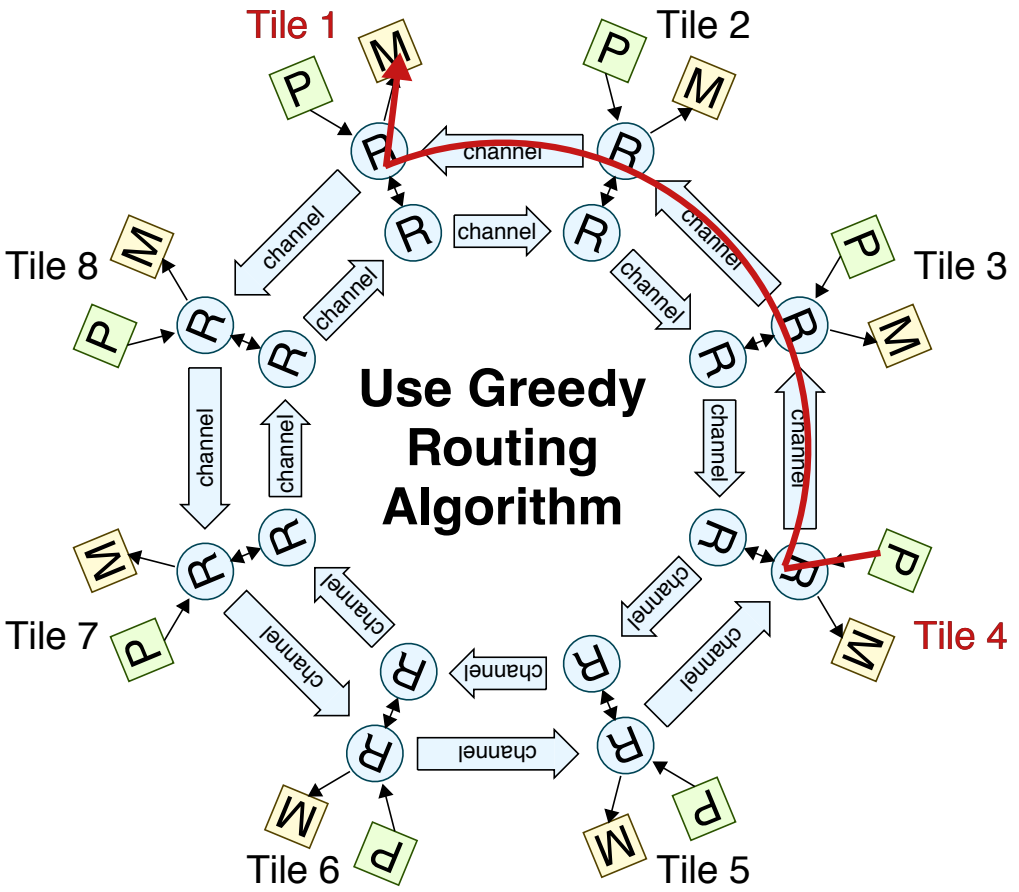
Verilog • SystemVerilog • VHDL

C++ • SystemC

Bluespec • Chisel • Python

Student-Based "Model" of On-Chip Network

- Processor/Memory – Student hands packets to appropriate router based on routing algorithm; waits to receive two packets from some other tile, and then raises hand.
- CCW/CW Network Routers – Student holds onto packets and hands them to the correct channel; if packet for that router's tile arrives, then hand packet to processor/memory.
- CCW/CW Network Channels – Student walks *one packet at a time* from upstream router to downstream router.



Results from Verilog Model of On-Chip Network

Greedy Routing Algorithm

```
       Counter   From
cycle  Clockwise Processor Clockwise
   1:  ........  ........  ........
   2:  ........  ..SSSS..  ........
   3:  ........  .S....SS  ........
   4:  ...SSSS.  S.S..#.S  ........
   5:  S.S....S  ...SSS.#  ........
   6:  SS.SSS##  .#S..#SS  ........
   7:  ##.SS#SS  SSS..S.#  ........
   8:  SSS##S##  ##.SS#SS  ........
   9:  ##SSSSSS  SS...S.#  ........
  10:  SSS#SSSS  ##SS.#.S  ........
  11:  SSSS##SS  ##..SS##  ........
  12:  SSSSSSS#  ##S.#.#S  ........
  13:  S#SSSSSS  #S.##S##  ........
  14:  SS.S#SS#  ##S#S.#S  ........
  15:  SSSSS#SS  #####S#.  ........
  16:  SS#SSSSS  ##S##.#S  ........
  17:  SSSSSSSS  #####.#.  ........
  18:  SSSSSSS#  ##S####S  ........
  19:  SSSSS##S  #SS#SSS#  ........
```

Experiment Execution Time
138 cycles

Adaptive Routing Algorithm

```
       Counter   From
cycle  Clockwise Processor Clockwise
   1:  ........  ........  ........
   2:  ........  ..SSSS..  ........
   3:  ........  .S....SS  ........
   4:  ...SSSS.  S.S..#.S  ........
   5:  S.S....S  ...SSS.#  ........
   6:  .S.SS##S  .#S..SSS  ......S.
   7:  #S.S.SS.  SSS....S  ..SSS...
   8:  S.S.#SSS  SS.SS#S.  .S...S#.
   9:  SSSSS.SS  .....S.S  SSSS..S.
  10:  #S#SSS.#  SSSS.S.S  S...S#.S
  11:  S.S.#S#S  .S..S.S.  S#S..SSS
  12:  #S#SS.S.  S.S.S.#.  .S.S#SSS
  13:  S#S#####  .S.SSSSS  SS..SSSS
  14:  #SSSSSSS  S.S...#.  S.SSSSS.
  15:  SS##S##S  .SSS.SS.  S..SSSSS
  16:  SSSSSSS#  S.###.SS  .SSSSS.S
  17:  SSSSSS#S  #.###.S.  S.SSS.S.
  18:  SSSS#SSS  #SS##S#S  S#SSS#.#
  19:  S#SSSSSS  #S###.S#  .SSS.S.S
```

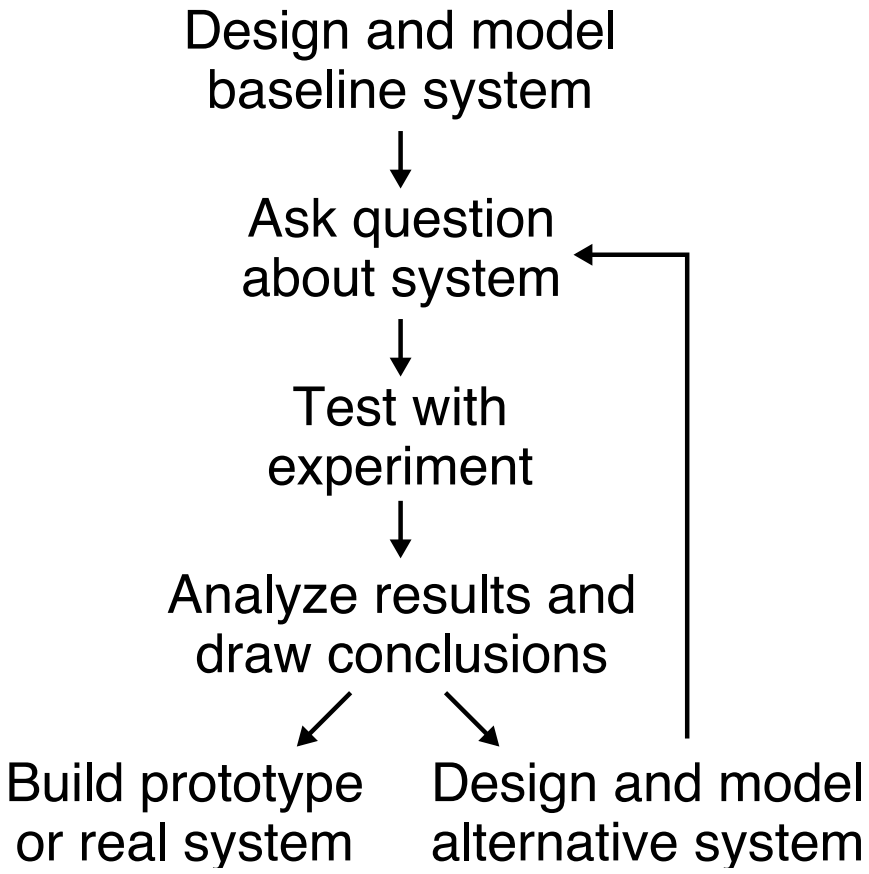
Experiment Execution Time
93 cycles (1.48× speedup)

