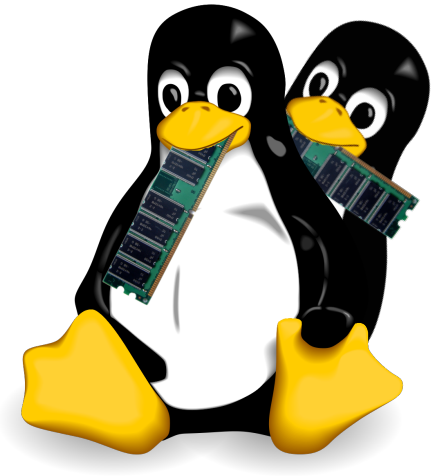
Linux NUMA evolution



survival of the quickest

or: related information on lwn.net, lkml.org and git.kernel.org

today Linux has some **understanding** on how to **handle** non-uniform mem access

- (Tux gnawing on mem modules)
- **get most out of hardware**
- 10 years ago: very different picture
- what we want to show: where are we today
  - and how did we get there
  - how did Kernel evolve: making it easier for developers

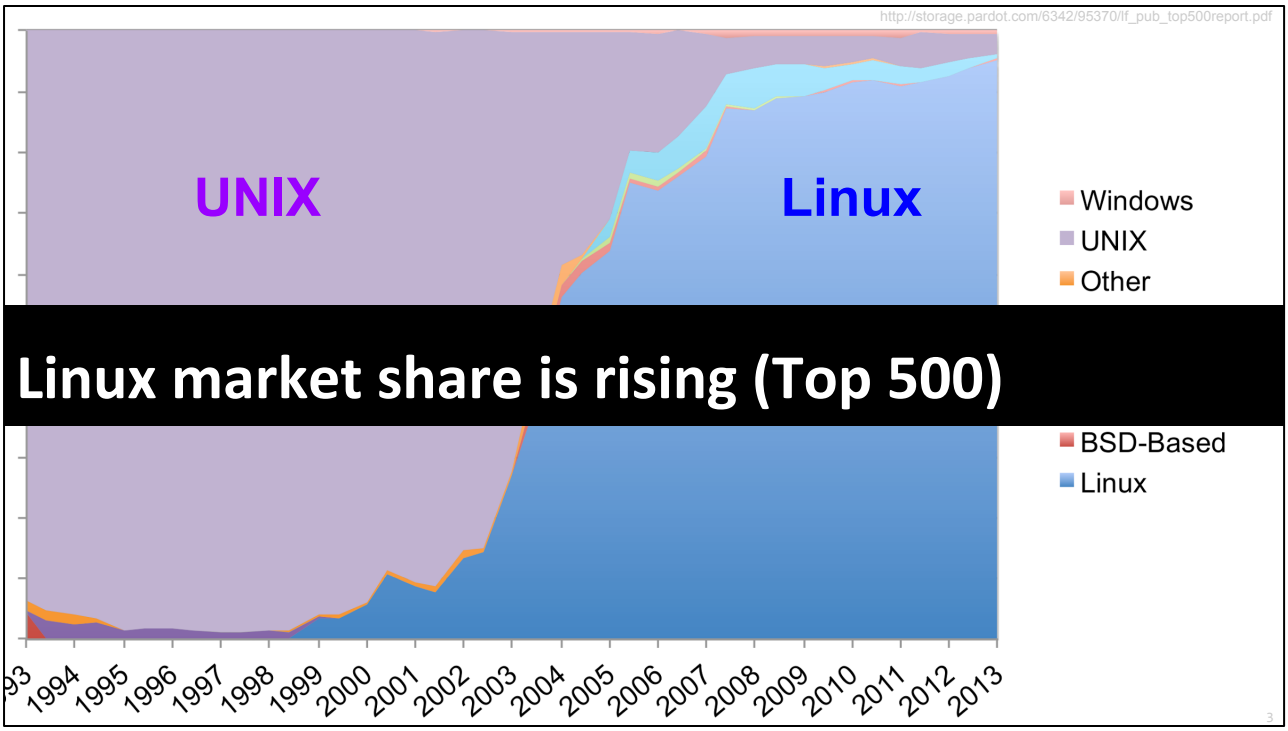
we got our information from

- lwn.net: linux weekly news -> articles, comments etc.
- lkml.org: linux kernel mailing list: lots of special sub-lists
  - discussion of design/implementation of features
    - include patches (source code)
- git.kernel.org
  - find out what got merged when
  - but for really old stuff that was not possible
  - so also change logs of kernels before 2005

Why Linux anyways?

Why Linux anyways?

- isn't Windows usually supported best?
- not for typical NUMA hardware



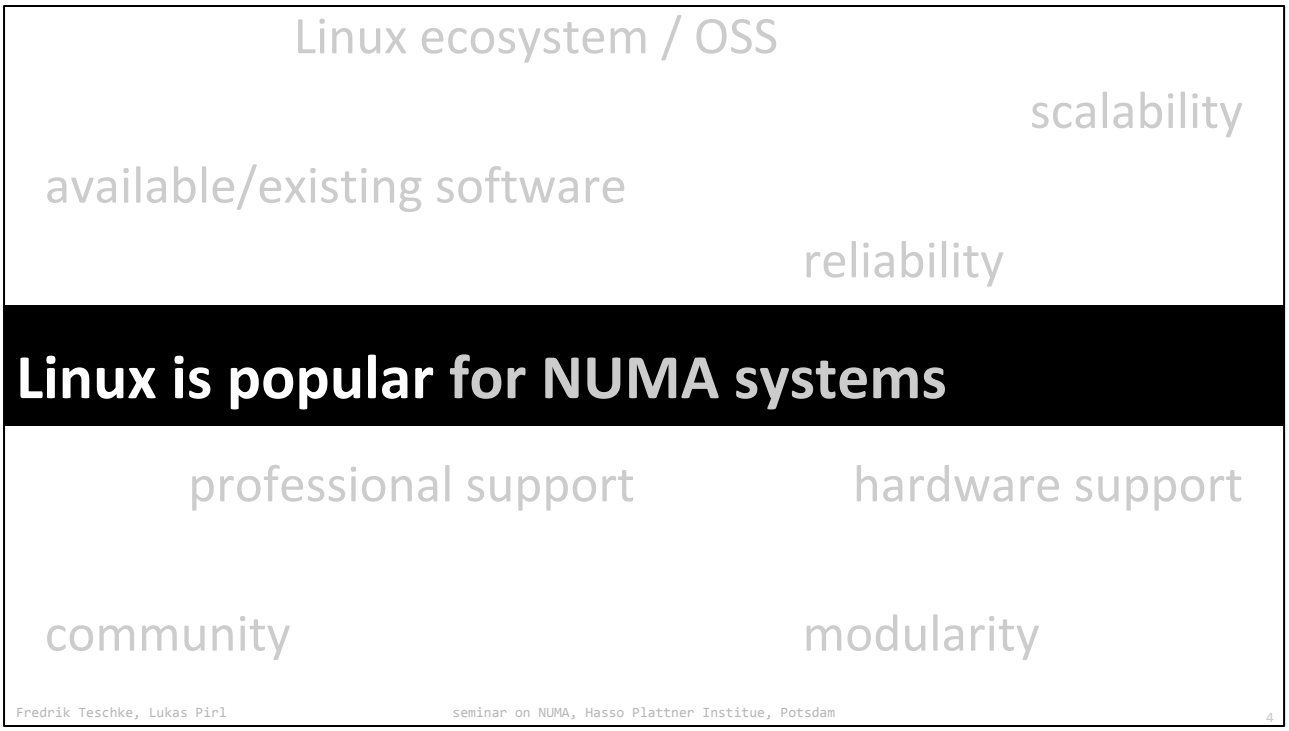
Linux market share is rising (Top 500)
top 500 supercomputers (http://top500.org/)

first Linux system: **1998**

- first basic NUMA support in Linux: 2002

from 2002: skyrocketed

- **not economical** to develop **custom OS** for every project
- no licensing cost! important if large cluster
- major vendors contribute



Linux is popular for NUMA systems

hardware in supercomputing: **very specific**

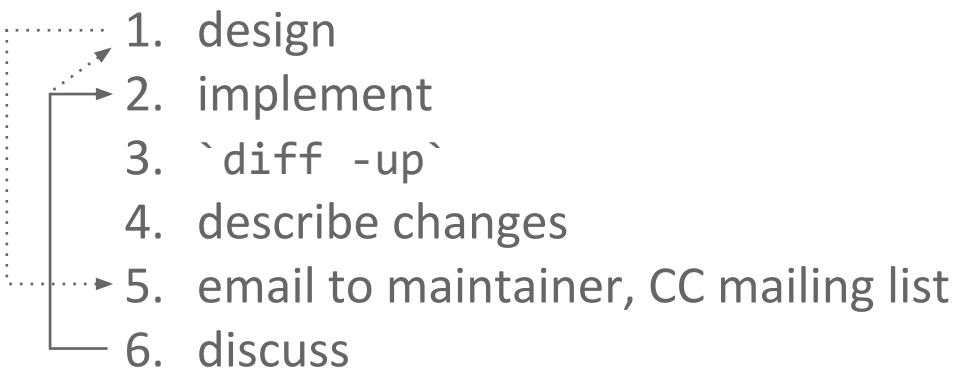
- develop OS support prior to hardware release

applications very specific

- **fine tuning** required
- **OSS** desired
  - easily adapt
  - knowledge base exists

https://www.kernel.org/doc/Documentation/SubmittingPatches (20.11.2014)

