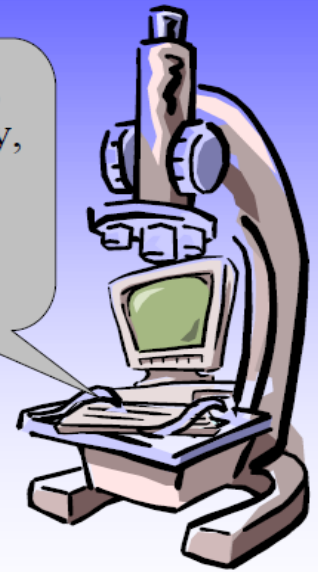


ΠΡΟΣΟΜΟΙΩΣΗ ΑΡΧΙΤΕΚΤΟΝΙΚΗΣ

Computer Architecture

- Processor Design
 - Single thread → pipeline, branch prediction
 - Multiple threads → SMT resource allocation, threads scheduling
- Ετερογενείς αρχιτεκτονικές
 - General Purpose Computing, Gaming, and Entertainment Devices
 - Intel Haswell, Broadwell, Skylake with/without Graphics (Iris Pro, HD)
 - AMD Fusion, AMD APUs (Jaguar in PS4)
 - IBM Power 9
 - Intel hybrid CPU-FPGA
 - 2010 Stellarton: Atom E600C + Altera FPGA
 - 2017 Skylake CPU + Aria 10 FPGA
 - ARM (big.LITTLE)
 - HSA architectures

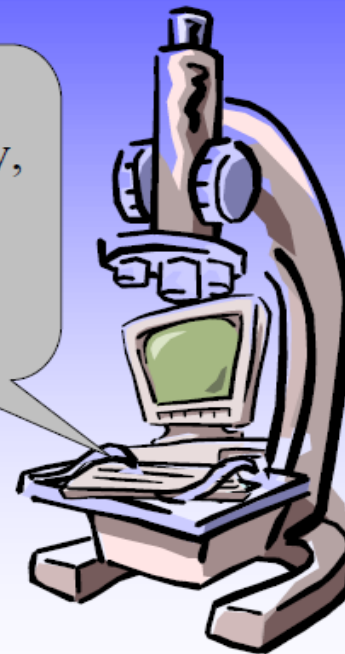
Complex,
code-heavy,
high-end
digital
system



Computer Architecture

- Ιεραρχία μνήμης
 - Multiple levels
 - Cache sharing
 - NUMA
 - Coherence Protocols
- Παράλληλα συστήματα
 - Coherence, Δίκτυα διασύνδεσης
 - Compilers, automatic parallelization
 - Programming Models, synchronization costs, locks, computation and communication overheads

