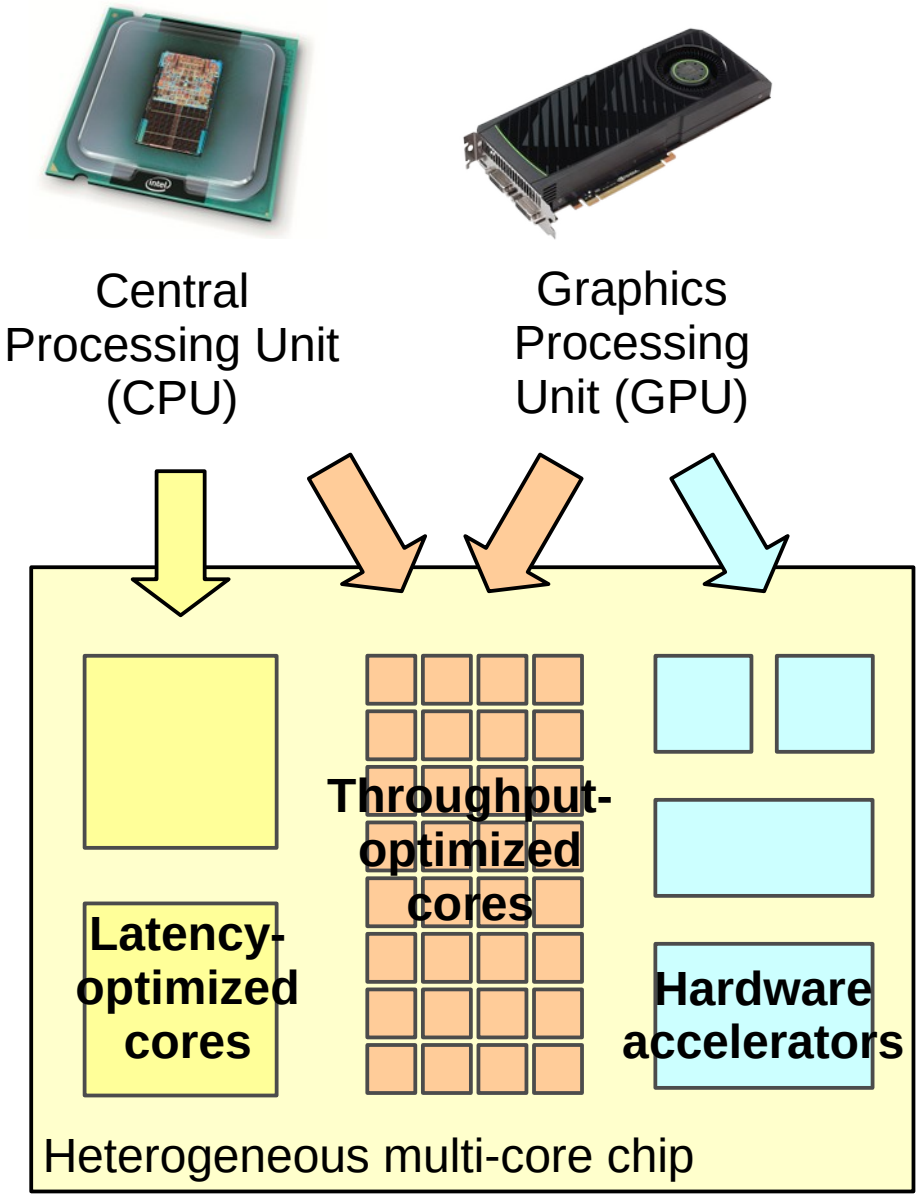
Simty:
a synthesizable general-purpose SIMT processor

