Synthetic Traffic Generation:
a Tool for Dynamic Interconnect Evaluation

W. Heirman, J. Dambre, J. Van Campenhout
ELIS Department, Ghent University, Belgium

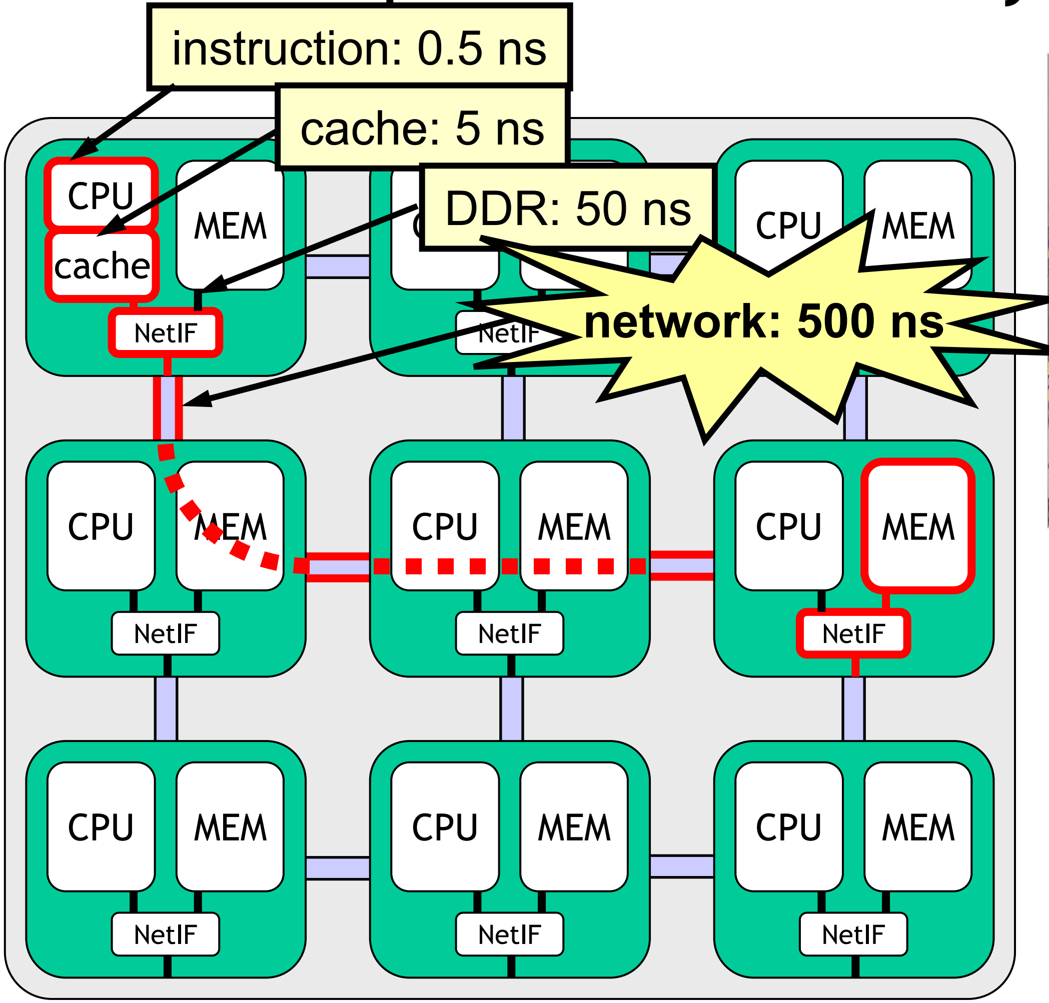
Sponsored by IAP-V *PHOTON* & IAP-VI *photonics@be*,
Belgian Science Policy Office

