SELF-OPTIMIZING ARCHITECTURES: TOWARD THE TRUE ONE-SIZE-FITS-ALL CPU

Prof. José F. Martínez

http://csl.cornell.edu/~martinez

Computer Systems Laboratory – Cornell University

We're Human after All

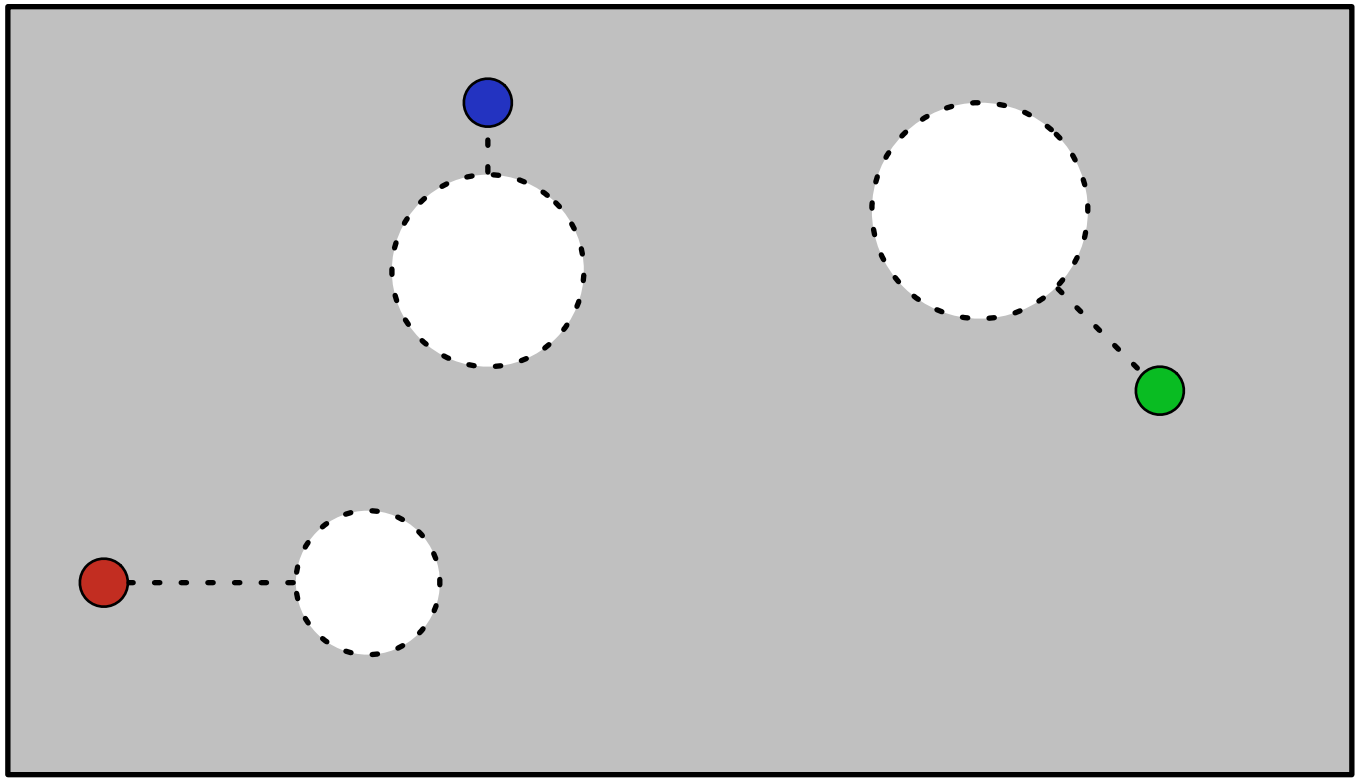
- Moore's Law: transistor density grows exponentially over time — in the billions now for server CPUs

- Architect's approach: Mostly expert intuition
  - "Intuition growth" highly unlikely exponential
    - "Let's just throw more engineers at the problem"
  - Simplistic, rigid, underperforming average-case solutions
  - Multicore by itself hardly the solution
    - "Why throw more engineers at the problem? Let's just throw in more cores!"

