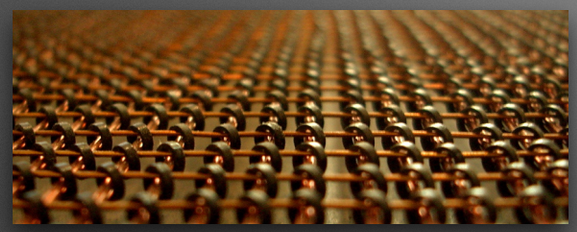
Adaptive Memory Protection for Many-Core Systems

Application-centric operating-system design

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

Memory protection



Working (main) storage

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

security
- protecting an entity from its environment

Address-space isolation

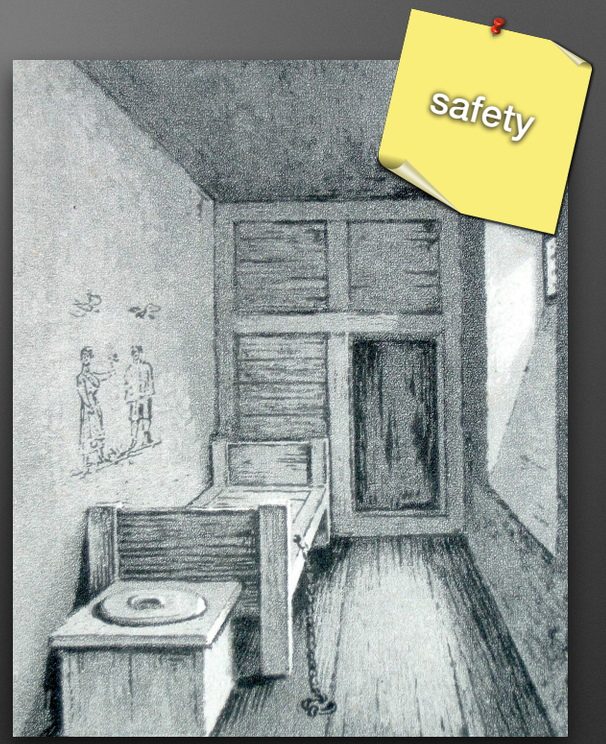
Software based

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

Hardware based

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

*Complete Software Interpreter Machine



safety
- protecting the environment from an entity

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

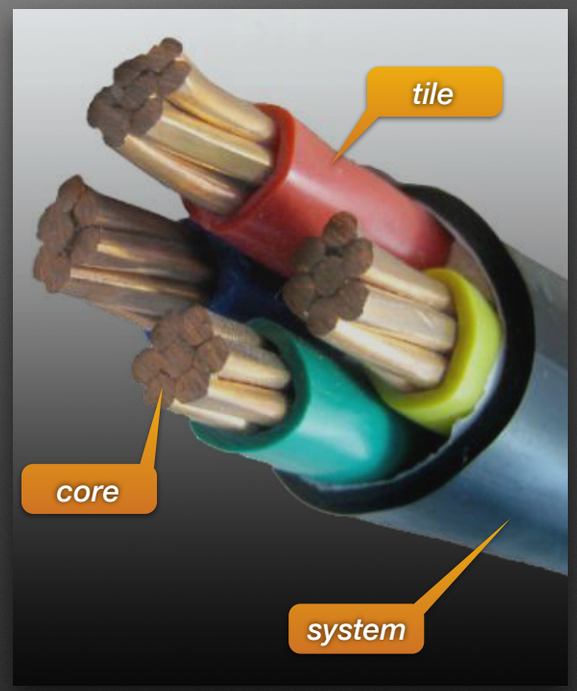


Image: The search result when, after reading Herb Sutter's article (The Free Lunch Is Over: A Fundamental Turn Toward Concurrency in Software, Dr. Dobb's Journal, 30(3), March 2005), I browsed on the Internet for the term "multi-core".