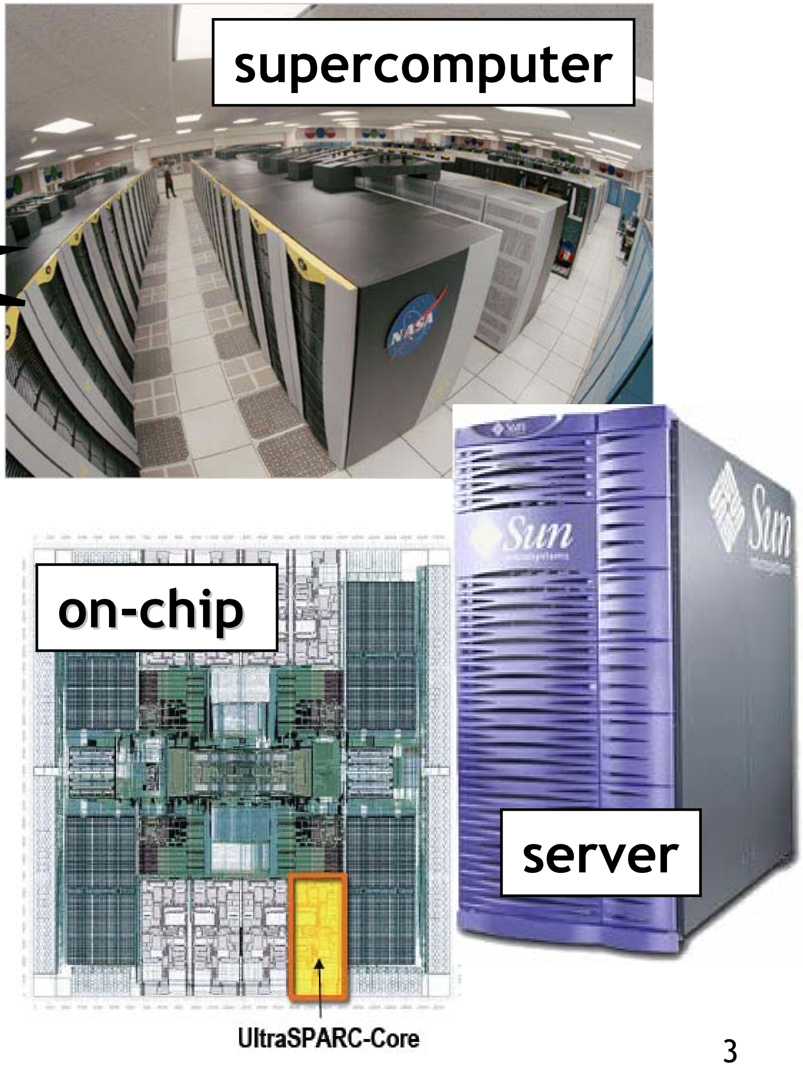
Outline

- Introduction
- Synthetic traffic generation
- Results
- Conclusions

Distributed shared-memory architecture

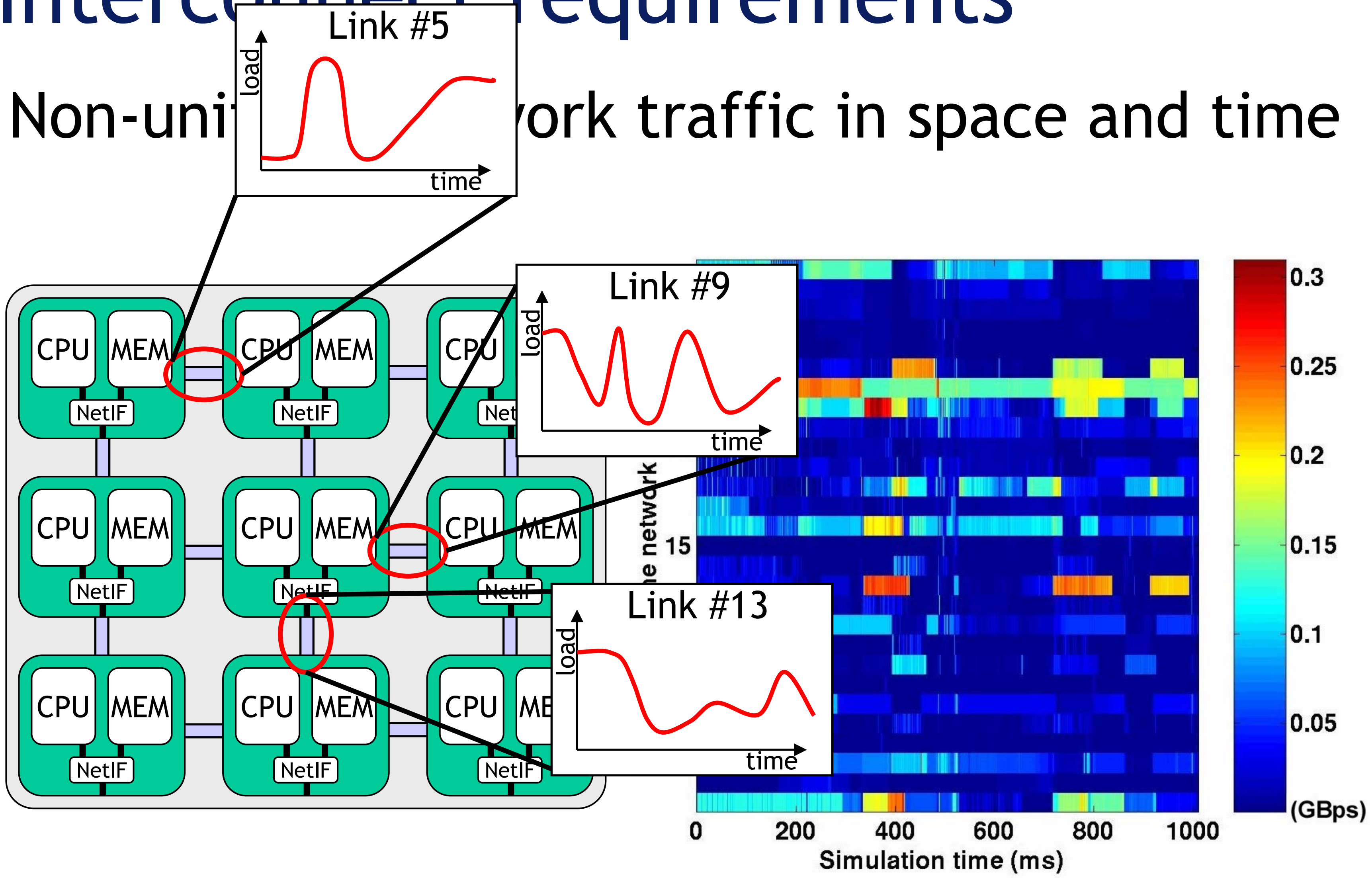
Network is part of the memory hierarchy





Interconnect requirements

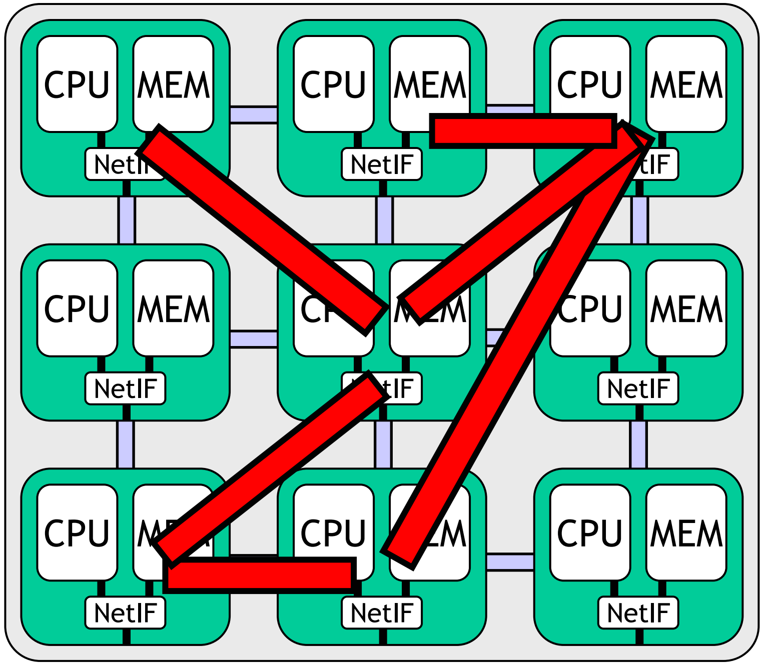
Non-uni ork traffic in space and time



=> Reconfigurable network?

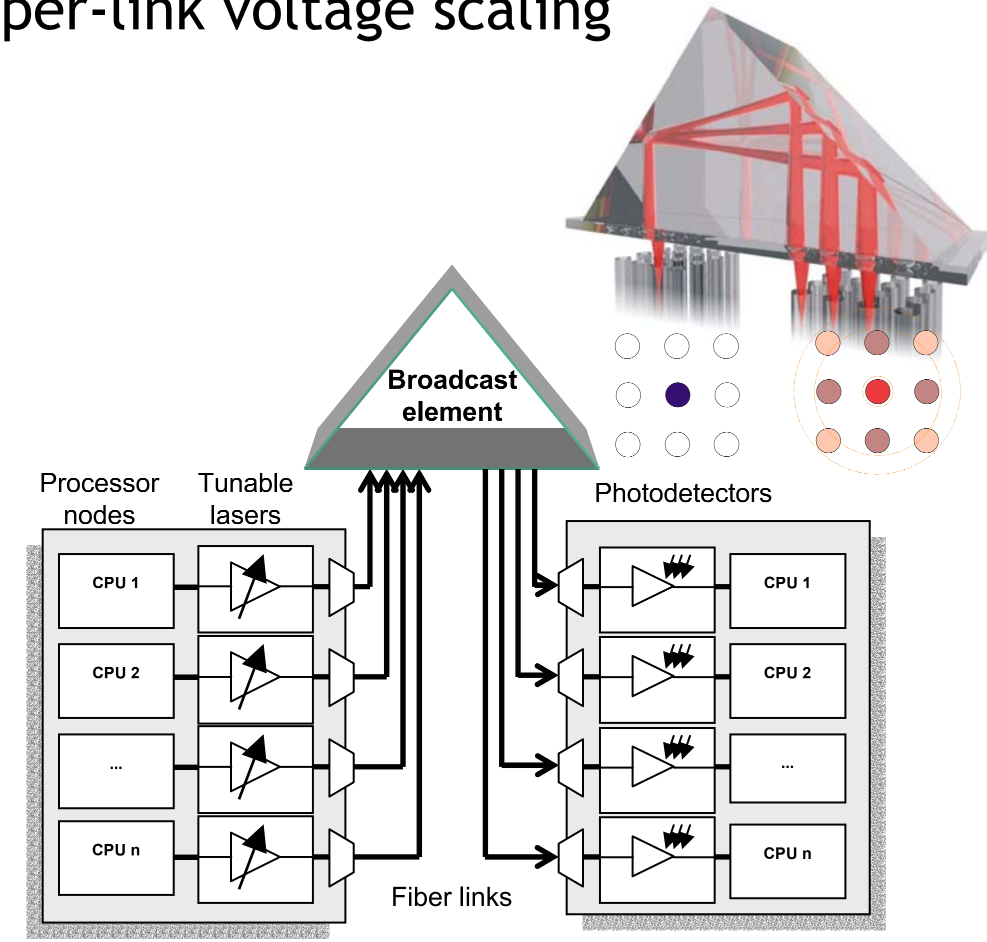
Reconfiguration implementation: base network + extra reconfigurable links

