

**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

Application

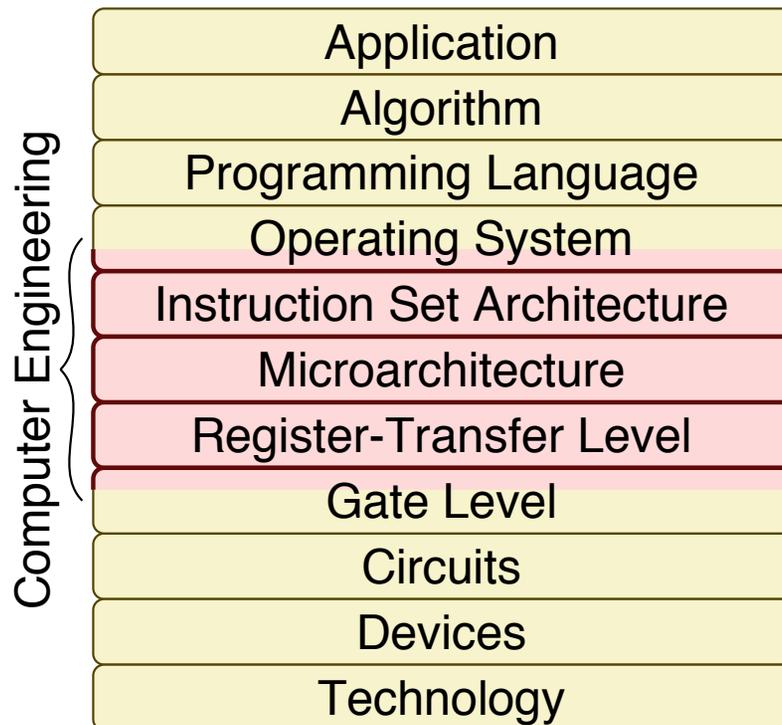


Gap too large to bridge in one step  
(but there are exceptions,  
e.g., a magnetic compass)



Technology

# The Computer Engineering Stack



## Sort an array of numbers

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

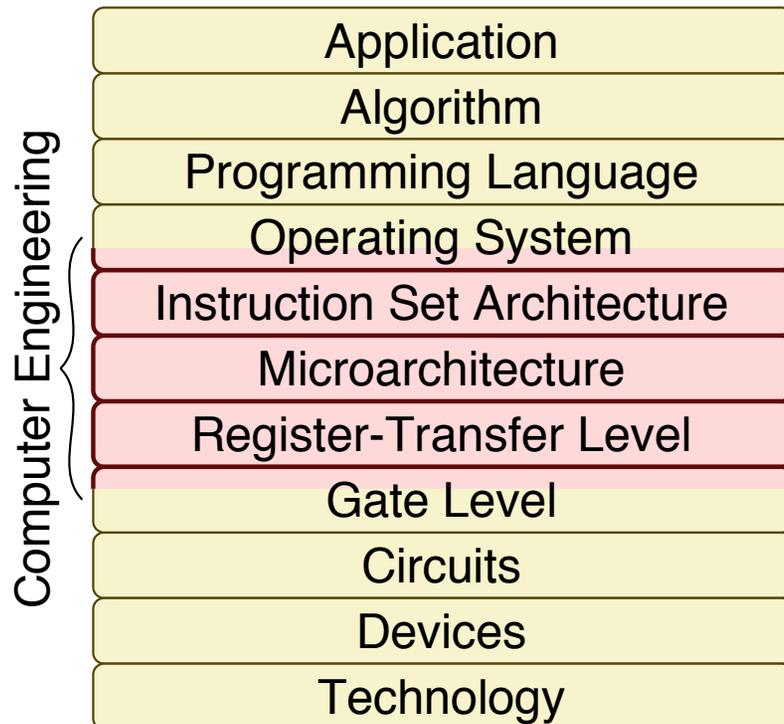
## Insertion sort algorithm

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

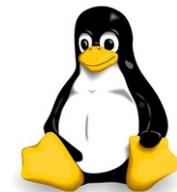
```
void isort( int b[], int a[], int n ) {
    for ( int idx, k = 0; k < n; k++ ) {
        int min = 100;
        for ( int i = 0; i < n; i++ ) {
            if ( a[i] < min ) {
                min = a[i];
                idx = i;
            }
        }
        b[k] = min;
        a[idx] = 100;
 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

# The Computer Engineering Stack



**Mac OS X, Windows, Linux**

Handles low-level hardware management



**MIPS32 Instruction Set**

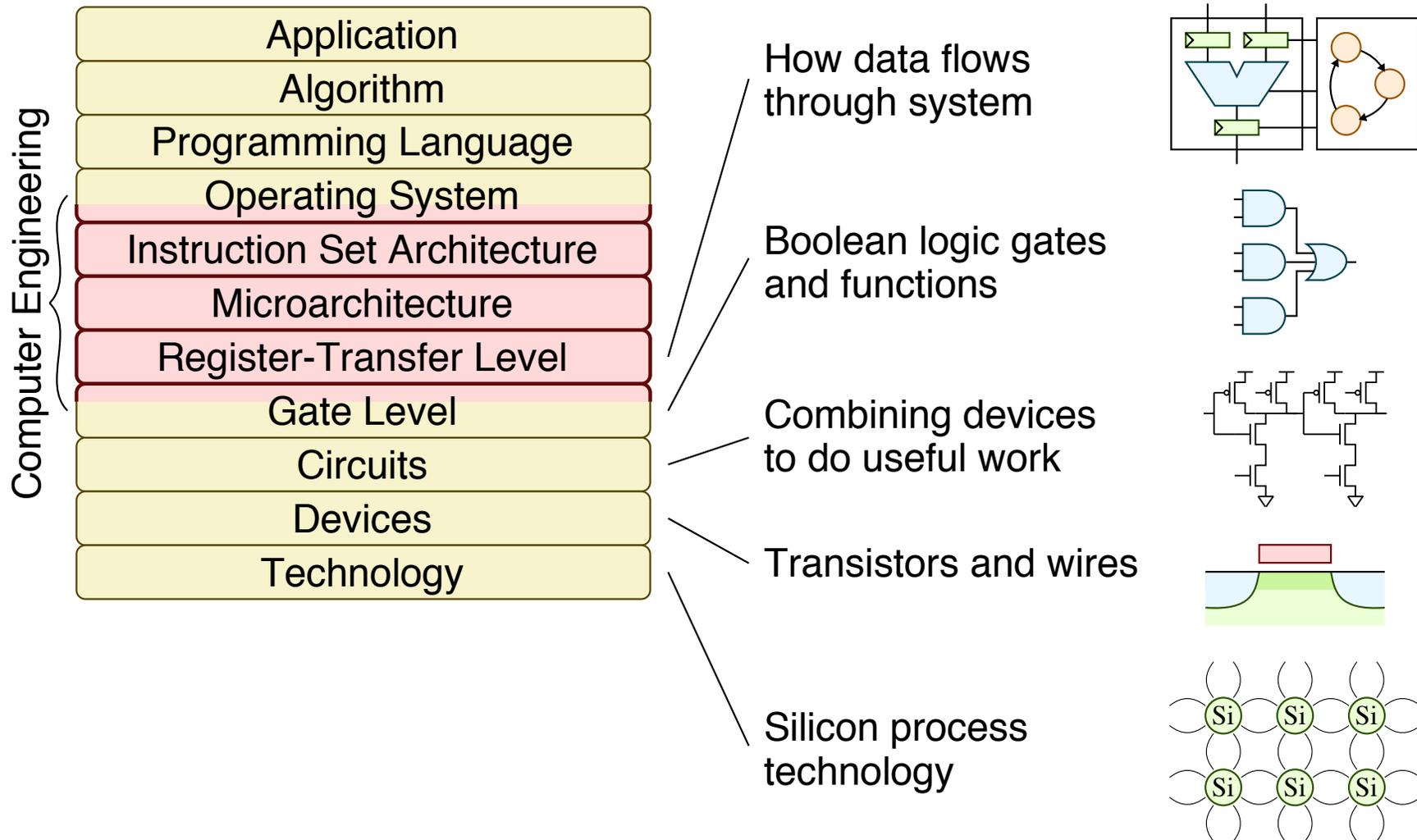
Instructions that machine executes

