Adaptive Memory Protection for Many-Core Systems

Application-centric operating-system design

Gabor Drescher, Wolfgang Schröder-Preikschat
Friedrich-Alexander-Universität Erlangen-Nürnberg (FAU)

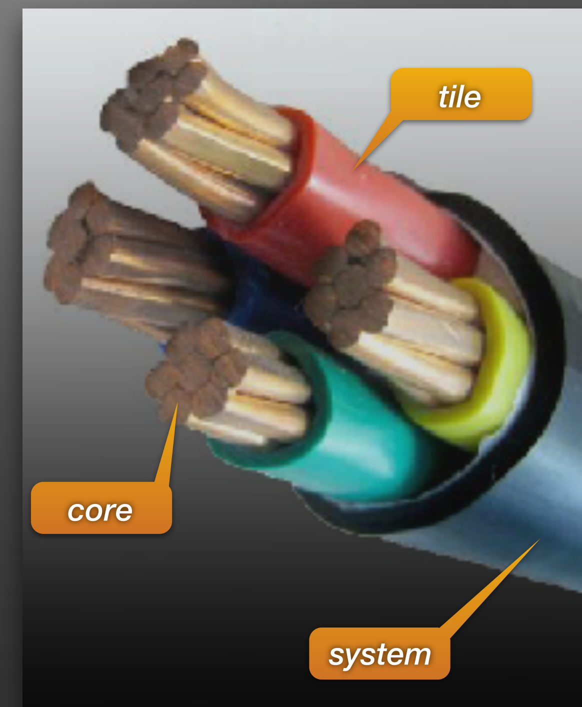
Multi-core system

Symbol of a state of the art tiled 28-core processor 😉

- 7 cores per tile, coherent
- 4 tiles per system, cache incoherent
- homogeneous at core but heterogeneous at tile level

“Many-core” goes far beyond

- several tens of cores as a lower limit
- hundreds or thousands not too far away



Memory protection

Working (main) storage

- differentiated access control
  - text, data, stack, other
  - read, write, execute
- using address monitoring

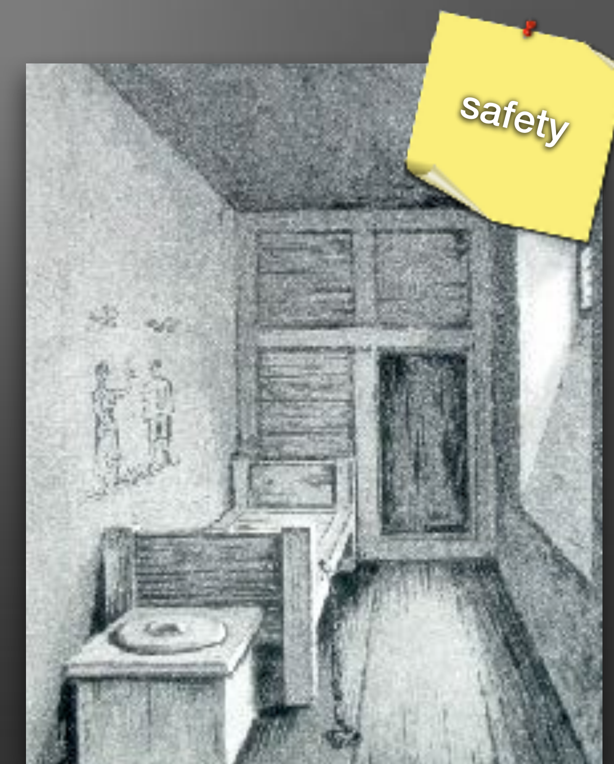
Address-space isolation

Software based

- full interpretation
  - CSIM* virtual machine
- compilation
  - type-safe language