other ‘dynamic networks’: e.g. per-link voltage scaling



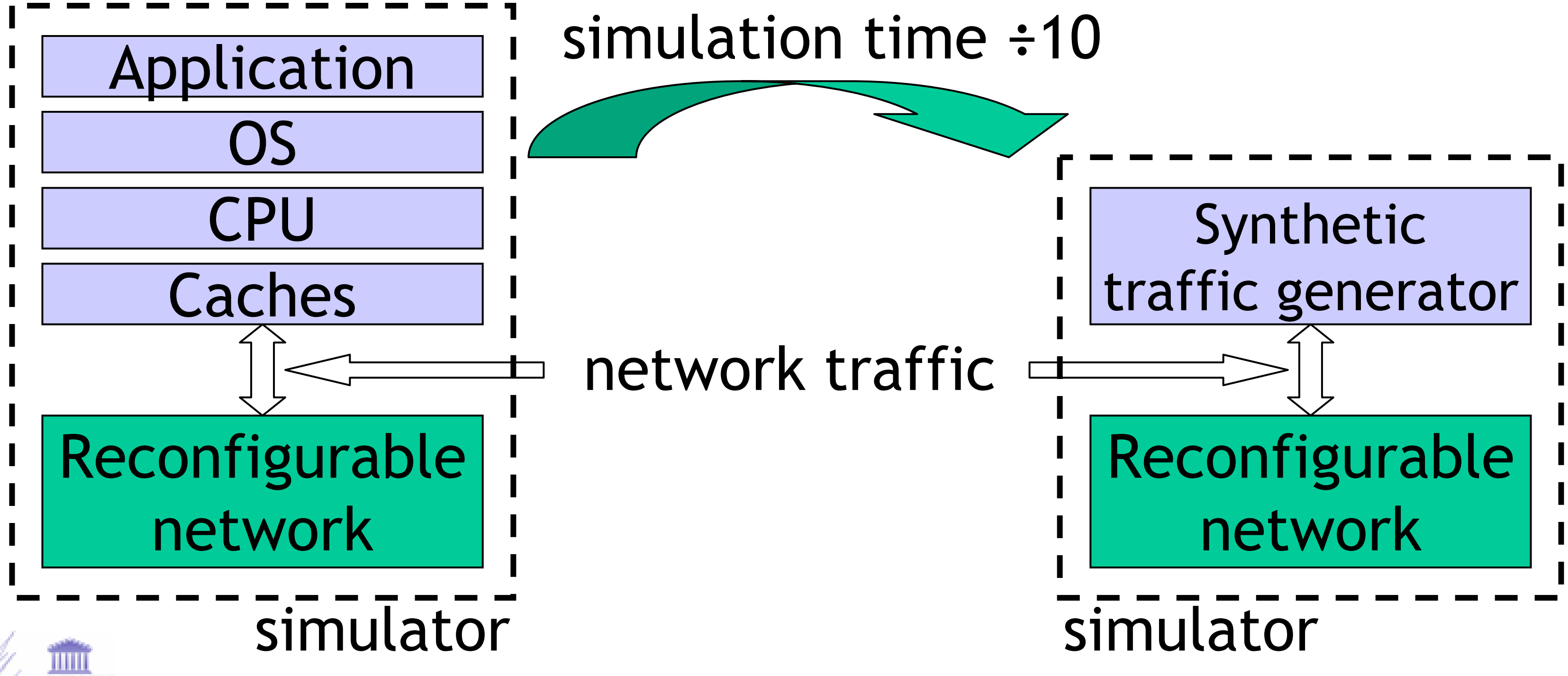
▭ Base network (fixed)

🟥 Extra links (reconfigurable)



Evaluate networks with synthetic traffic

- Mimics the behavior of real traffic
- But without the computational cost of modeling application, OS, CPUs, caches, …