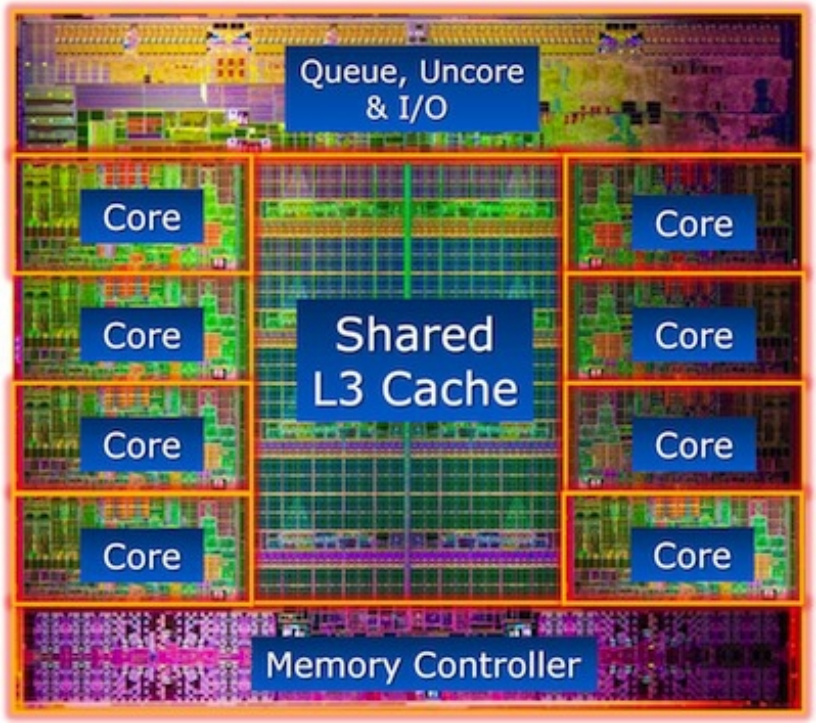
1 column/channel: `.` = idle, `S` = send packet, # = channel busy

But how *do* we design/model something so complex?

Computer Engineering

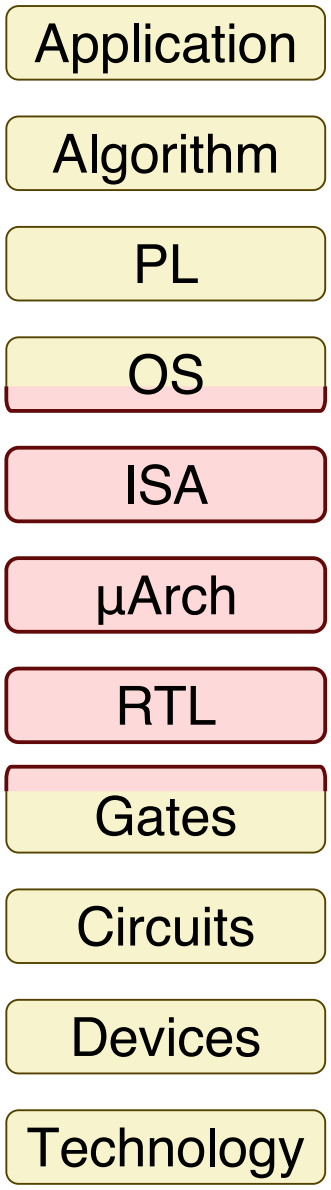
Explore design space for a new system





Fighter Airplane: ~100,000 parts

Intel Sandy Bridge E:
2.27 Billion transistors



Agenda

What is Computer Architecture?

