1. Vision
Just like parallelism, we want to elevate computational significance as a first class concern in the design of algorithms and systems. Exploit computational significance to build energy efficient hardware and software platforms that scale gracefully in case of errors induced by scaled voltages and/or parametric variations.

2. Motivation
- Power/Energy dissipation remains #1 constraint of future systems.
- Several application domains offer the opportunity to trade-off quality of service for significant improvements in energy consumption (e.g. JPEG).

3. Objectives
- Introduce computational significance as an algorithmic property and expose it to the level of the programmer.
- Deive techniques that facilitate automatic characterization of code and data significance using compile-time or runtime analysis.
- Lay the necessary foundations at all levels of system stack allowing controlled quality execution on unreliable hardware substrates.
- Enable drastic power dissipation reduction by opportunistically and aggressively powering parts of the platform below nominal values in a targeted way.
- Produce a vertically integrated system prototype to prove the efficacy of significance-based computing.

4. Semi-Automatic significance analysis
- Introduce a rigorous mathematical definition of computational significance.
- Apply a set of criteria to (semi)automatically characterize significance.
- Partition source code to slices of varying significance.
- Example of interval arithmetic and algorithmic differentiation:
  - Original code:
    \[
    y = v_1 + x_2; \\
    v_1 = 0.3465; \\
    x_1 = \log(x_1); \\
    v_2 = v_1 + x_2; \\
    y = v_2;
    \]
  - Input range for \( x_1 \): [1, 2]
  - Input range for \( x_2 \): [1, 20]
  - Significance criterion: \( \epsilon \|
  - Significance bound: \( \epsilon = 1 \)

5. Task-based programming model and runtime system
Program Model:
- Code is tagged with significance information at task or even subtask level.
- Significance determines the (un)reliability of the core used for task execution.
- Offer recovery mechanisms coupled with sanity checking functions.

Runtime System:
- Component-level control of power, performance and reliability
- Significance-aware scheduling and memory management
- Dynamic optimization under reliability constraints
- Significance-aware runtime system monitors the dynamic behavior of the cores:
  - Did the core crash due to a fault?
  - If not, how many faults could the hardware detect?
  - Based on the information conveyed by the hardware, the runtime system calls the sanity function and takes corrective actions.

6. Hardware modeling and design
- Develop instruction level power and behavior models under various degrees of voltage scaling and variations.
- Exploit the dynamic timing profile and modify the circuits in conjunction with the core microarchitecture to enable graceful performance degradation.
- Cores are enhanced with low cost error detection/correction mechanisms to help in adjusting their reliability and energy efficiency.
- Extended ISA to support both approximate and accurate instructions.
- A simulator that supports the overall significance driven vertical stack and allows its evaluation at various operating modes will be developed.