Compas'2016

Caroline Collange
INRIA Rennes / IRISA
caroline.collange@inria.fr

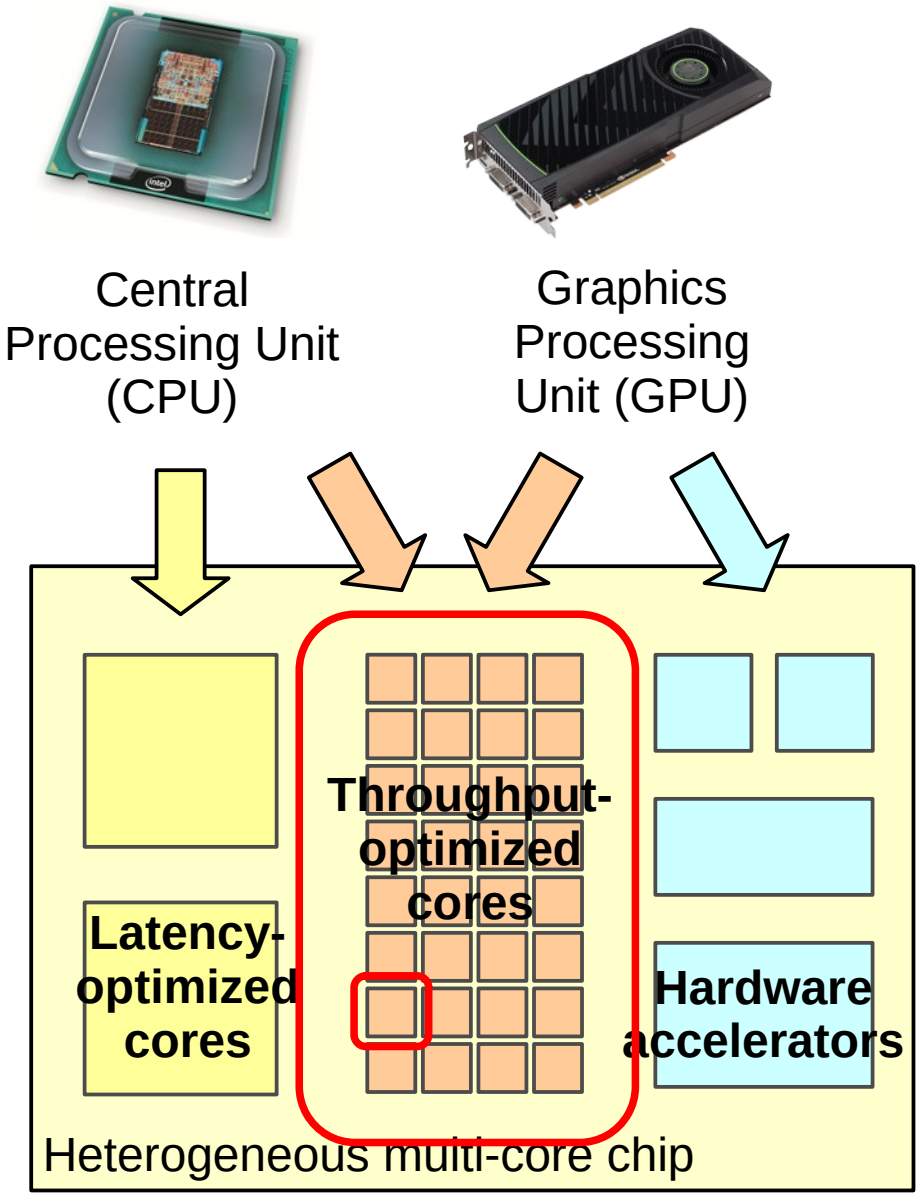
From GPU to heterogeneous multi-core

- Yesterday (2000-2010)
  - Homogeneous multi-core
  - Discrete components
- Today (2011-...)
  Heterogeneous multi-core
  - **Physically unified**
    CPU + GPU on the same chip
  - **Logically separated**
    Different programming models, compilers, instruction sets
- Tomorrow
  - Unified programming models?
  - **Single instruction set?**



From GPU to heterogeneous multi-core

- Yesterday (2000-2010)
  - Homogeneous multi-core
  - Discrete components
- Today (2011-...)
  Heterogeneous multi-core
  - **Physically unified**
    CPU + GPU on the same chip
  - **Logically separated**
    Different programming models, compilers, instruction sets
- Tomorrow
  - Unified programming models?
  - **Single instruction set?**
- **Defining the general-purpose throughput-oriented core**

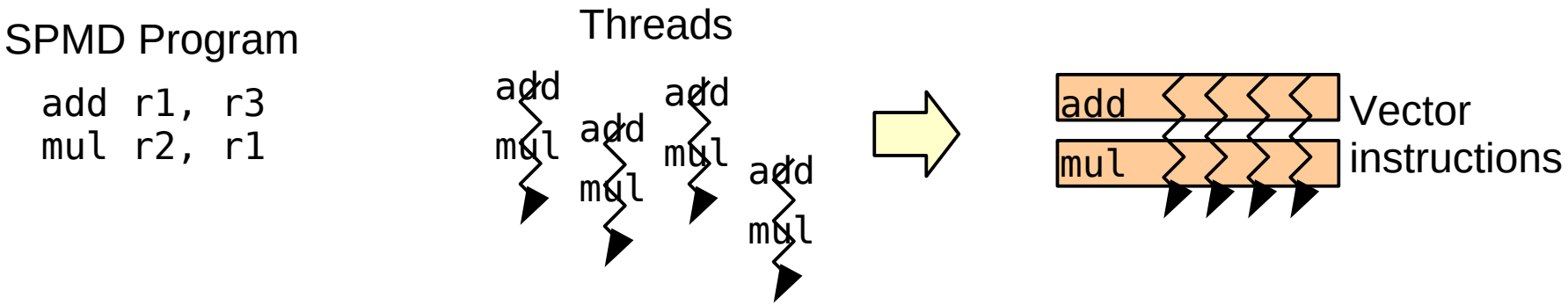


Outline

- Stateless dynamic vectorization
  - History
  - Principle
- The Simty core
  - Design goals
  - Micro-architecture

The enabler: dynamic vectorization

- Idea: hardware aggregates threads together to assemble vector instructions



  - Force threads to run in lockstep:
    threads execute the same instruction at the same time (or nothing at all)
  - Generalization of GPU's SIMT for general-purpose ISAs
- Benefits vs. static vectorization
  - Programmability: software sees only threads, not threads + vectors
  - Portability: vector width is not exposed in the ISA
  - Scalability: + threads → larger vectors or more latency hiding or more cores
  - Implementation simplicity: handling traps is straightforward

Goto considered harmful?

MIPS	NVIDIA Tesla (2007)	NVIDIA Fermi (2010)	Intel GMA Gen4 (2006)	Intel GMA SB (2011)	AMD R500 (2005)	AMD R600 (2007)	AMD Cayman (2011)
j	bar	bar	jmpi	jmpi	jump	push	push
jal	bra	bpt	if	if	loop	push_else	push_else
jr	brk	bra	iff	else	endloop	pop	pop
syscall	brkpt	brk	else	endif	rep	loop_start	push_wqm
	cal	brx	endif	case	endrep	loop_start_no_al	pop_wqm
	cont	cal	do	while	breakloop	loop_start_dx10	else_wqm
	kil	cont	while	break	breakrep	loop_end	jump_any
	pbk	exit	break	cont	continue	loop_continue	reactivate
	pret	jcal	cont	halt		loop_break	reactivate_wqm
	ret	jmx	halt	call		jump	loop_start
	ssy	kil	msave	return		else	loop_start_no_al
	trap	pbk	mrest	fork		call	loop_start_dx10
	.s	pret	push			call_fs	loop_end
		ret	pop			return	loop_continue
		ssy				return_fs	loop_break
		.s				alu	jump
						alu_push_before	else
						alu_pop_after	call
						alu_pop2_after	call_fs
						alu_continue	return
						alu_break	return_fs
						alu_else_after	alu
							alu_push_before
							alu_pop_after
							alu_pop2_after
							alu_continue
							alu_break
							alu_else_after

Control instructions in some CPU and GPU instruction sets

- GPUs: control flow divergence and convergence is explicit
  - Incompatible with general-purpose instruction sets