kernel development process



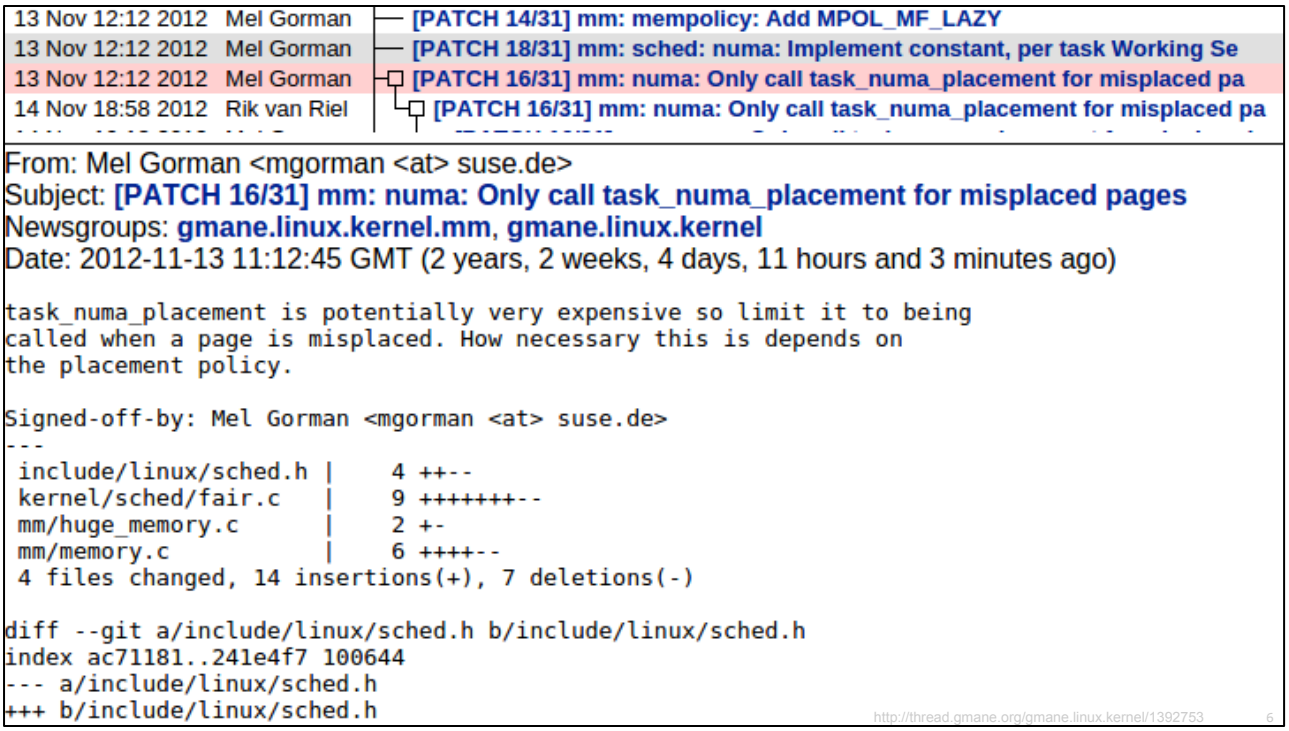
kernel development process depicted

1. design
2. implement
3. diff -up: list changes
4. describe changes
5. email to maintainer, CC mailing list
6. discuss

dotted arrow: Kernel Doc

- design often done without involving the community
- but better in the open if at all possible
- save a lot of time redesigning things later

if there are review complaints: fix/redesign



13 Nov 12:12 2012 Mel Gorman — [PATCH 14/31] mm: mempolicy: Add MPOL_MF_LAZY
13 Nov 12:12 2012 Mel Gorman — [PATCH 18/31] mm: sched: numa: Implement constant, per task Working Se
13 Nov 12:12 2012 Mel Gorman — [PATCH 16/31] mm: numa: Only call task_numa_placement for misplaced pa
14 Nov 18:58 2012 Rik van Riel — [PATCH 16/31] mm: numa: Only call task_numa_placement for misplaced pa

From: Mel Gorman <mgorman <at> suse.de>
Subject: **[PATCH 16/31] mm: numa: Only call task_numa_placement for misplaced pages**
Newsgroups: **gmane.linux.kernel.mm, gmane.linux.kernel**
Date: 2012-11-13 11:12:45 GMT (2 years, 2 weeks, 4 days, 11 hours and 3 minutes ago)

```
task_numa_placement is potentially very expensive so limit it to being
called when a page is misplaced. How necessary this is depends on
the placement policy.

Signed-off-by: Mel Gorman <mgorman <at> suse.de>
---
 include/linux/sched.h |    4 ++--
 kernel/sched/fair.c   |    9 +++++++--
 mm/huge_memory.c      |    2 +-
 mm/memory.c           |    6 ++++--
 4 files changed, 14 insertions(+), 7 deletions(-)

diff --git a/include/linux/sched.h b/include/linux/sched.h
index ac71181..241e4f7 100644
--- a/include/linux/sched.h
+++ b/include/linux/sched.h
```

http://thread.gmane.org/gmane.linux.kernel/1392753 6

development process example

at top: see that this is a patch set

each patch contains

- description of changes
- diff

and then replies via email

- so basically: all a bunch of mails
- this just happens to be Linus favourite form of communication

step 7: send pull request to Linus … mostly

Kernel Doc

- 2.6.38 kernel: only **1.3% patches** were **directly chosen by Linus**
- but **top-level maintainers ask Linus** to pull the patches they selected

getting patches into kernel depends on finding the right maintainer

- sending patches **directly to Linus** is **not normally the right way** to go

chain of trust

- subsystem maintainer may trust others
- from whom he pulls changes into his tree

https://www.kernel.org/doc/Documentation/development-process/2.Process, http://www.linuxfoundation.org/sites/main/files/publications/whowriteslinux.pdf

kernel development process

some other facts

major release: every 2–3 months

2-week merge window at beginning of cycle

linux-next tree as staging area

git since 2005

linux-kernel mailing list: 700 mails/day

some other facts

- **major release**: every **2–3 months**
- **2-week merge window** at beginning of cycle
- **linux-next** tree as staging area
- **git since 2005**
  - before that: patch from email was applied manually
  - made it difficult to stay up to date for developers
  - and for us: a lot harder to track what got patched into mainstream kernel
- linux-kernel **mailing list: 700** mails/day

https://www.kernel.org/doc/Documentation/development-process/2.Process

kernel development process

> There is [...] a somewhat involved (if somewhat informal) process designed to ensure that each patch is reviewed for quality and that each patch implements a change which is desirable to have in the mainline.
>
> **This process can** happen quickly for minor fixes, or**, in the case of large and controversial changes, go on for years.**

paragraph taken from Kernel documentation on dev process

- There is [...] a somewhat involved (if somewhat informal) process
- designed to ensure that each patch is reviewed for quality
- and that each patch implements a change which is desirable to have in the mainline.
- This process can happen quickly for minor fixes,
- or, in the case of large and controversial changes, go on for years.

recent NUMA efforts: lots of discussion

people

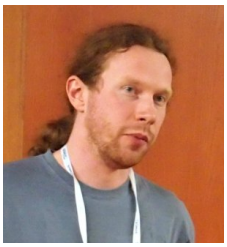
early days

Paul McKenney (IBM)

nowadays

Peter Zijlstra

redhat, now Intel: sched



Mel Gorman

IBM, now Suse: memory

Rik van Riel

redhat: mm/sched/virt

people

short look at kernel hackers working on NUMA

- there are **many more**, just the most important

early days: Paul McKenny (IBM)

- beginning of last decade

nowadays

- Peter Zijlstra
  - redhat, Intel sched
- Mel Gorman
  - IBM, Suse mm
- Rik van Riel
  - redhat mm/sched/virt

finding pictures quite difficult - just regular guys

work on kernel **full-time**

- for **companies** providing linux distributions

also listed: parts of kernel the devs focus on

- **mm**: memory management
- **sched**: scheduling

can see **two core areas**

- **scheduling**: which thread runs when and where
- **and mem mgmt:** where is mem allocated, paging
- both relevant for NUMA

recap: NUMA hardware

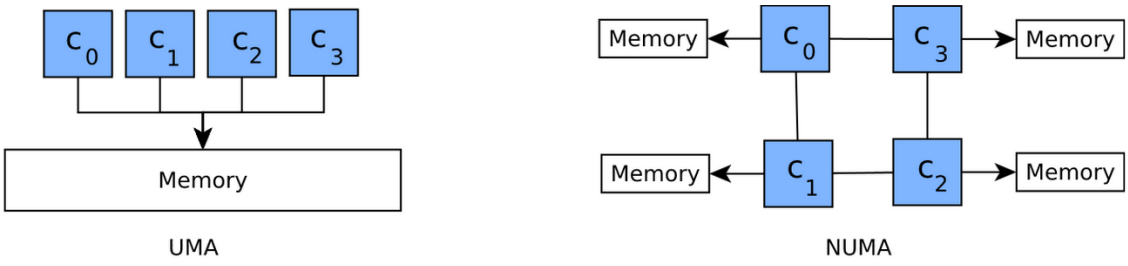


Figure 8: The UMA (*aka* SMP) and NUMA memory architectures.

A Survey on Parallel Computing and its Applications in Data-Parallel Problems Using GPU Architectures
(http://www.global-sci.com/openaccess/v15_285.pdf)

19.11.14 - Kirstin Heidler - NUMA Seminar

now **recap** of some areas

first: **NUMA hardware**
this slide: very basic - you probably know it by heart

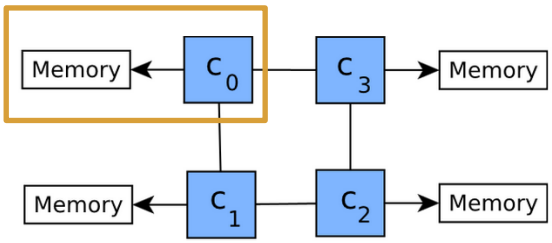
left: UMA

right: NUMA

- multiple **memory controllers**
- **access times** may **differ** (non-uniform)
- direct consequence: **several interconnects**

caution: terminology in the community

node NUMA node



task scheduling entity (process/thread)

caution: terminology in the community

Linux does some things different than others

- this influences terminology

node: as in **NUMA node**
highlighted area: one node
!= node (computer) **in cluster**
may have several processors

now **three terms** you have to be **very careful** with

- **task, process and thread**
- in Linux world: **task is not a *work package***
  - instead: scheduling entity
- that used to mean: task == process
  - then threads came along
- Linux is different: **processes and threads** are pretty much the **same**
  - **threads** are just configured to **share resources**
  - pthreads_create() -> new task spawned via clone()

we'll just talk about tasks

- means both processes and threads

---

http://www.makelinux.net/books/lkd2/ch03lev1sec3
https://en.wikipedia.org/wiki/Native_POSIX_Thread_Library

man pthreads

> Both of these are so-called 1:1 implementations,
> meaning that each thread maps to a kernel scheduling entity.
> Both threading implementations employ the Linux clone(2) system call.

