Core C++ 2025

19 Oct. 2025 :: Tel-Aviv

(Intel)ligent Verification: Applying AI to Tame Microprocessor Complexity

Lotem Dalal

Lotem Dalal

- Curious
- Lifelong learning advocate
- Science and technology oriented

Agenda

- Introduction - CPU R&D cycle and verification
- why using AI for CPU verification
- Main uses of AI in CPU verification @ intel

Introduction

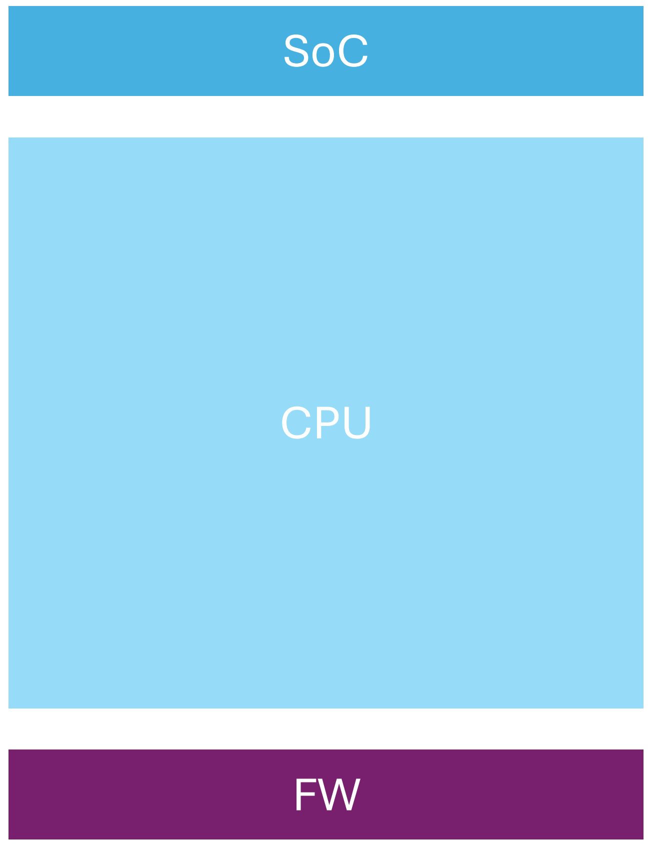
Designing a CPU core is like coordinating a million tiny, autonomous teams - each with different clocks, priorities and failure modes -

to produce one perfectly ordered stream of instructions.

Proving it never fails, for decades.

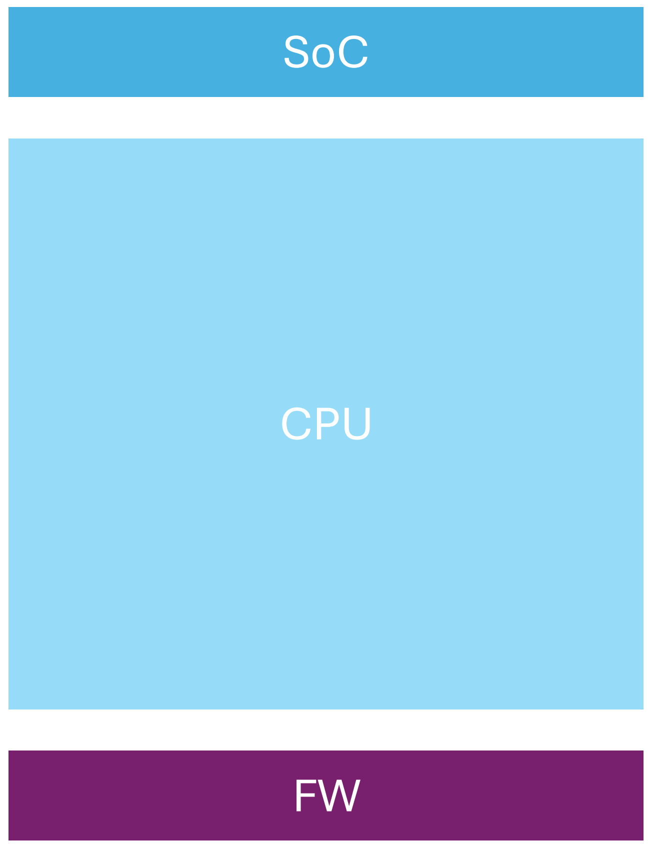
introduction

- Micro-processor consists mainly of:
  - **SoC** (system on chip) **–** peripheral subsystems
  (GPU, I/O controllers, memory controller, accelerators)
  - **CPU** (central processing unit) **–** processing cores
  (execution units, caches, control)
  - **FW** (firmware) – hardware-software interface
  embedded programmable SW layer (μcode, BIOS)



CPU CORE R&D cycle

- Core of a CPU is the most complex IP, with:
  - **2M+** Files
  - **500M+** transistors
  - **1k+** Engineers developing it