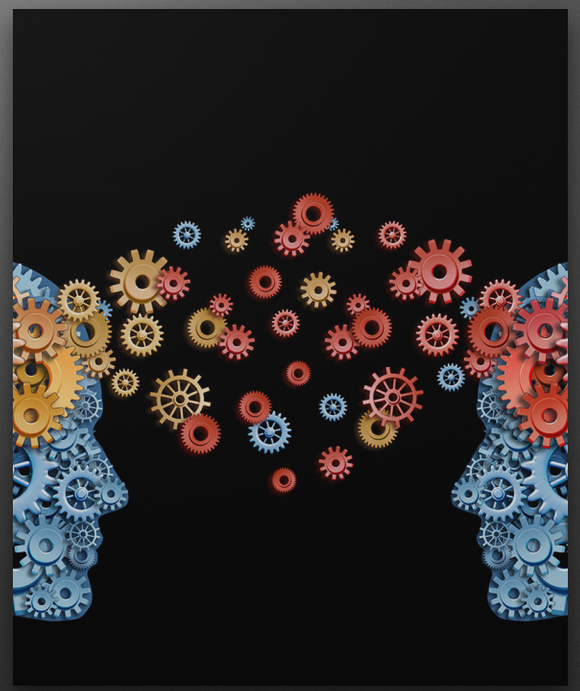
Shared (working) memory

Several processes having more or less things in common

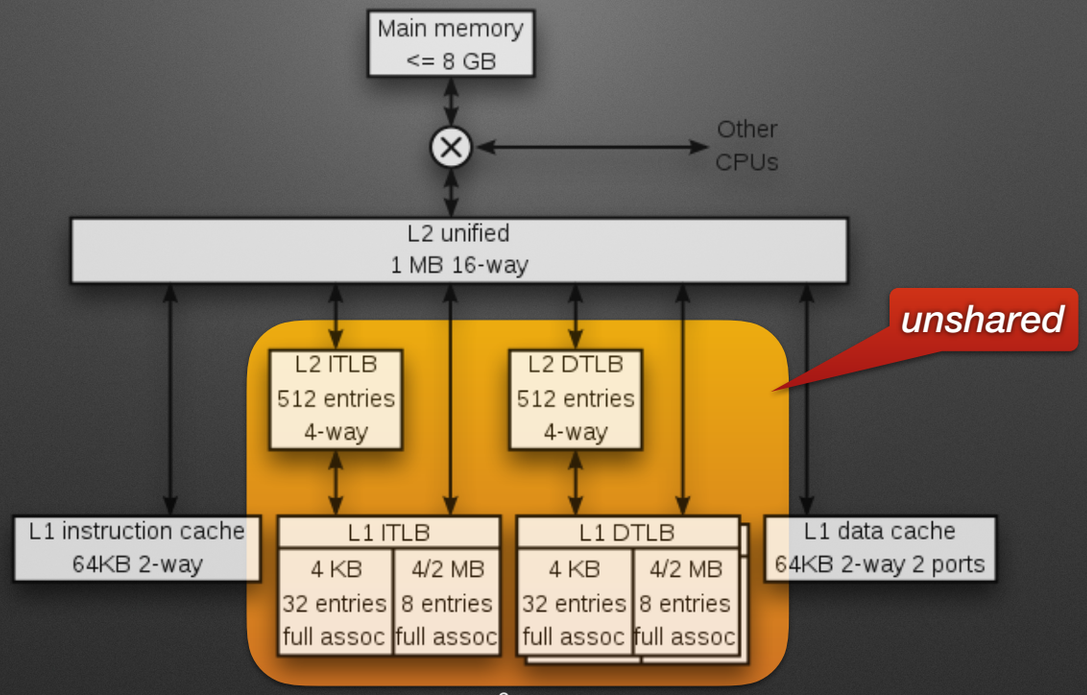
- whole program
- selected parts
- single variables

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.

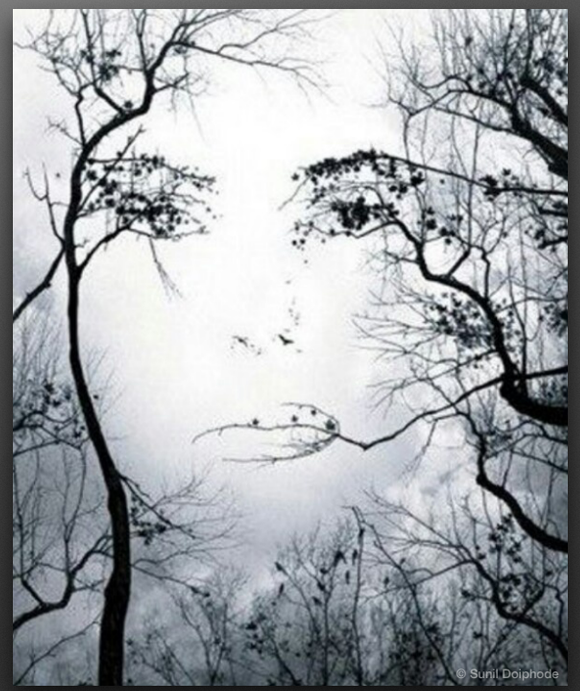
Interference

Superposition of several actions in space and time

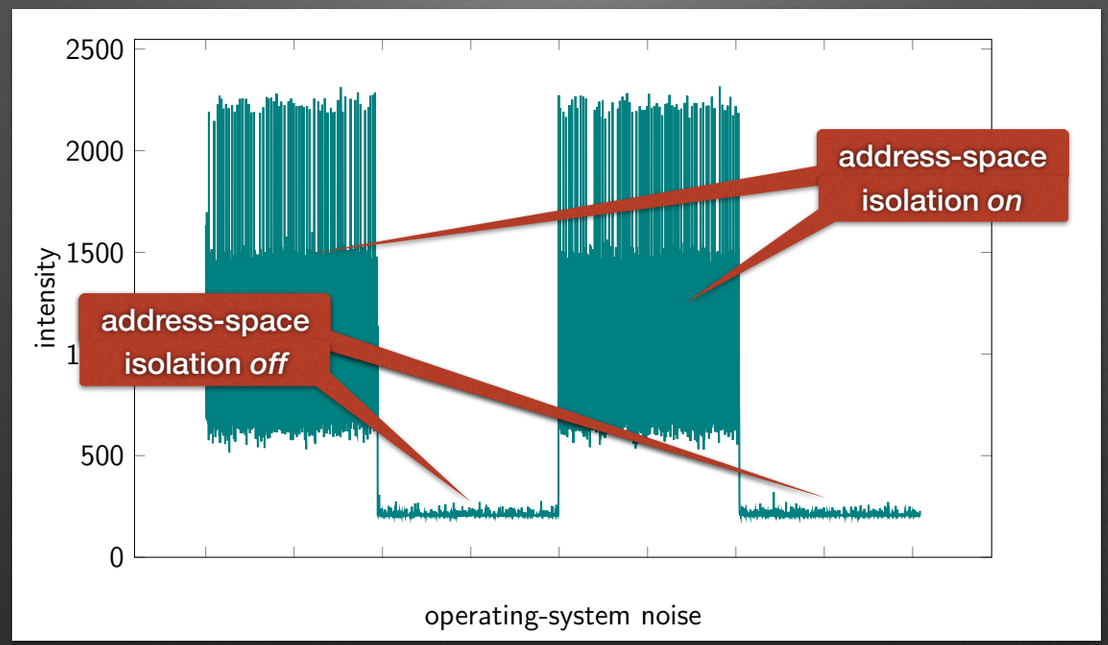
- in consequence of accessing a shared resource
- also triggered by conflicting planning/selection decisions

Obstruction of a process due to simultaneous external actions

- of other processes
- on the same or other CPU

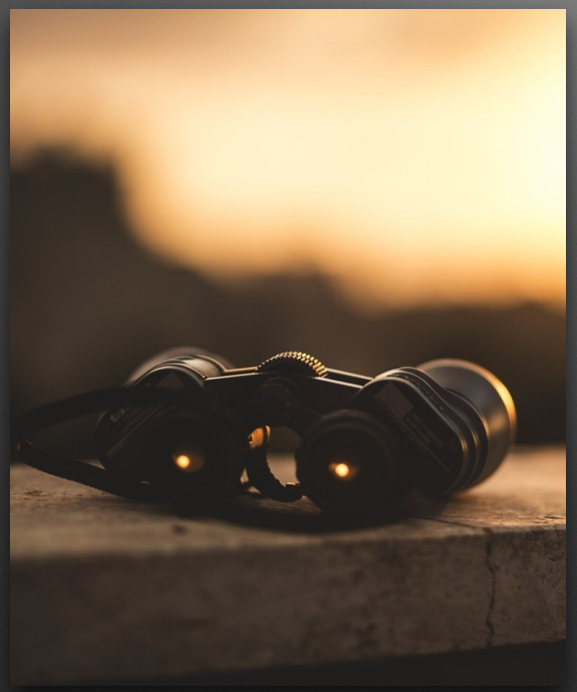


Noise when sharing



The solution to this is not Blockchain!