Hardware based

- partial interpretation
  - operating system
  - M{M,P}U



*Complete Software Interpreter Machine

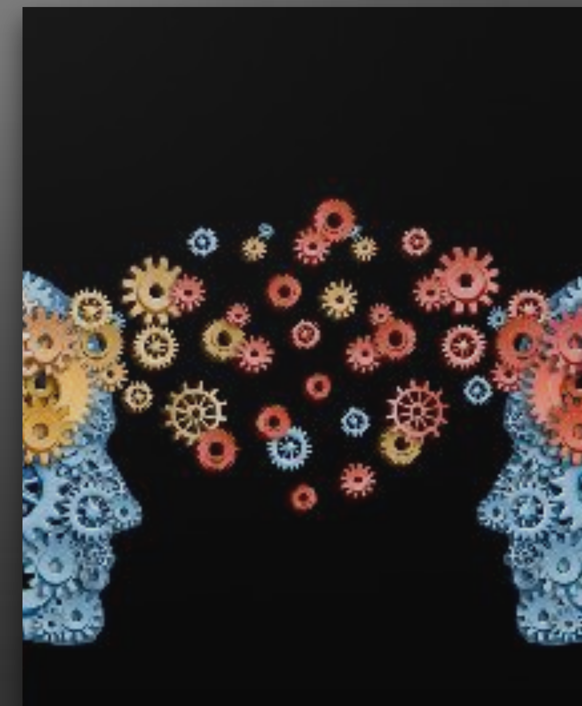
Shared (working) memory

Several processes having more or less things in common

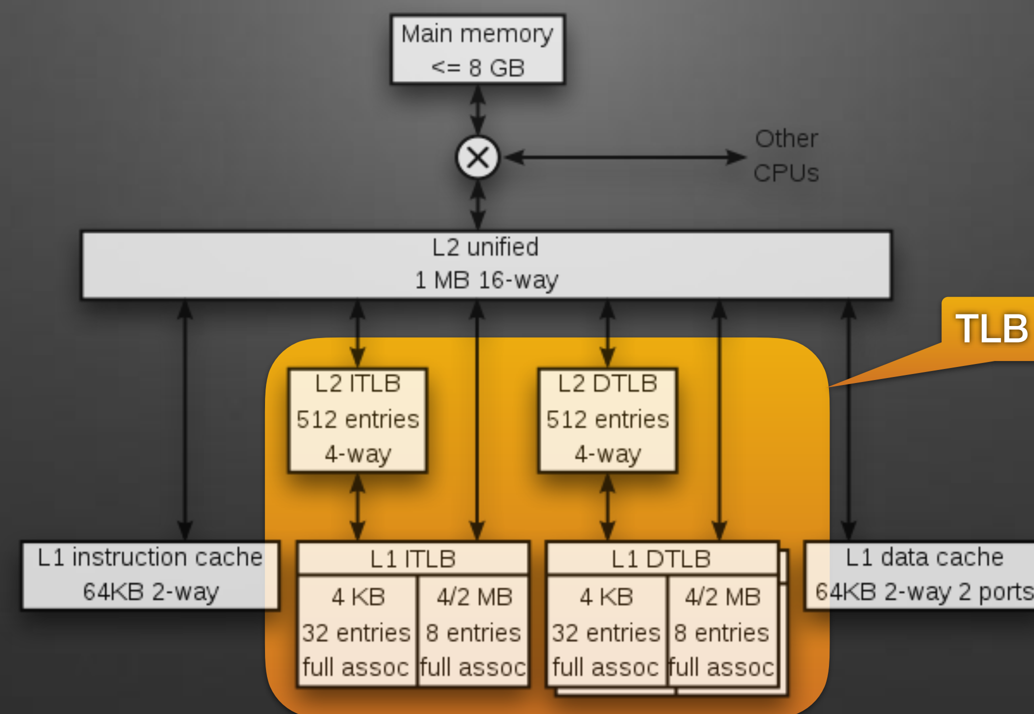
- whole program
- selected parts
- single variables

Almost noiseless for uni-core (i.e., single) processors

- single hardware resource
- CPU as well as MMU



Where the shoe pinches



Translation lookaside buffer

The address-translation cache as part of the MMU

- thus, one per CPU
- more precisely, one for each core

Breeds interference within a shared address space

- of simultaneous processes
- mapped to different cores

Figure: Hieronymus in the case, (dt. „Hieronymus im Gehäus“, Albrecht Dürer, 1514). Hieronymus applies, figuratively, as a patron saint of translators.