We need better synthetic traffic

Reconfiguration exploits low-frequency dynamics in the network traffic

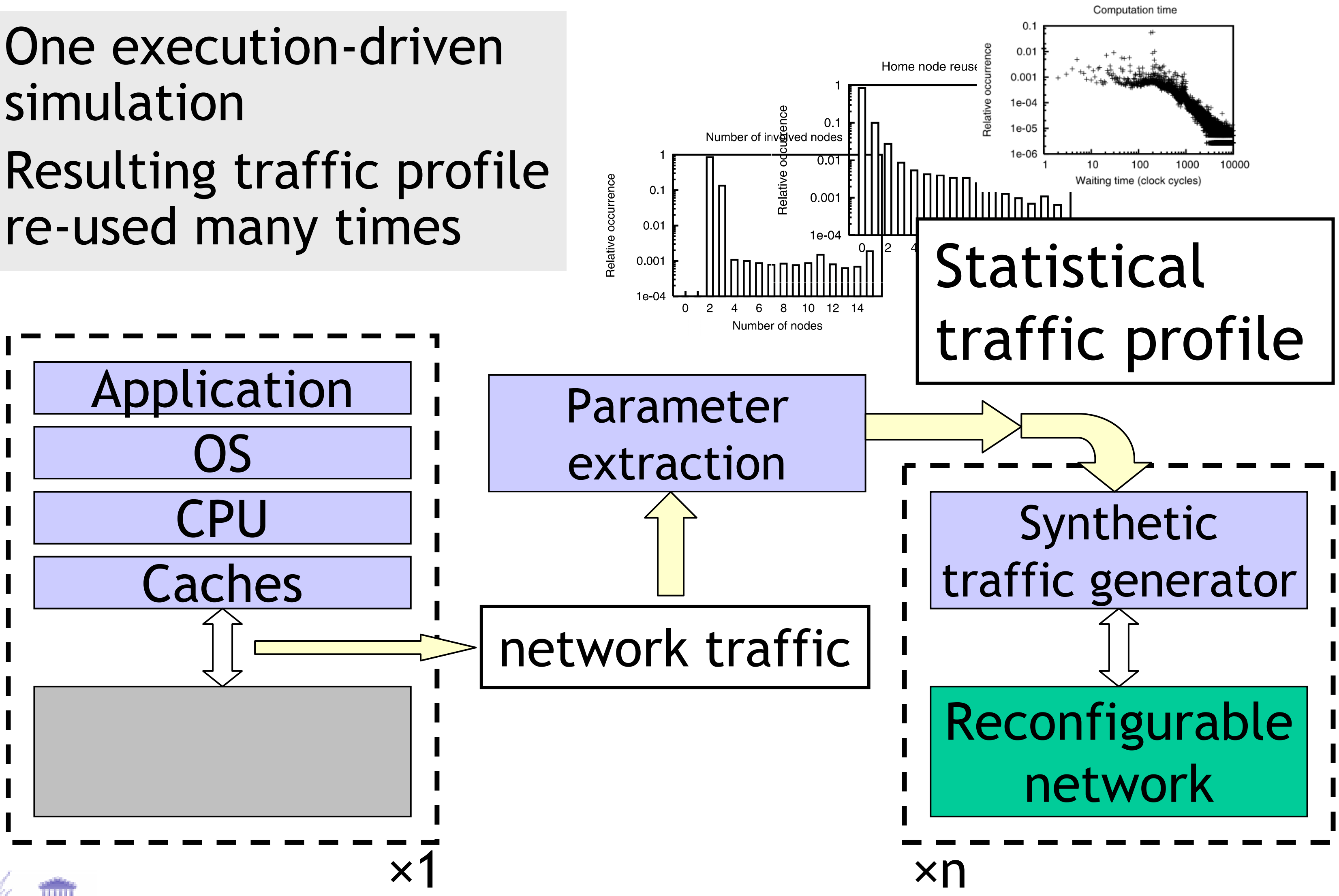
- Trace-driven simulation using static traffic patterns (uniform, hotspot, shuffle, …) won’t do!
- Full execution-driven simulation (traffic is driven by application: FFT, weather forecast, database) is too slow!

