Tutorial Overview

- Overview and basics
- Using the tool suite
- How to use sim-outorder
- How to build your own simulators
- How to modify the ISA
- How to use the memory extensions
- SimpleScalar limitations and caveats
- Wrapup

Memory hierarchy extensions

- Current implementation:
  - Calls cache module, returns load latency right away
  - fast
  - accurate if little contention in memory system
- Memory extensions:
  - use callbacks at each level of hierarchy, event-driven
  - memory operation is woken up when miss complete
  - address translation, page tables
  - slower, but maintains causality

Extension files

`tlb.{c,h}`

- Code for address translation and physical memory

`cache_simple.c`

- Functional cache module

`mshr.h`

- Data structures and macros for MSHR handling

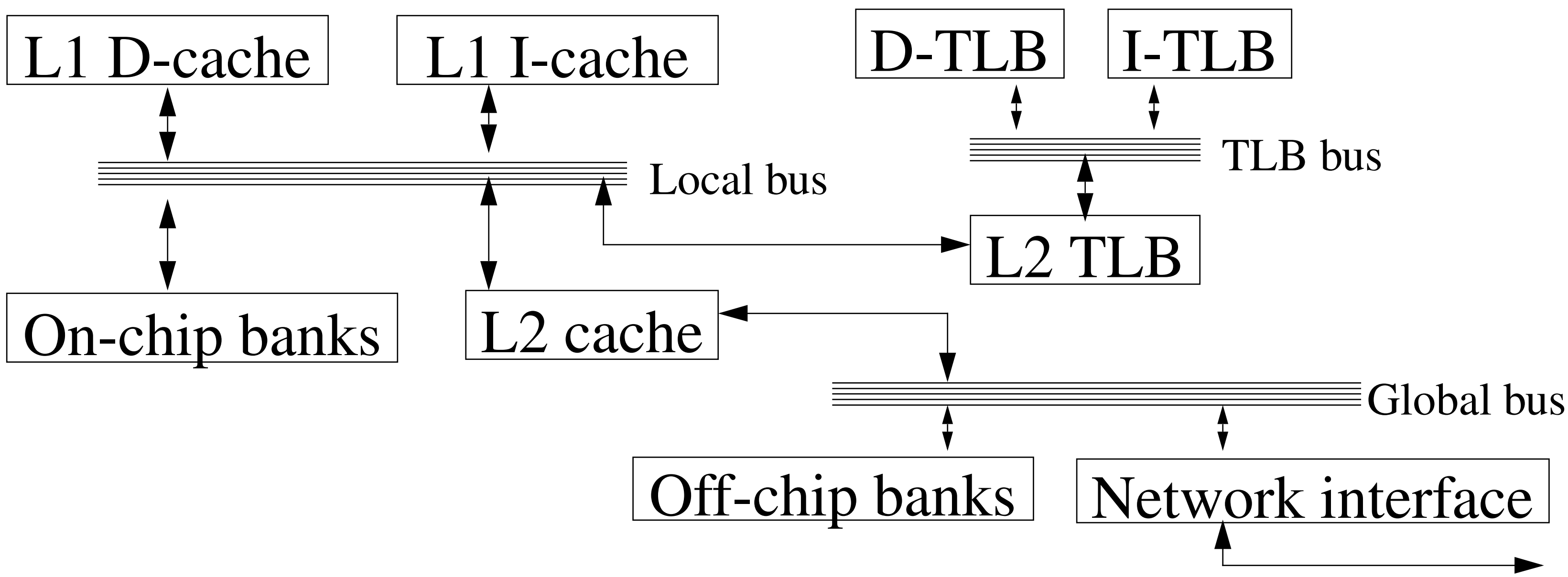
`interconnect.{c,h}`

- Code for buses

Structure of memory extensions

- Each cache or bus has the following fields:
  - `num_resources` - number of structures it connects to
  - `resource_code` - flag that determines selection of structures
  - `resources[]` - `(void *)` array of pointers to structures
- To define L1 instruction and/or data caches:
  - `-cache:icache <name>`
  - `-cache:dcache <name>`

Memory hierarchy example



Defining a memory hierarchy

Associativity Hit latency Translation Prefetch Resource names

-cache:define L1D:1024:32:2:l:1:vipt:0:2:1:Onbus:Globalbus

Name # sets Block size Replacement # resources Resource code

Width Arbitration # resources Resource names

-bus:define Onbus:32:8:1:1:0:2:1:L2:Onbank

Name Width Cycle ratio Inf. b/w Resource code

Name Banking code

-bank:define Onbank:20:0

Access penalty

Key functions

- `cache_access()` - performs lookup only
- `response_handler()` - deals with returning misses
- `get_bus()` - returns pointer to next bus (timing sim. only)
- `request_arrival()` - returns latency of bus access
- `get_next_memory_level()` - returns pointer to structure
- `blk_access_fn()` - calls appropriate memory routines

cache_access()

```
    check for hit;
    check for mshr_hit;
    allocate mshr (return full?);
    lat = request_arrival(bus == get_bus);
    ptr = get_next_memory_level(&type);
    schedule blk_access_fn(now+lat, ptr, type);
    return MISS;
mshr_hit:
    save target (return full?);
    return MISS;
cache_hit:
    return hit_latency;
```

