

Large Pages May Be Harmful on NUMA Systems

Fabien Gaud

Simon Fraser University

Baptiste Lepers

CNRS

Jeremie Decouchant

Grenoble University

Justin Funston

Simon Fraser University

Alexandra Fedorova

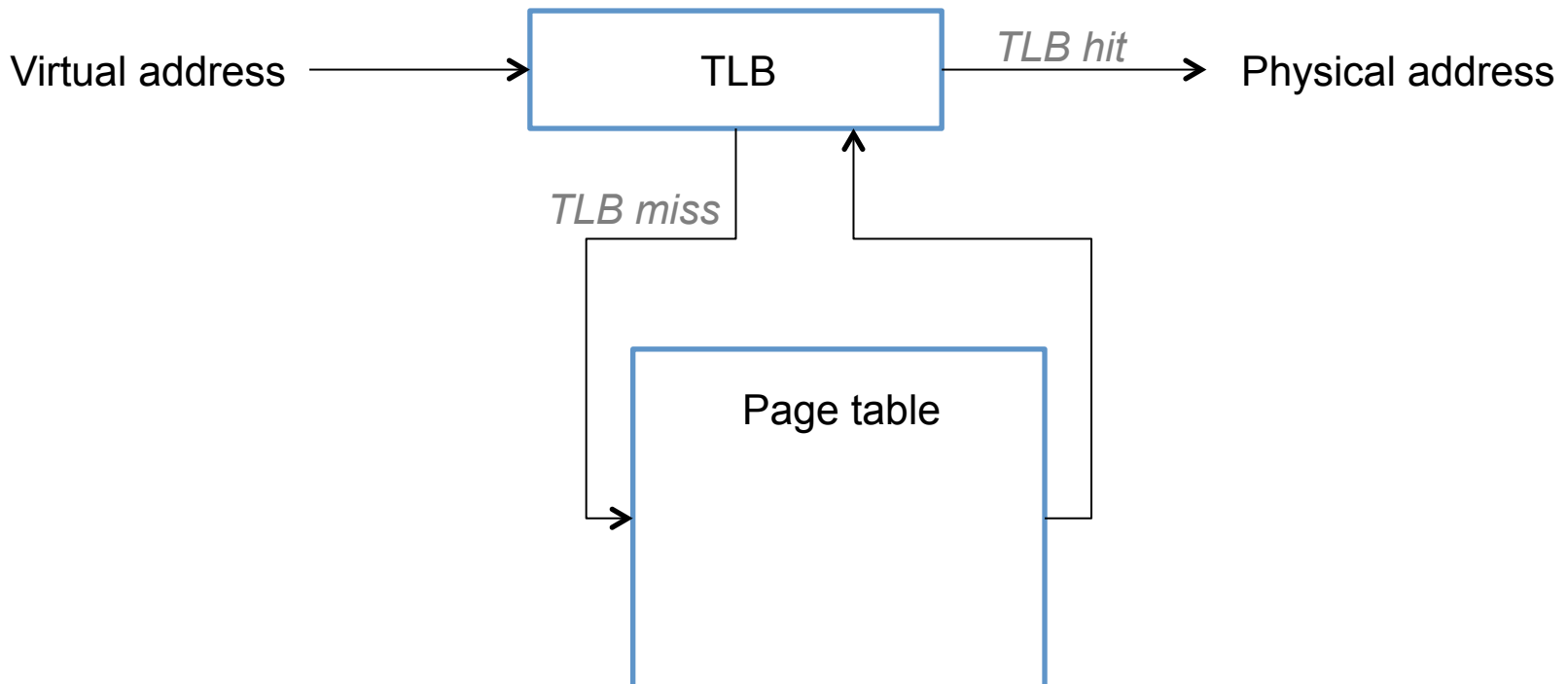
Simon Fraser University

Vivien Quéma

Grenoble INP

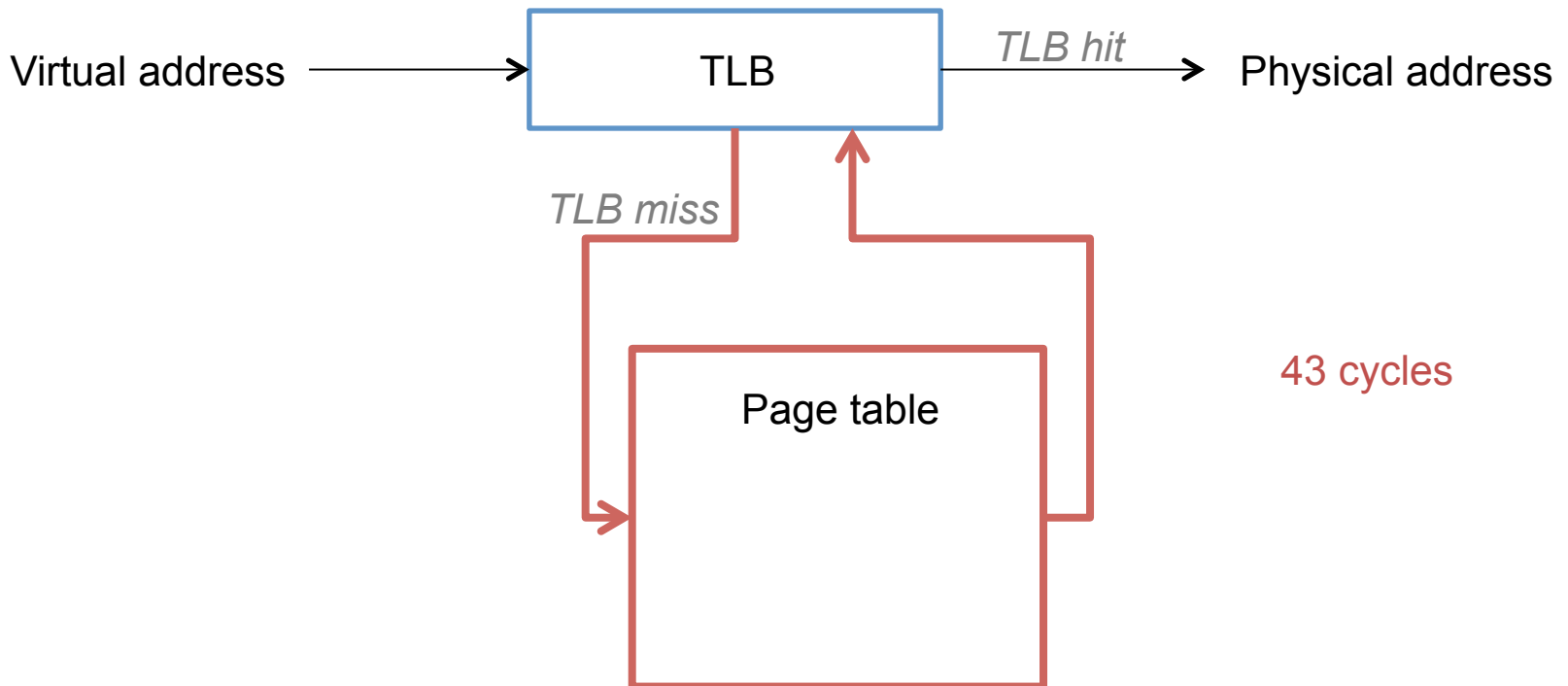


Virtual-to-physical translation is done by the TLB and page table



Typical TLB size: 1024 entries (AMD Bulldozer), 512 entries (Intel i7).

Virtual-to-physical translation is done by the TLB and page table



Typical TLB size: 1024 entries (AMD Bulldozer), 512 entries (Intel i7).