Complex,
code-heavy,
high-end
digital
system



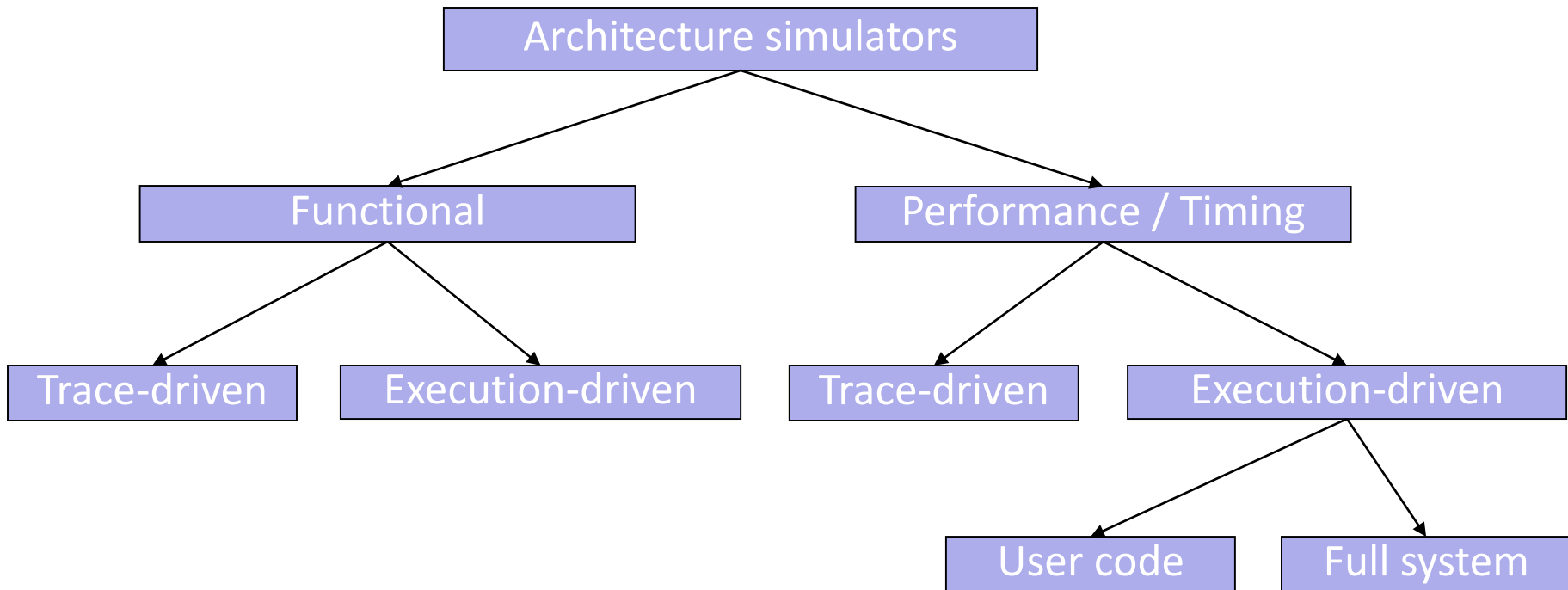
Τρόποι Υλοποίησης

- Χρήση υπαρχόντων μηχανημάτων
 - Μεγάλο κόστος
 - » Oracle SPARC Enterprise T5120 server (2*16*8=256 threads, 512GB mem) ~ 64K \$ (2014)
 - » Oracle SPARC Enterprise T8-1 server (32*8=256 threads, 1TB mem, 8 Data Analytics Accelerators) ~ ??K \$ (2017)
 - » Intel Sandy Bridge E5-4620 (4 * 8 cores, 64 threads) ~ 15K \$ (2013)
 - » Intel Xeon Gold 6138 (2 * 20 cores, 80 threads, 768GB mem) ~ 19K \$ (2018)
 - Αδυναμία παρέμβασης στο υλικό τους
 - » pipeline, caches, interconnection network
 - Περιορισμένη δυνατότητα παρακολούθησης και μετρήσεων
 - » performance counters → λίγοι, μικρό documentation
 - Περιορισμός στο σήμερα
 - » μελλοντικές αρχιτεκτονικές (π.χ. chip με 100 ή 1000 threads) ;
- Λύση : **Simulation** (προσομοίωση)

Τι είναι ο simulator;

- Ένα εργαλείο που αναπαράγει τη «συμπεριφορά» ενός υπολογιστικού συστήματος.
- Γιατί να χρησιμοποιήσουμε ένα simulator;
 - Πληροφορίες σχετικά με την εσωτερική λειτουργία
 - » Performance Analysis
 - Δυνατότητα ανάπτυξης λογισμικού για μη διαθέσιμες (ή και μη υπαρκτές) πλατφόρμες
 - Προβλέψεις απόδοσης για διαφορετικές αρχιτεκτονικές.
 - » Dream → Create → Simulate → Evaluate

Simulator taxonomy



Functional vs. Timing Simulators

- Functional Simulators

- Visible architectural state
- Προσομοίωση της λειτουργικότητας των εντολών (instructions semantics and functionality), μεταβολή του state (registers, memory)
- Σωστό program output
- Κύριος σκοπός: Software development and/or emulation

- Timing Simulators

- Microarchitecture details
- Λεπτομερής υλοποίηση των διαφορετικών δομών (pipeline, branch predictors, interconnection networks, memory hierarchy, etc.)
- Χρονισμός γεγονότων, προκειμένου να υπολογισθεί ο χρόνος εκτέλεσης του προγράμματος