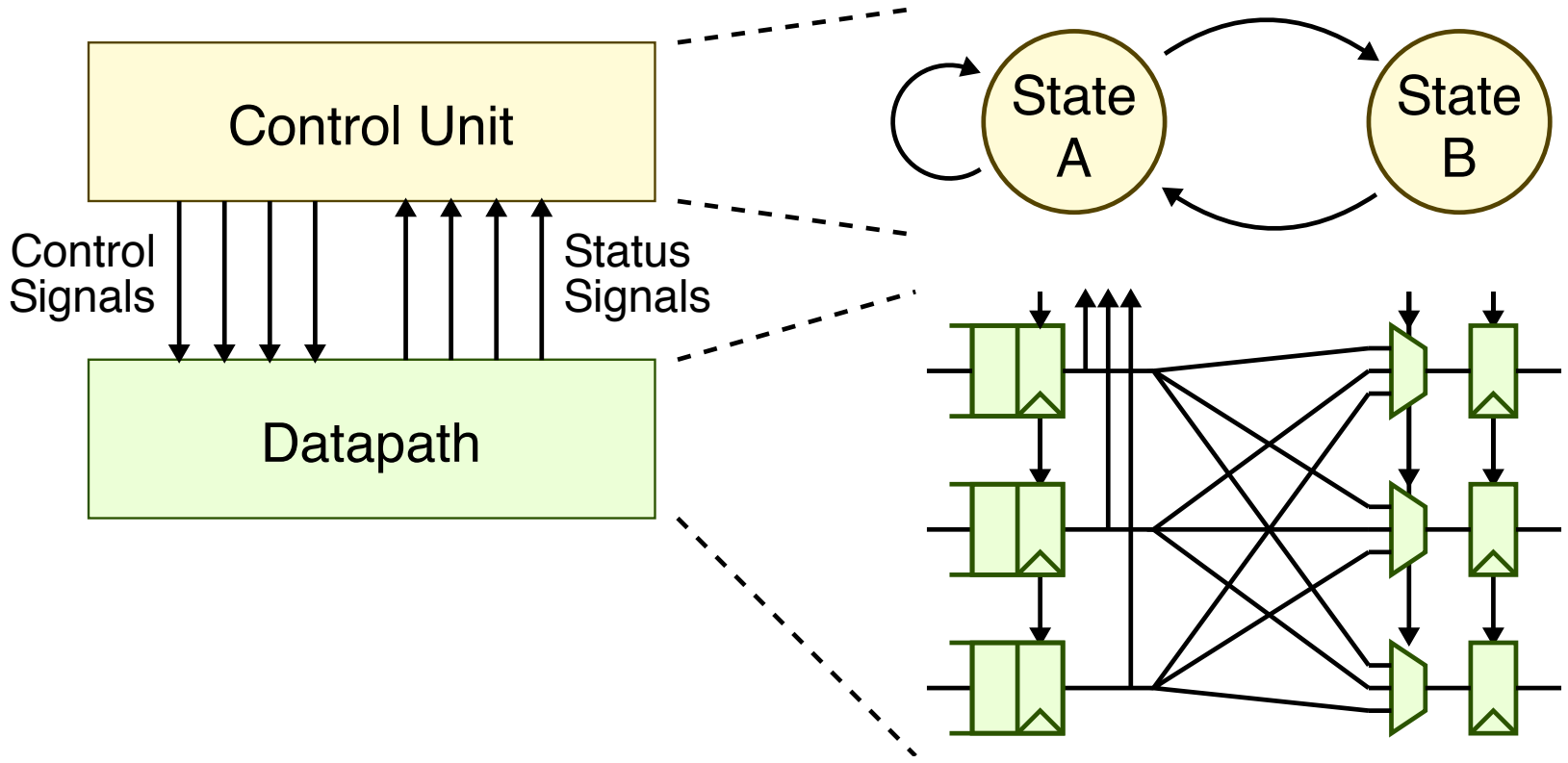
Design Example

Design Principles

Design Methodologies

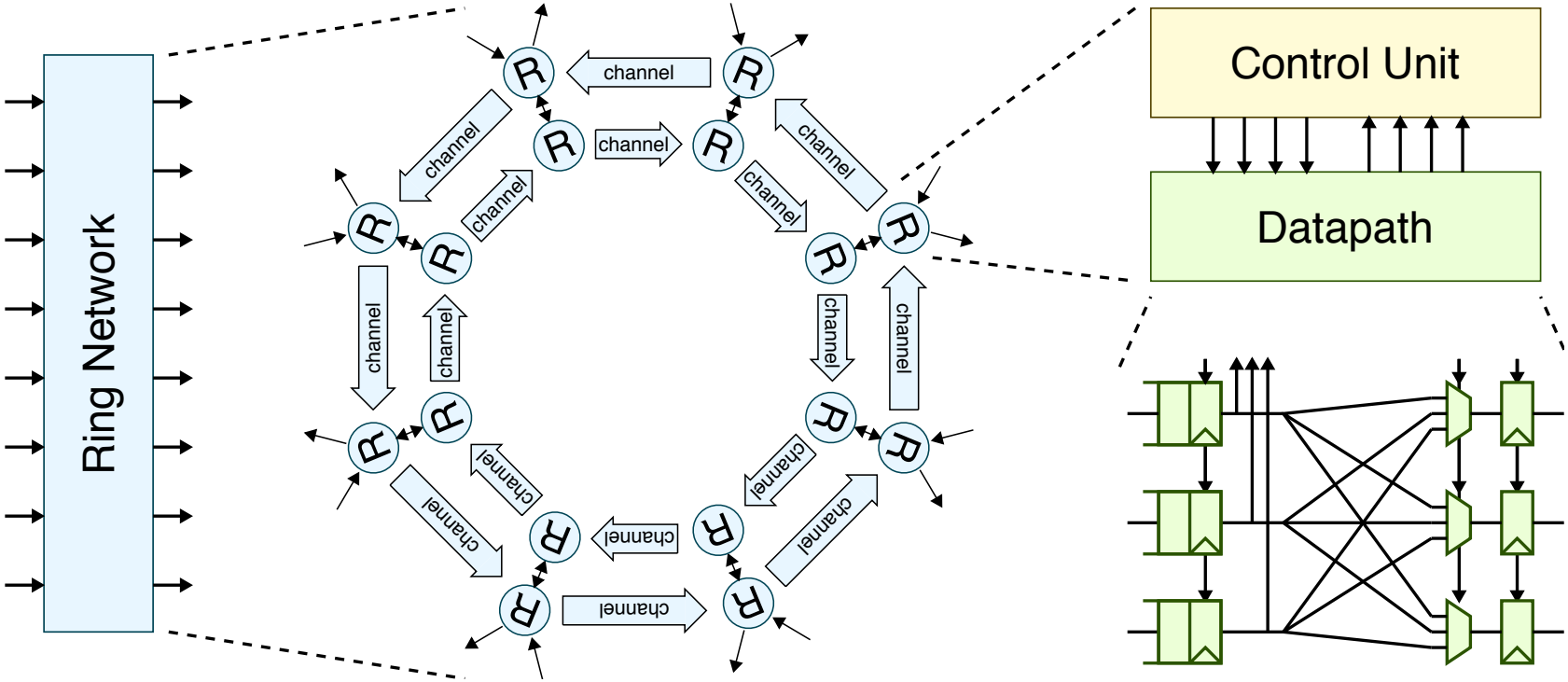
Design Principle: Modularity

- Decompose into components with well-defined interfaces
- A modular router design can be decomposed into a control unit and a datapath interconnected with control/status signals



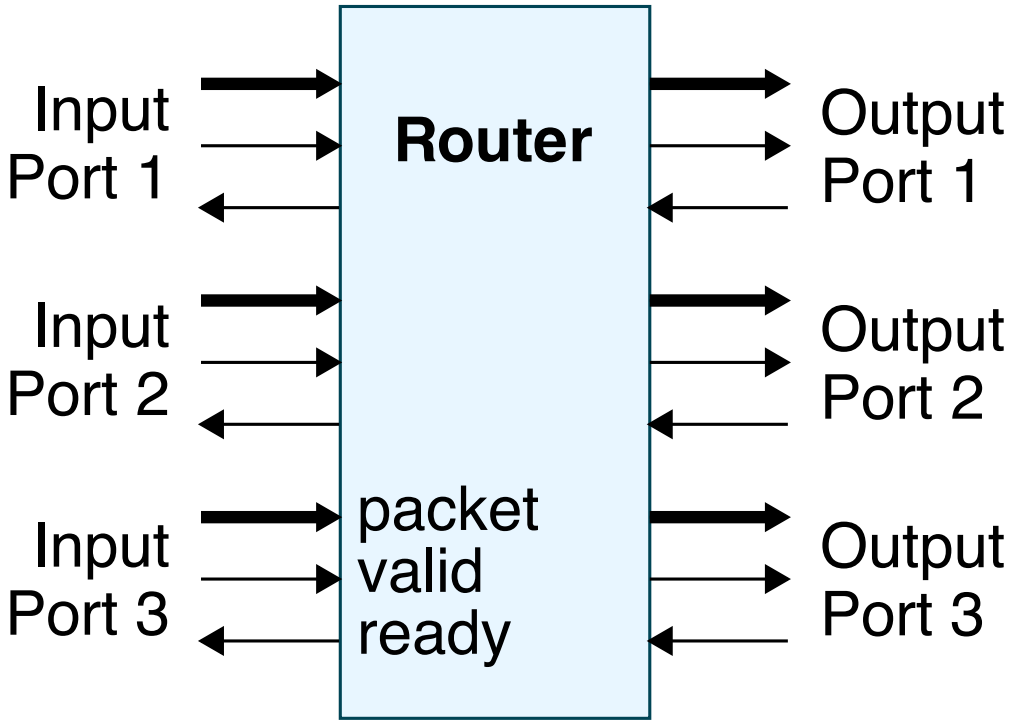
Design Principle: Hierarchy

- Recursively apply modularity principle
- A hierarchical network design can be decomposed into routers, which is in turn decomposed into a control unit and datapath, which is in turn decomposed into queues, muxes, and registers