Flashback: prior episodes

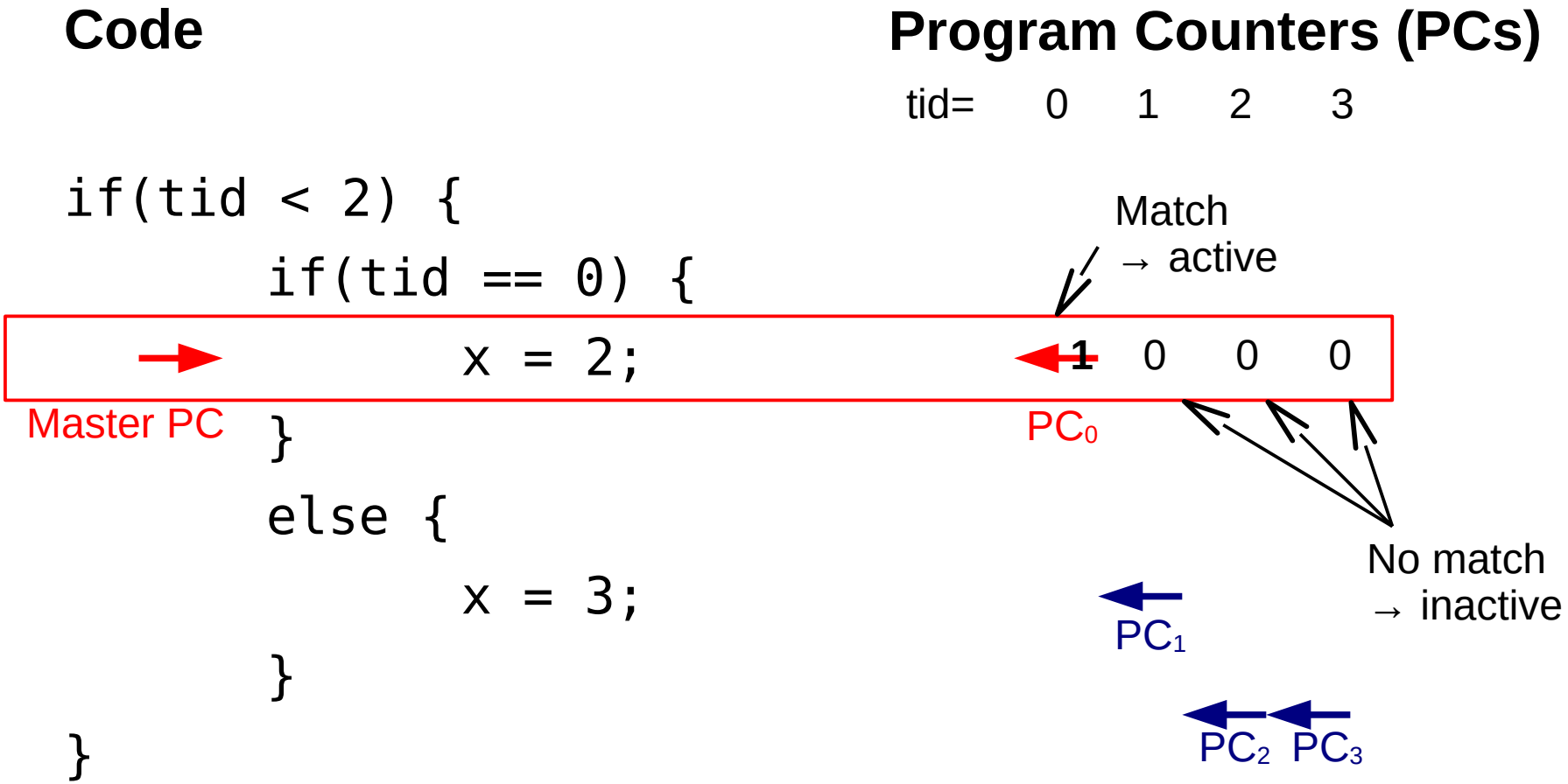
Goal: handle control flow divergence and reconvergence with branches only

First proposal: stateful dynamic vectorization [SympA 2009]

- Born out of necessity: reverse-engineer the Tesla ISA
  - (Eventually found a patent describing the Tesla implementation)
  - (Then found another patent that actually works)
- Benefits
  - Only branches, no more fancy control instructions ☺
- Limitations
  - Restricted to structured control flow
  - Stack-based like GPUs: shared state between threads

Stateless dynamic vectorization

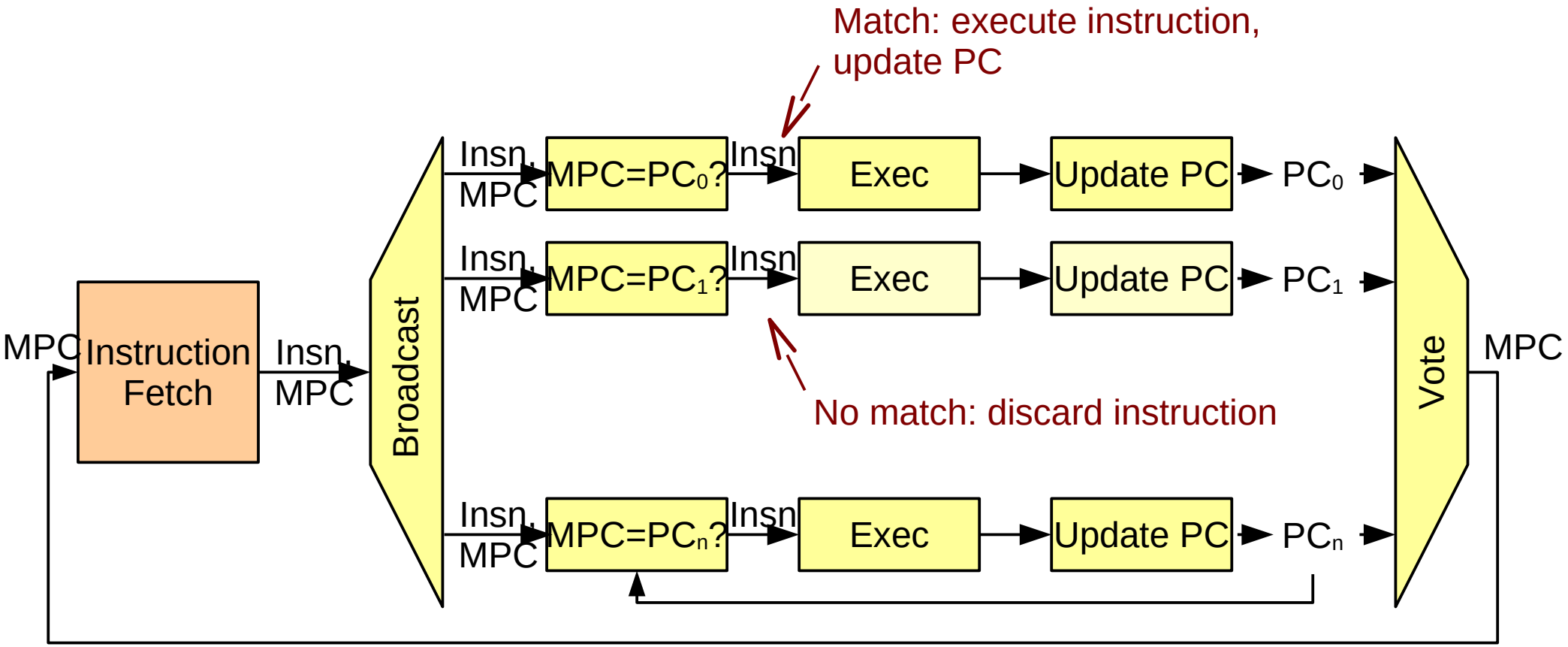
Forget about the stack, per-thread PCs are enough [Sympa 2011]