- **Functional simulation πολύ πιο γρήγορο**

Trace vs. Execution-driven Simulators

- Trace Simulators
 - Εκτέλεση της εφαρμογής σε πραγματική πλατφόρμα → trace (instruction, address, ...)
 - Τα traces χρησιμοποιούνται σαν inputs του simulator
- Execution-driven Simulators
 - Ο simulator εκτελεί την εφαρμογή
 - Διατήρηση application state και architecture state
- Trace-driven simulation συνήθως πιο γρήγορο
 - Διατήρηση μόνο του architecture state, δεν εκτελούνται όλοι οι υπολογισμοί
- Τα traces επιτρέπουν την προσομοίωση proprietary applications & input sets.
- **Πρόβλημα**: Τα traces δεν μπορούν να συλλάβουν/φανερώσουν την πλήρη δυναμική συμπεριφορά της εφαρμογής

User code vs. Full system simulators

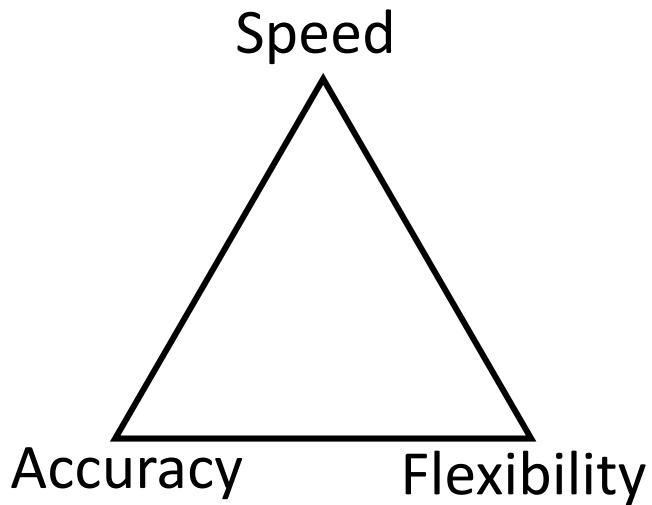
- User code Simulators

- Προσομοίωση μόνο του κώδικα της εφαρμογής
- System calls και I/O εκτελούνται με functional simulation
- Συνήθως το functional emulation πραγματοποιείται από το host OS
 - » host OS = target OS