CPU CORE Verification

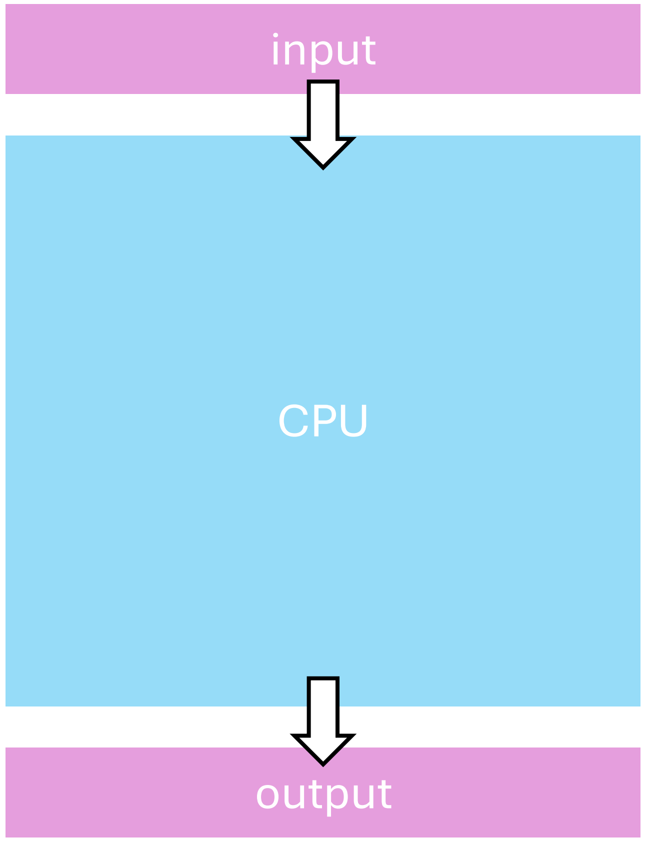
- Chip verification aims assuring no bugs are found in the DUT (**d**esign **u**nder **t**esting).

- The methodology to achieve it consists of:

  - **Stimuli** – injecting constrained-random inputs.

  - **Checking –** rules the DUT must follow

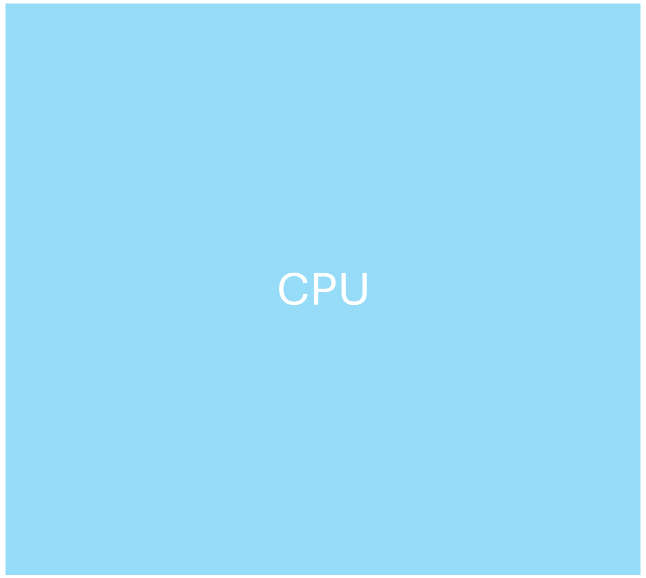
  - **Coverage -** quantitative indicators that show how much of the design or specification has been verified.



CPU CORE Verification - overview

- Each project includes the following stages:

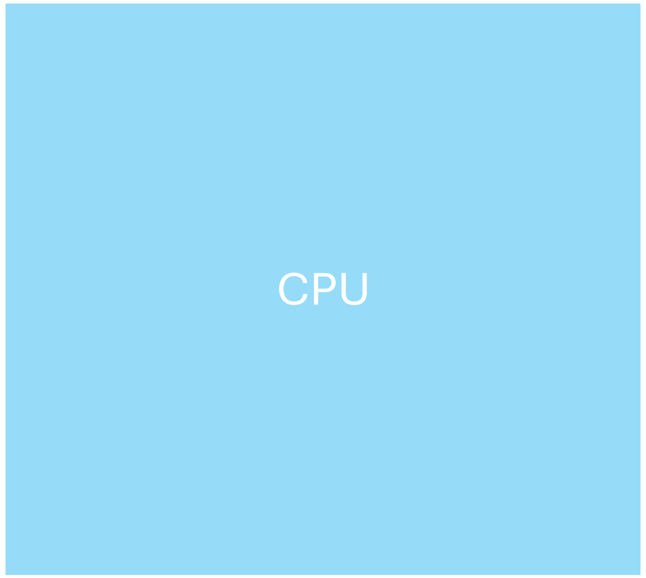
1. **Planning** 🔎
   - Architectural spec review
   - Design & Verification plans
2. **Execution** 💻
   - Introducing new features
   - Updating existing features
3. **Analysis** 🐞
   - Focus on quality – debug and coverage
4. **Closure** ⚖️
   - Closing coverage gaps
   - Bug fix is <u>limited</u>



CPU CORE Verification - overview

- Average time estimation of the stages:

1. **Planning** 🔎 10% (*dynamic)
   - Architectural spec review
   - Design & Verification plans
2. **Execution** 💻 70%
   - Introducing new features
   - Updating existing features
3. **Analysis** 🐞 15%
   - Focus on quality – debug and coverage
4. **Closure** ⚖️ 5%
   - Closing coverage gaps
   - Bug fix is <u>limited</u>

