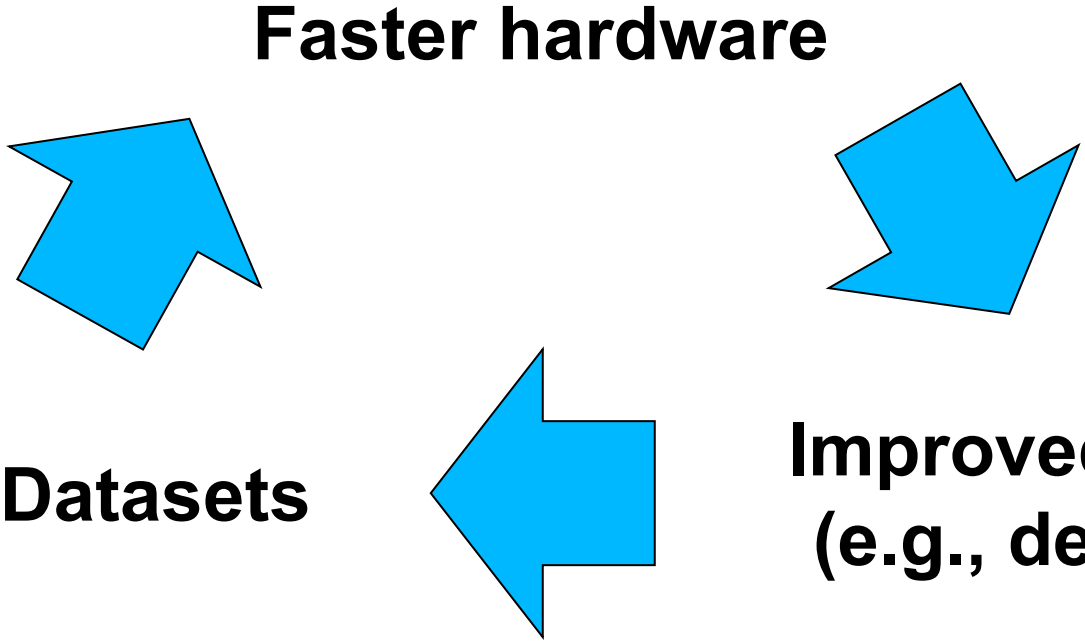
# Implementing Support for Extensible Power Modeling in gem5

Alex Smith and **Matthew D. Sinclair**

University of Wisconsin-Madison

sinclair@cs.wisc.edu

# Why Is This Important?




**Moore's Law has enabled a virtuous cycle of progress in many fields**

**… slowing of Moore's Law also threatens progress**

**Modern apps have ravenous (exponential) compute, power needs**

**Must co-optimize for performance and power**

# What Is Needed?

⇒ **Need disruptive, cross-layer changes to meet future sys. reqs.**

- Co-design arch, runtime, OS, compiler, network, batch sched.
- For both power and performance (and maybe other factors)
- Increasingly important as transistor sizes shrink

- Typically sim. & modeling tools enable early-stage design exploration
  - Recent work: scalably enable accurate co-design for performance
  - But what about power?
- Need credible, open-source modeling infrastructure **for both**

# Power Modeling State-of-the-Art

- 5 broad types of power models:
  1. Extrapolate first-principal models (e.g., CACTI [Wilton JSSC'96], McPAT [Li MICRO'09])
     - Were highly accurate, still widely used … but not updated in 8+ years
  2. Empirical measurement-based models (e.g., AccelWattch [Kandiah MICRO'21])
     - Difficult to generalize beyond specific devices they are measured on
     - Significant accuracy decrease for even minor perturbations
  3. ML-based models (e.g., [Kumar MLCAD'19], [Wu HPCA'15])
     - Tremendous potential, but accuracy often lacking for previously unseen devices
  4. Tools based on tape-out values
     - Time consuming, expensive, can only happen later in design process
  5. Low-level Spice models
     - Accurate, but often require proprietary information, hard to scale to large systems

**Early-stage power model tools divided, arch-specific, out-of-date**

**How do we support diverse options in gem5?**

# What Can We Do?

- Additional Challenges:
  - Each power modeling approach may be “best”
  - Often certain power models easier to integrate with certain simulators
- **Insight**: decouple “best” power model from simulator integration
  - Don’t pick which power model is the right one
  - Abstract away how simulators integrate → plug-and-play power models
- Vision: make power modeling as easy as performance modeling