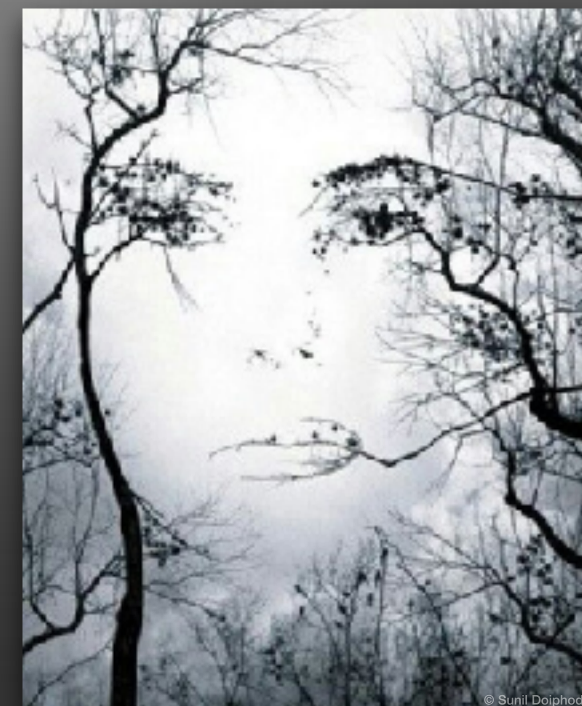
Interference

Superposition of several actions in space and time

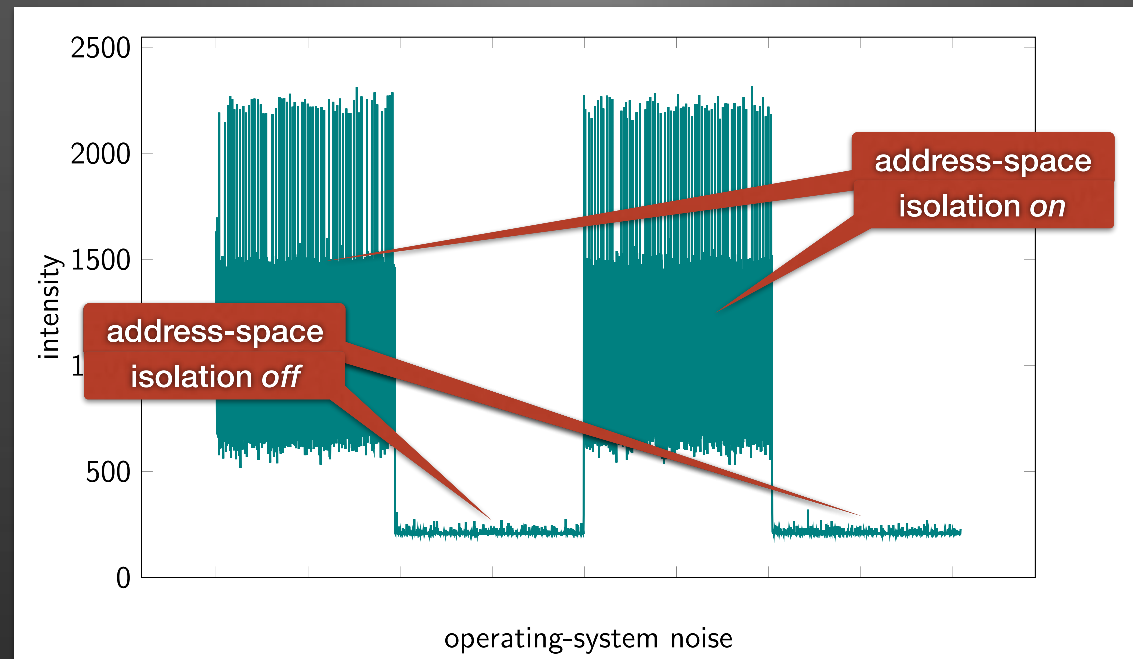
- in consequence of accessing a shared resource
- also triggered by conflicting planning/selection decisions

Obstruction of a process due to simultaneous external actions

- of other processes
- on the same or other CPU



Noise when sharing



Operating-system noise

Cause of interference

Background noise

Indirect operating-system costs as to space, time, and energy

- static for space, usually
- dynamic for time & energy
  - unsteady, suddenly
  - commonly unpredictable