```

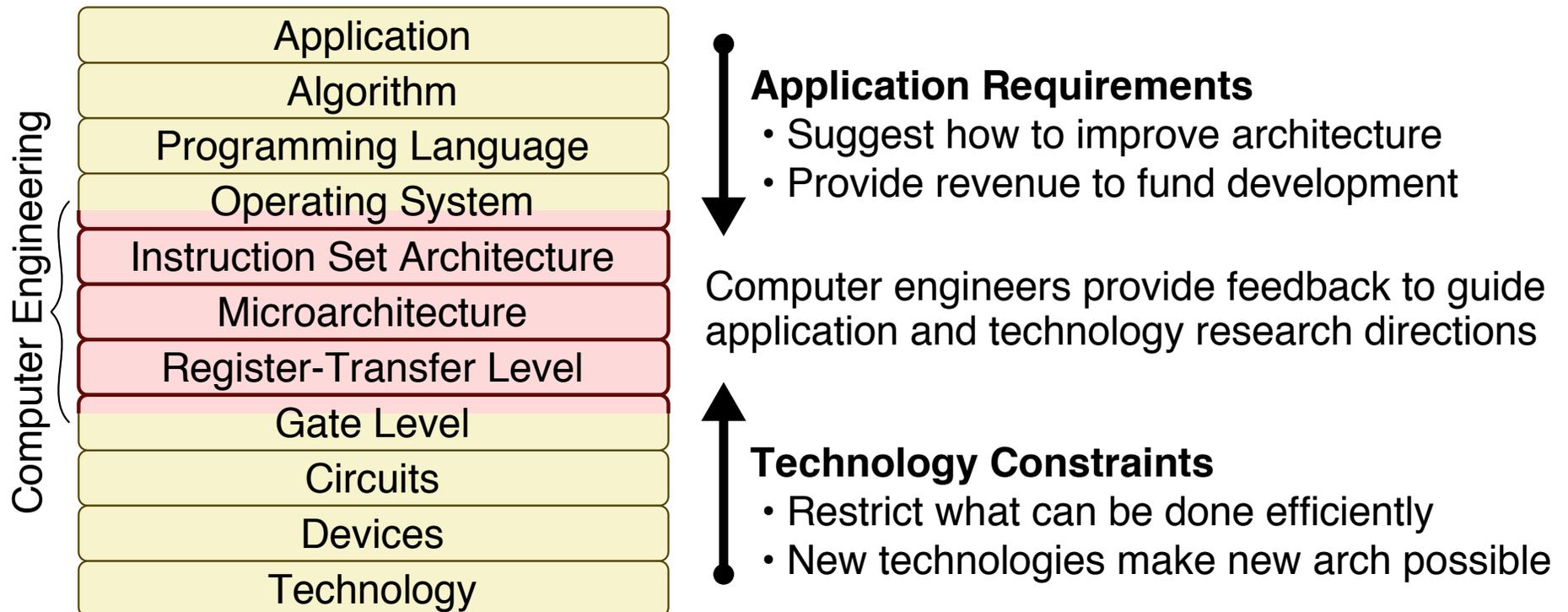
blez  $a2, done
move  $a7, $zero
li    $t4, 99
move  $a4, $a1
move  $v1, $zero
li    $a3, 99
lw    $a5, 0($a4)
addiu $a4, $a4, 4
slt   $a6, $a5, $a3
movn  $v0, $v1, $a6
addiu $v1, $v1, 1
movn  $a3, $a5, $a6

```

# The Computer Engineering Stack

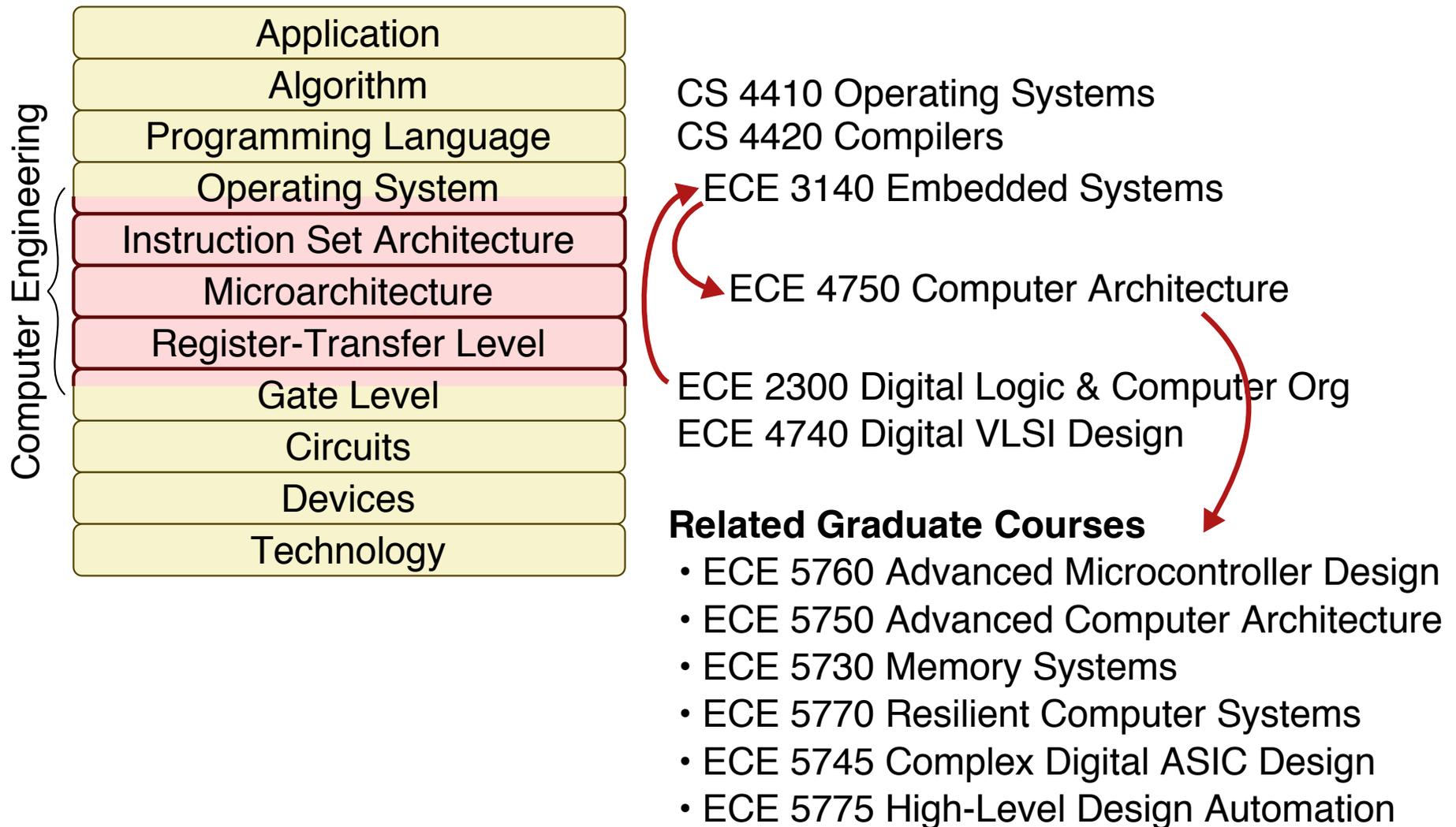


# What is Computer Architecture?

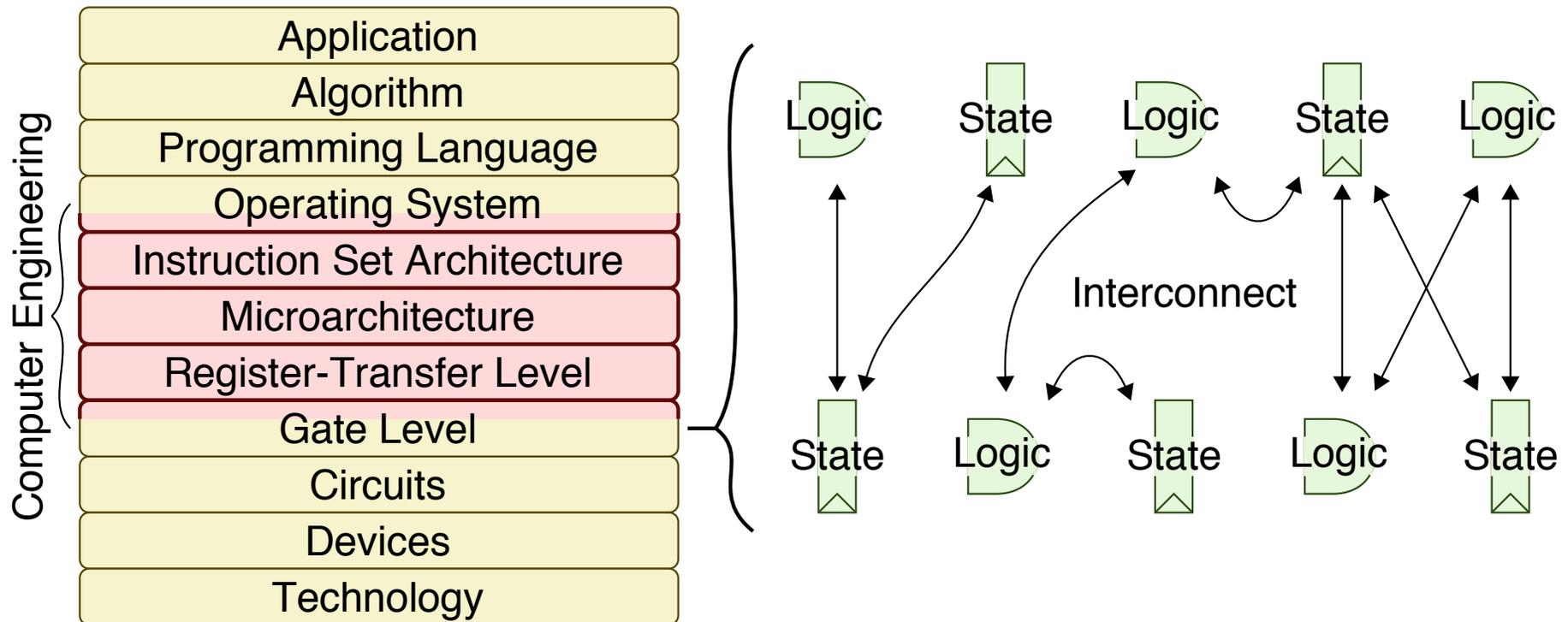


In its broadest definition, computer engineering is the **development of the abstraction/implementation layers** that allow us to execute