**Today’s Focus: Application to gem5 (with McPAT)**

# Outline

- Motivation
- **Background**
- Design
- Methodology & Results
- Conclusion & Future Work

# Current Power Modeling in gem5

- Power modeling API takes user-defined equations as strings
  - **Simulation statistics** passed in as variables (e.g., cache hits)

```
# Wire up some example power models to the CPUs
for cpu in root.system.descendants():
    if not isinstance(cpu, m5.objects.BaseCPU):
        continue

    cpu.power_state.default_state = "ON"
    cpu.power_model = CpuPowerModel(cpu.path())
```

```
class CpuPowerOn(MathExprPowerModel):
    def __init__(self, cpu_path, **kwargs):
        super().__init__(**kwargs)
        # 2A per IPC, 3pA per cache miss
        # and then convert to Watt
        self.dyn = (
            "voltage * (2 * {}.ipc + 3 * 0.000000001 * "
            "{}.dcache.overallMisses / simSeconds)".format(cpu_path, cpu_path)
        )
        self.st = "4 * temp"
```

# Current Power Modeling in gem5 (Cont.)

- Power modeling API takes user-defined equations as strings
  - **Simulation statistics** passed in as variables (e.g., cache hits)
  - **Limitation**: Difficult for users to express complex functions, novel models
- Partial McPAT integration
  - But not updated in many years …
- Partial DVFS & thermal support
- Some ISAs have better power support (ARM [Reddy PATMOS‘17])
  - But also not updated in many years …