- Full system Simulators

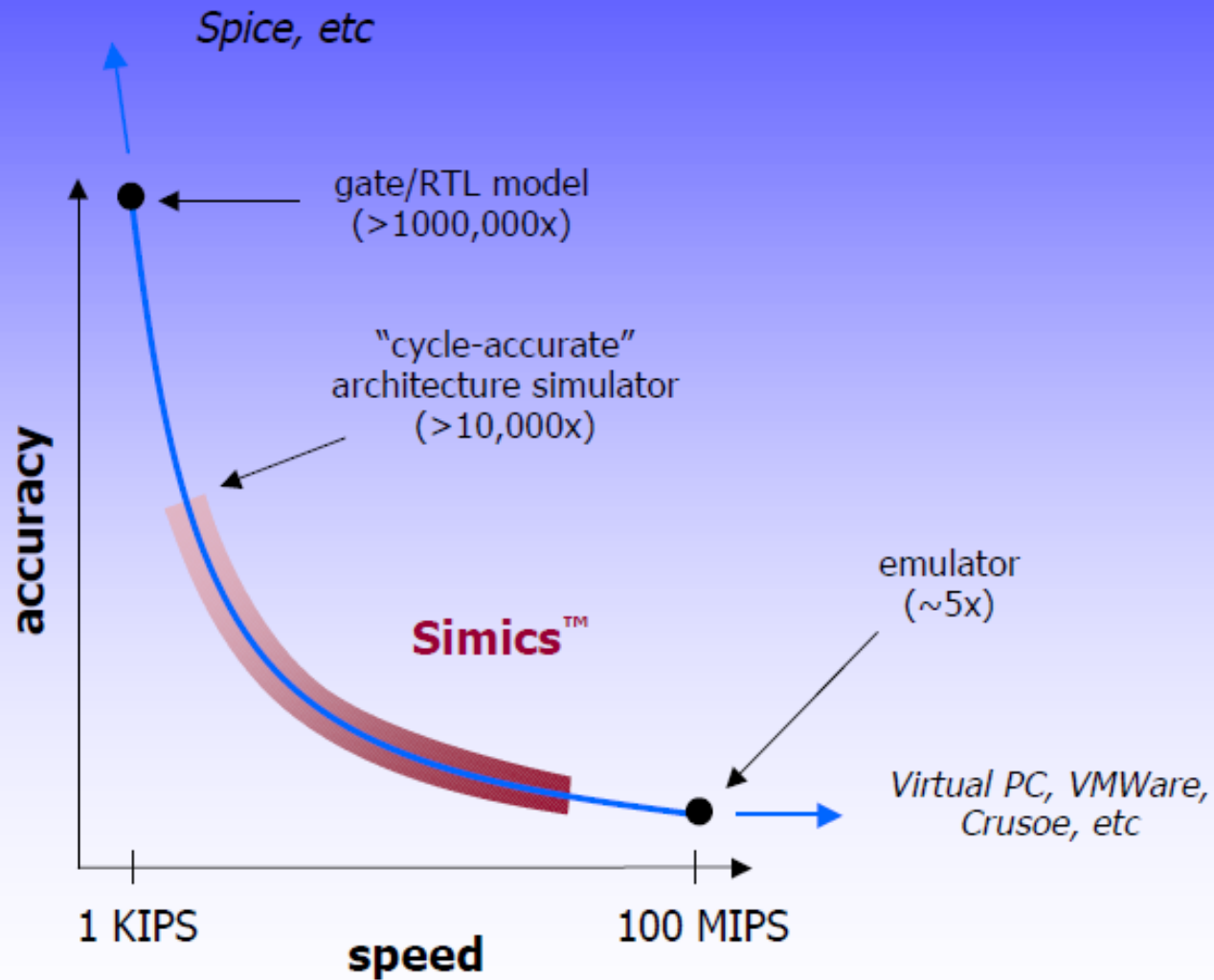
- Προσομοίωση της εφαρμογής
- Προσομοίωση του OS
- Προσομοίωση των devices (disks, network, etc.)

The Zen of architecture simulators



- Δεν υπάρχει ο τέλειος simulator
- FPGA prototypes
 - Γρήγορα, ακριβή, αλλά έλλειψη ευελιξίας
- Λεπτομερή software μοντέλα
 - Ακριβή, ευέλικτα, αλλά αργά
- Αφηρημένα software μοντέλα
 - Γρήγορα, ευέλικτα αλλά όχι ακριβή

Speed vs. Accuracy



Παράδειγμα χρόνων προσομοίωσης (1)

- spec2k with gcc and small inputs

	Time	Ratio to Native	Ratio to Functional
Native	1,054s	--	--
sim-fast	2m 47s	158x	1
sim-outorder	1h 11m 07s	4,029x	25x
simics	7m 41s	437x	1
w/Ruby	11h 27m 25s	39,131x	89x
w/Ruby + Opal	43h 13m 41s	147,648x	338x

Παράδειγμα χρόνων προσομοίωσης (2)

- Οι πραγματικές εφαρμογές παίρνουν ώρες σε πραγματικά μηχανήματα
- Χρειαζόμαστε αρκετά μεγάλες ταχύτητες για να τρέξουμε ένα σημαντικό κομμάτι αυτών των εφαρμογών.

	Number of Ops (B)
Windows XP Boot	5
Linux RH 6.0 Boot	4
Windows XP Install	361
SPECint2000 (train)	279

Προσομοίωση Αρχιτεκτονικής (1)

- Απαιτήσεις
 - Γενικότητα (Generality)
 - » Μπορεί το εργαλείο να αναλύσει τα workloads?
 - » Parallel Systems, Multithreading, Multiple address spaces, OS code, Network Systems, etc.
 - Πρακτικότητα (Practicality)
 - » Μπορεί το εργαλείο να χρησιμοποιηθεί αποδοτικά;
 - » Host assumptions, compiler assumptions, OS modifications, workload language assumptions
 - Εφαρμοσιμότητα (Applicability)
 - » Μπορεί το εργαλείο να απαντήσει στα ερωτήματα μας;
 - » Restricted state that can be monitored, restrictions on parameter visibility, restricted length of observations.

