1. Faster simulators
2. Internet in a box
3. Data center in a box
4. Real-time OS/QoS
5. OS scheduling
6. Compilers
7. Chipset/SOC prototype
8. Reliable / fast aging
9. Thermal / power (S/AC)
10. Common tools, research community
11. Education
What about on I do with POWER
that I can't do otherwise?

1)  
- FUNCTIONAL
- PROTOTYPE
- TIMING

- LONG RUNNING SIMULATION
  - e.g. DYNAMIC CACHE ADAPTATION
- HD C CODES

- DESIGN SPACE EXPLORATION
  - INTER & AMD / RUN OWN EXPERIMENT
- POWER
  - USING ML TO FIND
    RELATIONSHIP BTW
    SIGNALS AND POWER

- RELIABILITY ARGUMENT
  - PROTOTYPE & PERFORMANCE

- PARALLEL DEPLOYMENT
- MULTICORE SW PARALLEL
- DEBUG PLATFORM
  - ADJUSTING PARAMETERS
- MONTE'S DEMON
  - ADVISABLE / GIVES RACE
  - REPEATED BY PATTON
What does interoperability mean? Usage scenarios

1) Sim as service
2) Transplant GUI to SWI for debugging/visibility
3) Database incremental bringup

Simulators to connect to:

- C/C++ style...
- Flex?
- SCE-MI? maybe...

Opportunistic simulator bindings
1. Define simulator and prototype/direct emulation/direct configuration
2. Define strengths and weaknesses of each
3. When do you do one and when do you do the other?
4. Unified theory
5. Questions for RAMP

1. Functional emulator has no notion of time. Direct RTL has a notion of time that can (hopefully) be transformed into a notion of time in your final system. Simulation divorces simulated time from real time.

2. Simulator can give you better observability. Also can be designed to allow you to explore a design space more easily.

2. Virtualizing threads makes better use, and possibly achieves better frequency.

2. “We will live and die in time.”

2. Compile the simulator only once then compiling applications is faster.

2. Case for direct implementation
- Validation/prototyping (killer app)
- Starting point to add functionality (Raksha)
- Product or revenue source
- Perhaps you already have
- Might be easier to write than a simulator
- A Tensilica-like product aims to allow you to do the design-space exploration and final implementation in the same product.

3. You just have a spec. When do you go to simulation and when do you skip over and then just start writing RTL?
   - Do you understand all the specs of your system?
   - Are you confident that they are correct and meet all efficiency requirements.
   - But at the end of the day you have to do implementation.
   - Is the implementation (RTL) easier than the simulator (RTL)

3. What is the cost in terms of extra time to write your simulator. Does this time investment give you a sufficient return on investment that justifies writing your simulator. You need to know your application and set of metrics by which you know you have a good solution.

3. Do you have a fundamental mismatch between host and target.
   The larger the mismatch the more appealing simulation becomes.

4. What is the “result” of your project? A computation or a new system? “Are you building something to learn or are you learning something to build.”
4. Simulation is an instance of reconfigurable computing.

4. A unified theory of Simulation/Emulation – It’s all just computation!

5. Is it RAMP’s job to tell you “Your system runs fast.” or is it “I can run your software fast.” Does correct function depend on correct time? IE do software programmers ever care about time? How often is running on the real hardware very different than running on the FPGA-based emulation. How much value does it have to say that a prototype is correct? How can you know that the prototype is correct if you don’t have time. (Example: multi-core ISA/OS).

6. Is every project evaluated on meeting a bound on a consistent set of metrics (power, speed, thermal). What else is there? Are there metrics where I can give you a horrendous notion of time and the metric still had a low margin of error????? (Correctness for certain designs? Cost to manufacture? Circuit area of final system?)

6. Are there other systems in reconfigurable computing which need an explicit representation of time? (Radio Astronomy). It seems like it’s when the output is a function of the input + time. Or time is an output. (Do you meet a real-time constraint?)
By heterogeneous we really meant very different cores. Two example systems were a cellphone SoC and a CPU+GPU combination.

Many issues in simulation this system are independent of RAMP technology (software toolchain, how to communicate/synchronize in the target machine).

One idea was to do high-level modeling of accelerator as a software functional model running on a regular core, with timing model approximating expected performance. Can narrow design space down (e.g. is it worth building accelerator?). Once RTL design is known, can plug in to replace high-level model.

Power modeling needs RTL or abstract model.

For FPGA accelerators, some thought on how to do timing accurate modeling. One idea is to stop clock on portion of FPGA holding mapped configuration. Can split larger target configurations over multiple host FPGAs.