Miss Status Holding Registers

- Initial implementation by Alain Kägi
- MSHR miss: allocate an MSHR, initialize one target
- MSHR hit: allocate one target
- When response returns, fire all targets
- If no available MSHRs or targets (L1 only)
  - Place load back in issue ready queue
  - Prevent store from committing
  - Continue stalling i-fetch

TLB setup

- ITLB and DTLB can be defined
  - use `-tlb:define` (args same as cache)

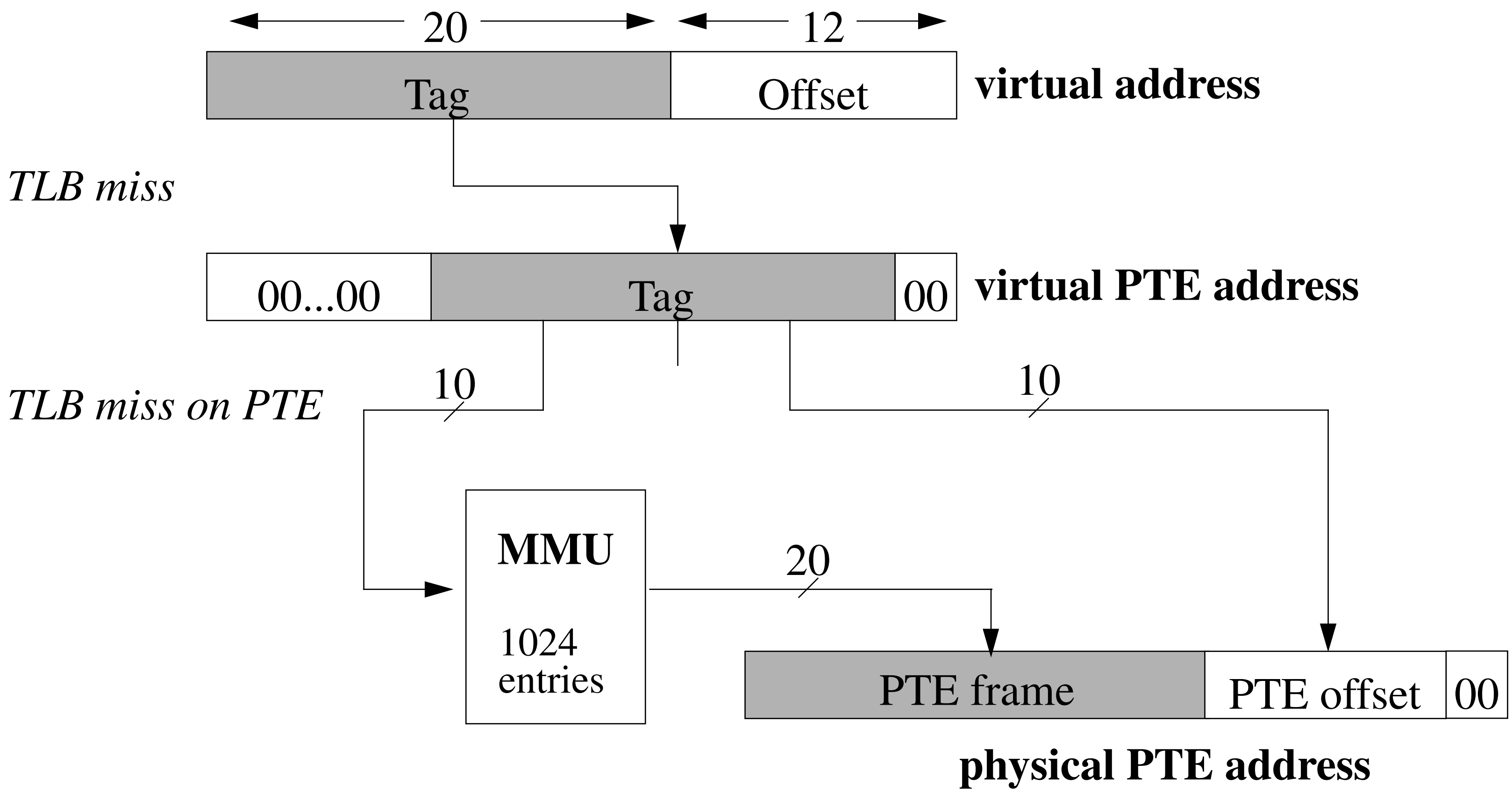
    `-tlb:dtlb <name>`

    `-tlb:itlb <name>`
- L1 icache uses itlb if defined
- All other caches use dtlb pointer if necessary
- Caches automatically access TLB if translation needed
  - Send request down and stall on TLB miss (or if no TLB)

Address translation

- 4-byte page table entries
  - 4KB pages assumed (possible to change, not parameterized)
  - 4 MB (1 M entries) thus maps 4GB (32-bit address space)
- Page table pages mapped in virtual address space
  - Mapped in low 4MB of virtual address space
  - Therefore need address translation on PTEs also
  - Translations for page table pages stored in 4KB MMU

Address translation (con't)



Limitations of the memory extensions

- PRERELEASE
  - Probably bugs
  - Code will be cleaner for release
- No back-pressure on non-L1 caches
  - Must have enough MSHRs at lower levels
- Code is slower
  - Slowdown = 2 - 4 x (proportional to miss rate)
  - Not needed if memory system not an issue