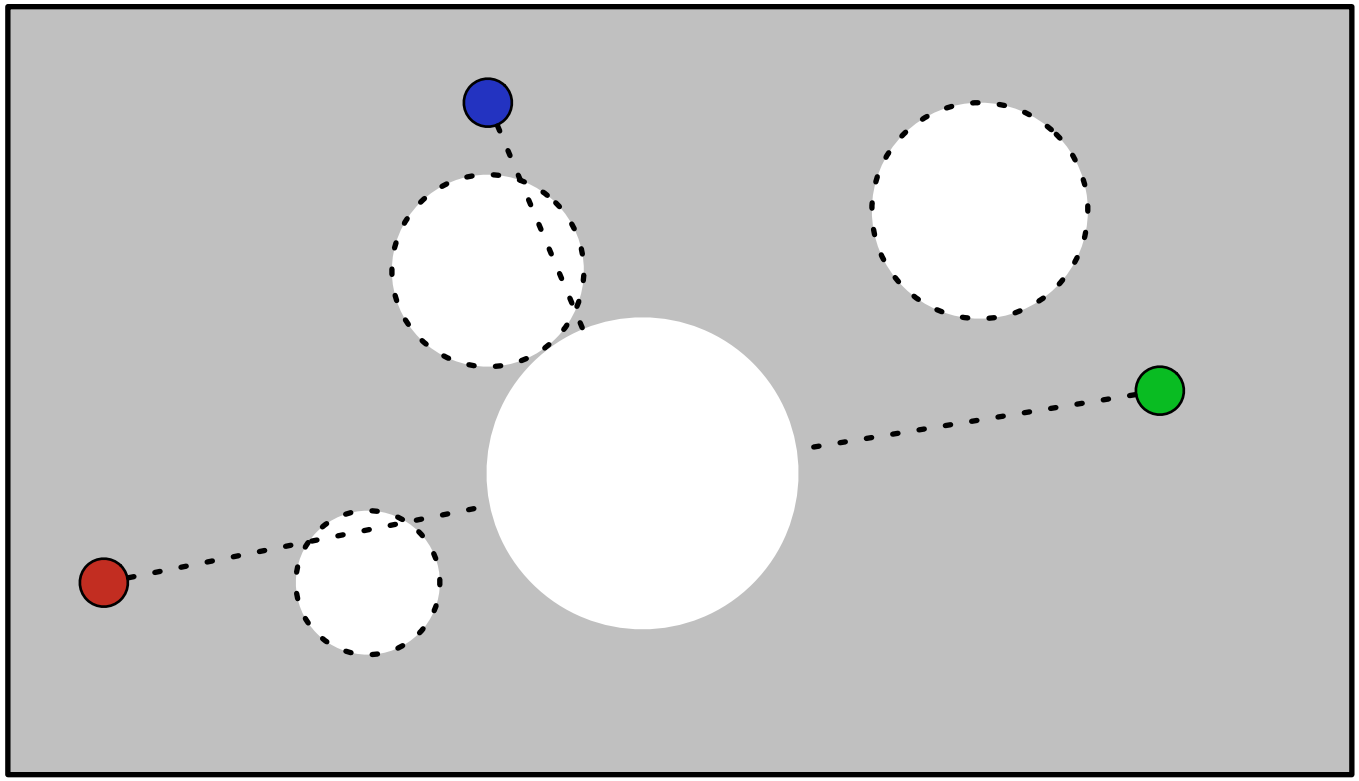
Meanwhile in the Real World…

- Data centers backbone of cloud computing
  - Must service wide variety of workloads
  - Continuously changing demands
- Cost is king
  - One-size-fits-all CPU?
    - Works for Intel!
- These seem contradictory
  - They are if using traditional designs