Cause for delay, jitter, or failure of a process

- possible deadline violation
- troubles real-time operation

Case study: MPSoC*

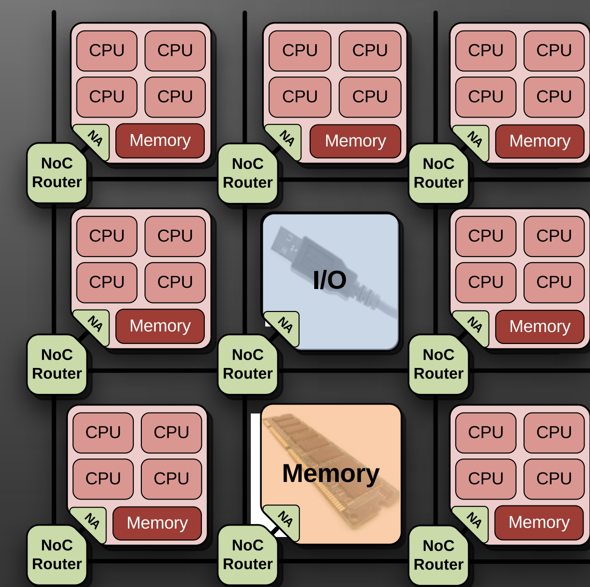
Multi-tile processor architecture

- compute tile
- memory tile
- I/O tile

Network on chip (NoC)

- network adapter (NA)
- router

Partitioned global address space (PGAS)



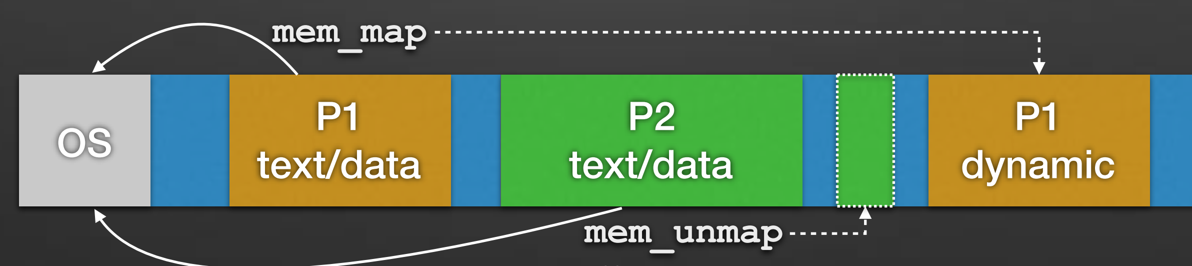
*Multiprocessor system on chip

PGAS

Several memory partitions combined into a *single address space*

- coexisting machine programs reside at different address regions
  - private regions may be mapped to identical address ranges
  - all other regions are mapped to different address ranges
- mostly *single-level store*, in principle shareable by all processes

Retrieve and release of memory results in *address-space changes*



Process isolation features inter-processor interrupts

A. multi-level page table (PT), per process
   - defines access to the actual static/dynamic memory regions

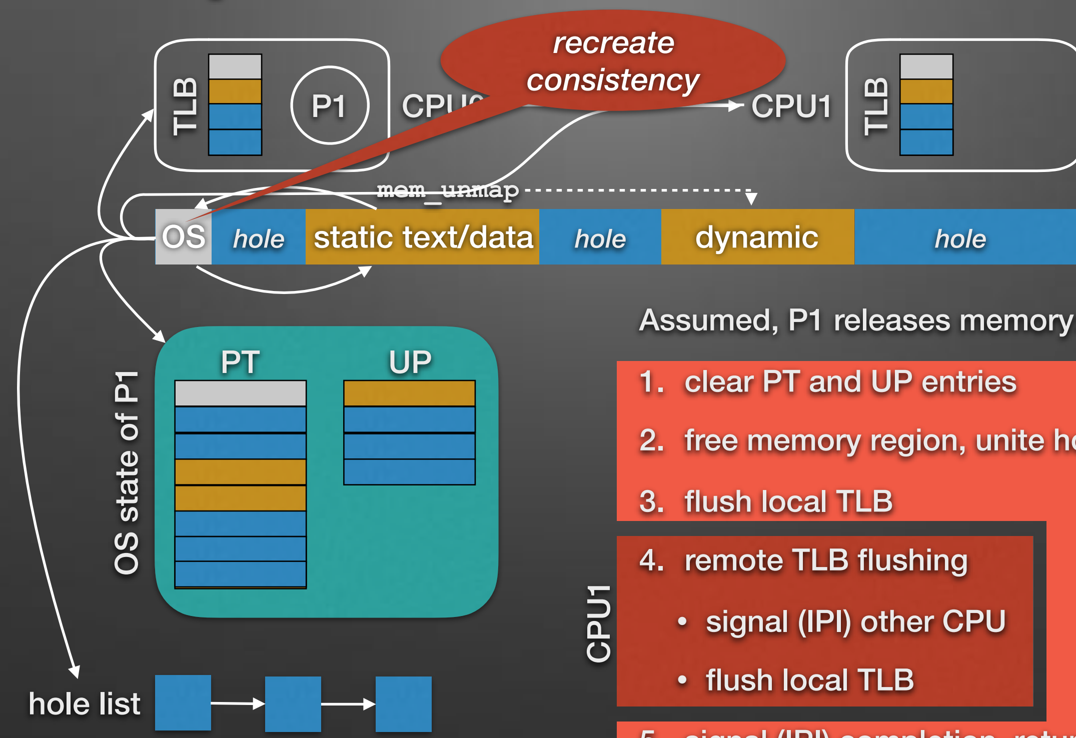
B. use-pieces (UP) of memory, per process
   - keeps track of allocated static/dynamic segments, bookkeeping