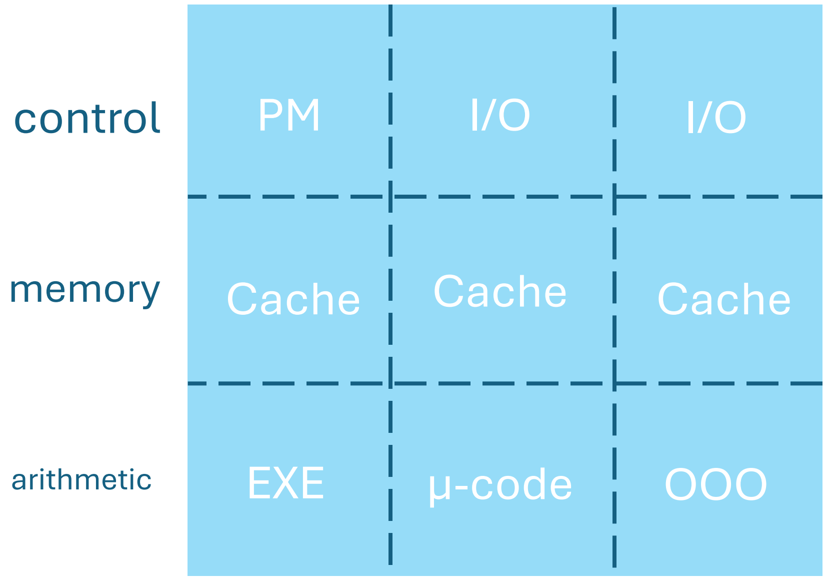


Most of the CPU (pre-Si) projects take ~2 years

CPU CORE Verification – domain level

- Each project includes the following phases:

1. **Planning** 🔎
   - Architectural spec review
   - Design & Verification plans
2. **Execution** 💻
   - Introducing new features
   - Updating existing features
3. **Analysis** 🐞
   - Focus on quality – debug and coverage
4. **Closure** ⚖️
   - Closing coverage gaps
   - Bug fix is <u>limited</u>



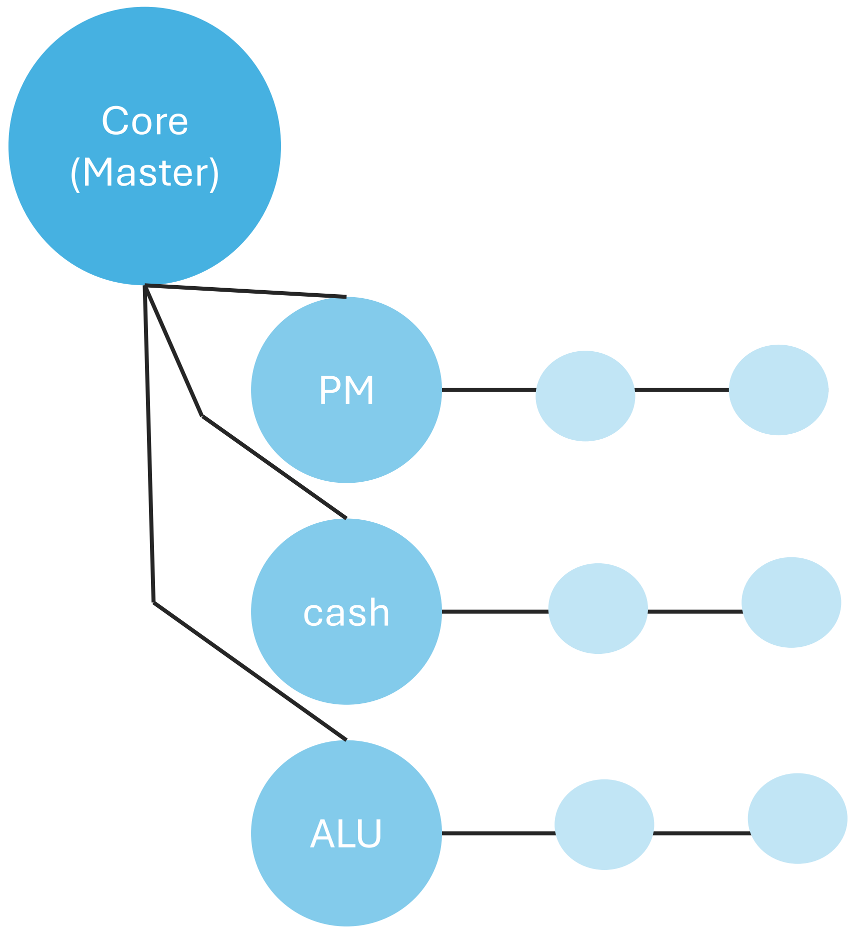
Most of the verification is done in a **domain level**

CPU CORE Verification – domain level

- Each project includes the following phases:

1. **Planning** 🔎
   - Architectural spec review
   - Design & Verification plans
2. **Execution** 💻
   - Introducing new features
   - Updating existing features
3. **Analysis** 🐞
   - Focus on quality – debug and coverage
4. **Closure** ⚖️
   - Closing coverage gaps
   - Bug fix is limited