http://en.wikipedia.org/wiki/Scheduling_%28computing%29

recap: scheduling goals

fairness	CPU share adequate for tasks' priority
load	no idle times when there is work
throughput	maximize tasks/time
latency	until first response/completion

recap: scheduling goals

- **fairness**
  - each process gets its fair share
  - no process can suffer indefinite postponement
  - equal time != fair (*safety control* and *payroll* at a nuclear plant)
- **load**
  - no idle times when there is work
- **throughput**
  - maximize tasks/time
- **latency**
  - time until first response/completion

http://lwn.net/Articles/254445/

recap: the problem

observe scheduling goals

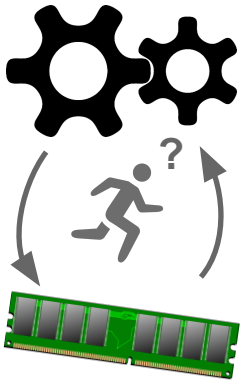
even in complex NUMA topology

approaches

keep task close to memory (scheduling vs. memory mgmt)
keep related tasks close to each other
avoid congestion of memory controllers/interconnects

keep in mind

overhead
short- vs. long-running tasks
shared memory (global, groups)



recap: the problem

when talking about NUMA

- still **observe scheduling goals**
- e.g. in supercomputing: high throughput

in other presentations: already heard about **possible approaches**

- **preserve memory locality**: keep task close
  - because takes longer to access remote memory
  - two ways to do this: scheduling (task placement) vs. mm
- **keep related tasks close**: if they share memory
- **avoid congestion** of mem controllers, interconnects
  - that would then be **bottleneck** for application

few things you should **keep in mind**

- **overhead**: if we want to make more complex decisions
  - have to arrive there somehow: probably also gathering data / calculating heuristics
  - scheduling invoked very frequently: is it worth the overhead?
- **short vs. long-running tasks**
  - applications where NUMA makes sense normally don't run for 50ms

    - short-running task: probably not worth rescheduling to different node
        - also not worth overhead gathering statistics, and making decisions
    - **empirical observation** that we found in multiple places
- **shared memory**
    - tasks not always isolated
    - share memory: global level (C lib) / task groups aka threads
    - latter ideally placed on same node

kernel development and academic science

academic research **seldomly** referenced

- almost never
- but there *are* theoretical considerations
  - mailing list discussions
  - the developers' experience

kernel development and academic science

how do the two mix?

no references to academic work

- mails
- discussions
- articles

instead: mailing list discussions serve as theoretical considerations

- we know such work exists (see Fabian Eckert's presentation)

https://www.cs.sfu.ca/~fedorova/papers/usenix-numa.pdf

related academic work

DINO

A Case for NUMA-aware Contention Management on Multicore System, Blagodurov, 2001

main concern: NUMA-agnostic task migrations
far more serious than remote access latency

mechanisms
scheduling: thread placement
memory migration: only move subset

✔ source published

✘ never announced on mailing list
esp. no patch sent

2001

DINO
avoid NUMA-agnostic migrations
thread placement

scheduling: predefined thread **classes**
based on **cache misses** / time
keep classes on one node

memory migration
migrate a fixed number K of pages
different strategies (pattern detection etc.)
empirically determined K which seems optimal

migrate memory too often
interconnect stress
migrate memory not often enough
memory controller stress

https://www.cs.sfu.ca/~fedorova/papers/asplos284-dashti.pdf

related academic work

Carrefour

Traffic Management: A Holistic Approach to Memory Placement on NUMA Systems, Dashti, 2013

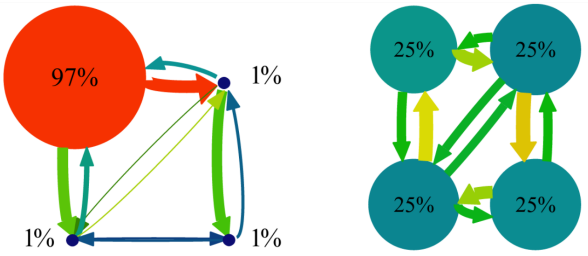
main concern: congestion on memory controllers and interconnects
not remote access costs per se

mechanisms
page co-location, interleaving, replication
thread clustering

✔ source published

✔ announced on mailing list

✘ no patch attached



Traffic Imbalance

some **overlap in authors**
same **basic assumption: remote access cost not the problem**

2013 -> worked on Kernel 3.6 (released end of 2012)

main concern: congestion of mem controller / interconnect

mechanisms: page co-location, interleaving, **replication**
thread clustering

“So even without improving locality (we even reduce it for PCA), we are able to substantially improve performance”

kernel development and academic science

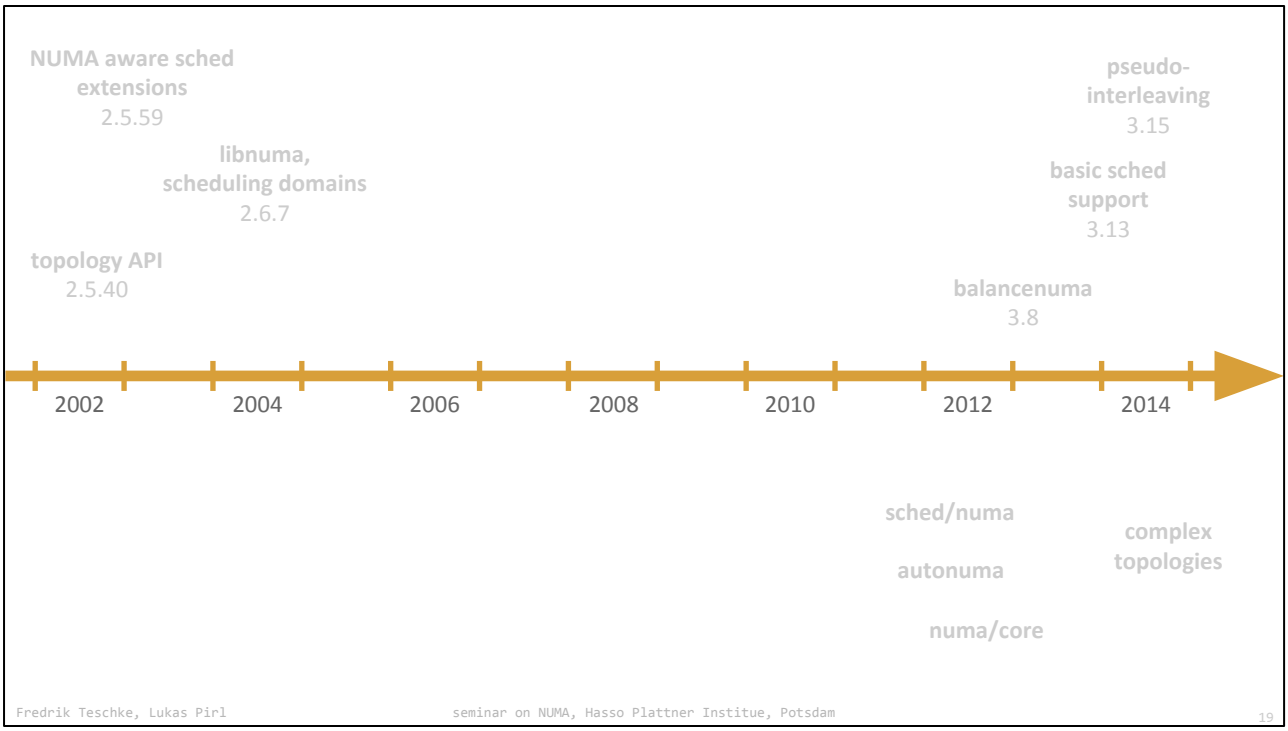
learning

no patch ⟶ no attention

stop writing papers, hack!

if you want to contribute to the Linux kernel

- if no patch submitted to kernel mailing list
- chances of receiving attention are low
  - again: formal requirements are very high
    - plain-text only
    - no attachments
    - only include text you are specifically replying to
    - patches directly pasted into an email
  - ignorance is high if violated



- 2002 → today
- gap 2006 – 2011
- dating of changes
  - where available: kernel release dates
  - otherwise: date of main article referring to patch set
- kernel version: contains merged code
  - = above timeline
- below the timeline = not merged into mainstream

<= 2.5.40 (<=2002) imagine…

no understanding of nodes

unaware of memory locations/latencies

no memory migration between nodes

no affinity

processing, memory allocations

- no understanding of nodes
  - unaware of memory locations/latencies
    - no memory migration between nodes
  - no affinity
    - processing, memory allocations

⇒

- performance of application may vary
  - system load
    - where is the process scheduled
  - may be all allocations remote
  - …
- basically, everything can happen!
- if system ends up unbalanced, no chance to fix this

2.5.40 Oct 2002 — topology API

rudimentary “discovery” of topology

- obtained from firmware
- supposed to map to any kind of system
- elements
  - processor (physical)
  - memory block
  - node

node: container for any elements

- **not** necessarily 1-1 mapping to hardware

but: no data on interconnects

- rudimentary “discovery” of topology
  - by McKenney, IBM
  - obtained from firmware
  - supposed to map to any kind of system
  - elements
    - processor (physical)
    - memory block
      - memory block: physically contiguous block of mem
    - node
      - node: container for any elements
      - not necessarily 1-1 mapping to hardware
- does not represent
  - attached hardware
    - NIC
    - IO controller
    - …
  - interconnects
  - how to pin process close to hardware?
    - manual?

symbol at bottom right: this was merged into the Kernel!

2.5.40 Oct 2002 topology API

asm/topology.h

```
int __cpu_to_node(int cpu);
int __memblk_to_node(int memblk);

unsigned long __node_to_cpu_mask(int node);


int __parent_node(int node); # /!\ supports hierarchies
```

- brief API overview
- __cpu_to_node(int cpu);
  - returns node the CPU belongs to
- __memblk_to_node(int memblk);
  - returns node the memory belongs to
- __node_to_cpu_mask(int node);
  - useful for pinning/affinity
- __parent_node(int node);
  - supports hierarchies!
- no distances/latencies

now you – as a developer – can

1. manually discover nodes and their CPUs/RAM
2. manually pin tasks to CPUs

- manually discover nodes and their CPUs/RAM
  - derive placement approach
- manually pin tasks to CPUs
  - provoke less migrations over nodes

2.5.59 Jan 2003 NUMA-aware scheduling

keep task & mem on same node

scheduler pools CPUs by node

```
int __cpu_to_node(int cpu);
```

assigns static home node per task

run & allocate memory here

initial load balancing

node with minimum number of tasks

policies: same node / new node if own memory mgmt. / always new node

- scheduler **pools** CPUs by node
  - 1st time active consideration of nodes
  - __cpu_to_node(int cpu);
- assigns static home node per task
  - run & allocate memory here
- **initial load balancing**
  - node with minimum number of tasks
  - policies
    - same node
    - new node if own memory mgmt.
    - always new node
- system might get unbalanced over time

2.5.59 Jan 2003 NUMA-aware scheduling

dynamic load balancing

invoked frequently per CPU

- idle CPUs: every tick
- loaded CPUs: every 200ms

⇒ “multi-level balance”:

1. inside node
2. across nodes

```
L = local_node();

# regular load balancing as for multicore
# (O(1) scheduler):
balance_node(L);

N = most_loaded_node();
C = most_loaded_cpu(N);
if load(L) <= system_load()
      steal_tasks_from_cpu(C);
```

dynamic load balancing

- invoked frequently per CPU
  - idle CPUs: every tick
  - loaded CPUs: every 200ms

⇒ “multi-level balance”:

1. inside node
2. across nodes

now you – as a developer – can

lean back and trust the kernel

(but you should tune manually for long-running tasks)