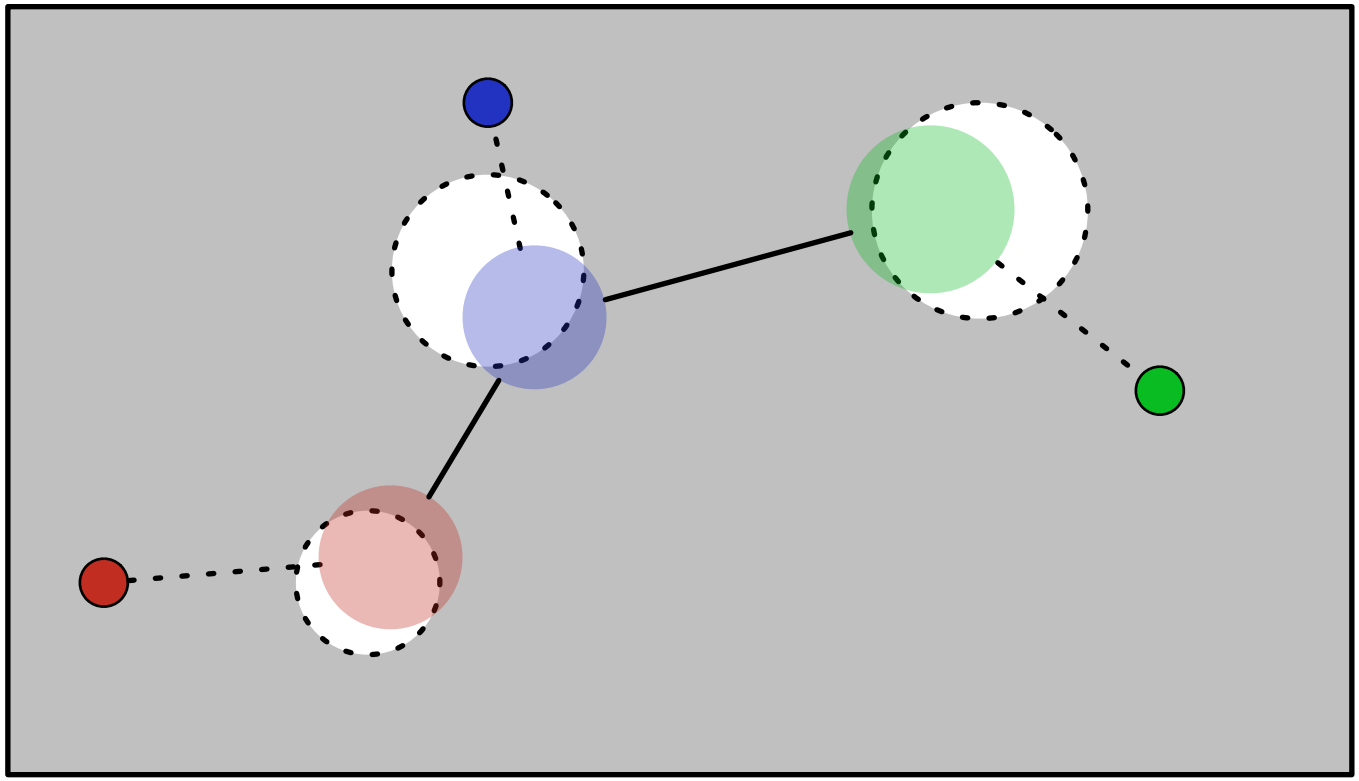
Application-Architecture Space



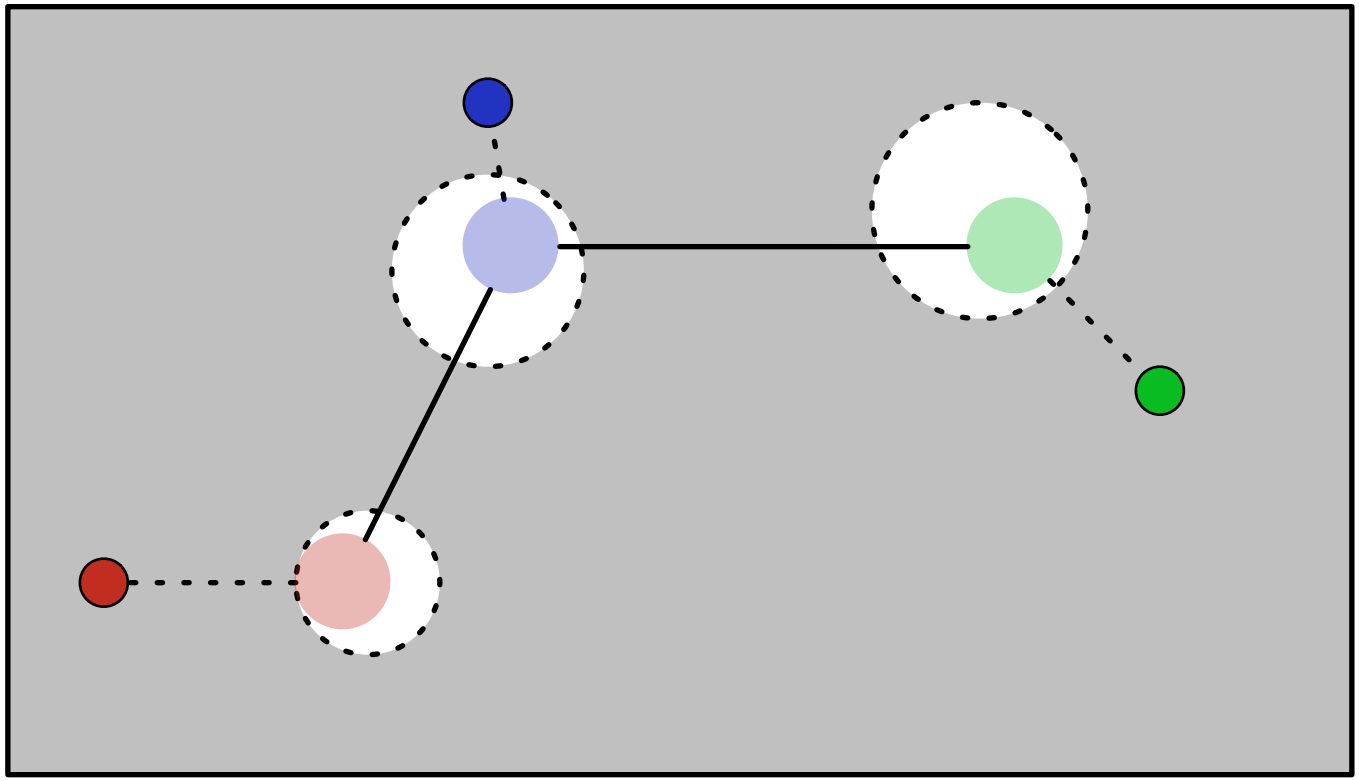
Traditional General-Purpose



Reconfiguration



Self-optimization



Self-optimizing Architectures

- **Let silicon itself handle the complexity**
  - Remember: Transistors *are* plentiful
  - Designer focuses on "what"
