Building Buggy Chips - That Work!

Todd Austin

Advanced Computer Architecture Laboratory
University of Michigan

The DIVA Project

http://www.eecs.umich.edu/diva

- Researchers
  - Chris Weaver (lead), Pat Cassleman, Amit Marathe, Saugata Chatterjee (alum), Todd Austin, Maher Mneimneh (FV), Fadi Aloul (FV), Karem Sakallah (FV)
- Key technology: ***Dynamic Verification***
  - Simple, fast and reliable online checkers that detect and correct system faults
- Benefits we are exploring
  - Improved quality and time-to-market through reduced burden of verification
  - More reliable designs with high resistance to radiation and noise
  - More efficient (or aggressive) circuit technologies via online electrical verification
  - Reduced complexity via performance (rather than correctness) focused designs
- Technology demonstration vehicles
  - **REMORA** self-checked microprocessor
  - **DIVA Demo** self-checked crypto-system (using commercial off-the-self parts)

Talk Overview

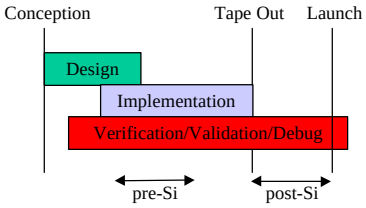
- Verification Challenges
- Dynamic Verification: Seatbelts for Your CPU
- Checker Processor Architecture
- Value-Added Optimizations
- Ongoing Work
- Conclusions

Correctness As Value

- What do you *value* most about your computer system?
  - Performance?
  - Cost?
  - *Correctness?*
- Correctness is uncompromising, all value is predicated on it!
  - A correct system may have value
  - An incorrect system design will be perceived as worthless
- Correctness disasters
  - Intel FDIV bug, failing FP divider resulted in $475 million recall
  - MIPS R10000 faltered out of the chute, many early parts recalled
  - Transmeta recalled most early Crusoe parts

Designing Correct Systems

- When is a design correct?
  - *" starting states ($state_i$, $inputs_i$), next state ($state_{i+1}$) is correct*
- When is a design complete?
  - When it is correct
    - Employ verification
    - *Did we build the system right?*
  - When it meets customers' needs
    - Employ validation
    - *Did we build the right system?*



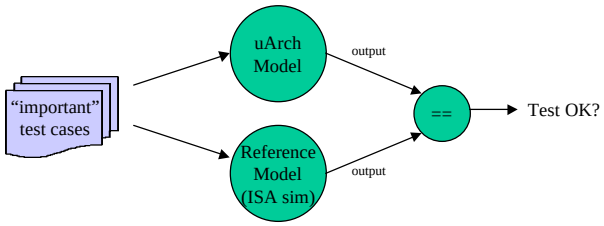
- Verification generally considered a more difficult task as it must consider all programs, not just important ones

The Burden of Verification

- Immense test space
  - Impossible to fully test the system
  - For example, 32 regs, 8k caches, 300 pins = 2^{132396} states
  - Conservative estimate, microarchitectural state increases the test space
- Done with respect to ill-defined reference
  - What is correct? Often defined by PRM + old designs + guru guidance
- Expensive
  - Large fraction of design team dedicated to verification
  - Increases time-to-market, often as much as 1-2 years
- High-risk
  - Typically only one chance to "get it right"
  - Failures can be costly: replacement parts, bad PR, lawsuits, fatalities

Simulation Based Verification

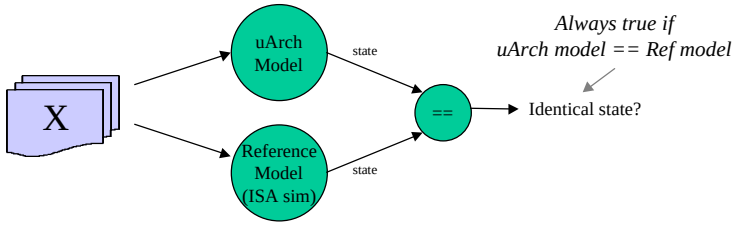
- Determines if design is functionally correct at the logic level
- Implemented with *co-simulation* of "important" test cases
  - Mostly before tape out using RTL/logic level simulators