information processing **applications** efficiently using available manufacturing **technologies**

# Computer Architecture in the ECE/CS Curriculum



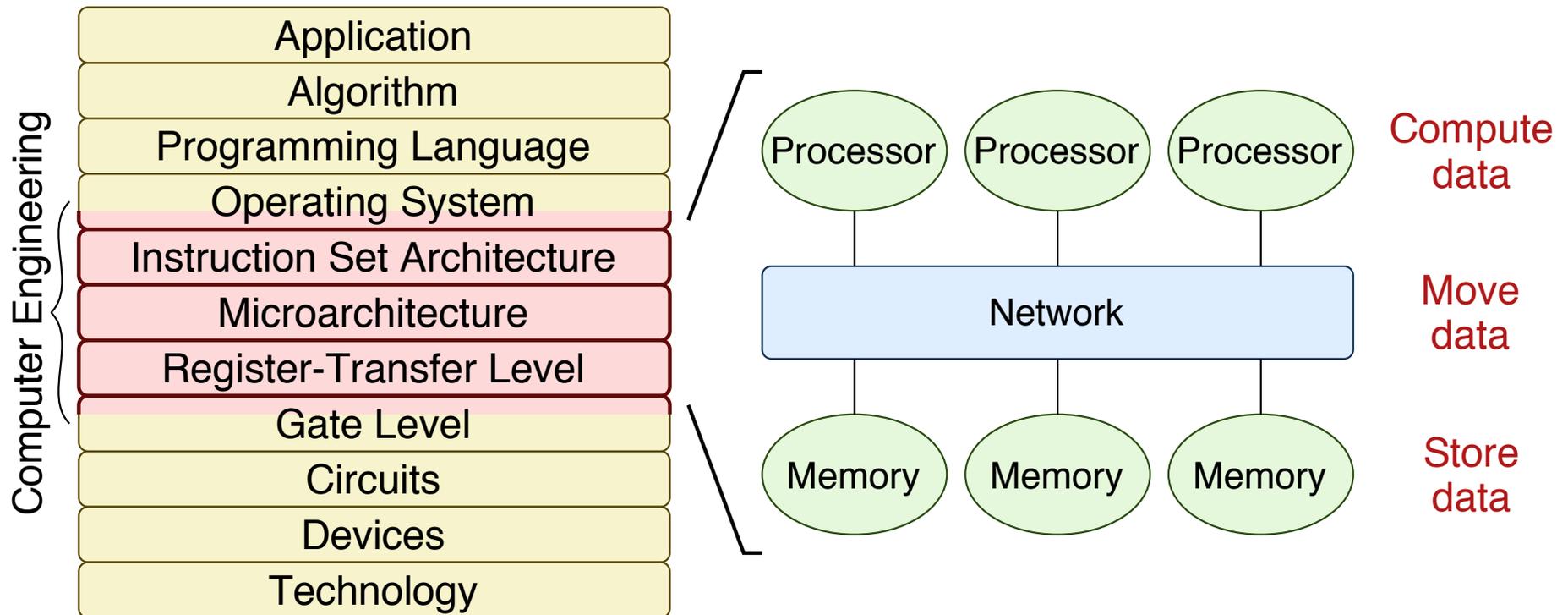
# Logic, State, and Interconnect



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

Application

Algorithm

PL

OS

ISA

$\mu$ Arch

RTL

Gates

Circuits

Devices

Technology

# 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

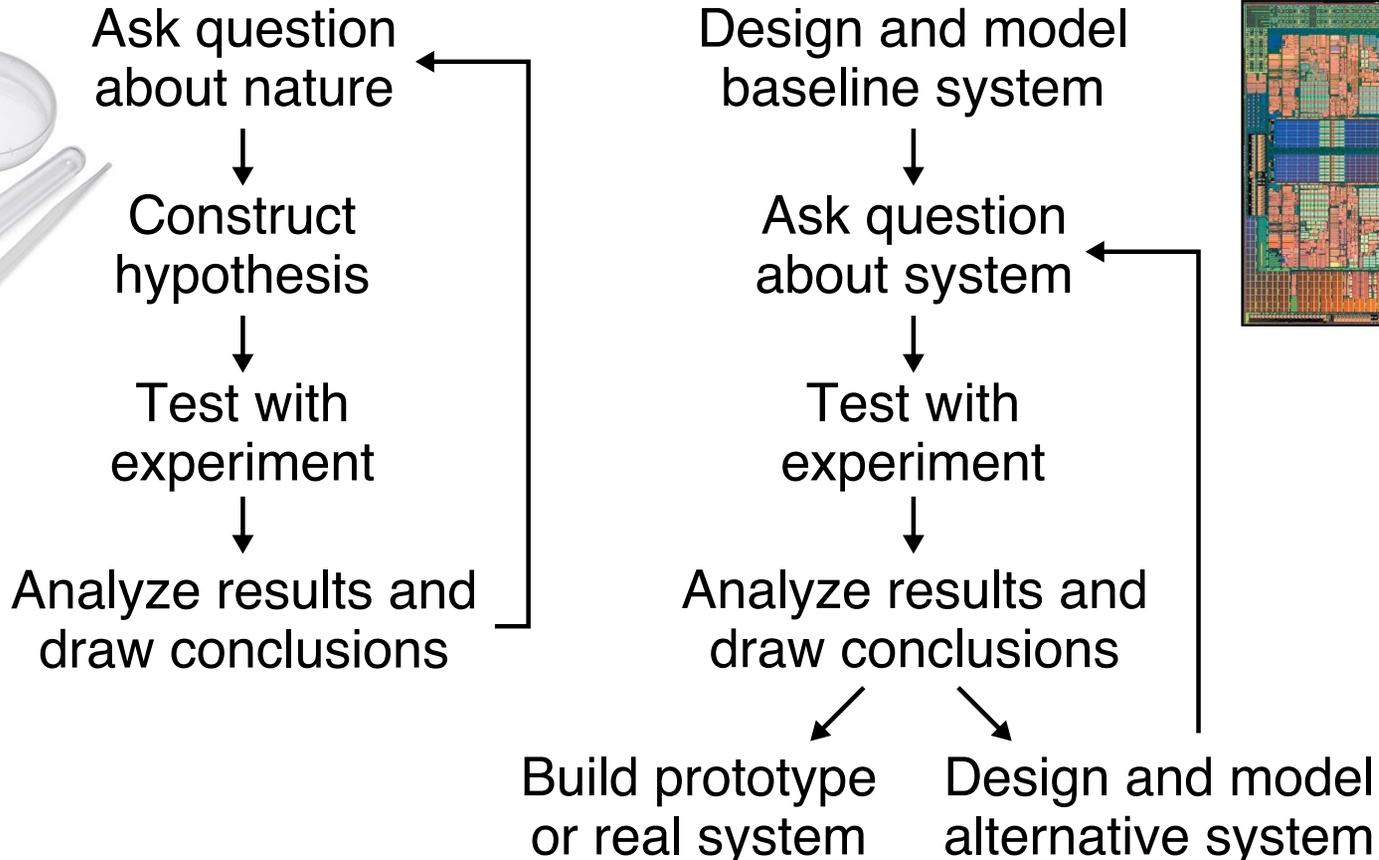
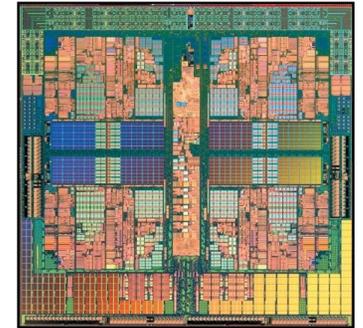
Ask question about system