C. TLB, per CPU
   - does neither recognise PT modifications nor other TLB units

D. operating-system (OS) level memory management, per tile

E. global PT, per system — captures all machine programs plus OS

Operation with isolation



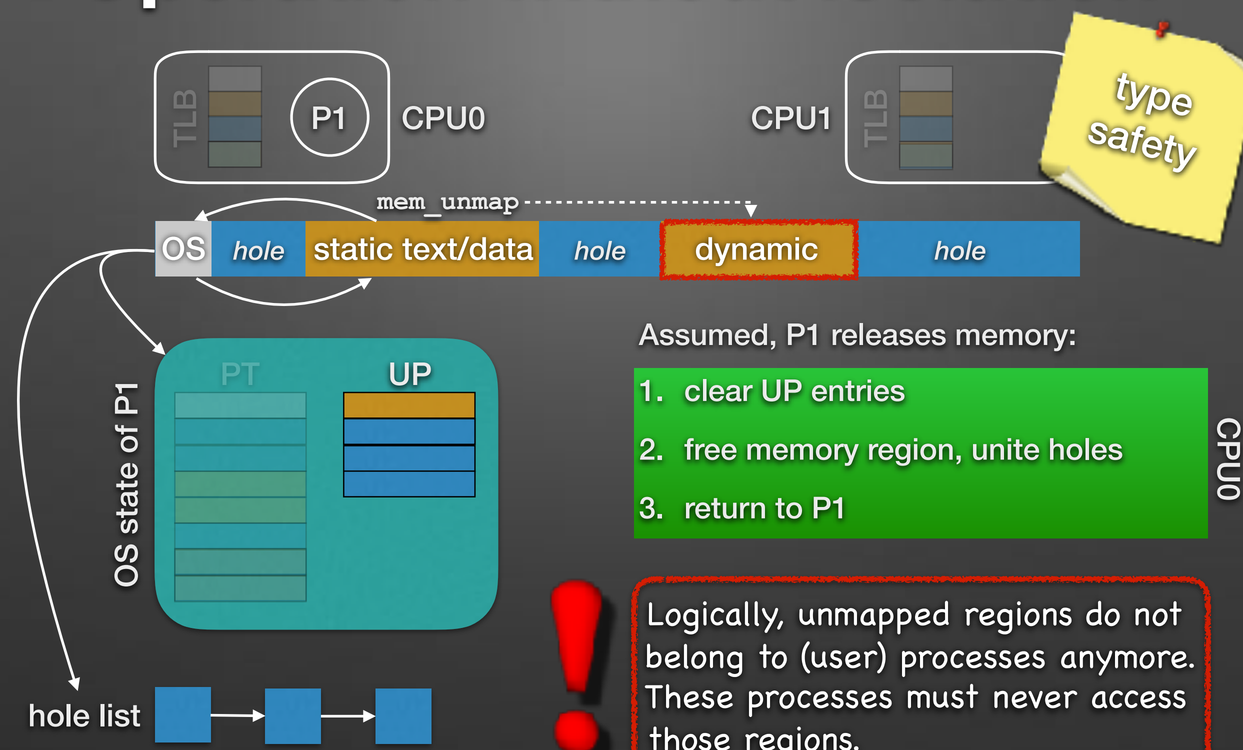
Assumed, P1 releases memory:

1. clear PT and UP entries
2. free memory region, unite holes
3. flush local TLB
4. remote TLB flushing
   - signal (IPI) other CPU
   - flush local TLB
5. signal (IPI) completion, return to P1

CPU1

CPU0

Operation without isolation



Assumed, P1 releases memory:

1. clear UP entries
2. free memory region, unite holes
3. return to P1

CPU0

Logically, unmapped regions do not belong to (user) processes anymore. These processes must never access those regions.