- Policy: MPC = min(PC_i) inside deepest function
  - Intuition: favor threads that are behind so they can catch up
  - Earliest reconvergence with code laid out in reverse post order

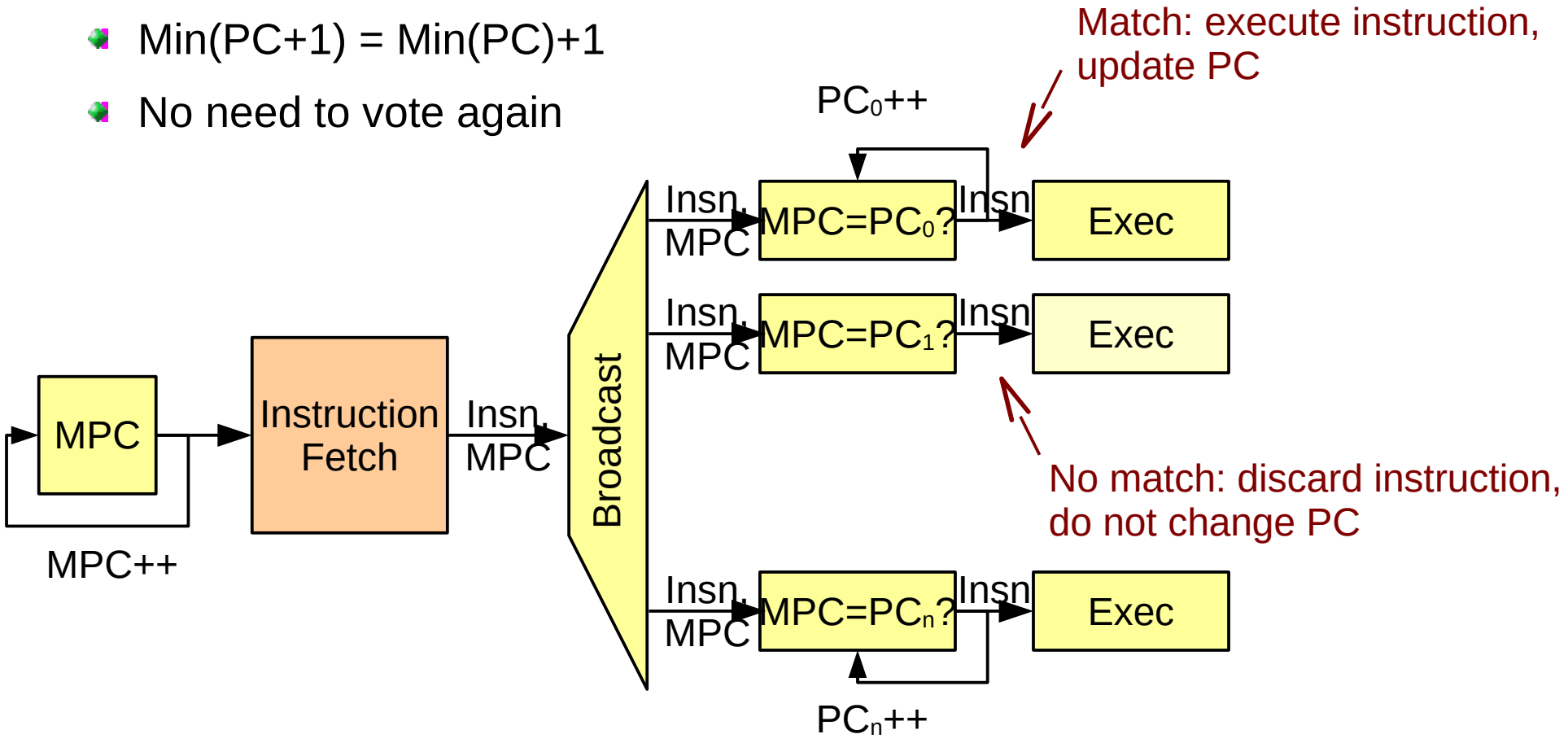
Functional view

- Control flow instruction or exception



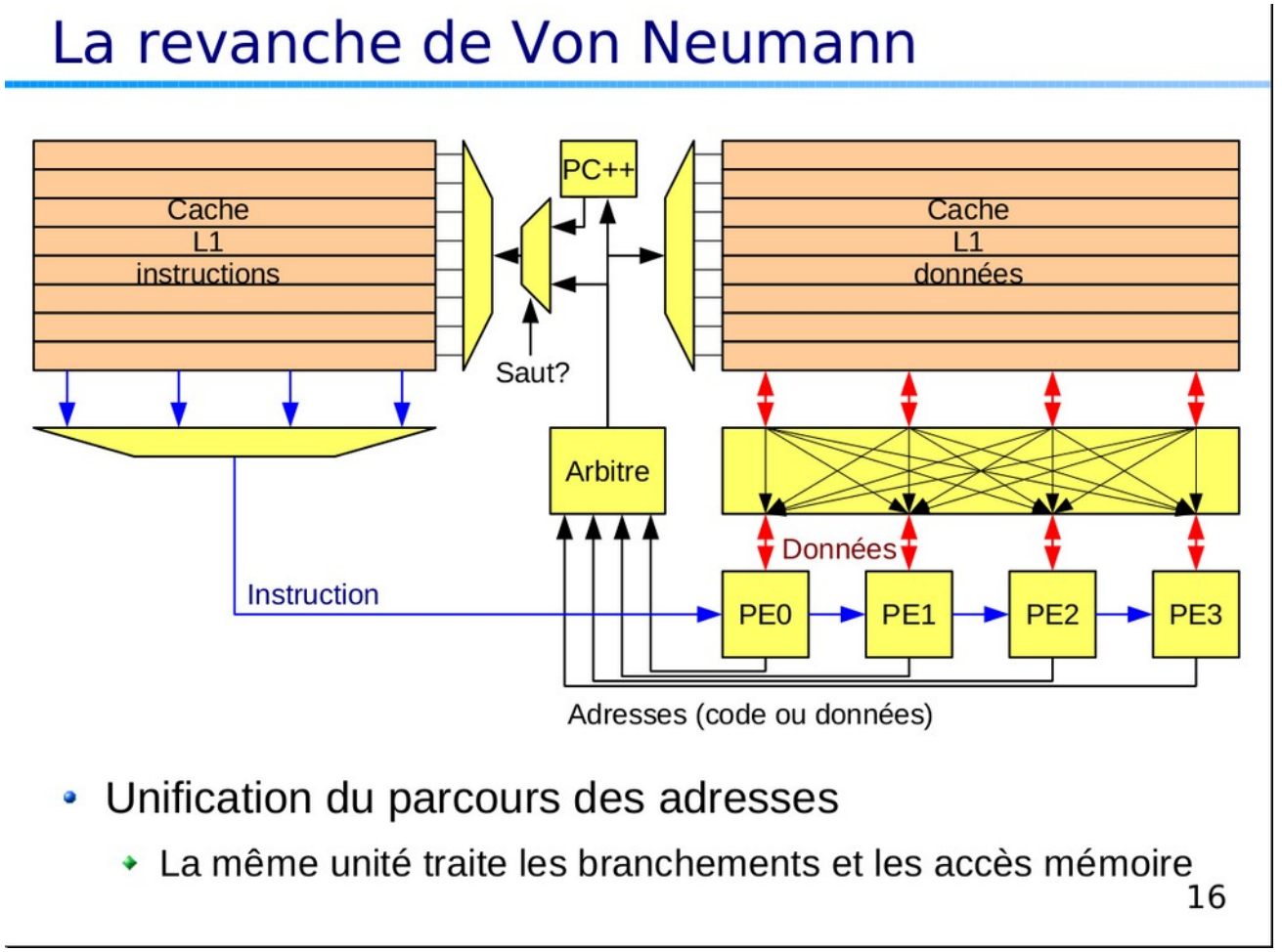
Functional view

- Arithmetic instruction
  - Min(PC+1) = Min(PC)+1
  - No need to vote again



Implementation 1: shared reduction tree

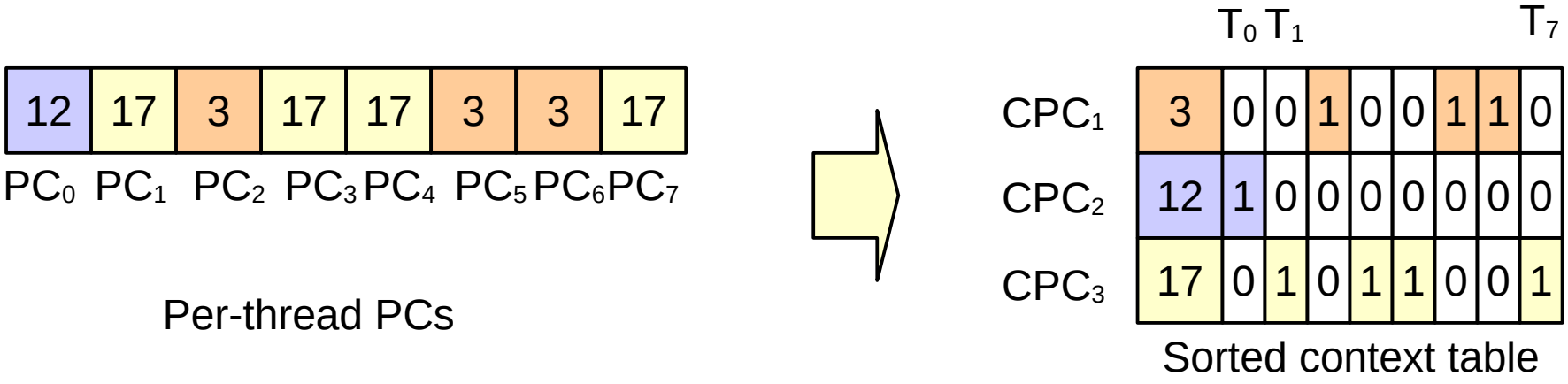
- Needs hardware reduction tree to compute the min(PCs)
- Analogy with memory access bank arbitration: implementation can reuse existing hardware



Implementation 2: sorted context table

[ISCA 2012]

- Common case: few different PCs
- Order stable in time
- Keep Common PCs+activity masks in sorted heap