- **compute load** probably **balanced well**
- still, main **problem**:
  - memory spreads out
    - CPU affinity might help
  - “no return”

2.6.7 Jun 2004 libnuma

new kernel API

set memory policy for process/memory area

```
BIND
PREFERRED     # prefers a specific node
DEFAULT       # prefers current node
INTERLEAVE
```

libnuma

- by Andi Kleen (Suse)
- syscalls
- library
- command-line utility

- mem alloc policies
  - BIND
    - set specific node
  - PREFERRED
    - prefers a specific node
  - DEFAULT
    - prefers current node
  - INTERLEAVE
    - only on nodes with decent-sized memory

- home node == “preferred”

- adds flexibility

2.6.7 Jun 2004 schedu…ains

minimize cost of moving task (& mem)

put CPUs in hierarchy

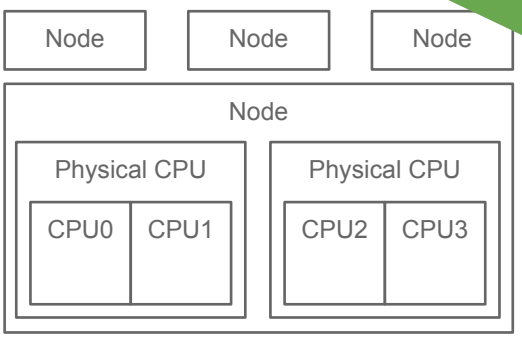
task migration cost not a constant

scheduling policy

HT, core, CPU, node

generalized approach

traverse group hierarchy bottom → top
at each level: balance groups?
domain policy influences decision
prefer balancing at lower level



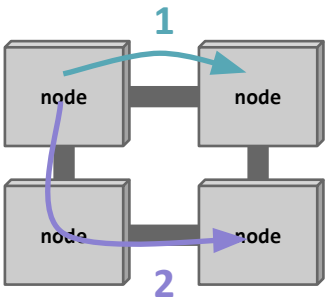
- levels
  - **hyperthreading**
    - share all caches
  - **cores** have own caches
  - **node**: own memory
- balancing intervals
  - HT CPU: every 1-2ms
    - even small differences
  - physical CPU: less often
    - rarely if whole system busy
    - process loses cache affinity after few ms
  - node: rarely
    - longer cache affinity
- enhanced scheduling approach
  - traverse hierarchy bottom → top
  - at each level: balance groups?
  - domain policy influences decision
  - prefer balancing at lower level

2.6.?? node distance

unclear when introduced exactly

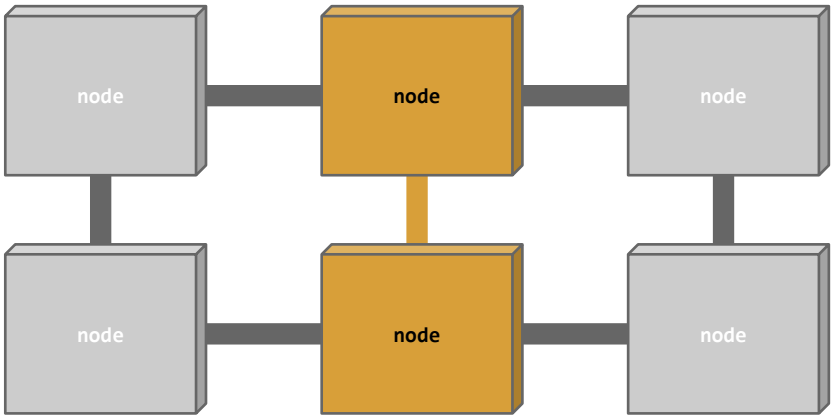
obtained from ACPI 2.0

System Locality Information Table



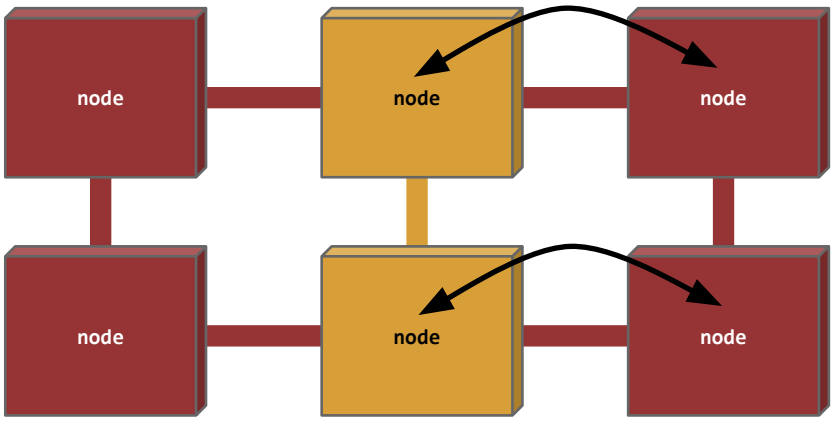
- distance between nodes obtainable from ACPI
  - SLIT - System Locality Information Table
- apparently not used for node balancing
  - à la “if another node required, take a closer one”
  - why?
    - track access patterns better?
      - DINO, Carrefour
    - highly app-specific?
      - assumption same parent == same data might be wrong
        - ex. Linux’ “init” process
  - even though: knowing the distance is not enough

knowing the distance is not enough…



- app needs threads on two nodes
  - (concurrency > CPUs/node)

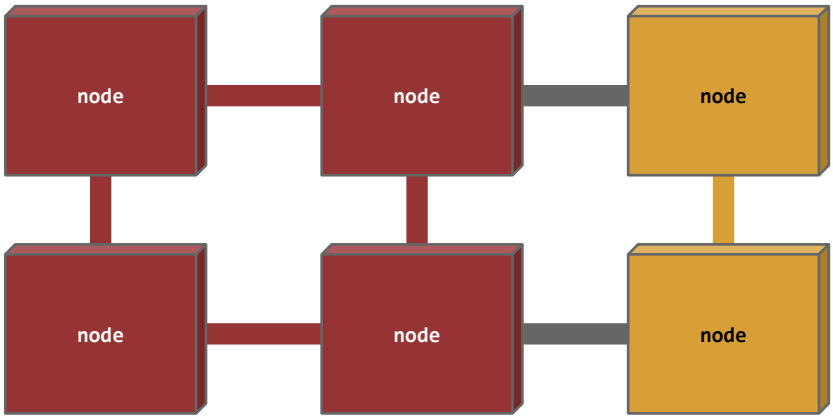
knowing the distance is not enough…



- another app needs 4 nodes
- scheduled on idle nodes

- **bad:** 4-node load separated by 2-node load
- swap 2 to relaxe interconnects

knowing the distance is not enough…



- resulting, better placement

⇒

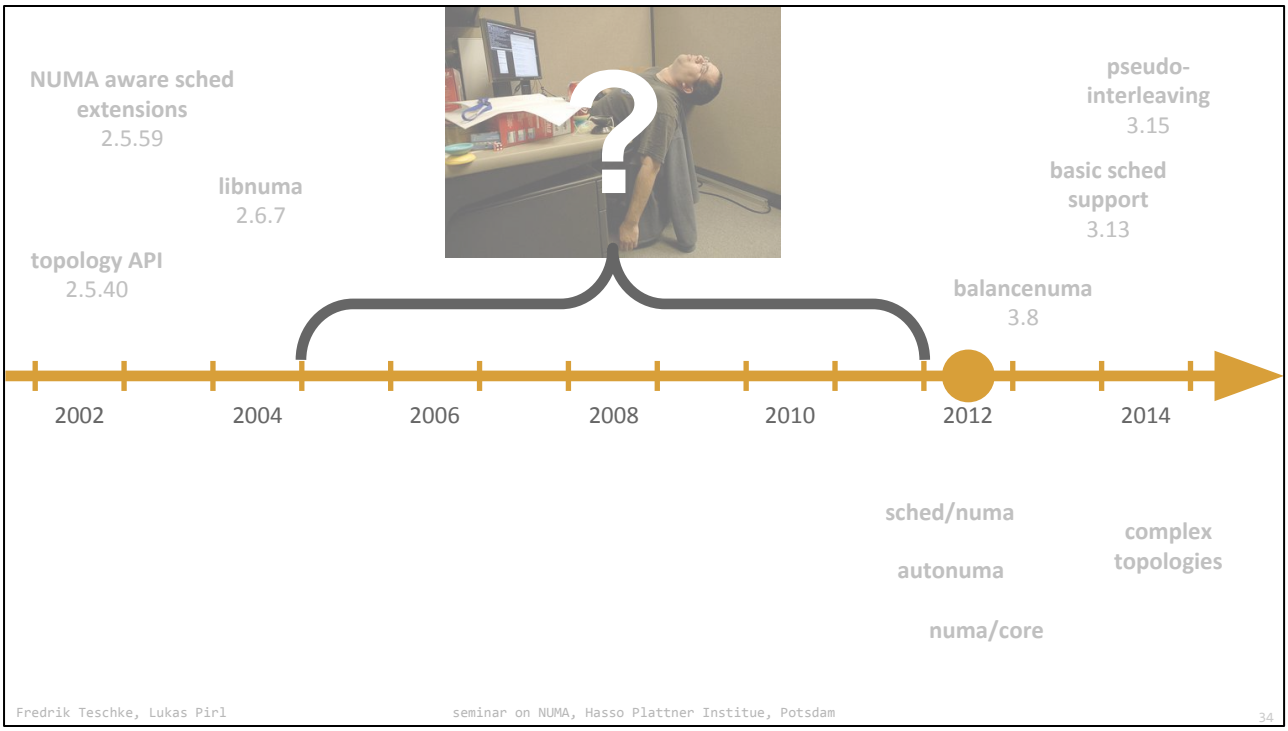
- placement complex
  - esp. for not fully connected
- a lot of work ahead

now you – as a developer – can

lean back and trust the kernel

(but still… think about the desired **memory allocation policy** and **set it manually**)

- memory allocation policies should be set
  - for long-running
  - allocation-intense tasks



timeline: gap of 7 years

groundwork is laid

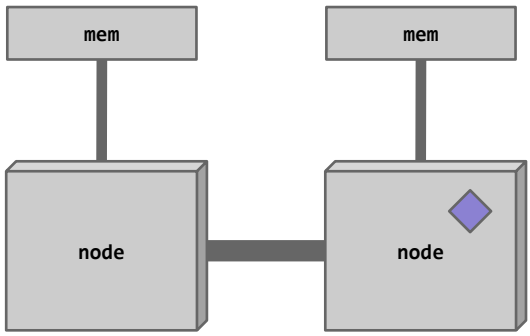
- **API calls to** read topology
- **memory policies:** NUMA-aware allocation
- **scheduler knows** balancing between **NUMA** nodes is more **expensive**
  - will try to avoid that

sounds good?

