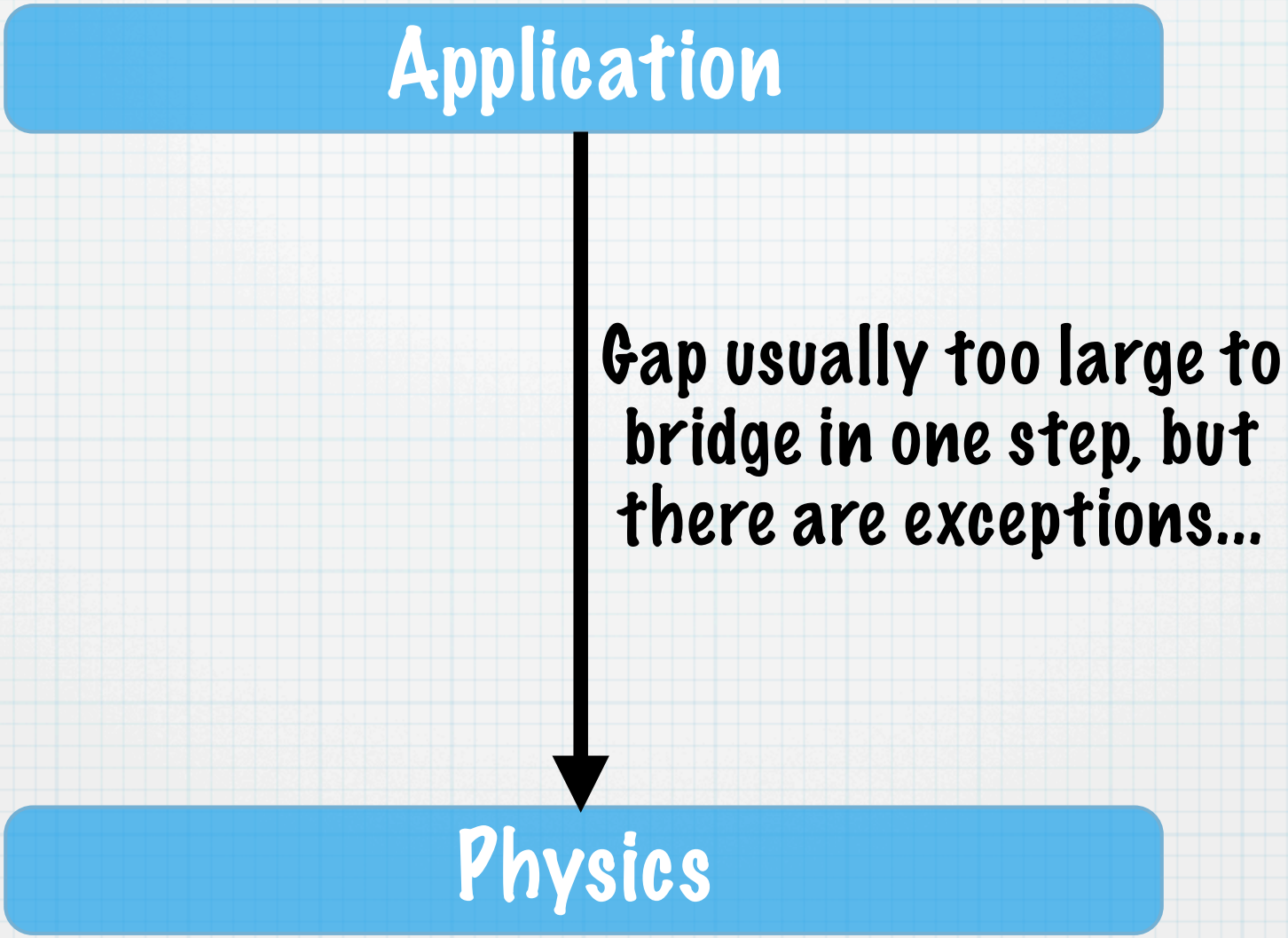
CS250 VLSI Systems Design

Lecture 7: Project Details

John Wawrzynek, Krste Asanovic
with
John Lazzaro
and
Brian Zimmer

Engineering Challenge



Magnetic Compass

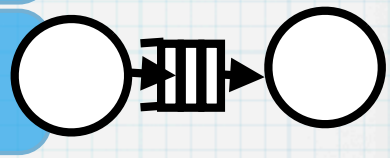
Application

Physics

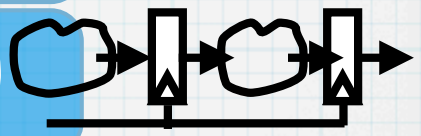
Design Abstraction Stack

Application

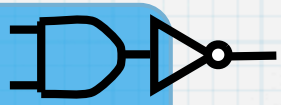
Unit-Transaction Level (UTL)



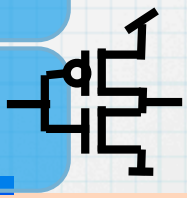
Register-Transfer Level (RTL)



Gates

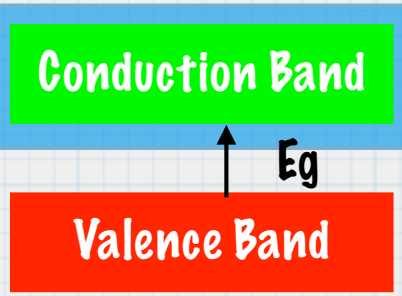


Circuits



Devices (Transistors)