Operating-mode transitions

- considerably simplified by using a *flat (logical) address space*
- in addition, (user) process address-space areas do not overlap

Unprotect process address space 👉 **establish single protection domain**

- disable PT for the threads of the respective process
- if applicable, just apply global (system specific) PT

Protect process address space 👉 **establish multiple protection domains**

- restore PT from UP entries, (re-) enable PT for the respective threads
- apply local (process specific) PT

- synchronous IPI multicast to relevant processors
- flush TLB of the respective MMU

Empirical study

Noise measurement

Experimental set-up

Hardware

- 4 x Intel Xeon E7-4830 v3 @ 2.10 GHz (8 cores each)
- 512 GB DDR4 @ 1333 MHz

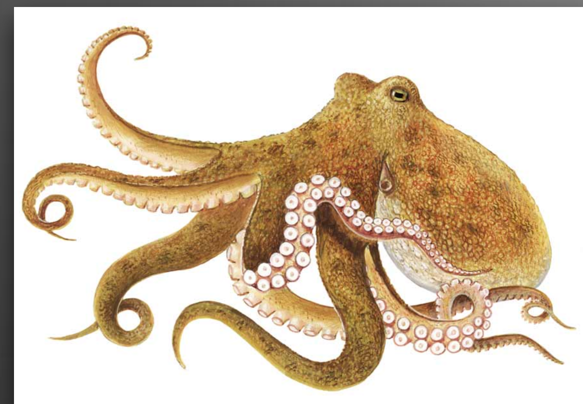
Operating modes

- address-space isolation dynamically turned on and off
- statically unprotected system

Benchmarks

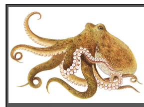
- interference provoked through IPIs and TLB flushing
- average values of 16 runs per reading, hot caches
- coefficient of variation less than 5%

