128-bit RISC-V proposal: implications on HPC applications, data-center working sets, and object-oriented computing

Osman S. Ünsal

The First 128 bit RISC-V European Workshop,
Hipeac 2025, Barcelona

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

RISC-V 128-bit addressing/data: how about applications?

- HPC applications: Beyond double precision
- Object-oriented computing: Tracking billions of objects/methods
- Data-center use cases: Huge working sets

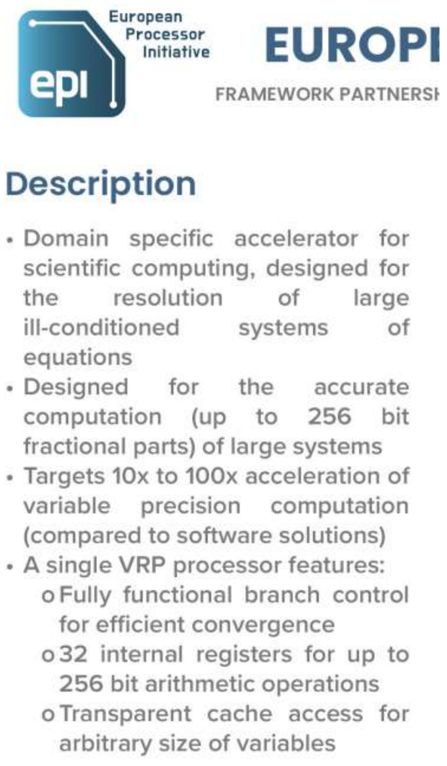
HPC: Beyond Double Precision

- All INT compute is exact, all FP computation is approximate
- Usually multiply does not commute for FP
- You are applying approximate computing every time you declare a real*4 vs real*8
- Consider A/B versus A*(1.0/B)

Is Double Precision too approximate?

- Depending on the application, yes
- Variable Precision Core in EPI

  Guthmuller et al. Xvpfloat IEEE TC 2024
- 128 address/data space would help


European Processor Initiative

EUROPI

FRAMEWORK PARTNERSH

Description

- Domain specific accelerator for scientific computing, designed for the resolution of large ill-conditioned systems of equations
- Designed for the accurate computation (up to 256 bit fractional parts) of large systems
- Targets 10x to 100x acceleration of variable precision computation (compared to software solutions)
- A single VRP processor features:
  - Fully functional branch control for efficient convergence
  - 32 internal registers for up to 256 bit arithmetic operations
  - Transparent cache access for arbitrary size of variables

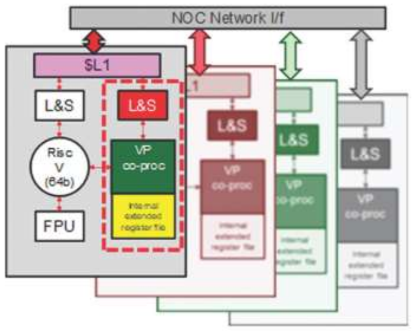


Figure 1 Variable precision tile structure

European Processor Initiative
epi
EUPEX European Pilot for Exascale
THE EUPILOT Pilot using Independent Local & Open Technologies
EUROPEAN PROCESSOR INITIATIVE AND THE ADVANCED EUROPEAN PILOT SUPERCOMPUTERS

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

RISC-V 128-bit addressing/data: how about applications?

- HPC applications: Beyond double precision
- Object-oriented computing: Tracking billions of objects/methods
- Data-center use cases: Huge working sets

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

Object-oriented computing: Tracking billions of objects/methods

- 128-bit addressing could provide the capability of a naming space for each unique object/method
- Melding memory and object addressing space facilitates security – easy to detect leaks
- To be addressed later by Steve Wallach

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

Motivation: Object Oriented Computer Architecture (OOCA)

Markovic et al NDCA-2: New Directions in Computer Architecture 2011

- What is OOCA ?
  - Direct path from Object Oriented Programming Model to hardware, hardware design is object aware
  - Special hardware for special object maintaining object oriented interface
- Why OOCA ?
  - Object Oriented Languages
    - Show high locality:
      - Current hardware is not exploiting what programmer has expressed or its purpose!
    - Protection
      - Memory notion is different (objects) -Between objects there is only communication through methods
      - Object addressing namespace is embedded in the 128 bit address space
      - Entry points to an object are well-defined with public methods and properties (*messages*)

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

ISA Extensions

- ISA extensions consists of three basic instructions:
  - *Call* – creates new Context or triggers execution of hardware implemented instructions like add, sub etc.
  - *Send* – sends the reference of an *Object* to another Object or *Context*
  - *Select* – fetches the reference to an Object encapsulated inside the other Object (class attribute e.g. getVal)

OOCA Objects

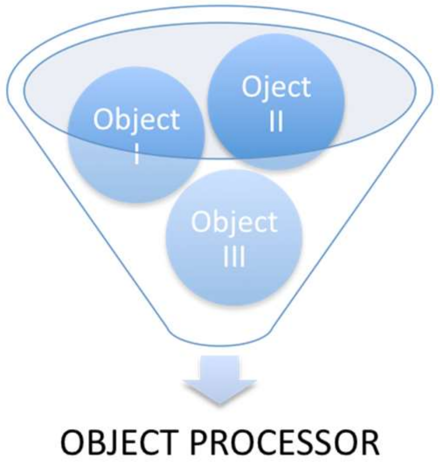
- OOCA has several types of objects:
  - Execution objects:
    - *Context* – object for user defined functions, has its own memory space
  - Data objects:
    - *Basic objects* – objects that manage and store multiple versions of basic data types eg. Int, float, char etc.
    - *Complex objects* – objects for user defined types like classes and structures

OOCA Execution Model

- Asynchronous execution model
  - Method are executed asynchronously
    - Input/output parameters are sent asynchronously.
    - Methods are executing in its own context
- Distributed execution model
  - OO Model is supported by hardware abstract layer
    - Objects are easy distributed around a complex network

OOCA Architectre

- Several Object can be mapped to the same OP
- OP is minimum execution unit that implements virtual hardware layer
- Each OP is internally flexible. It can support different hardware implementations.
- Many possible cores can manage OP internal requirements (flexible and compatible):



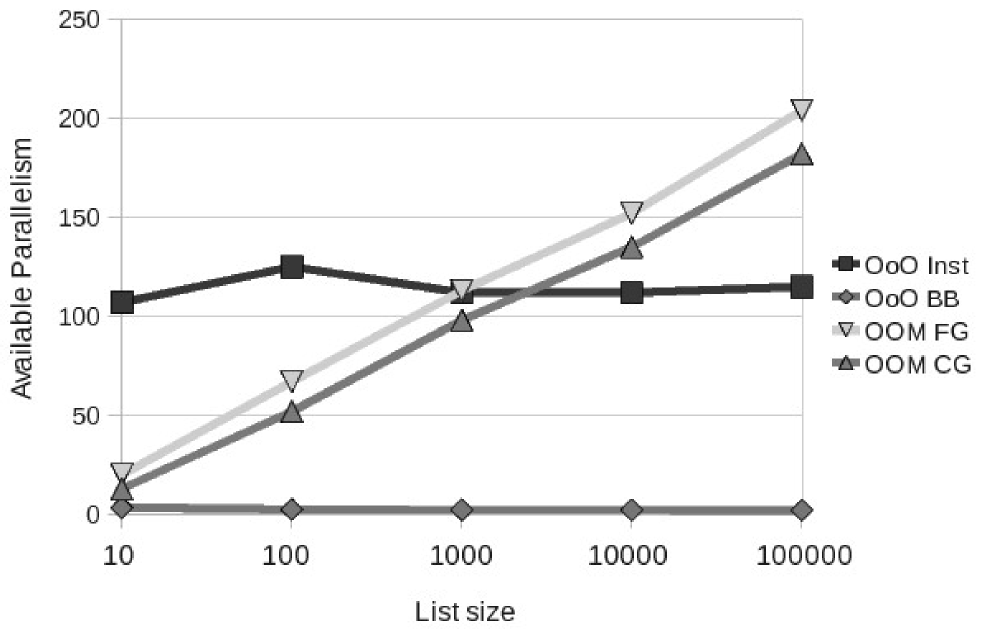
(out-of-order processor, In-order processor (embedded), Multiprocessor, Vector processor, Processor + FPGA (Reconfigurable Architectures), Data flow processor)

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

OOCA Results

- DDG (Dynamic Dependency Graph) methodology Austin et al. 1992
- DDG containing only true data dependencies to give upper bound on the available parallelism
- We compare the level of available parallelism extracted form sequential quicksort algorithm that our ideal model can achieve over ideal OoO
- For ideal OoO we use Pin Tool to generate traces
- For ideal OOM we use our functional level simulator

OOCA Results



Barcelona Supercomputing Center
Centro Nacional de Supercomputación

OOCA Advantages

- Data locality
  - Objects are smaller areas of locality (smaller working sets, better for internal data management)
  - Data reuse (previous method calls may have created temporal structures that can be reused on future execution - depends on garbage collection)
- Execution locality
  - Better Control Predictors (local & global predictors)
  - Method prefetching
    - Statically : Compiler knows what other methods could be called inside one method
    - Dynamically: predictor can help to improve static strategies

OOCA Advantages

- Execution recycling & reusing
  - It can use objects and methods created for previous execution:
  - Object allocation (memory and other internal objects)
  - Code allocation (faster code prefetching)
  - Result and local reuse (Haskell like)

RISC-V 128-bit addressing/data: how about applications?