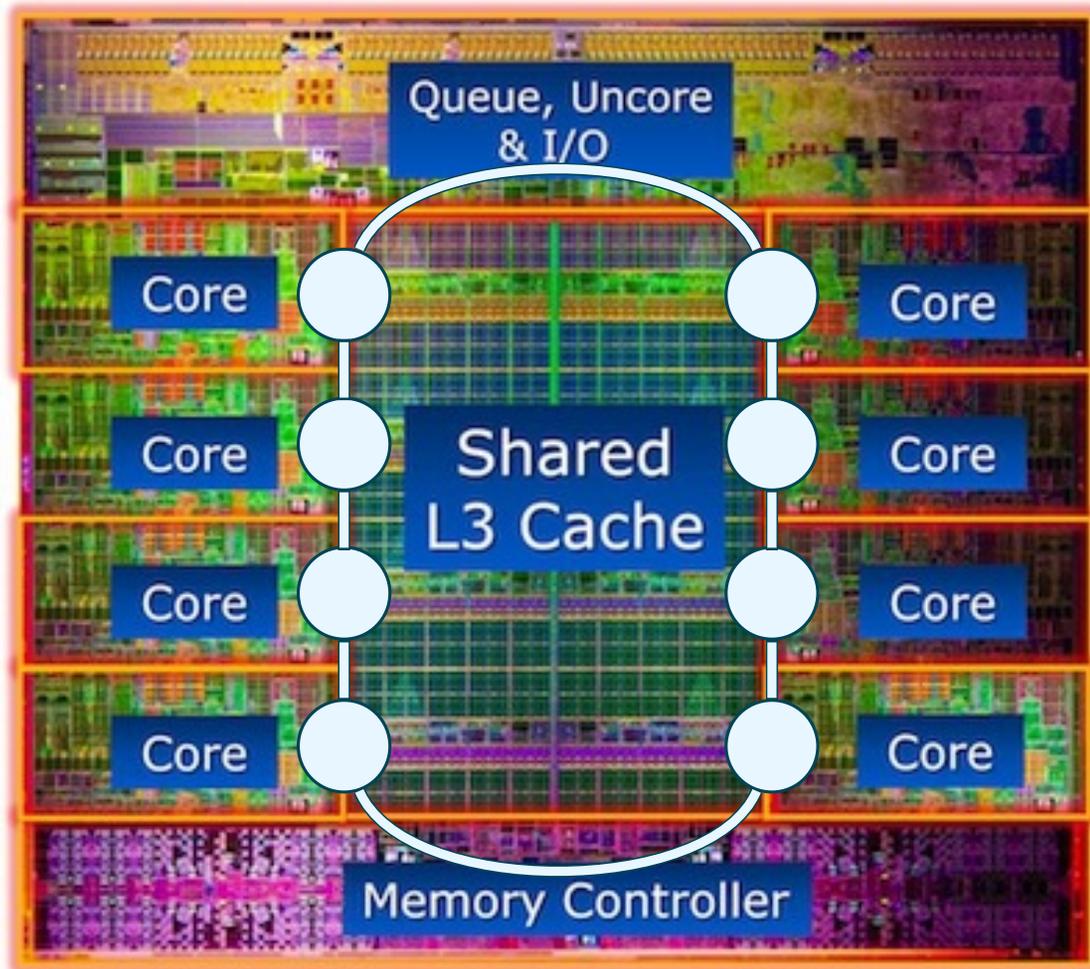
Test with experiment

Analyze results and draw conclusions

Build prototype or real system

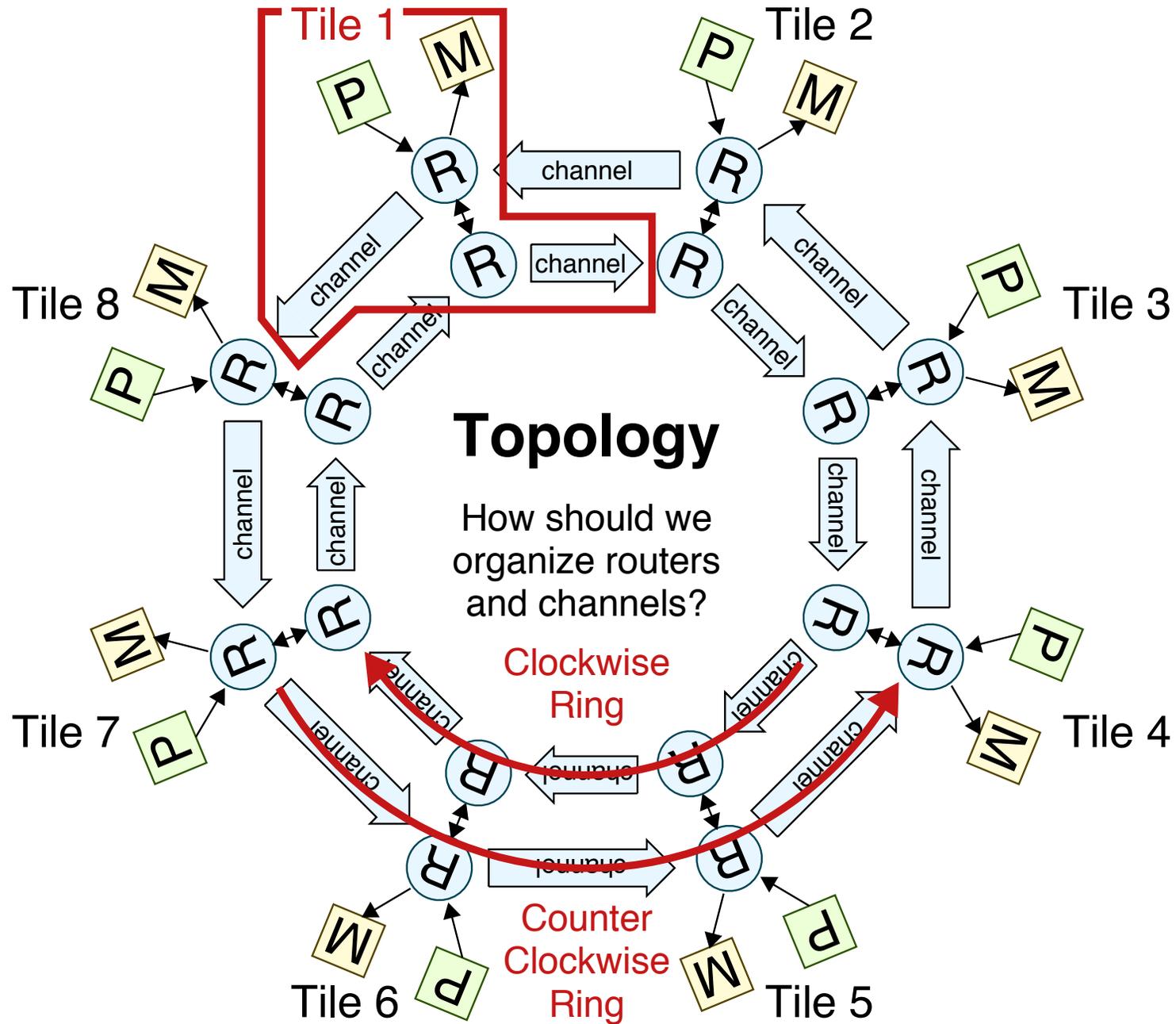
Design and model alternative system

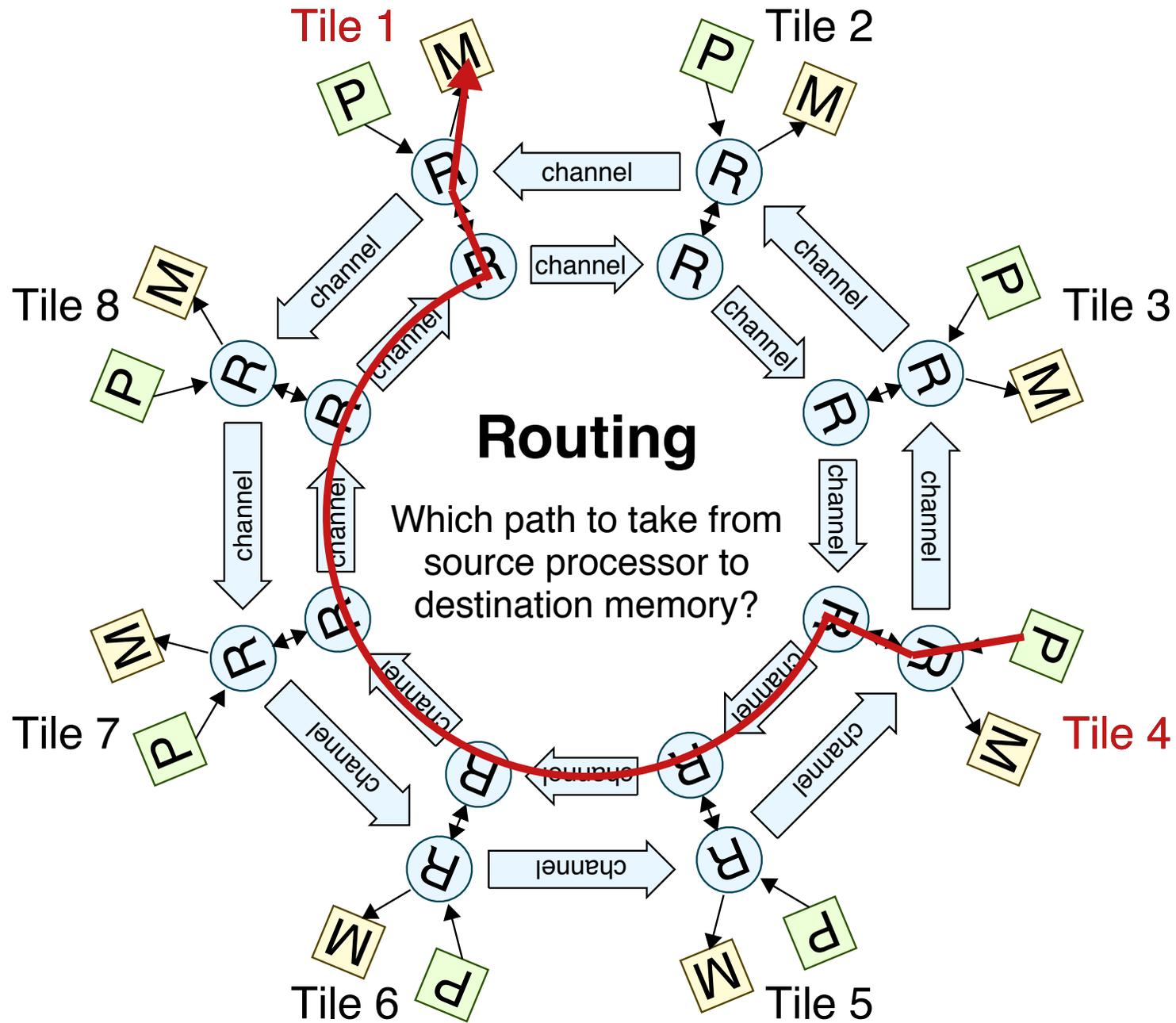


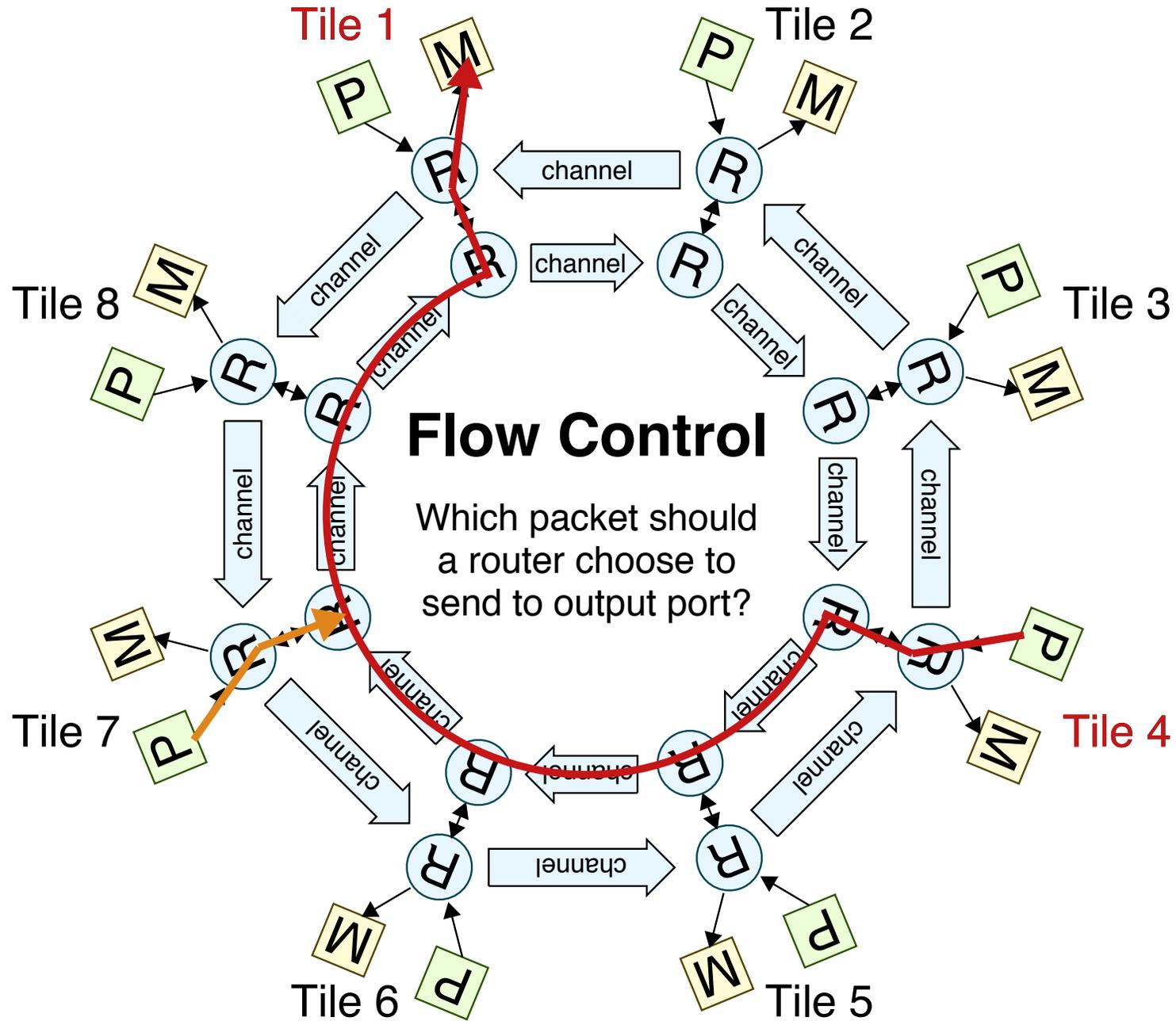


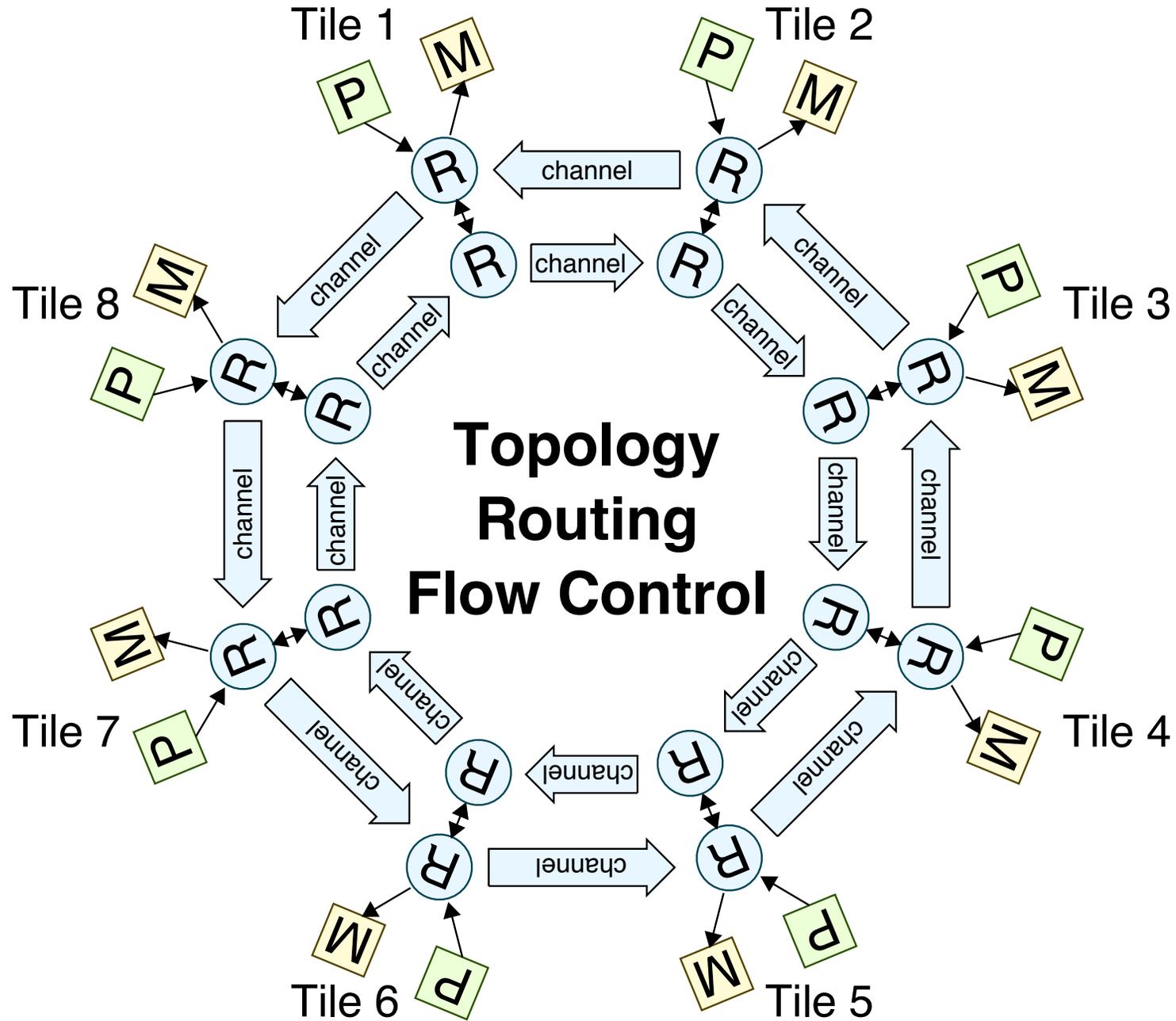
## Example Design Problem: On-Chip Interconnection Network

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









# Modeling in Computer Architecture

## 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