  - Hardware focuses on "how"

Much better use of human ingenuity!

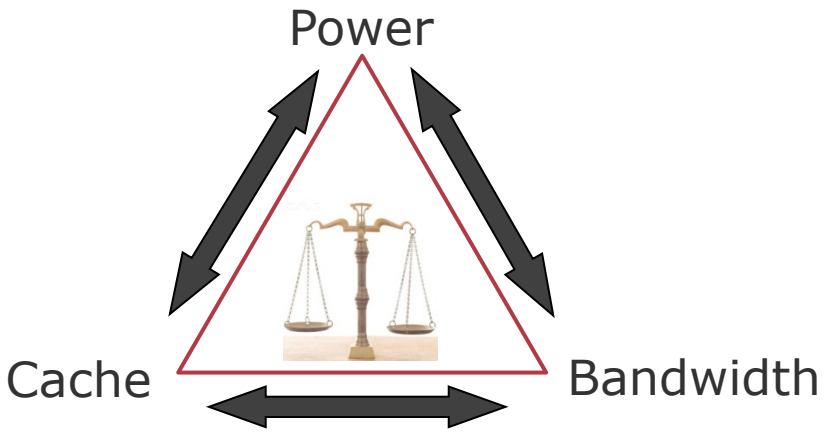
- **Reconfigurable architectures**
  - Optimize "baseline"
- **"Gray-box" optimizers** based on machine learning
  - CPUs are the "big data" of silicon
    - 2.6 (i+d)/cycle × 64b × 1.2 GHz ≈ 200 Gbps