- Differences found at output drive debug
- Process continues until "sufficient" coverage of test space

Formal Verification

- Formal verification speeds testing by comparing models
  - Compare reference and uArch model using formal methods (e.g., SAT)
  - If models shown functionally equivalent, any program renders same result
  - Much better coverage than simulation-based verification



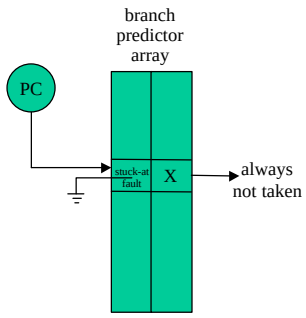
- Unfortunately, intractable task for complete modern pipeline
  - Problems: imprecise state, microarchitectural state, out-of-order operations
  - Machines we build are **not** functionally equivalent to reference machine!

Deep Submicron Reliability Challenges

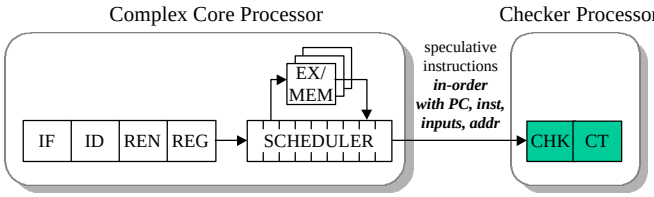
- More difficult to build robust systems in denser technologies
  - Degraded signal quality
    - Increased interconnect capacitance results in signal crosstalk
    - Reduced supply voltage degrades noise immunity
    - Increased current demands (*di/dt* spikes) create supply voltage noise
  - Single event radiation/soft errors (SER)
    - Alpha particles (from atomic impurities) and gamma rays (from space)
    - Energetic particle strikes destroy charge, may switch small transistors
    - Inexpensive shielding solutions unlikely to materialize
  - Increased complexity
    - More transistors will likely mean greater complexity
    - Verification demands and probability of failure will increase

Motivating Observations

- Speculative execution is *fault-tolerant*
  - Design errors, timing errors, and electrical faults only manifest as performance divots
  - Correct checking mechanism will fix errors
- What if all computation, communication, control, and progress were speculative?
  - Any incorrect computation fixed
    - *maximally speculative*
  - Any core fault fixed
    - *minimally correct*