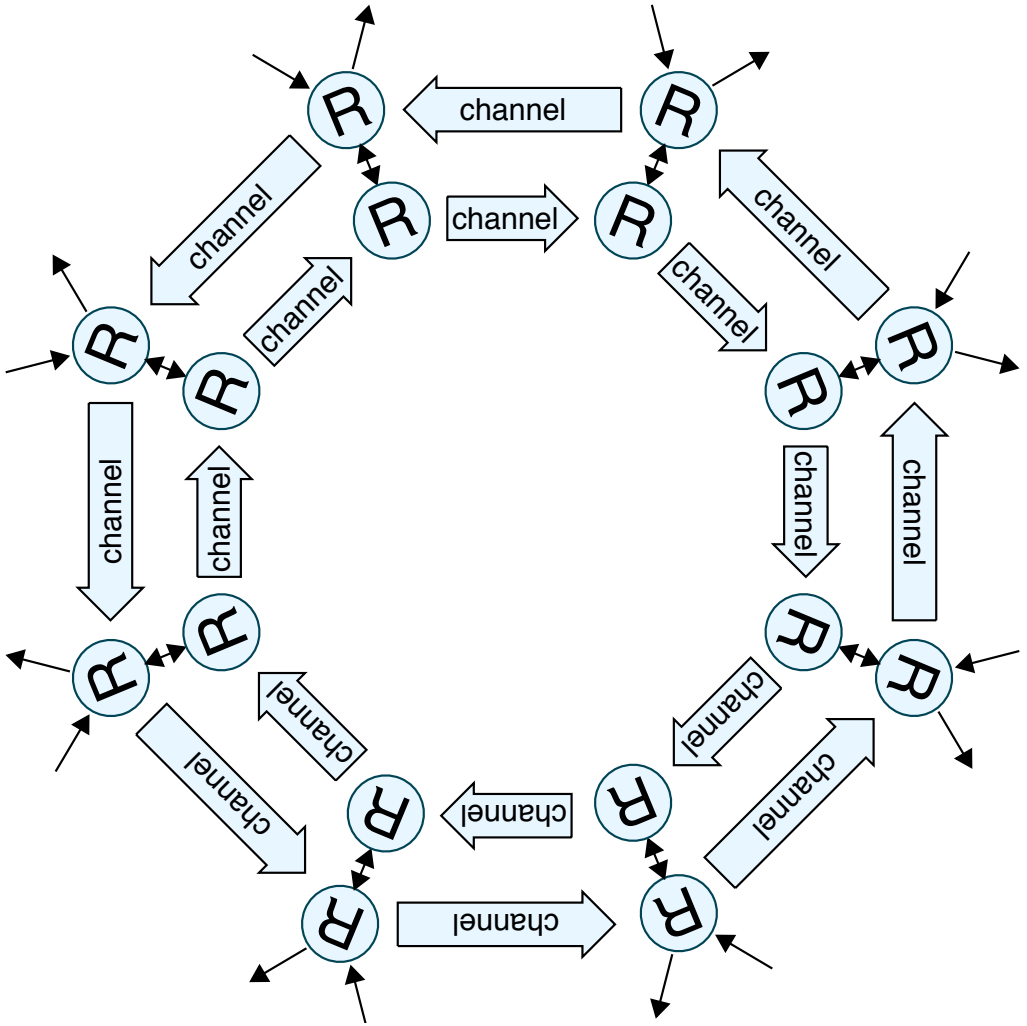
Design Principle: Encapsulation

- Hide implementation details from interfaces
- An encapsulated router design can hide the latency of the router microarchitecture along with any details related to stalls due to full queues or arbitration



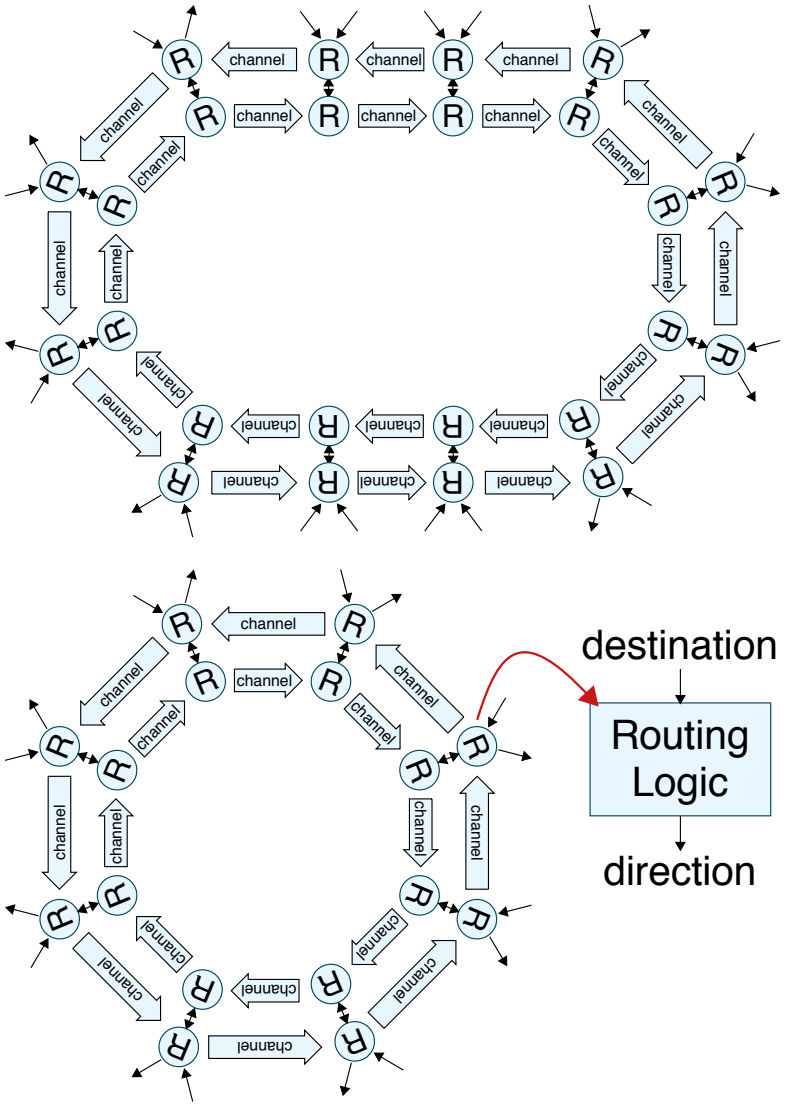
Design Principle: Regularity

- Leverage structure at various levels of abstraction
- A regular router design can exploit logical structure to enable a single router to be instantiated eight times
- A regular network and router design can exploit physical structure to simplify the chip floorplan and layout



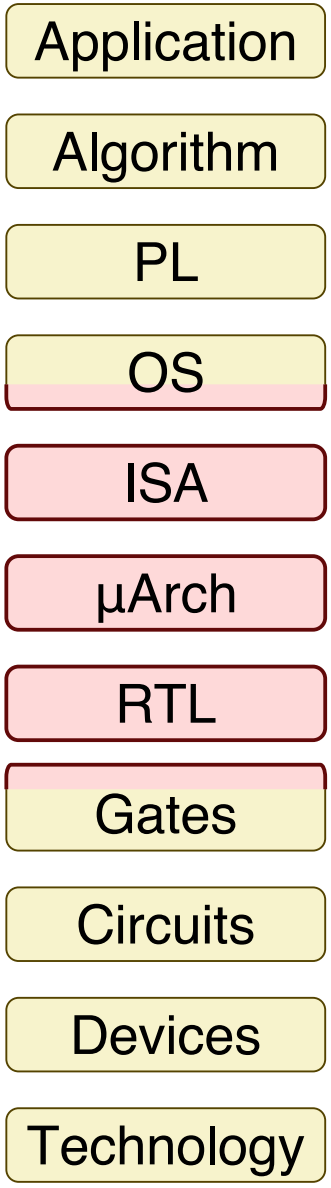
Design Principle: Extensibility

- Include mechanisms/hooks to simplify future changes
- An extensible network and router design can enable easily implementing ring networks with various numbers of routers
- An extensible network and router design can enable easily changing the routing algorithm



Design Principles in Computer Architecture

- Modularity – Decompose into components with well-defined interfaces
- Hierarchy – Recursively apply modularity principle
- Encapsulation – Hide implementation details from interfaces
- Regularity – Leverage structure at various levels of abstraction
- Extensibility – Include mechanisms/hooks to simplify future changes



Agenda

What is Computer Architecture?