Outline

- Introduction
- Synthetic traffic generation
- Results
- Conclusions

Realistic synthetic traffic generation

- One execution-driven simulation
- Resulting traffic profile re-used many times



Statistical traffic profile

Application

OS

CPU

Caches

network traffic

Parameter extraction

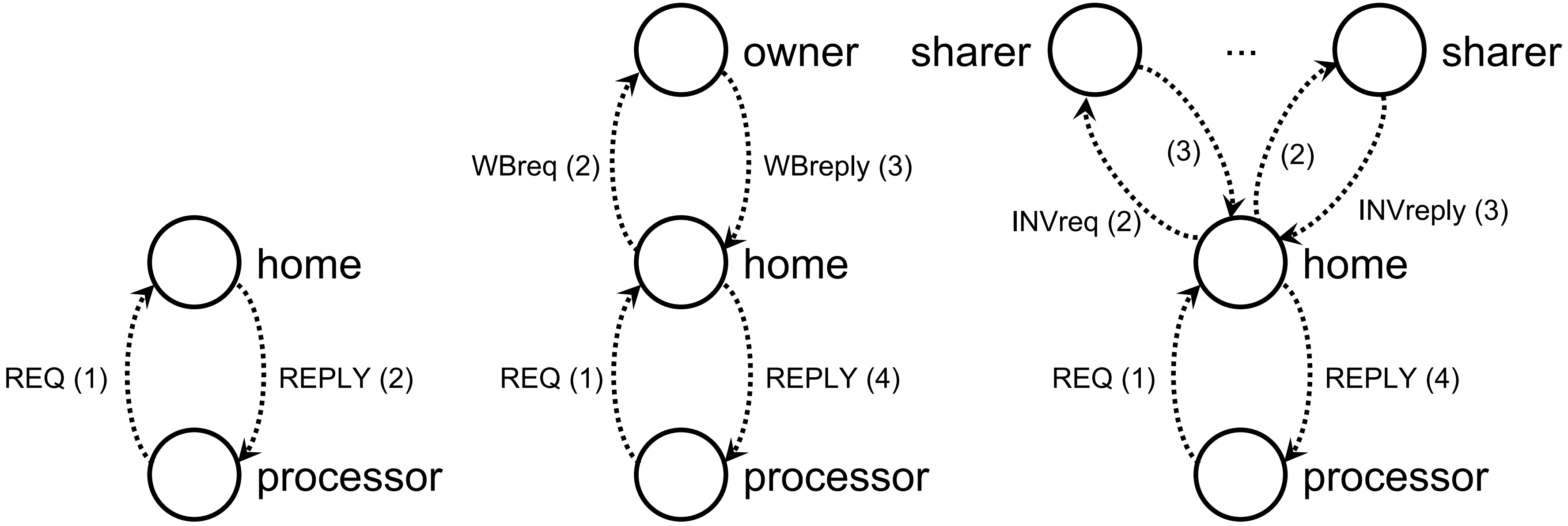
Synthetic traffic generator

Reconfigurable network

×1

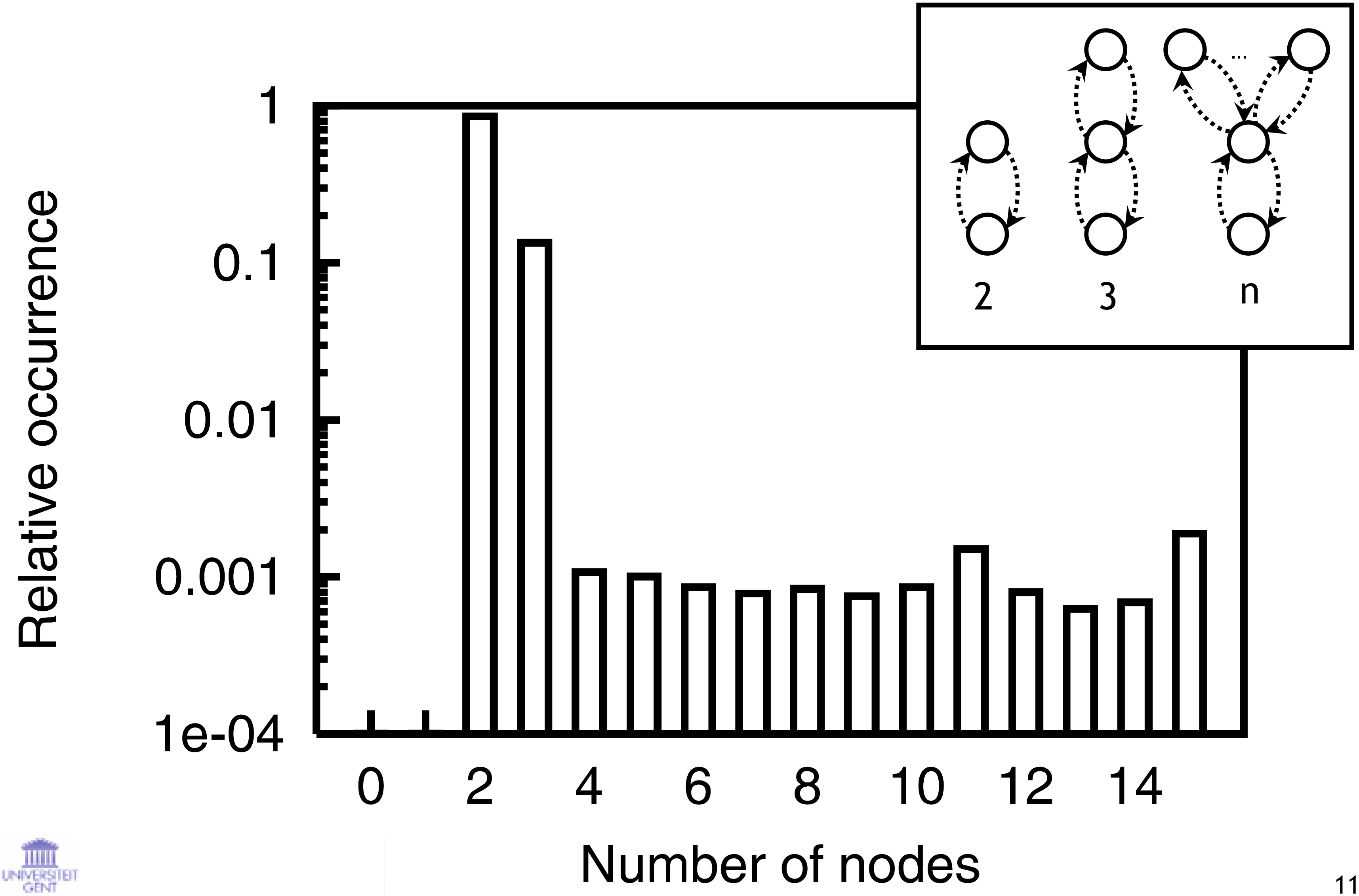
×n

Preserve packet-interdependencies by using packet groups

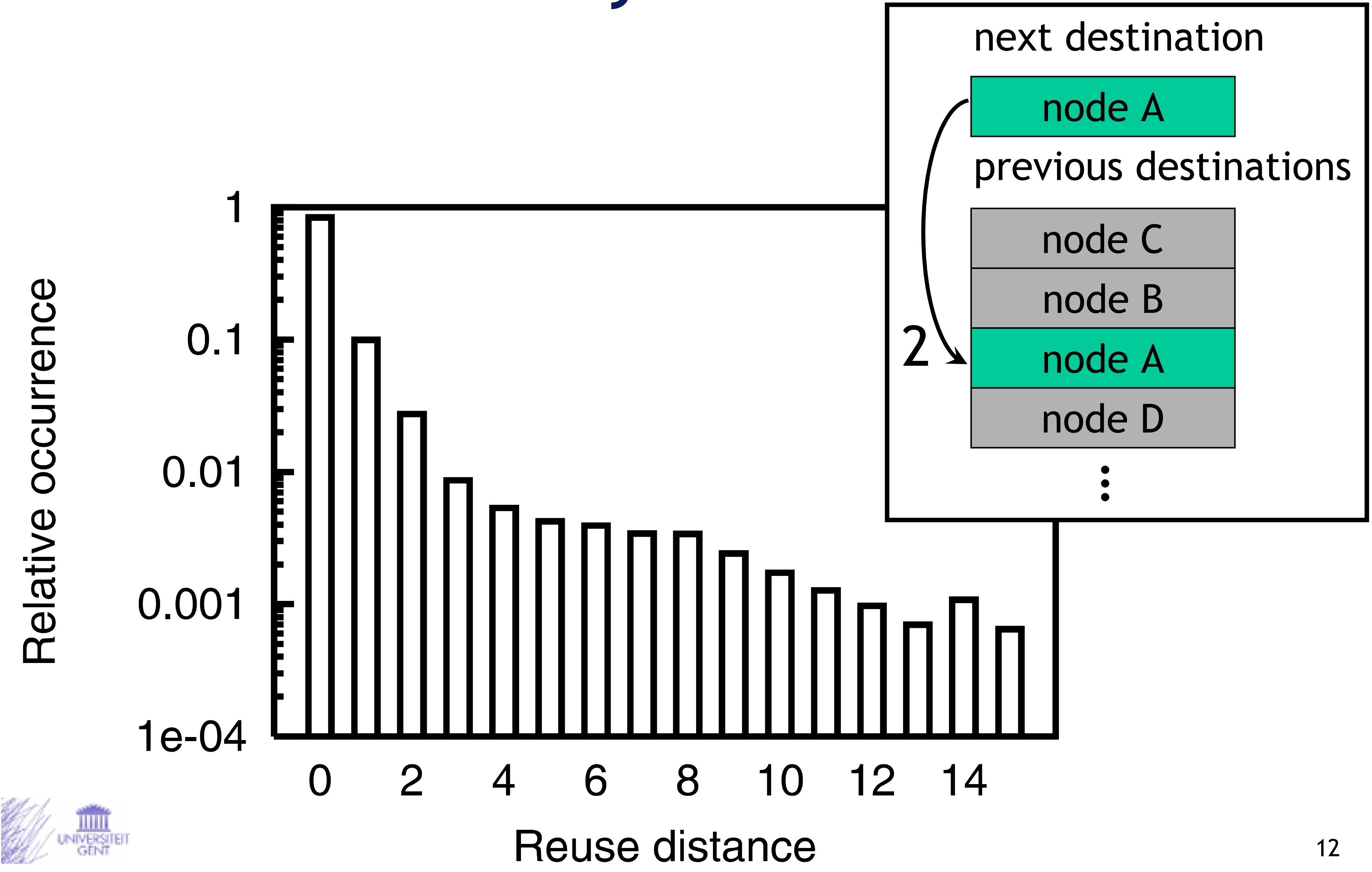


Packets are processed/generated in groups, corresponding to one memory operation each