- Branch = insertion in sorted context table
- Convergence = fusion of head entries when CPC_1=CPC_2

Outline

- Stateless dynamic vectorization
  - History
  - Principle
- The Simty core
  - Design goals
  - Micro-architecture

Why synthesizable RTL?

Revisiting prior papers

- SympA 2011: sensible idea, impractical implementation
  - RTL synthesis results thanks to Nicolas Brunie [TSI 2013]
    non-trivial overhead
  - Branch arbitration and memory access fall into
    different pipeline stages when optimizing the critical path
- ISCA 2012
  - Partial RTL synthesis results
  - Overestimated complexity of other components
  - Not much impact on the community (impression of complexity?)
- Need for RTL-level implementation for the whole pipeline

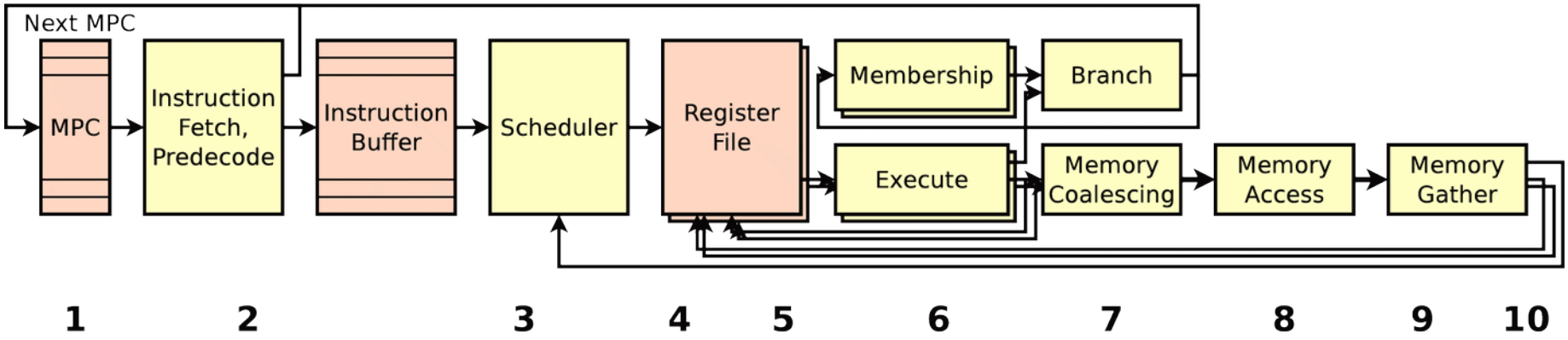
Simty: illustrating the simplicity of SIMT