- HPC applications: Beyond double precision
- Object-oriented computing: Tracking billions of objects/methods
- Data-center use cases: Huge working sets

Redundant Memory Mappings (ISCA2015)

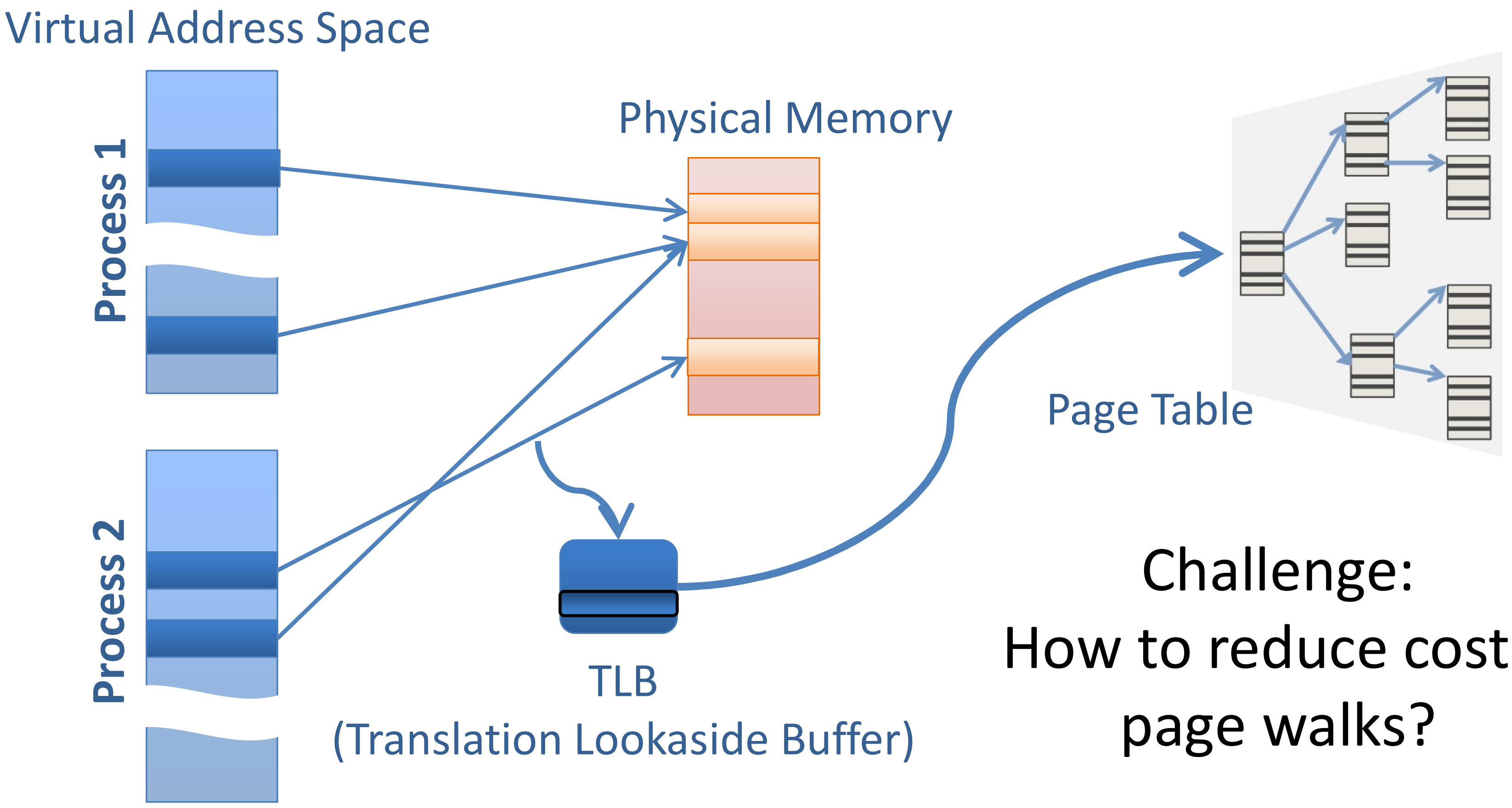
Summary

- **Problem:** Virtual memory overheads are high (up to 41%)
- Proposal: **Redundant Memory Mappings**
  - **Propose** compact representation called *range translation*
  - **Range Translation** – arbitrarily large contiguous mapping
  - **Effectively cache, manage and facilitate** range translations
  - **Retain flexibility** of 4KB paging
- Result:
  - **Reduces overheads** of virtual memory to **less than 1%**

Outline

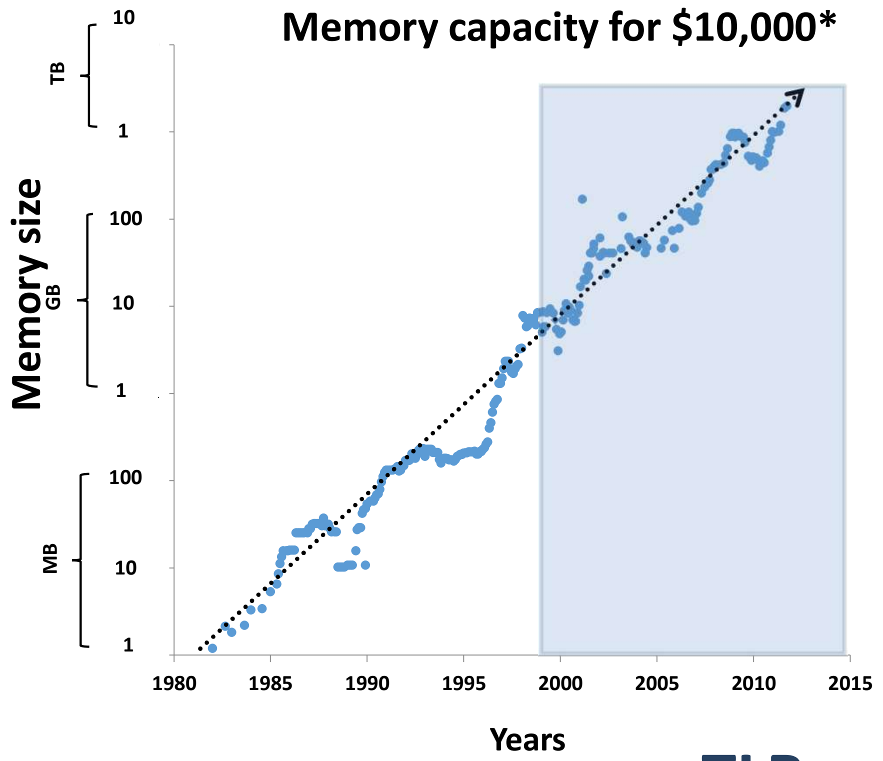
- Motivation ⇐
    - Virtual Memory Refresher + Key Technology Trends
    - Goals + Key Observation
- Design: Redundant Memory Mappings
- Results
- Conclusion

Virtual Memory Refresher



Challenge:
How to reduce costly page walks?

Two Technology Trends

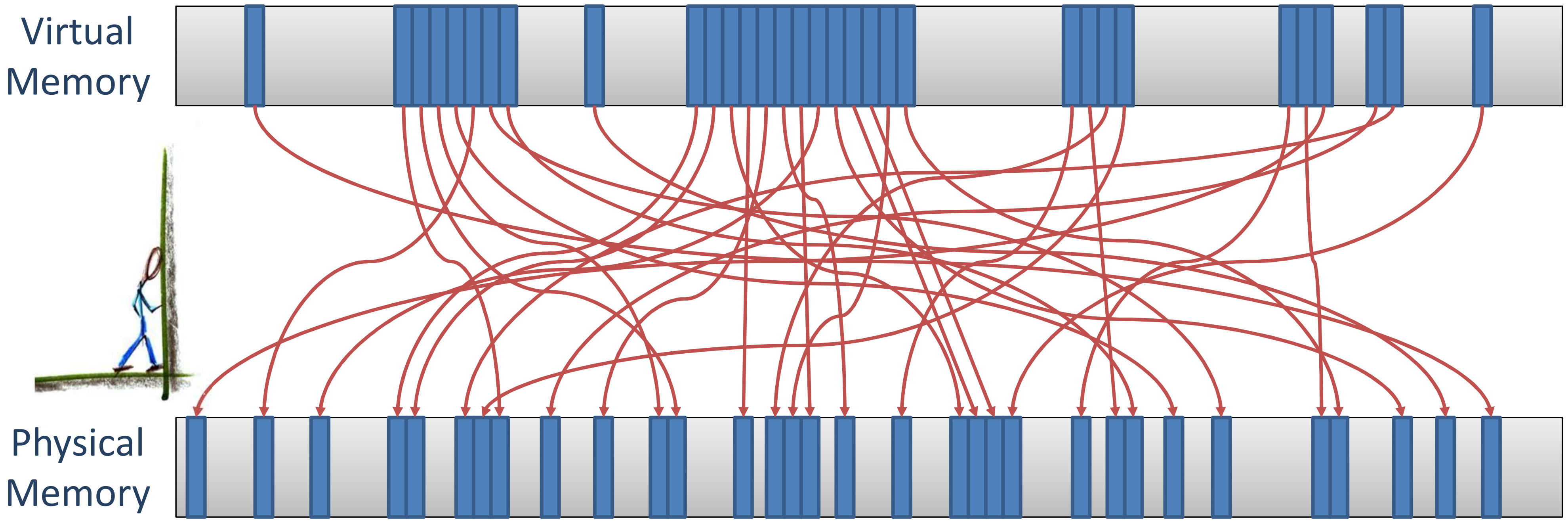


Year	Processor	L1 DTLB entries
1999	Pent. III	72
2001	Pent. 4	64
2008	Nehalem	96
2012	IvyBridge	100
2015	Broadwell	100