Distribution of # involved nodes



Reuse distance of home nodes: introduce locality



next destination

node A

previous destinations

node C

node B

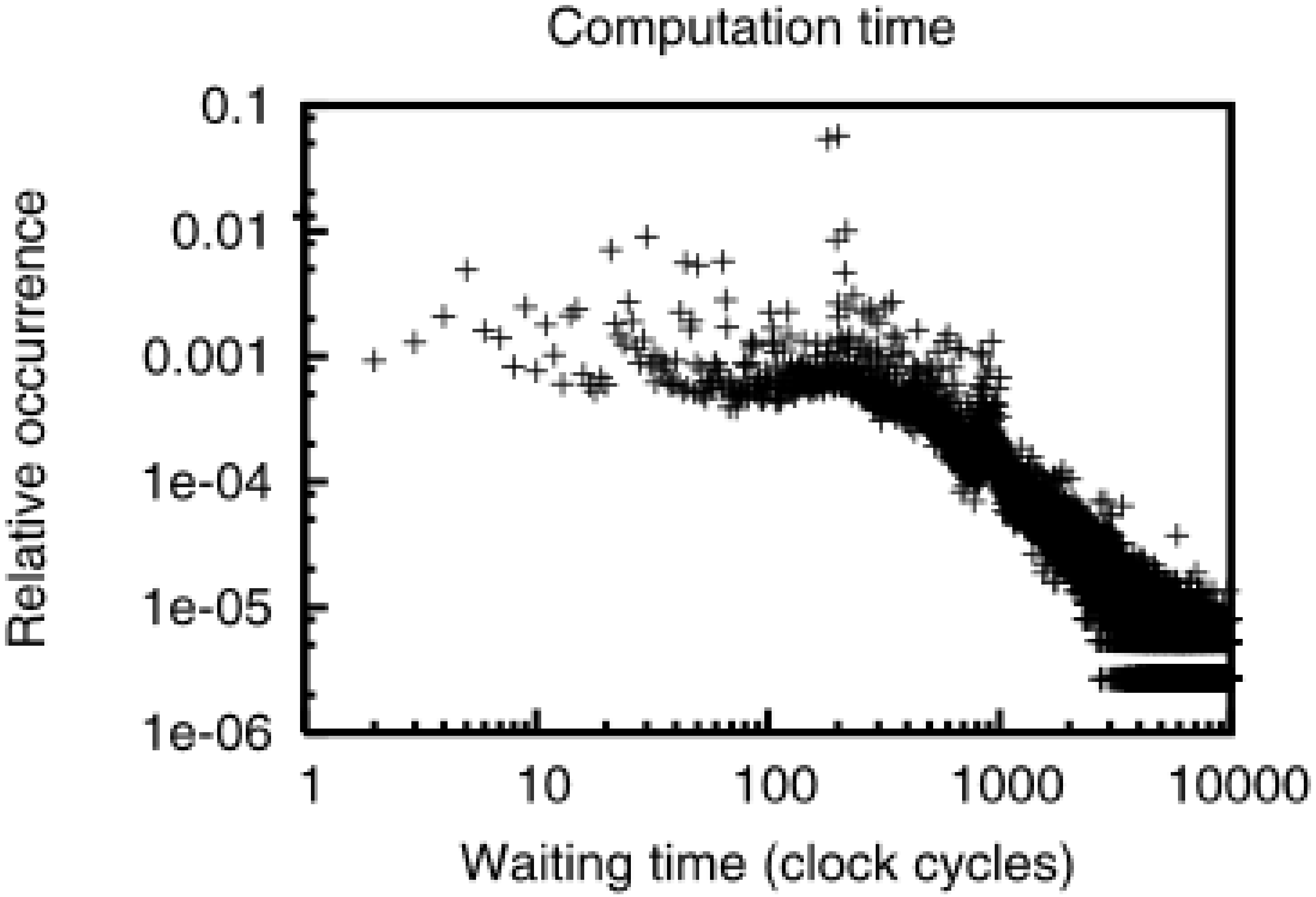
node A

node D

⋮

Computation or ‘think’ time

Models time delay between subsequent requests

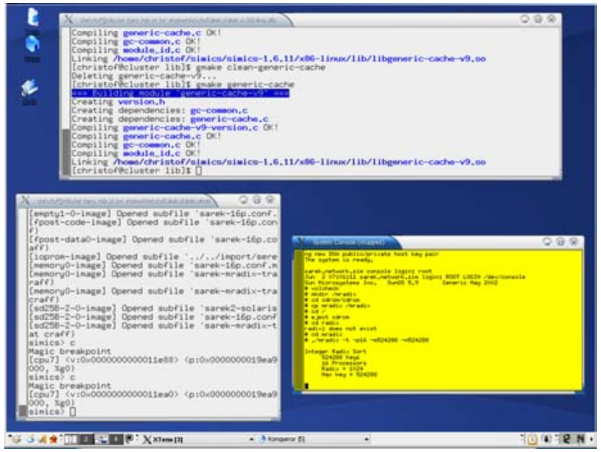


Outline

- Introduction
- Synthetic traffic generation
- Results
- Conclusions

Simulations

- Simulation platform: Simics, providing *functional* multiprocessor simulation
  - 16 UltraSPARC III processors
  - SPLASH-2 parallel benchmarks
- Timing model:
  - Computes the latency for each memory access
  - Models caches, interconnection network
  - Base network: 4x4 torus
  - Extra links: configurable number, fan-out, reconfiguration interval



```
Compiling generic-cache.c OK!
Compiling gc-common.c OK!
Compiling module_id.c OK!
Linking /home/christof/simics/simics-1.6.11/x86-linux/lib/libgeneric-cache-v9.so
[christof@cluster lib]$ gmake clean-generic-cache
Deleting generic-cache-v9...
[christof@cluster lib]$ gmake generic-cache
=== Building module "generic-cache-v9" ===
Creating version.h
Creating dependencies: gc-common.c
Creating dependencies: generic-cache.c
Compiling generic-cache-v9-version.c OK!
Compiling generic-cache.c OK!
Compiling gc-common.c OK!
Compiling module_id.c OK!
Linking /home/christof/simics/simics-1.6.11/x86-linux/lib/libgeneric-cache-v9.so
[christof@cluster lib]$ 
```

```
[empty1-0-image] Opened subfile 'sarek-16p.conf.
[fpost-code-image] Opened subfile 'sarek-16p.con
f)
[fpost-data0-image] Opened subfile 'sarek-16p.co
aff)
[ioprom-image] Opened subfile '../../import/sere
[memory0-image] Opened subfile 'sarek-16p.conf.m
[memory0-image] Opened subfile 'sarek-mradix-tra
raff)
[memory0-image] Opened subfile 'sarek-mradix-tra
craff)
[sd258-2-0-image] Opened subfile 'sarek2-solaris
[sd258-2-0-image] Opened subfile 'sarek-16p.conf
[sd258-2-0-image] Opened subfile 'sarek-mradix-t
at craff)
simics> c
Magic breakpoint
[cpu7] <v:0x0000000000011e88> <p:0x000000019ea9
000, %g0)
simics> c
Magic breakpoint
[cpu7] <v:0x0000000000011ea0> <p:0x000000019ea9
000, %g0)
simics> 
```

System Console (stopped)

```
ng new DSA public/private host key pair
The system is ready.

sarek.network.sim console login: root
Jun  2 07:55:11 sarek.network.sim login: ROOT LOGIN /dev/console
Sun Microsystems Inc.   SunOS 5.9       Generic May 2002
# volcheck
# mkdir /mradix
# cd cdrom/cdrom
# cp mradix /mradix
# cd /
# eject cdrom
# cd radix
radix: does not exist
# cd mradix
# ./mradix -t -p16 -n524288 -m524288

Integer Radix Sort
     524288 Keys
     16 Processors
     Radix = 1024
     Max key = 524288
```