- apparently thats what most people thought
- **7 year gap**
- but as we will see: **still plenty that is missing**

and continue in **2012**
sched/numa

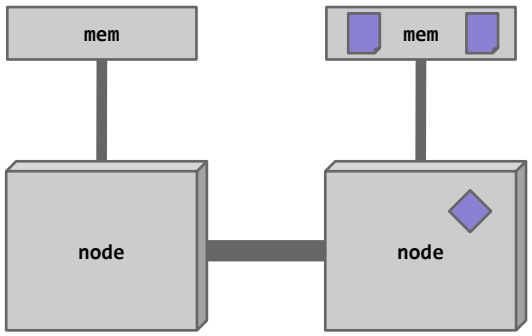
a typical long-running computation…



a typical long-running computation…

process starts main controlling thread

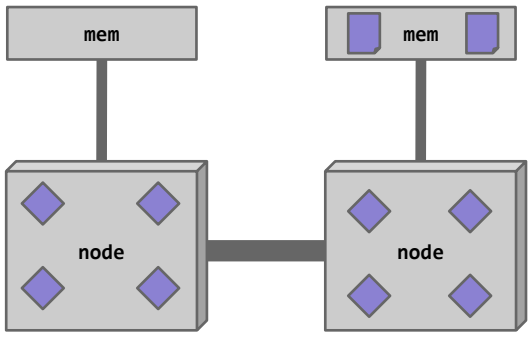
a typical long-running computation…



process loads its data for computation

allocations done where it runs (DEFAULT)

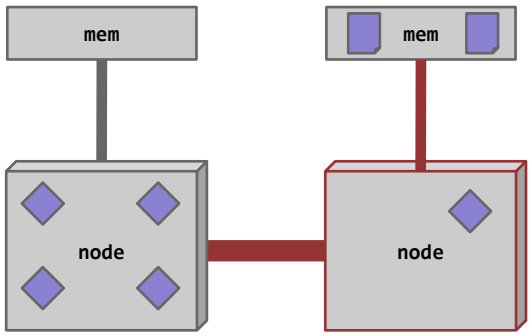
a typical long-running computation…



process starts worker threads

due to load: some scheduled on other node

a typical long-running computation…



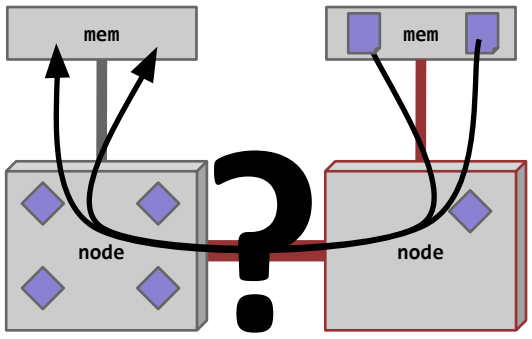
lets say some workers finish early

e.g. input sanitizers: finished cleaning up input

what happens: spread out after all

unnecessary load on interconnects

a typical long-running computation…

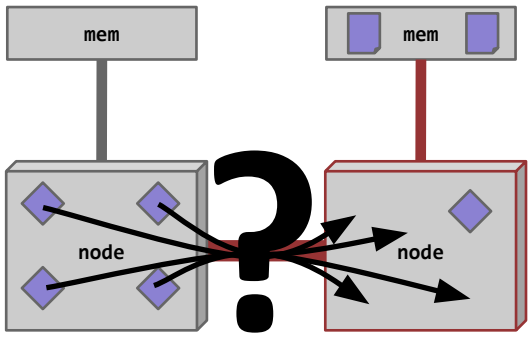


what possibilities do we have?

remember basic approaches: mm vs. sched

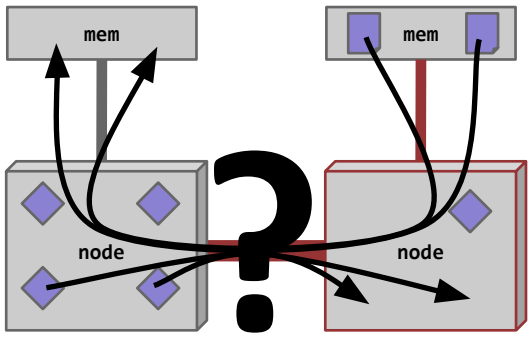
so we could migrate the memory

a typical long-running computation…



or reschedule the threads

a typical long-running computation…



or maybe do a combination of both

http://lwn.net/Articles/486858/

sched/numa Feb 2012 — the challenge

tasks scheduled on varying nodes

but

memory allocated where task runs

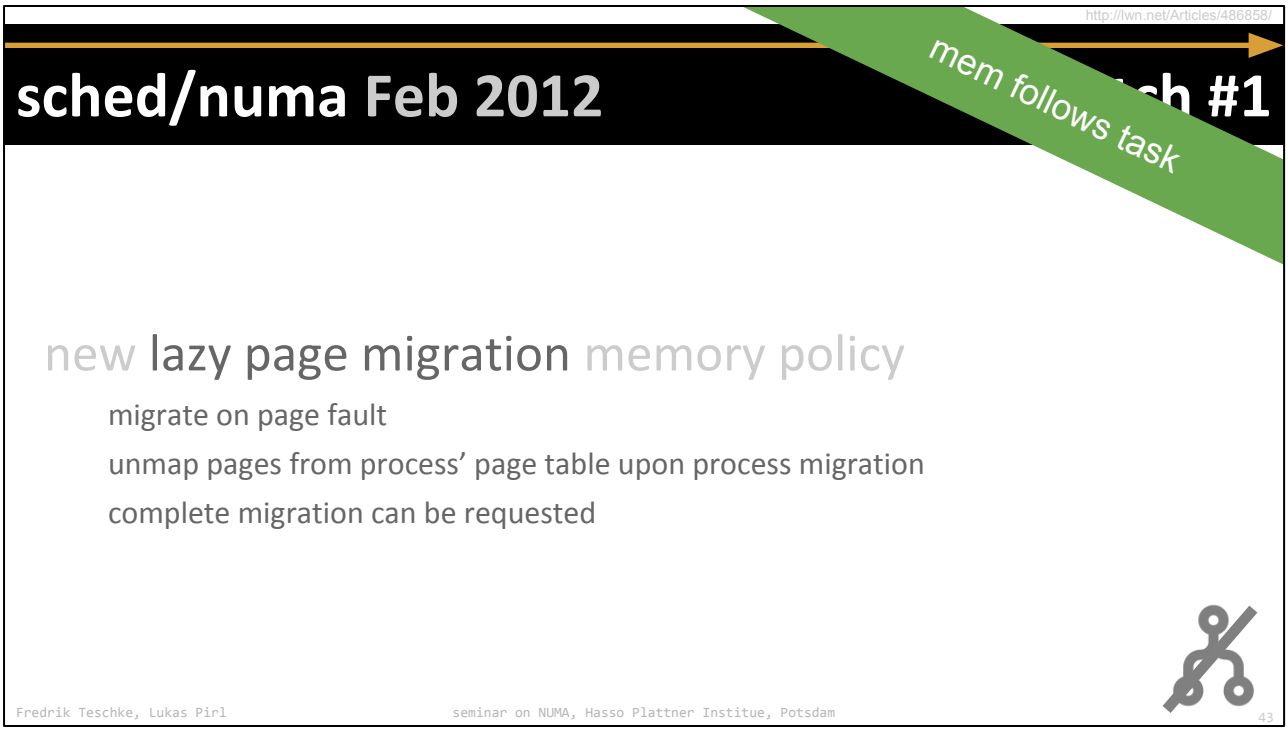
⇒ memory spreads over nodes

esp. for long-running, memory-intense tasks

sched/numa Feb 2012
the challenge

this was **just one scenario**
but represents what may happen: **memory spread out** over nodes

especially if tasks run for long time
and are memory intense



sched/numa Feb 2012

first possibility: migrating memory

- tackles two questions
  - when
  - how to do that efficiently

when: on page fault

- page still in page table
- but marked as not present (concept we will see again later)
- this bit is set:
  - when task is migrated to different node
  - or when task explicitly requests migration of all its memory

how to do it **efficiently**

- so page is only migrated to node when requested
  - by fault handler
- this spreads load out over time
  - e.g. no dedicated kernel thread that does batch-migrations
- and only migrates pages that are actually used

effectively: mem follows task

http://lwn.net/Articles/486858/

sched/numa Feb 2012 ...ch #2

task follows mem

load imbalance might change home node

ex: a lot non-local allocations for a tasks

expensive: only tasks running >=1s

lazy migration to new home node

that was mm

now scheduling part

- so far: static home node
- scheduler tried to keep task there (and allocate mem there)

but as seen: situation may change

- e.g. lots of remote memory
- then assign new home node for task
- and request lazy migration for mem on other nodes

this is the task follows memory part

http://lwn.net/Articles/486858/

sched/numa Feb 2012 ...ch #3

tasks w/ shared mem on same node

define NUMA groups

new system call

share home node

define memory per NUMA group

bind memory to NUMA group

set allocation policy

also something novel

NUMA groups

- declar group of tasks as NUMA group
- via system call

effect

- they share the same home ndoe
- if one is migrated, all are

you can actually bind memory to the group

what is this good for?

- tasks w/ shared mem (e.g. threads) run on one node (hopefully)

http://lwn.net/Articles/488709/

autonuma Mar 2012

things will spread out (Andrea Arcangeli)

clean up afterwards

- sched: migrate task?
- mm: migrate page?

how to decide?