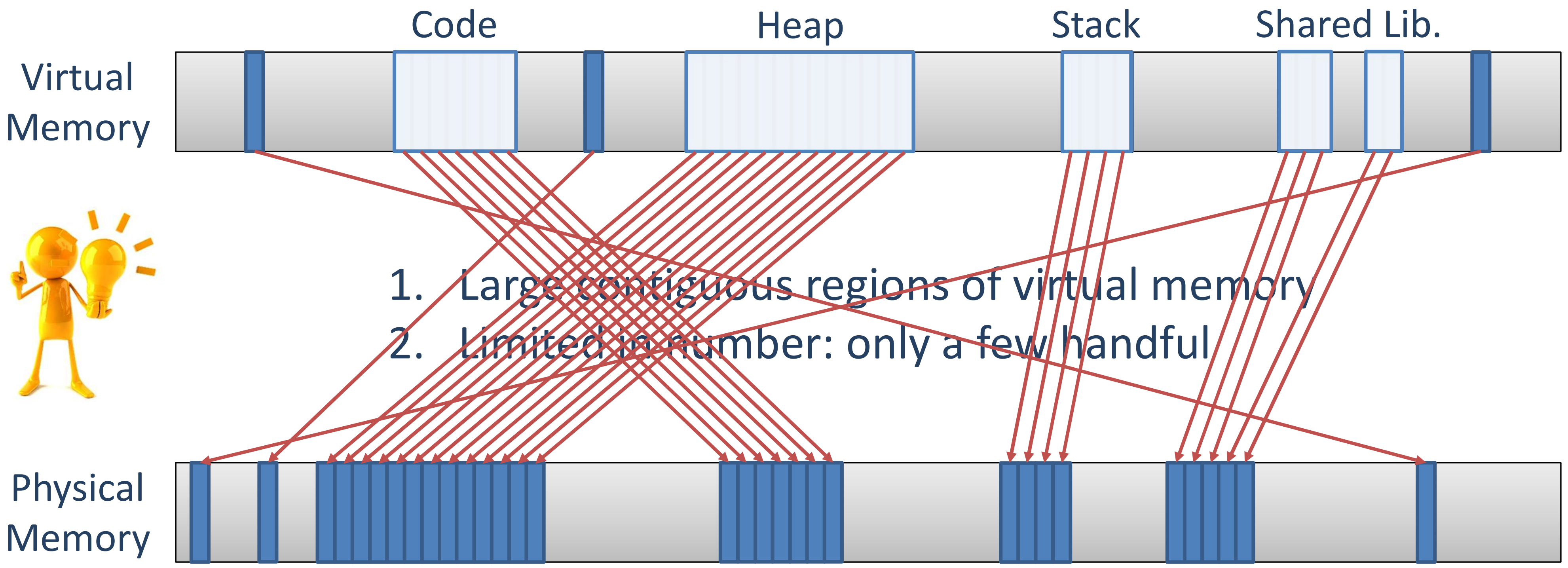
TLB reach is limited

**Inflation-adjusted 2011 USD, from: jcmit.com*

Key Observation



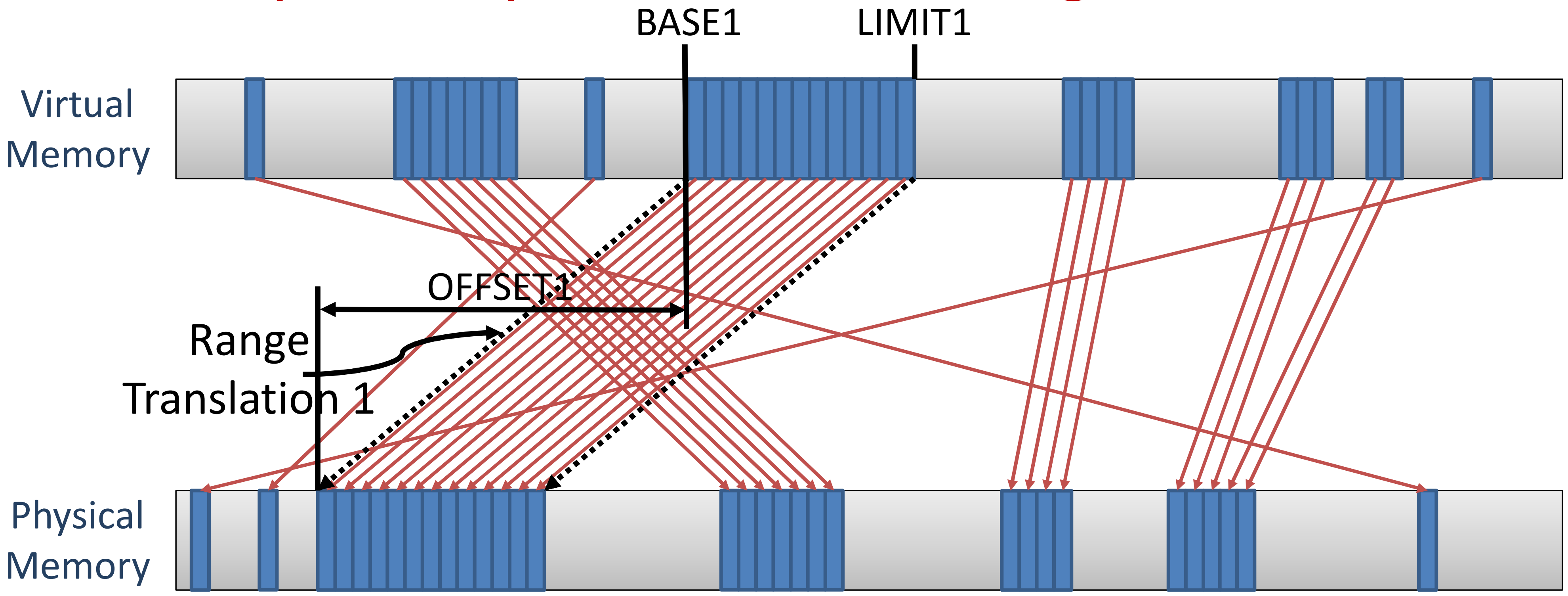
Key Observation



1. Large contiguous regions of virtual memory
2. Limited in number: only a few handful

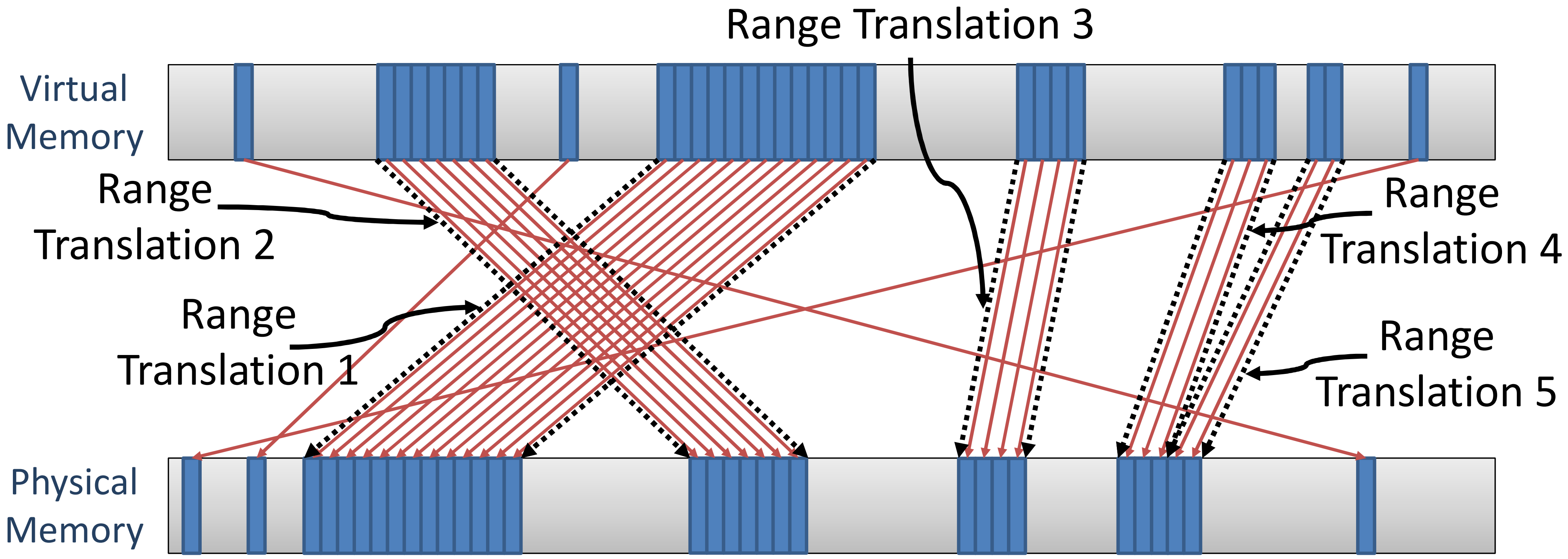
Contiguity in physical memory: good for 128 bit address space

Compact Representation: Range Translation



Range Translation: *is a mapping between contiguous virtual pages mapped to contiguous physical pages with uniform protection*

Redundant Memory Mappings

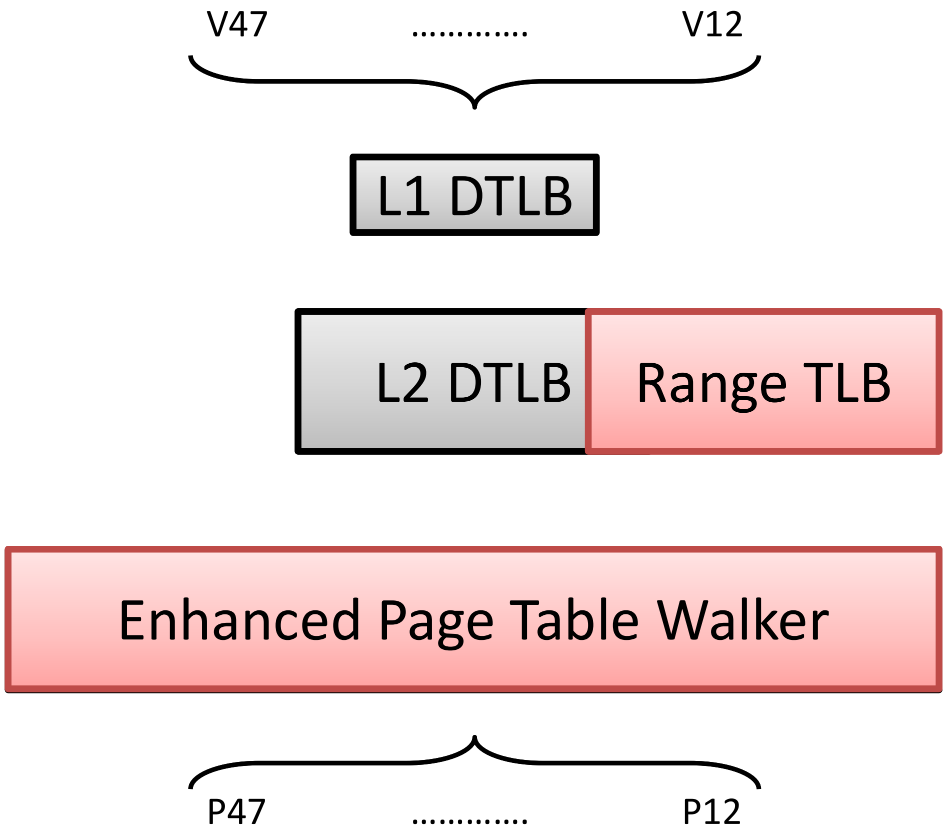


Map most of process's virtual address space redundantly with modest number of range translations in addition to page mappings