Προσομοίωση Αρχιτεκτονικής (2)

- Πλεονεκτήματα
 - Early availability
 - Ευκολία χρήσης
 - » Πλήρης διαφάνεια και ευκολία παρακολούθησης και μετρήσεων
 - » Διαφορετικά επίπεδα λεπτομέρειας και ακρίβειας
 - Pipelines, caches, branch predictors, ...
 - Hardware devices (timer, drives, cards, ...)
 - » Έλεγχος καινοτόμων προτάσεων/ιδεών
 - Κόστος
 - » Open source (Free)
 - » Academic licenses (Free ή μικρό κόστος για support)

Προσομοίωση Αρχιτεκτονικής (3)

- Προκλήσεις
 - Χρόνος ανάπτυξης των μοντέλων (modeling time)
 - Έλεγχος ορθότητας μοντέλων (validation)
 - Ταχύτητα προσομοίωσης
- Ενεργό ερευνητικό πεδίο
- Πληθώρα επιλογών
 - WindRiver Simics (x86, SPARC, MIPS, Leon, ARM...)
 - AMD SimNow (x86)
 - SimpleScalar (Alpha, PISA, ARM, x86)
 - SimFlex
 - MARSSx86
 - SniperSim
 - Gems5

Επιλογή περιβάλλοντος προσομοίωσης

- Κριτήρια Επιλογής
 - Modularity simulator
 - Extensibility simulator
 - Επίπεδο ακρίβειας simulator
 - Ταχύτητα simulator
 - Μέγεθος του design space που θέλουμε να μελετήσουμε
 - Επιλογή κατάλληλων benchmarks

Αποτελέσματα Προσομοίωσης(1)

- Ο σκοπός ενός timing simulation είναι η συγκέντρωση πληροφοριών και μέτρηση διαφόρων μεγεθών
 - IPC
 - Memory access cycles
 - On-chip network contention
- Τα προγράμματα παρουσιάζουν διαφορετικές φάσεις
 - Initialization phase
 - Main phase
 - Wrap-up phase
- Πότε παίρνουμε τα στατιστικά που μας ενδιαφέρουν;