- maintain statistics using page faults
  - per-task counters: pages per node
  - per-page field: last node to access

autonuma Mar 2012

new player: Andrea Arcangeli

- also redhat employee at the time

things spread out anyways

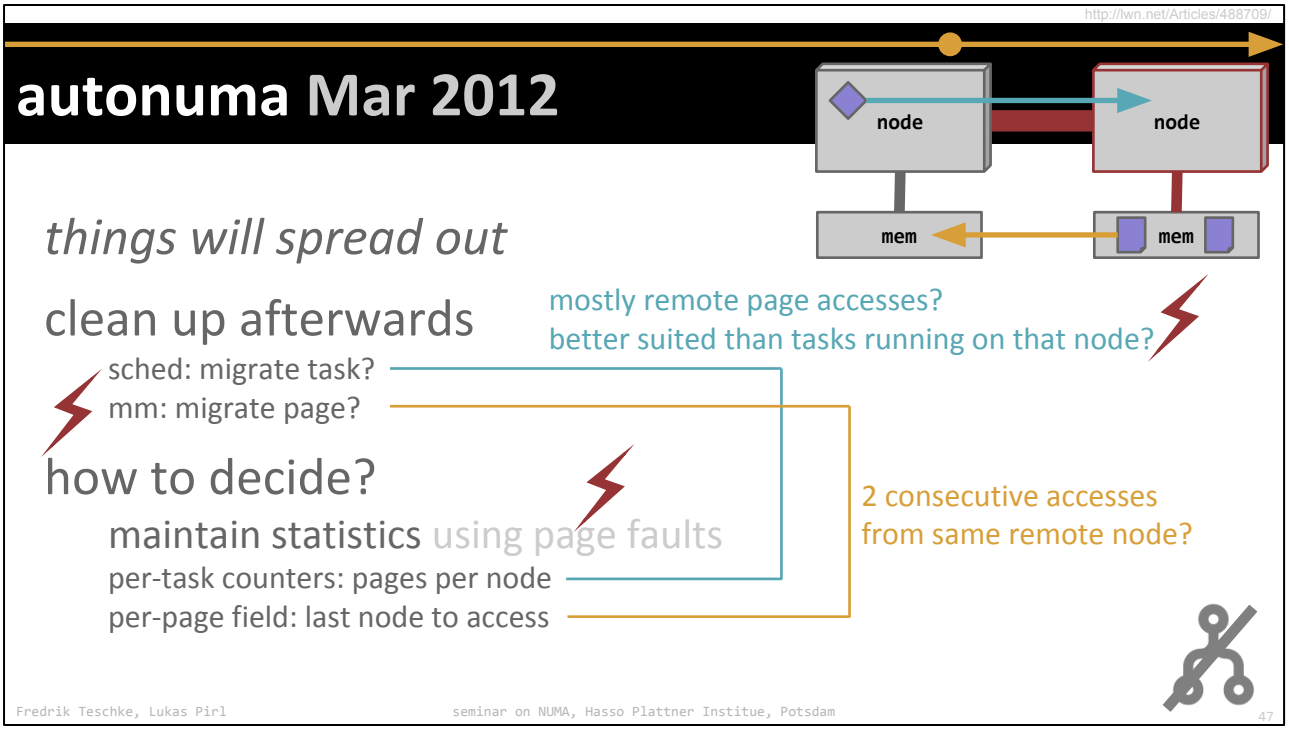
- remember example just now
- e.g. tasks that do not fit on one node
- basically says: forget the home node/ preferred node

different approach: **clean up**

- two possibilities: sched vs mm
- migrate task or page

decide based on **page access statistics**

- gather using page faults
  - k-thread periodically marks anonymous pages as “not present”
  - upon access: fault generated
  - in fault handler update statistics
  - for each task record: how many pages on each node
  - for each page record: what was last node to access it



migrate task?

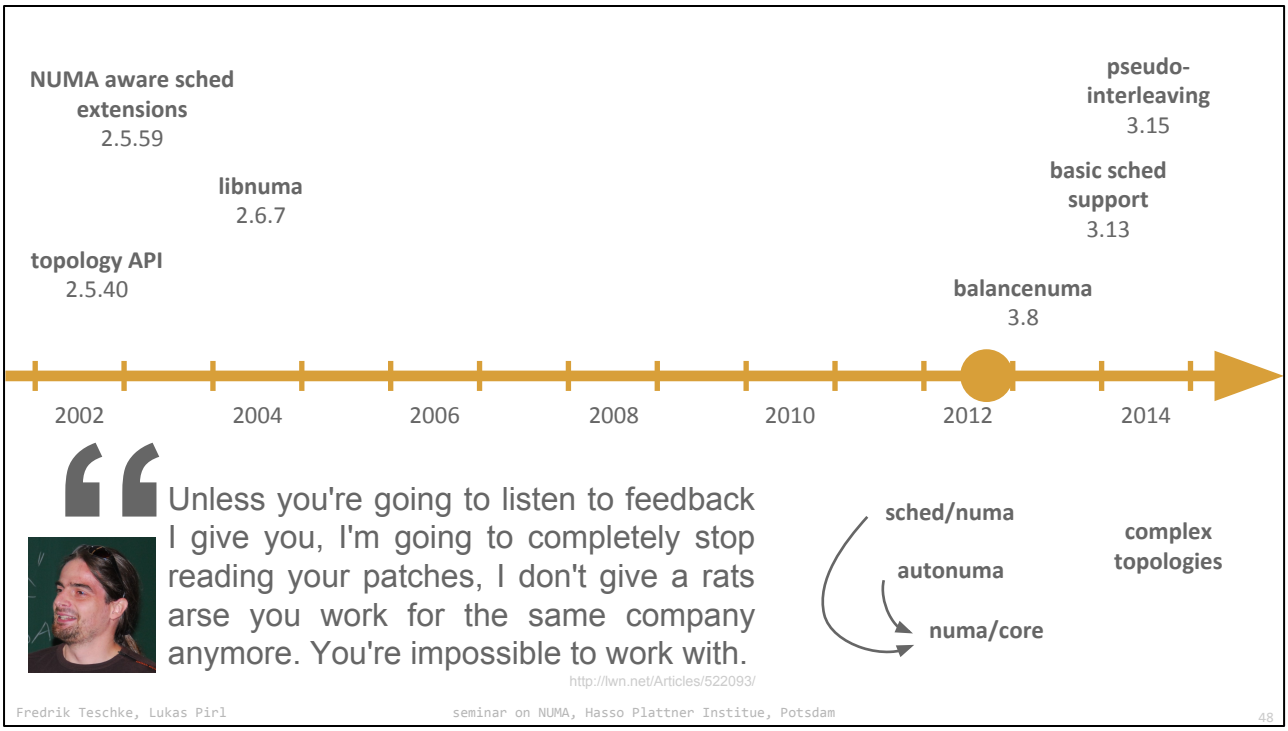
- if mostly remote page accesses
- and other tasks currently running on that node are not that well suited

migrate memory?

- b/c memory may be spread out: can only migrate task to largest part
- heuristic: if on 2 subsequent faults -> access from same remote node
  - then add to migration queue

problems (pointed out by Peter)

- kernel worker threads used
  - to scan address space -> force page faults
  - and to migrate queued pages
  - e.g. if system slow: now direct accountability -> why is it slow?
- for-each-CPU loop in scheduler
  - in schedulers hot path
  - doesn't scale with # CPUs



timeline

discussion btw Peter and Andrea two **grew a bit out of hand**

- Unless you're going to listen to feedback I give you,
- I'm going to completely stop reading your patches,
- I don't give a rats arse you work for the same company anymore.
- You're impossible to work with.

apart from that: **short comparison** of sched/numa and autonuma

- **sched/numa**
  - **avoid separation** in first place -> home node
    - move mem with task (**lazy**)
    - possibly **change home node** of task
  - dev can **explicitly define NUMA group** -> share home node
- **autonuma**
  - **scleanup** afterwards
  - **statistics** gathering via **page faults**

next step: combination into numa/core

- maybe redhat stepped in
- Peter tried to combine the best of both

http://lwn.net/Articles/522093/, http://lwn.net/Articles/524535/

numa/core Oct 2012

combine existing ideas

lazy page migration (sched/numa)
page faults to track access patterns (autonuma)

modify some things

scan address space: proportional to task runtime http://article.gmane.org/gmane.linux.kernel/1392192
only if task gathered >1s runtime http://article.gmane.org/gmane.linux.kernel/1392189

add some new stuff

private vs. shared pages: analyze CPU access patterns http://article.gmane.org/gmane.linux.kernel/1392193
add last_cpu to page struct -> **auto-detect NUMA groups**
move memory-related tasks to same node

numa/core Oct 2012

combine existing ideas

- **lazy** page migration (sched/numa)
  - benefit: **less performance impact** when task is migrated
- **page faults** to **track** access patterns (autonuma)
  - determine ideal placement **dynamically**: no static home node

modify some things

- scan address space: proportional to task runtime
  - problem before: task w/ little work but lots of mem -> large impact
  - only if task gathered **>1s runtime**
    - ignore short-running (theory: don't benefit from NUMA aware placement)

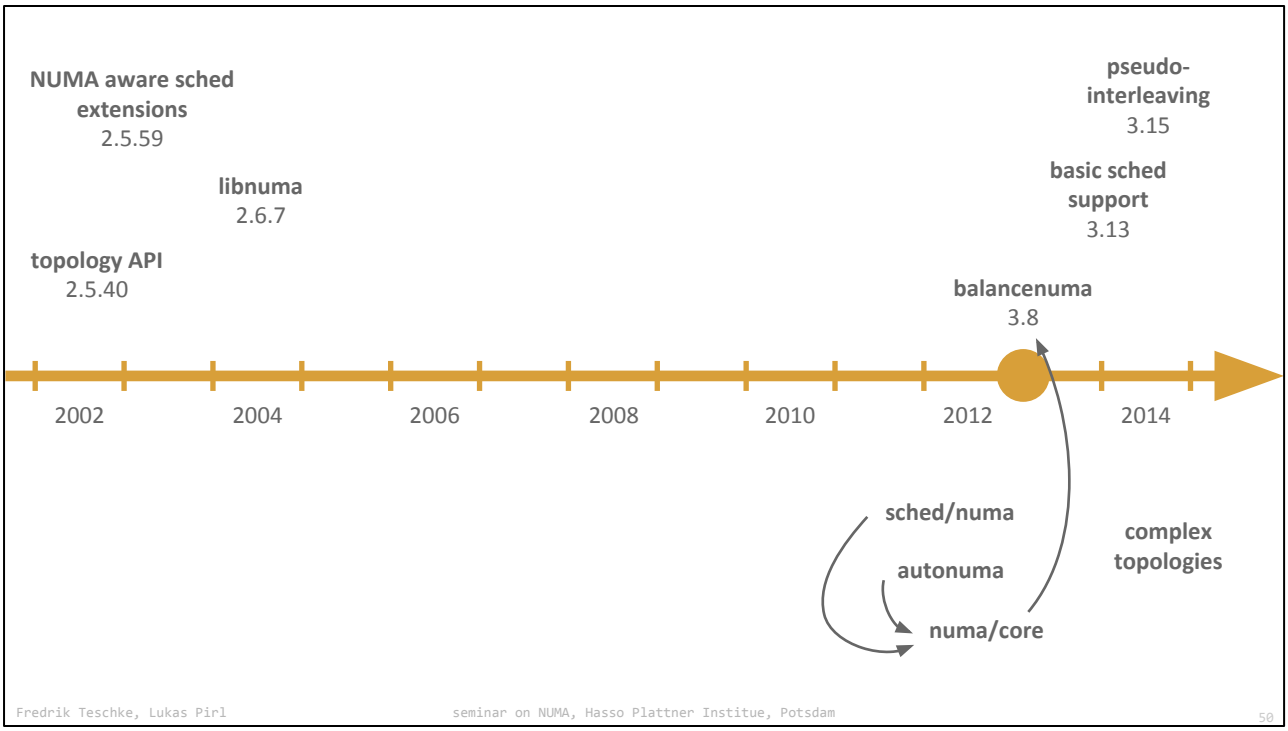
add some **new stuff**

- identify shared pages from CPU access patterns
- add **last_cpu** to **page struct** -> **auto-detect NUMA groups**
  - assume task remains on CPU for some time
  - page fault: accessed by other CPU == other task?
  - **instead of manually** defining them as before