Outline for what follows



- Operating system noise
  - cause of interference
- Experimental study
  - extent of interference
- Resource-aware computing
  - control of interference
- Summary

Operating-system noise

Cause of interference

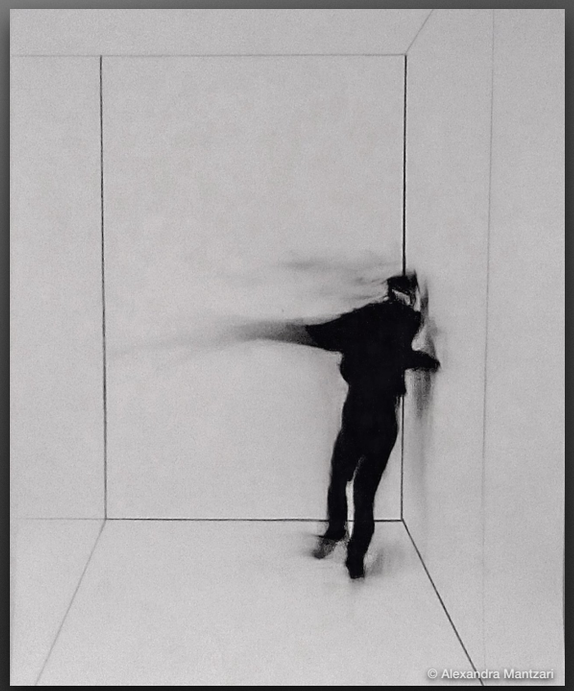
Background noise

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*

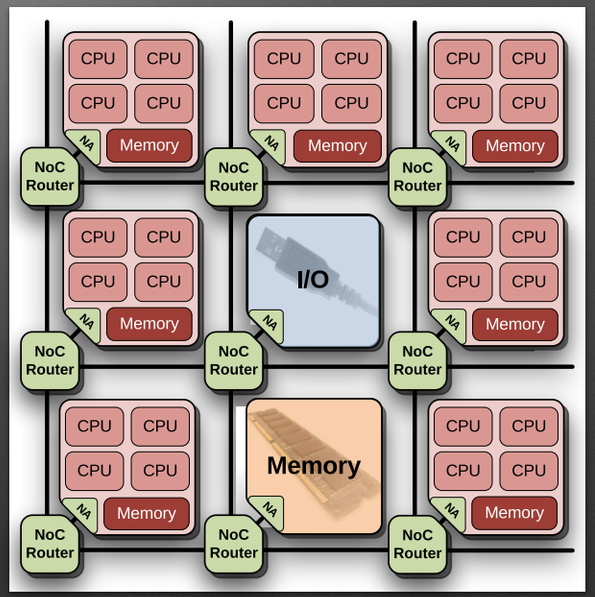
Multi-tile processor architecture

- compute tile
- memory tile
- I/O tile

Network on chip (NoC)

- network adapter (NA)
- router

Partitioned global address space (PGAS)



*Multiprocessor system on chip

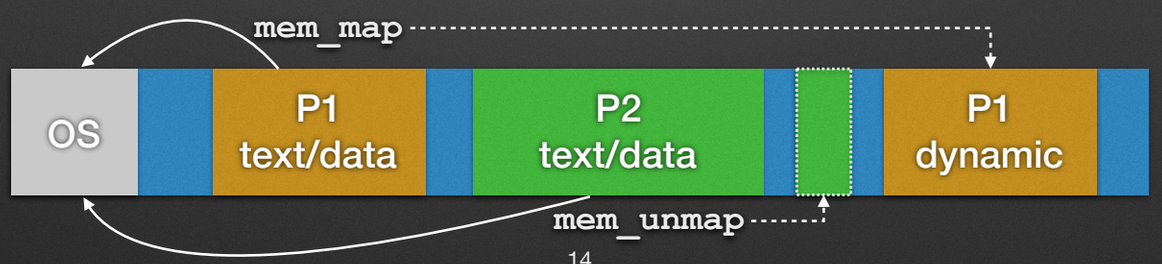
PGAS

(From the perspective of an operating system)

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

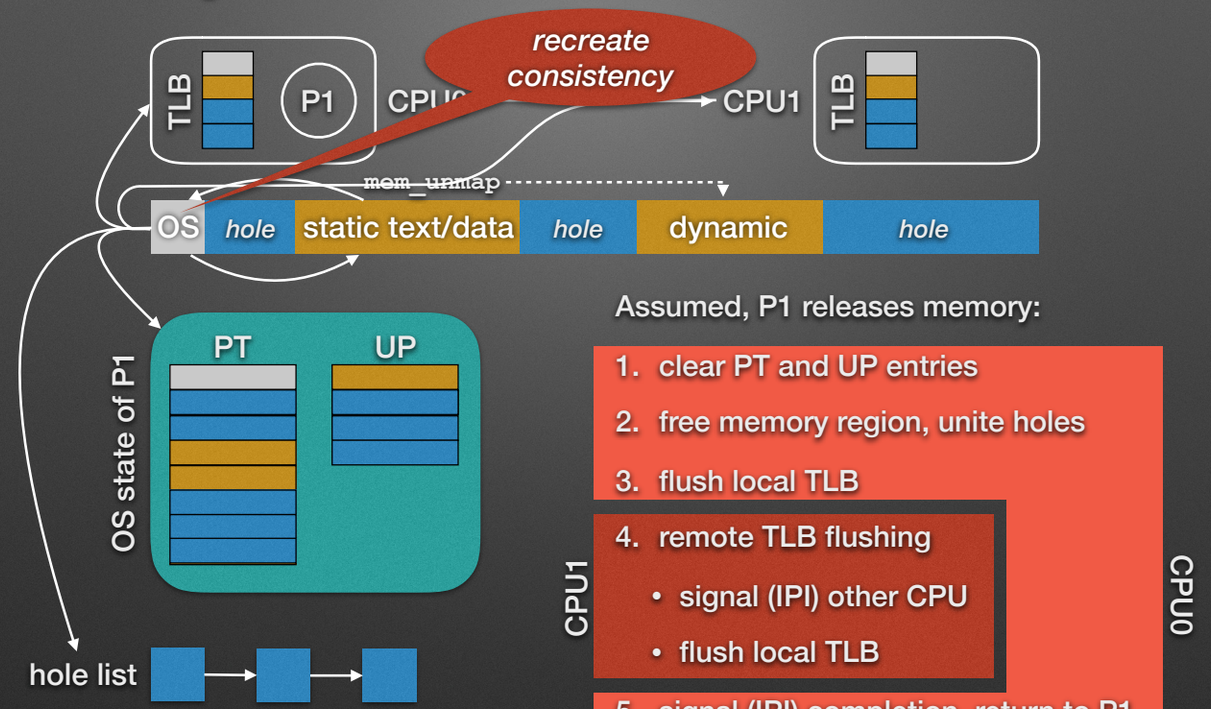
B. use-pieces (UP) of memory, per process
   - keeps track of allocated static/dynamic segments, bookkeeping