Three Quick Examples

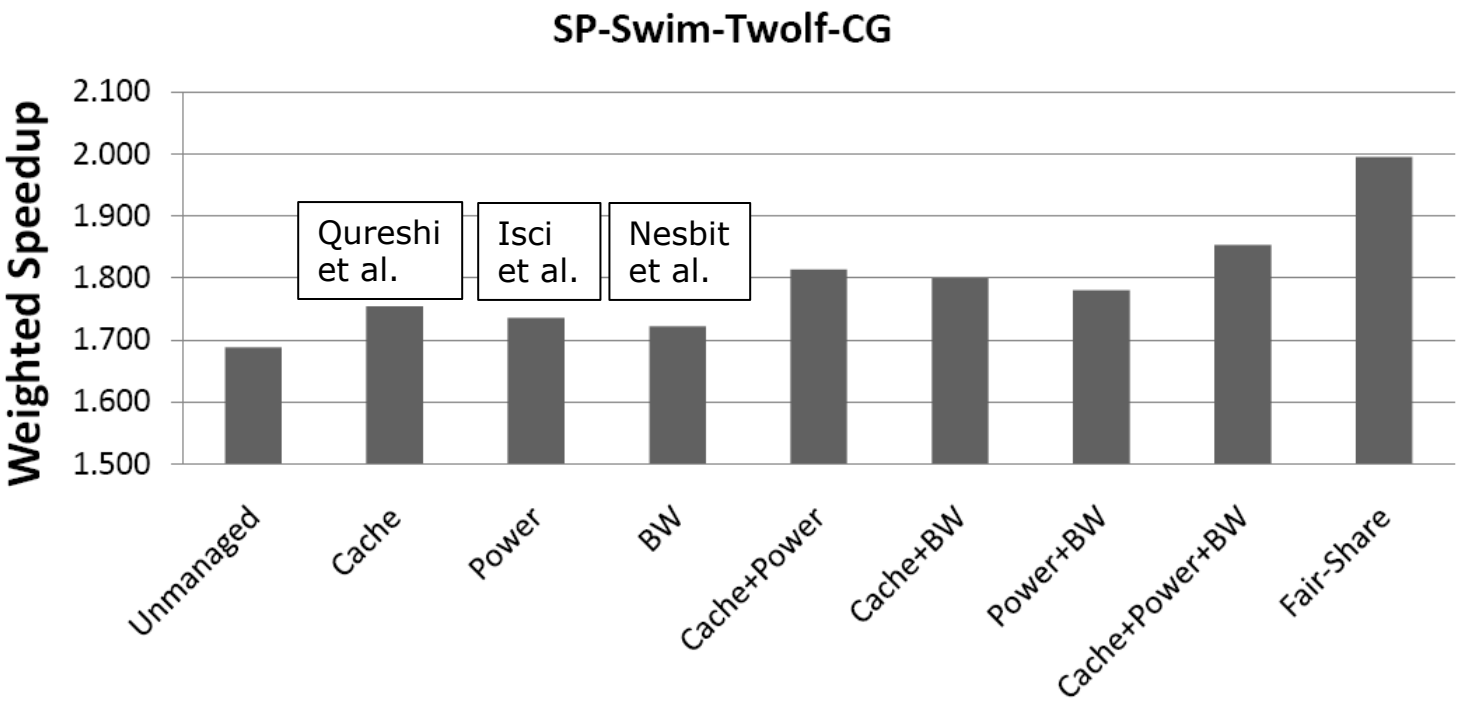
- **Core Fusion:** On-the-fly multicore configuration
  - Coarse- vs. fine-grain problem goes away [ISCA'07]
- "Smart" **multi-objective memory scheduling:** On-the-fly optimization of memory operations
  - Employs *genetic programming* and *reinforcement learning* [ISCA'08,HPCA'12]
- "Smart" **multi-resource allocation:** On-the-fly optimization of shared multicore resource utilization
  - Employs *supervised learning*

Multi-Resource Allocation in Multicore

- Unrestricted resource sharing can lead to destructive interference (monopoly)
- Several proposals that manage power, cache, or off-chip bandwidth exist but lack complete control over them
- **Need to take resource interactions into account** for optimal performance