Physics

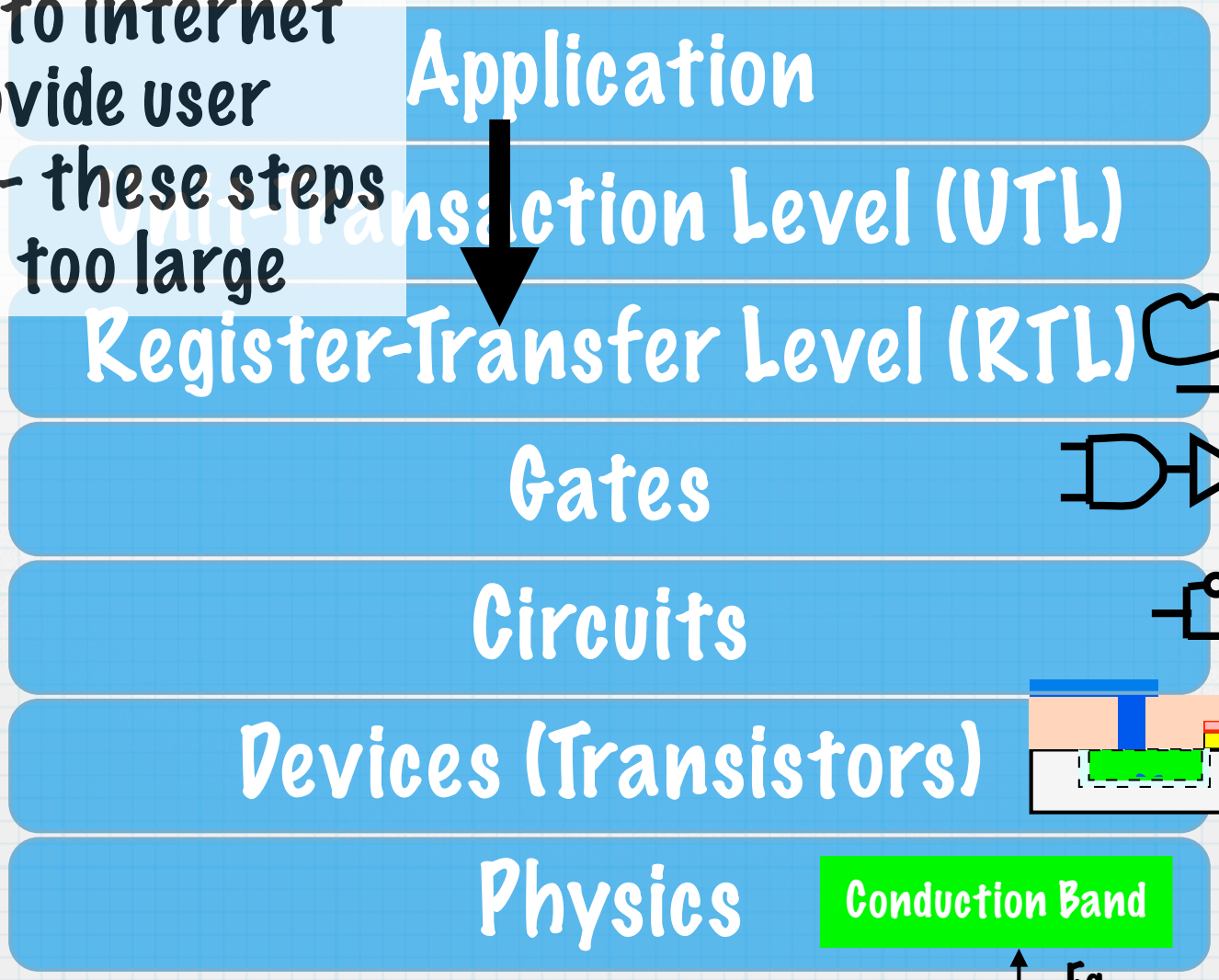


Properties of a Useful Abstraction

- Hides less important details
  - e.g., for RTL, don't worry how combinational logic is decomposed into logic gates
- Allows control of more important details
  - e.g., RTL designer still controls how much logic is performed between any two registers
- If done right, provides portable efficiency
  - i.e., same RTL can be implemented as custom logic, standard cells, FPGA, or even vacuum-tube logic, with reasonably good results

Design Abstraction Stack

For complex applications that talk to internet and provide user interfaces - these steps are way too large



Software-Centric MP-SoCs

- Almost all devices based on large ASICs need to run sophisticated software
- MP-SoC (Multiprocessor System-on-a-Chip) already standard in many devices
  - smartphones, music players, set-top boxes, games consoles, digital cameras, internet routers, cars, ...
- Typical ASIC team ratios:
  - 1 Hardware designer per
  - 2 verification engineers per
  - 6 software engineers

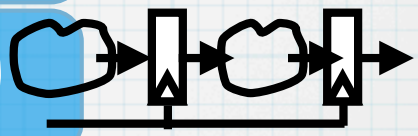
Manycore, a new abstraction layer?

Parallel Application Program

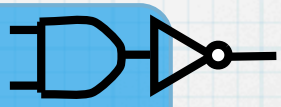
Operating System

Manycore System (UTL)

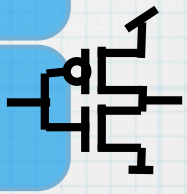
Register-Transfer Level (RTL)



Gates



Circuits



Devices (Transistors)