Executive

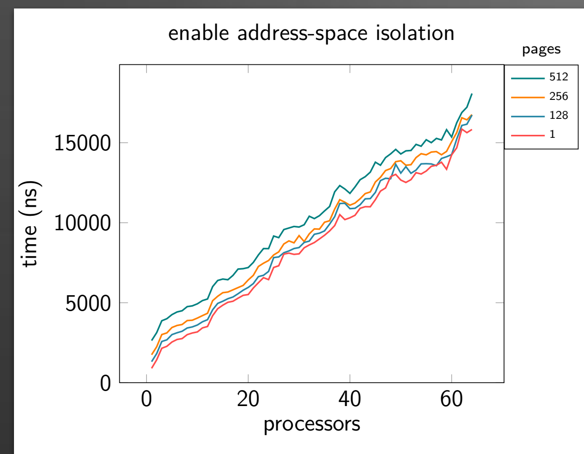


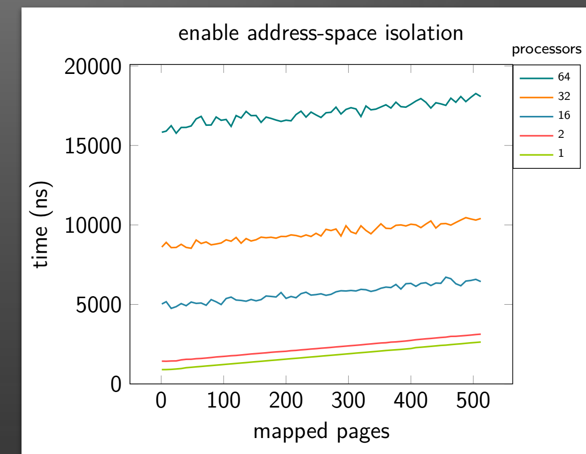
OctoPOS

A Parallel Operating System for Invasive Computing

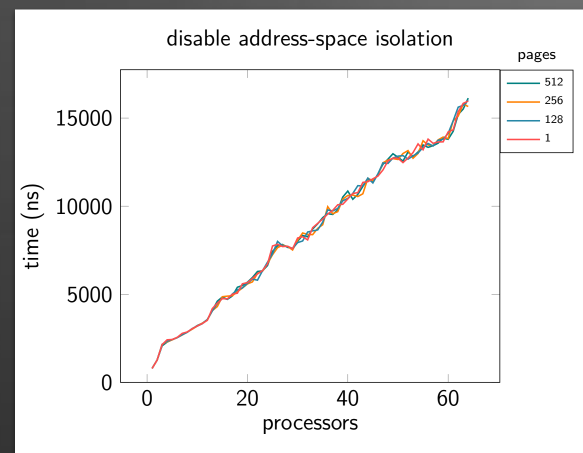


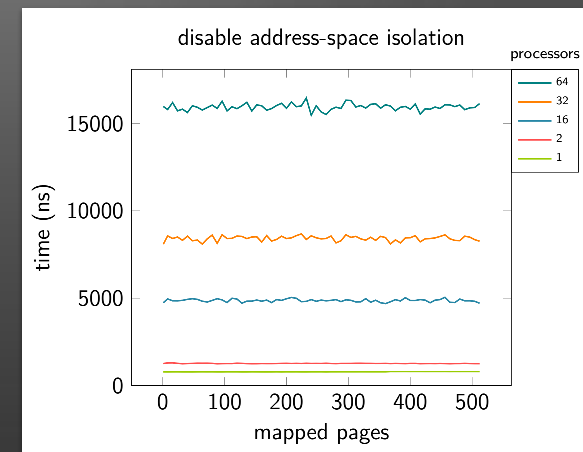
Memory protection put into operation



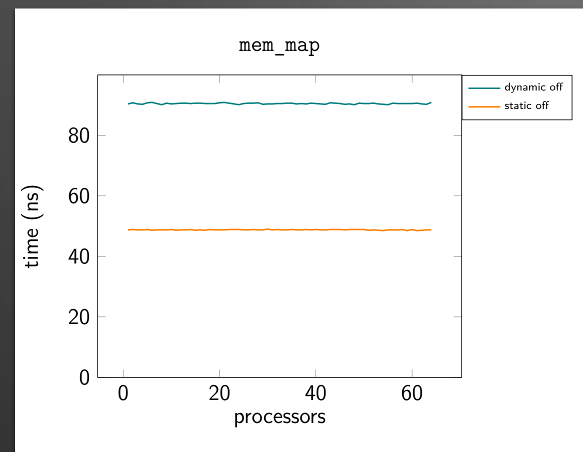


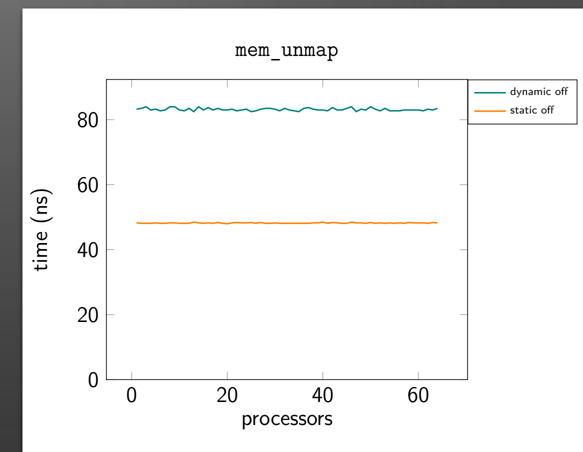
Memory protection put out of operation



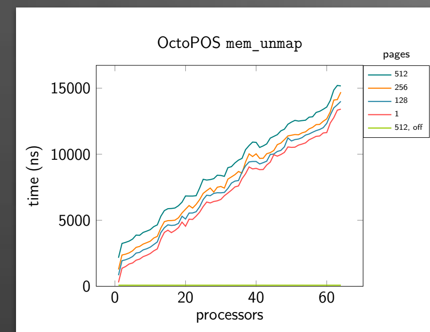


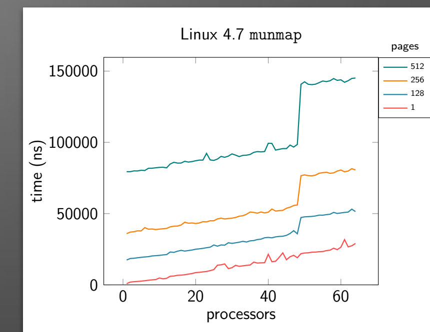
Memory protection minimal additional costs