Simulations

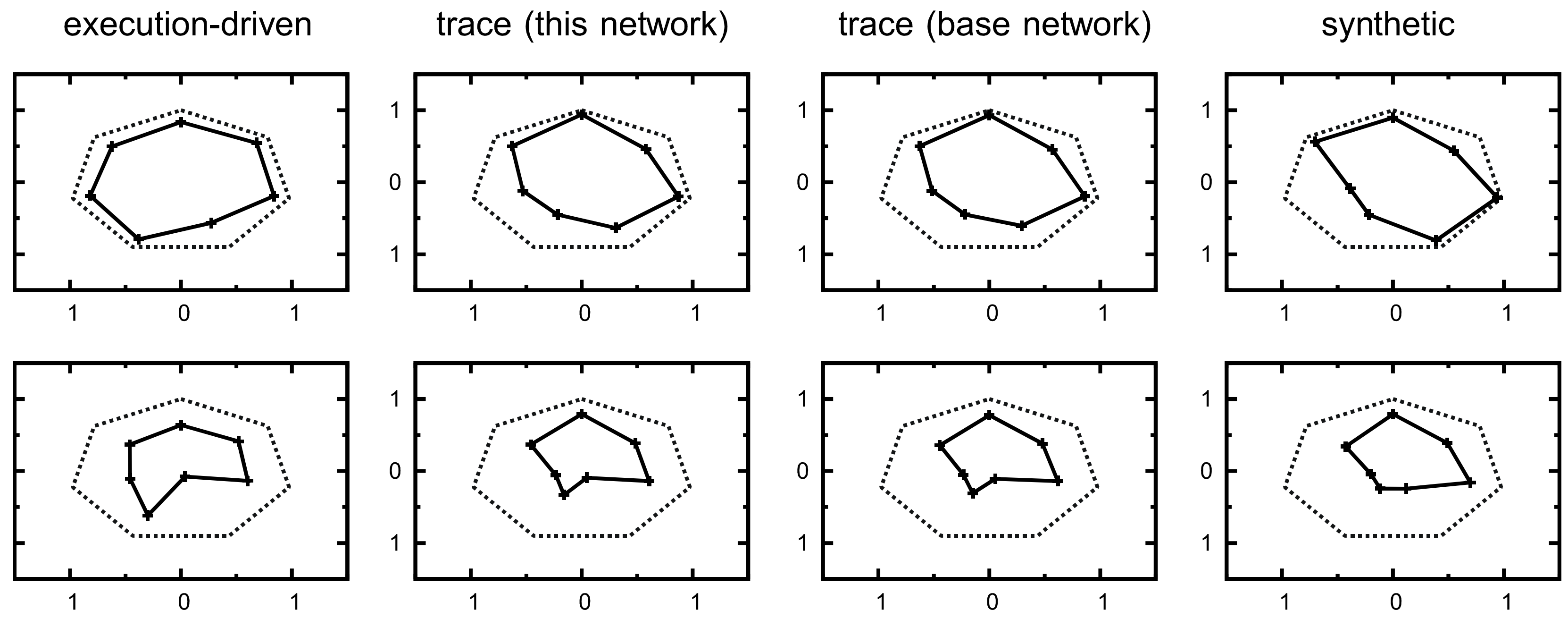
Once per benchmark:

Simulate execution of the benchmark, base network only, measuring traffic profile (1)

For each set of extra link parameters:

- Execution-driven simulation with reconfigurable network (2)
  ➔ *“correct” result*
- Trace-driven simulation using (simplified) traffic from (2)
  ➔ *tracing error*
- Trace-driven simulation using (simplified) traffic from (1)
  ➔ *traffic-dependence on network*
- Trace-driven simulation using synthetic traffic
  ➔ *total error*

Several parameters can be measured



execution-driven

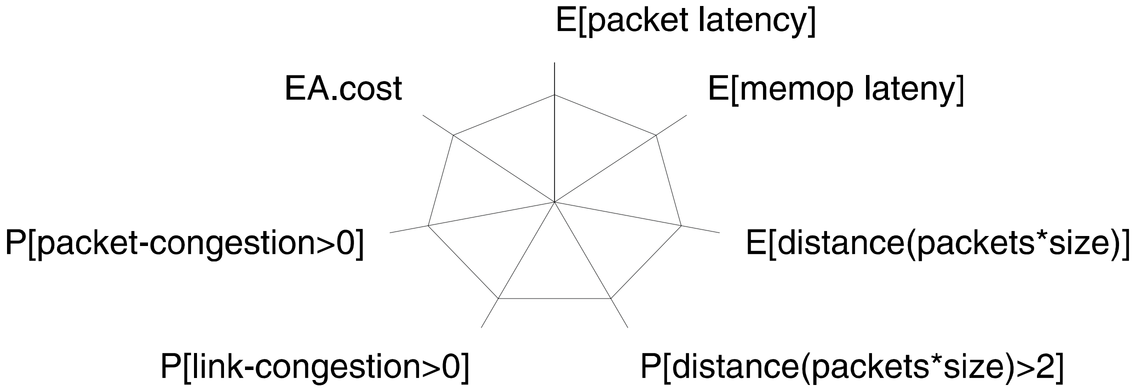
trace (this network)

trace (base network)

synthetic

n= 2
f = 2
Δt= 1 ms

n= 8
f = 2
Δt= 10„s



E[packet latency]

E[memop lateny]

E[distance(packets*size)]