Physics

Manycore Abstraction

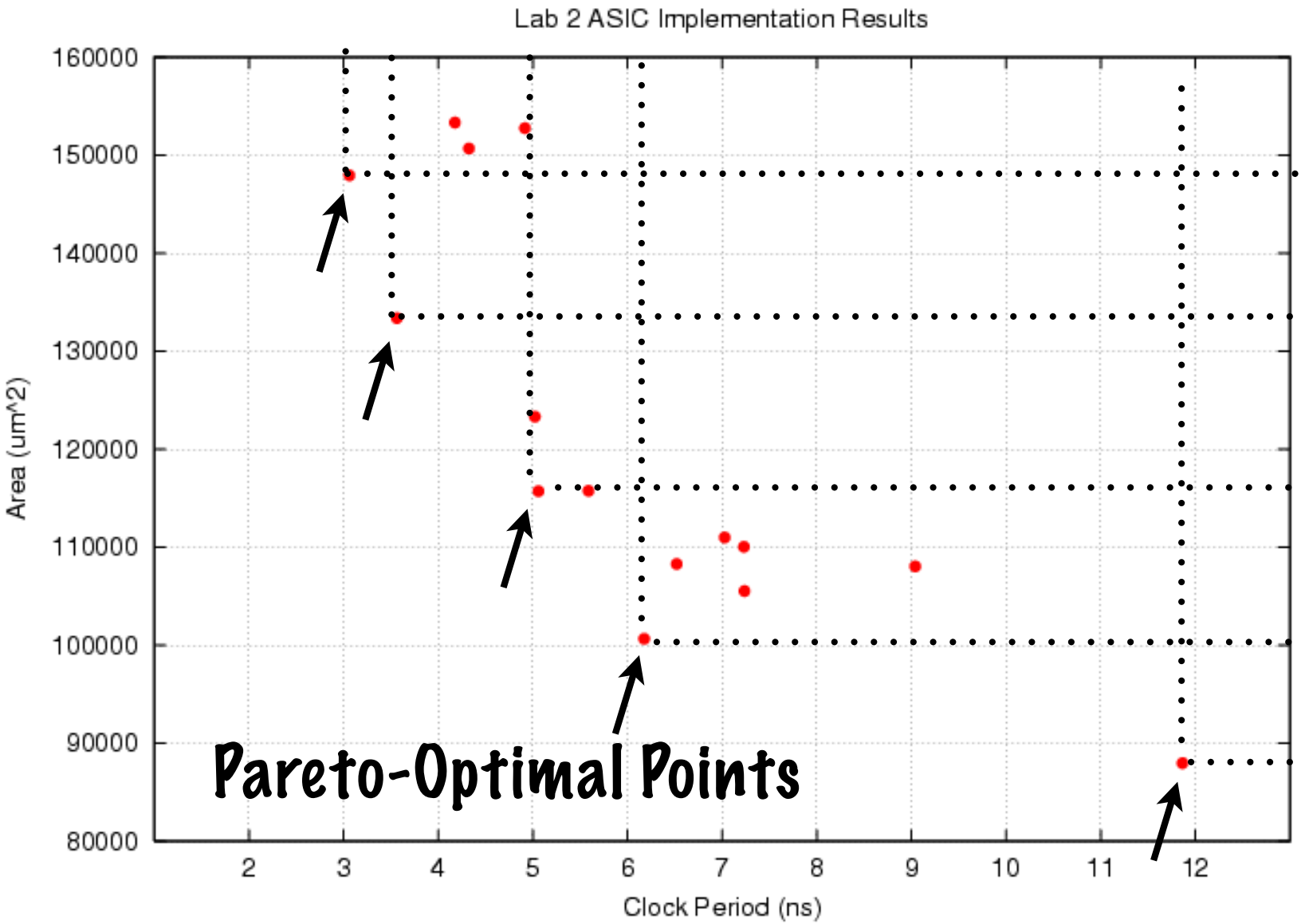
- Hides less important details
  - e.g., programmer doesn't worry how code is compiled and executed on each processor pipeline
- Allows control of more important details
  - e.g., parallel programmer controls how application code and data is distributed among cores
- If done right, provides portable efficiency
  - i.e., same parallel program can be executed on different multiprocessor platforms (general-purpose x86 platform, MP-SoC, FPGA soft cores)

Project Topics

- This year's theme: If "the processor is the new transistor"*, what does the standard-processor library look like?
  - Assume any MP-SoC will be built from a heterogeneous mix of processor types
- Your task is to explore variations of RISC-V microprocessors in class projects
  - i.e., pick a general class of RISC-V processor, then explore design space for that class of processors
- Output of class could be used to begin populating a "standard-processor" library

[*Rowen, Tensilica]