To reduce the number of TLB misses, developers can use “large pages”

Page size	512 entries coverage	1024 entries coverage
4KB (default)	2MB	4MB
2MB	1GB	2GB
1GB	512GB	1024GB

In Linux:

- Manually: `mmap(..., flags | MAP_HUGETLB)`
- Automatically: using Transparent Huge Pages (THP). THP uses 2MB pages for anonymous memory and clusters groups of 4K pages periodically.

Large pages known advantages & downsides

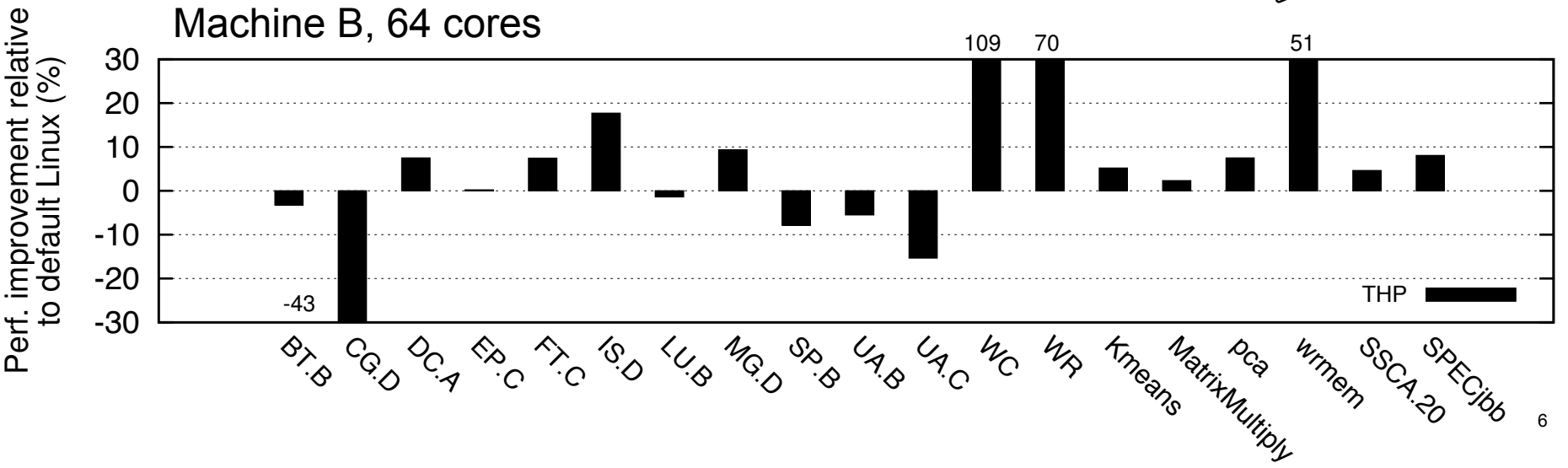
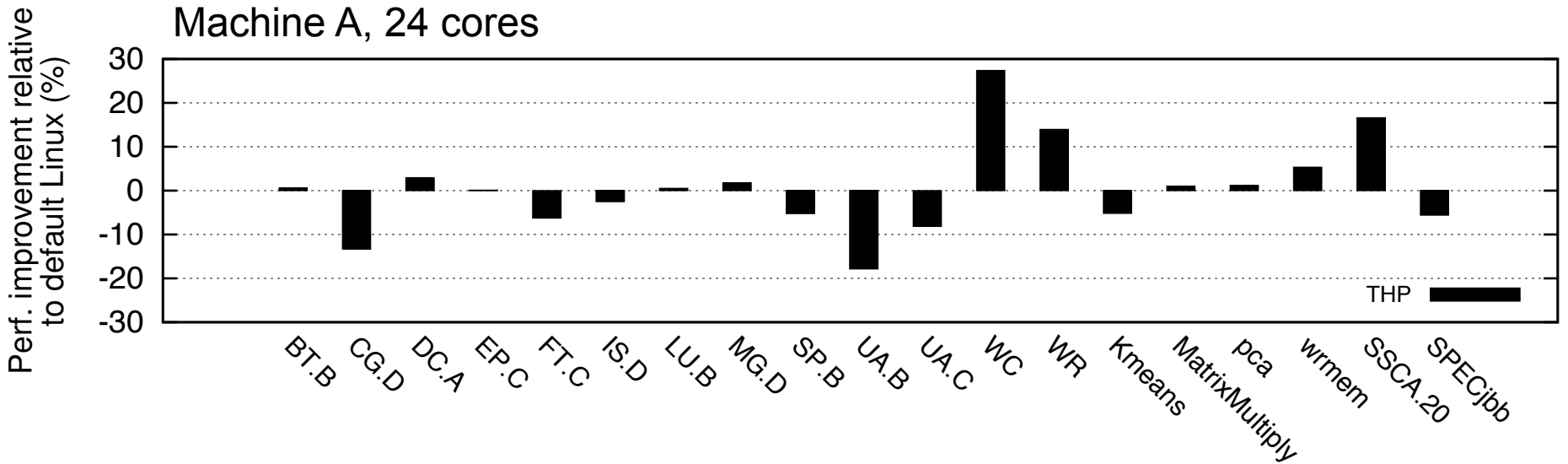
Known advantages:

- Fewer TLB misses
- Fewer page allocations (reduces contention in the kernel memory manager)

Known downsides:

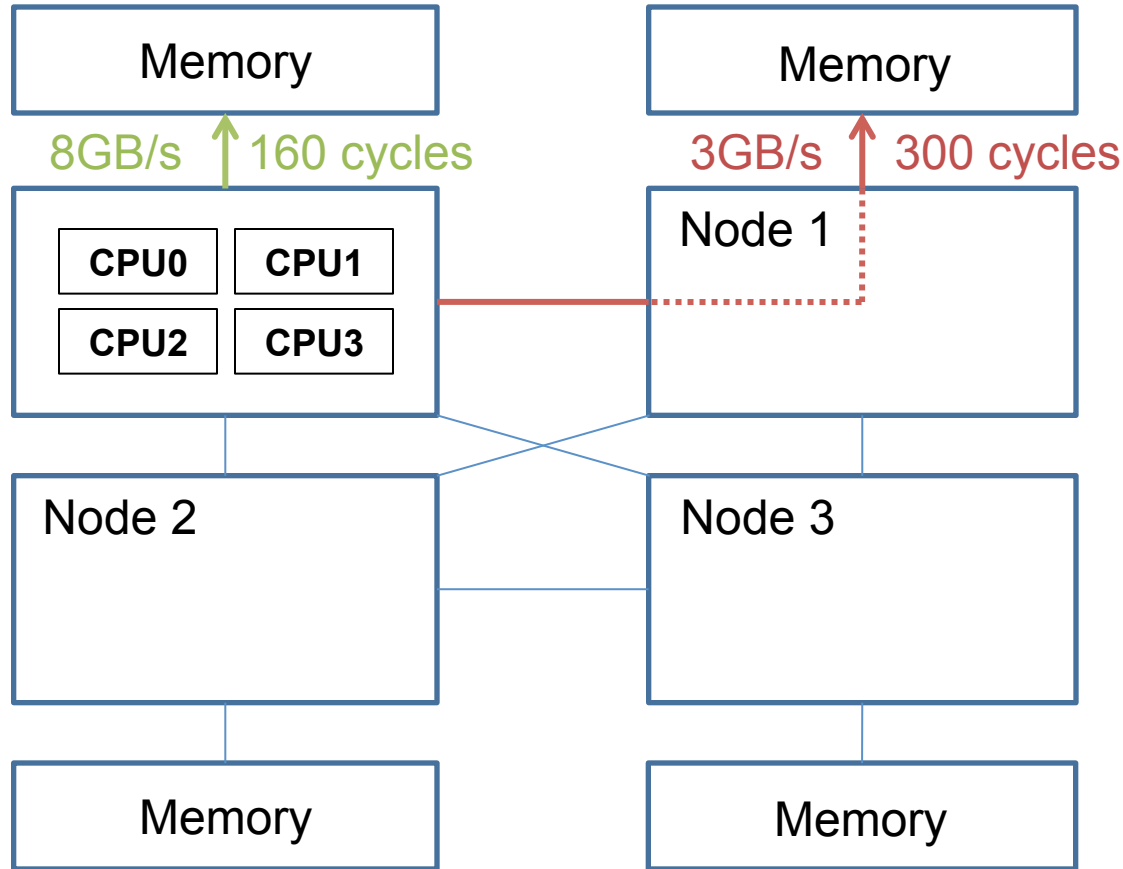
- Increased memory footprint
- Memory fragmentation