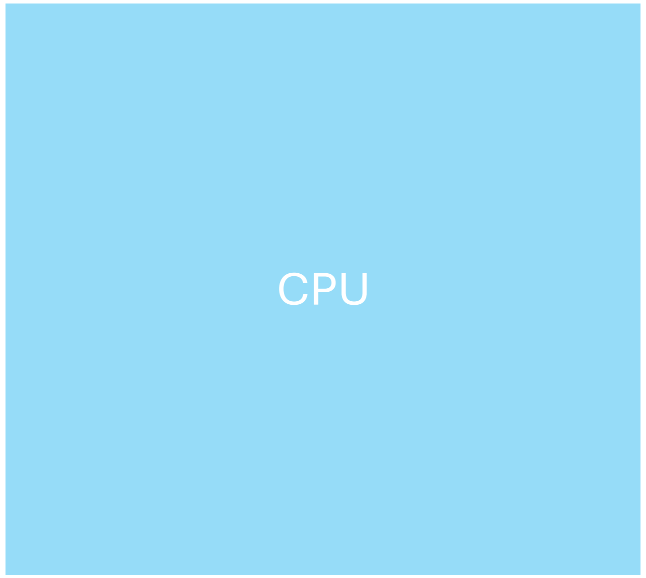
Most of the verification is done in a **domain level**



CPU CORE Verification – core level

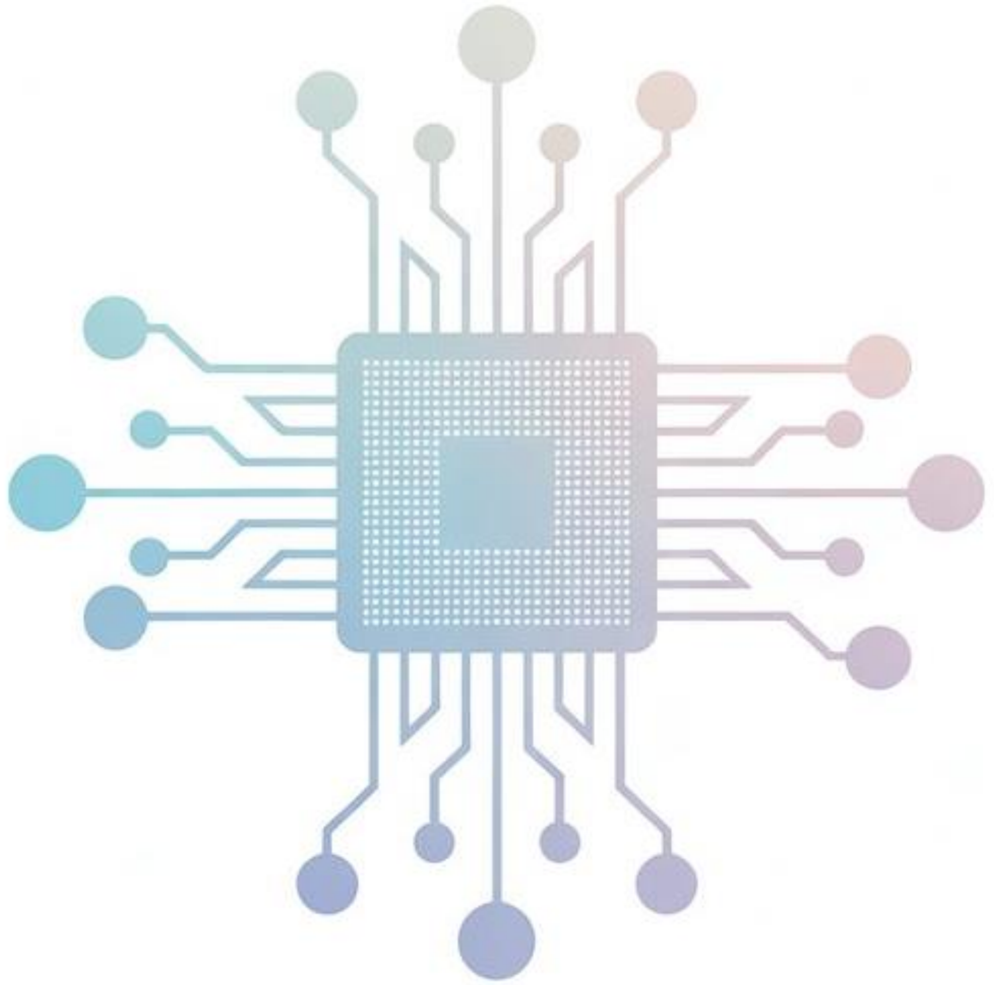
- Each project includes the following phases:

1. **Planning** 🔎
   - Architectural spec review
   - Design & Verification plans
2. **Execution** 💻
   - Introducing new features
   - Updating existing features
3. **Analysis** 🐞
   - Focus on quality – debug and coverage
4. **Closure** ⚖️
   - Closing coverage gaps
   - Bug fix is <u>limited</u>