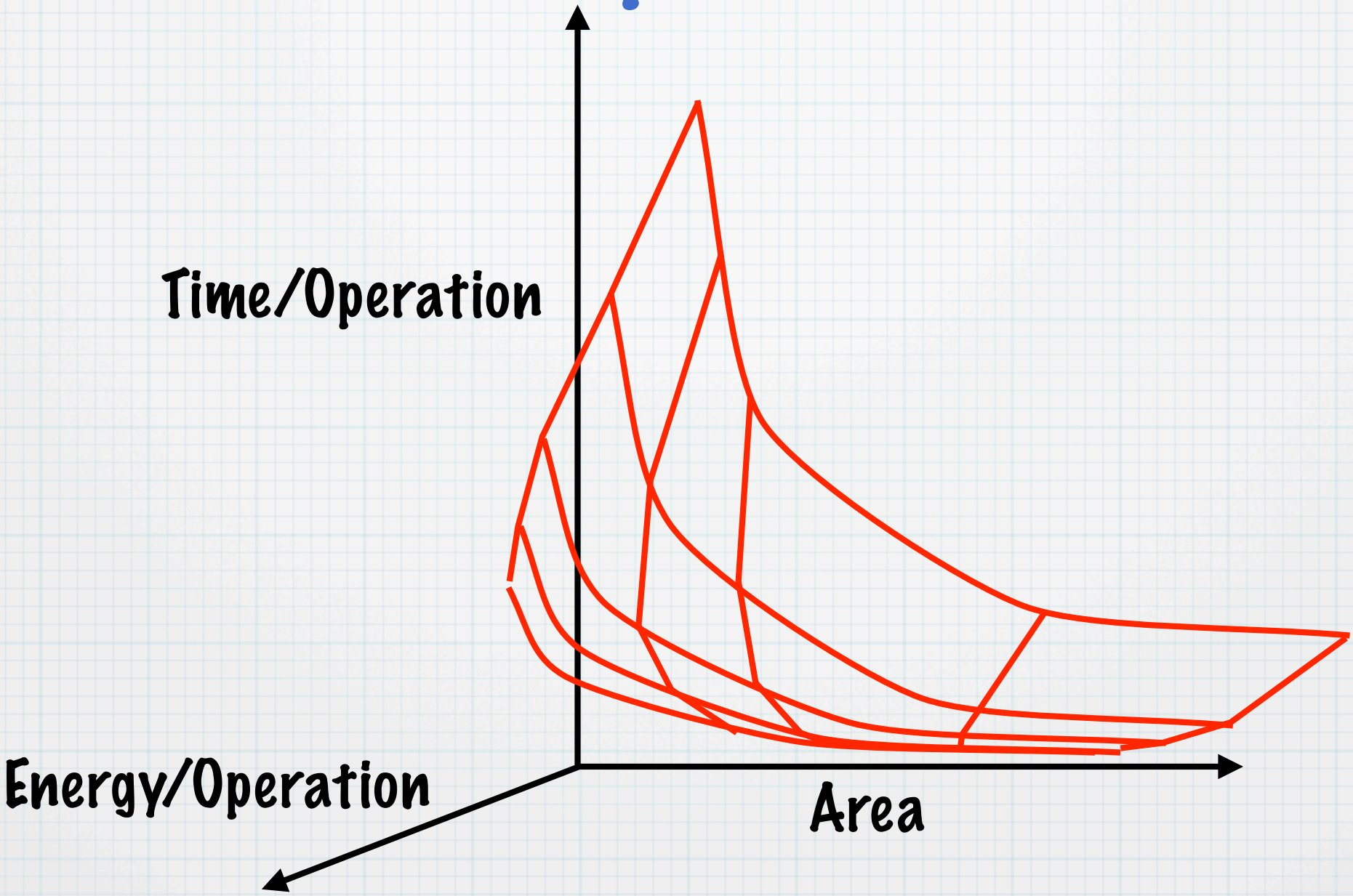
Results from MIT Lab Exercise (6.884 2005)



Quality-of-Results (QoR)

- QoR is term used to describe overall "goodness" of design. Usually a multi-dimensional vector including multiple metrics:
  - Area (mm^2)
  - Performance (Operations/Second)
  - Energy efficiency (Operations/Joule)
  - System-Level Cost
  - Correctness (not binary in practice)
  - Reliability (Undetected errors/Year, MTBF)
  - Manufacturability (%Yield)
- We will focus on first three in project
  - Require projects to be 100% correct...

Pareto-Optimal in 3D



Our Project Expectations

- B grade
  - Single working design
- A grade
  - Thorough design-space exploration

General Project Info

- A large-scale VLSI design experience
- Most significant component of class
  - 70% of grade!
- Work in teams of 2
  - In exceptional circumstances, might allow other team sizes (1 or 3), but have to clear with us ASAP

Initial Proposal

- Due October 10 before class
  - email to instructors+TA as PDF file
  - Not any other file format, must be PDF
- Must include:
  - Title
  - Team members' names
  - 2-page description of what you want to do.
  - What does the design space look like and how you will explore it
  - Must be in PDF (in case you weren't paying attention)