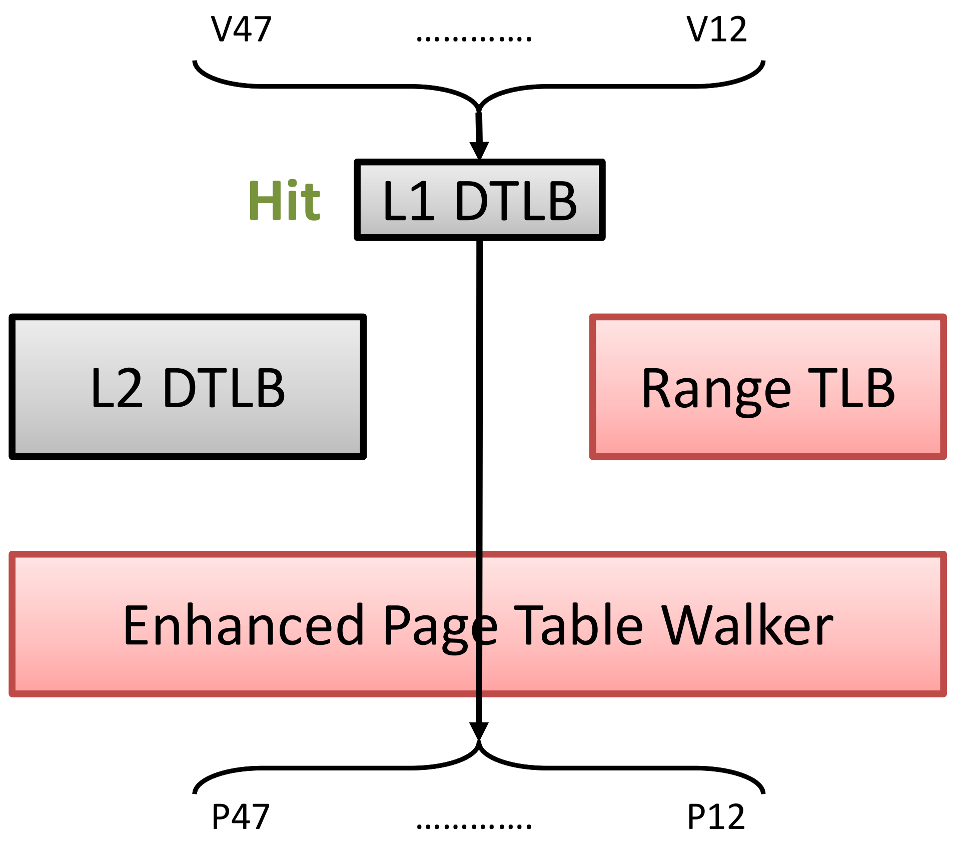
Outline

- Motivation
- Design: Redundant Memory Mappings 🡐
    - A. Caching Range Translations
    - B. Managing Range Translations
    - C. Facilitating Range Translations
- Results
- Conclusion