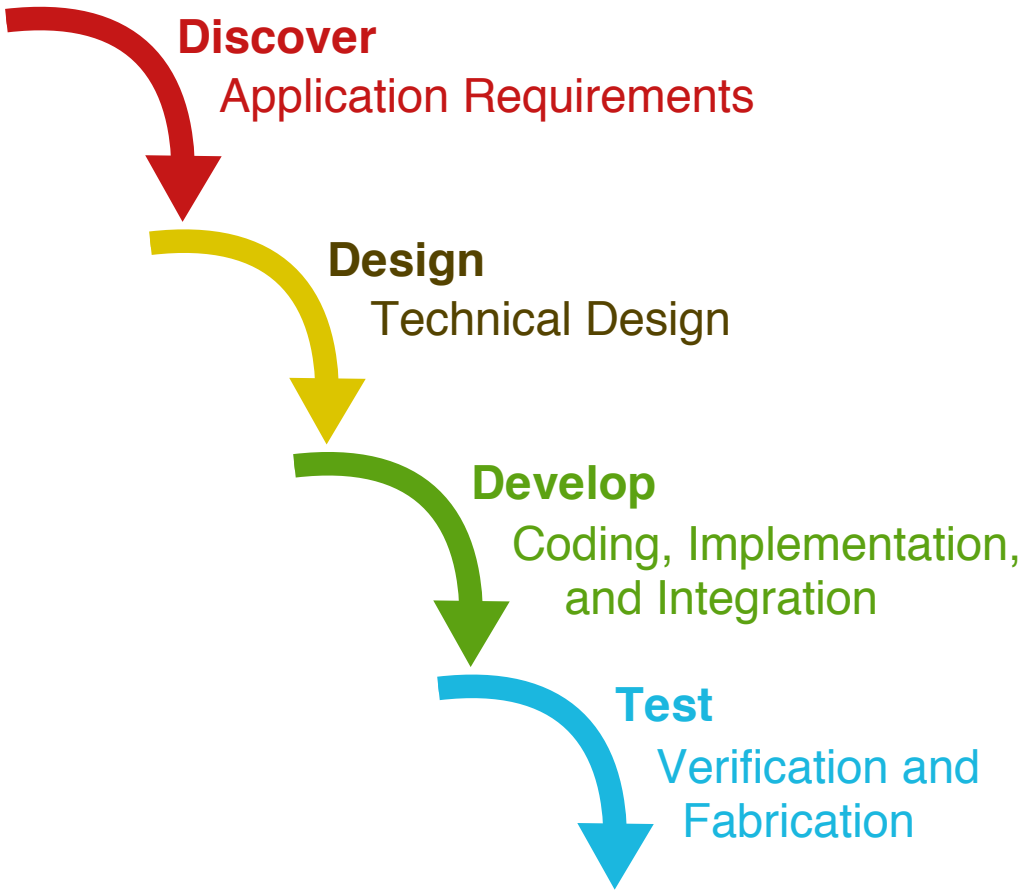
Design Example

Design Principles

Design Methodologies

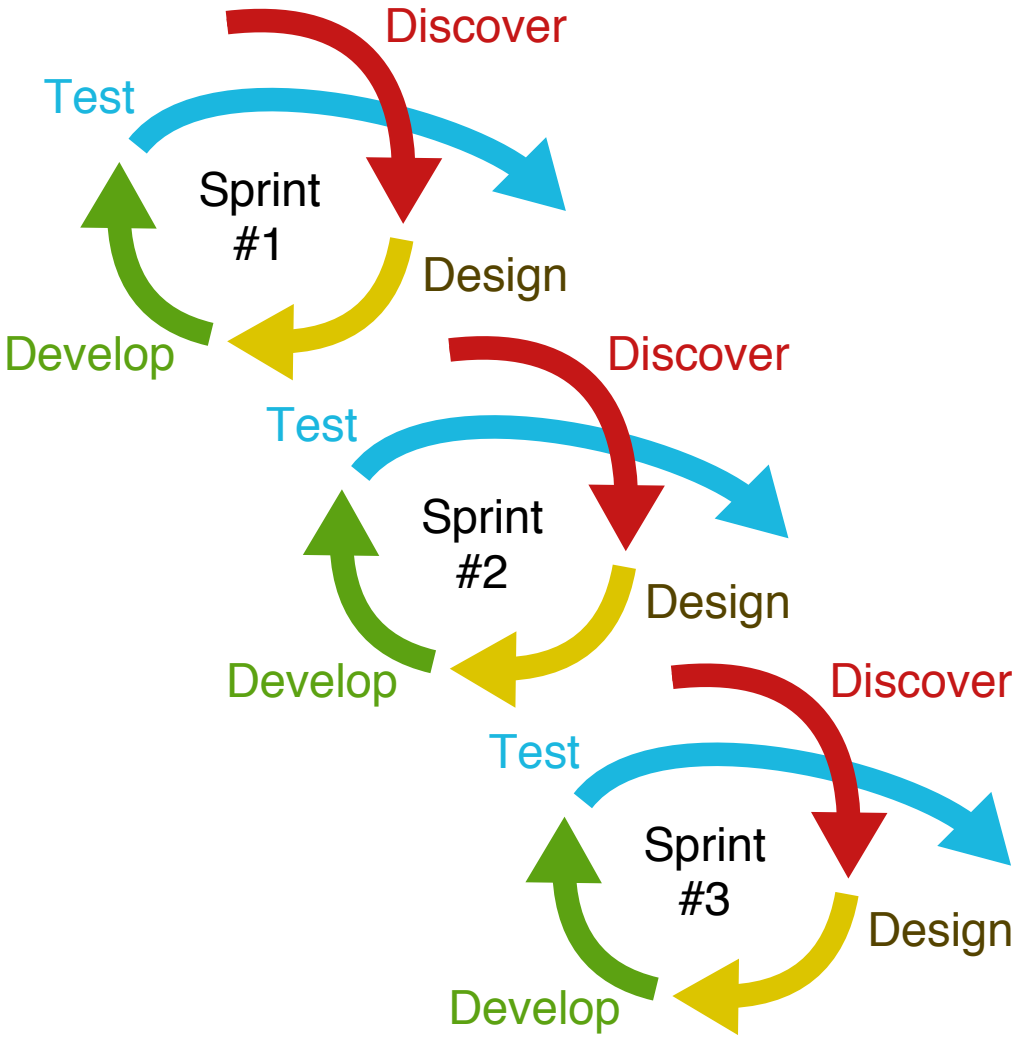
Waterfall Development Methodology

- Traditional sequential development methodology; each stage is completed before beginning the next stage
- For example, most of the design is completed before beginning the development, and most of the development is completed before beginning testing