Project Meetings

- Public presentation days:
  - Each group gives 15 minute presentation to whole class
  - Everyone must attend, give feedback
- Private project meetings:
  - Approx. 20 minutes scheduled with each group alone with staff to give private guidance
  - Class splits into two sets of project groups, one meets on Mondays, other on Wednesdays
- Plus, come to office hours, arrange other meetings

Public Presentations

- Oct 12: Present project proposal (10 mins)
  - Describe your project idea, get feedback
- Oct 31/Nov 2: Present initial results on at least one design point
  - 15 minute presentation max. to whole class (~10 slides)
  - Prepare concise, informative presentation.
  - Ask class for feedback, 5 minutes
- Nov 14/16: Present some design-space exploration results
  - Same 15 minute + 5 minute discussion format
- Dec 5/7: Final project presentations
  - 20 minutes each, time limit strictly enforced (practice!)

Private Meetings

- Each group has a scheduled 20 minute meeting with staff to provide guidance/feedback
- Same time slot during class time every meeting week, and also used for final presentation
- Schedule picked randomly
  - Cannot change unless you can convince another group to switch
- Come prepared with results/questions

Final Project Report

- Report should read like a conference paper
  - <=12 pages
  - >= 10pt font
  - one or two-column
  - must be PDF
- Report due 6AM Monday Dec 12
  - No extensions!
  - Email PDF file to staff