New observation: large pages may hurt performance on NUMA machines



Machines are NUMA

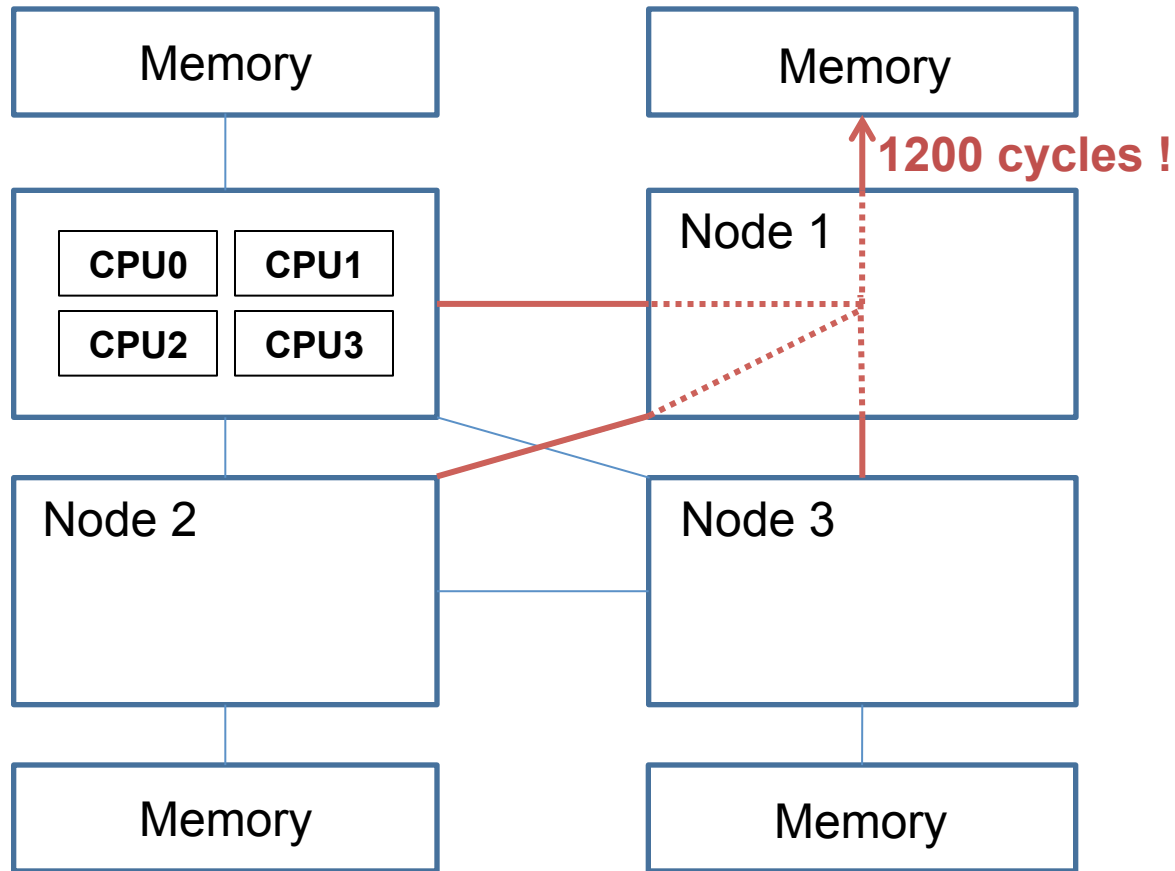
Remote memory accesses hurt performance



2

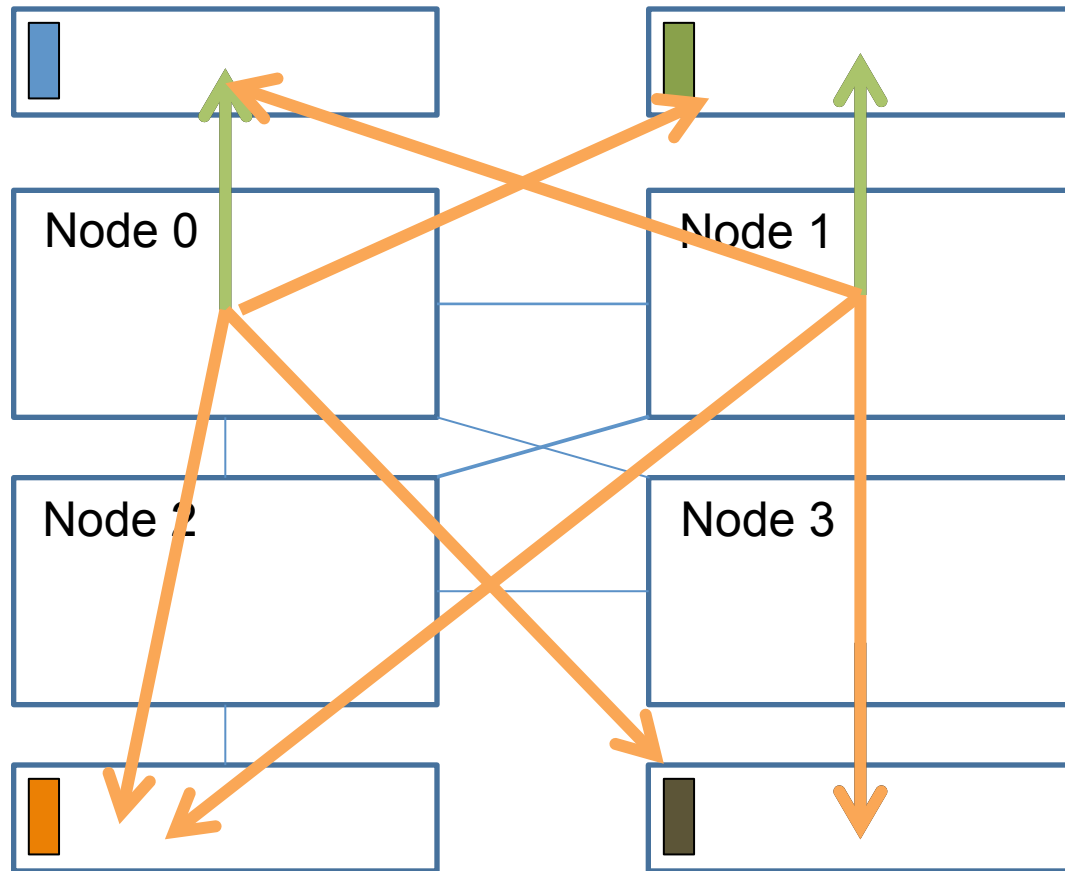
Machines are NUMA

Contention hurts performance even more.



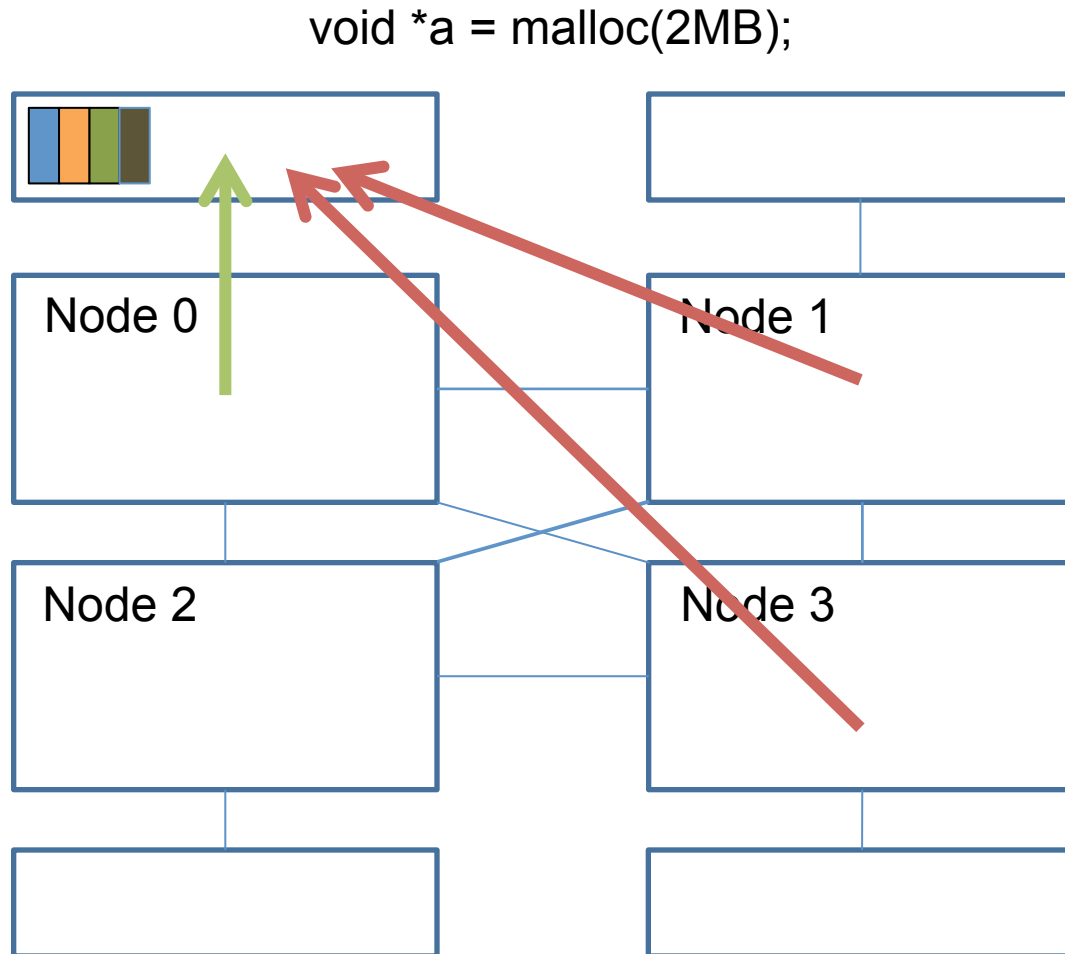
Large pages on NUMA machines (1/2)

```
void *a = malloc(2MB);
```