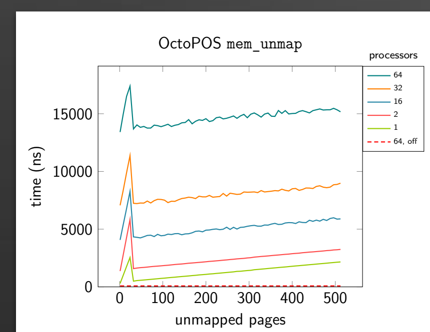


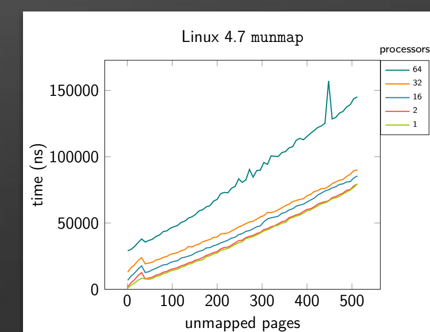


Memory protection relatively considered



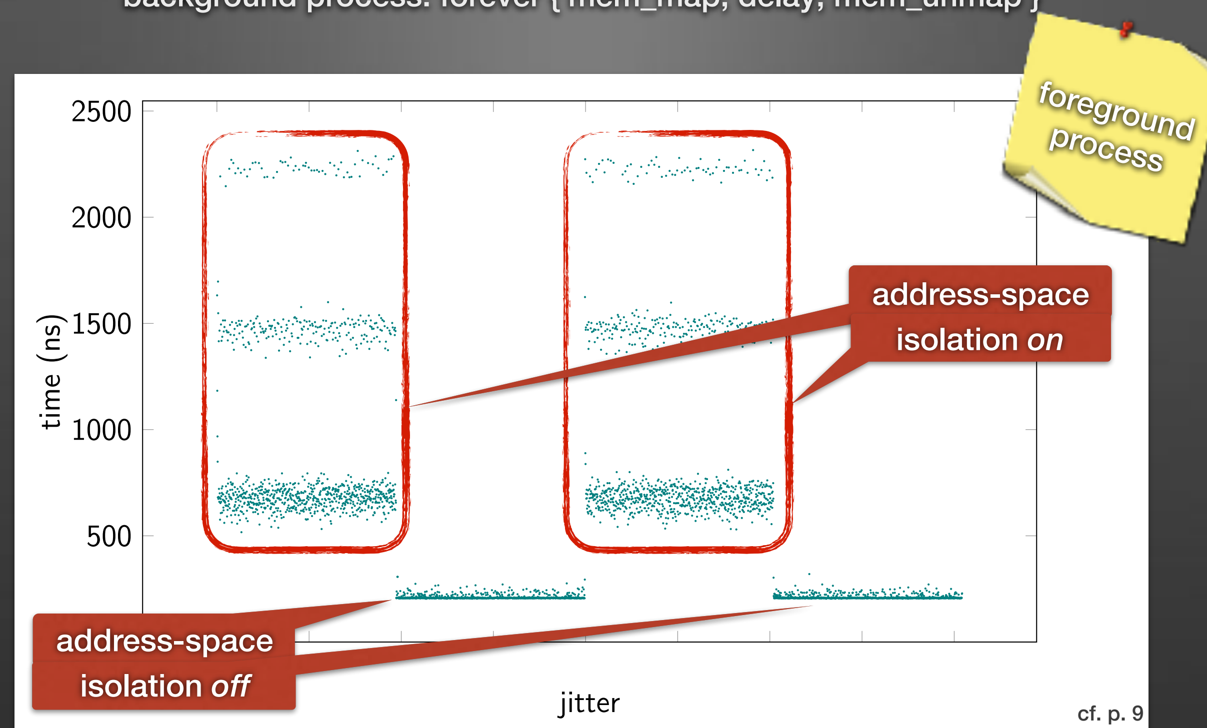






Interference in action

background process: forever { mem_map; delay; mem_unmap }



Address-space isolation 'on demand'

"Some users may require only a subset of the services or features that other users need. These 'less demanding' users may demand that they not be forced to pay for the resources consumed by the unneeded features."

*–David Parnas, 1979**

*Designing Software for Ease of Extension and Contraction, IEEE TSE, vol. SE-5, no. 2

Summary

Daß dies mit Verstand geschah
war Herr Lehrer Lämpel da.

Of this wisdom an example
To the world was Master Laempel.

(Max and Moritz — A Juvenile History in Seven Tricks by Wilhelm Busch, here: Fourth Trick)

Think application-centric…

Memory protection serves the safety and security

- hardware-based solutions are by far not dogma
- if anything, only type-unsafe processes have to be “imprisoned”

Address-space solation is not without cost and causes uncertainty

- time-dependent processes suffer from interference
- interference is the cause of many evils for predictability

Design for predictability is an overarching aspect that crosscuts the whole computing system

Acknowledgement