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

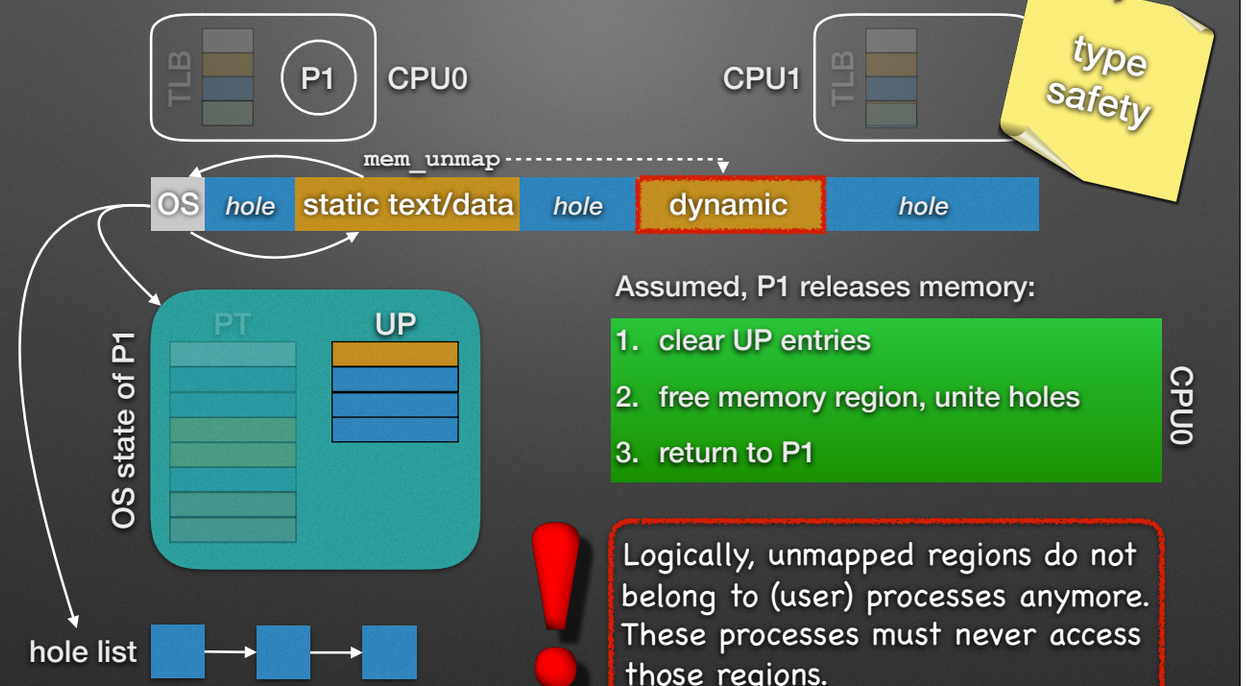
D. operating-system (OS) level memory management, per tile

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

Operation with isolation



Operation without isolation



Out of the dilemma

- ~~"cache-coherent" TLB~~
  - ~~hardware function, possibly managed by software~~
  - ~~micro-architecture level~~
- operating-mode transitions
  - dynamic reconfiguration of system software
  - operating-system level

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

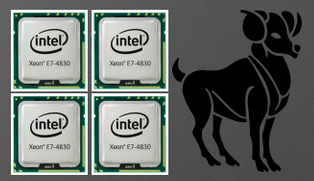
Empirical study

Noise measurement

Experimental set-up

Hardware

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



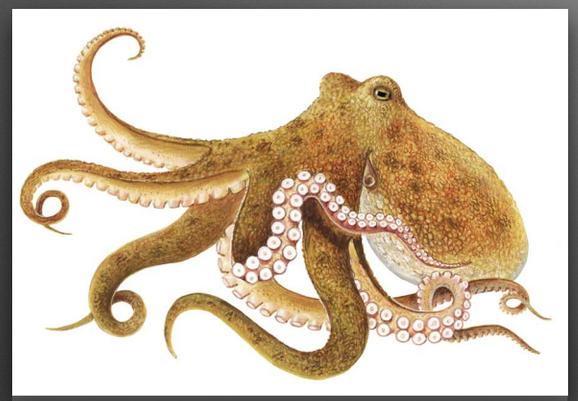
Operating modes

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

Benchmarks

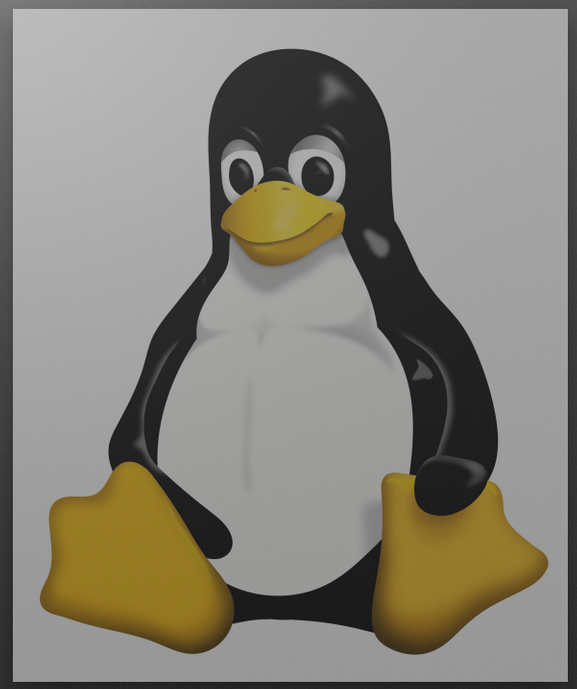
- interference provoked through IPIs and TLB flushing
- average values of 16 runs per reading, hot caches
- coefficient of variation less than 5%