**Foundation to build off/learn from**

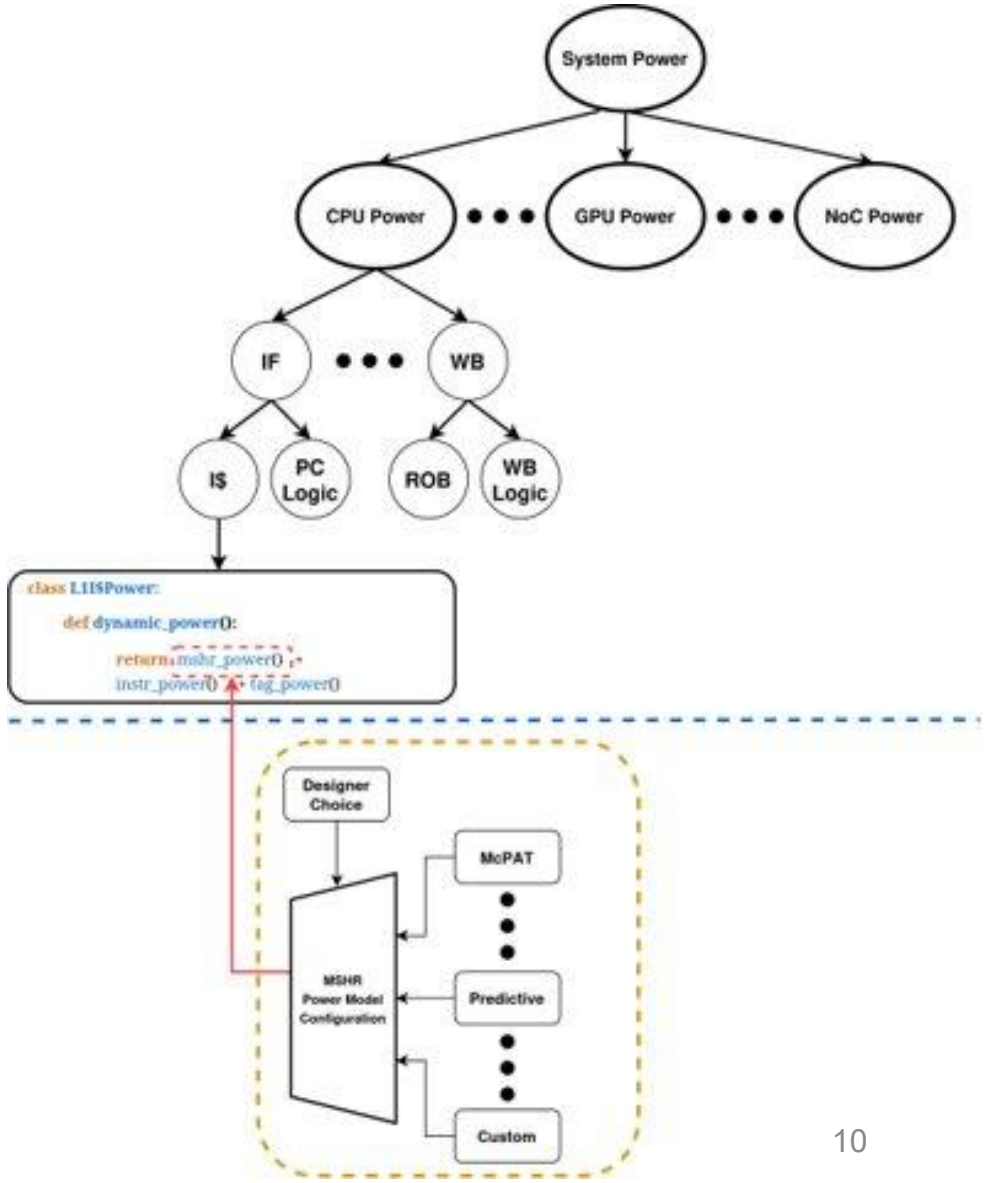
# Outline

- Motivation
- Background
- **Design**
- Methodology & Results
- Conclusion & Future Work

# Extending Power Modeling API [OSCAR'24]

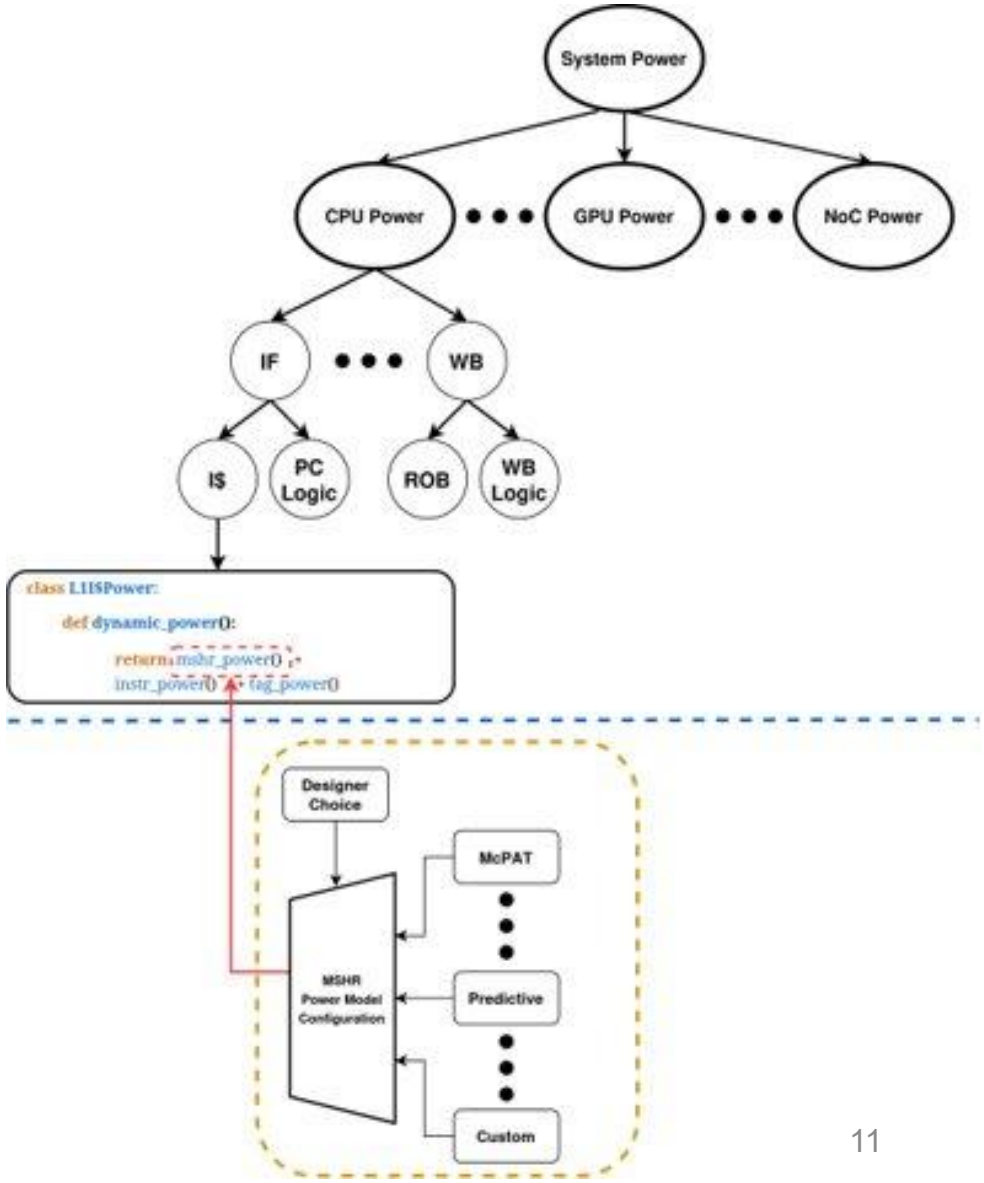
- ***How should we extend the power modeling API?***