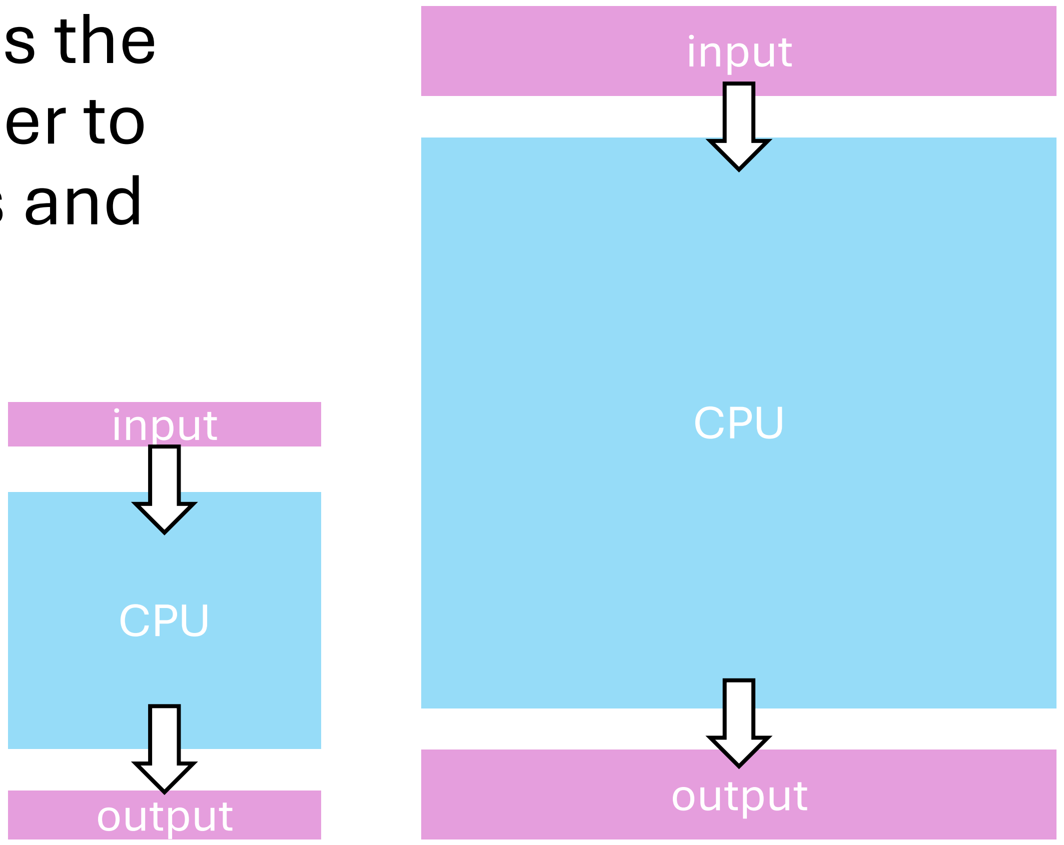
Main functionalities are also done in a **core level**

How can AI enhance CPU R&D?



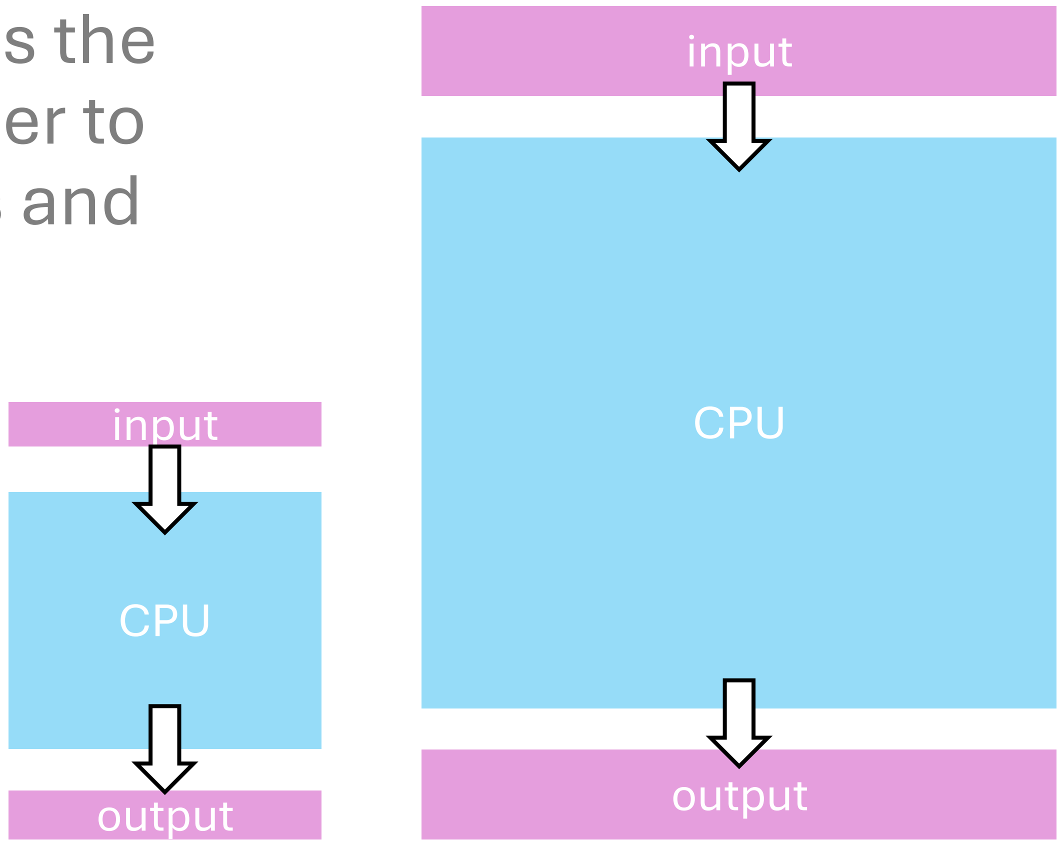
How can AI enhance CPU R&D?