Executive



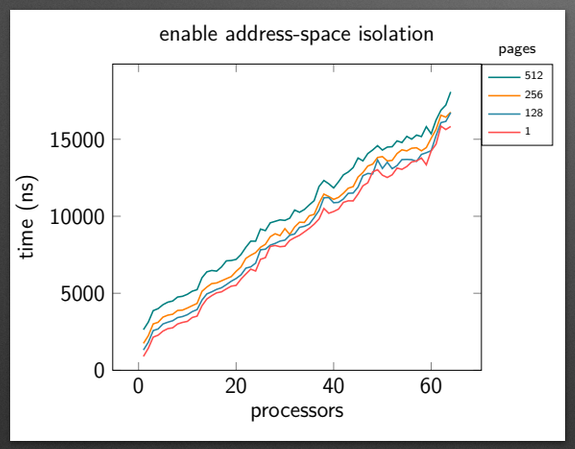
OctoPOS

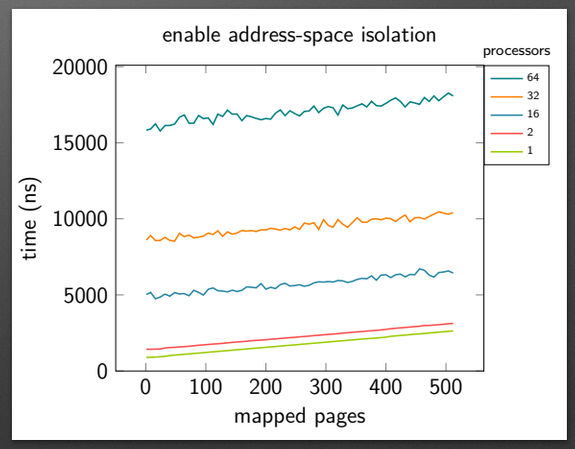
A Parallel Operating System
for Invasive Computing



Linux 4.7

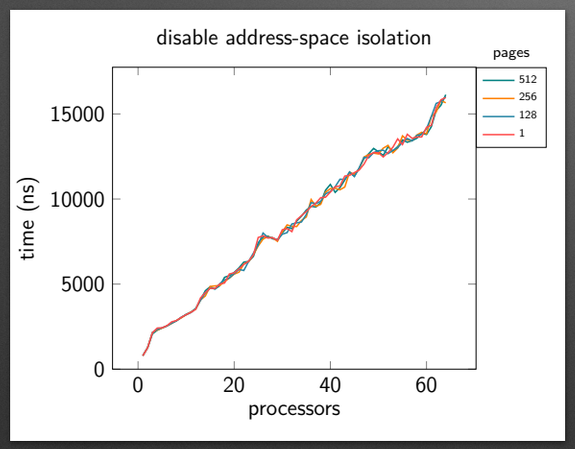
Memory protection put into operation

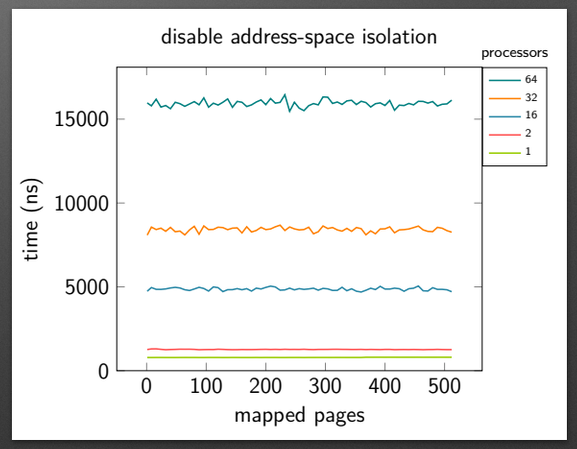




Overhead depending on the process size (in use-pieces) per CPU (left) and number of pages (right).

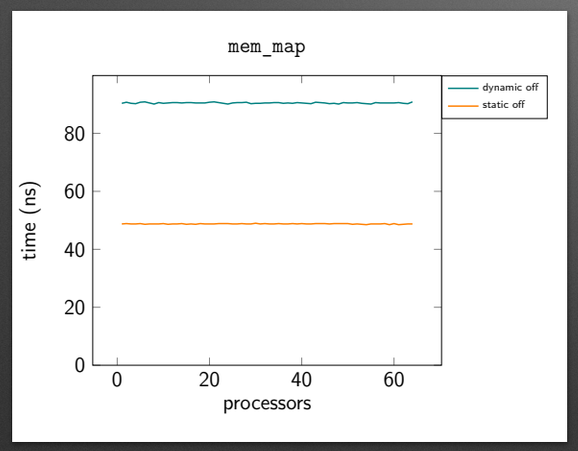
Memory protection put out of operation

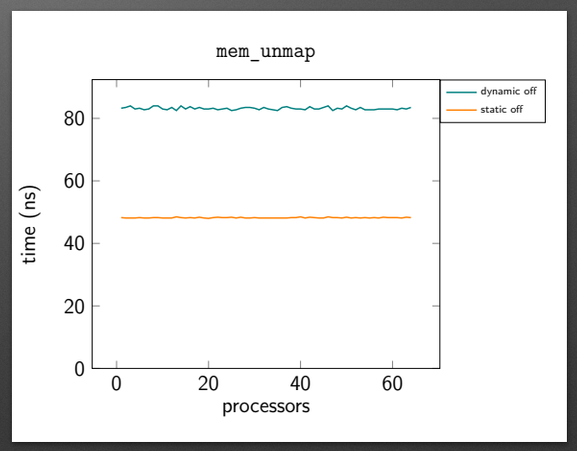




Overhead depending on the process size (in use-pieces) per CPU (left) and number of pages (right).