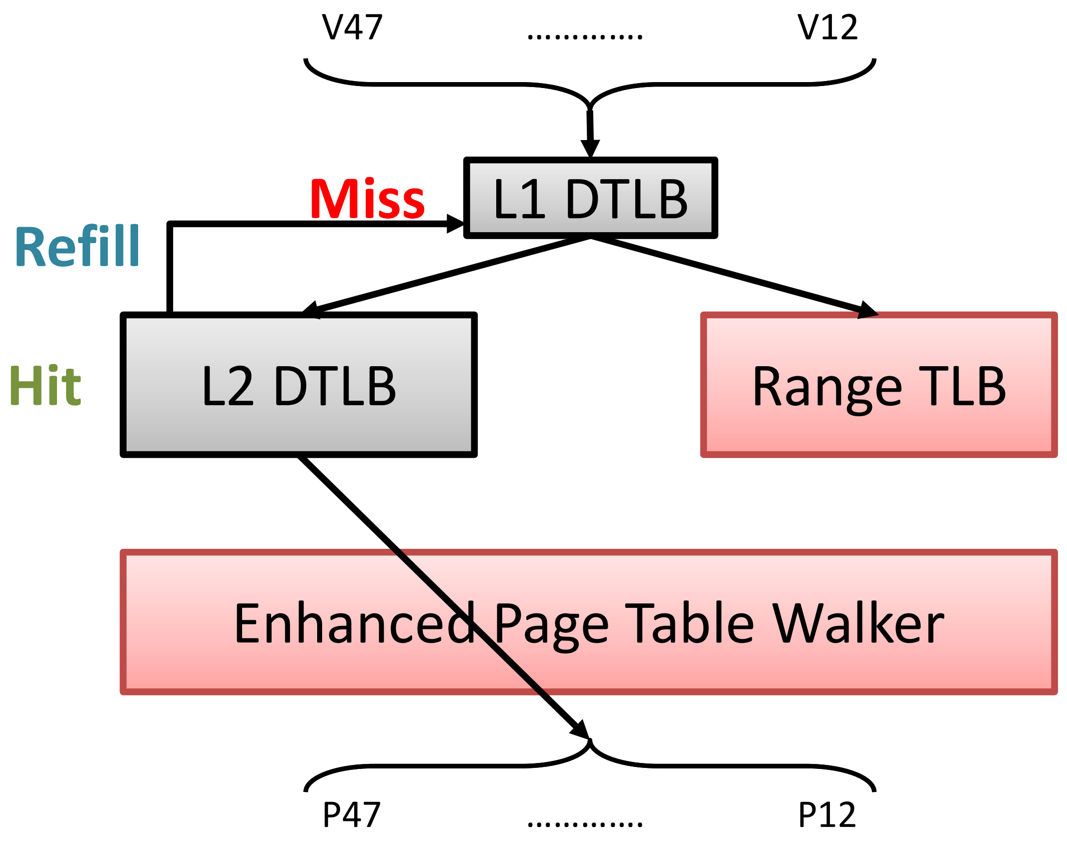
A. Caching Range Translations



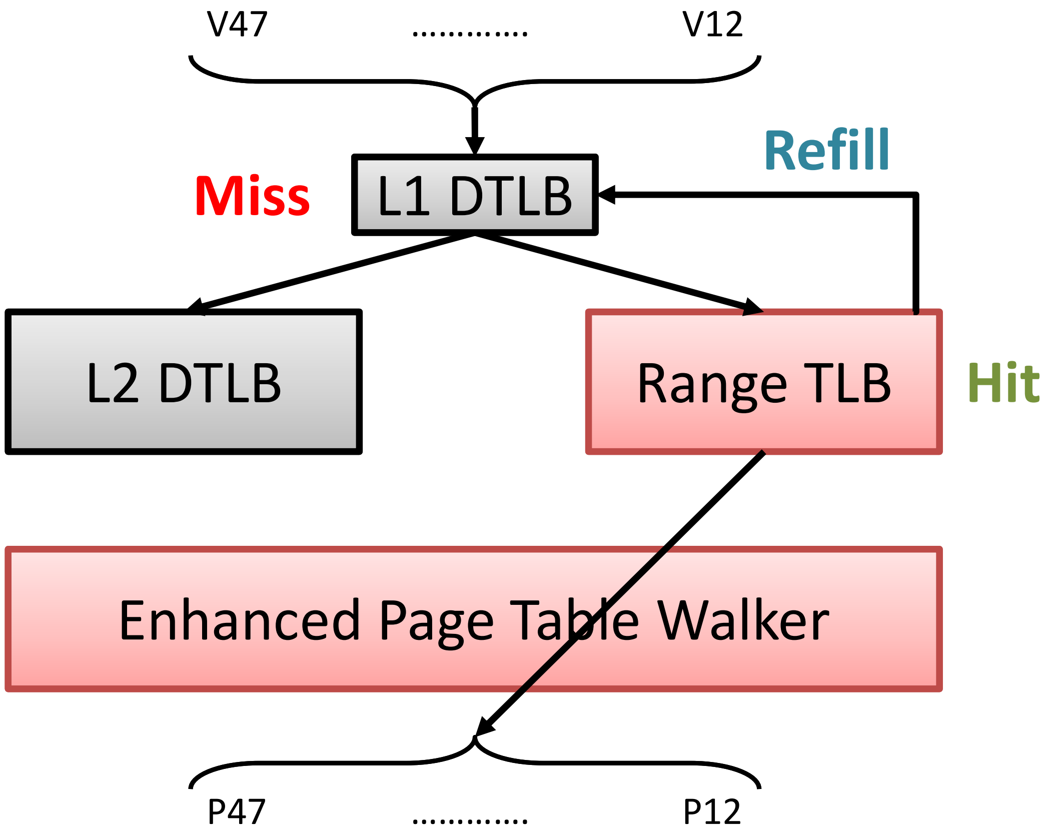
A. Caching Range Translations



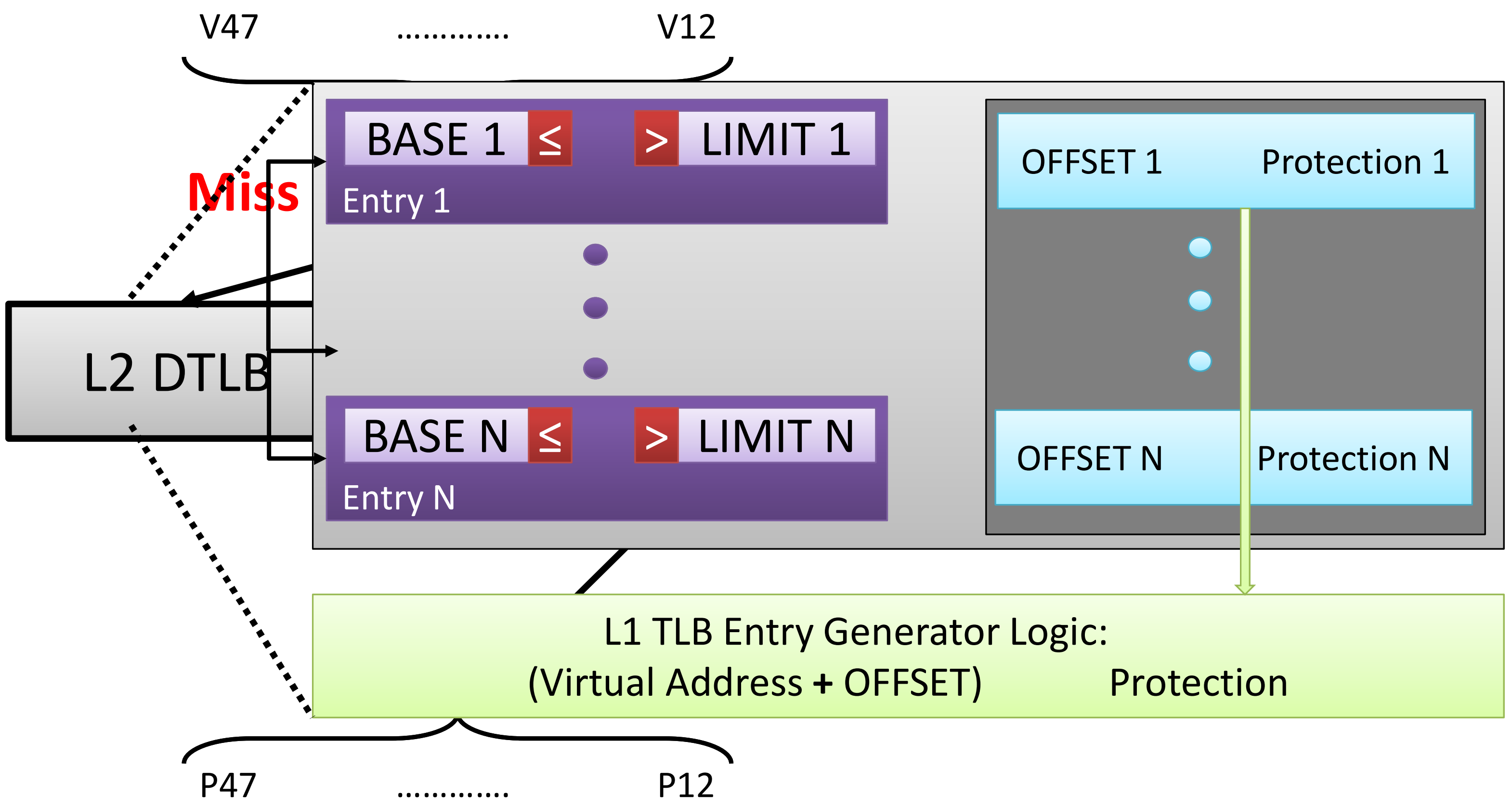
A. Caching Range Translations



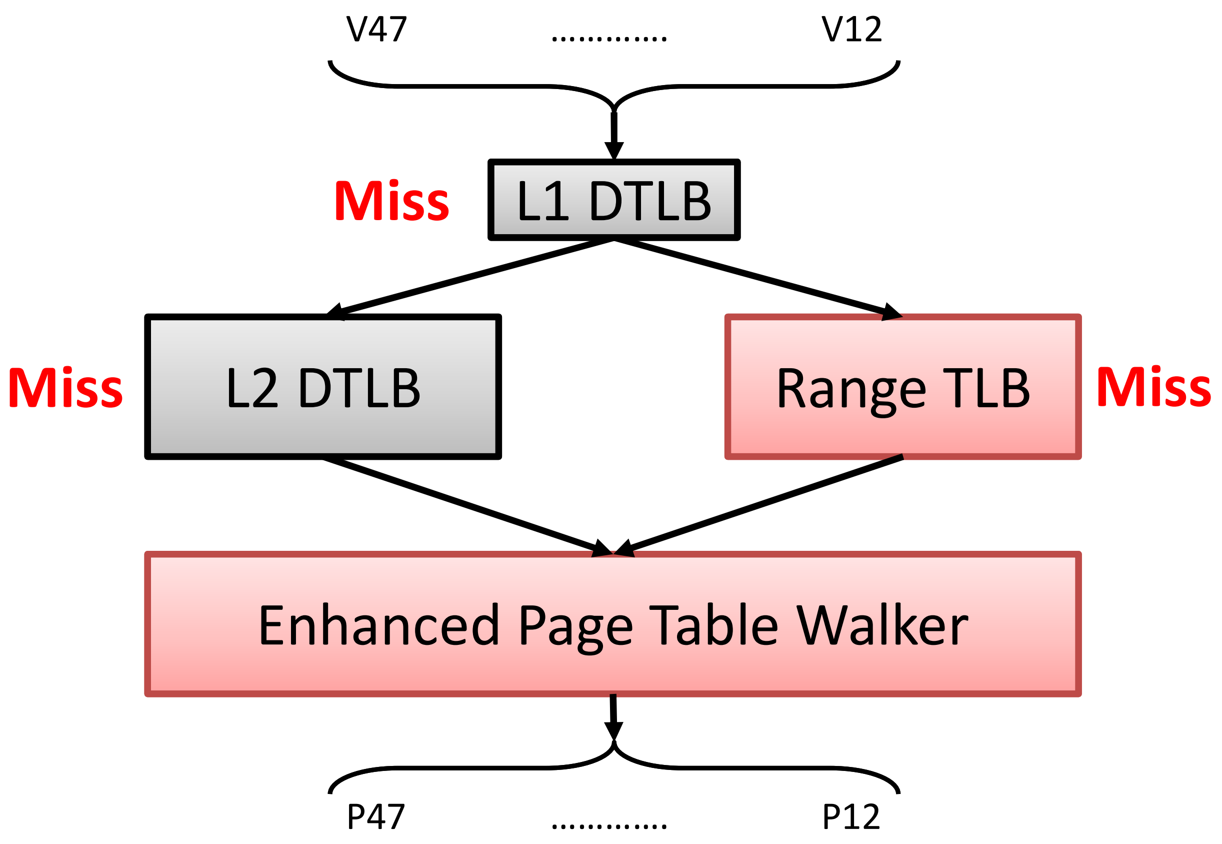
A. Caching Range Translations



A. Caching Range Translations

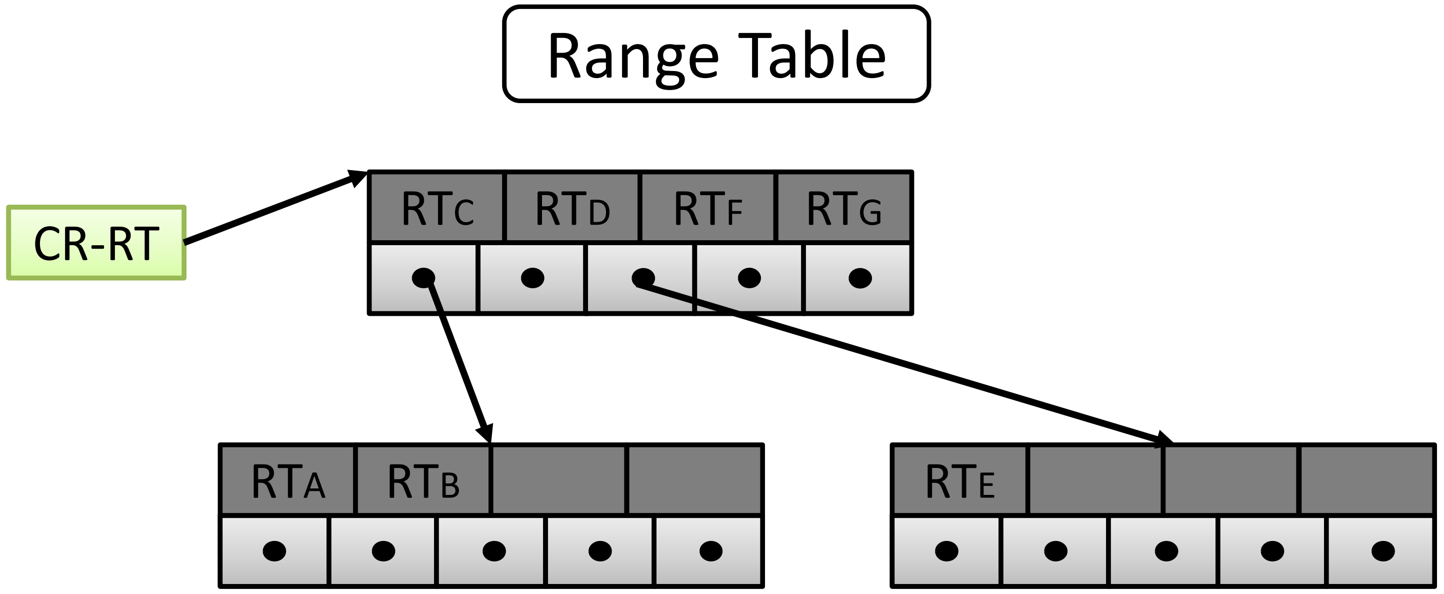


A. Caching Range Translations



B. Managing Range Translations

- Stores all the range translations in a OS managed structure
- Per-process like page-table