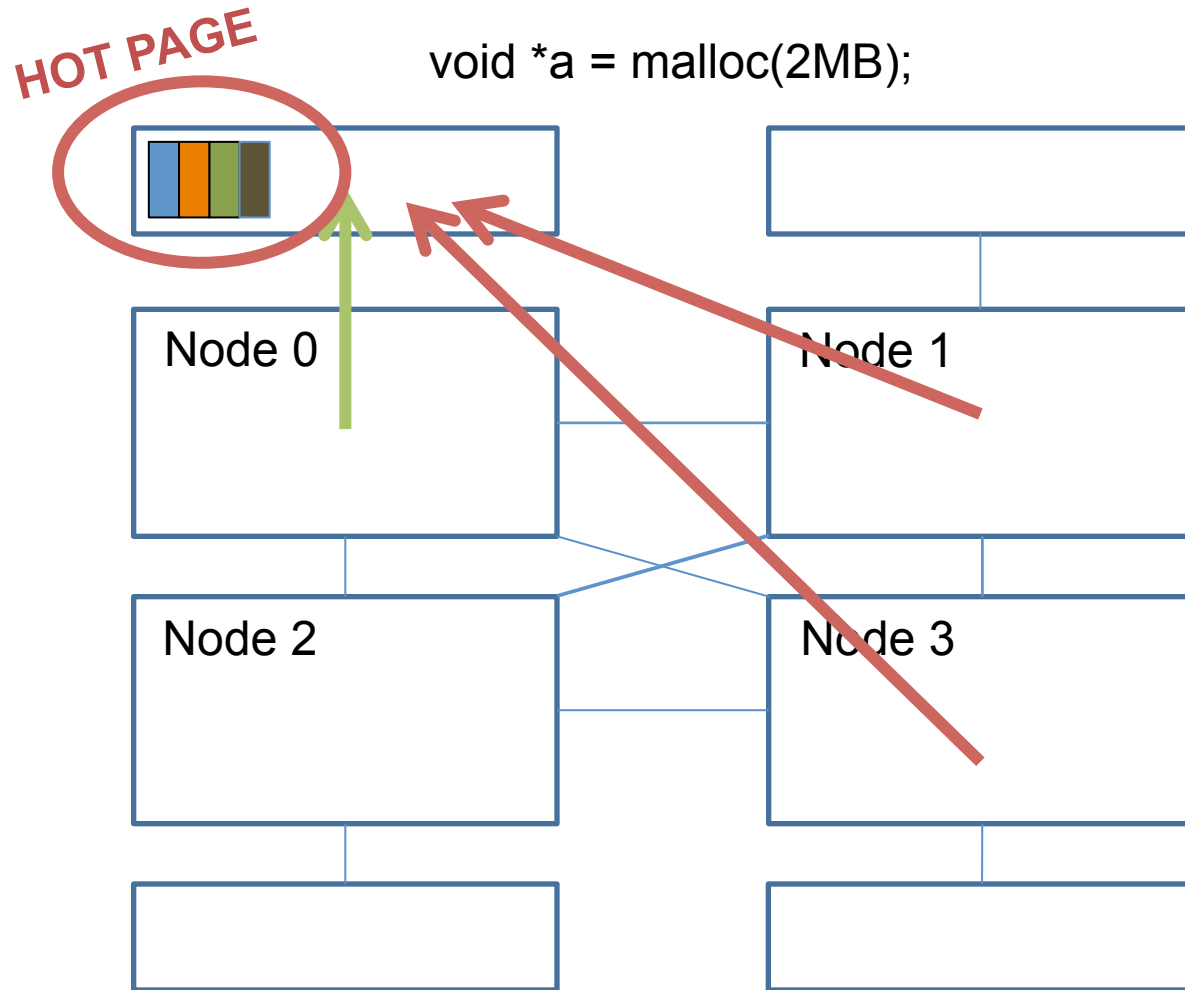
With 4K pages, load is balanced.

Large pages on NUMA machines (1/2)



With 2M pages, data are allocated on 1 node => contention.

Large pages on NUMA machines (1/2)



With 2M pages, data are allocated on 1 node => contention.

Performance example (1/2)

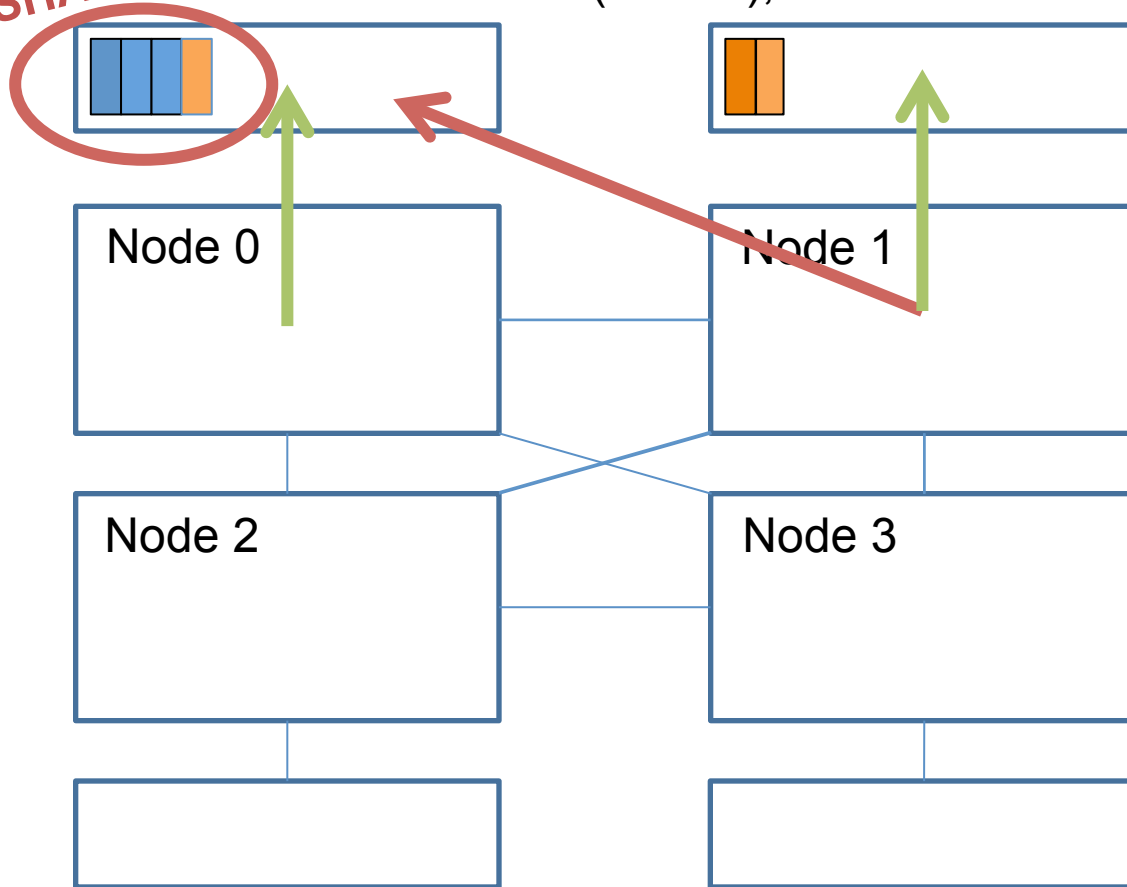
App.	Perf. increase THP/4K (%)	% of time spent in TLB miss 4K	% of time spent in TLB miss 2M	Imbalance 4K (%)	Imbalance 2M (%)
CG.D	-43	0	0	1	59
SSCA.20	17	15	2	8	52
SpecJBB	-6	7	0	16	39