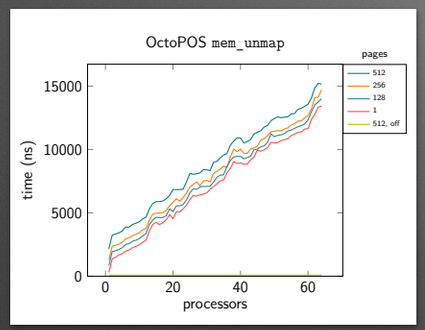
Memory protection minimal additional costs

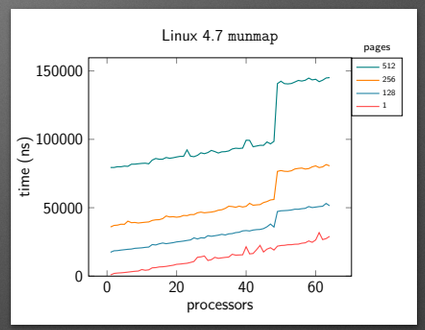


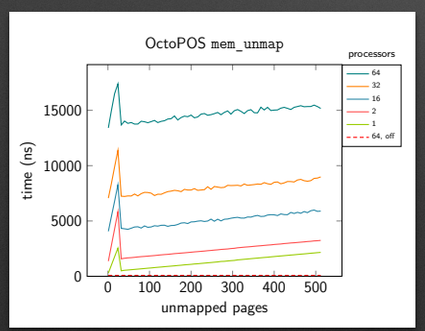


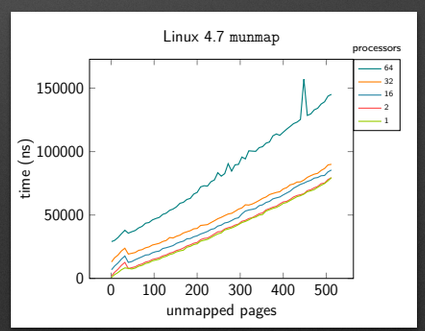
Overhead per process per CPU with address-space isolation turned on, but disabled (top), and off (bottom).

Memory protection relatively considered





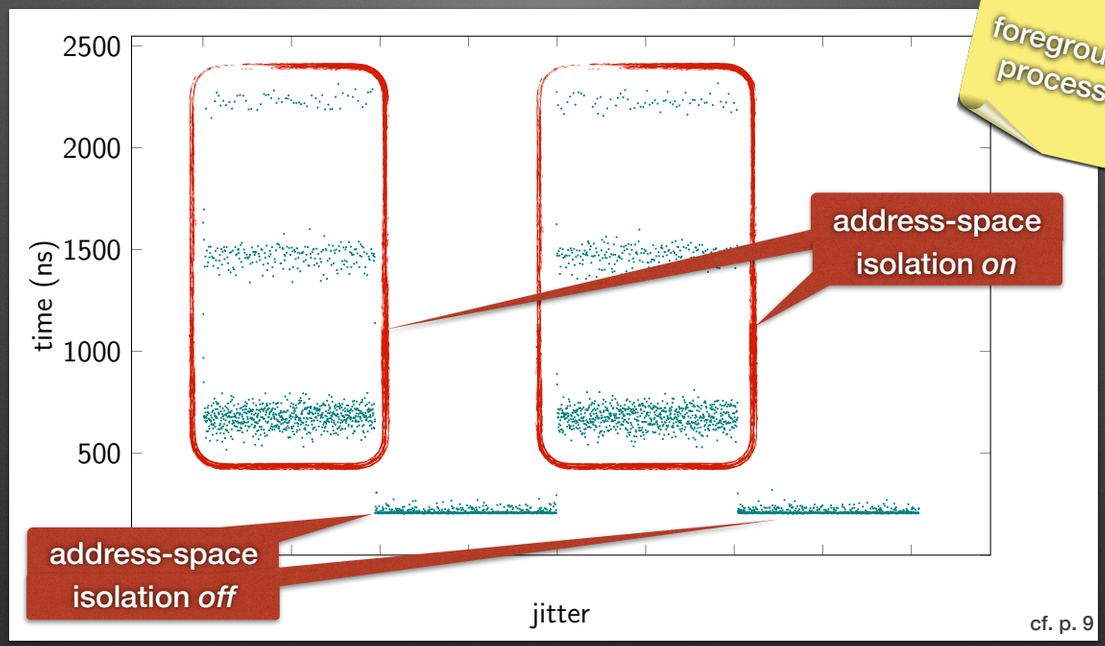




Overhead per pages unmapped for OctoPOS (left) and Linux (right): 10-fold increase from left to right.

Interference in action

background process: forever { mem_map; delay; mem_unmap }



cf. p. 9

*ray tracing application

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**

Resource-aware computing

Invasive programming

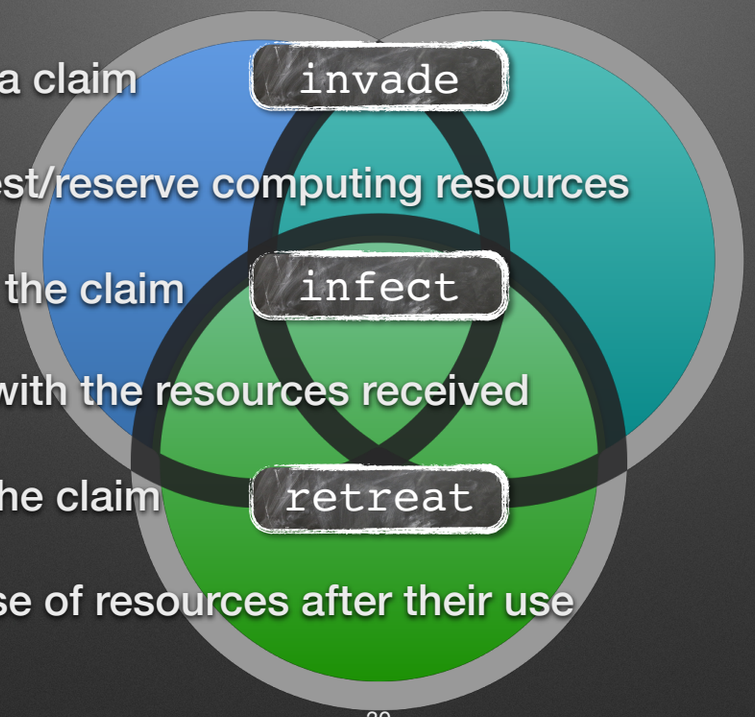
A symbol for Mindfulness.

Invasive Programming denotes the capability of a program running on a parallel computer:
- to request and temporarily claim processor, communication and memory resources in the neighbourhood of its actual computing environment (invade),
- to then execute in parallel the given program using these claimed resources (infect),
- and to be capable to subsequently free these resources again (retreat).