    - try to move memory-related tasks to **same node**

actually **made it into linux-next**

- staging tree for next kernel release



timeline

while Peter and Andrea were arguing

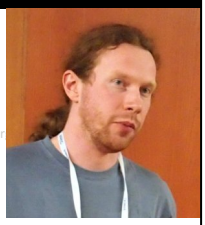
- other devs had noticed (Mel Gorman, IBM)

while Peter worked on numa/core

- Gorman worked on balancenuma

http://lwn.net/Articles/524977/, http://thread.gmane.org/gmane.linux.kernel/1392753

3.8 Feb 2013 balancenuma



objections to sched/numa and autonuma

http://thread.gmane.or
kernel/1389408

add basic infrastructure

lazy page migration, tracking via page faults
with some improvements
vmstats: measure benefit of policy

baseline policy **MORON** (Migrate On Reference Of pte_numa Node)
in future: test different policies (e.g. rebase sched/numa and autonuma on top)

balancenuma

Mel Gorman: **objections to implementation** of both approaches
but **also**: objections to **approaches themselve**s

- both **specific solutions** on how to schedule / move memory
- tested, but not widely tested on lots of NUMA hardware
- and not compared to many different approaches

his vision: compare more policies

- more of an **academic** approach
- first step: **make it easier** to **build & evaluate** such policies
- basic mechanisms can be shared btw policies

add basic infrastructure

- page fault mechanism
- lazy migration
- **vmstats** (virtual memory statistics)
  - **approximate cost** of policy

on top of this: implemented **baseline** policy **MORON**

- mem follows task

- migrates memory on page fault && remote access

his **suggestion for going forward**

- test other policies
- e.g. rebase sched/numa and autonuma onto this foundation

finally merged

- after 1 year of back and forth

---

pte: page table entry

sched/numa
    obscures costs
    hard-codes PROT_NONE as hinting fault even (should be architecture-specific decision)
    well integrated, work in context of process that benefits
autonuma
    kernel threads: mark pages to capture statistics
    obscures costs
    some costs: in paths that sched programmers are weary of blowing up
    performance tests: best performing solution.

now you – as a kernel hacker – can

build NUMA-aware policies

scheduling

memory management

that reuse basic mechanisms (e.g. lazy page migration)

evaluate your policies

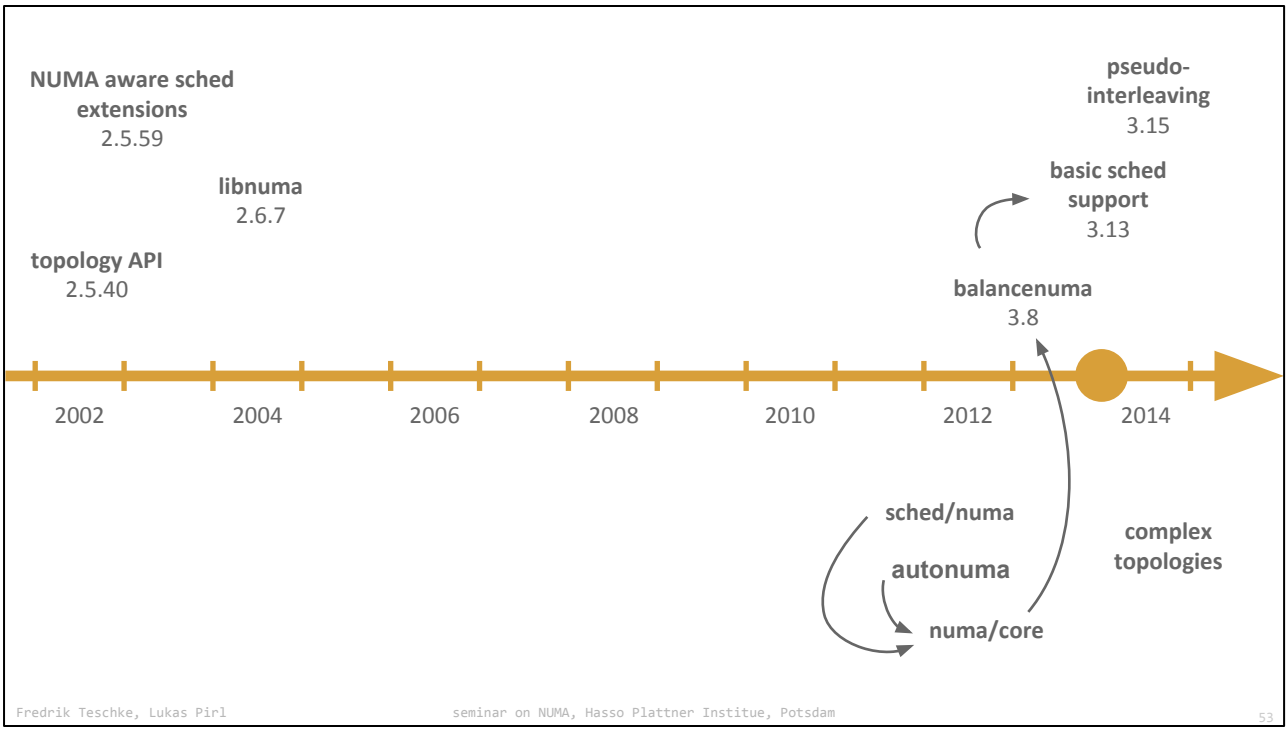
now you – as a kernel hacker – **can**

build NUMA-aware policies

- consider both
  - scheduling
  - memory management
- now made easier
  - reuse basic mechanisms (e.g. lazy page migration)

evaluate your policies

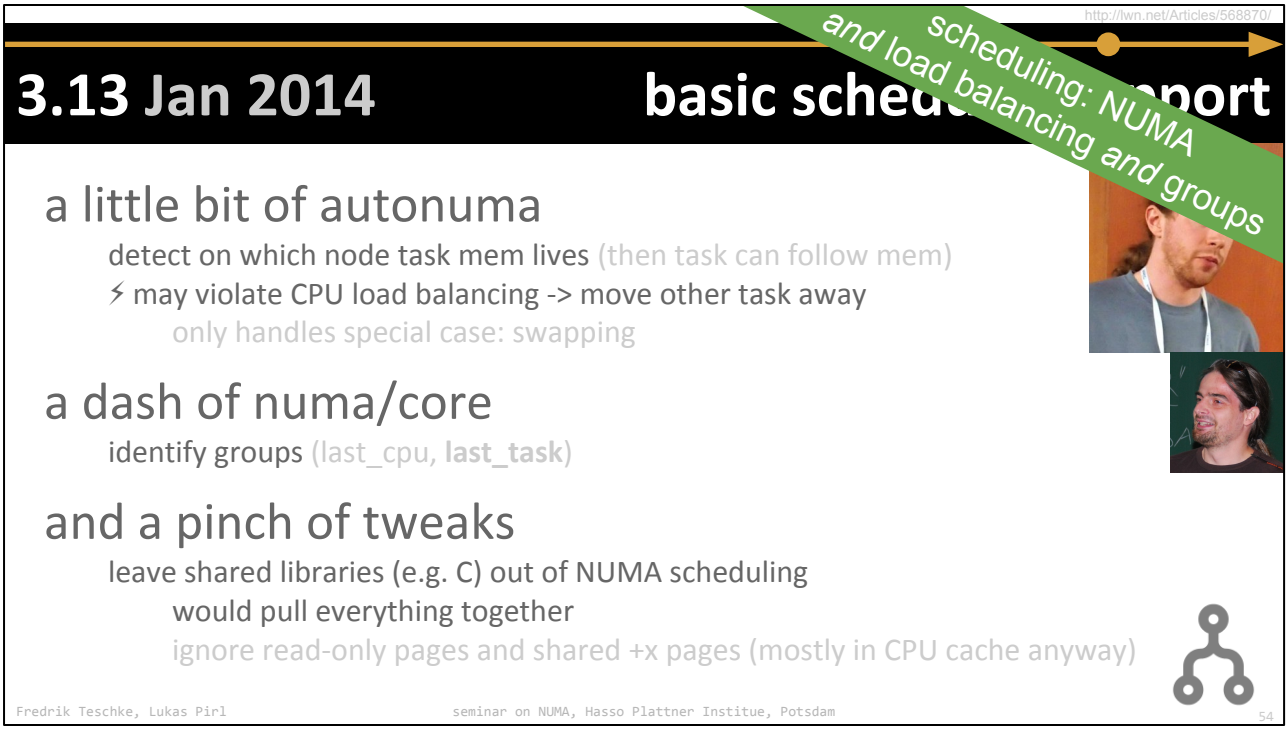
- compare them to existing



timeline

next step on top of balancenuma

- not only mem mgmt
- but also scheduling



basic scheduler support

Peter started pitching in again
reuse of more existing stuff

a little bit of **autonuma**

- **per task counters**: detect where task **mem lives**
- **problem**: NUMA scheduling **possibly in conflict** with **scheduling goal** of max. load
  - only handle **special case** for now: **swap** w/ other task that also benefits

a dash of **numa/core**

- agreed that **identifying groups** was a good thing
- but less heuristic: **remember** which **task** accesses page
  - not enough space in page_struct for full task id
  - use **bottom 8 bits**: collisions possible

and a pinch of **tweaks**

- **ignore shared libraries**
  - would pull everything together
- by ignoring read-only pages and shared executable pages

    - mostly in CPU cache anyway

summary

- NUMA-aware scheduling (not just mm)
- try to uphold load balancing goal
- and auto detect NUMA groups

also in Kernel!