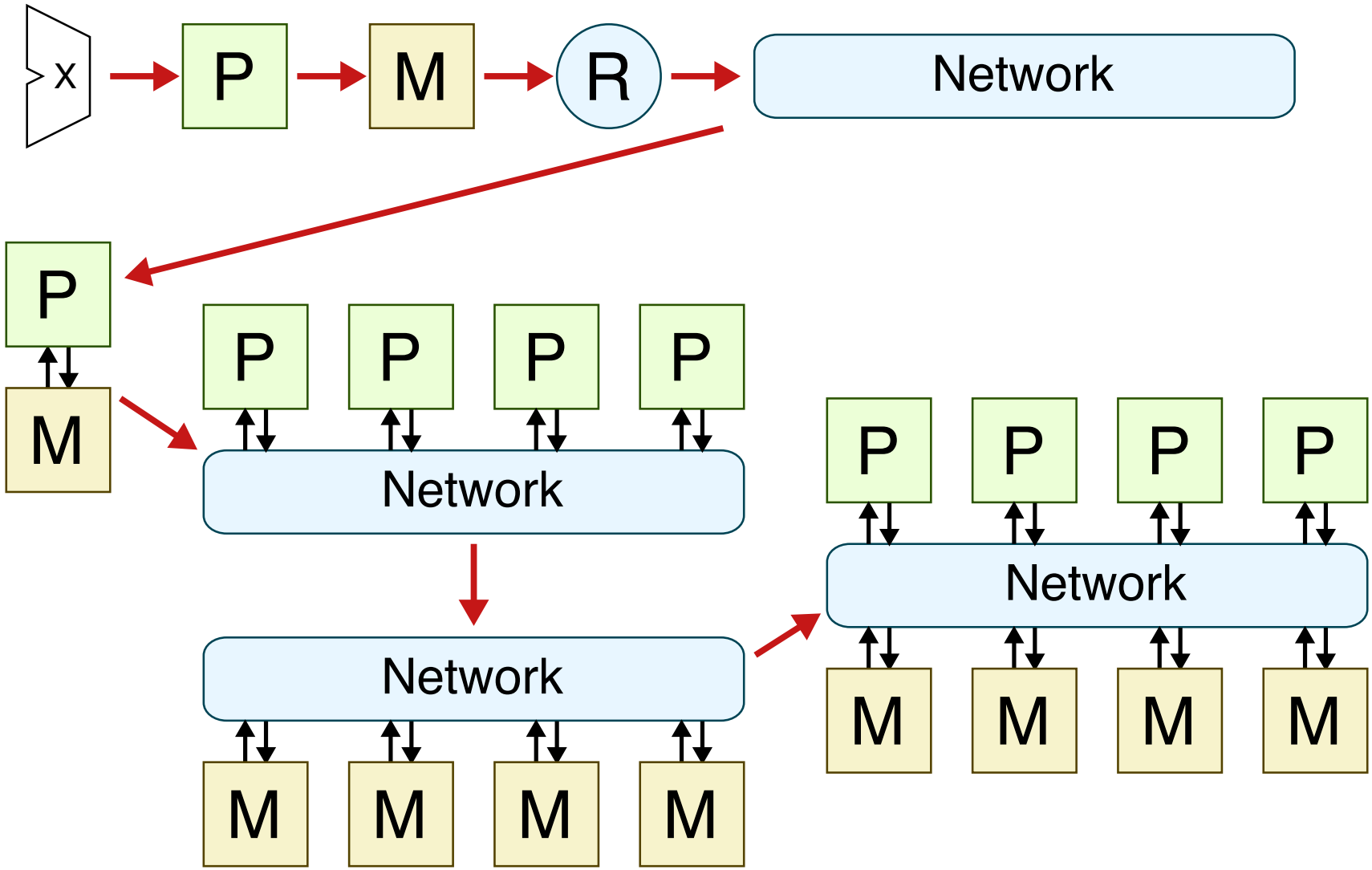


Agile Development Methodology

- Emerging iterative development methodology; move rapidly through all stages and then iterate back again through the stages
- For example, a component is designed, developed, and tested before moving onto another component; or a minimal yet complete system is designed, developed, and tested before incrementally adding features

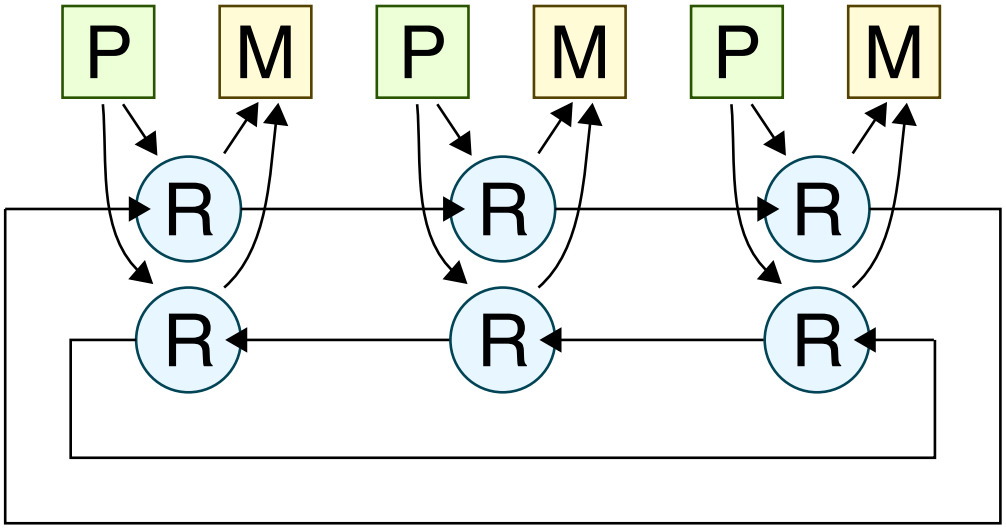


Agile Argues for Incremental Development



Agile Argues for Test-Driven Development

- Test Types
  - Unit tests vs. integration tests
  - Directed vs. random tests
  - Whitebox vs. blackbox tests
- Goal is to write tests first then implement design to pass these tests



- Write tests for higher level of abstraction, refine implementation until passes tests, add new tests
- Capture design bugs with new tests
- Use continuous integration to ensure that design changes do not result in regressions