Resource awareness

Trinity of requisition, usage, and restoration

1. make a claim — `invade`
   - request/reserve computing resources
2. utilise the claim — `infect`
   - deal with the resources received
3. drop the claim — `retreat`
   - release of resources after their use

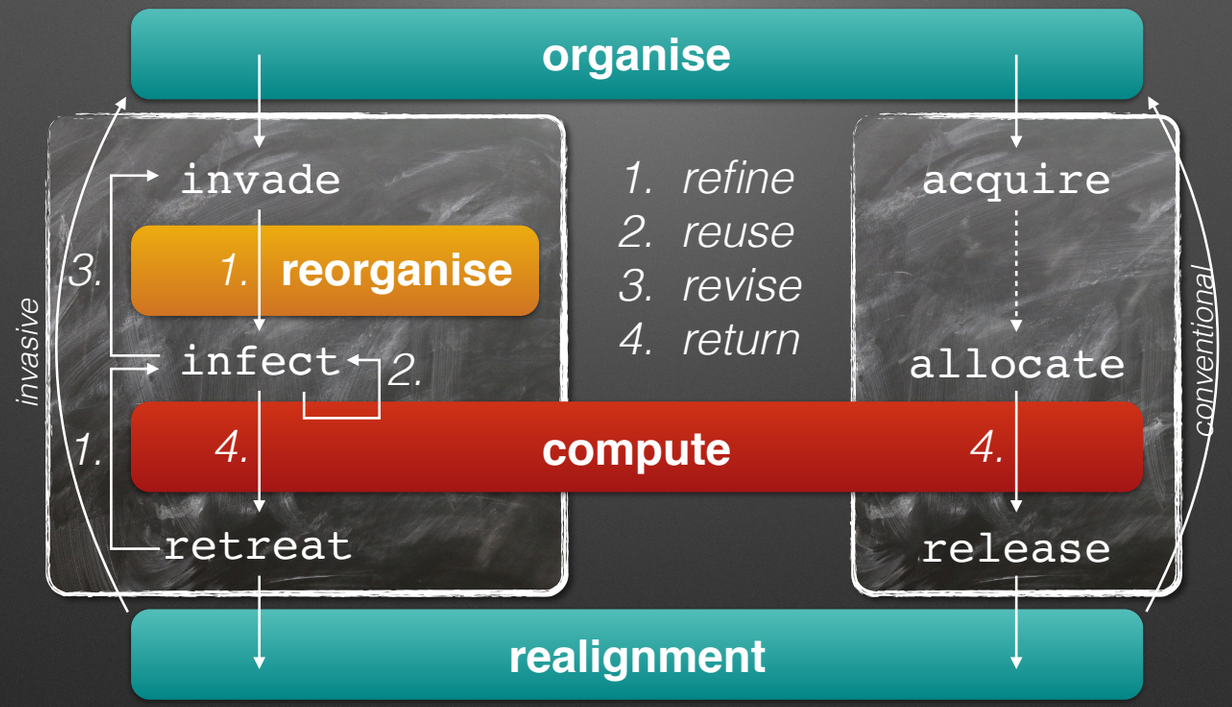


invade
- explore and claim resources in the (logical) neighbourhood of a processor running a given program

infect
- copy the same or another program into all claimed processors and start execution

retreat
- may terminate the program or just allow the invasion of its invaded resources by other programs

Resource usage cycle



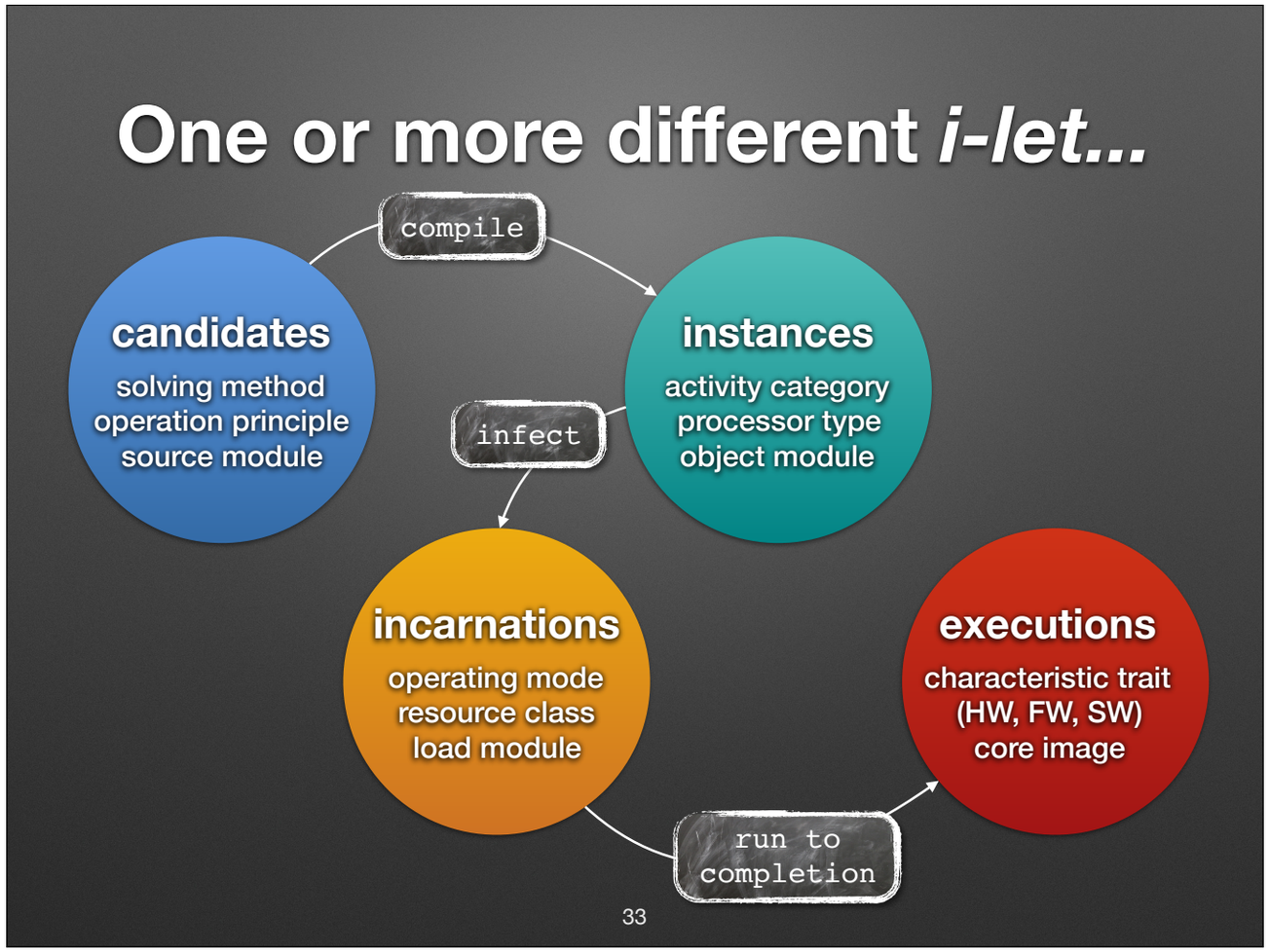
Units of “invasion”