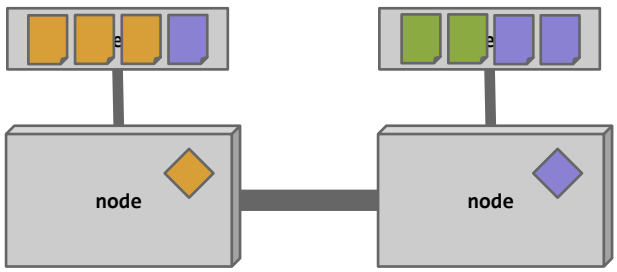
http://thread.gmane.org/gmane.linux.kernel/1631332

3.15 Jun 2014 pseudo-interleaving

a new tweak

workload (e.g. group) > 1 node

so far: mem distribution between nodes random



pseudo-interleaving

also already in kernel

basically yet **another tweak**

- for special case: **workload** (e.g. group) **> 1 node**
- if that happens, then so far
  - mem distribution btw nodes is random

example

- begin with one task (purple)
- starts allocating mem
- among that mem also some that will be shared by other task (green)
  - e.g. threads in same process
- maybe at some point mem spills over into other node
- then other task that shares some of the mem comes (orange)
  - scheduled on other node (e.g. b/c of load)
- starts allocating memory
- now not ideal distribution
-

goals

keep private mem local to each thread
    avoid excessive NUMA migration of pages
    distribute shared mem across nodes (max. mem bandwidth)

how-to
identify active nodes for workload
    balance mem lazily btw these

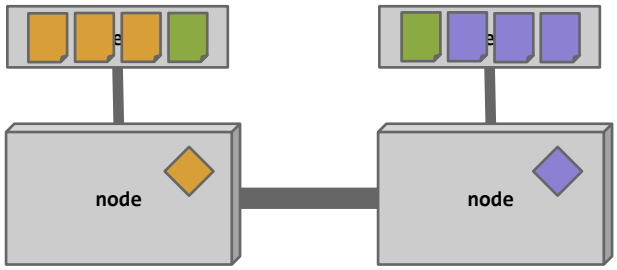
http://thread.gmane.org/gmane.linux.kernel/1631332

3.15 Jun 2014 pseudo-interleaving

a new tweak

workload (e.g. group) > 1 node

so far: mem distribution between nodes random



what would really be **ideal**

- **private pages local** for each task
- **shared pages distributed** evenly
  - reduce congestion of interconnect

http://thread.gmane.org/gmane.linux.kernel/1631332

3.15 Jun 2014 pseudo-interleaving

goals

keep private mem local to each task
avoid excessive NUMA migration of pages
distribute shared mem across nodes (max. mem bandwidth)

how-to

identify active nodes for workload
balance shared memory lazily between these

these are exactly the **goals** of this patch

- keep private mem local to each thread
- avoid excessive NUMA migration of pages (back and forth)
- distribute shared mem across nodes (max. mem bandwidth)

how to achieve that?

- identify active nodes for workload
- balance shared mem lazily btw these

now you – as a developer – can

lean further back

kernel will try to optimize
also for NUMA groups (e.g. threads)
and even if workload > 1 node

...or you can still manually tune

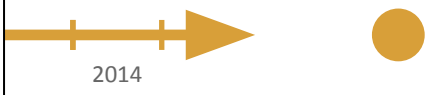
now you – as a developer – can

lean further back

- kernel will try to optimize
- also for NUMA groups (e.g. threads)
- and even if workload > 1 node

...or you can still manually tune

future work



http://thread.gmane.org/gmane.linux.kernel/1808344

future: complex topologies (2014)

recap scheduling domains:
HT, cores, node

What about node topologies?

scheduling domains and complex topologies

- elements in one hierarchy level (node level)
  - might not be equally expensive to migrate to

http://thread.gmane.org/gmane.linux.kernel/1808344

future: complex topologies (2014)

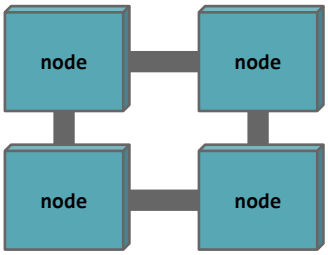
mesh topology

connection might through other nodes

hops between two nodes > 2

⇔

∃ intermediate node



mesh topology

- topology does not really matter
  - there is always a neighbor with distance = 1
- distance straight forward
- ⇒ no new domain hierarchy

http://thread.gmane.org/gmane.linux.kernel/1808344

future: complex topologies (2014)

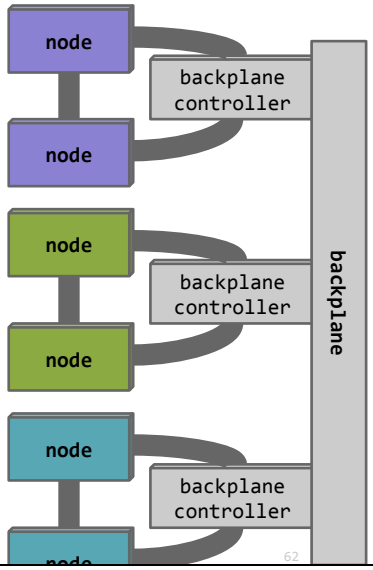
backplane topology

new scheduling domains

nodes in same group

⇔

both have same number hops to all other nodes



ex: backplane toplogy

- controllers: nodes w/o memory
  - cannot run tasks
- problems
  - controllers add 1 to distance
  - controllers in same domain as nodes
    - but cannot run tasks
- distances for all combinations of nodes
  - new scheduling domain
    - groups of nodes
    - nodes with same distance to all other nodes

outlook

notes from the ***Storage, Filesystem, and Memory Management Summit*** 2014

feedback needed!

performance, problems, enhancements, …

Big chance! Devs seldomly say “What could be better? I’ll implement it!”

notes from *Storage, Filesystem, and Memory Management Summit 2014*

- feedback needed!
  - performance, problems, enhancements, …
  - devs are willing to improve for others problems!
    - this is rare!

outlook

notes from the ***Storage, Filesystem, and Memory Management Summit*** 2014

4-node system close to optimal

performance drop for more nodes

page access tracking too expensive?
need more awareness of topology?

performance test highly individual

a benchmark would be an enrichment
(your chance to get famous!)

- 4-node system
  - close to optimal
- 4+ nodes
  - bad performance
  - page access tracking too expensive?
  - need more awareness of topology?
  - not fully meshed
- performance test highly individual
  - a benchmark needed
  - possible?
  - highly app specific

outlook

notes from the ***Storage, Filesystem, and Memory Management Summit*** 2014

page cache pages (IO cache) still location-unaware

good or bad?

force reclaim of memory for page cache?
- page cache saves IO **but** swapping can eliminate benefits

introduce page aging?
- *unused* pages swap out in favor for IO cache
- useful pages stay in memory
- more cross-node traffic (page cache is interleaved)

IO cache

- location unaware
- force free of memory for page cache
  - swapping vs. uncached IO
- page aging
  - swap out unused pages
  - page cache is interleaved

outlook

notes from the ***Storage, Filesystem, and Memory Management Summit*** 2014

add IO awareness

prefer nodes with corresponding adapter

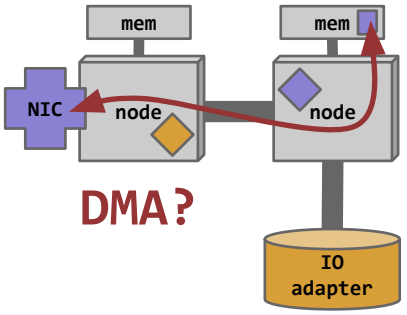
group networking processes?

add awareness of which node holds NIC

“swap” to other nodes

in case of low free memory

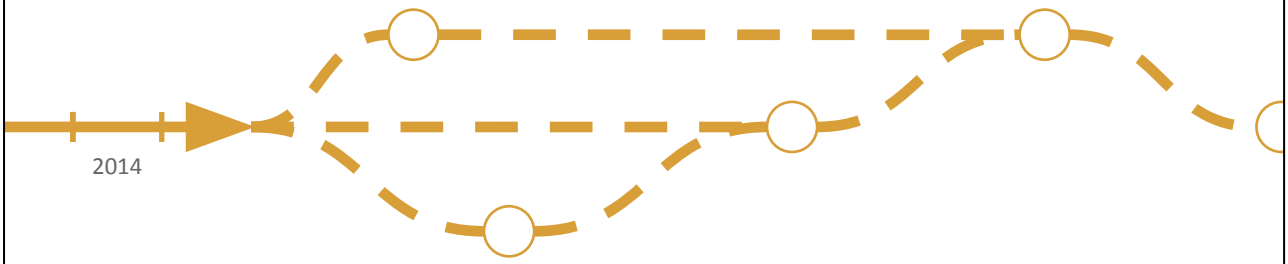
still, swap to disk if all nodes low on free memory



IO / device awareness

- group network processes
- group IO-heavy processes
- multi-level swap
  1. to other nodes
  2. to disk

future work - much

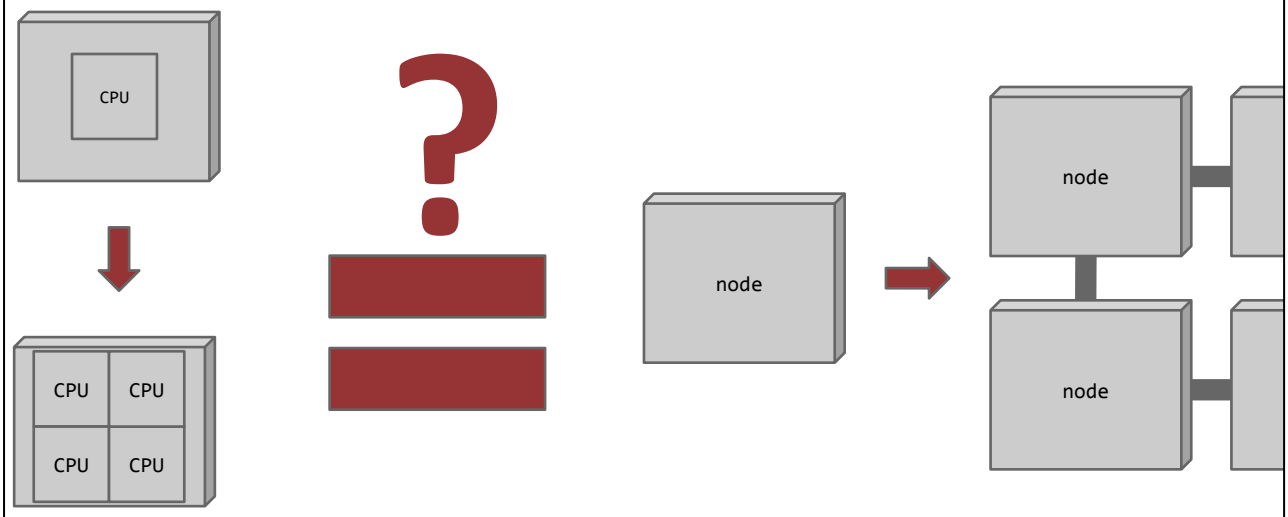


much to to
many possible ways
again:
- test
- feedback
- develop

by Tagxedo

Questions?
… or per email.

single → multi-processing ≈ SMP → NUMA ?



- SMP <-> NUMA
- caches should be warm <-> memory should be close
  - HT <-> same node
  - migration costs
- largely done by kernel (from the beginning?) <-> needs manual optimization