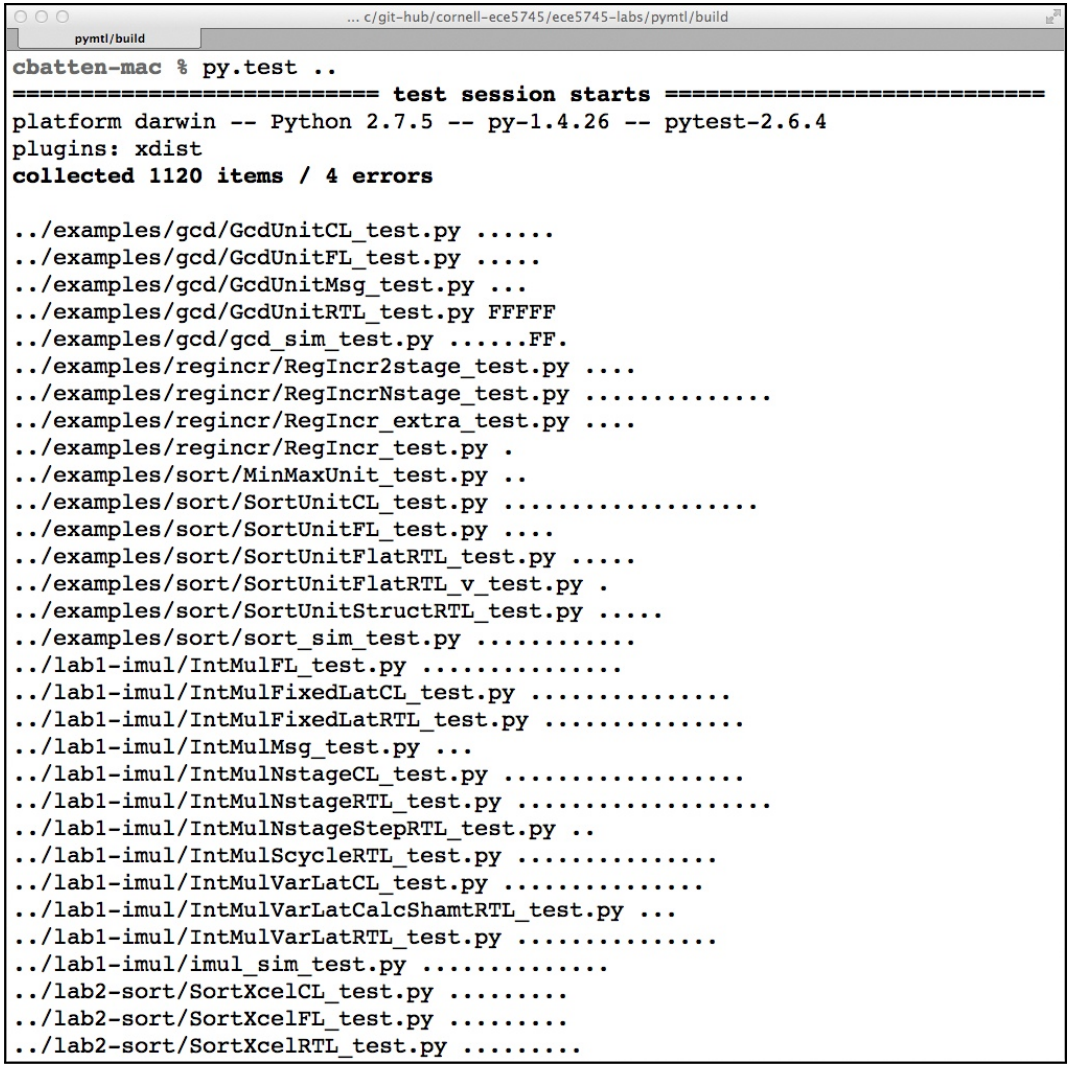
Waterfall vs. Agile Development Methodologies

Waterfall Methodology	Agile Methodology
Highly critical requirements	Less critical requirements
Less experienced engineers	More experienced engineers
Requirements change rarely	Requirements change often
Large engineering team	Small engineering team

A hybrid approach that includes aspects of both the traditional waterfall methodology and the emerging agile methodology is an attractive option for future hardware design projects.

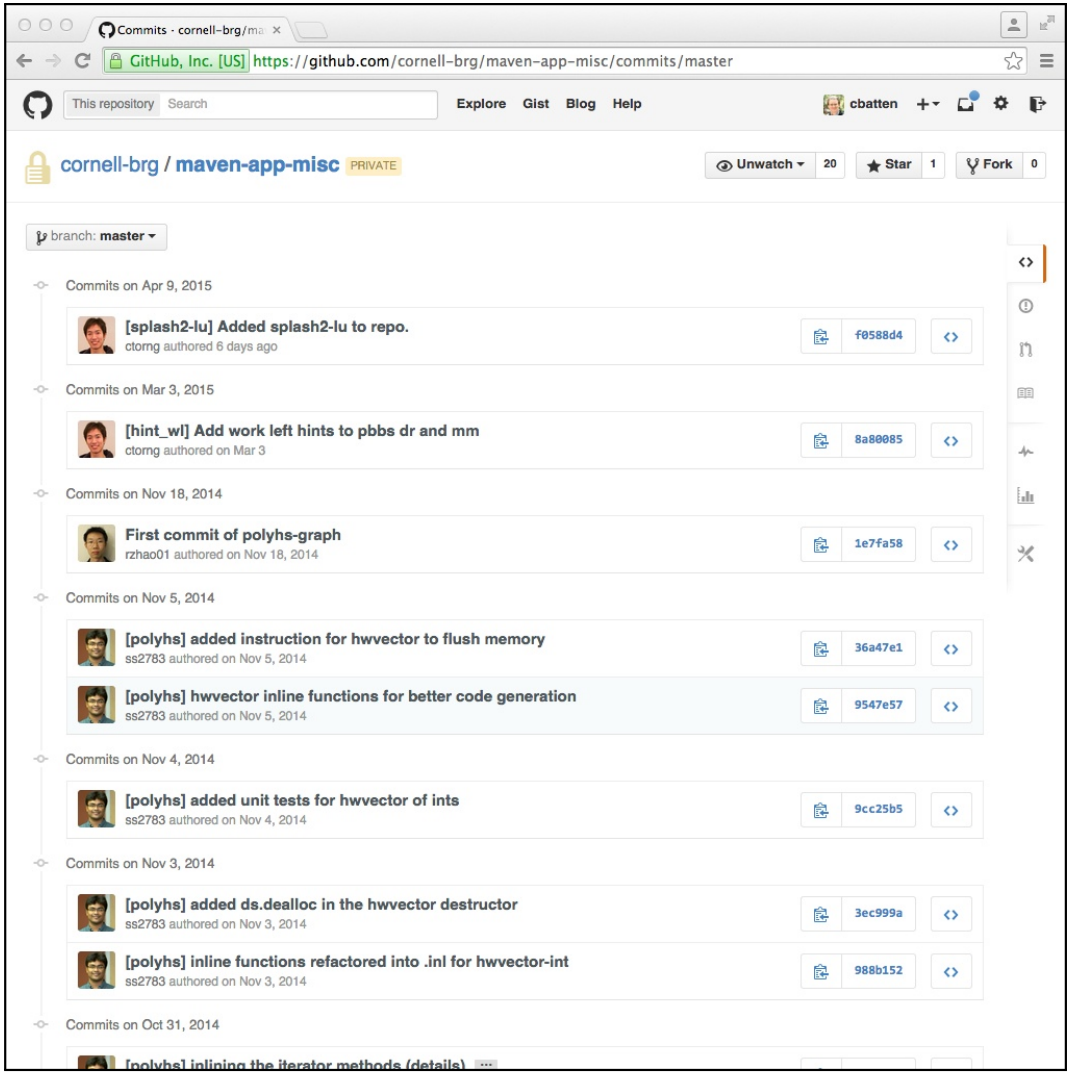
Tools for a Productive Development Methodology

- Command line (Linux, Max OS X)
- Powerful text editor (Emacs, VI)
- Scripting language (Python, Ruby, Perl)
- **Unit testing framework (py.test, UVM/OVM)**
- Distributed version control (Git, Mercurial)
- Cloud-based collaboration (GitHub, Bitbucket)
- Continuous integration (TravisCI)