Using large pages, 1 node is overloaded in CG, SSCA and SpecJBB.
Only SSCA benefits from the reduction of TLB misses.

Large pages on NUMA machines (2/2)

**PAGE-LEVEL
FALSE SHARING**

```
void *a = malloc(1.5MB); // node 0  
void *b = malloc(1.5MB); // node 1
```



Page-level false sharing reduces the maximum achievable locality.

Performance example (2/2)

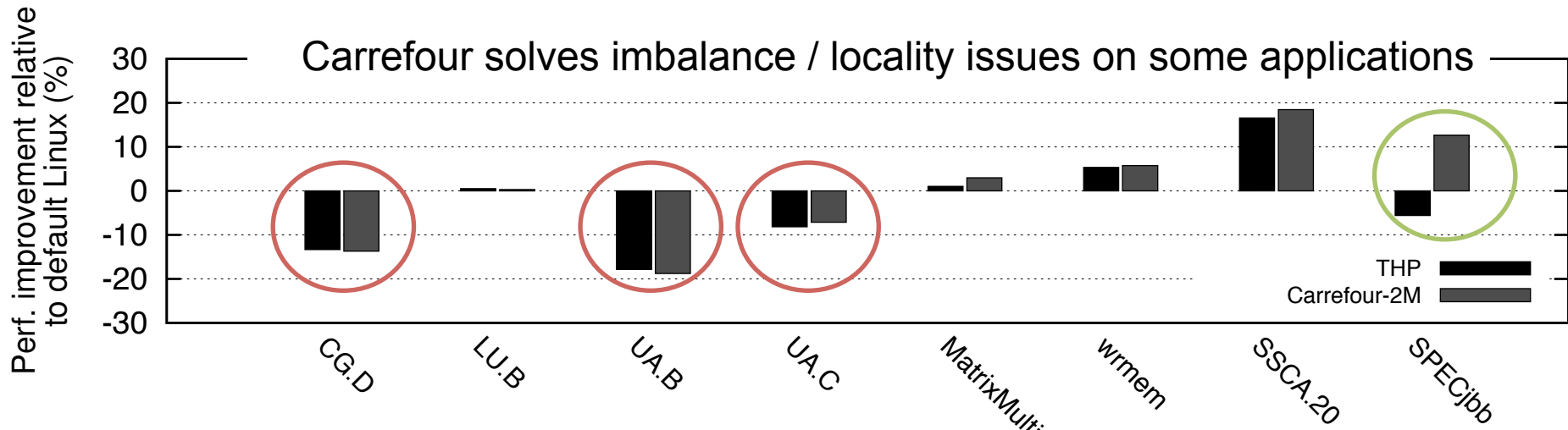
App.	Perf. increase THP/4K (%)	Local Access Ratio 4K (%)	Local Access Ratio 2M (%)
UA.C	-15	88	66

The locality decreases when using large pages.

Can existing memory management algorithms solve the problem?

Existing memory management algorithms do not solve the problem

We run the application with Carrefour[1], the state-of-the-art memory management algorithm. Carrefour monitors memory accesses and places pages to minimize imbalance and maximize locality.



But does not improve performance on some other applications
(hot pages or page-level false sharing)

[1] DASHTI M., FEDOROVA A., FUNSTON J., GAUD F., LACHAIZE R., LEPERS B., QUEMA V., AND ROTH M. Traffic management: A holistic approach to memory placement on NUMA systems. ASPLOS 2013.

We need a new memory management
algorithm

Our solution – Carrefour-LP

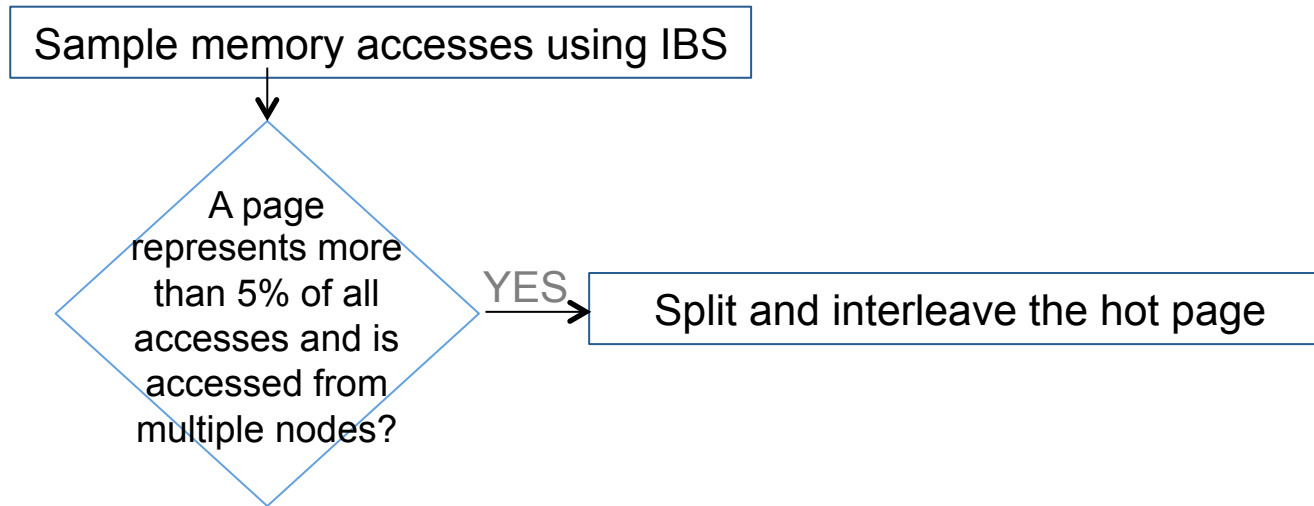
- Built on top of Carrefour.
- By default, 2M pages are activated.
- Two components that run every second:

Reactive component	Conservative component
<p>Splits 2M pages Detects and removes “hot pages” and page-level “false sharing”.</p>	<p>Promotes 4K pages When the time spent handling TLB misses is high.</p>
<p>Deactivate 2M page allocation</p>	<p>Forces 2M page allocation In case of contention in the page fault handler.</p>

- We show in the paper that the two components are required.

Implementation

Reactive component (splits 2M pages)



Implementation

Reactive component (splits 2M pages)

Sample memory accesses using IBS

- Compute observed local access ratio (LAR1)
- Compute the LAR that would have been obtained if each page was placed on the node that accessed it the most.

LAR1 can be significantly improved?

YES

Run carrefour

NO

- Compute the LAR that would have been obtained if each page was **split** and then placed on the node that accessed it the most.

LAR1 can be significantly improved?

YES

Split **all 2M pages** and run carrefour

Implementation challenges

Reactive component (splits 2M pages)

Sample memory accesses using IBS

COSTLY

- Compute observed local access ratio (LAR1)
- Compute the LAR that would have been obtained if each page was placed on the node that accessed it the most (without splitting).

LAR1 can be significantly improved?

YES

Run carrefour

NO

IMPRECISE

- **Compute the LAR** that would have been obtained if each page was **split** and then placed on the node that accessed it the most.

LAR1 can be significantly improved?

YES

Split all 2M pages and run carrefour

COSTLY

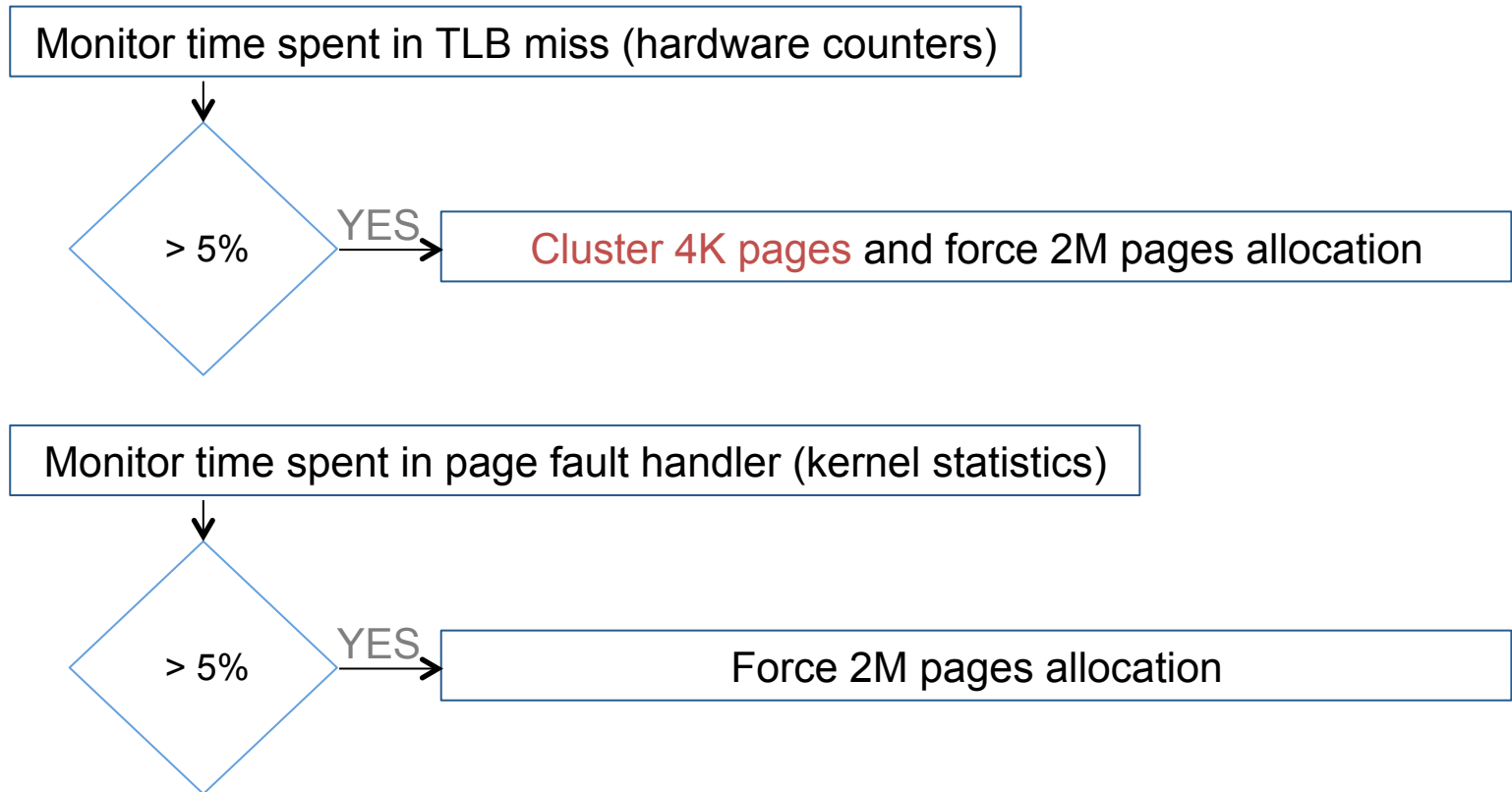
Implementation challenges

Reactive component (splits 2M pages)

- We only have few IBS samples.
- The LAR with “2M pages split into 4K pages” can be wrong.
- We try to be conservative by running Carrefour first and only splitting pages when necessary (splitting pages is expensive).
- Predicting that splitting a 2M page will increase TLB miss rate is hard. This is why the conservative component is required.