```
// 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
```

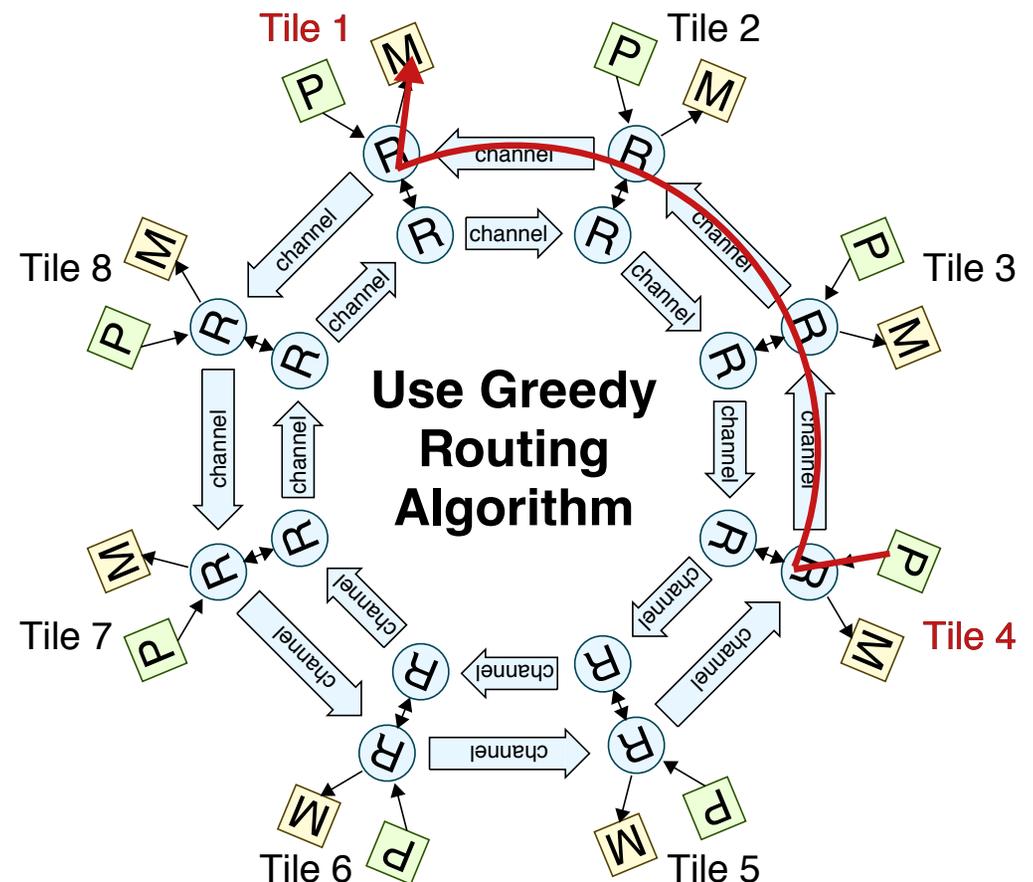
**Verilog • SystemVerilog • VHDL**

**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

cycle	Counter Clockwise	From 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

cycle	Counter Clockwise	From 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

Design and model  
baseline system

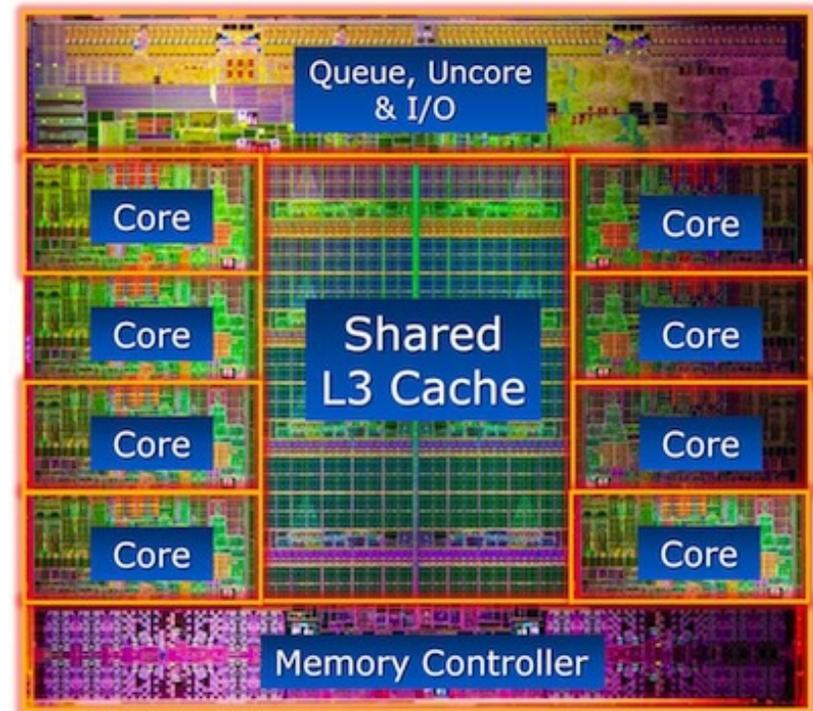
↓  
Ask question  
about system

↓  
Test with  
experiment

↓  
Analyze results and  
draw conclusions

↙  
Build prototype  
or real system

↘  
Design and model  
alternative system



**Fighter Airplane:** ~100,000 parts

**Intel Sandy Bridge E:**  
2.27 Billion transistors

Application