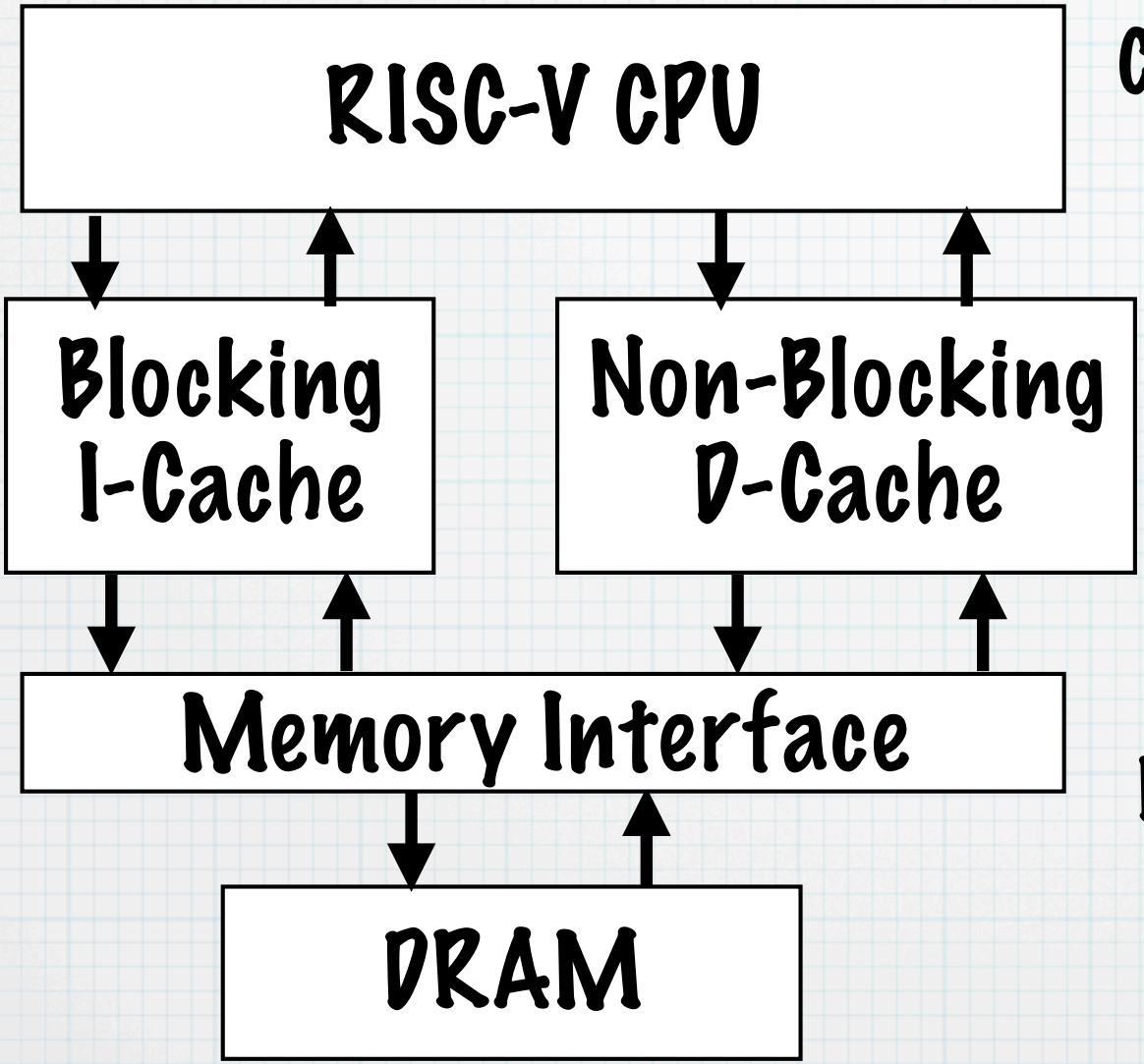
Project Source Code

- Must be kept in repository so we can build, test, and inspect design
- Must be checked in before each meeting
- You'll want to use repo for group communication/archiving

Possible Project Ideas

- General-Purpose processor improvements
- Domain-specific accelerators

Standard Processor Configuration



CPU to be completed to layout level

Caches can be modeled or completed to layout level depending on project

Memory system to be modeled functionally

Project Infrastructure

- We will provide a RISC-V processor core (Rocket) + test harness + test code
- Later lecture will cover Rocket microarchitecture in detail
- Some projects may choose to start from a simpler RISC-V core

Deeply Pipelined Processor

- Investigate effect on metrics as pipeline depth increased
  - e.g., 5-stage, 7-stage, 8-stage, 9-stage, 11-stage pipeline
- Should aim to get below 25 FO4, maybe as low as 16 FO4 per clock cycle
- Will need some microarchitectural tricks to cope with increased pipeline hazards
  - Time/program = insts/program * cycles/inst * seconds/cycle

Multithreaded Processor

- Investigate adding multithreading to processor
  - e.g., 1,2,3, 4 or more threads on each pipeline
- Will need multithreaded benchmarks, but can also show results for multiprogramming (running a mix of benchmarks)
- Can remove pipeline logic, e.g., bypassing, to show effect on design metrics
  - Trade single-thread performance for better multi-threaded efficiency

In-Order Superscalar

- Investigate moving to dual, triple or quad instruction issue, with different combinations of instructions allowed to issue together