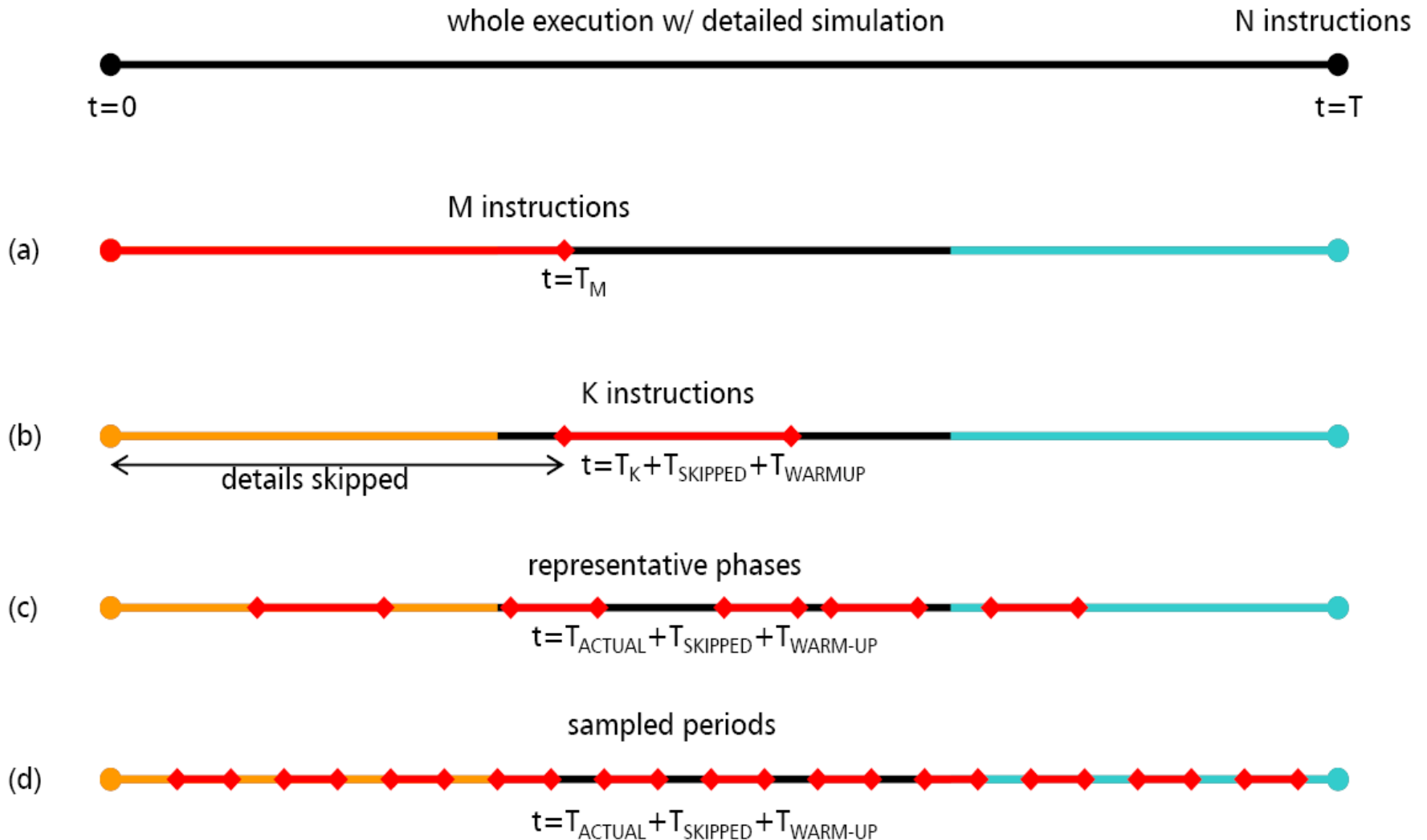
Αποτελέσματα Προσομοίωσης(2)

- Η προσομοίωση είναι ένα single-thread process
 - Ακόμα και αν προσομοιώνουμε ένα παράλληλο σύστημα
- Η ταχύτητα προσομοίωσης δε βελτιώνεται πια με την τεχνολογία
 - Παράλληλα συστήματα τρέχουν πολλαπλές προσομοιώσεις ταυτόχρονα.
- Δεν μπορούμε να φτιάξουμε γρήγορους simulators
 - Δεν προσομοιώνουμε ολόκληρο το σύστημα
 - Δεν προσομοιώνουμε ολόκληρη την εφαρμογή

Αποτελέσματα Προσομοίωσης(3)

- Προσομοίωση μικρότερου συστήματος
 - π.χ. 16 cores αντί για 1024
 - Δεν εκθέτει θέματα κλιμακωσιμότητας (scalability)
 - » Ανταγωνισμός για shared resources
 - » Conflicts
 - » Race conditions
- Προσομοίωση μικρότερης εφαρμογής
 - π.χ. Matrix multiply 1K x 1K αντί για 1M x 1M
 - Δεν εξετάζει τα όρια του hardware
 - » Το working set χωράει στις caches
 - » Capacity/conflict issues
 - » Λιγότερη επαναχρησιμοποίηση data/code
 - » Initialization vs. steady state

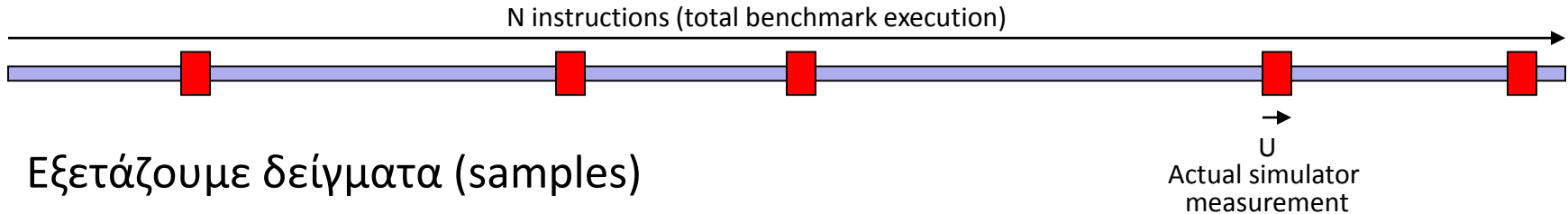
Τεχνικές Προσομοίωσης (1)