- As technology advances, so does the CPU complexity – making it harder to simulate ALL possible scenarios and getting FULL coverage.



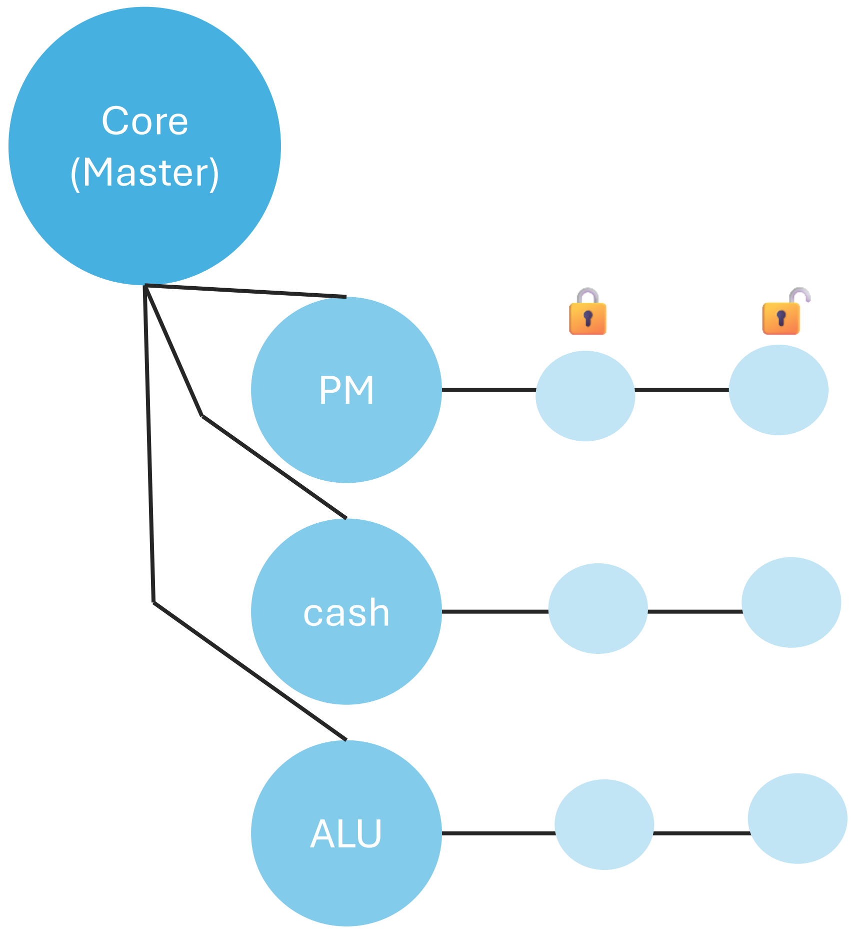
How can AI enhance CPU R&D?

- As technology advances, so does the CPU complexity – making it harder to simulate ALL possible scenarios and getting FULL coverage.
- Using AI and ML in CPU verification allows us to make the best out of the R&D resources.