- To attain reasonable performance, will need to improve other parts of pipeline
  - e.g., better fetch path

Some Combinations

- Deep Pipelining + Multithreading
  - Pipelining introduces hazards, but threading can avoid them
- Multiple-issue + Multithreading
  - Issue multiple instructions but from different threads to reduce superscalar issue logic complexity
- Deep Pipelining + Multiple-issue
  - Maximum single-thread performance

Branch Prediction Strategies

- Investigate performance/area/energy tradeoffs for different branch predictors
  - BTB vs. BHT vs. hybrids
  - Tournament predictors
  - Return address stacks
  - Perceptron predictors?

Simple Out-of-Order Superscalar

- Full blown out-of-order design too complex for class
- Try some simple OoO enhancements
  - Single-issue OoO
  - Tomasulo-style OoO

RISC-V 16-bit Compressed ISA

- RISC-V incorporates a 16-bit compressed instruction format mixed with 32-bit format
  - Every 16-bit instruction expands to an equivalent 32-bit RISC-V instruction
  - Compiler/assembler available
- Project would investigate different strategies to implement decompression, evaluate performance/energy, and possibly suggest new opcodes (e.g., load multiple/store multiple)

Crypto Accelerator

- Investigate ways to accelerate some classes of cryptographic operation
- E.g., SHA, AES signature schemes, or RSA key operations

FFT Accelerator

- Explore design space for FFT accelerators
- Instruction-set extensions or autonomous coprocessor

Stencil Accelerator

- Design coprocessor to optimize stencil operations (i.e., structured grid codes)
  - Motion estimation
  - Machine vision kernels (SIFT?)