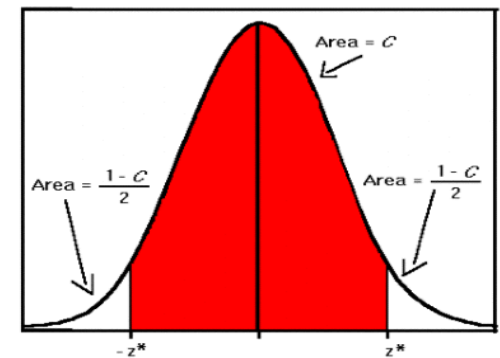
Τεχνικές Προσομοίωσης – Sampling (2)

- Sampling
 - Προσομοίωση τυχαίων σημείων της εφαρμογής
- Fast-forwarding
 - Μέχρι να φτάσουμε στο sample point
- Warm-up
 - Γρήγορη προσομοίωση πριν τη φάση των μετρήσεων
- Checkpointing
 - Αποθήκευση architectural state πριν το sample
- Phase detection
 - Επιλογή των samples μετά από ανάλυση της εφαρμογής

Τεχνικές Προσομοίωσης – Sampling (2)



- Εξετάζουμε δείγματα (samples)
- Μαθηματική προσέγγιση
 - confidence margin (eg. 95%)
 - confidence interval (eg. +/- 2.5)
- Δυο προσεγγίσεις σχετικά με την επιλογή δειγμάτων
 - Systematic sampling (π.χ. sample every N instructions)
 - Random sampling
 - Non-random sampling (phase detection, simpoints)
- Μεθοδολογία Προσομοίωσης:
 1. Fast-forwarding
 2. Warm-up (caches, branch predictors, TLBs, OS)
 3. Checkpointing
 4. Μέτρηση



Ενορχήστρωση - Instrumentation

- Εισαγωγή επιπλέον κώδικα στην εφαρμογή με στόχο την συλλογή πληροφοριών για την εκτέλεση (cache misses, total instructions executed etc.)
 - Ανάλυση προγράμματος: Performance profiling, error detection, capture & replay
 - Μελέτη Αρχιτεκτονικής: Processor & cache simulation, trace collection
 - Binary translation: Emulate unsupported instructions, modify program behavior

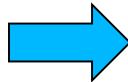
Τεχνικές Ενορχήστρωσης

- Source instrumentation
 - Εισαγωγή εντολών στον πηγαίο κώδικα της εφαρμογής
- Binary instrumentation
 - Εισαγωγή εντολών στο εκτελέσιμο της εφαρμογής
 1. Static binary instrumentation
 - » Εισαγωγή των εντολών **πριν** την εκτέλεση
 2. Dynamic binary instrumentation
 - » Εισαγωγή των εντολών **δυναμικά κατά την εκτέλεση**

Source Instrumentation – Παράδειγμα (1)

Αρχικό πρόγραμμα

```
void foo() {  
    bool found = false;  
    for (int i=0; i<100; ++i) {  
        if (i==50) break;  
        if (i==20) found=true;  
    }  
  
    printf("foo\n");  
}
```



Instrumented

```
char inst[5];  
void foo() {  
    bool found = false;inst[0]=1;  
    for (int i=0; i<100; ++i) {  
        if (i==50) {inst[1]=1;}break;  
        if (i==20) {inst[2]=1;found=true;  
        inst[3]=1;  
    }  
  
    printf("foo\n");  
    inst[4]=1;  
}
```

Binary Instrumentation – Παράδειγμα (2)

Total Instructions counter

```
counter++;  
Sub    $0xff, %edx  
counter++;  
cmp    %esi, %edx  
counter++;  
jle    <L1>  
counter++;  
mov    $0x1, %edi  
counter++;  
add    $0x10, %eax
```