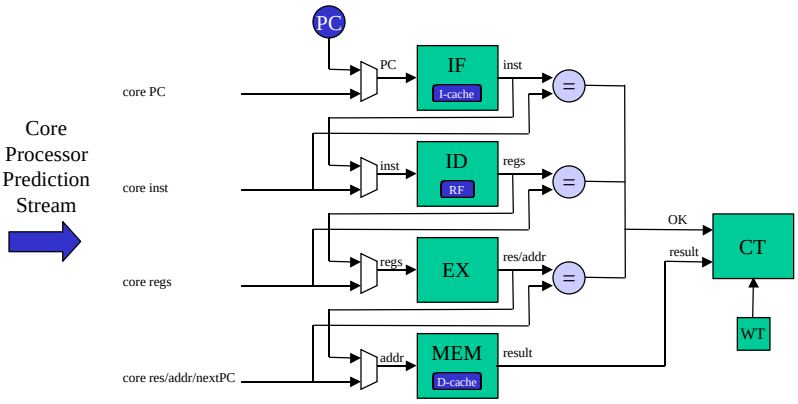


Dynamic Verification: Seatbelts for Your CPU

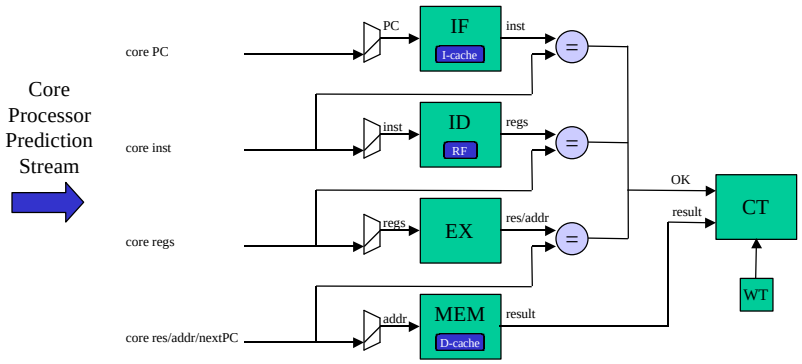


- Core computation, communication, and control validated by checker
  - Instructions verified by checker in program order before retirement
  - Checker *detects and corrects* faulty results, restarts core
- Checker relaxes the burden of correctness on the core processor
  - Robust checker corrects faults in any core structure not used by checker
  - Tolerates core design errors, electrical faults, silicon defects, and failures
  - Core only has burden of high accuracy prediction
- Key checker requirements: *simple*, *fast,* and *reliable*

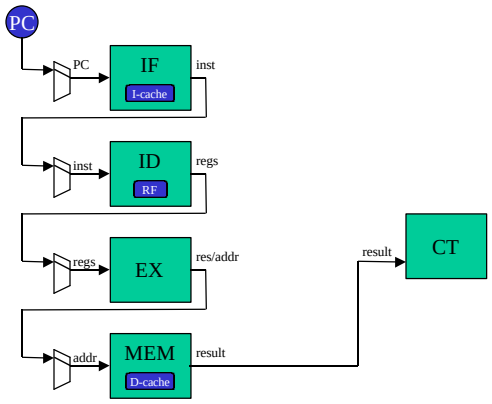
Checker Processor Architecture



Check Mode

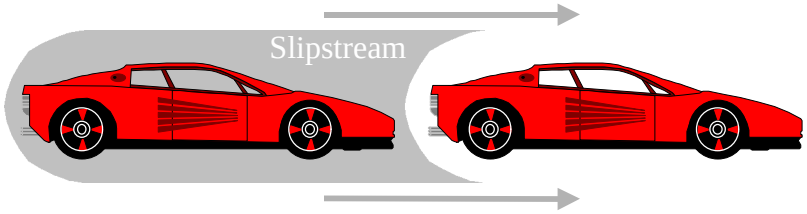


Recovery Mode



How Can the Simple Checker Keep Up?

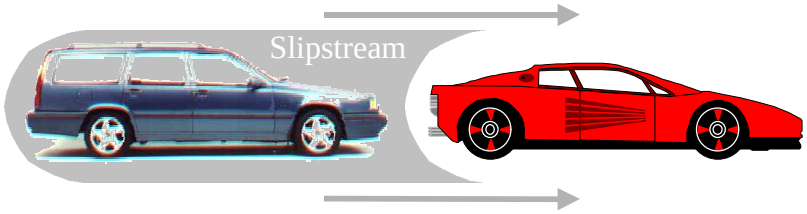
Redundant Core Advance Core



- Slipstream effects reduce power requirements of trailing car
  - Checker processor executes in the core processor slipstream
  - fast moving air $\Rightarrow$ branch/value predictions and cache prefetches
  - Core processor slipstream reduces complexity requirements of checker
- Symbiotic effects produce a higher combined speed

How Can the Simple Checker Keep Up?

Simple Checker Complex Core



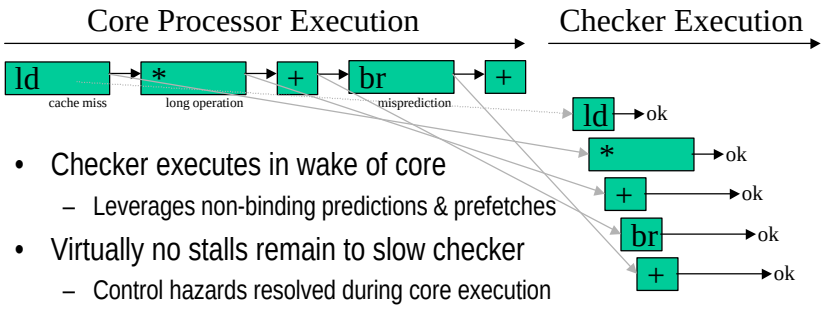
- Slipstream effects reduce power requirements of trailing car
  - Checker processor executes in the core processor slipstream
  - fast moving air $\Rightarrow$ branch/value predictions and cache prefetches
  - Core processor slipstream reduces complexity requirements of checker
- Symbiotic effects produce a higher combined speed