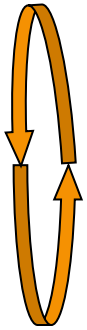


Ignoring Interactions: Bad Idea



Using Hardware Predictive Models

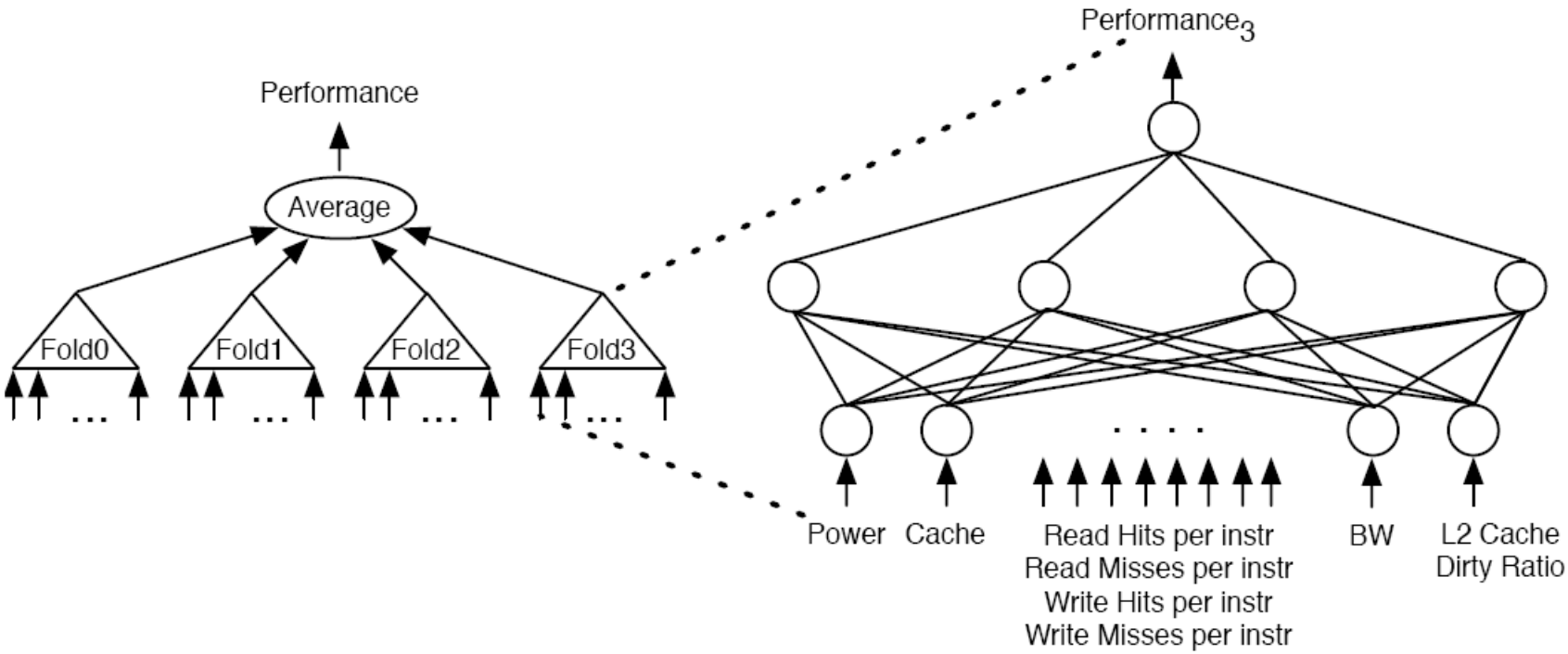
- Optimizing configuration online via trial-and-error can be costly
  - Huge search space; learning curve affects performance
- Proposal: Use "sideline" approach
  - Monitor applications (sparse sampling)
  - Learn using on-silicon predictive models (ANNs)
  - Search for optimum allocation via "on the side" querying
    - Less invasive, much faster
  - Install resulting allocation periodically



- **Architect can focus on more productive challenges**

Challenge #1: Find Algorithm

- Artificial neural network (ANN) ensemble
  - Feed appropriate system attributes to model apps
  - Performance prediction will guide allocation search