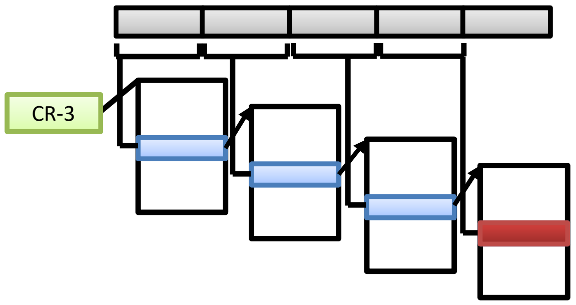
B. Managing Range Translations

Redundancy to the rescue

***One** bit in page table entry denotes that page is part of a range*

1

Page Table Walk



Part of a range

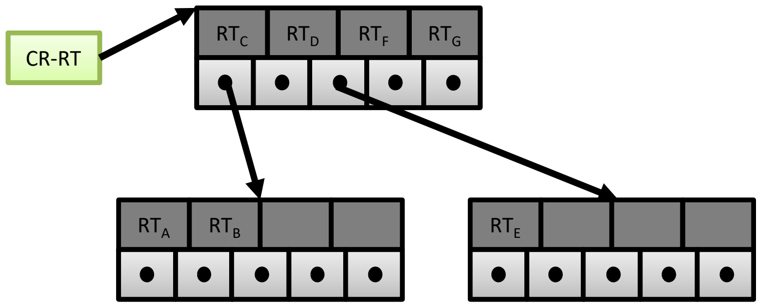
Insert into L1 TLB

2

Application resumes memory access

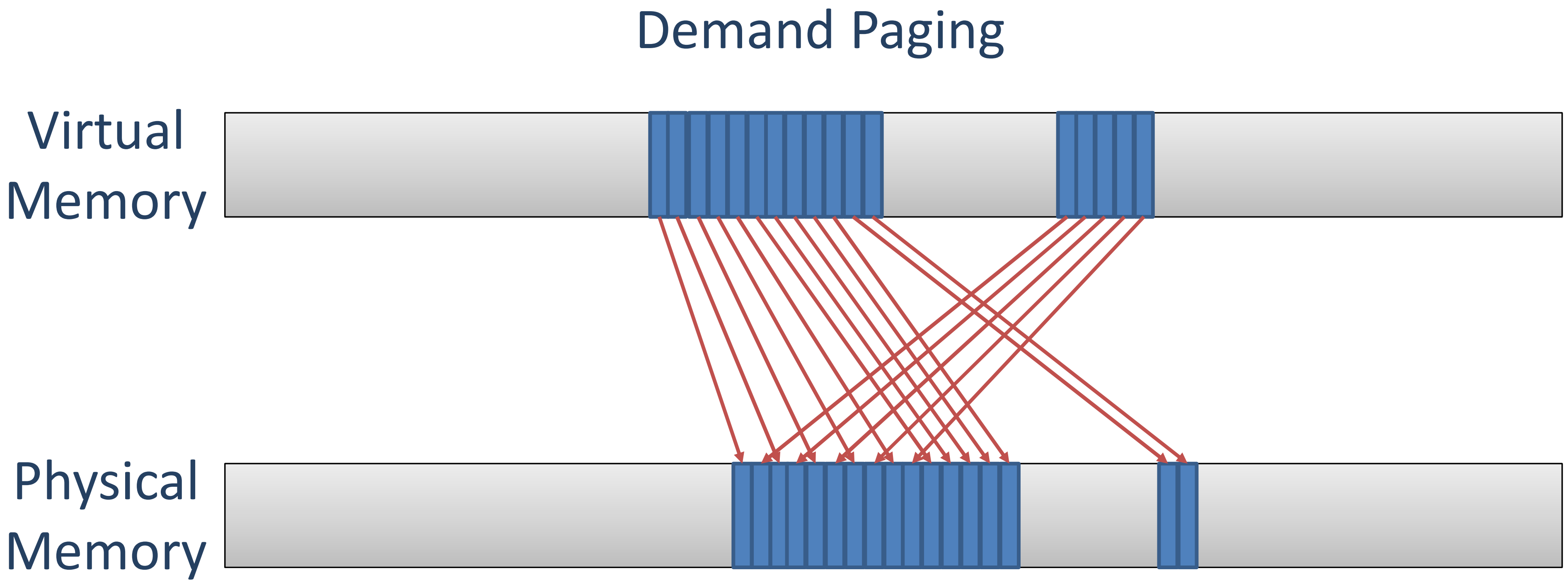
3

Range Table Walk (Background)



Insert into Range TLB

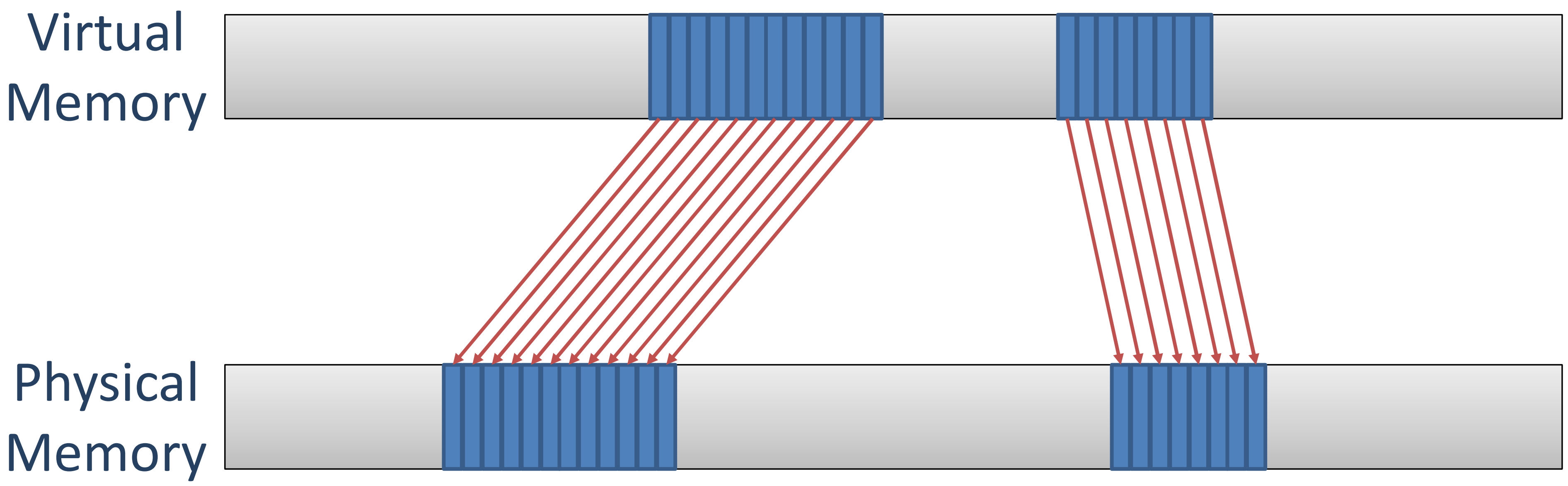
C. Facilitating Range Translations



Does not facilitate physical page contiguity for range creation

C. Facilitating Range Translations

Eager Paging



Allocate physical pages when virtual memory is allocated

Increases range sizes → Reduces number of ranges

Outline

- Motivation
- Design: Redundant Memory Mappings
- Results 🡐
  - Methodology
  - Performance Results
  - Virtual Contiguity
- Conclusion

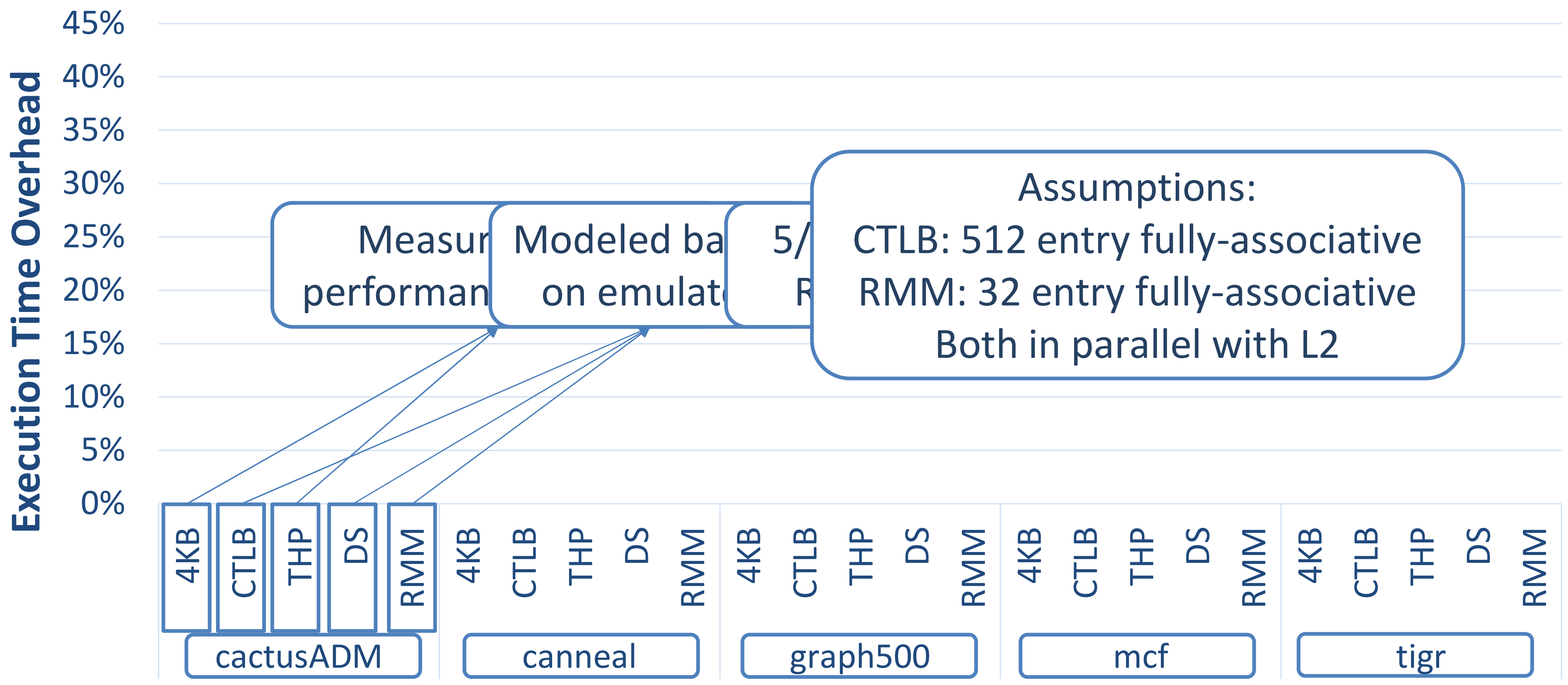
Methodology

- Measure cost on page walks on real hardware
  - Intel 12-core Sandy-bridge with 96GB memory
  - 64-entry L1 TLB + 512-entry L2 TLB 4-way associative for 4KB pages
  - 32-entry L1 TLB 4-way associative for 2MB pages
- Prototype Eager Paging and Emulator in Linux v3.15.5
  - BadgerTrap for online analysis of TLB misses and emulate Range TLB
- Linear model to predict performance
- Workloads
  - Big-memory workloads, SPEC 2006, BioBench, PARSEC