Advanced Computer Architecture Lab
University of Michigan

Building Buggy Chips - That Work!
Todd Austin

Speeding the Checker with Core Computation

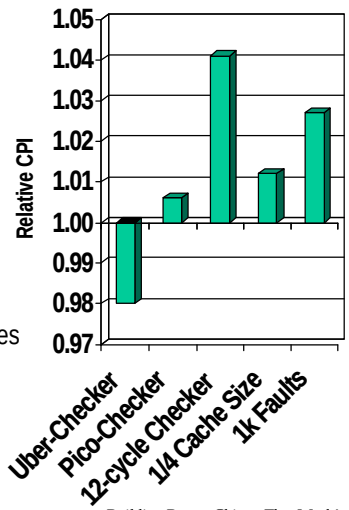
```
            ld f1,(X)
            f4 = f1 * f2 + f3
            br f4 < 0, skip
              r8 = r8 + 1
skip:       ...
```



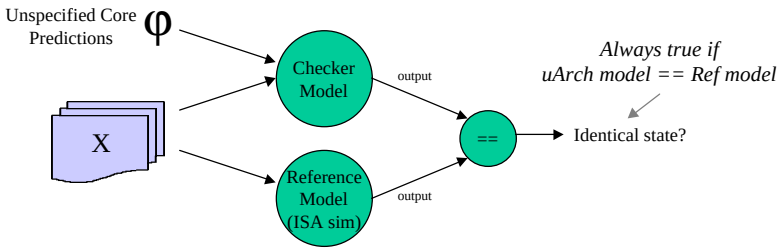
- Checker executes in wake of core
  - Leverages non-binding predictions & prefetches
- Virtually no stalls remain to slow checker
  - Control hazards resolved during core execution
  - Data hazards eliminated by prefetches and input value predictions
- Complex microarchitectural structures only necessary in core

Checker Performance Impacts

- Checker *throughput* bounds core IPC
  - Only cache misses stall checker pipeline
  - Core warms cache, leaving few stalls
- Checker *latency* stalls retirement
  - Stalls decode when speculative state buffers fill (LSQ, ROB)
  - Stalled instructions mostly nuked!
- *Storage hazards* stall core progress
  - Checker may stall core if it lacks resources
- *Faults* flush core to recover state
  - Small impact if faults are infrequent



Verifying the Checker Processor



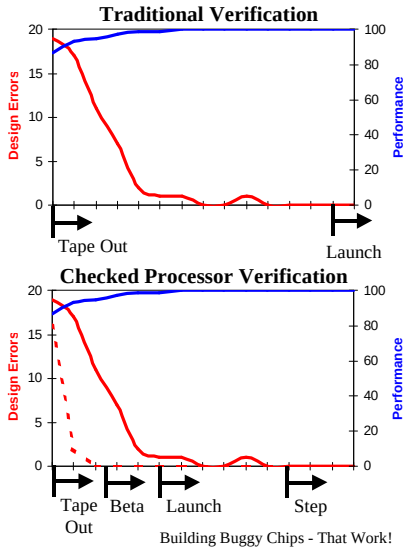
- Simple checker permits complete functional verification
  - In-order blocking pipelines (trivial scheduler, no rename/reorder/commit)
  - No "internal" non-architected state
- Fully verified design using Sakallah's GRASP SAT-solver [DAC01]
  - For Alpha integer ISA without exceptions
  - With small register file and memory, and small data types