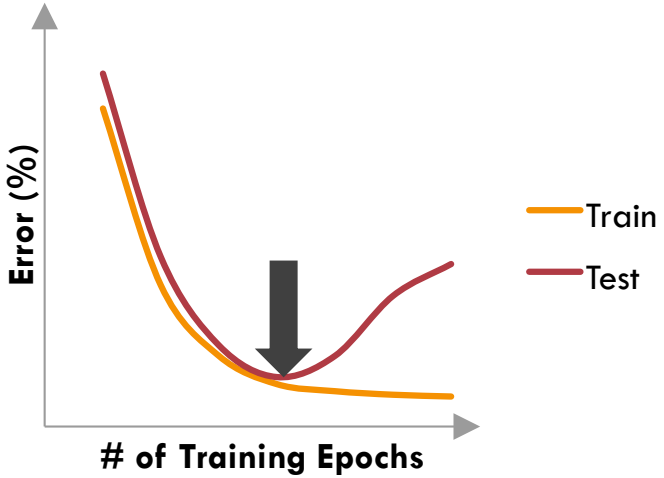
Challenge #2: Prediction Error

- ANNs not always able to predict with sufficient accuracy
- Identify and discard these cases for better performance
- **Simple mechanism based on variance across four ANNs**
  - Intuition: Large variance → Large error

Challenge #3: Building the Model

- Problem 1: Overfitting
  - Solution: Set aside some data to detect overfitting (**early stopping**)

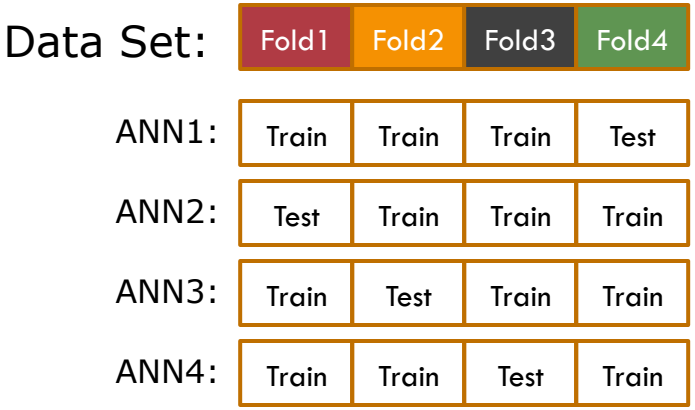
Data Set: 



Challenge #3: Building the Model

- Problem 1: Overfitting
  - Solution: Set aside some data to detect overfitting (**early stopping**)

- Problem 2: Some training data wasted
  - Solution: **cross validation over the four ANNs**



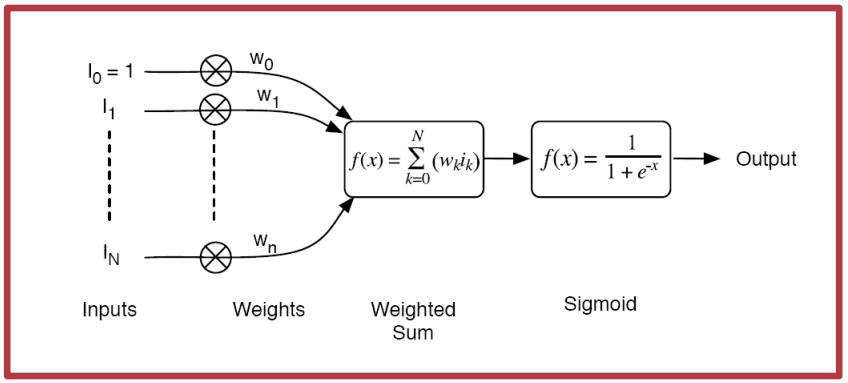
Challenge #4: Search Algorithm

- **Stochastic hill-climbing**: 2,000 steps at start of each interval
  - Estimate performance for all neighbors of current optimum
    - Too many neighbors to try → Move to a better neighbor immediately
  - Try random points with 20% chance
    - Move there if it is better than the best allocation found so far
  - Detect if stuck in local minima
    - Jump to a random point and resume HC