Binary Instrumentation – Παράδειγμα (3)

Instruction Trace

```
Print(ip);  
sub    $0xff, %edx  
Print(ip);  
cmp    %esi, %edx  
Print(ip);  
jle    <L1>  
Print(ip);  
mov    $0x1, %edi  
Print(ip);  
add    $0x10, %eax
```

Πλεονεκτήματα Ενορχήστρωσης

- Binary Instrumentation
 - Language independent
 - Machine-level view
 - Instrument legacy/proprietary software
- Dynamic Instrumentation
 - Δεν χρειάζεται recompile/relink
 - Εντοπισμός κώδικα κατά την εκτέλεση
 - Δυνατότητα χειρισμού κώδικα που παράγεται δυναμικά
 - Δυνατότητα ενορχήστρωσης running processes

Intel's PIN

- Εργαλείο για dynamic binary instrumentation.
- Αναπτύσσεται και συντηρείται από την Intel.
- Περισσότερες πληροφορίες:



<https://software.intel.com/en-us/articles/pin-a-dynamic-binary-instrumentation-tool>

- Χαρακτηριστικά/Πλεονεκτήματα
 - Εύκολη εντοπιστική
 - Programmable (APIs)
 - Multiplatform (x86, x86-64, Windows, Linux, OSX, Android)
 - Robust
 - » Real-life apps: Database, web browseres, ...
 - » Multithreaded apps
 - » Signals
 - Υψηλής Απόδοσης (compiler optimizations on instrumentation code)
 - Ευρεία αποδοχή από την ακαδημαϊκή κοινότητα και την βιομηχανία

PIN – Χρήση

Download latest version:

<https://software.intel.com/en-us/articles/pintool-downloads>

Unzip:

```
$ tar xvfz pin-2.14-71313-gcc.4.4.7-linux.tar.gz
```

```
$ cd pin-2.14-71313-gcc.4.4.7-linux
```

```
$ ls -aF
```

```
./ ../ doc/ extras/ ia32/ intel64/ LICENSE pin pin.sh README redist.txt  
source/
```

Execute:

```
$ ./pin.sh -t pintool - application
```

```
ή
```

```
$ ./pin.sh -t pintool -pid 1243
```

PIN – Παράδειγμα

```
$ ./pin.sh -t inscount0.so -- /bin/ls
```



Total instructions count pinctool

```
$ cd source/tools/ManualExamples/  
$ make inscount0.so  
$ ls obj-intel64/inscount0.so
```


Pintools

- Προγράμματα γραμμένα σε C++
- Χρησιμοποιούνται κατά την εκτέλεση του PIN για την ανάλυση της εφαρμογής
- Χρησιμοποιούν το PIN API:
 - » `INS_AddInstrumentFunction(...);`
 - » `INS_InsertCall(...);`
 - » ...
- PIN API reference:
 - https://software.intel.com/sites/landingpage/pintool/docs/55942/Pin/html/group__API__REF.html

Pintool code

```
#include <iostream>
#include "pin.h"
```

```
UINT64 icount = 0;
```

```
void docount() {
    icount++;
}
```

analysis routine

```
void Instruction(INS ins, void *v) {
    INS_InsertCall(ins, IPOINT_BEFORE, (AFUNPTR) docount, IARG_END);
}
```

instrumentation routine

```
void Fini(INT32 code, void *v) {
    std::cerr << "Count " << icount << "\n";
}
```

end routine

```
int main(int argc, char *argv[]) {
    PIN_Init(argc, argv);
    INS_AddInstrumentFunction(Instruction, 0);
    PIN_AddFiniFunction(Fini, 0);
    PIN_StartProgram();
    return 0;
}
```

Δυνατότητες Pin & Pintools

- Αντικατάσταση των application functions.
- Εντοπισμός/καταγραφή/έλεγχος κάθε application instruction.
- Πέρασμα παραμέτρων στο instrumentation routine
 - register values, register values by reference (for modification)
 - memory addresses read/written
 - full register context
- Εντοπισμός syscalls και αλλαγή arguments
- Εντοπισμός/έλεγχος application threads
- ...
- ...