- **Provides users interface which enables fine-grained customization**



# Extending the Power Modeling API (Cont.)

- **Hierarchical system of components**
  - Overall power model: sum of components
  - Separate simulator, power model
- Break model into 3 Key Pieces:
  - Simulator organizes hardware into sub-groups (e.g., known good models)
  - Power model(s): express static, dynamic power per component
  - Interface: pick between power models for a given component



# Putting It All Together

- **Uses gem5's Python front-end**
  - Can change power model choice as easily as cache size (no recompile)
  - Richer way to add support, easy to modify
  - Could use different power models for different components
- **Architecture-agnostic**
  - New power model "just" provides static, dynamic power values per component
  - Simulator handles rest of integration (plug-and-play)
- Can integrate and compare/validate different power models
  - **Existing** power models (e.g., McPAT)
  - **Pre-built** power models
  - **Custom** power models (e.g., in-house)

# Example: McPAT Power Model

- Integrate McPAT [Li MICRO'09] into our new interface
  - Uses first principles for hierarchical modeling
  - Models 5 stages (Fetch, Execute, LSU, Memory, Renaming)

- Basic Flow with new interface:
  - Grab McPAT activation energies
  - Activation energies with gem5 stats → power/component (e.g., BP, RF)
  - Hierarchically sum power/component into per stage, then per core, etc.

# Example: McPAT Power Model

- Integrate McPAT [Li MICRO'09] into our new interface
  - Models 5 stages (Fetch, Execute, LSU, Memory, Renaming)
  - Uses first principles for hierarchical modeling

- Our interface enables breaking down PM into separate functions!

```
class O3McPATCPUPowerOn(PowerModelPyFunc):
    def __init__(self, cpu: BaseO3CPU, act_energies):
        ...
        self._fetch = O3McPATFetchPower(cpu, act_energies, 1.0, 0.9)
        self._exec = O3McPATExecutePower(cpu, act_energies, 1.0, 0.76)
        self._lsu = O3McPATLsuPower(cpu, act_energies, 1.0, 0.71)
        self._mmu = O3McPATMmuPower(cpu, act_energies, 1.0, 0.71)
        self._rnu = O3McPATRenamingUnitPower(cpu, act_energies, 1.0)

    def dynamic_power(self):
        return self._fetch.dynamic_power() + ... + self._rnu.dynamic_power()
```

# Example: McPAT Power Model (Cont.)

- Deeper Dive Into Fetch stage modeling:

```
from mcpat_power_model import McPATPowerModel # Base class defining helper fns
class O3McPATFetchPower(McPATPowerModel):
    def __init__(self, cpu, act_energies, pipeline_act_factor, ifu_act_factor):
        ...
        self._decode = O3McPATDecodePower(cpu, act_energies)
        ...
    def dynamic_power(self):
       return (
          self._decode.dynamic_power()
          + ...
       )
```

```
class O3McPATDecodePower(McPATPowerModel):
    def __init__(self, cpu, act_energies):
        ...
    def dynamic_power(self):
        return self.to_watts(self.inst_buffer_energy() + self.inst_decode_energy())
    def inst_buffer_energy(self):
        decoded_insts = self.get_stat(decode.decodedInsts)
        return (
           decoded_insts * self.act_energies["IB"]["Read"]
           + decoded_insts * self.act_energies["IB"]["Writes"]
        )
```