Algorithm

PL

OS

ISA

$\mu$ Arch

RTL

Gates

Circuits

Devices

Technology

# Agenda

---

What is Computer Architecture?

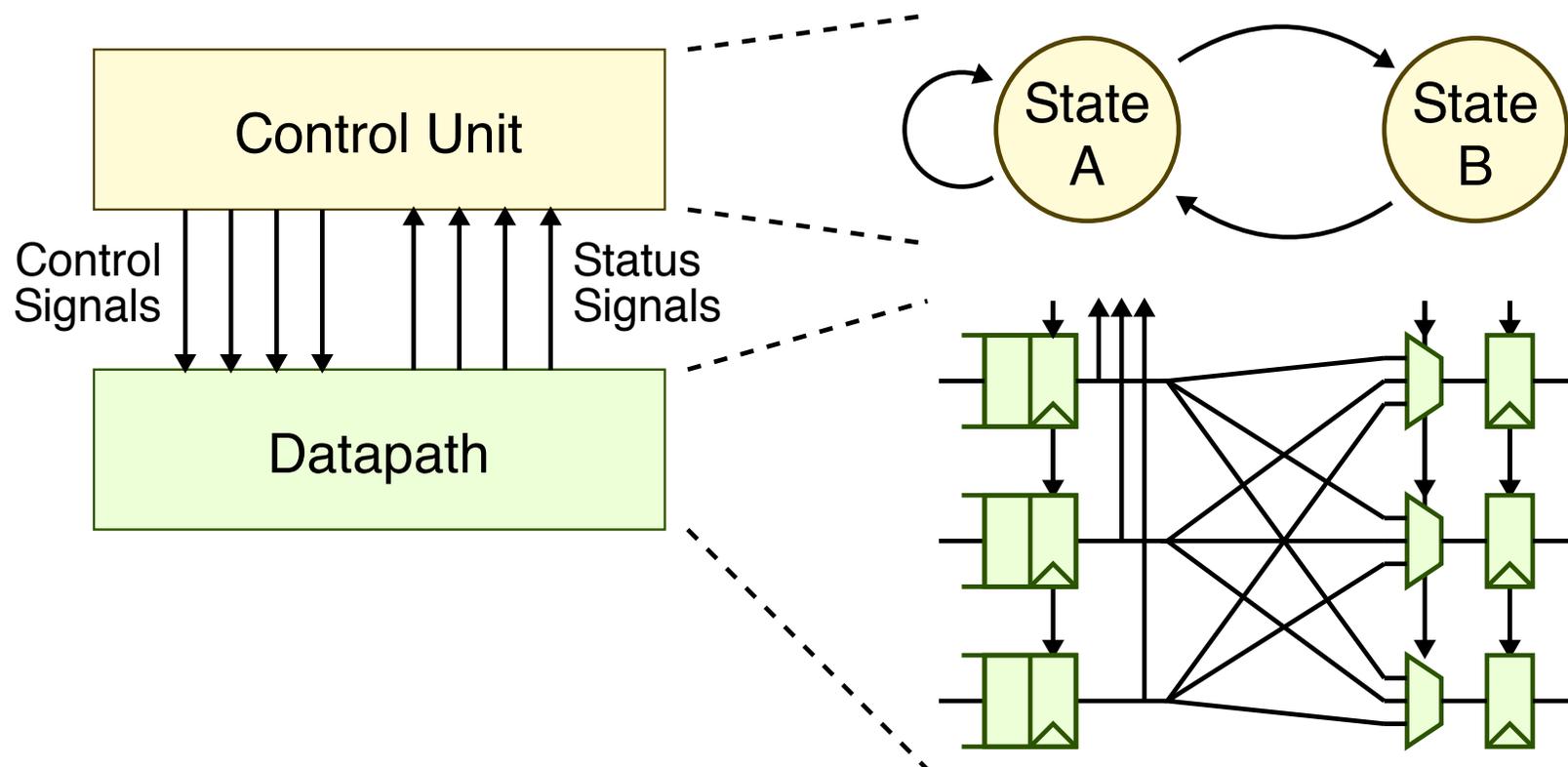
Design Example

**Design Principles**

Design Methodologies

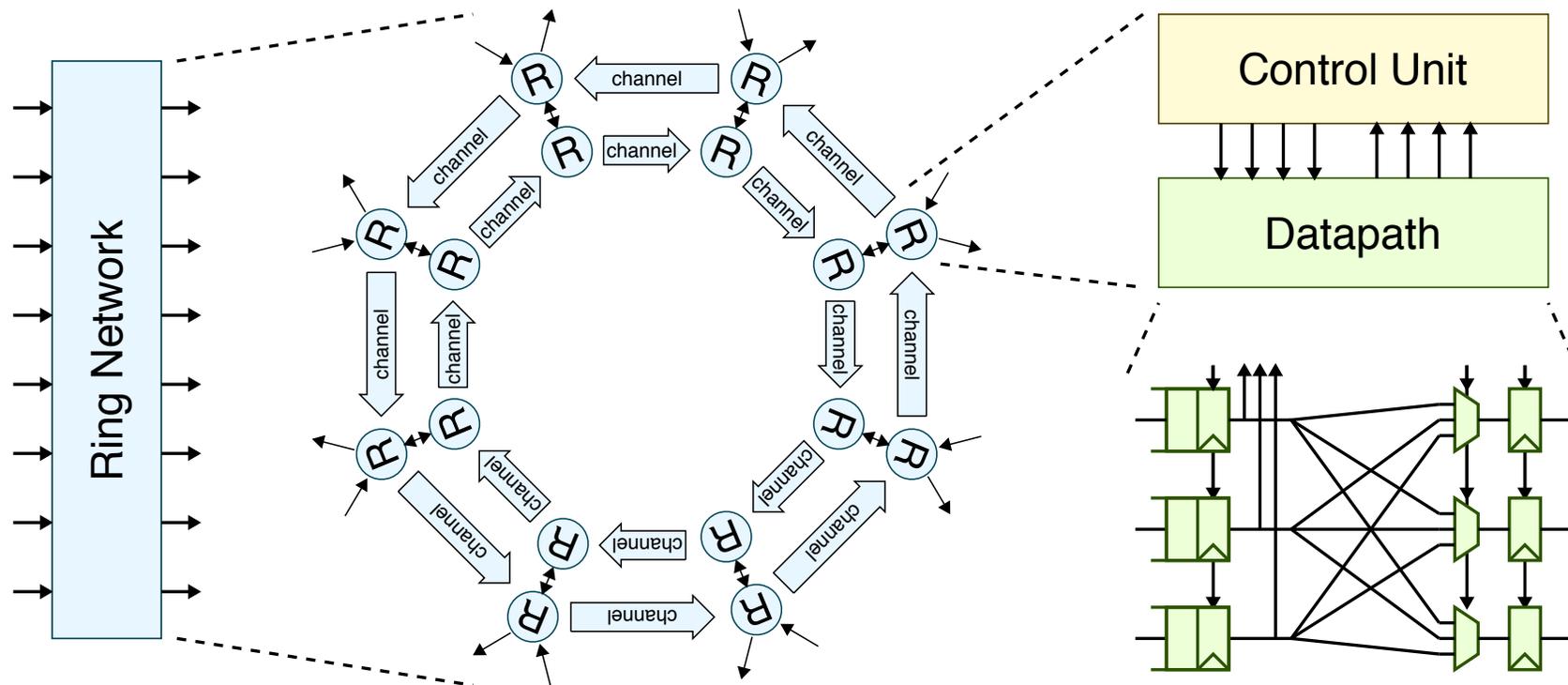
# Design Principle: Modularity

- ▶ 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



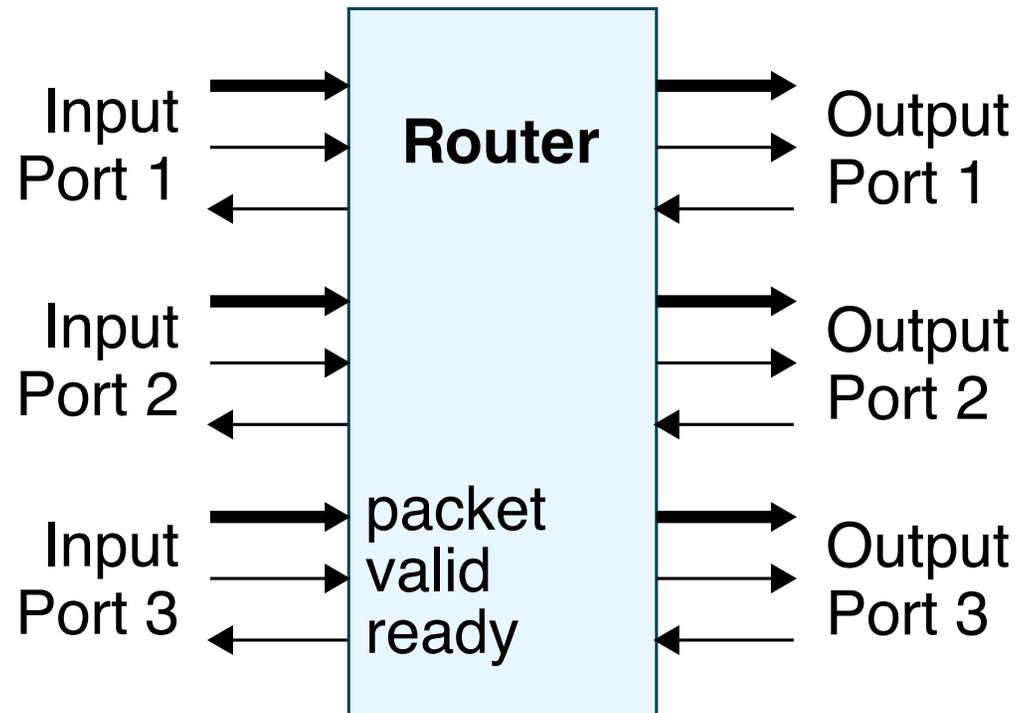
# Design Principle: Hierarchy

- ▶ 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



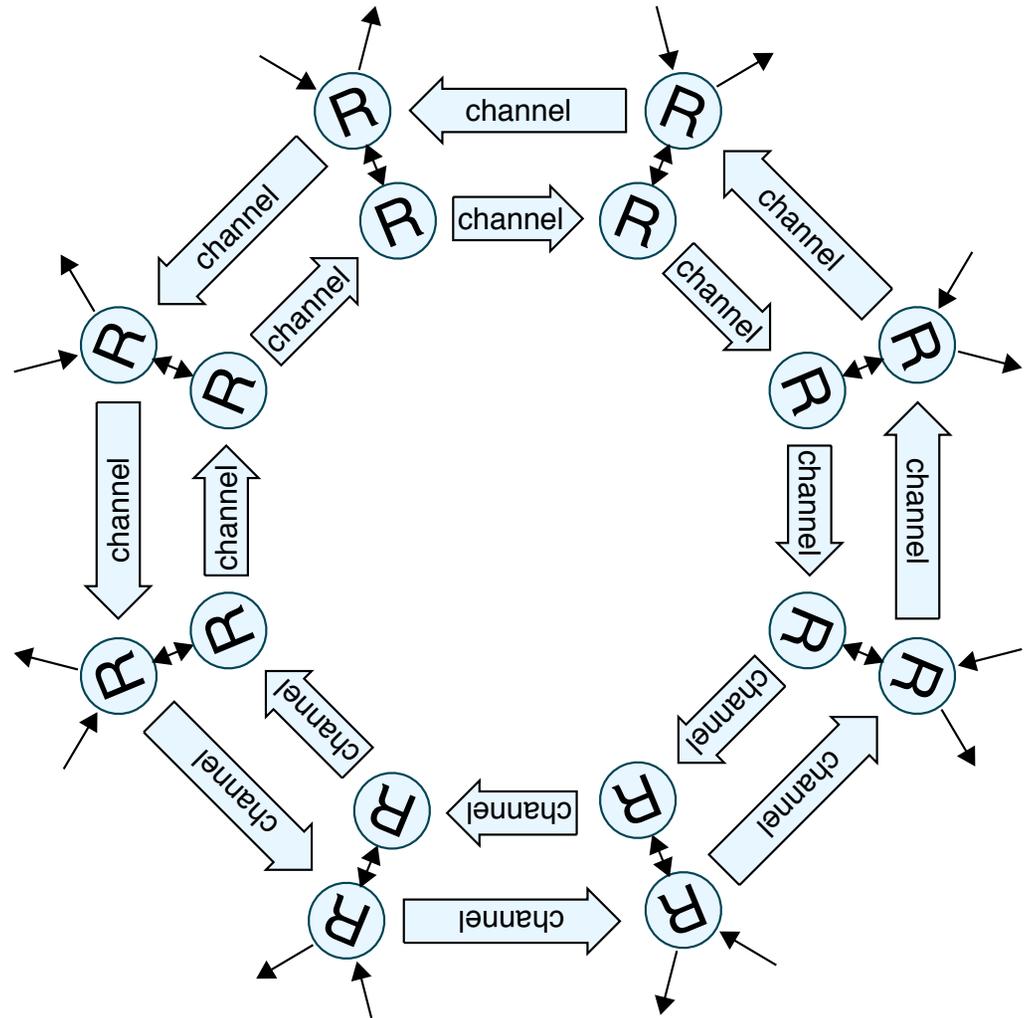
# Design Principle: Encapsulation

- ▶ 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



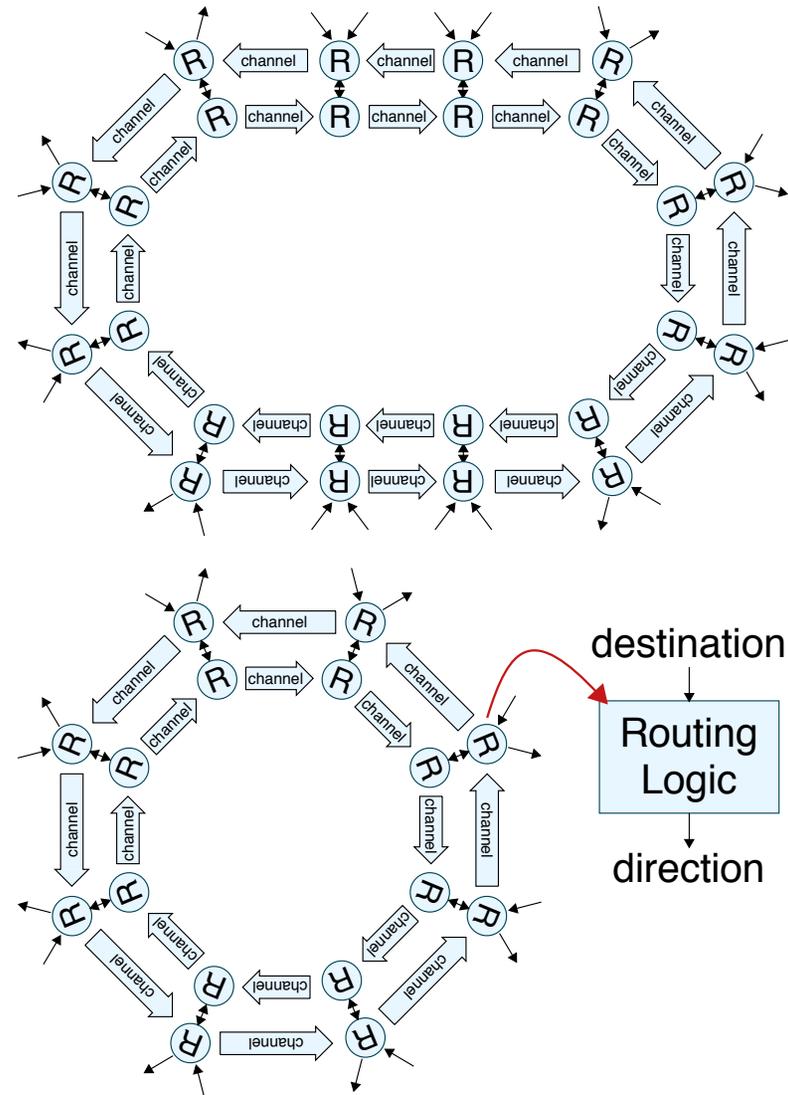
# Design Principle: Regularity

- ▶ 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