Implementation

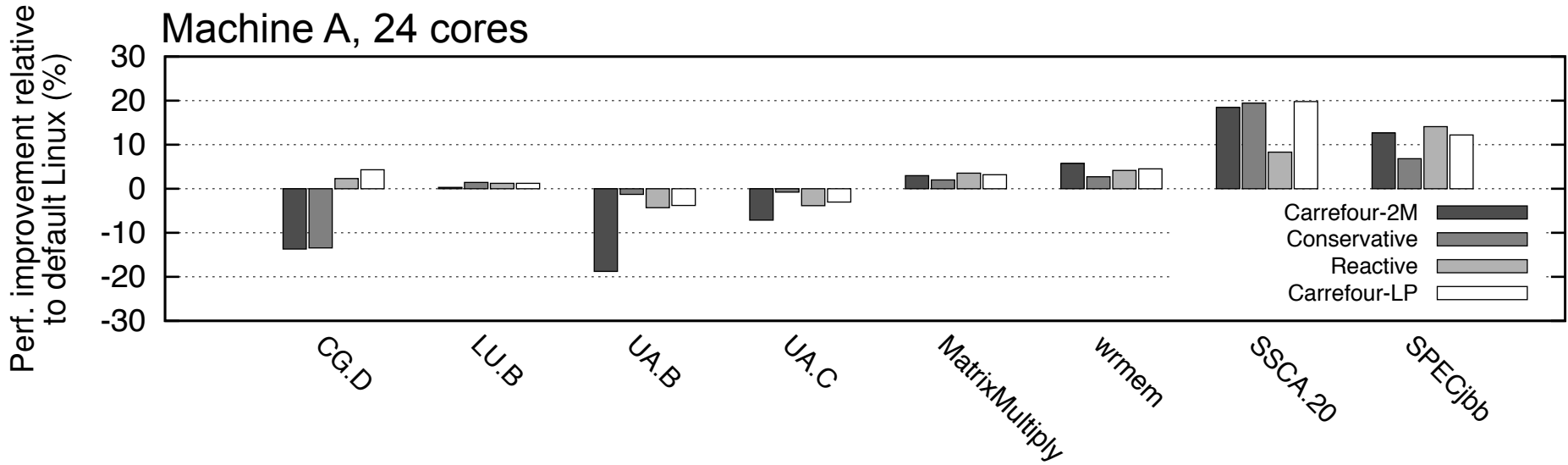
Conservative component



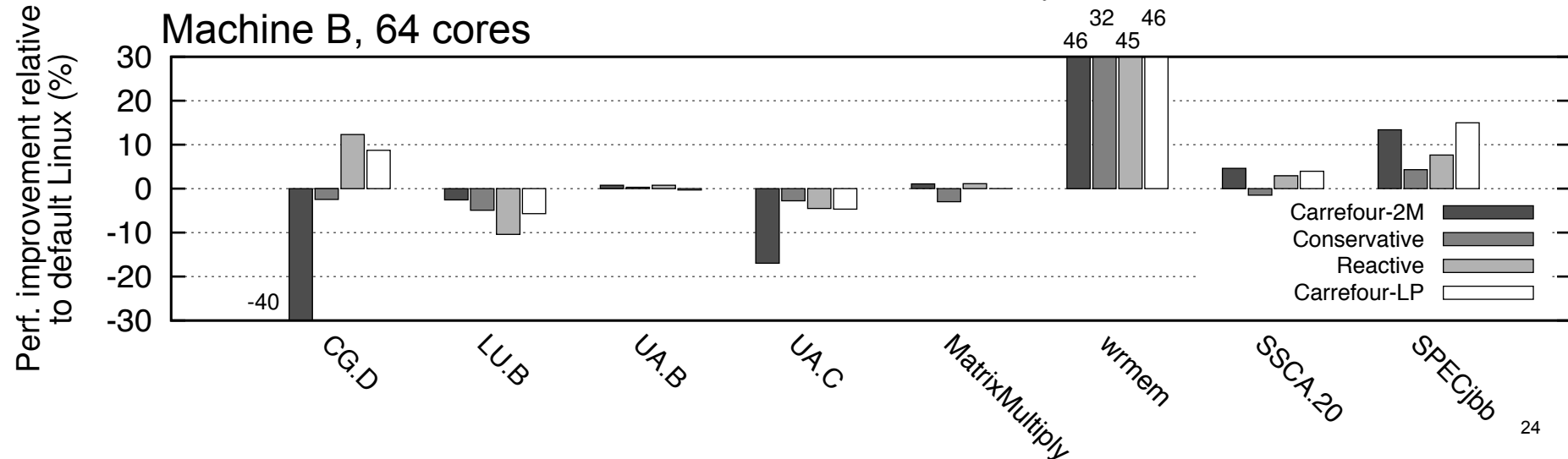
Evaluation

The reactive and conservative components work together.

Machine A, 24 cores



Machine B, 64 cores



Evaluation

- On the selected set of applications, our solution performs up to:
 - 46% better than Linux
 - 50% better than THP.

(The full set of applications is available in the paper.)
- Overhead:
 - Less than 3% CPU overhead.

Conclusion

- Large pages can hurt performance on NUMA systems.
- We identified two new issues when using large pages on NUMA systems: “hot pages” and “page-level false sharing”.
- We designed a new algorithm, Carrefour-LP, that:
 - Splits large pages when they hurt performance.
 - Promotes 4K pages and uses 2M page allocation when beneficial.
- Carrefour-LP restores the performance when it was lost due to large pages and makes their benefits accessible to applications.

Questions?

Performance example

App.	Perf. increase THP/4K	Time spent in page fault handler 4K	Time spent in page fault handler 2M	Local access ratio 4K (%)	Local Access ratio 2M (%)	Imbalance 4K (%)	Imbalance 2M (%)
CG.D	-43	2200ms (0.1%)	450ms (0.1%)	40	36	1	59
UA.C	-15	100ms (0.2%)	50ms (0.1%)	88	66	14	12
WR	109	8700ms (38%)	3700ms (32%)	50	55	147	136
SSCA.20	17	90ms (0%)	150ms (0%)	25	26	8	52
SpecJB B	-6	8400ms (2%)	5900ms (1.5%)	12	15	16	39