Challenge #5: Implementation

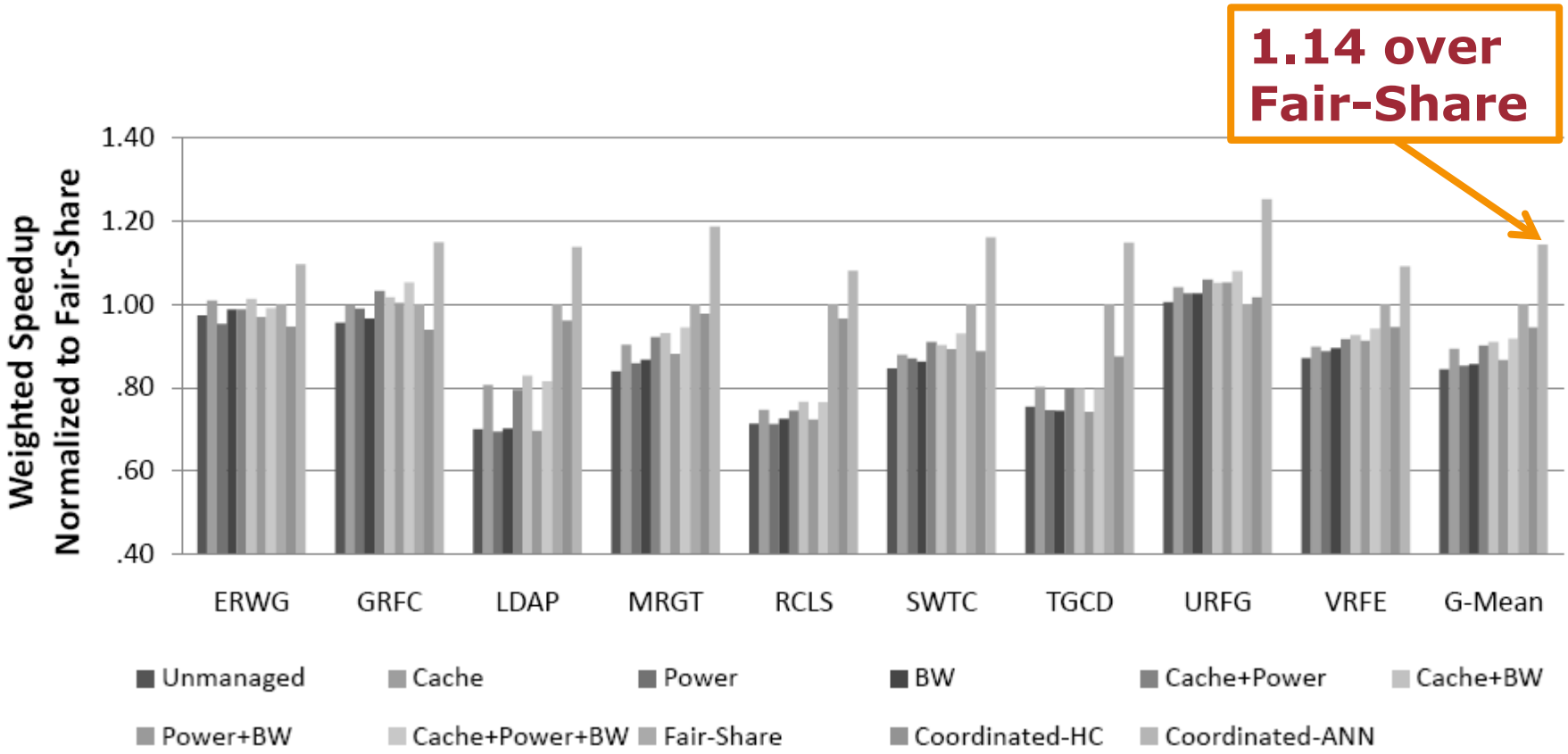
- Impractical to build all 16 ANNs in HW
  - Have 4 applications in our setup
  - Each application needs 4 ANNs

- **Build single ANN and time-share it**
  - Narrow-width, pipelined multipliers
  - Lookup tables for sigmoid function approximation



- Area overhead: ~6mm²
- Power overhead: ~3W

Performance Results: 4 Cores/Apps



Bitirgen et al., MICRO '08

Self-optimization: Takeaway Slide

- General-purpose doesn't mean average
  - Nothing more general-purpose than humans!
  - Key: ability to specialize over time

- Transistors are there; put them to work *smart*
  - Give transistors a "higher education"
    - Then watch them do great things on their own
  - Leverage architect's intuition where it matters
    - I.e., don't re-invent the ANN !