Application

Algorithm

PL

OS

ISA

 $\mu$ Arch

RTL

Gates

Circuits

Devices

Technology

# Agenda

---

What is Computer Architecture?

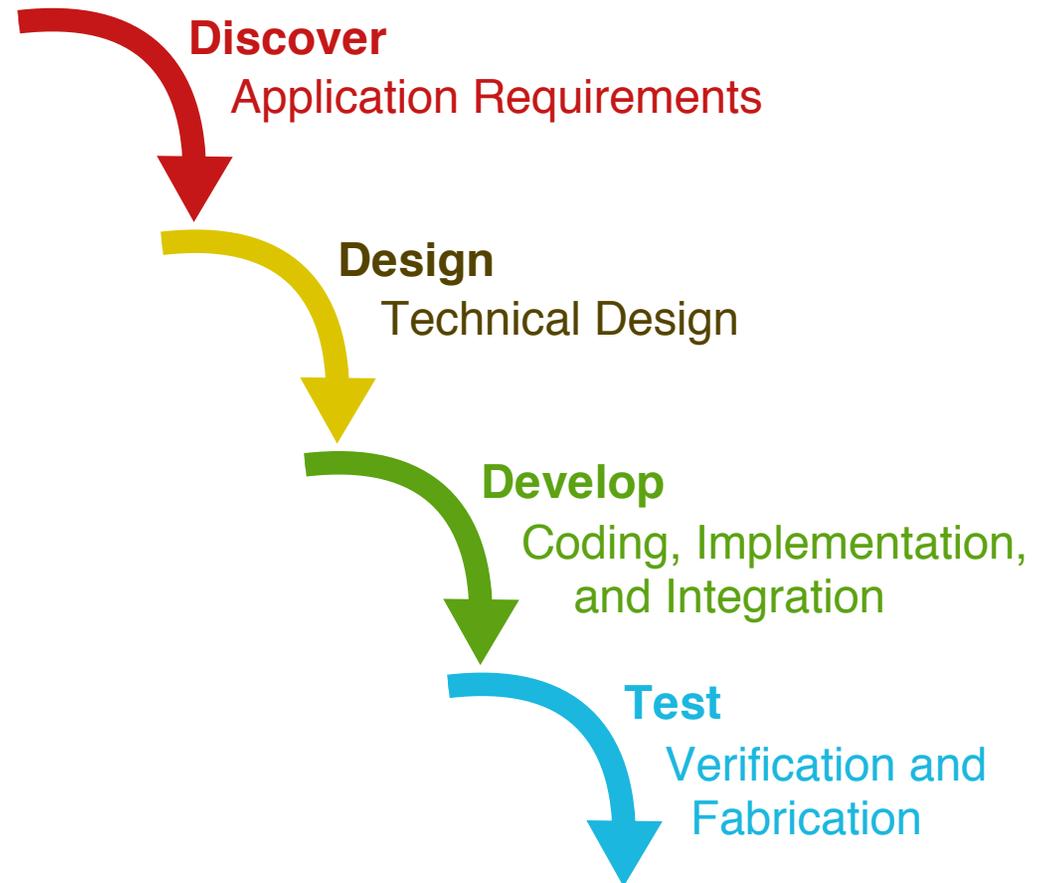
Design Example

Design Principles

Design Methodologies

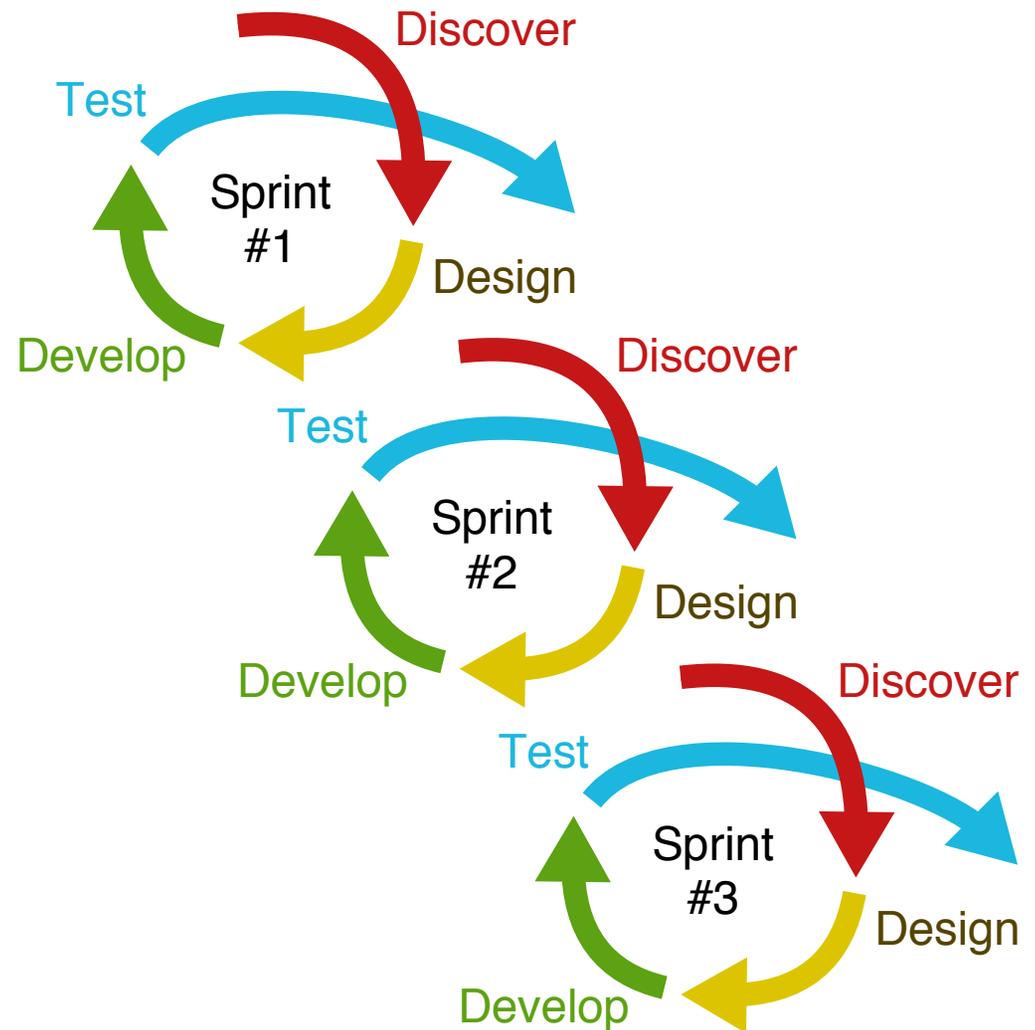
# Waterfall Development Methodology

- ▶ 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

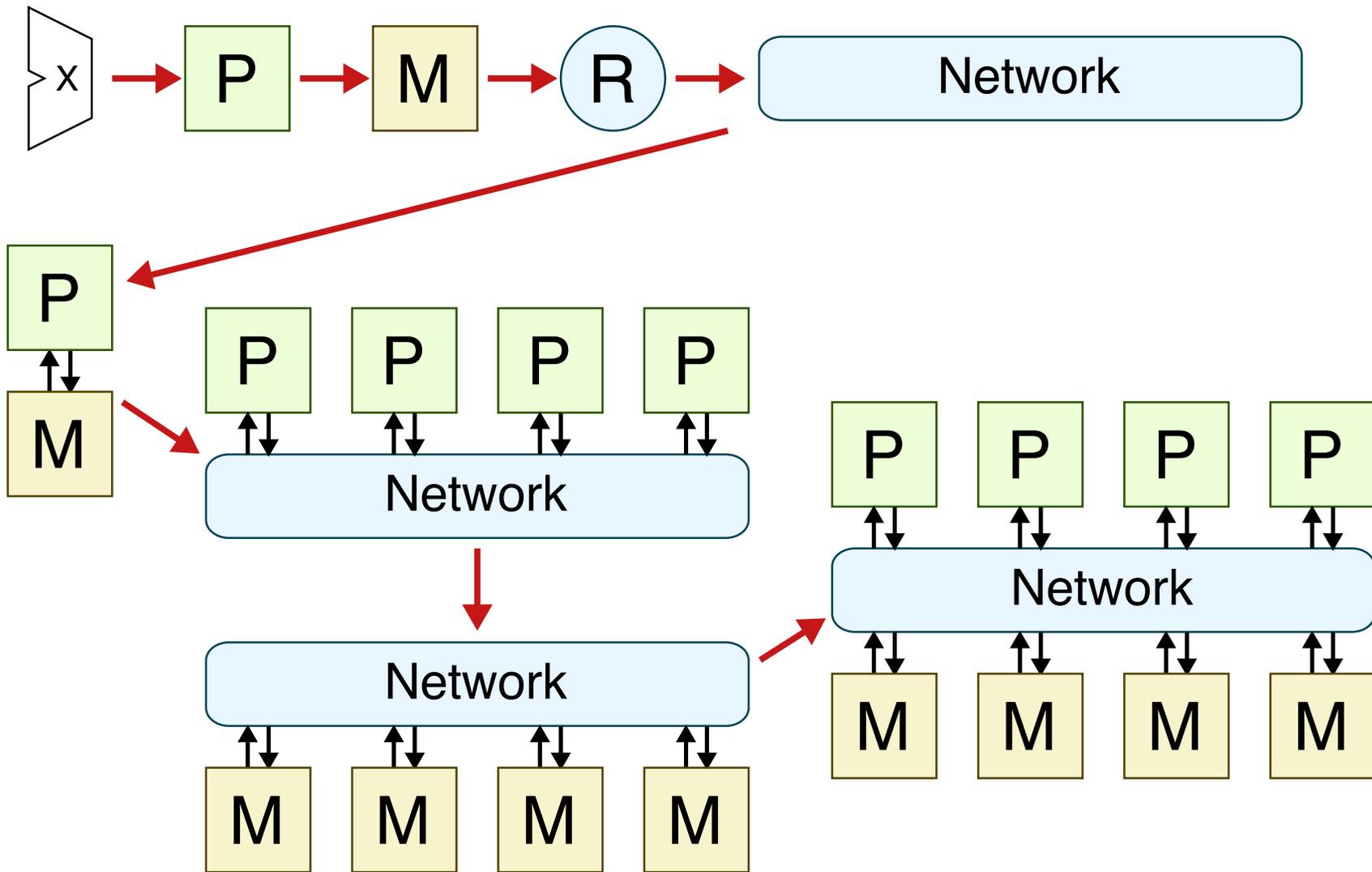


# Agile Development Methodology

- ▶ 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

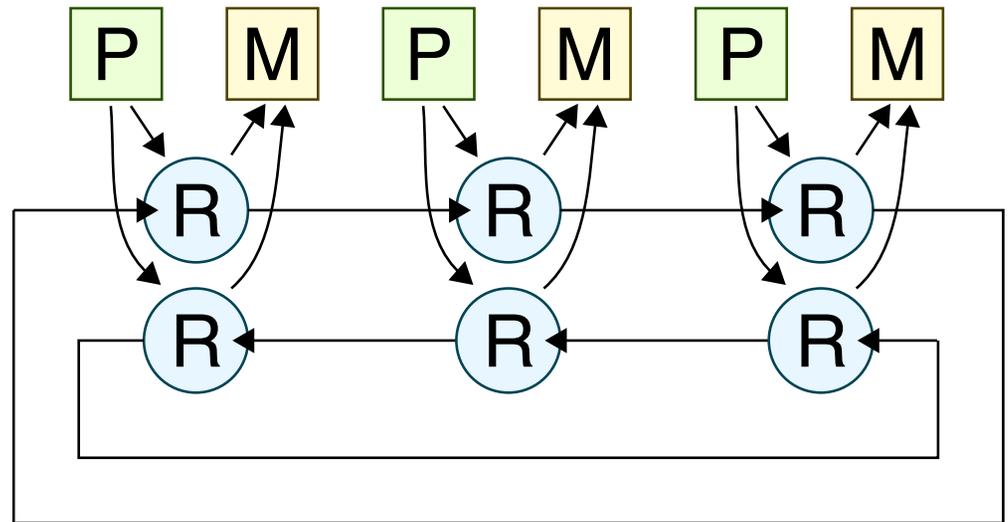
- ▷ Unit tests vs. integration tests
- ▷ Directed vs. random tests
- ▷ Whitebox vs. blackbox tests

## ▶ 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)