Value-Added Optimizations

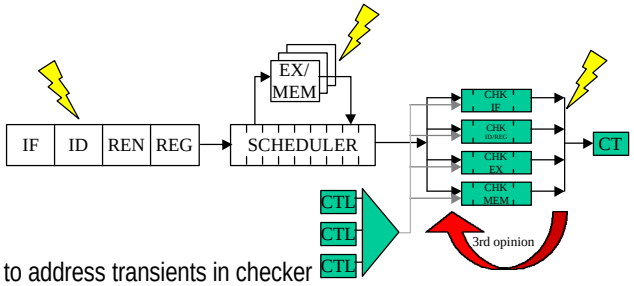
- Fault tolerant and deeply speculative core processor permits many fundamental assumptions of system design to be revisited!
  - Beta-Release Processors
  - Low-Cost SER and Noise Protection
  - Fully Testable Microprocessor Designs
  - Self-tuned Digital Systems

Beta-Release Processors

- Traditional verification stalls launch until debug complete
- Checked processor verification could overlap with launch
  - Beta-release when checker works
  - Launch when performance stable
  - Step as needed without recalls



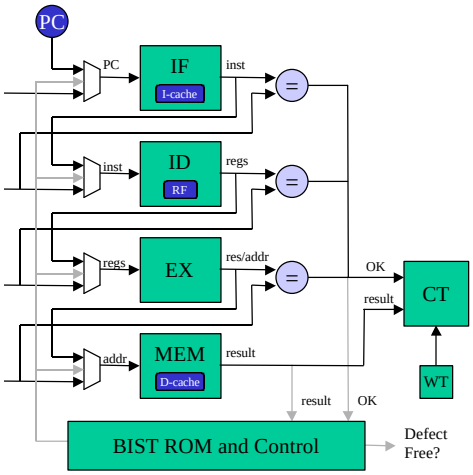
Low-Cost SER and Noise Protection



- Only need to address transients in checker
  - Checker detects and corrects noise-related faults in core
  - Core processor designed without regard to strikes (e.g., no ECC…)
- Recycle checker inputs suspected core fault
  - If no error on third execution, transient strike in checker processor
  - If error on third execution, core processor fault occurred (e.g., SER, design error)
- Protect critical checker control with triple-modular redundant (TMR) logic
  - TMR on simple control results in only 1.3% larger checker (synthesized design)

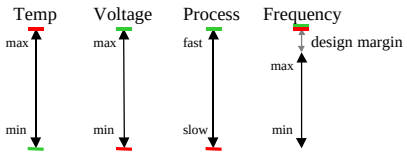
Fully Testable Microprocessor Designs

- Checker structure facilitates manufacturing tests
  - All checker inputs exposed to built-in-self-test logic
  - Checker provides built-in test signature compression
- Checker can be fully tested with small BIST module
  - less than 0.5% area increase
- Reduces burden of testing on core
  - Missed core defects corrected
  - Checker acts as core tester



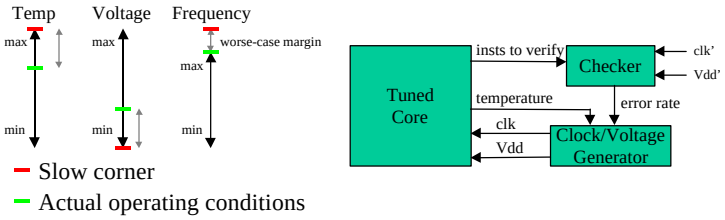
Self-Tuned Digital Systems

- Electrical verification determines if implementation is robust
  - Design must be functionally correctly for all valid *(T,V,p,f)*
  - Design must meet mean-time-to-failure goals (via power/current analysis)



- Verify functional correctness at slow corner (T_{max},V_{min},p_{slow},f_{max})
- Verify power/current characteristics at fast corner (T_{min},V_{max},p_{fast},f_{max})
- Additional margin on clock used to *avoid* any electrical faults

Self-Tuned Digital Systems



- Modern logic design is too conservative for dynamic verification
  - Unnecessary design margins consume power and performance
  - System may not be operating at slow corner
- Checker enables a self-tuned clock/voltage strategy
  - Push clock, drop voltage until desired power-performance characteristics
  - If system fails, reliable checker will correct error, notify control system
  - Reclaims design margins plus any temperature and voltage margins