Proof of concept for dynamic vectorization

- Focus on the core ideas → the RISC of dynamic vectorization
- Simple programming model
  - Many scalar threads
  - General-purpose RISC ISA
- Simple micro-architecture
  - Single-issue RISC pipeline
  - SIMD execution units
- Highly concurrent, scalable
  - Interleaved multi-threading to hide latency
  - **Dynamic vectorization** to increase execution throughput
  - Target: hundreds of threads per core

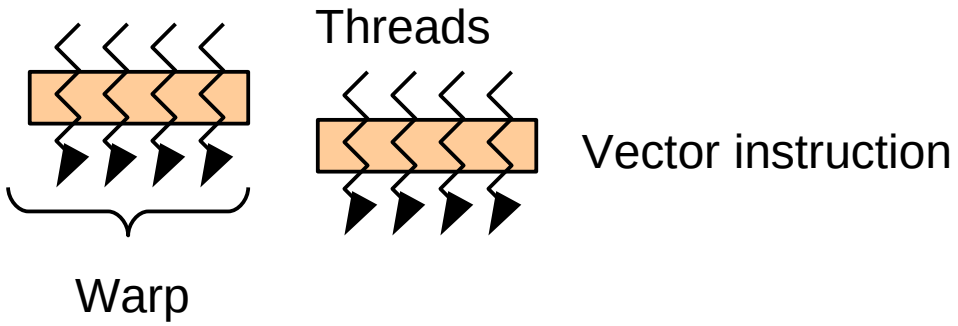
Simty implementation

- Written in synthesizable VHDL
- Runs the RISC-V instruction set (RV32I)
- Fully parametrizable warp size, warp count
- 10-stage pipeline



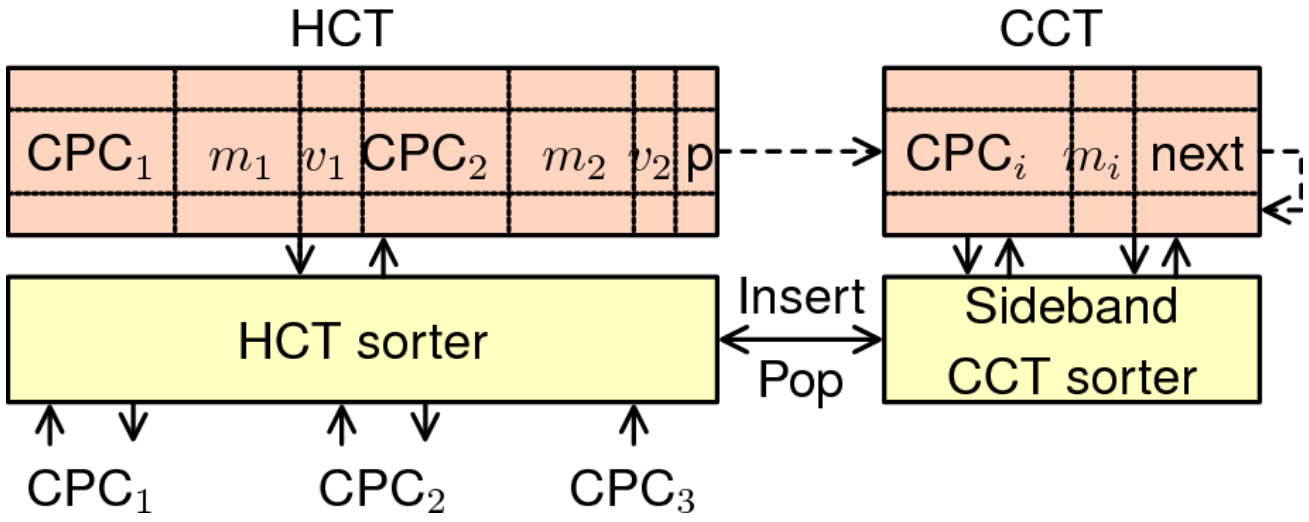
Multiple warps

- Wide dynamic vectorization is counterproductive
  - Threads that hit in the cache wait for threads that miss
  - Breaks latency hiding capability of interleaved multi-threading
  - Performs “as bad” as a vector processor
- Two-level approach : partition threads into warps, vectorize inside warps
  - Standard approach on GPUs



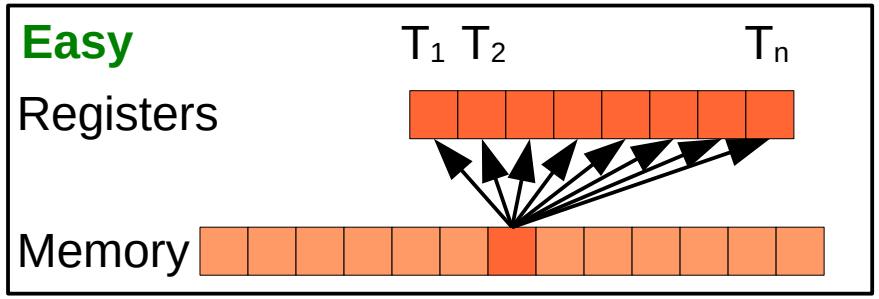
Two-level context table

- Cache top 2 entries in the *Hot Context Table* register
  - Constant-time access to CPC_i, activity masks
  - In-band convergence detection
- Other entries in the *Cold Context Table*
  - Branch → incremental insertion in CCT
  - Out-of-band CCT sorting: inexpensive insertion sort in $O(n^2)$
  - If CCT sorting cannot catch up: degenerates into a stack (=GPUs)