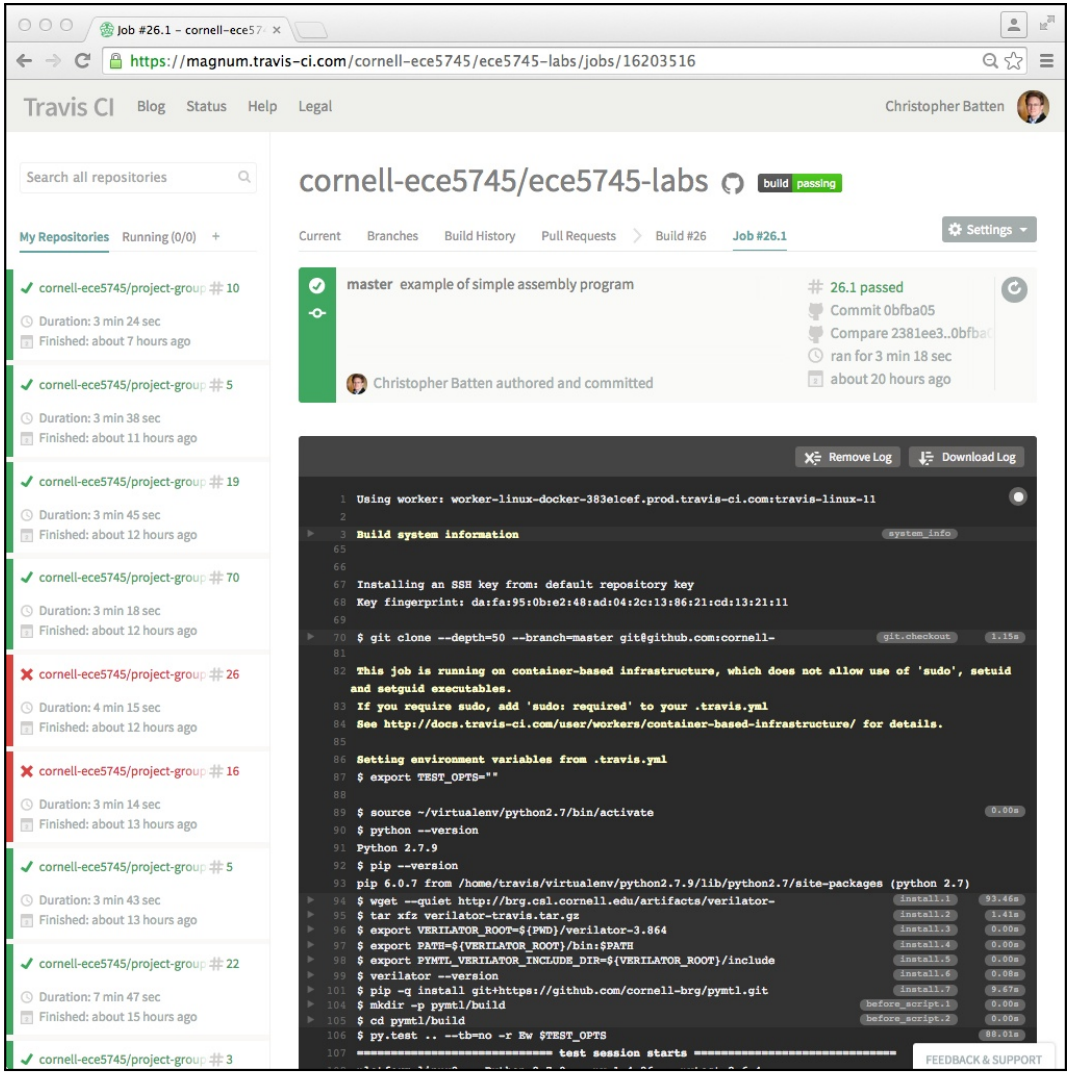
Tools for a Productive Development Methodology

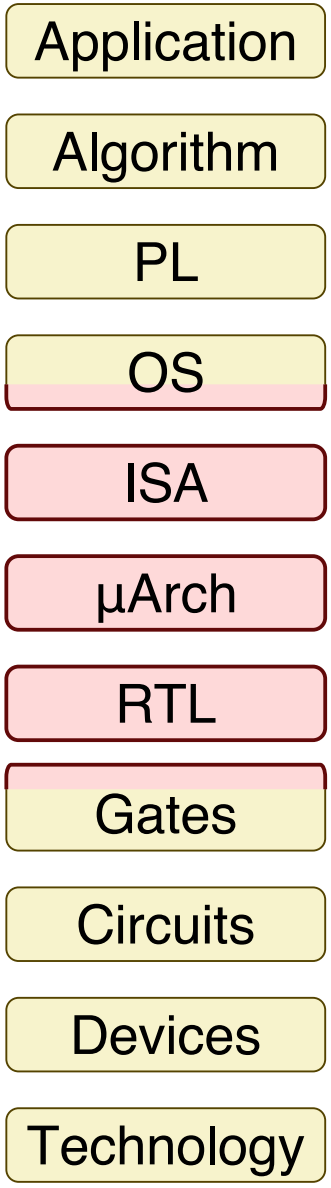
- Command line (Linux, Max OS X)
- Powerful text editor (Emacs, VI)
- Scripting language (Python, Ruby, Perl)
- Unit testing framework (py.test, UVM/OVM)
- Distributed version control (Git, Mercurial)
- **Cloud-based collaboration (GitHub, Bitbucket)**
- Continuous integration (TravisCI)



Tools for a Productive Development Methodology

- Command line (Linux, Max OS X)
- Powerful text editor (Emacs, VI)
- Scripting language (Python, Ruby, Perl)
- Unit testing framework (py.test, UVM/OVM)
- Distributed version control (Git, Mercurial)
- Cloud-based collaboration (GitHub, Bitbucket)
- **Continuous integration (TravisCI)**





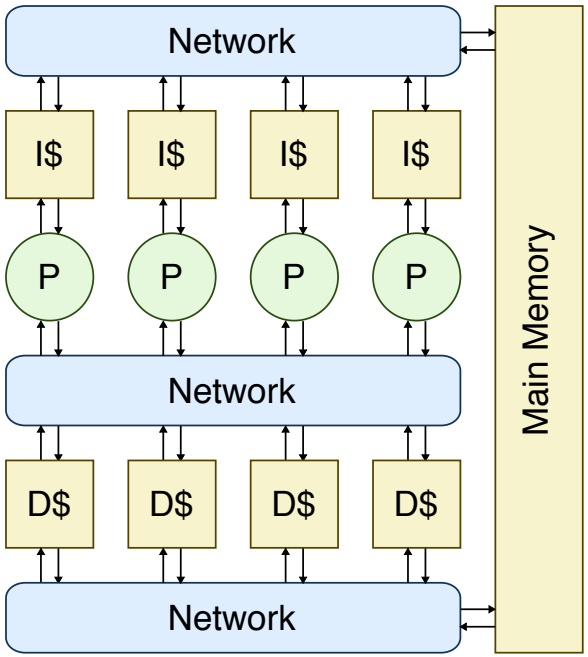
Take-Away Points

- Computer engineering is an iterative process involving designing and modeling systems, evaluating trade-offs between various design alternatives, which in turn motivates designing and modeling new systems
- Design principles such as modularity, hierarchy, modularity, encapsulation, regularity, and extensibility and design methodologies such as waterfall and agile hardware development can help manage the significant complexity inherent in building modern computing systems

ECE 4750 Computer Architecture

http://www.csl.cornell.edu/courses/ece4750

- Part 1: Fundamental Processors – FSM processors; pipelined processors; structural, data, and control hazards
- Part 2: Fundamental Memories – memory technology; cache hierarchies; pipelined cache microarchitecture
- Part 3: Fundamental Networks – torus and butterfly topologies; routing algorithms; flow control; pipelined router microarchitecture



- Part 4: Advanced Processors – superscalar execution; branch prediction; out-of-order execution; register renaming; memory disambiguation; VLIW, vector, and multithreaded processors
- Part 5: Advanced Memories – non-blocking caches; memory coherence, synchronization, consistency; memory translation, protection, virtualization