Comparisons

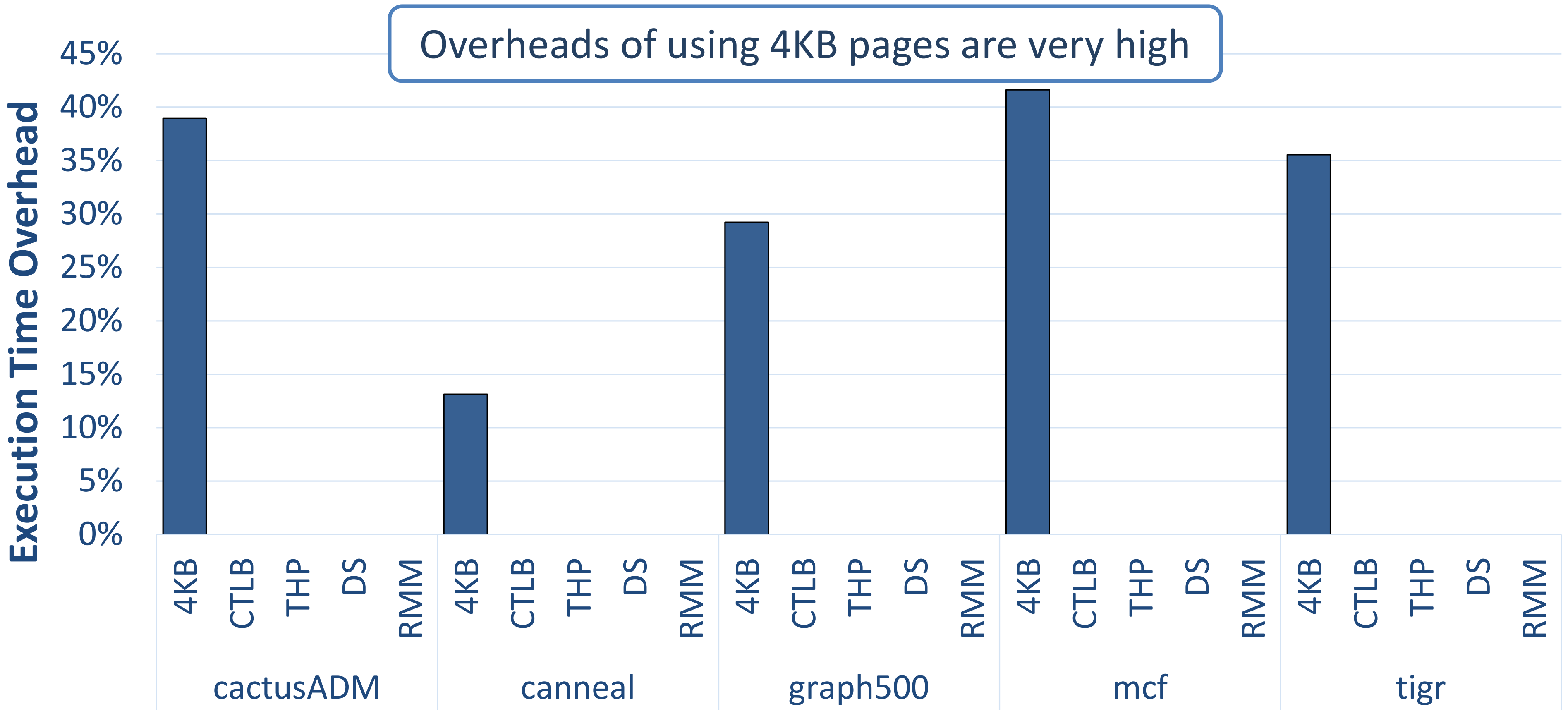
- **4KB:** Baseline using 4KB paging
- **THP:** Transparent Huge Pages using 2MB paging *[Transparent Huge Pages]*
- **CTLB:** Clustered TLB with cluster of 8 4KB entries *[HPCA'14]*
- **DS:** Direct Segments *[ISCA'13 and MICRO'14]*
- **RMM:** Redundant Memory Mappings *[ISCA'15]*

Performance Results

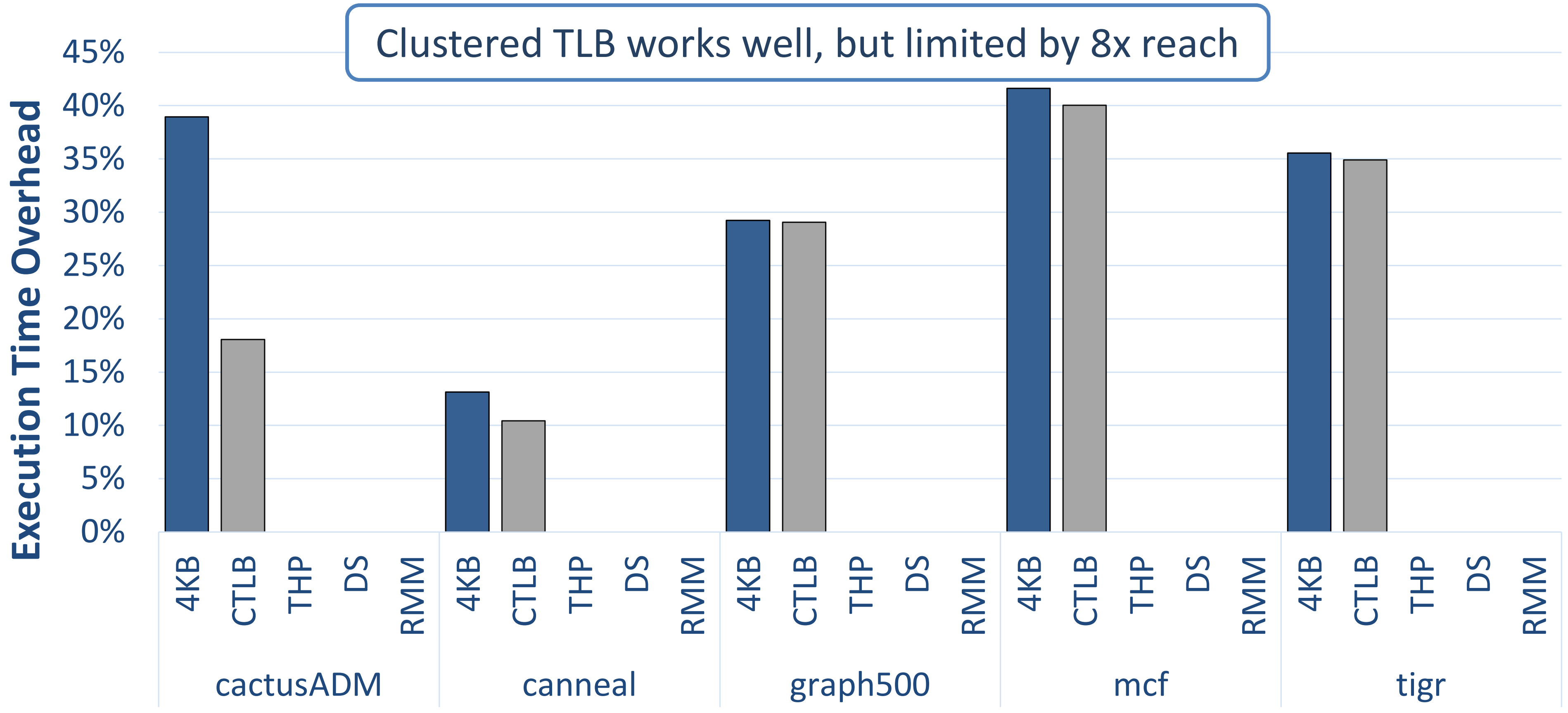


Assumptions:
CTLB: 512 entry fully-associative
RMM: 32 entry fully-associative
Both in parallel with L2