1. Optimizing gating lists

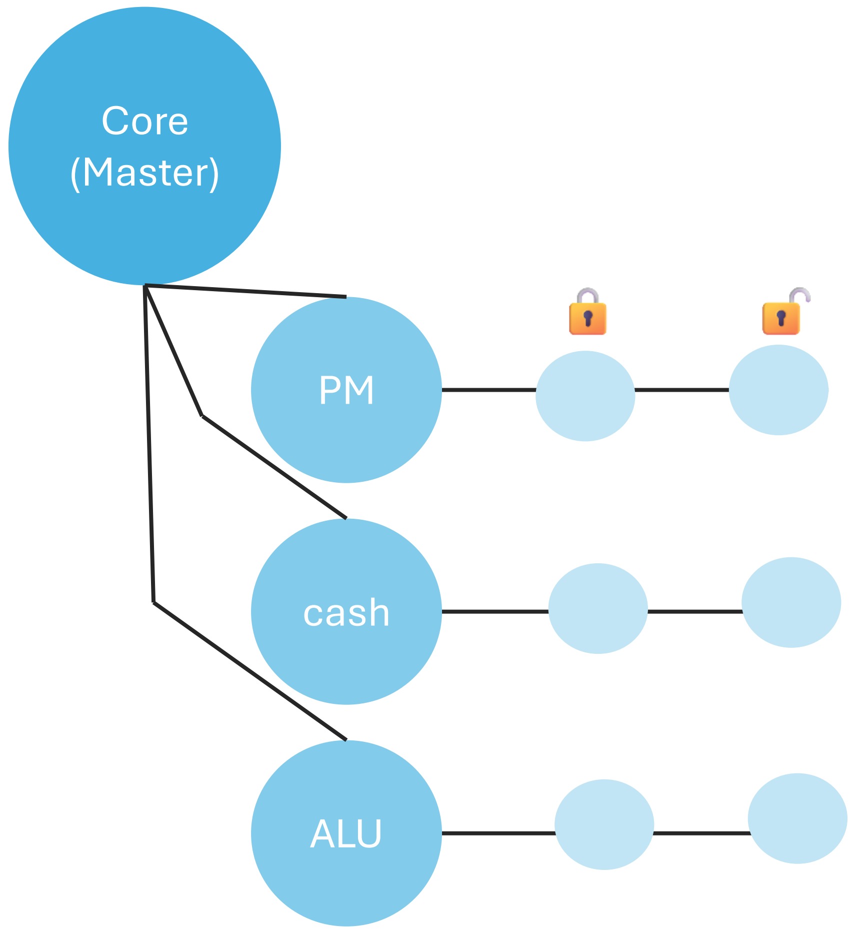
- “gate keeper” – identifying level 0 errors/bugs before merging to repo
  - Cover wide range of scenarios
  - **Not randomized**
  - Save compute resources
  - Minimum runtime – enhance R&D productivity



1. Optimizing gating lists

- "gate keeper" – identifying level 0 errors/bugs before merging to repo

- How can AI help me?
  - Analyze already written tests
  - Suggest minimum tests that cover maximum scenarios
  - Improving runtime of tests



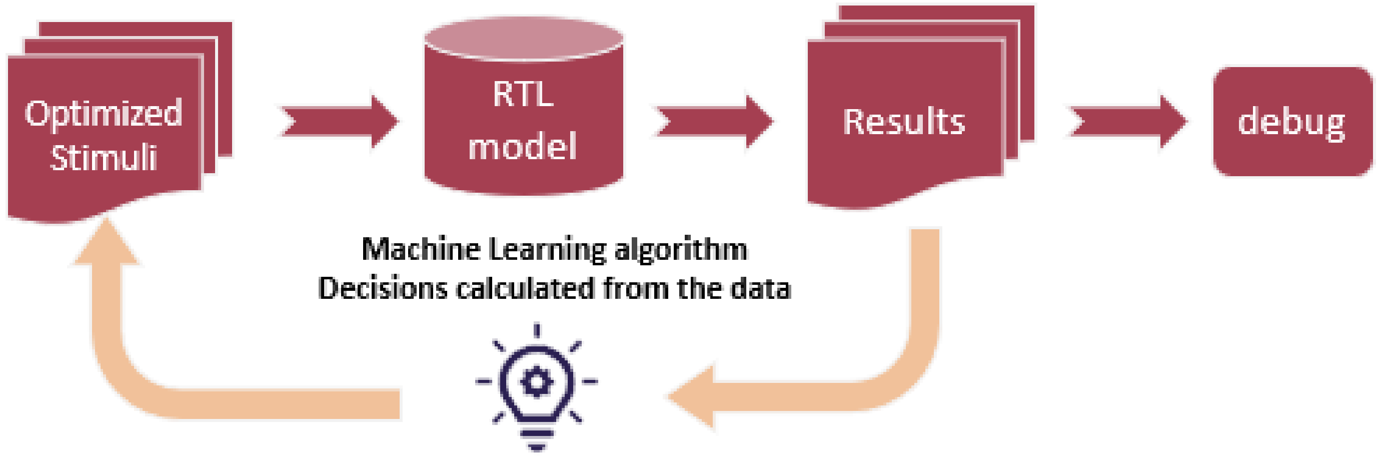
2. Optimizing Weekly regression

- Predefined high count **static** lists running on a weekly basis
  - Cover all desired scenarios
  - Maximum randomization
  - **Save compute resources**
  - Minimum runtime – enhance R&D productivity

2. Optimizing Weekly regression

- Predefined high count **<u>static</u>** lists running on a weekly basis

- How can AI help me?
  - Analyze static lists ("batch 0") results → recommend new lists ("ML batches")
  - Running ML lists based on prior runs – those lists are **<u>dynamic</u>**.


Optimized Stimuli
RTL model
Results
debug
Machine Learning algorithm
Decisions calculated from the data