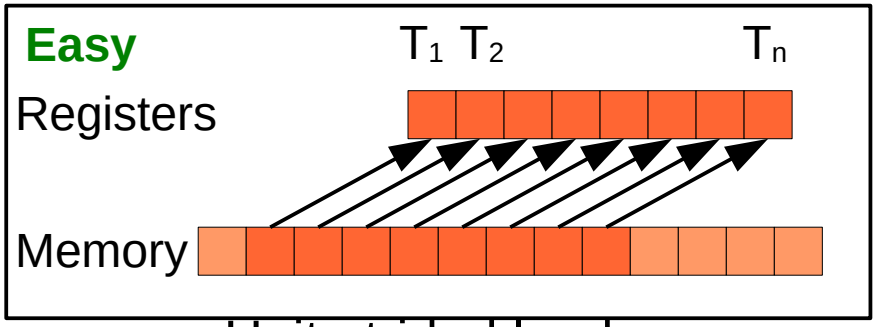
Memory access patterns

In traditional vector processing



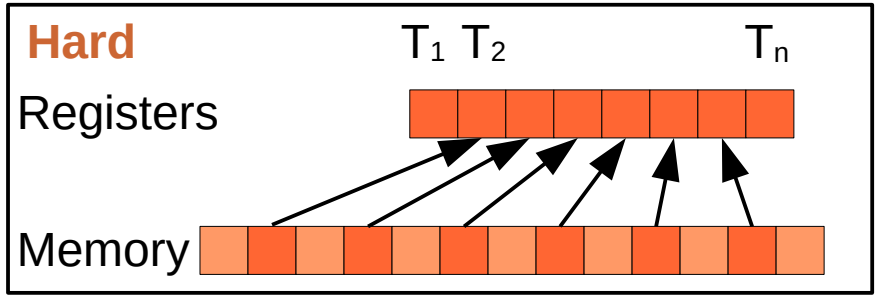
Scalar load & broadcast

Reduction & scalar store



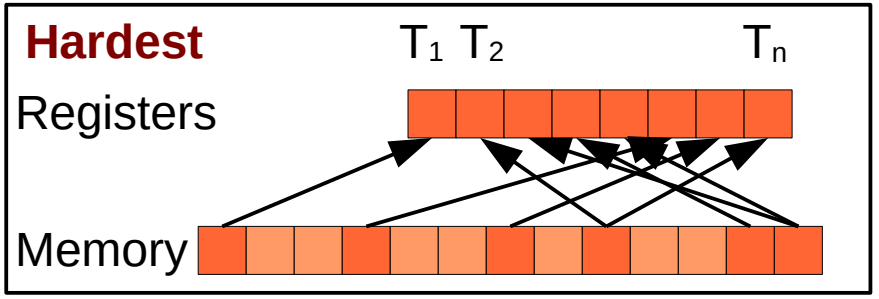
Unit-strided load

Unit-strided store



(Non-unit) strided load

(Non-unit) strided store

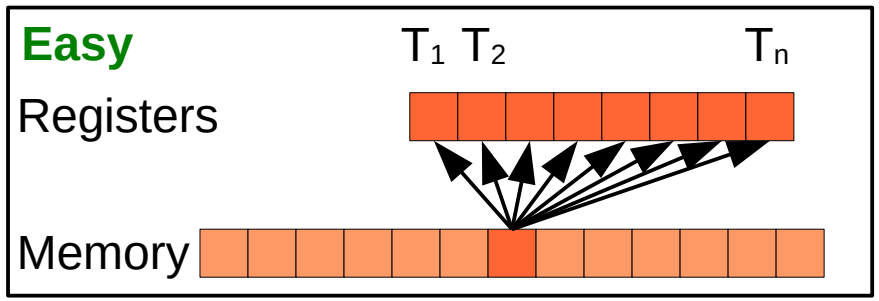


Gather

Scatter

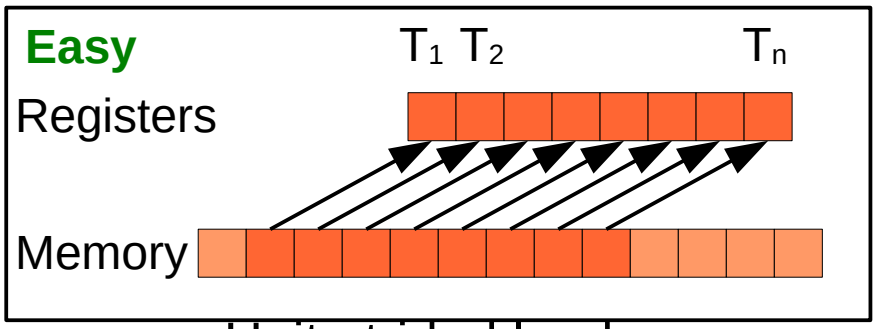
Memory access patterns

With dynamic vectorization



Scalar load & broadcast

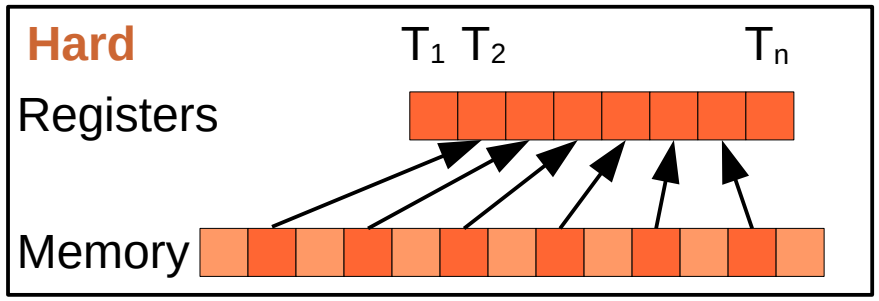
Reduction & scalar store



Unit-strided load

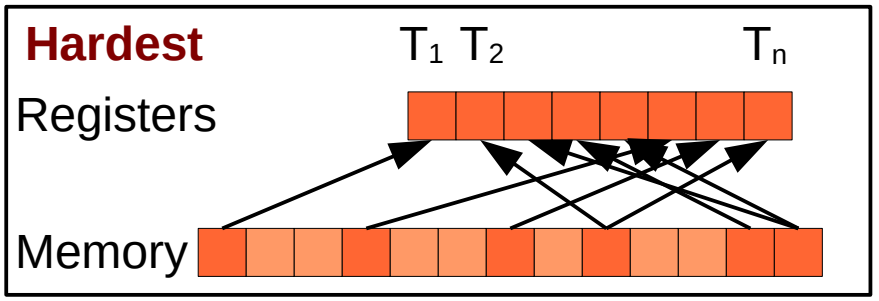
Unit-strided store

Common case



(Non-unit) strided load

(Non-unit) strided store



Gather

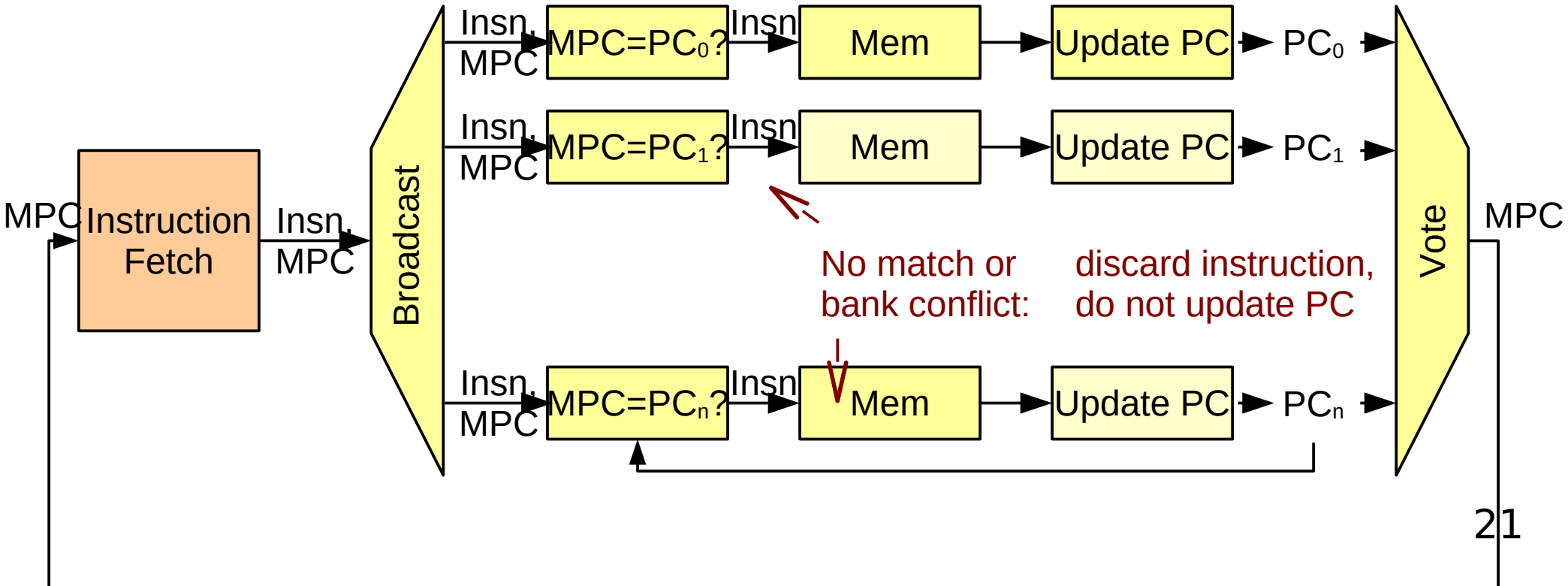
Scatter

General case

- Support the general case, optimize for the common case

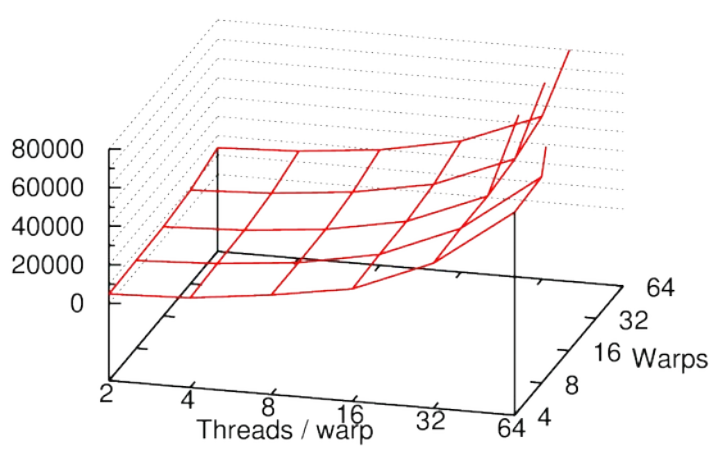
Memory access unit

- Scalar and aligned unit-strided scenarios: single pass
- Complex accesses in multiple passes using replay