Ongoing Work: DIVAlution

- Goal: transfer dynamic verification technology to industry
- Approach
  - Developed lower-impact dynamic verification techniques (baby steps)
  - Build real technology demonstrations (hands-on demos)
- Technology demonstration vehicles
  - REMORA self-checked processor
    - 4-wide checker, 0.5k I-cache, 4k D-cache plus simple core
    - Unpipelined checker prototype: 325MHz, 12mm^2, 941mW in 0.25um
  - DIVA Demo self-checked crypto-processor
    - Leverages built-in CRC mechanism to implement low-cost checker
    - Built with commercial-off-the-shelf parts (dual StrongARMs)

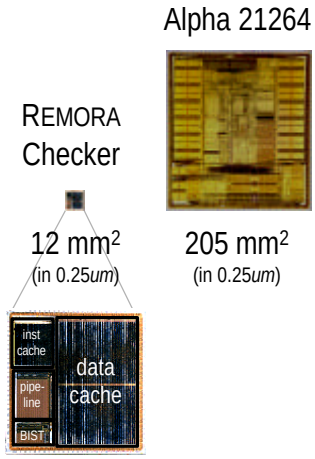
Conclusions

- Dynamic verification makes a core processor fully speculative
  - Maximally speculative, minimally correct design approach
  - Core processor tolerates any permanent or transient error
  - Core processor supports any form of speculation
- Checker processor design is simple, reliable, and fast
  - High-quality prediction stream keeps checker design simple
  - Simple latency-insensitive design permits robust implementations
  - Core processor eliminates hazards that could slow checker pipeline
- Pushing speculation to the limit may yield many benefits
  - Beta-release processors could overlap verification with launch
  - Checker processor provides single event radiation (SER) protection
  - Highly testable checker reduced manufacturing test burden on core logic
  - Fault-tolerant core can leverage aggressive circuits (self-tuned systems)

Backups

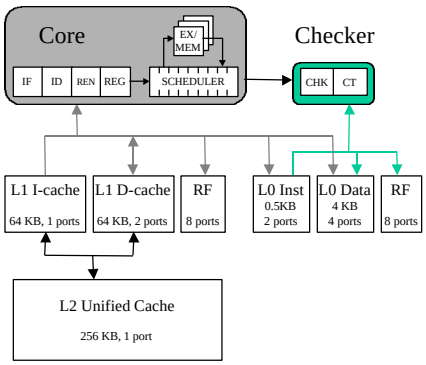
REMORA: Physical Checker Design

- Physical checker design effort underway
  - Alpha integer ISA subset
  - 4-wide checker, 0.5k I-cache, 4k D-cache
  - Synthesized design (using Synopsys tools)
- Physical design estimates
  - 325 MHz clock speed
  - 12 mm^2 total area in 0.25*um* technology
  - 941 mW worst-case power
- Design to be fabricated also includes
  - Pipelined checker design
  - Simple core pipeline
  - Clock/voltage tuning infrastructure
  - Extensive BIST support



Optimized System Architecture

- Performance impacts eliminated
  - Checker RF allows core commit
  - No storage hazards
  - Few checker cache misses
  - Less expensive core storage architecture (same as baseline)
- Core cache failures affect checker

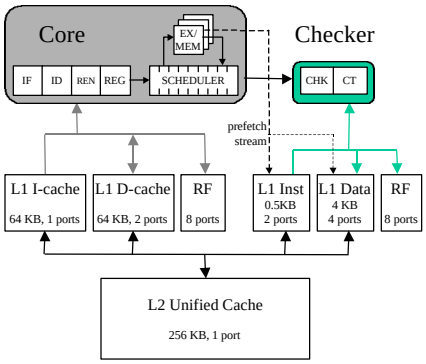


Slowdowns

-0.4% Best: -3.2%
Worst: 0.2%

Fully Decoupled System Architecture

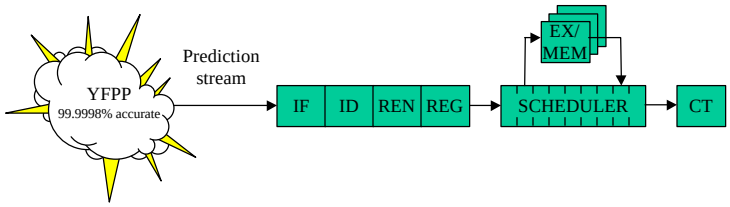
- Checker fully decoupled
  - Core L1 caches may fail
  - All L2 writebacks from checker
  - Core caches flushed on fault
  - Core accesses and misses warm up checker caches
- Eliminates common mode core cache failures
  - But, generates more L2 traffic
  - Further optimizations possible