# Outline

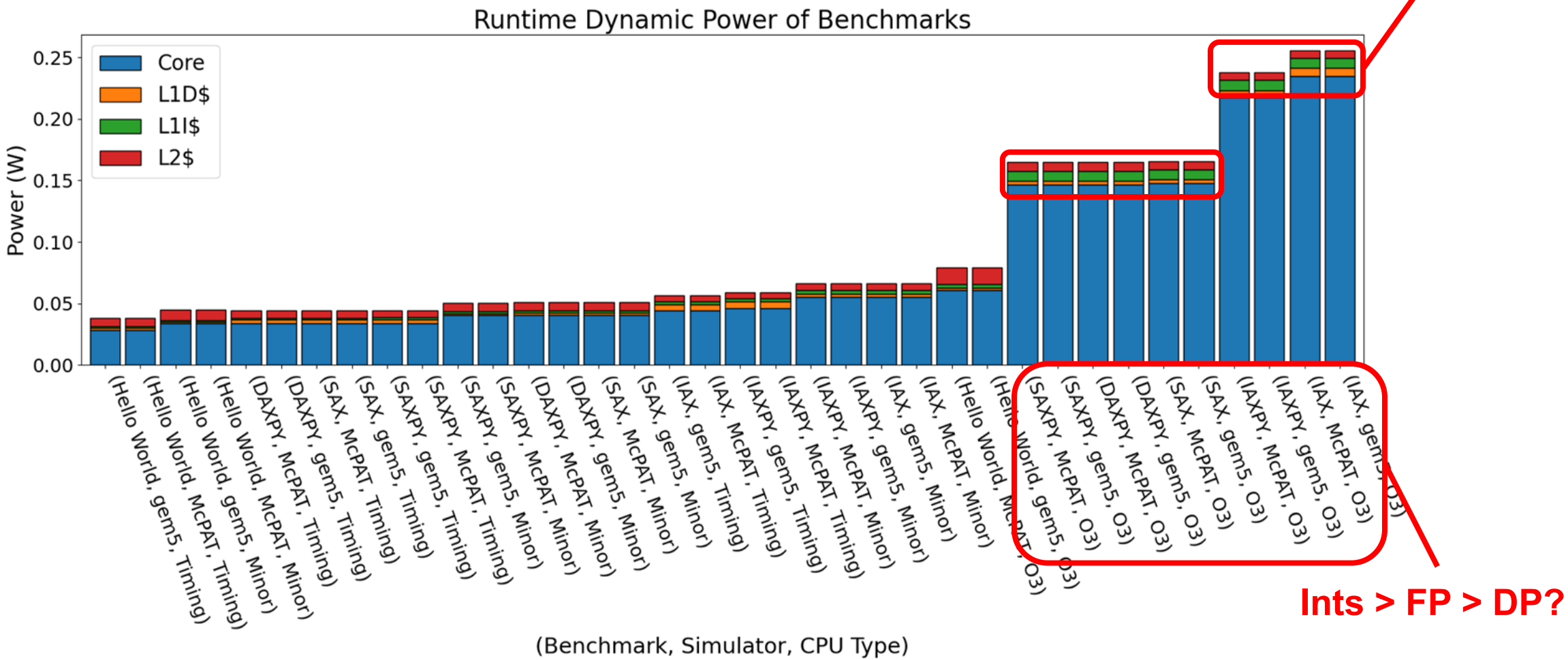
- Motivation
- Background
- Design
- **Methodology & Results**
- Conclusion & Future Work

# Methodology

- Benchmarks:
  - gem5-Resources (e.g., Hello World)
  - a*X + Y (*AX, *AXPY) variants
- CPUs: Timing, Minor, O3 (all 1 core/thread)

- Goal: validate gem5 McPAT integration vs. standalone McPAT
  - Use same configuration and statistics as gem5
  - Turned off m5ops (ROI markers) due to instr. count variations

# Preliminary Results




**a*X > a*X + Y?**

**Ints > FP > DP?**

**Mostly follow expectations (e.g., O3 > Minor > Timing), closely match McPAT!**

**Why Do These Results Occur?**

# Preliminary Results (Cont.)

- Several unintuitive results … did we integrate McPAT poorly?
  - No!  Results properly reflect McPAT's behavior
- Unintuitive results highlight McPAT several flaws:
  - Does not model vector instructions
  - Does not distinguish between single and double precision
- Takeaways:
  - New interface faithfully models McPAT
  - Enables rapid prototyping of power models via Python
  - Interface allows quality evaluation of known/new power models

# Outline

- Motivation
- Background
- Design
- **Conclusion & Future Work**

# Conclusion

- Future systems need to balance power, performance even more
- But power models are out-of-date, brittle, or proprietary
- Insight: decouple simulator power model integration, power model
  - Simulator devs: focus on how power should be integrated …
  - … without worrying about specifics of underlying power model
- Potential Benefits:
  - Easily support & simple to change between many different power models
  - Better maintainability – separate power model and simulator design
  - Easier to integrate new power models (e.g., for novel accelerators)
  - Integration with mainline gem5 ongoing
- **Make power modeling as easy as performance modeling**