Performance Results

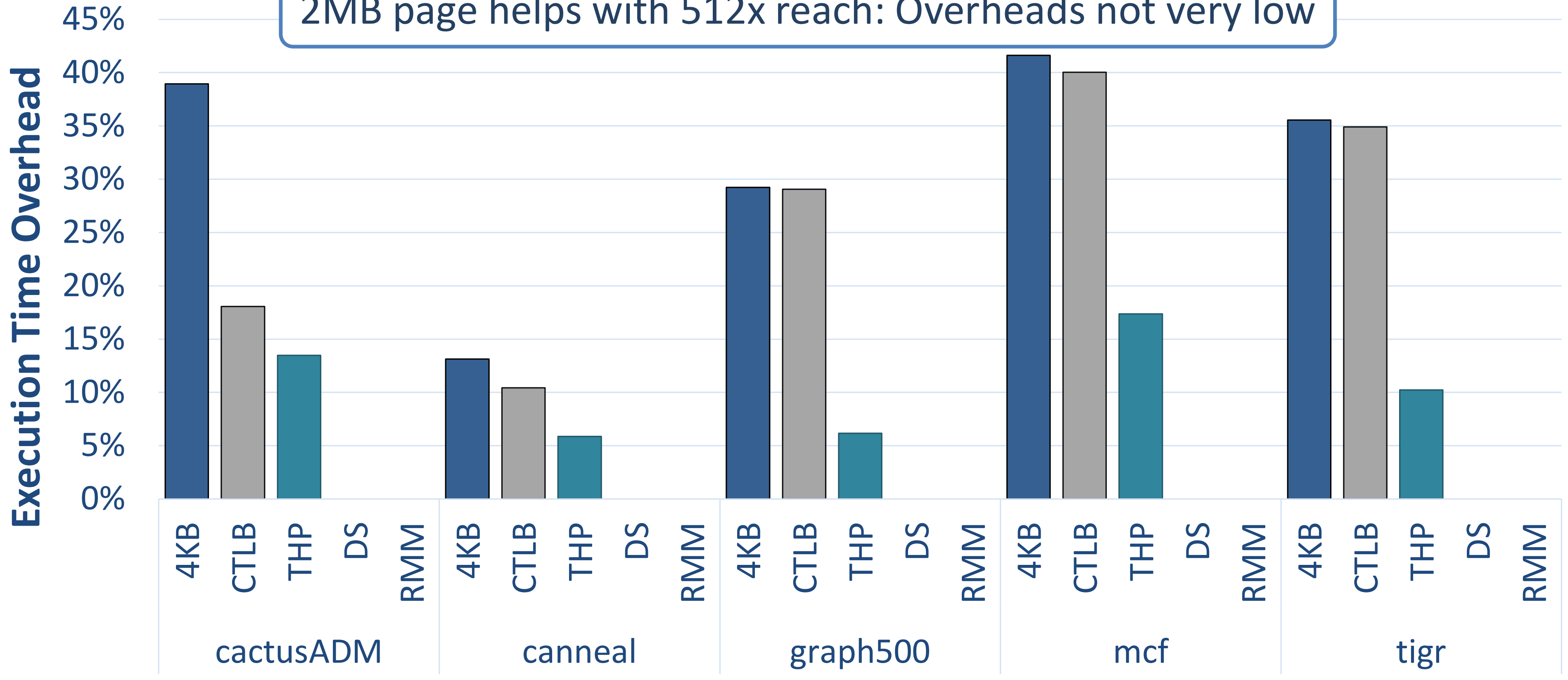


Performance Results



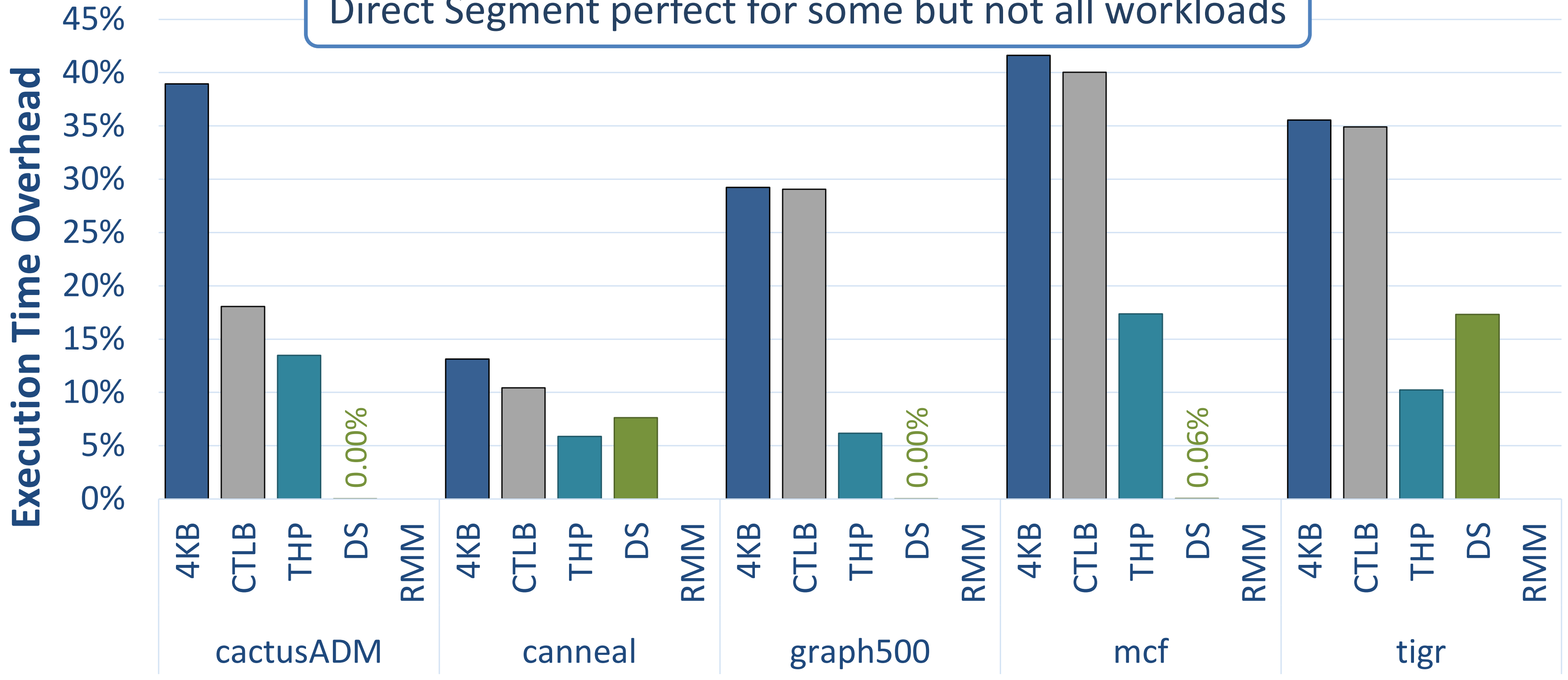
Performance Results

2MB page helps with 512x reach: Overheads not very low



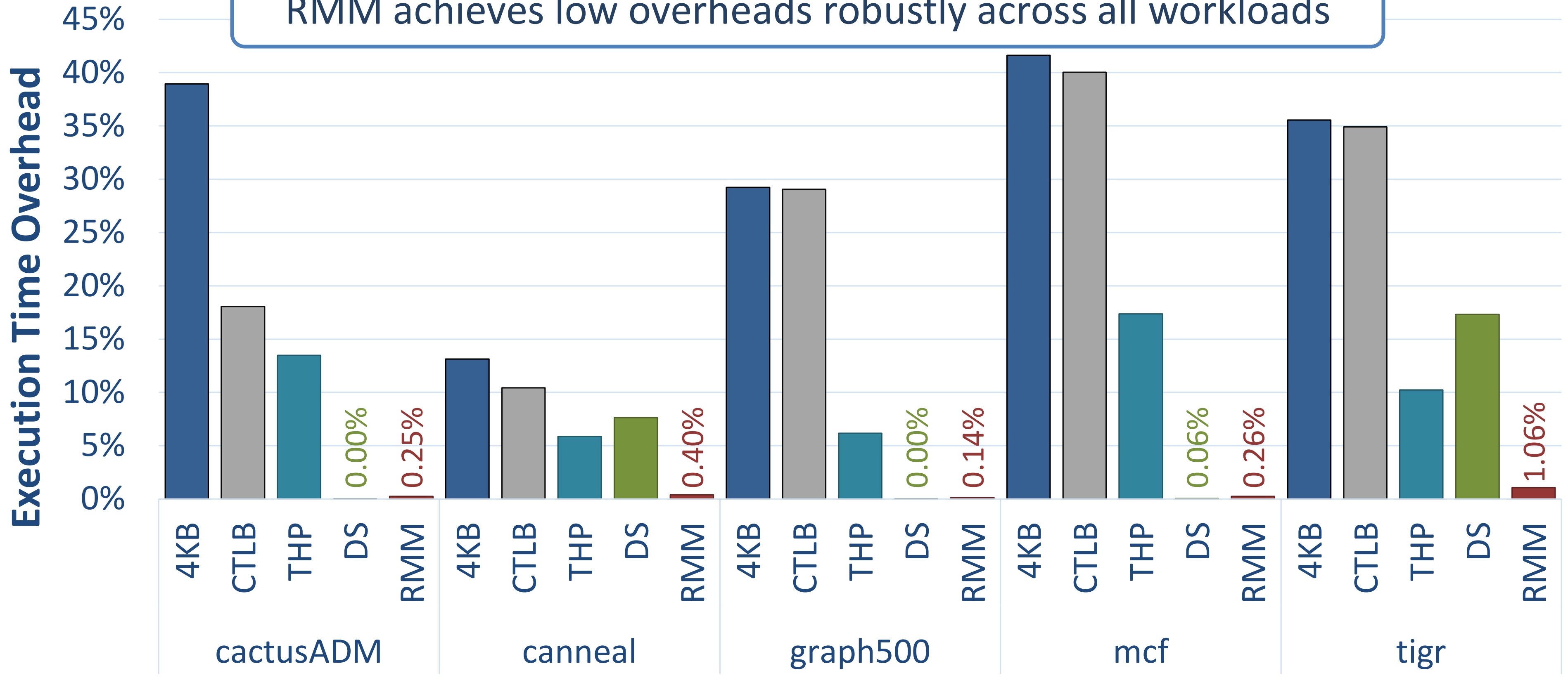
Performance Results

Direct Segment perfect for some but not all workloads



Performance Results

RMM achieves low overheads robustly across all workloads



Why low overheads? Virtual Contiguity

Benchmark	Paging	Ideal RMM ranges
	4KB + 2MB THP	
cactusADM	1365 + 333	
canneal	10016 + 359	
graph500	8983 + 35725	
mcf	1737 + 839	
tigr	28299 + 235	

Only few ranges for 99% coverage

Summary

- **Problem:** Virtual memory overheads are high
- Proposal: **Redundant Memory Mappings**
  - **Propose** compact representation called *range translation*
  - **Range Translation** – arbitrarily large contiguous mapping
  - **Effectively cache, manage and facilitate** range translations
  - **Retain flexibility** of 4KB paging
- Result:
  - **Reduces overheads** of virtual memory to **less than 1%**

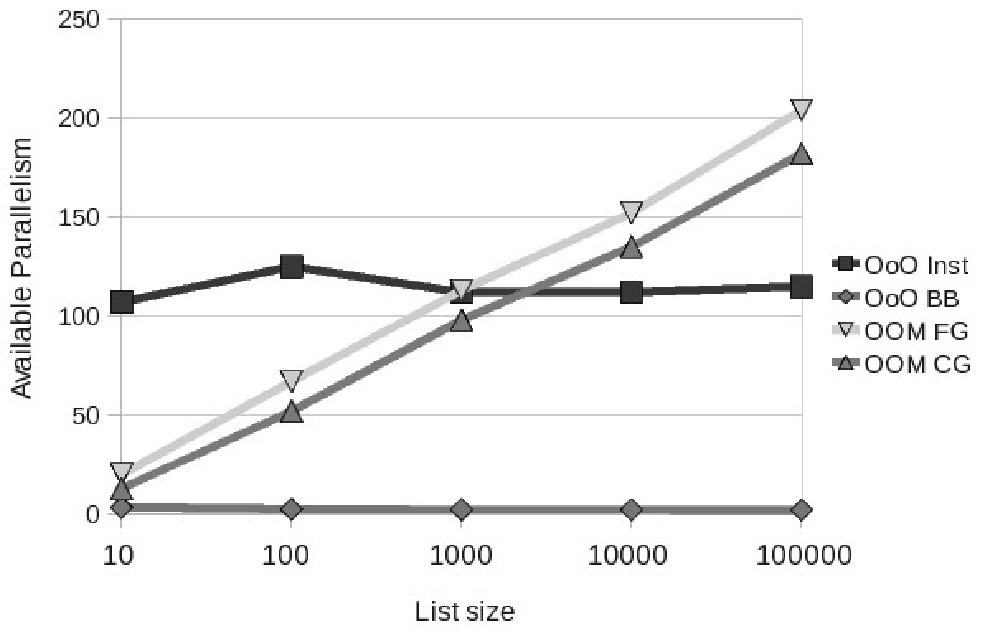
Thank you

osman.unsal@bsc.es

Results

- DDG (Dynamic Dependency Graph) methodology Austin et al. 1992
- DDG containing only true data dependencies to give upper bound on the available parallelism
- We compare the level of available parallelism extracted form sequential quicksort algorithm that our ideal model can achieve over ideal OoO
- For ideal OoO we use Pin Tool to generate traces
- For ideal OOM we use our functional level simulator

Results


250
200
150
100
50
0
Available Parallelism
10
100
1000
10000
100000
List size
OoO Inst
OoO BB
OOM FG
OOM CG