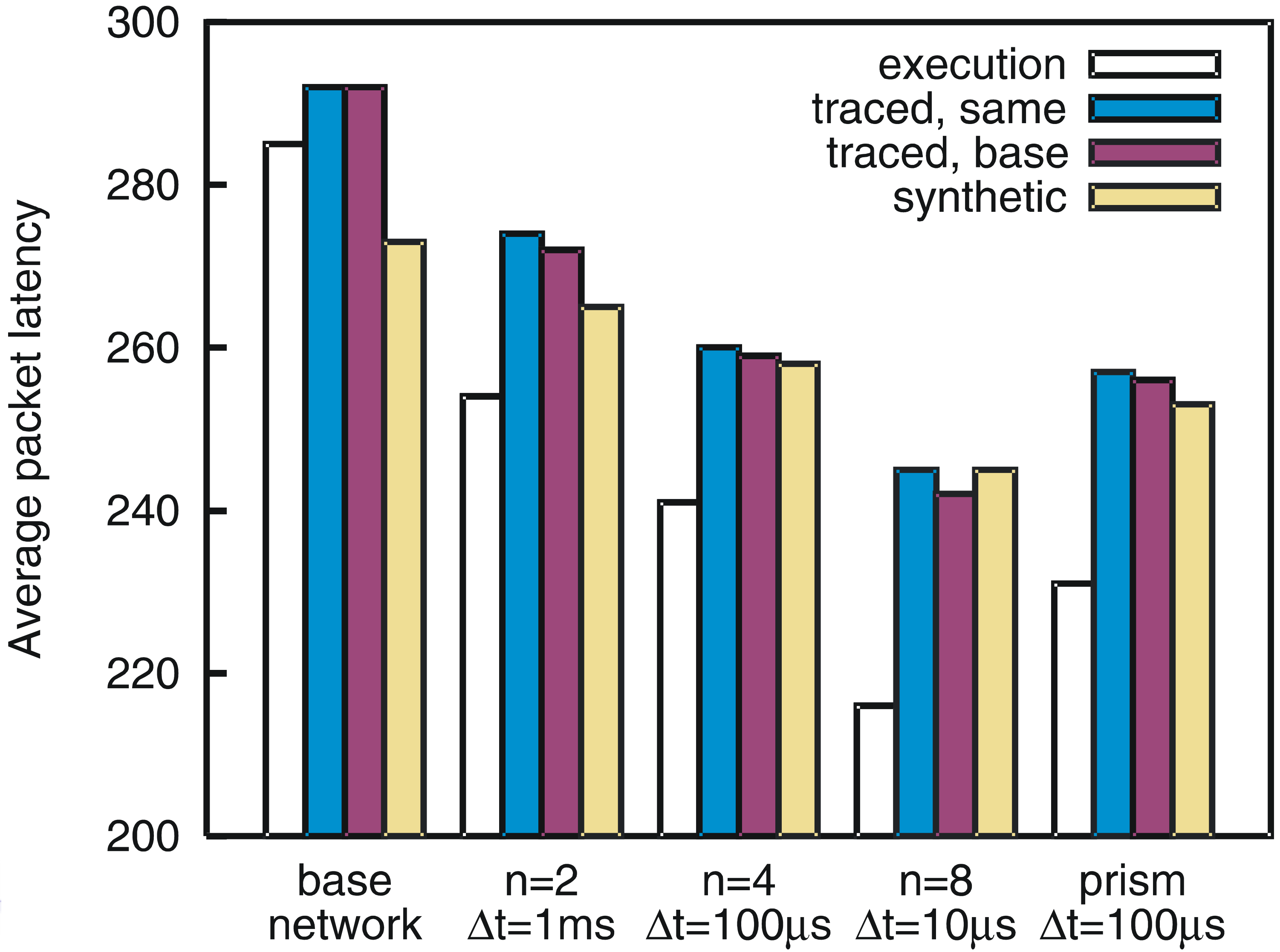
P[distance(packets*size)>2]

P[link-congestion>0]

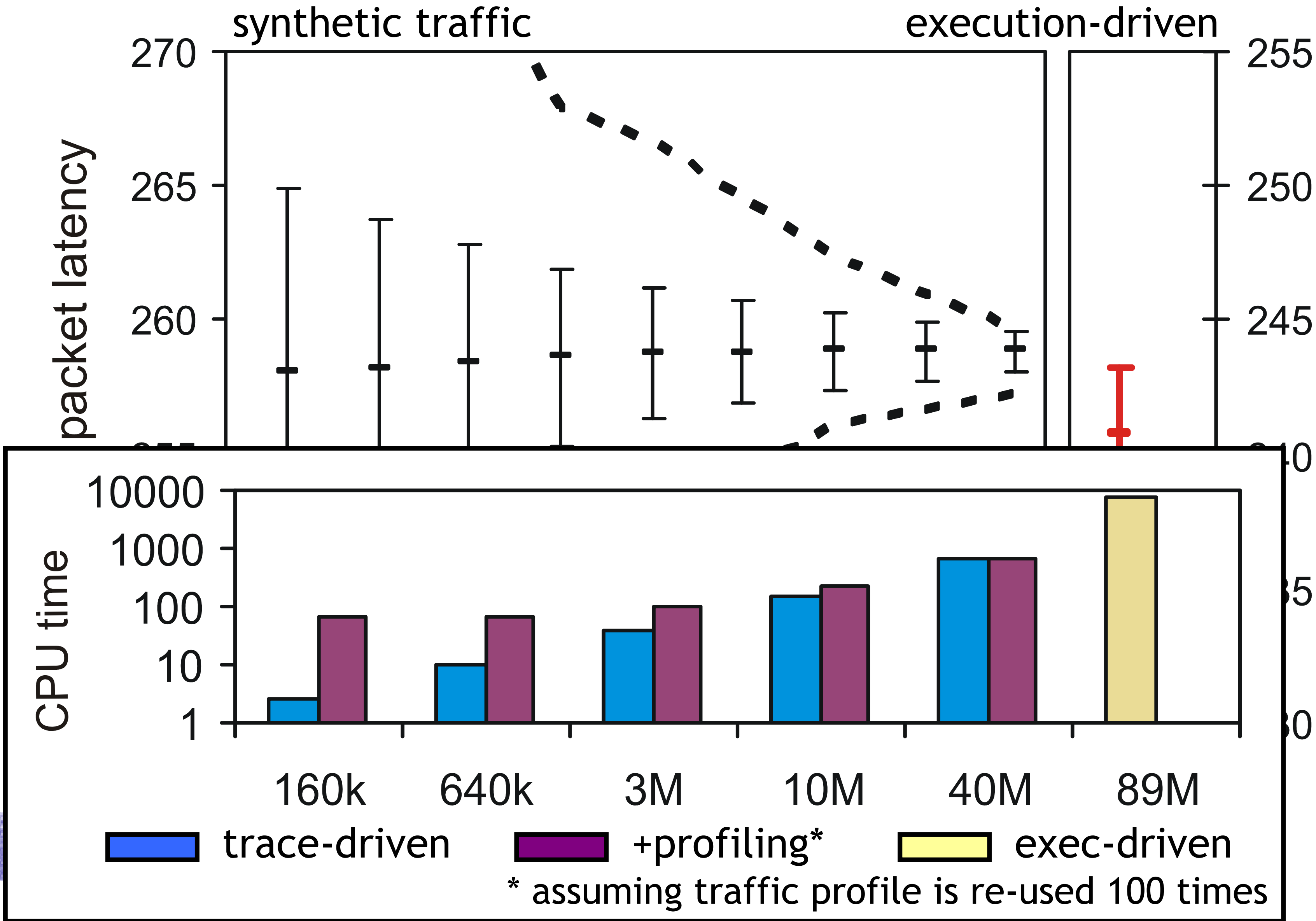
P[packet-congestion>0]

EA.cost

Detailed view of “average packet latency”



Variability for shorter traces



* assuming traffic profile is re-used 100 times

Outline

- Introduction
- Synthetic traffic generation
- Results
- Conclusions

Conclusions

- Synthetic traffic generation was extended to
  - shared-memory cache-coherence protocols,
  - reconfigurable networks
- Good relative accuracy for different network topologies
- Much less computationally expensive (x10), even more so for shorter traces (x100)
- Reproducibility equal to or better than execution-driven simulations