Program section being aware of potential parallel execution

operating-system entities {
1. candidate
2. instance
3. incarnation
4. execution

invasive (based on Merriam-Webster):
- tending to spread especially in a quick manner,
- disseminating from a localised area throughout the system,
- extending and dispersing easily possibly at the expense of existing entities and balance.

To ensure situational, urgent non-sequential processes depending on internal and external constraints.



Depending on the level of abstraction considered, different i-let entities and associated properties are distinguished:

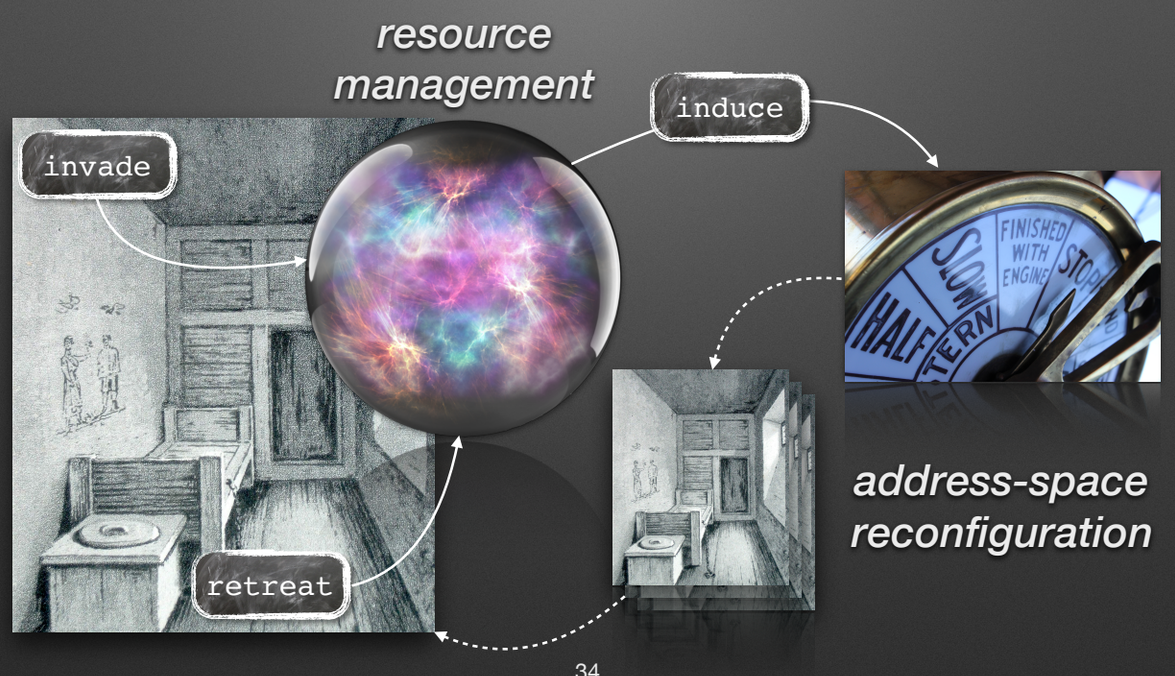
candidate
(a) prospect out of a family of algorithms for the same problem to be solved,
(b) potential cause of a specific operation principle of the (parallel) processor as to be enforced by the operating system and
(c) possibly represented and maintained as a separate source module.

instance
(a) medium of activity of an invasive-parallel program,
(b) specification of a virtual processor for it and
(c) possibly represented and maintained as a separate object module.

incarnation
(a) characteristic of the mode of operation to be realised by the operating system,
(b) ground anchor for the resources virtually needed for making progress in parallel processing and
(c) possibly represented and maintained as a separate load module.

execution
(a) actual disposition of a portion of an invasive-parallel program running on a real processor,
(b) effective unit of processing implemented in soft-, firm-, or hardware
(c) associated with a dedicated memory image.

Address-space isolation ‘on demand’ (revisited)



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

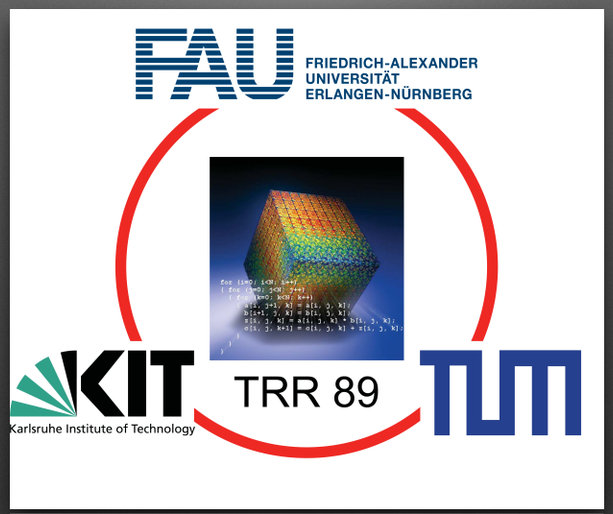
Design for predictability is an overarching aspect that crosscuts the whole computing system

Recommended reading

- J. Teich et al., Invasive Computing — Concepts and Overheads, 2012
- D. Lohmann, Tailorable System Software, 2014
- G. Drescher and W. Schröder-Preikschat, An Experiment in Wait-Free Synchronisation of Priority-Controlled Simultaneous Processes: Guarded Sections, 2015
- T. Hönig, Proactive Energy-Aware Computing, 2017
- G. Drescher, Adaptive Address-Space Management for Resource-Aware Applications, 2019

Figure: An important missive (dt. „Die Lektüre“), Carl Spitzweg, around 1870.

Acknowledgement



http://invasic.informatik.uni-erlangen.de