- Execution of a scatter/gather is interruptible
  - Allowed by multi-thread ISA
  - No need to rollback on TLB miss or exception

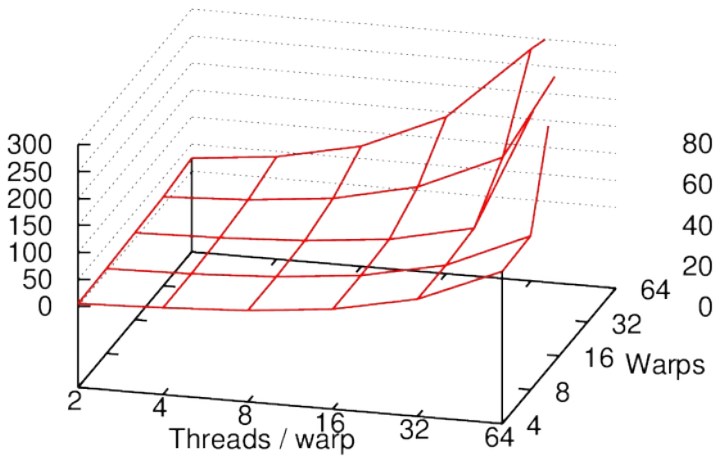


FPGA prototype

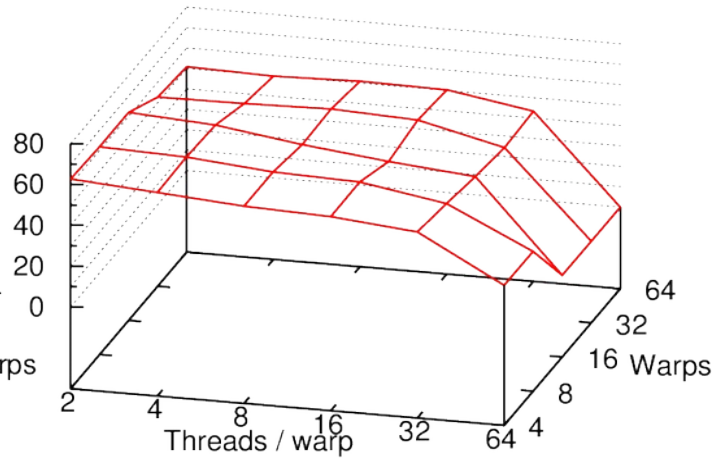
On Altera Cyclone IV



Logic area (LEs)



Memory area (M9Ks)



Frequency (MHz)

- Up to 2048 threads per core: 64 warps × 32 threads
- Sweet spot: 8x8 to 32x16

Latency hiding
multithreading depth

Throughput
SIMD width

Conclusion

- Stateless dynamic vectorization works
- Unexpectedly inexpensive
  - Overhead amortized even for single-issue RISC without FPU
- Scalable
  - Parallelism in same class as state-of-the-art GPUs
- Minimal software impact
  - Reuse the RISC-V software infrastructure: gcc and LLVM backends
  - OS changes to manage ~10K threads?
- One step on the road to single-ISA heterogeneous CPU+GPU

Simty:
a synthesizable general-purpose SIMT processor

Compas'2016

Caroline Collange
INRIA Rennes / IRISA
caroline.collange@inria.fr