```

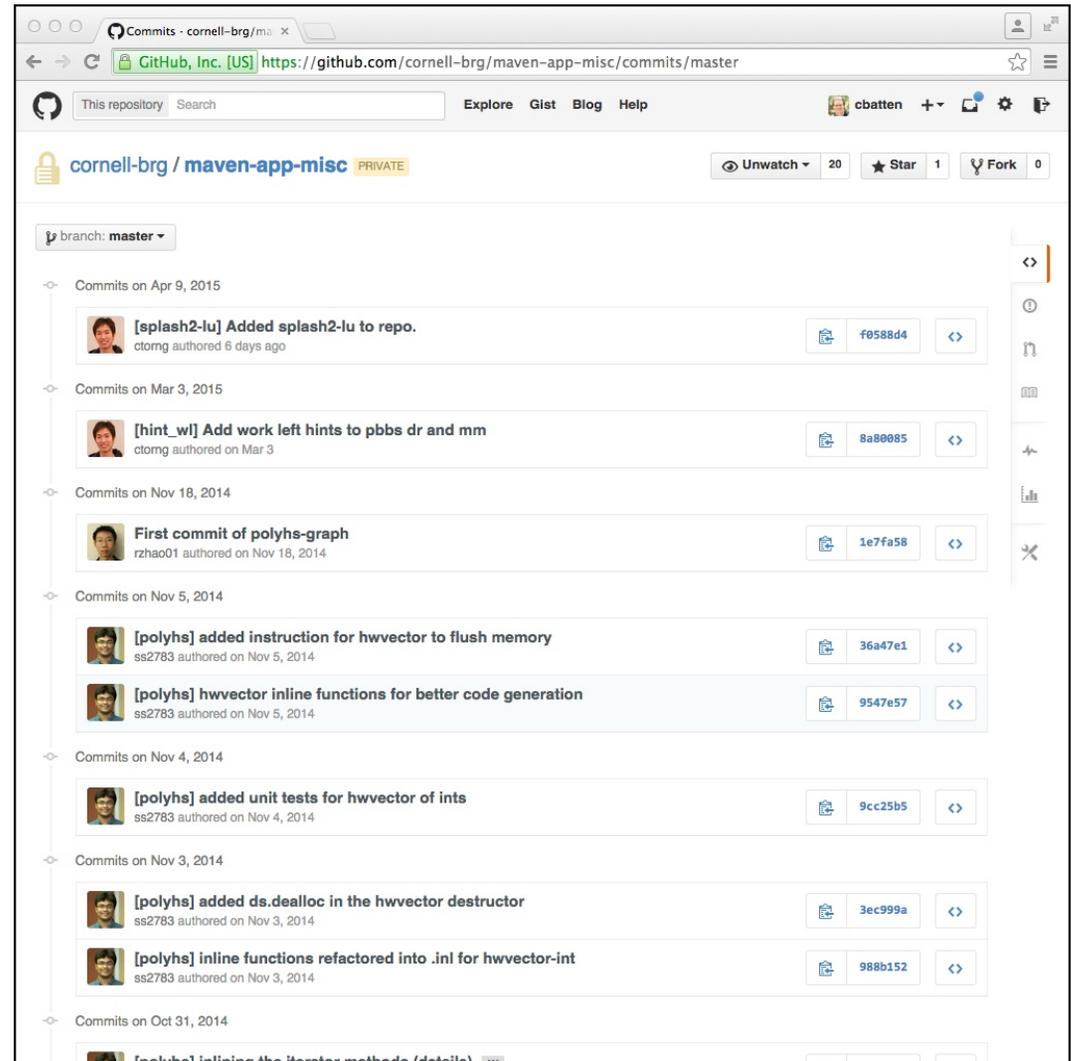
cchatten-mac % py.test ..
===== test session starts =====
platform darwin -- Python 2.7.5 -- py-1.4.26 -- pytest-2.6.4
plugins: xdist
collected 1120 items / 4 errors

../examples/gcd/GcdUnitCL_test.py .....
../examples/gcd/GcdUnitFL_test.py .....
../examples/gcd/GcdUnitMsg_test.py ...
../examples/gcd/GcdUnitRTL_test.py FFFFF
../examples/gcd/gcd_sim_test.py .....FF.
../examples/regincr/RegIncr2stage_test.py ....
../examples/regincr/RegIncrNstage_test.py .....
../examples/regincr/RegIncr_extra_test.py ....
../examples/regincr/RegIncr_test.py .
../examples/sort/MinMaxUnit_test.py ..
../examples/sort/SortUnitCL_test.py .....
../examples/sort/SortUnitFL_test.py ....
../examples/sort/SortUnitFlatRTL_test.py .....
../examples/sort/SortUnitFlatRTL_v_test.py .
../examples/sort/SortUnitStructRTL_test.py .....
../examples/sort/sort_sim_test.py .....
../lab1-imul/IntMulFL_test.py .....
../lab1-imul/IntMulFixedLatCL_test.py .....
../lab1-imul/IntMulFixedLatRTL_test.py .....
../lab1-imul/IntMulMsg_test.py ...
../lab1-imul/IntMulNstageCL_test.py .....
../lab1-imul/IntMulNstageRTL_test.py .....
../lab1-imul/IntMulNstageStepRTL_test.py ..
../lab1-imul/IntMulScycleRTL_test.py .....
../lab1-imul/IntMulVarLatCL_test.py .....
../lab1-imul/IntMulVarLatCalcShamRTL_test.py ...
../lab1-imul/IntMulVarLatRTL_test.py .....
../lab1-imul/imul_sim_test.py .....
../lab2-sort/SortXcelCL_test.py .....
../lab2-sort/SortXcelFL_test.py .....
../lab2-sort/SortXcelRTL_test.py .....

```

# 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)**

The screenshot displays the Travis CI interface for a repository named 'cornell-ece5745/ece5745-labs'. The current build, #26.1, is in a 'passing' state. The build log shows the following steps:

```

1 Using worker: worker-linux-docker-383e1cef-prod.travis-ci.com:travis-linux-11
2
3 Build system information
4
5
6
7 Installing an SSH key from: default repository key
8 Key fingerprint: da:fa:95:0b:e2:48:ad:04:2c:13:86:21:cd:13:21:11
9
10 $ git clone --depth=50 --branch=master git@github.com:cornell-
11
12 This job is running on container-based infrastructure, which does not allow use of 'sudo', 'setuid
13 and setgid executables.
14 If you require sudo, add 'sudo: required' to your .travis.yml
15 See http://docs.travis-ci.com/user/workers/container-based-infrastructure/ for details.
16
17 Setting environment variables from .travis.yml
18 $ export TEST_OPTS=""
19
20 $ source ~/virtualenv/python2.7/bin/activate
21 $ python --version
22 Python 2.7.9
23 $ pip --version
24 pip 6.0.7 from /home/travis/virtualenv/python2.7.9/lib/python2.7/site-packages (python 2.7)
25
26 $ wget --quiet http://brg.csl.cornell.edu/artifacts/verilator-
27 $ tar xfz verilator-travis.tar.gz
28 $ export PATH=${VERILATOR_ROOT}/bin:$PATH
29 $ export PYMTL_VERILATOR_INCLUDE_DIR=${VERILATOR_ROOT}/include
30 $ verilator --version
31
32 $ pip -q install git+https://github.com/cornell-brg/pymtl.git
33
34 $ mkdir -p pymtl/build
35 $ cd pymtl/build
36 $ py.test . --tb=no -x -v $TEST_OPTS
37
38 ===== test session starts =====
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000

```

Application

Algorithm

PL

OS

ISA

 $\mu$ Arch

RTL

Gates

Circuits

Devices

Technology

## 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