3. Intelligent stimuli generation

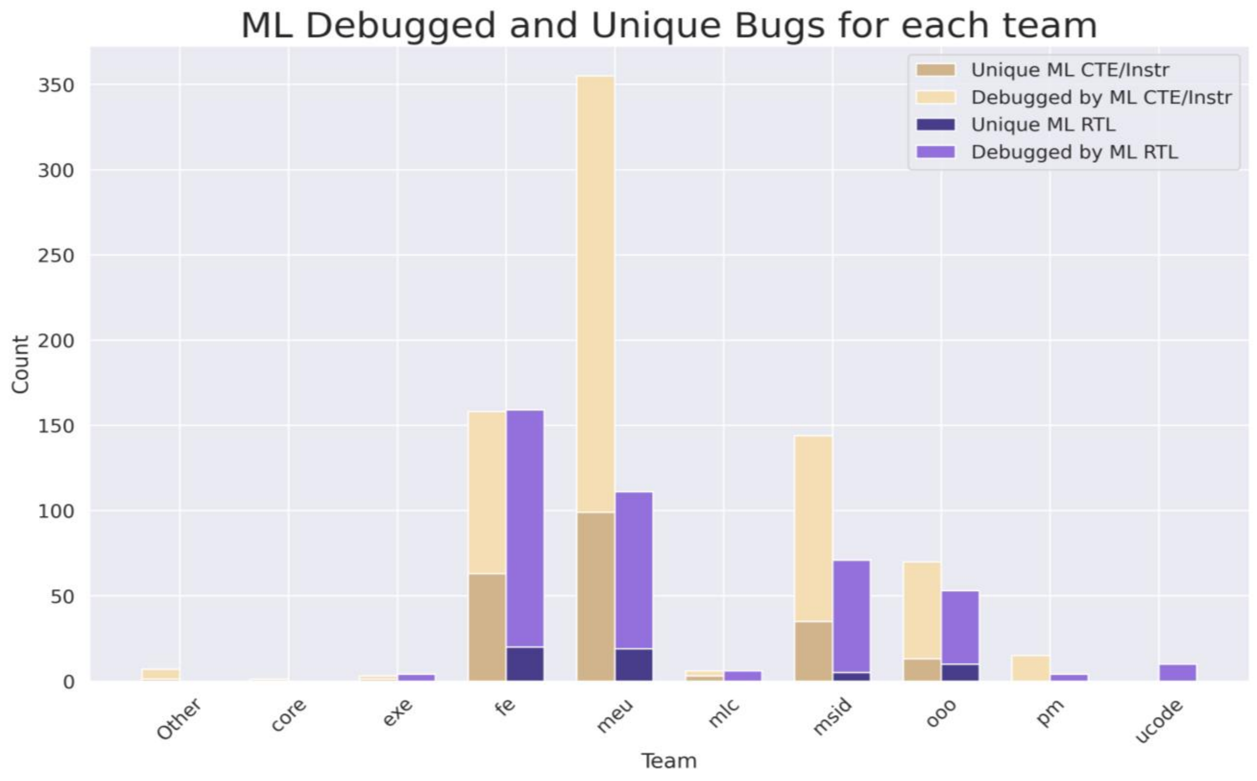
- Our target is to maximize DUT coverage while maintaining moderate compute resources

- We want to maximize coverage
    - Code coverage
    - Functional coverage

- We want to minimize:
    - Number of tests
    - simulation runtime

3. Intelligent stimuli generation

- How can AI help?
  - identify redundant or ineffective tests in weekly regression
  - Suggest configurations to enhance coverage (non hit/ low count scenarios)
  - Analysis of the total coverage map, point out blind spots that otherwise might be skipped



4. Reproducing rare failures

- During **Closure** ⚖️ stage we focus on debug and coverage gaps
  - Debugging rare failure clusters is challenging:
    - Low test count
    - less data for ML analysis
    - Not reproduce in <u>each</u> weekly regression
    - lower priority for debug
    - The rarest failures sometimes “hide” a unique RTL bug

4. Reproducing rare failures

- During **Closure** ⚖️ stage we focus on debug and coverage gaps
  - Using AI we can:
    - review historical regression data
    - Analyze the data and find patterns
    - Creating a list with ML configurations

⬇

    - Analyze the results to optimize ML recommendations

Prob(X)=50%

Prob(X given A=True and C=False)=**75%**

Input A	Input B	Input C	Input D	Target
TRUE	TRUE	FALSE	FALSE	X
TRUE	FALSE	FALSE	TRUE	X
TRUE	TRUE	FALSE	TRUE	Y
FALSE	FALSE	TRUE	FALSE	Y
FALSE	FALSE	TRUE	TRUE	Y
TRUE	TRUE	FALSE	TRUE	X

4. Reproducing rare failures

- During **Closure** ⚖️ stage we focus o[illegible] coverage gaps

Examples from PNC:

Vanishing buckets lists hit RTL bugs in FE, OOO and CTE bug in MSID

- Using AI we can:
  - review historical regression data
  - Analyze the data and find patterns
  - Creating a list with ML configurations

  - Analyze the results to optimize ML recommendations

Prob(X)=50%

Prob(X given A=True and C=False)=**75%**

Input A	Input B	Input C	Input D	Target
TRUE	TRUE	FALSE	FALSE	X
TRUE	FALSE	FALSE	TRUE	X
TRUE	TRUE	FALSE	TRUE	Y
FALSE	FALSE	TRUE	FALSE	Y
FALSE	FALSE	TRUE	TRUE	Y
TRUE	TRUE	FALSE	TRUE	X

summary

- CPU R&D and verification is a highly complicated project
- As technology advances, the challenge of achieving high confidence in the product prior to TAPE-IN is increasing
- AI is optimizing many of the verification tasks to enhance productivity and better use our resources.

Thank you!

- annieda666@yahoo.com
- linkedin.com/in/lotem-dallal/