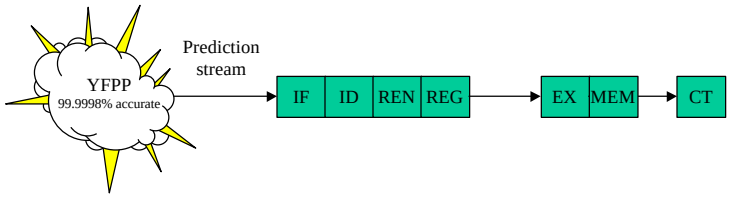
Slowdowns

1.2% Best: 0%
Worst: 6.7%

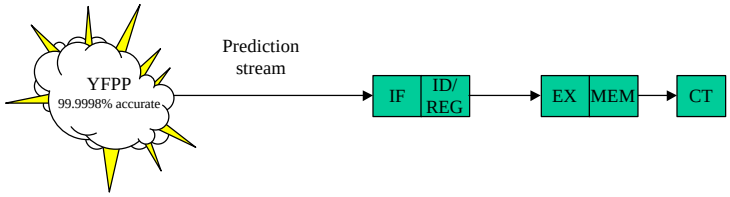
Deriving Dynamic Verification



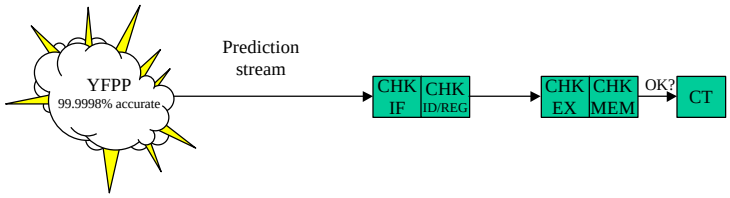
Deriving Dynamic Verification



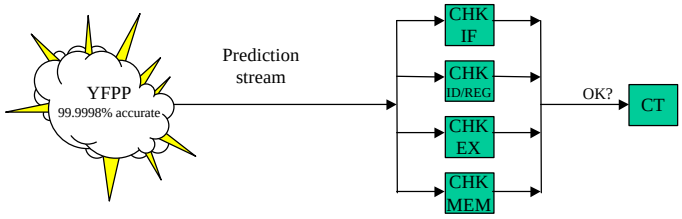
Deriving Dynamic Verification



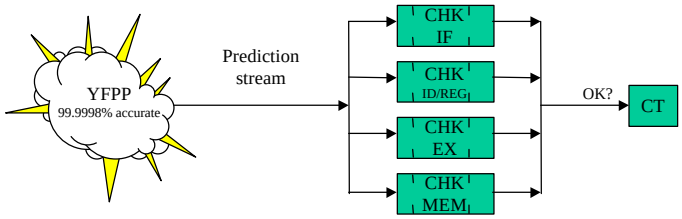
Deriving Dynamic Verification



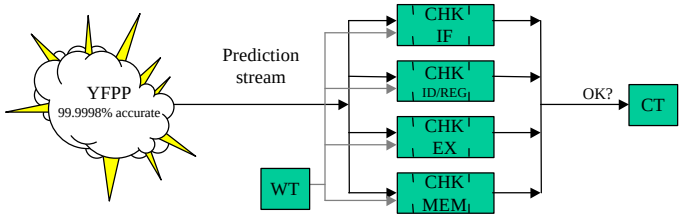
Deriving Dynamic Verification



Deriving Dynamic Verification



Deriving Dynamic Verification



Deriving Dynamic Verification